

EMX User Manual

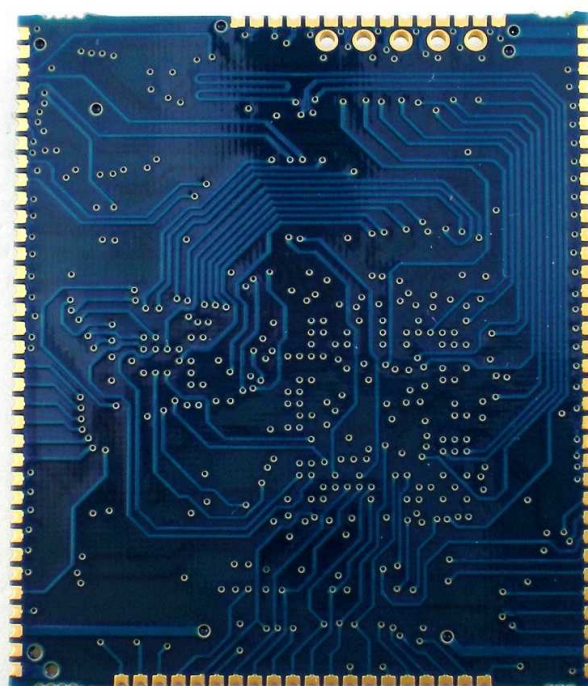
Rev.1.3

March 10, 2011

User Manual



EMX Module Top



EMX Module Bottom

Document Information

| Information | Description |
|-------------|--|
| Abstract | This document covers information about EMX Module, specifications, tutorials and references. |

| Revision History | |
|------------------|-----------------------------------|
| Date | Modification |
| 03/10/11 | Various updates |
| 09/14/10 | Updated In-field update section |
| 08/03/10 | Updated power functions |
| 07/21/10 | Updated information for NETMF 4.1 |
| 04/28/10 | Pin-out table updated |
| 04/26/10 | Updated network section |
| 04/02/10 | Updates / Fixes |
| 03/05/10 | First full version |
| 03/02/10 | Preliminary document |

Table of Contents

| | | | |
|---|----|---|----|
| 1. Introduction..... | 4 | I2C..... | 37 |
| 1.1. What is Microsoft .NET Micro Framework (NETMF)?..... | 4 | CAN..... | 37 |
| 1.2. NETMF - Porting vs. Using..... | 4 | One-wire..... | 37 |
| 1.3. GHI's .NET Micro Framework Based Solutions..... | 5 | 9.5. Networking (TCP/IP)..... | 37 |
| 1.4. What is EMX Module?..... | 5 | MAC address setting..... | 37 |
| 1.5. Extended Features with EMX Module..... | 6 | IP address (DHCP or static)..... | 38 |
| 1.6. EMX Module Key Features..... | 6 | Ethernet..... | 38 |
| 1.7. Example Applications..... | 6 | Wireless LAN WiFi (IEEE 802.11b)..... | 39 |
| 2. EMX Development System..... | 7 | PPP (TCP/IP access through serial modems)..... | 40 |
| 3. EMX Module Architecture H/W & S/W..... | 9 | SSL..... | 40 |
| 3.1. Block Diagram..... | 9 | 9.6. Graphics..... | 40 |
| 3.2. LPC2478 Microcontroller..... | 10 | 9.7. Touch Screen Control..... | 41 |
| 3.3. SDRAM..... | 10 | 9.8. USB Client (Device) | 41 |
| 3.4. FLASH..... | 10 | USB cable connection detection..... | 43 |
| 3.5. Ethernet PHY..... | 10 | 9.9. USB Host and Supported Class Drivers..... | 43 |
| 3.6. Runtime Loadable Procedure (RLP)..... | 10 | 9.10. Storage Devices (SD, USB MS) / File System..... | 43 |
| 4. Pin-Out Description..... | 11 | SD/MMC Memory..... | 43 |
| 4.1. EMX Module Pin-out Table..... | 12 | USB Mass Storage..... | 44 |
| 5. EMX on boot up..... | 15 | 9.11. Analog Inputs/Outputs..... | 44 |
| 5.1. GHI Boot Loader vs. TinyBooter vs. EMX Firmware..... | 17 | 9.12. PWM..... | 44 |
| 5.2. EMX Access Interface..... | 18 | 9.13. Output Compare..... | 44 |
| Emergency GHI Boot Loader Access..... | 18 | 9.14. Battery RAM..... | 45 |
| Other Interfaces..... | 18 | 9.15. Power Control / Hibernate..... | 45 |
| 6. GHI Boot Loader..... | 19 | Power Control..... | 45 |
| 6.1. GHI Boot Loader Commands..... | 19 | Hibernate..... | 45 |
| 6.2. TinyBooter Update through GHI Boot Loader..... | 19 | 9.16. Real Time Clock..... | 45 |
| 7. TinyBooter..... | 22 | 9.17. Processor Register Access..... | 45 |
| 7.1. EMX Firmware Update Through TinyBooter..... | 22 | 9.18. In-Field Update..... | 45 |
| 8. EMX firmware..... | 26 | 9.19. Managed Application Protection..... | 46 |
| 8.1. Getting Started with EMX..... | 26 | 9.20. Runtime Loadable Procedure RLP..... | 46 |
| All you need to start up..... | 26 | 9.21. Watchdog..... | 46 |
| First Power-up..... | 27 | 10. Advanced Users..... | 47 |
| Adding GHI NETMF Library..... | 30 | 11. EMX design Consideration..... | 47 |
| 8.2. EMX Emulator..... | 33 | 11.1. Hardware..... | 47 |
| 9. EMX Module Features..... | 34 | 11.2. Software..... | 47 |
| 9.1. Application Flash/RAM/EWR..... | 34 | 11.3. EMX Placement..... | 48 |
| Extended Week References (EWR)..... | 34 | Machine Placement..... | 49 |
| 9.2. Debugging Interface (Access Interface)..... | 34 | Appendix A: MFDeploy Tool..... | 50 |
| 9.3. Digital Inputs/Outputs..... | 36 | Legal Notice..... | 51 |
| 9.4. Serial Peripherals..... | 36 | Licensing..... | 51 |
| Serial Port (UART)..... | 36 | Disclaimer..... | 51 |
| SPI..... | 36 | | |

1. Introduction

1.1. What is Microsoft .NET Micro Framework (NETMF)?

Microsoft .NET Micro Framework is a lightweight implementation of .NET Framework. It focuses on the specific requirements of resource-constrained embedded systems. Supporting development in C# and debugging on an emulation or the device, both using Microsoft's Visual Studio. The .NET Micro Framework is also open source, released under the Apache 2.0 license and completely free.

Developers already experienced with .NET and Visual Studio can take advantage of their skills immediately reducing the learning curve. The actual C# application development process is completely shielded from the low-level design details of the hardware platform. Combining the benefits with off-the-shelf, low-cost, network-enabled embedded systems creates a rapid product development solution.

1.2. NETMF - Porting vs. Using

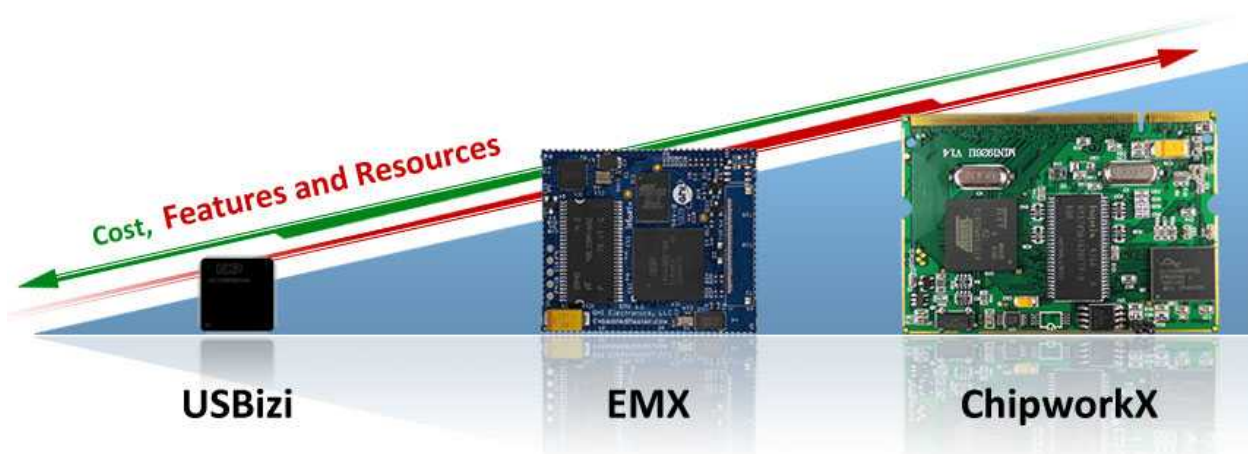
There are two sides to working with NETMF, porting it and using it. For example, writing a JAVA game on a cell phone is much easier than porting the JAVA virtual machine (JVM) to the phone. The phone manufacturer did all the hard work of porting JAVA to their phone allowing the game programmers to use it with ease. NETMF works the same way, porting is not easy but using it is effortless.

NETMF can be split into two major components, the core (CLR) and HAL (Hardware Access Layer). The core libraries are made so they are hardware independent. Usually, no modifications are needed on the core libraries. A developer porting NETMF for a hardware platform will need to make the HAL to handle interfacing the hardware control to upper layers.

According to GHI's experience with NETMF porting, it is not feasible to work on porting NETMF to your new hardware in case you are targeting medium or low quantities annually (less than 100,000 units). A faster-to-market option is by using one of the available OEM modules/chipsets. These OEM devices have everything you need built in the hardware and software.

1.3. GHI's .NET Micro Framework Based Solutions

With GHI Electronics, you're getting an experienced partner that offers a wide range of .NET Micro Framework hardware and software capabilities using the various drop-in modules/chipsets such as ChipworkX, Embedded Master, EMX and USBizi. In addition, our free unlimited support is available to assist you at any point. New features and fixes come seamlessly to your product at no cost to you.



On top of the great features that .NET Micro Framework provides, such as Ethernet, graphics and touch screen, GHI solutions has additional exclusive features such as USB host, PPP (GPRS/3G), database and native code runtime libraries (RLP). All these exclusive features are included at no extra cost to you.

1.4. What is EMX Module?

EMX Module is a combination of hardware (ARM Processor, Flash, RAM, Ethernet PHY...etc) on a very small (1.55"x1.8") SMT OEM 8-Layer board that hosts Microsoft .NET Micro Framework with various PAL/HAL drivers. In addition to the benefits of .NET Micro Framework, EMX includes exclusive software and hardware features, such as support for USB host, PPP networking and more.

EMX Module is a vastly sophisticated piece of hardware. This complexity provides the end-user with a remarkably simple platform to implement in any hardware design. Looking at the EMX Development System schematic shows just how simple it really is. All you need is 3.3 volts and some connections to bring the latest technologies to your products. With manageable features like USB host and WiFi, the possibilities are boundless.



1.5. Extended Features with EMX Module

EMX supports a complete set of .NET Micro Framework features such as TCP/IP, SSL, FAT, USB device and more. Including support for other exclusive GHI features such as full USB host stack, CAN, ADC, DAC,PPP, GPRS, 3G, etc. EMX also allows developers to load their own compiled native code. EMX includes protection against firmware or user application piracy.

1.6. EMX Module Key Features

- .NET Micro Framework
- 72Mhz 32Bit Processor
- 16MB RAM
- 4.5MB FLASH
- Embedded LCD controller
- Embedded Ethernet PHY with fast DMA communication.
- Runtime Loadable Procedure
- Full TCP/IP Stack
- Web Services
- SSL
- ZG2100 WiFi Driver
- PPP (GPRS/ 3G)
- DPWS
- Embedded USB host/device
- 76 GPIO Pins
- 39 Interrupt Inputs
- 2 SPI (8/16bit)
- I2C
- 4 UART
- 2 CAN Channels
- 7 10Bit Analog Inputs.
- 10Bit Analog Output
- 4Bit SD/MMC Memory card interface
- 6 PWM
- 160 mA current consumption with everything enabled
- 40mA Hibernate Mode
- -40°C to +85°C Operational
- RoHS Lead Free

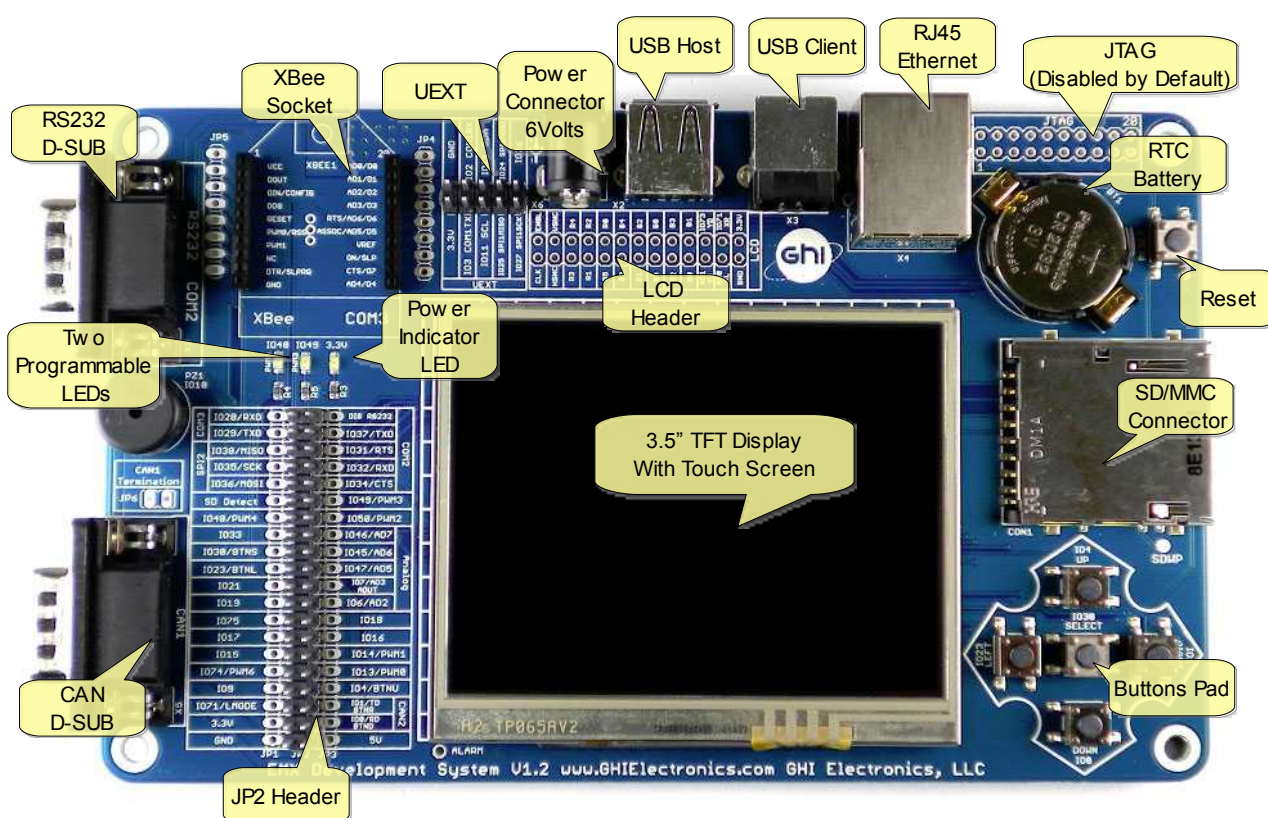
1.7. Example Applications

- Designs with intensive processing or time-critical routines (using RLP)
- Vending machine
- Measurement tool or tester
- Network server device
- Robotics
- GPS navigation
- Medical instrument (with a color touch screen display).
- Central alarm system
- Smart appliances
- Industrial automation devices
- Windows SideShow devices

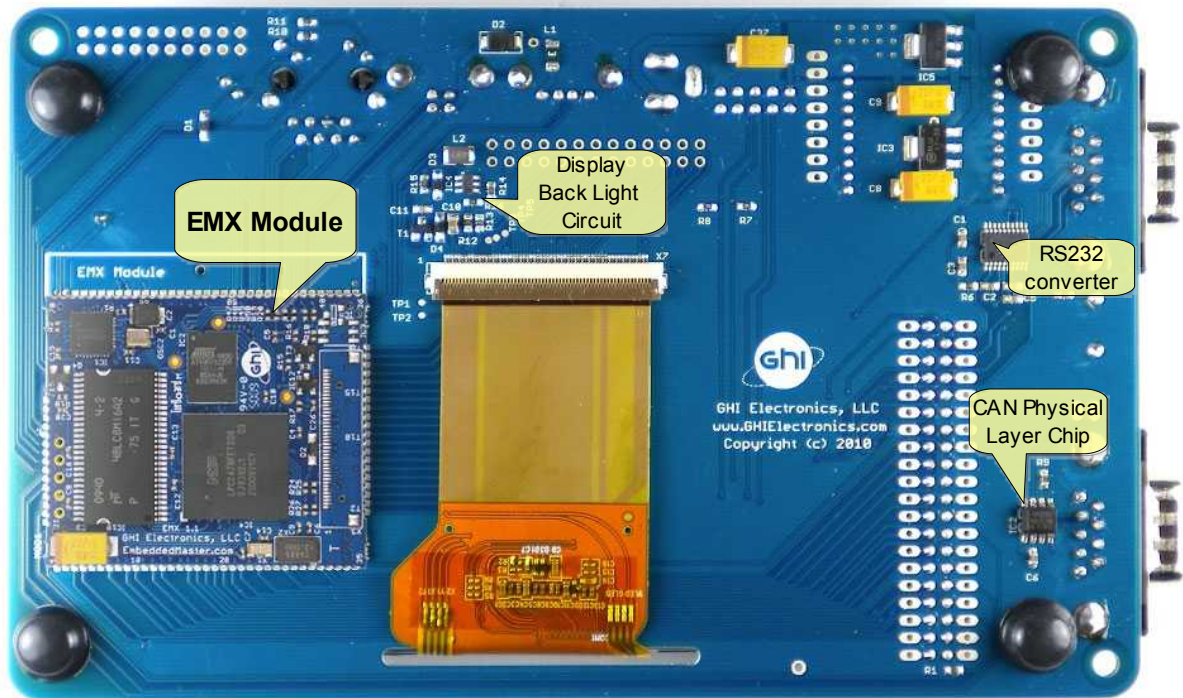
2. EMX Development System

EMX Development System is the official kit from GHI Electronics for the EMX module. This kit exposes the various peripherals and interfaces that make it an ideal starting point for any .NET Micro Framework project. Furthermore, most of EMX module signals such as GPIO, SPI and UART are accessible on a 0.1" header for rapid prototyping.

EMX Development System [Brochure](#) and [Pin-outs Document](#) provides for a more detailed view of this system.



Front View

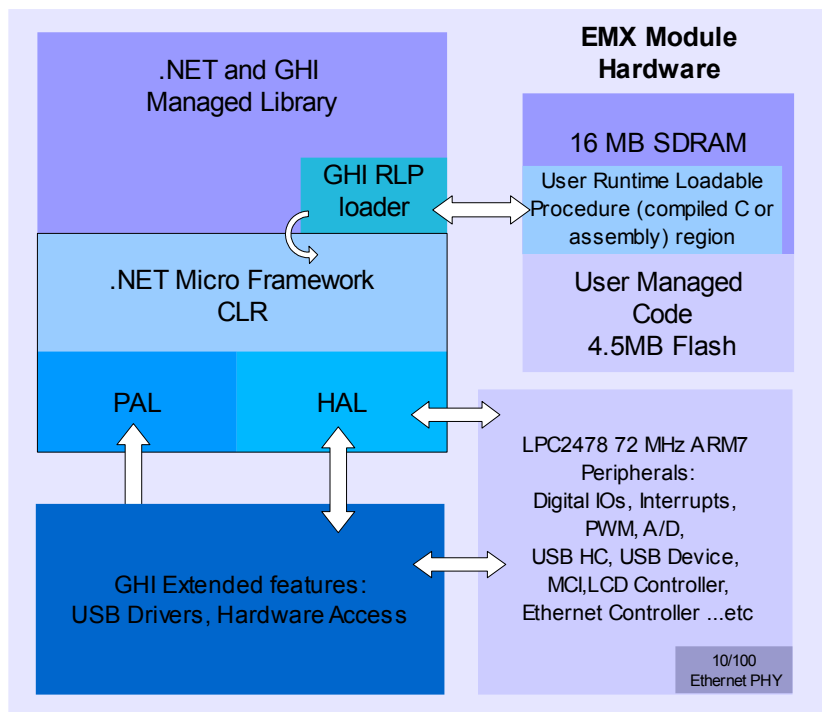
**Back View**

3. EMX Module Architecture H/W & S/W

EMX Module is a combination of hardware (ARM Processor, Flash, RAM, Ethernet PHY...etc) that hosts Microsoft .NET Micro Framework with various PAL/HAL drivers. In addition to the benefits of .NET Micro Framework, EMX includes exclusive software and hardware features, such as support for USB host, PPP networking and more.

The small 1.55"x1.8" module contains everything needed to run .NET Micro Framework. We also designed the module to be incredibly flexible while keeping costs low. Applications ranging from simple data loggers to high-end security systems and industrial controls can be created. The module is a sophisticated piece of hardware developed with a complex 8-layer BGA design. This complexity provides the end-user with a remarkably simple platform to implement in any hardware design. Looking at the EMX Development System schematic shows just how simple it really is. All you need is 3.3 volts and some connections to bring the latest technologies to your products.

3.1. Block Diagram



3.2. LPC2478 Microcontroller

The LPC2478 72Mhz ARM7 32Bit processor is the core of EMX Module. The LPC2478 contains a memory acceleration interface with 128Bit internal FLASH memory. This lets the processor core run with zero wait states. Comparing to executing code from 16Bit external FLASH we see over 10 times the execution speed. The internal FLASH is 0.5MB that is used to run the complete .NET Micro Framework core very efficiently. Also, the processor includes an RTC that can operate while the processor is off. EMX Module already has the needed circuitry to run the RTC. Users only need to add a battery or a super capacitor to VBAT pin.

Further more, LPC2478 has a wide range of peripherals that adds a lot of functions and features to EMX such as PWM, GPIO, LCD Controller, USB HC ...etc.

3.3. SDRAM

16MB of SDRAM comes standard with EMX Module.

3.4. FLASH

4MB of external flash is available on EMX Modules. This doesn't include the 0.5MB internal flash used for Micro Framework CLR execution. External flash is used for system assemblies, boot loader, user deployment and EWR storage. About 1MB of the external FLASH is used for boot loader, system assemblies and other internal GHI resources. About 3MB is reserved for deployed managed applications, including resources. 256KB is reserved for two EWR (Extended Week References) regions, each region is 128KB and one of them is reserved for CLR use.

3.5. Ethernet PHY

EMX Module hardware includes an industrial Ethernet PHY along with the needed circuitry. The Ethernet oscillator is controlled by the processor allowing the user to control it's power consumption. The designer only needs to wire the signals to the Ethernet connector. The recommended Ethernet connector is J0011D01BNL.

3.6. Runtime Loadable Procedure (RLP)

A highly useful and unique feature in EMX is allowing users to load their own compiled native code (C or assembly) and run it directly through managed code. This feature is similar to the use of DLLs on PCs. RLP can be used to implement processing intensive and time-critical routines.

4. Pin-Out Description

The LPC2478 72Mhz ARM7 32Bit processor is the core of EMX. The processor has a wide range of peripherals that add a lot of functions and features to EMX such as PWM, GPIO, USB HC, LCD Controller, etc.

Most signals on EMX are multiplexed to offer more than one function for every pin. It is up to the developer to select which one of the functions to use. GHI drivers and .NET Micro Framework does checking to make sure the user is not trying to use two functions on the same pin. The developer should still understand what functions are multiplexed so there is no conflict. For example, analog channel 3 (ADC3) and the analog output (AOUT) are on the same pin IO22. Either function can be used but not both of them simultaneously. Visit [Advanced Users section](#).

- The schematics of EMX Development System board should be used as a reference design.
- Advanced details on oscillator and power tolerance can be found in the LPC2478 datasheet from NXP website.
- Digital I/O pins are named **IOxx**, where xx is an assigned number.

4.1. EMX Module Pin-out Table

EMX Module is based on LPC2478 from NXP.

| No. | Name | | | EMX Module Pin Description |
|-----|-------------------------|--------|-------------------------|--|
| | LPC2478 H/W Name | EMX IO | 2 nd Feature | |
| 1 | | 3.3V | | Connect to 3.3 volt source. |
| 2 | | GND | | Connect to Ground. |
| 3 | P0.4 | IO0* | CAN2/ Down Button | RD CAN Channel 2 Data Receive pin (In) and TinyBooter/Firmware Down Button (Check hardware design consideration). |
| 4 | P0.5 | IO1* | CAN2 | TD CAN Channel 2 Data Transmit pin (Out). |
| 5 | P0.3 | IO2 * | COM1 | Serial port (UART) RXD receive signal (In) for COM1. |
| 6 | P0.2 | IO3* | COM1 | Serial port (UART) TXD transmit signal (Out) for COM1. |
| 7 | P2.30 | IO4* | UP Button | General purpose digital I/O and TinyBooter/Firmware Up Button (Check hardware design consideration). |
| 8 | P0.24 | IO5* | ADC1/ Touch_Y_UP | ADC1 (10Bit Analog to Digital Input) or Touch Screen Y-axis Up analog signal. |
| 9 | P0.25 | IO6* | ADC2/ COM4 | ADC2 (10Bit Analog to Digital Input) or Serial port (UART) TXD transmit signal (Out) for COM4. |
| 10 | P0.26 | IO7* | ADC3/ DAC/ COM4 | ADC3 (10Bit Analog to Digital Input) or DAC (Digital to Analog Output) or Serial port (UART) RXD receive signal (In) for COM4. |
| 11 | P0.23 | IO8* | ADC0/ Touch_X_Left | ADC0 (10Bit Analog to Digital Input) or Touch Screen X-axis Left analog signal. |
| 12 | P4.29 | IO9 | N/A | General purpose digital I/O |
| 13 | P4.28 | IO10 | Piezo | Piezo hardware control. |
| 14 | P0.28 | IO11* | I2C | (open drain pin) I2C Interface SCL |
| 15 | P0.27 | IO12* | I2C | (open drain pin) I2C Interface SDA |
| 16 | P3.16 | IO13 | PWM0 | PWM0 (Pulse Width Modulation Output) LPC2478 PWM Timer 0. |
| 17 | P3.24 | IO14 | PWM1 | PWM1 (Pulse Width Modulation Output) LPC2478 PWM Timer 1. |
| 18 | P3.25 | IO15 | N/A | General purpose digital I/O |
| 19 | P1.19 | IO16 | N/A | General purpose digital I/O |
| 20 | P2.21 | IO17* | N/A | General purpose digital I/O |
| 21 | P0.11 | IO18* | N/A | General purpose digital I/O |
| 22 | P2.22 | IO19* | N/A | General purpose digital I/O |
| 23 | P0.1 | IO20* | CAN1 | TD CAN Channel 1 Data Transmit pin (Out) |
| 24 | P0.10 | IO21* | N/A | General purpose digital I/O. |
| 25 | P0.0 | IO22* | CAN1 | RD CAN Channel 1 Data Receive pin (In) |
| 26 | P1.30 | N/A | USB_VBUS ¹ | USB device power detect signal. Connect to power pin on USB device. |
| 27 | P2.10 | IO23* | N/A | General purpose digital I/O |
| 28 | RTC_VBAT | | | Connect to 3.3 volt backup battery to keep the real-time clock running. |
| 29 | USB D- USB Host Feature | | | USB negative data line of the USB hosting feature. |
| 30 | USB D+ USB Host Feature | | | USB positive data line of the USB hosting feature. |
| 31 | P0.12 | IO45* | ADC6 | ADC6 (10Bit Analog to Digital Input). |
| 32 | P0.13 | IO46* | ADC7 | ADC7 (10Bit Analog to Digital Input). |
| 33 | P1.31 | IO47 | ADC5 | ADC5 (10Bit Analog to Digital Input). |
| 34 | | 3.3V | | Connect to 3.3 volt source. |
| 35 | P3.27 | IO48 | PWM4 | PWM4 (Pulse Width Modulation Output) LPC2478 PWM Timer 1. |
| 36 | | GND | | Connect to Ground. |
| 37 | | 3.3V | | Connect to 3.3 volt source. |
| 38 | | N/C | | Not Connected. |

| Name | | | | | |
|------|---------------------|--------|-------------------------|---|--|
| No. | LPC2478 H/W Name | EMX IO | 2 nd Feature | EMX Module Pin Description | |
| 39 | P3.26 | IO49 | PWM3 | PWM3 (Pulse Width Modulation Output) LPC2478 PWM Timer 1. | |
| 40 | P3.17 | IO50 | PWM2 | PWM2 (Pulse Width Modulation Output) LPC2478 PWM Timer 0. | |
| 41 | USBD- device | | | USB negative data line of the USB debugging interface and for the USB client feature. | |
| 42 | USBD+ device | | | USB positive data line of the USB debugging interface and for the USB client feature. | |
| 43 | Ethernet RD- | | | Ethernet receive data minus. | Recommended Ethernet connector is J0011D01BNL. Ethernet PHY is not needed since it is embedded in EMX hardware. |
| 44 | Ethernet RD+ | | | Ethernet receive data plus. | |
| 45 | Ethernet TD- | | | Ethernet transmit data minus. | |
| 46 | Ethernet TD+ | | | Ethernet transmit data plus. | |
| 47 | P0.18 | IO24* | SPI1 | SPI master bus interface MOSI signal (Master Out Slave In) for SPI1. | |
| 48 | P0.17 | IO25* | SPI1 | SPI master bus interface MISO signal (Master In Slave Out) for SPI1. | |
| 49 | P0.16 | IO26* | N/A | General purpose digital I/O. | |
| 50 | P0.15 | IO27* | SPI1 | SPI master bus interface SCK signal (Clock)for SPI1. | |
| 51 | P4.23 | IO28 | COM3 | Serial port (UART) RXD receive signal (In) for COM3. | |
| 52 | P4.22 | IO29 | COM3 | Serial port (UART) TXD transmit signal (Out) for COM3. | |
| 53 | P2.11 | IO30* | Select Button | General purpose digital I/O and TinyBooter/Firmware Select Button (Check hardware design consideration). | |
| 54 | P3.30 | IO31 | COM2 | Serial port (UART) RTS hardware handshaking signal for COM2. | |
| 55 | P2.1 | IO32* | COM2 | Serial port (UART) RXD receive signal (IN) for COM2. | |
| 56 | P0.6 | IO33* | N/A | General purpose digital I/O. | |
| 57 | P3.18 | IO34 | COM2 | Serial port (UART) CTS hardware handshaking signal for COM2. | |
| 58 | P0.7 | IO35* | SPI2 | SPI master bus interface SCK signal (Clock)for SPI2. | |
| 59 | P0.9 | IO36* | SPI2 | SPI master bus interface MOSI signal (Master Out Slave In) for SPI2. | |
| 60 | P2.0 | IO37* | COM2 | Serial port (UART) TXD transmit signal (Out) for COM2. | |
| 61 | P0.8 | IO38* | SPI2 | SPI master bus interface MISO signal (Master In Slave Out) for SPI2. | |
| 62 | P1.12 | IO39 | SD_DAT3 | SD card 4Bit data bus, data line no. 3. | |
| 63 | P1.11 | IO40 | SD_DAT2 | SD card 4Bit data bus, data line no. 2. | |
| 64 | P1.7 | IO41 | SD_DAT1 | SD card 4Bit data bus, data line no. 1. | |
| 65 | P1.2 | IO42 | SD_CLK | SD card 4Bit data bus, clock line. | |
| 66 | P1.6 | IO43 | SD_DAT0 | SD card 4Bit data bus, data line no. 0. | |
| 67 | P1.3 | IO44 | SD_CMD | SD card 4Bit data bus, command line. | |
| 68 | SD_PWR | | | SD memory power (connect directly to SD socket power pin). | |
| 69 | GND | | | Connect to Ground. | |
| 70 | RESET# | | | Hardware reset signal, Reset state is on Low. | |
| T1 | P2.12 | IO69* | LCD R0 | TFT Display, Red signal bit 0. | |
| T2 | P2.6 | IO65* | LCD R1 | TFT Display, Red signal bit 1. | |
| T3 | P2.7 | IO66* | LCD R2 | TFT Display, Red signal bit 2. | |
| T4 | P2.8 | IO67* | LCD R3 | TFT Display, Red signal bit 3. | |
| T5 | P2.9 | IO68* | LCD R4 | TFT Display, Red signal bit 4. | |
| T6 | P1.20 | IO51 | LCD G0 | TFT Display, Green signal bit 0. | |
| T7 | P1.21 | IO52 | LCD G1 | TFT Display, Green signal bit 1. | |
| T8 | P1.22 | IO53 | LCD G2 | TFT Display, Green signal bit 2. | |
| T9 | P1.23 | IO54 | LCD G3 | TFT Display, Green signal bit 3. | |
| T10 | P1.24 | IO55 | LCD G4 | TFT Display, Green signal bit 4. | |
| T11 | P1.25 | IO56 | LCD G5 | TFT Display, Green signal bit 5. | |
| T12 | P2.13 | IO70* | LCD B0 | TFT Display, Blue signal bit 0. | |
| T13 | P1.26 | IO57 | LCD B1 | TFT Display, Blue signal bit 1. | |
| T14 | P1.27 | IO58 | LCD B2 | TFT Display, Blue signal bit 2. | |

| Name | | | | |
|------|---------------------|--------|-------------------------|---|
| No. | LPC2478 H/W Name | EMX IO | 2 nd Feature | EMX Module Pin Description |
| T15 | P1.28 | IO59 | LCD B3 | TFT Display, Blue signal bit 3. |
| T16 | P1.29 | IO60 | LCD B4 | TFT Display, Blue signal bit 4. |
| T17 | P2.2 | IO61* | LCD CLK | TFT Display, Clock. |
| T18 | P2.4 | IO63* | LCD EN | TFT Display, Enable. |
| T19 | P2.5 | IO64* | LCD H-Sync | TFT Display, Horizontal sync. |
| T20 | P2.3 | IO62* | LCD V-Sync | TFT Display, Vertical sync. |
| J1 | | | ALARM | The alarm pin is an RTC controlled output. This is a 1.8 V pin. |
| J2 | P3.23 | IO71 | LMODE | General purpose digital I/O is used to choose the access interface for EMX between USB (Low) or COM1(High or not connected) on startup (refer to EMX access interface section). |
| J3 | P2.23 | IO72* | T_X_Right | Touch Screen X-axis Right digital output signal. |
| J4 | P3.31 | IO73 | T_Y_Down | Touch Screen Y-axis Down digital output signal. |
| J5 | P3.29 | IO74 | PWM5 | PWM5 (Pulse Width Modulation Output) LPC2478 PWM Timer 1 . |
| J6 | P4.31 | IO75 | N/A | General purpose digital I/O |
| J7 | JTAG TMS | | | JTAG TMS signal. |
| J8 | JTAG TCK | | | JTAG TCK signal. |
| J9 | JTAG TDO | | | JTAG TDO signal. |
| J10 | JTAG TRST | | | JTAG TRST signal. |
| J11 | JTAG RTCK | | | JTAG RTCK signal. |
| J12 | JTAG TDI | | | JTAG TDI signal. |
| J13 | Ethernet Speed | | | Connect to Ethernet Connector Speed LED. High = 100 Mbps Low = 10 Mbps |
| J14 | Ethernet Link | | | Connect to Ethernet Connector Link LED. High = Ethernet activity. |
| J15 | GND | | | Connect to Ground. |

* Interrupt capable input.

5. EMX On Boot Up

EMX includes three pieces of embedded software, GHI boot loader, TinyBooter and EMX firmware.

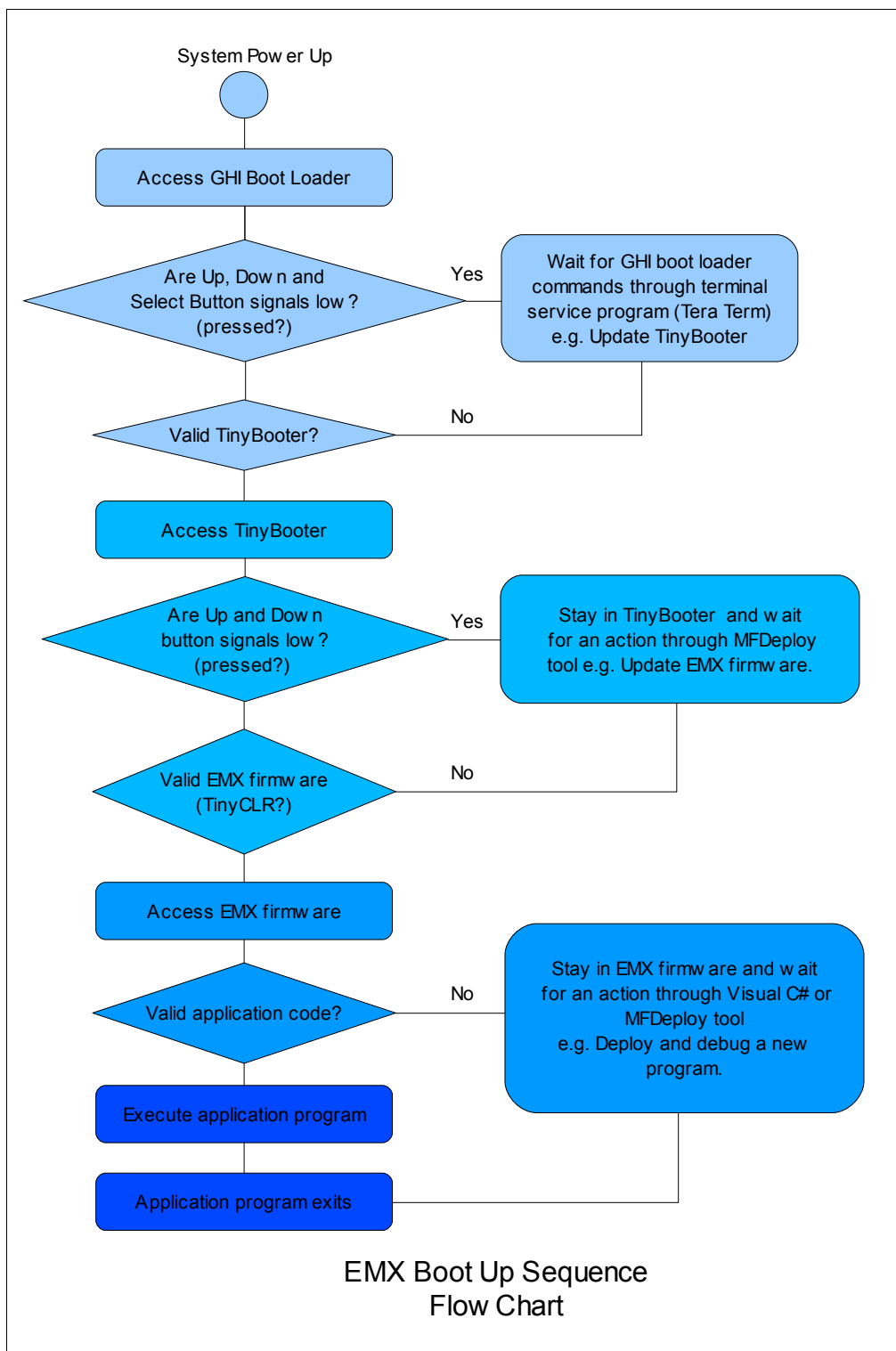
On system boot up, GHI boot loader initializes Flash and RAM memory then it looks for a valid TinyBooter and lets it execute from RAM. After TinyBooter takes over the hardware, it prepares the resources to be handled by EMX firmware. EMX firmware is the main software that runs .NET Micro Framework core and the user managed application.

During boot up a user can interrupt the sequence to remain in boot loader, TinyBooter, or firmware by changing the state of the following signals on **start-up**:

| Pin 7 | Pin 3 | Pin 53 | Description |
|---------------------|---------------------|----------------------|---|
| Up Button signal | Down Button signal | Select Button signal | |
| High or unconnected | High or unconnected | High or unconnected | This indicates the user has no interference on boot up process, and the system will boot in normal mode sequence. |
| Low | Low | Low | Hold the system in GHI boot loader mode access |
| Low | Low | High or unconnected | Hold the system in TinyBooter mode access |

These pins are exposed on EMX Development System to Up, Down and Select buttons with a high default state. In other words, the pin is low when the button is pressed.

The following flow chart clearly explains the boot up sequence:



5.1. GHI Boot Loader vs. TinyBooter vs. EMX Firmware

The following table lists the major properties of each software:

| GHI Boot Loader | EMX TinyBooter | EMX firmware |
|---|---|---|
| Used to update EMX TinyBooter or for low level EMX flash maintenance. | Used to update EMX firmware, maintenance application code region, get system information and to update system configurations such as networking settings. | Used to deploy, execute and debug the managed NETMF application code. In other words, it plays the role of a virtual machine. |
| Emergency use or when GHI releases a new TinyBooter. | frequently used | always used |
| Pre-burnt on the chipset's flash memory. | The user can download to EMX Module, through GHI boot loader for instance. | The user can download to EMX Module, through TinyBooter for instance. |
| Fixed and can not be updated. | Latest file is included with every GHI NETMF SDK, not necessarily changed in every new SDK | Latest file is included with every GHI NETMF SDK, not necessarily changed in every new SDK |
| Access interface can be USB or COM1 serial port on EMX Module. | Access Interface (debugging interface) can be USB or COM1 serial port. | Access Interface (debugging interface) can be USB, Ethernet or COM1 serial port. |
| User interface is a simple command line interface through any terminal service software such as TeraTerm or Hyper Terminal. | User access it through MFDeploy tool to maintain firmware, configurations (networking, USB) and application code region. | Users access it through Microsoft Visual C# to deploy, execute and debug the managed NETMF application through the debugging interface. Users can access it using Microsoft NETMF MFDeploy tool to maintain the firmware or application code region. |
| Very compact to accomplish only the flash memory and firmware maintenance functions. | Compact enough to handle the assigned functions | Highly sophisticated with .NET Micro Framework and requires HAL and PAL drivers to provide the various EMX features. |

Next sections provide more details.

5.2. EMX Access Interface

Access interface is defined as the EMX Module's hardware interface used by the user's station to access EMX's GHI boot loader, TinyBooter and EMX Firmware (TinyCLR).

With EMX, access interface can be USB or Serial Port (COM1) which is a UART interface. Hardware designer chooses between these two interfaces by setting/resetting LMODE pin using a 10K pull-up or pull-down resistor.

| LMODE | | High or unconnected | Low |
|--------------------------------|------------------------|--|--|
| H/W Access Interface | | Serial Port (COM1) TTL levels | USB Client |
| Drivers and Interface Settings | GHI boot loader | No driver needed Baud rate: 115200 Bus width: 8 bits Parity: none Stop bits: 1 bit Flow Control: none | Driver: In GHI NETMF SDK, GHI Bootloader Interface. (USB CDC device, Virtual serial port). Serial port settings (baud rate, bus width ...etc) are ignored. |
| | TinyBooter | | Driver: In GHI NETMF SDK, GHI NETMF Interface. |
| | EMX Firmware (TinyCLR) | | Driver: In GHI NETMF SDK, GHI NETMF Interface. |

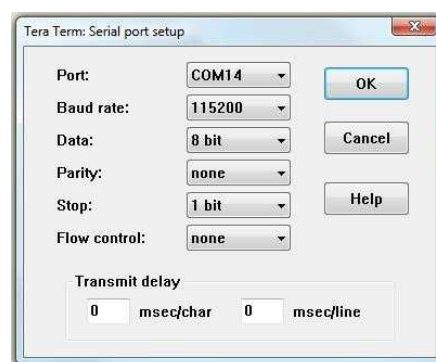
Note 1: Up (IO4), Down (IO0) and Select (IO30) button pins in addition to LMODE (IO71) pin can still be used from managed code after EMX firmware boots up.

Note 2: On **EMX Development System** LMODE pin is pulled down low to the ground with 10K resistor (R1) making USB the default access interface. Users can change it back to serial by connecting LMODE to 3.3 volts, or by removing R1 (not recommended).

Emergency GHI Boot Loader Access

EMX provides an emergency direct access to GHI boot loader through COM1 which can be done by sending the % character continuously and quickly to COM1 on power up from terminal service software (TeraTerm) with the following settings:

- Baud rate: 115200
- Bus width: 8 bits
- Parity: none
- Stop bits: 1 bit
- Flow Control: none



Other Interfaces

You can set other access interfaces and even save them to the device using software. In case problems occur for the access interface, holding center and down buttons upon start-up will force EMX to ignore the software settings and use the LMODE pin as described above.

Please see [debug interface](#) section for details.

6. GHI Boot Loader

EMX Boot Loader is a software developed by GHI and is included on all EMX Module. It is used to update the EMX TinyBooter or for low level EMX flash maintenance.

GHI boot loader accepts simple commands in ASCII characters sent with help of a terminal service software (TeraTerm). Thus, the user sends the desired command character and the boot loader performs an action. The results are returned in a human friendly format followed by a "BL" indicating that the boot loader is ready for the next command.

[EMX on boot up](#) section provides the required information on how to choose access interface and how to access GHI boot loader.

GHI boot loader is different than TinyBooter or EMX firmware, [GHI boot loader vs. TinyBooter vs. EMX firmware](#) section lists the features and properties of each piece of software.

6.1. GHI Boot Loader Commands

| CMD | Description | Notes |
|-----|--|---|
| V | Returns the GHI Loader version number. | Format X.XX e.g. 1.06 |
| E | Erases the Flash memory (except the boot loader region). | Confirm erase by sending Y or any other character to abort. This command erases EMX firmware and the user's application region. |
| X | Loads the new TinyBooter file | TinyBooter Update section explains this command process in more detail. |
| R | Runs firmware. | Exits GHI boot loader mode and forces running TinyBooter. |
| H | Returns the hardware version | e.g. X11 (EMX 1.1) |
| B | Changes the baud rate to 921600 | User needs to change the baud rate on the terminal service accordingly. (Serial Port access interface only) |

6.2. TinyBooter Update through GHI Boot Loader

At power up, a GHI boot loader takes over the processor and validates TinyBooter stored in FLASH. If TinyBooter was found and was valid, execution is transferred to TinyBooter. More information about TinyBooter is available in these sections [EMX On Boot up](#) and [TinyBooter](#) section.

Usually, a user would never need to update TinyBooter as it is not used in the final application but for very rare cases especially when changing to a different .NET Micro Framework version - e.g. 3.0 to 4.0 - **or when it is mentioned in the release notes of a new GHI NETMF SDK to update the TinyBooter**, there is a way to update it through GHI boot loader.

First, it is better to start fresh before loading the new firmware:

1. Access the boot loader using TeraTerm as explained earlier in [EMX on boot up](#) section.
2. Erase the flash memory using **E** command then press **Y** to confirm (this will take several seconds).
3. Loading new firmware is simple but it requires a terminal that supports XMODEM file transfer. XMODEM has many versions, GHI boot loader requires 1K transfers with 16Bit CRC error checking. Keep on using TeraTerm software.

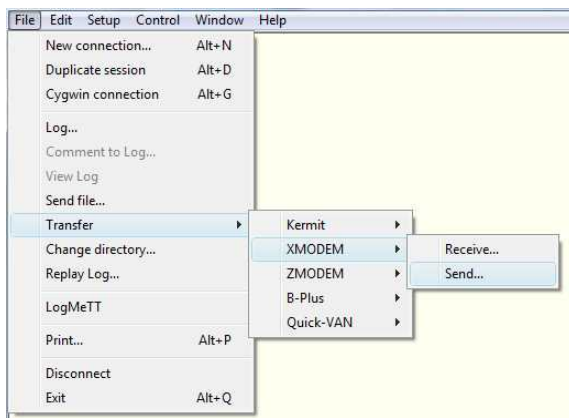
Transfer is initiated using the **X** command. After the **X** command is entered, the GHI boot loader will start sending back the “C” character continuously. This “C” is an indicator that tells XMODEM a device is waiting for data. Once you see the “C” character appearing on the terminal window, you can select XMODEM transfer and point the software to the firmware file "TinyBooter.GHI."

Entering **X** command...



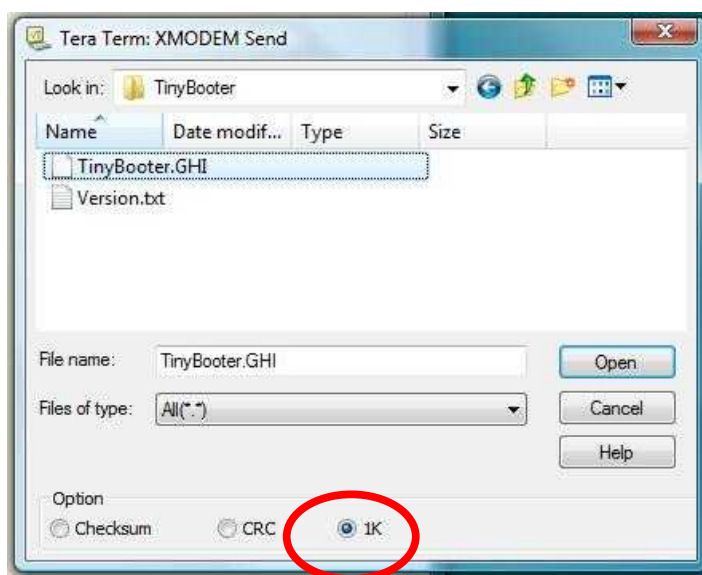
```
BL
BL
BL
BL
1.01
BL
Start File Transfer
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
```

In the menu, select **File > Transfer > XMODEM > Send...**



Next, select the [TinyBooter.GHI](#) file from...

GHI Electronics\GHI NETMF SDK\EMX\Firmware\TinyBooter



Updating the firmware takes a few seconds to load. Once loading has finished and the file is valid, the new firmware is executed automatically and you will not see “BL” again.

7. TinyBooter

EMX Module implements embedded software from Microsoft, called TinyBooter. This software can be used to update the EMX firmware, to maintain the application code region, to get system information and to update system configurations such as networking settings.

[EMX on boot up section](#) provides the required information on how to choose an access interface and how to access TinyBooter.

TinyBooter is different than GHI boot loader or EMX firmware, [GHI boot loader vs. TinyBooter vs. EMX firmware](#) section lists the features and properties of each software.

Typically, a user would never need to update TinyBooter as it is not used in the final application. For rare cases, especially when changing to a different .NET Micro Framework version -- e.g. 3.0 to 4.0 -- **or when it is mentioned in the release notes of a new GHI NETMF SDK to update the TinyBooter**, there is a way to update it through GHI boot loader.

TinyBooter update can be done [through GHI boot loader](#) or it can be done with [In-Field Update feature](#).

The end-user software interface that communicates with TinyBooter is [MFDeploy](#), a tool provided with Microsoft .NET Micro Framework SDK.

The following section explains how to access TinyBooter and update EMX firmware through TinyBooter using MFDeploy.

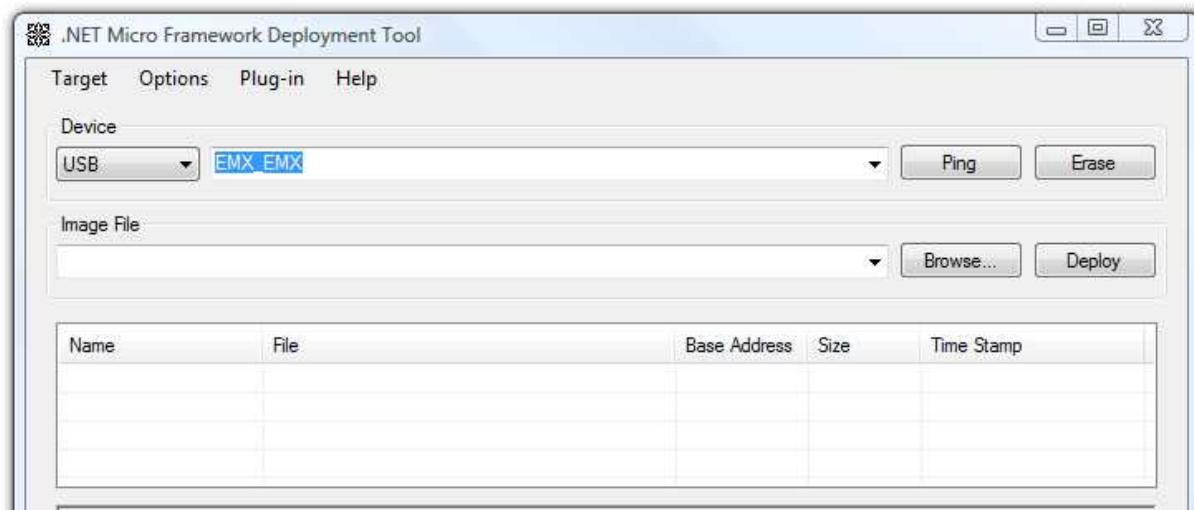
7.1. EMX Firmware Update Through TinyBooter

The objective of this section is to provide simple steps to access TinyBooter on your EMX-based system from your PC, so you're ready to update EMX firmware using [MFDeploy](#).

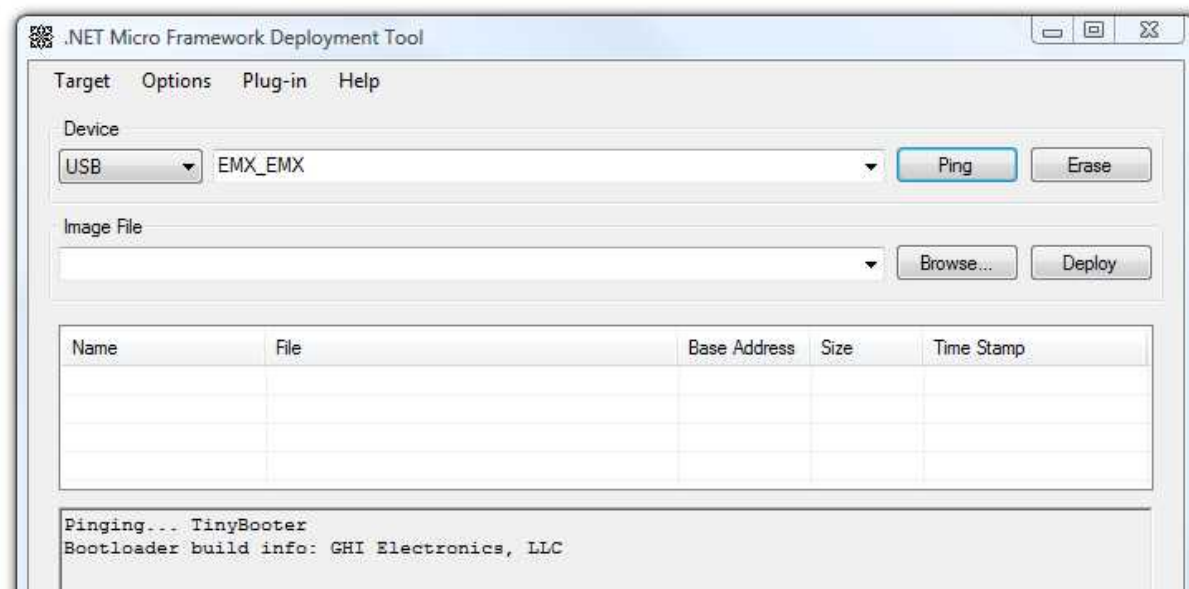
In the following steps it is assumed that the user is using the USB access interface with the GHI NETMF interface driver installed. Refer to the [EMX access interface section](#) for more details.

1. First, install the latest GHI NETMF SDK (which includes EMX firmware).
2. Insure there is no need to update the TinyBooter. This information is usually mentioned in the GHI NETMF SDK release notes. If a new TinyBooter is needed, [update the TinyBooter](#) then update the EMX firmware.
3. Press and **hold** down Up and Down buttons then press and **release** Reset button. Once you see Tinybooter mode on the LCD, you may release Up and Down buttons. Refer to the [EMX on boot up](#) section to learn about the boot up sequence.
4. Run MFDeploy and select USB from the Device list, you should see EMX_EMX in

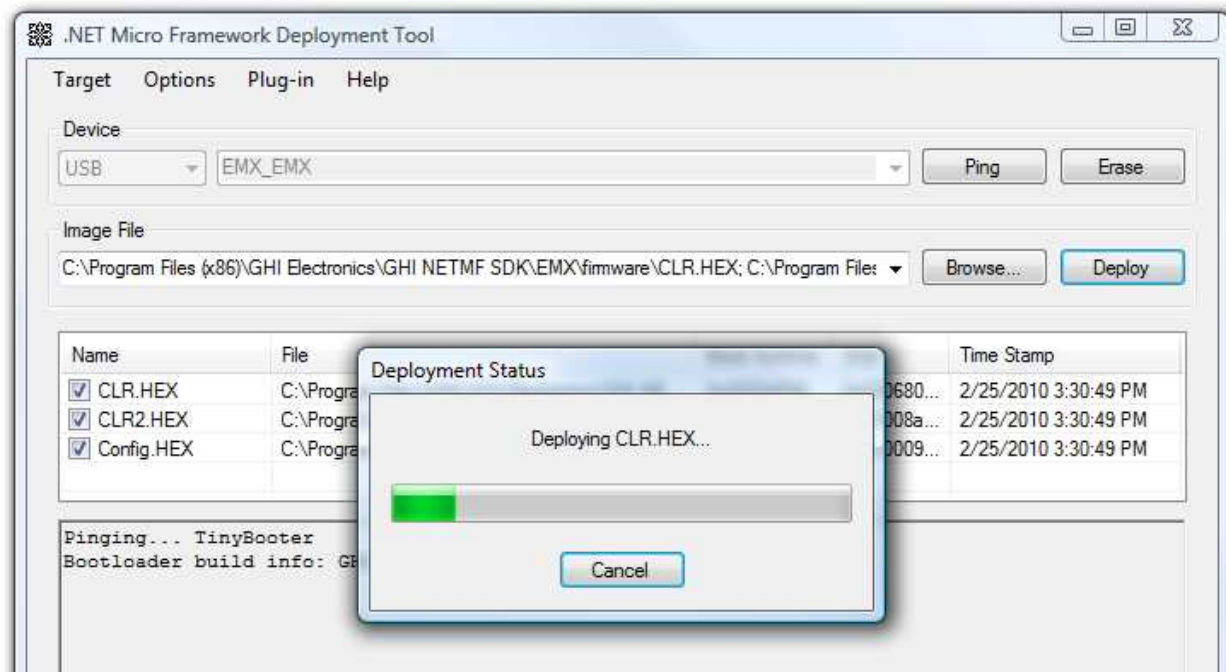
the dropdown.



5. Check the communication between MFDeploy and TinyBooter by pinging the device. Press Ping and you should see this message:



6. Now we can lead MFDeploy to the new EMX firmware files. Click Browse and direct MFDeploy to the firmware HEX files. These can be found under EMX\firmware folder in the SDK. The other files with “sig” extension must exist in the same folder as the HEX files. Select **ALL** of the HEX files at once and start deploying the firmware by pressing Deploy.



7. Loading the files takes about a minute, on completion the firmware will execute. Double check the version number to make sure the correct firmware is loaded.
8. Loading new firmware will not erase the deployed managed application. If you need to erase the managed application click Erase.

Important Note: If you see a message after updating the EMX firmwar on the LCD or on EMX access debugging interface stating you need to update the Tinybooter, the TinyBooter version is not suitable for the current firmware. In this case, [update the TinyBooter](#) then update EMX firmware again.

8. EMX Firmware

EMX firmware is the main piece of embedded software in the EMX Module which hosts .NET Micro Framework core with the required HAL drivers to provide the various EMX features a user can control with C#. A user deploys and debugs the managed application code directly on EMX Module from Microsoft Visual Studio through EMX debugging interface.

[EMX on boot up](#) section provides the required information on how to choose an access interface and how to access EMX firmware.

EMX firmware is different than TinyBooter or GHI boot loader, [GHI boot loader vs. TinyBooter vs. EMX firmware](#) section lists the features and properties of each piece of software.

User can update EMX firmware through TinyBooter. Refer to the [TinyBooter](#) to learn how to update the Firmware. EMX firmware can be updated with [In-Field Update feature](#).

The end-user software interface that communicates with EMX firmware is MFDeploy, which comes with Microsoft .NET Micro Framework SDK and Microsoft Visual C# with installed .NET Micro Framework SDK.

8.1. Getting Started with EMX

The objectives of this section are to provide simple steps to setup your EMX-based system on your PC, so you're ready to develop your application on Visual Studio C# with .NET Micro Framework.

All you need to start up

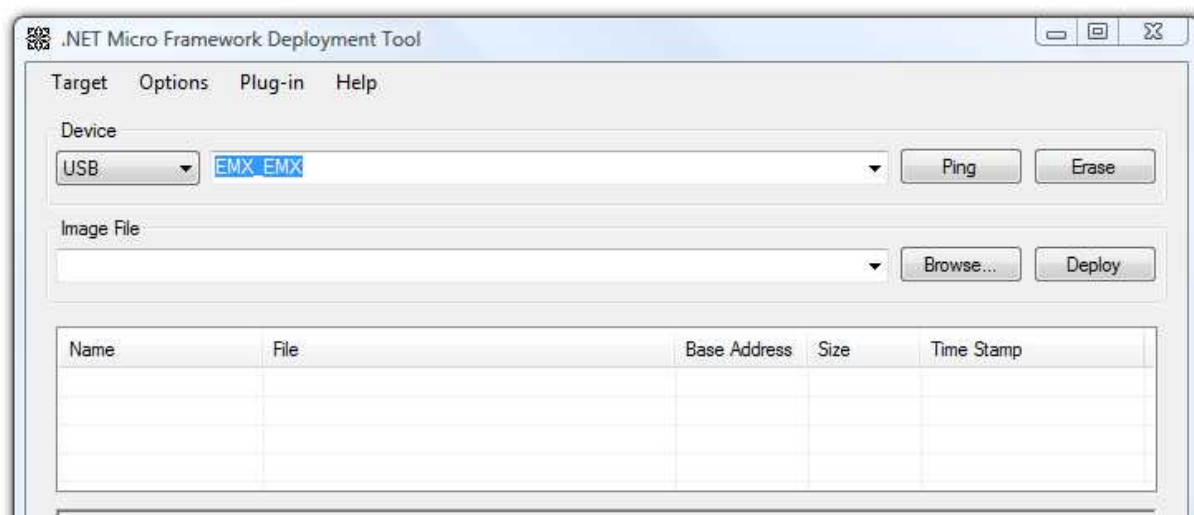
- EMX-based system such as EMX Development System or you custom hardware design.
- USB cable.
- Microsoft Visual Studio 2010 or Microsoft [Visual C# Express 2010](#) (free download) installed with the latest updates.
- Microsoft [.NET Micro Framework SDK Version 4.1](#).
- Latest GHI NETMF SDK, available on GHI Electronics website.

If you got a new EMX Development System, it is recommended that you [update EMX firmware](#) and [TinyBooter](#) if needed, use the files available in the latest GHI NETMF SDK within EMX folder before you start these steps.

The suggested access interface in these steps is USB (the default on EMX Development System).

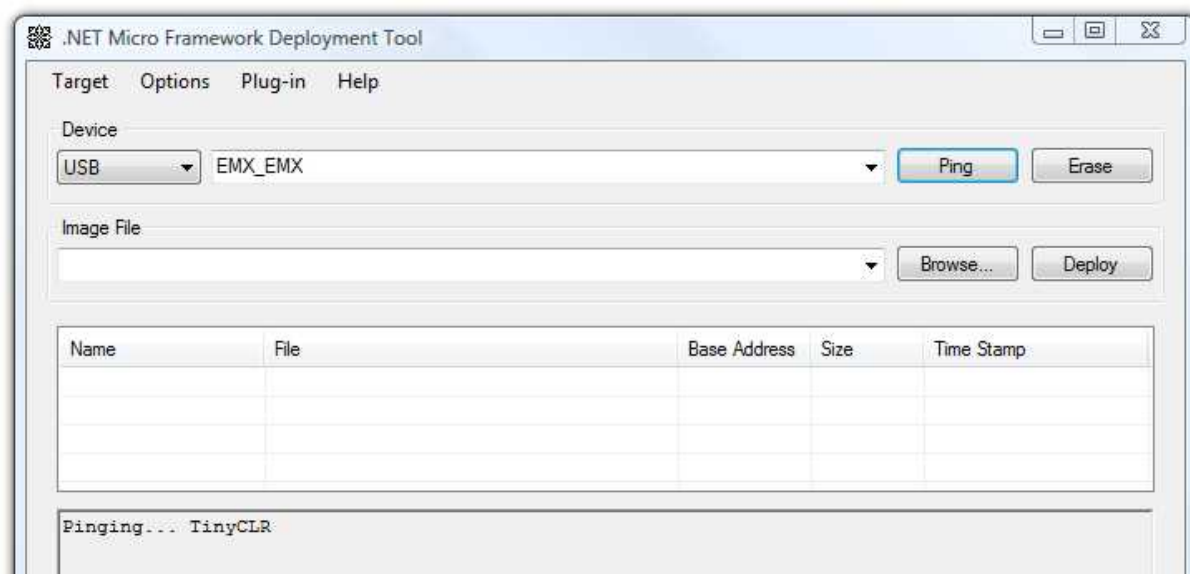
First Power-up

1. Install the latest Microsoft .NET Micro Framework SDK Version 4.1.
2. Install the latest GHI NETMF SDK.
3. LMODE = Low. (Skip this step if you are using EMX Development System since LMODE is [pulled down low](#) on this development system).
4. Up, down, and select buttons are high or unconnected. (Leave up, down and select unpressed on EMX Development system to accomplish this step)
5. Power up the system (connect USB cable).
6. Run the [MFDeploy tool](#) and choose USB from the Device list, and you'll see *EMX_EMX* appear in the dropdown.

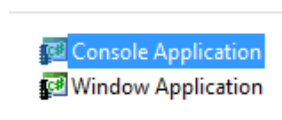


Note: If you did not see that string you may have a different default debugging interface (you might have installed the driver incorrectly, or the processor is shutdown).

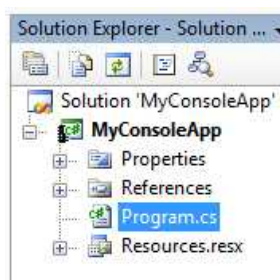
7. Press Ping on MFDeploy. It should return "TinyCLR." This verifies that the board is responsive. Be sure to review the [MFDeploy description in appendix A](#).



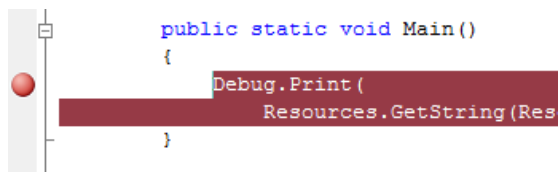
8. Open Visual Studio and start a new Micro Framework project with the “Console Application” template. This is the simplest application that can be loaded. All it does is print a string to the debug output. Name your project “MyConsoleApp.”



9. Visual Studio will now generate all the needed project files. One of the files is called *Program.cs*, open it...

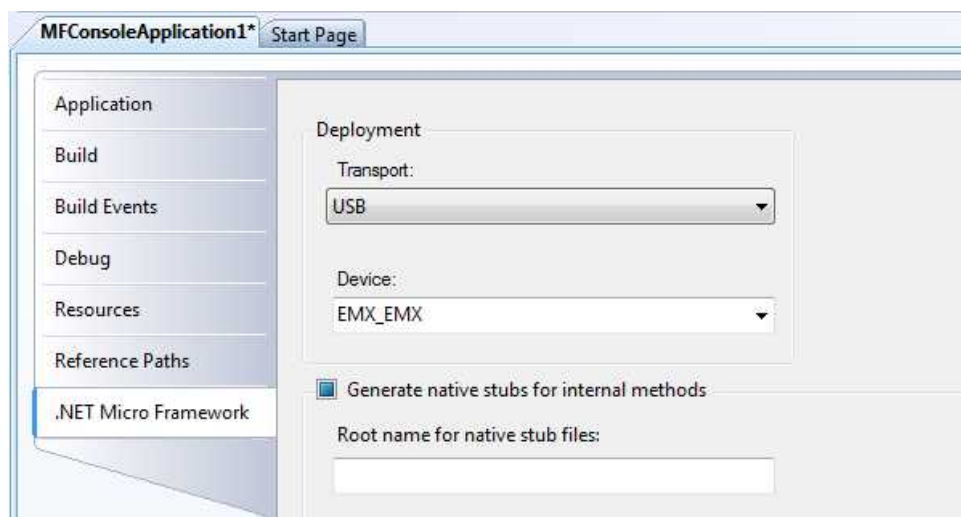
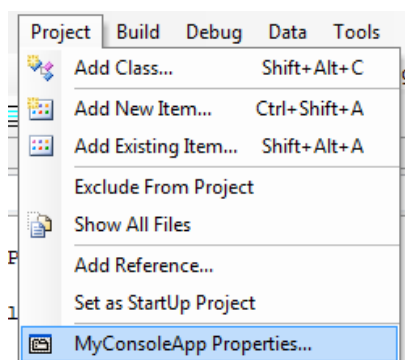


10. Place a breakpoint on the `Debug.Print` line. You can do this by clicking on the line and pressing F9.

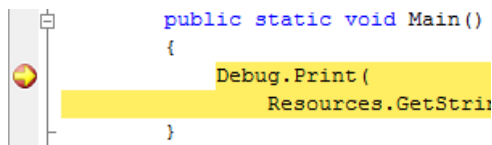


```
public static void Main()  
{  
    Debug.Print(  
        Resources.GetString(Res  
    )  
}
```

11. Compile the application. There should be no errors.
12. Go to the menu and select **Project > MyConsoleApp Properties...** and in the new window select the “.NET Micro Framework” tab. In the tab, there are options for deployment. Select *USB* from the Transport drop-down and select *EMX_EMX* from the Device drop-down.

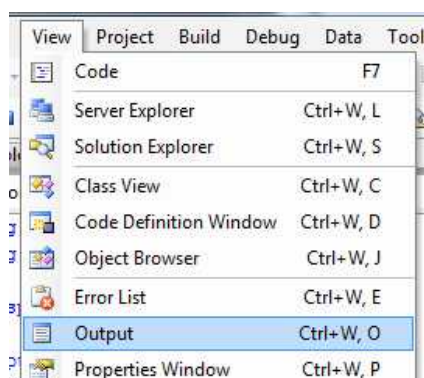


13. Press F5 (Debug) and you'll see how Visual Studio loads the application and runs it. Visual Studio should pause at the breakpoint we placed in step 4.

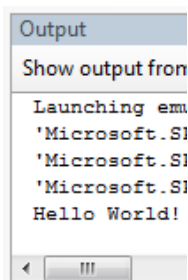


```
public static void Main()  
{  
    Debug.Print(  
        Resources.GetString(  
    )  
}
```

14. Make sure you have the Output window open. If not, you can open the Output window from **View > Output**.



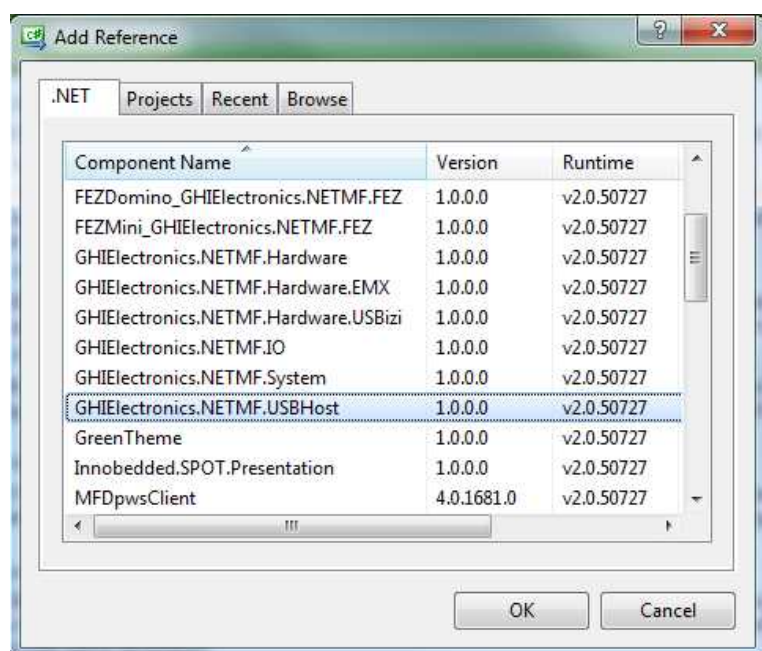
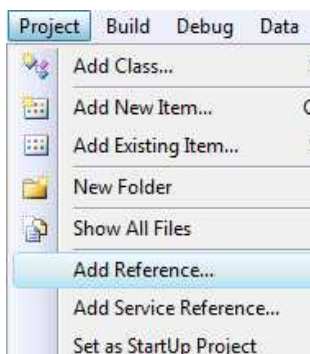
15. Press F10 to step over Debug.Print and watch the Output window. The Output window should display “Hello World!”



16. Press F5 and the code will continue executing until it reaches the end of the program.

Adding GHI NETMF Library

1. Go to the **Project** tab and click **Add Reference**.



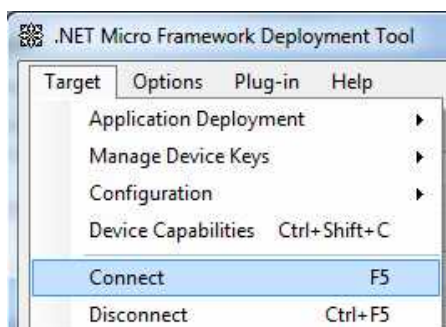
2. Let's add USB Host library. Select it and click OK.
3. Add "using" for the name space at the beginning of the file:
`using GHIElectronics.NETMF.USBHost;`
4. As an example, we will get a list of currently connected devices. Add this in Main() method:
`USBH_Device[] devices = USBHostController.GetDevices();`
5. Similarly, you can use any other functionality provided by GHI library. Press F5 in visual studio and the program will run.

If the program does not run, then there is something incompatible on your system. For example, you are using a newer or incorrect version of the GHI library and older or incorrect version of the firmware is running on your hardware. This is simply resolved by upgrading the firmware to the one included in your SDK and making

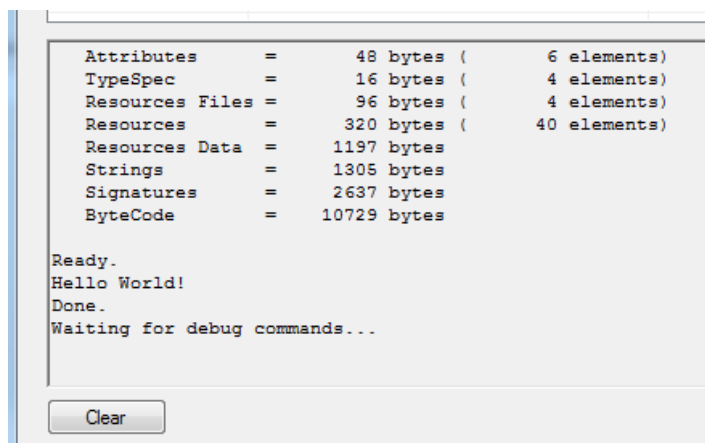
sure the Added Reference is from the SDK as well.

MFDeploy is helpful to investigate these errors as explained next.

Using MFDeploy, you can see any debug messages, exceptions or errors from your device. Make sure Visual Studio is not in debug mode or close it. Open MFDeploy and make sure you can ping as explained in previous steps. Now, Click on **Target->Connect**.



Now, reset your hardware and click ping. You will see debug output of what the device is doing, for example loading assemblies and any debug messages printed by your application.



In case the program did not run because of incompatibility, the debug output will show these errors. This is useful for debugging certain applications.

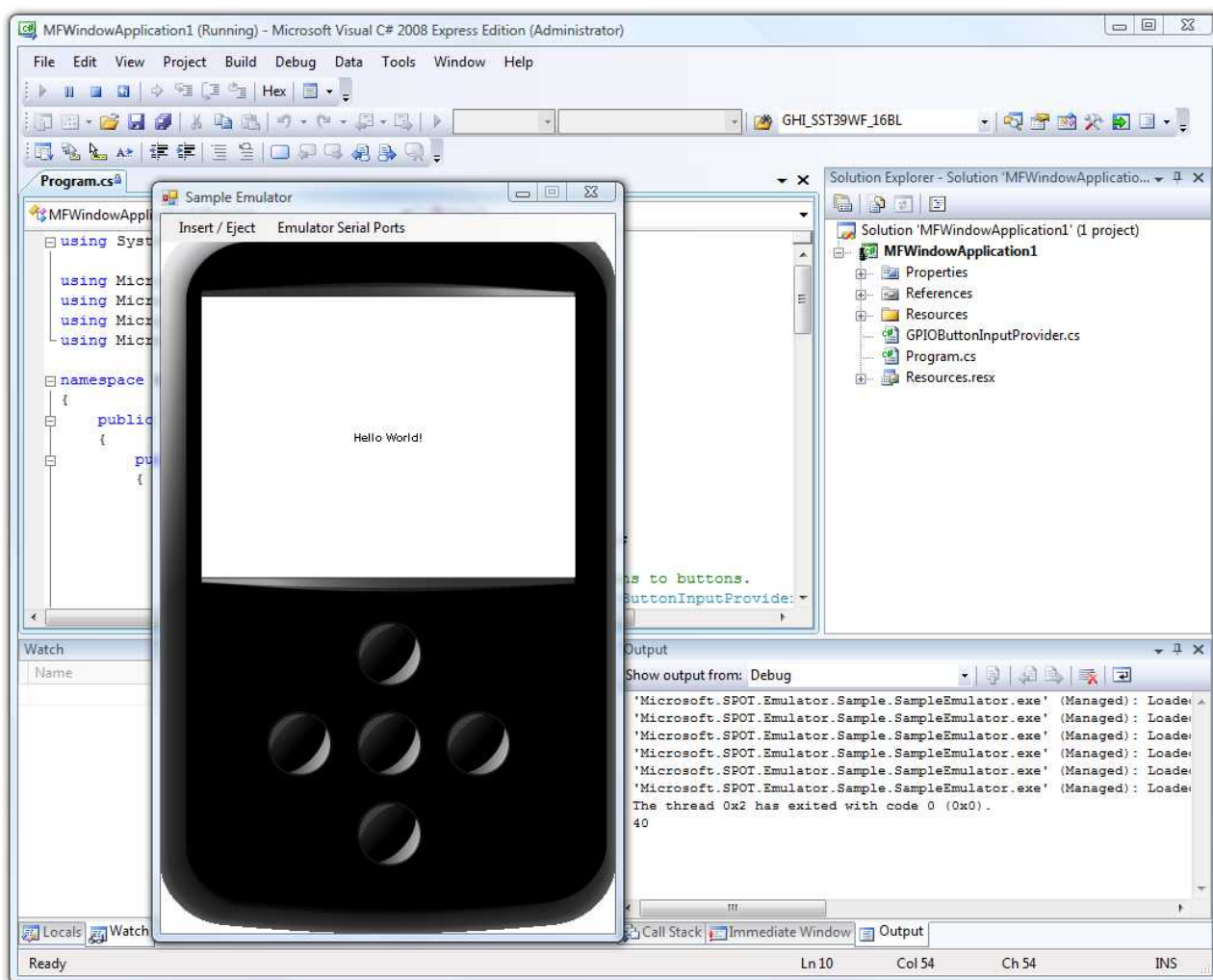
Note: If you Connect through MFDeploy, you cannot deploy using Visual Studio anymore. MFDeploy must be disconnected or closed first and then you can go back to Visual Studio. Only one of these two programs can be connected to your platform at one time.

8.2. EMX Emulator

.NET Micro Framework has a powerful emulator that can be extended or changed to suite the developer's needs. This is very useful as you can do most of the development and testing before building the actual hardware.

EMX has an emulator available that maps the buttons and LCD dimensions as provided on the Development System. However, support for the extended features provided by GHI (PWM, USB Host, etc.) are not supported in the emulator. Using any of these extended features will result in an error on the emulator.

A user can choose the emulator from the Device list in Visual studio project properties.



9. EMX Module Features

EMX firmware supports all the necessary features of .NET Micro Framework version with all the required HAL and PAL drivers such as FAT File System. .NET Micro Framework SDK includes full documentation and examples about the usage of these features with the related libraries.

Furthermore, EMX supports other exclusive GHI hardware and software features such as USB host, PWM, ADC and DAC. The SDK includes the required library files with full documentation and examples about the usage of these features with the related libraries.

The following sections clarify necessary guidelines about EMX features.

9.1. Application Flash/RAM/EWR

4MB of external flash is available on EMX Modules. This doesn't include the 0.5MB internal flash used for Micro Framework CLR execution. External flash is used for system assemblies, boot loader, user deployment and EWR storage.

About 3MB is reserved for deployed managed applications, including resources.

16MB of SDRAM comes standard with EMX Module. Enough for applications using .NET Micro Framework and SideShow.

Extended Week References (EWR)

EWR is a way for managed applications to store data on non-volatile memory. Consult .NET Micro Framework documentation for more details.

256KB of flash memory is reserved for EWR.

If more storage is needed, SD memory cards and/or USB memory devices can be used. EWR does not work with removable media devices.

9.2. Debugging Interface (Access Interface)

[Access Interface](#) with EMX firmware is usually named NETMF debugging interface which is the communication interface between EMX firmware and the application code terminal (Visual C# debugger). It can be configured as USB, serial port or Ethernet.

Access Interface section provides the required information on how to access EMX debugging interface.

Changing the debug interface might be necessary for some applications. The default debug interface is USB, but some application might need to use the USB Client feature to connect to PC as a different device, for example a USB Storage. In this case, you should change the debug interface.

Other access interfaces can be enabled using software. Using GHI library, you can set the interface and it is saved. So, it will keep this setting after you reset the device. Only TinyCLR (Firmware) and TinyBooter interfaces can be changed. The Bootloader cannot be changed using software.

You can force EMX to ignore the software settings and use LMODE pin to select the debug interface (USB or COM1). This is helpful in case the incorrect settings are stored. This is done by holding Center and Down buttons upon startup. Note that for Embedded Master (the older version) only, holding Center and Down buttons will force the interface to COM1 because the LMODE pin is not available.

If you are not able to access the device after setting the debug interface, for example it was set incorrectly, you can reboot the device in bootloader mode, erase and update TinyBooter and firmware again.

Software settings is done using GHI NETMF library under:

GHIElectronics.NETMF.Hardware.[Configuration](#)

9.3. Digital Inputs/Outputs

All Digital IO pins are 3.3V and 5V tolerant. This means that signals coming from another circuits can be 5V (e.g. connecting EMX to a 5V microcontroller).

All pins support input and output with pull-up and pull-down resistors.

Refer to the [Pin-Out Description](#) section for more information about Digital I/O assignment to EMX hardware pins.

Most digital I/O pins are interrupt capable. Interrupt pins asynchronously call functions in managed applications. Interrupts can be activated on rising or falling edges with an optional glitch filter. Enabling interrupts for both rising and falling edges is supported but in this case the glitch filter is disabled. Interrupt capable pins are marked in the pin-out table.

Important Note: Inputs are 5V tolerant but EMX cannot be powered by 5 volts.

9.4. Serial Peripherals

Serial Port (UART)

One of the oldest and most common protocols is UART (or USART). EMX hardware exposes four UART ports

| Serial Port | LPC2478 UART | Hardware Handshaking |
|-------------|--------------|----------------------|
| COM1 | UART0 | Not Supported |
| COM2 | UART1 | Supported |
| COM3 | UART2 | Not Supported |
| COM4 | UART3 | Not Supported |

Important Note: Serial port pins have 3.3V TTL levels where the PC uses RS232 levels. For proper communication with RS232 serial ports (PC serial port), an RS232 level converter is required. One common converter is MAX232.

Note: If the serial port is connected between two TTL circuits, no level converter is needed but they should be connected as a null modem. Null modem means RX on one circuit is connected to TX on the other circuit, and vice versa.

Refer to the [Pin-Out Description](#) section for more information about UART signals assignment to EMX hardware pins.

SPI

EMX supports two SPI interfaces, SPI1 and SPI2. SPI Bus is designed to interface multiple SPI slave devices, the active slave is selected by asserting Chip Select line on the relative slave device.

Refer to the [Pin-Out Description](#) section for more information about SPI signals assignments to EMX hardware pins.

I2C

I2C is a two-wire addressable serial interface. EMX supports one master I2C port.

Refer to the [Pin-Out Description](#) section for more information about I2C signals assignments to EMX hardware pins.

CAN

Controller Area Network is a common interface in industrial control and automotive. CAN is remarkably robust and works well in noisy environments. All error checking and recovery methods are done automatically on the hardware. TD (Transmit Data) and RD (Receive Data) are the only pins needed. These pins carry out the digital signals that need to be converted to analog before it can be used. There are different CAN transceivers. The most common one is dual-wire high speed transceivers, capable of transferring data up to 1MBit/second.

Refer to the [Pin-Out Description](#) section for more information about SPI signals assignments to EMX hardware pins.

This is available through GHI NETMF library.

One-wire

Through one-wire a master can communicate with multiple slaves using a single digital pin. One-wire can be activated on any Digital I/O on EMX.

This is available through GHI NETMF library.

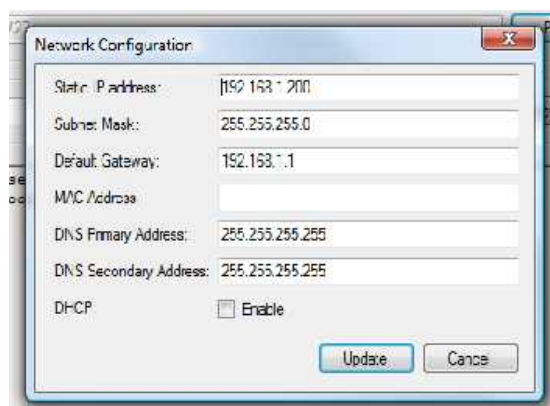
9.5. Networking (TCP/IP)

Networking is a crucial part today's embedded devices. .NET Micro Framework includes a full TCP/IP stack with complete socket support for managed applications. EMX networking implementation includes PPP, WiFi, Ethernet, TCP/IP, SSL, HTTP, and Device Profile for Web Services.

MAC address setting

User can use MFDeploy to update the correct MAC address before the device is connected to a network. Network settings can also be changed dynamically from the managed code.

```
NetworkInterface[] netif = NetworkInterface.GetAllNetworkInterfaces();  
// Set new MAC address  
byte[] newMAC = new byte[] { 0x00, 0x1A, 0xF1, 0x01, 0x42, 0xDD };  
netif[0].PhysicalAddress = newMAC;
```



IP address (DHCP or static):

DHCP (dynamic) IP and Static IP are supported when using Ethernet or WiFi on EMX. If using dynamic IP, EMX will not obtain IP lease at power up. DHCP can only be enabled from software. MFDeploy has a DHCP enable option but it has no effect on getting the IP lease on startup.

```
NetworkInterface[] netif = NetworkInterface.GetAllNetworkInterfaces();  
// Get an IP address from DHCP server  
if (netif[0].IsDhcpEnabled)  
{  
    netif[0].RenewDhcpLease();  
}  
else  
{  
    netif[0].EnableDhcp();  
}
```

Ethernet

EMX Module hardware includes an industrial Ethernet PHY along with the needed circuitry. The Ethernet oscillator is controlled by the processor allowing the user to control its power consumption. The designer only needs to wire the signals to the Ethernet connector. The recommended Ethernet connector is J0011D01BNL.

Refer to the [Pin-Out Description](#) section for more information about Ethernet signal assignments to EMX hardware pins.

GHI Electronics supplies a dedicated MAC address for each EMX Module. The MAC address is printed out on a ROHS static Dissipative Polyimide label, compatible with Surface Mount Technology.

Wireless LAN WiFi (IEEE 802.11b)

EMX includes drivers for ZeroG ZG2100/ZG2101 SMT modules. Which are SPI bus-based, low-priced and FCC certified. The only difference between ZG2100 and ZG2101 is that ZG2100 hosts an on-board antenna and ZG2101 includes a connection for an external antenna.

Note: the MAC address is provided on the SMT module.



ZG2100 module front view



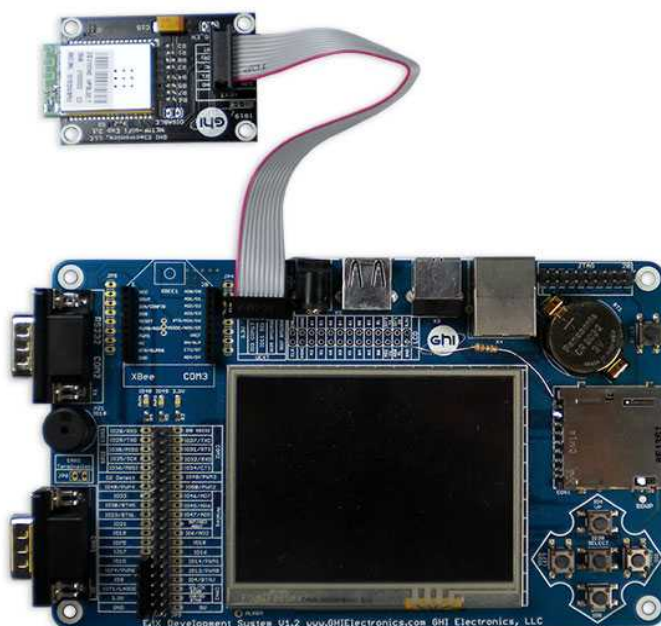
ZG2100 module back view

GHI Electronics LLC is an ZeroG authorized design partner:

<http://www.microchip.com/zerog/partners/partnersdevelop.html>

To get started with ZeroG WiFi modules on EMX, GHI Electronics offers NETMF WiFi expansion that hosts ZeroG ZG2100 module. and can be easily plugged in ChipworkX, EMX or Embedded Master development system.

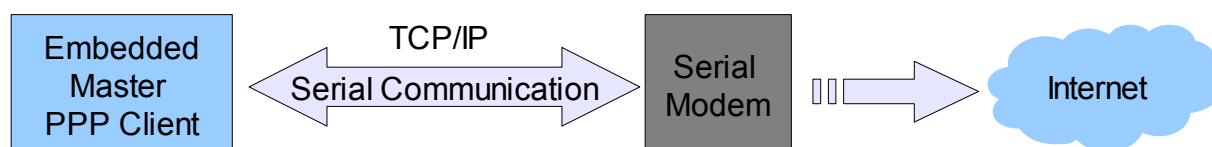
<http://www.ghielectronics.com/product/126>



PPP (TCP/IP access through serial modems)

Using this feature, users can create sockets and communicate over links that are not Ethernet, serial or wireless links for example.

This includes PPP Client with PAP authentication protocol. This feature allows the user to dial in through serial modem (V.90/GPRS/3G) to access the Internet or Extranet.



In this case, network settings will be taken from the hosting terminal server (e.g. Internet Service Provider).

Important Note: If the terminal server (ISP) does not require authentication credentials, a user must use this type of communication anyway with any random user name and password.

Important: Ethernet port or WiFi cannot be used when using GHI PPP Stack. but Ethernet cable or WiFi physical link can be traced.

This is available through GHI NETMF library. Example code is also included in the SDK.

SSL

.NET Micro Framework includes an SSL stack to enable secure network communication. **The user must update the SSL seed through MFDeploy before using SSL**, *MFDeploy > Target > Manage Device Keys > Update SSL Seed*.

Consult .NET Micro Framework documentation for more information about SSL.

9.6. Graphics

EMX Module supports 16Bit color displays. The default resolution is 320x240 which matches the 3.5" PT0353224T-A802 TFT display available on EMX Development System.

Developers can use almost any digital TFT display. This is accomplished by connecting HSYNC, VSYNC, CLK, ENABLE and 16Bit color lines. The color format is 5:6:5 (5Bits for red, 6Bits for green and 5Bits for blue). If the display has more than 16Bits, connect the MSB (high Bits) to EMX and the extra LSB (low Bits) to ground.

For developers wanting to connect VGA monitors, EMX supports 640x480 (actual 480x480) resolution. A simple circuit is still needed to convert the 16Bit digital signals to analog RGB colors. If a higher resolution is required, frame generator chips like Chrontel's CH7025 can be used.

Currently the highest supported resolution is 800x600. If your application requires a higher resolution, please contact us.

Refer to the EMX Development System schematic for more information about hardware

design (Back Light circuit, TFT signal connections).

Refer to the [Pin-Out Description](#) section for more information about TFT signals assignments to EMX hardware pins.

With EMX graphics support, users can leverage .NET Micro Framework graphics features such as:

- Windows Presentation Foundation (WPF)
- BMP, GIF and JPEG image files.

Consult .NET Micro Framework documentation for more information on graphics support.

9.7. Touch Screen Control

EMX Module supports displays with a four-wire resistive touch screen without the need for any additional hardware.

Refer to the [Pin-Out Description](#) section for more information about touch screen signals (YU,YD,XL,XR) assignments to EMX hardware pins.

Developers can support different kinds of touch screens and touch controllers (if needed) easily by writing a simple driver and expose the position parameters to touch screen methods.

9.8. USB Client (Device)

USB Client (device) and USB Host are completely different. Many designers confuse USB when it comes to host and device. USB Host is the master of the bus where all the work is done. USB devices are simple compared to host and they can only connect/communicate with a host and not other devices. USB host and device on EMX are two separate peripherals, so there would be no conflict when using them both simultaneously.

The USB client interface is usually used as an EMX access interface for debugging and application deployment through Microsoft Visual Studio. However, developers have full control over the USB client interface. For example, the USB client can be made to simulate a USB keyboard or USB mass storage.

Controlling a EMX USB client requires intricate knowledge of how USB works. The user should refer to .NET Micro Framework documentation for complete details on how to use this feature.

Fortunately, GHI Electronics offers a USB Client library (available in the SDK) to ease development and provide direct support for some USB devices, such as, Mass Storage (Virtual Disk) and CDC (Virtual COM Port). The library is capable of creating a USB client that's composed of multiple USB interfaces. Please refer to GHI NETMF Library for more information.

EMX Module contains USB host and USB client (both can work simultaneously).

Refer to the [Pin-Out Description](#) section for more information about USB device signals assignment to EMX hardware pins.

Important Notes:

- **Be CAREFUL when changing the USB configuration and settings**, as you go on with development and creating your USB device and connecting it to the PC, Windows might save the device information in its registry. Therefore, if you change the USB device settings/interfaces and connect it again, it might not work properly. Make sure to be careful with changing your USB device settings. You may also need to delete all the settings from Windows registry manually.
- By default, Micro Framework debug interface is USB. If you need to use the USB Client feature to build a USB device, you should select a different debug interface first (COM1).
- Make sure to select 64 bytes as the **bMaxPacketSize0** in the Device Descriptor.
- EMX uses LPC2478 as the core processor which has a fixed endpoint configuration and the user must comply with these restrictions, otherwise the USB device configuration will be refused by EMX. Here's a table of how the endpoints are assigned: (LPC24xx user manual has complete reference).

| Endpoint Number | Endpoint Type | Direction | Double Buffer |
|-----------------|---------------|-----------|---------------|
| 0 | Control | In/Out | No |
| 1 | Interrupt | In/Out | No |
| 2 | Bulk | In/Out | Yes |
| 3 | Isochronous | In/Out | Yes |
| 4 | Interrupt | In/Out | No |
| 5 | Bulk | In/Out | Yes |
| 6 | Isochronous | In/Out | Yes |
| 7 | Interrupt | In/Out | No |
| 8 | Bulk | In/Out | Yes |
| 9 | Isochronous | In/Out | Yes |
| 10 | Interrupt | In/Out | No |
| 11 | Bulk | In/Out | Yes |
| 12 | Isochronous | In/Out | Yes |
| 13 | Interrupt | In/Out | No |
| 14 | Bulk | In/Out | Yes |
| 15 | Bulk | In/Out | Yes |

USB cable connection detection

USB VBUS (USB power) can be connected, through a protection resistor, to any digital I/O to detect the presence of a USB cable.

9.9. USB Host and Supported Class Drivers

USB Client (device) and USB Host are completely different. Many designers confuse USB when it comes to host and device. USB Host is the master of the bus where all the work is done. USB devices are simple compared to host and they can only connect/communicate with a host and not other devices. USB host and device on EMX are two separate peripherals, so there would be no conflict when using them both simultaneously.

USB Host allows the use of USB Hubs, USB storage devices, joysticks, keyboards, mice, printers and more. With EMX supported class drivers, you don't have to worry about the inner workings. For USB devices that do not have a standard class, low level USB access is supported.

EMX Module contains USB host and USB client (both can work simultaneously).

Refer to the [Pin-Out Description](#) section for more information about USB Host signals assignment to EMX hardware pins.

This is available through GHI NETMF library.

9.10. Storage Devices (SD, USB MS) / File System

File System lets you create and manipulate files and folders on the connected SD and USB storage devices.

With .NET Micro Framework V4.1, FAT32 and FAT16 are supported by NETMF. The user should refer to .NET Micro Framework documentation for details on handling files and folders.

Note: FAT32 and FAT16 formats are supported, but FAT12 is not. You can format your storage device on a PC with a FAT32 or FAT16 option before using on EMX.

Before using the storage devices and accessing them with NETMF, the user must mount the file system first. This is done using the EMX library provided with the SDK. SD cards and USB storage devices are **NOT** mounted automatically.

Please refer to library documentation: [GHIElectronics.NETMF.IO.PersistentStorage](#)

SD/MMC Memory

SD and MMC memory cards have similar interfaces. EMX supports both cards and also supports SDHC (over 2GB) cards. The interface runs at 4Bits when using SD cards and 1Bit when using MMC cards.

There are two smaller versions of SD cards, mini SD and micro SD. All three card sizes are identical as far as the interface. All card sizes work with EMX.

Refer to the [Pin-Out Description](#) section for more information about SD signals assignment to EMX hardware pins.

A user might be interested in mounting or unmounting the file system on the SD card automatically when a SD card is inserted or ejected. To do this, there is a pin on the SD card connector called Card Detect which works like a switch. Connect this to a digital I/O [InterruptPort](#) on EMX and call mount or unmount appropriately.

USB Mass Storage

USB mass storage devices such as USB hard drives or memory sticks are directly supported on EMX.

Please refer to library documentation: [GHI Electronics.NETMF.IO.PersistentStorage](#)

9.11. Analog Inputs/Outputs

Analog inputs can read voltages from 0V to 3.3V with 10Bit resolution. Similarly, The analog output can set the pin voltage from 0V to 3.3V (VCC to be exact) with 10Bit resolution.

Although the pins are 5V tolerant, the ADC multiplexing is not and this can cause wrong readings on the affected pin or other analog pins. Please consult LPC24xx user manual for more details.

Refer to the [Pinout Description](#) section for more information about Analog input/output assignments to EMX hardware pins.

This is available through GHI NETMF library.

9.12. PWM

The available PWM pins have built-in hardware to generate the signals. No resources are needed to generate PWM.

Note that some PWM pins share the same timer. Changing one PWM frequency will affect the others.

PWM0 and PWM 2 share the same timer.

PWM1, PWM3, PWM4 and PWM5 share the same timer.

This is available through GHI NETMF library.

9.13. Output Compare

Using output compare developers can generate different waveforms. This is available on any digital output pin.

This is available through GHI NETMF library.

9.14. Battery RAM

EMX has 2KB of RAM that is backed-up by battery. Data is retained on power loss. The developer only needs to wire a 3V battery or a super capacitor to the VBAT pin.

This is available through GHI NETMF library.

9.15. Power Control / Hibernate

Power Control

EMX is running at 72MHz. Different low power modes are possible.

This feature is still under development.

Hibernate

Hibernate is supported to save power. The processor will go to sleep and wakeup on specific events.

This is available through GHI NETMF library.

9.16. Real Time Clock

LPC2478 includes a real-time clock that can operate while the processor is off. The developer only needs to wire a 3V battery or a super capacitor to the VBAT pin. A 32KHz crystal is already included on the EMX module. RTC also provides alarm functionality.

This is available through GHI NETMF library.

9.17. Processor Register Access

EMX Module allows direct access to the LPC2478 registers. The user can write, read or manipulate the bits as needed. This can be useful, enabling some features that may not be already exposed.

This is available through GHI NETMF library.

9.18. In-Field Update

This functionality allows devices that are deployed in the field to update their software automatically without external help. This is very useful in remote and end users' applications.

You can update the managed application only or the entire device (including GHI firmware files). Also, this feature includes a managed C# bootloader that the user will provide. This

is different from the GHI low level bootloader that already exists on the device.

This is available through GHI NETMF library.

9.19. Managed Application Protection

Using the EMX library you can disable reading the deployed application on EMX. This is useful if you need to protect your managed application against copying, tampering or disassembling.

Although extensive testing is done on this feature, GHI cannot guarantee or be held responsible for the possibility of hacking or bypassing protection.

This is available through GHI NETMF library.

9.20. Runtime Loadable Procedure RLP

A highly useful and unique feature in EMX is allowing users to load their own compiled native code (C or assembly) and run it directly through managed code. This feature is similar to the use of DLLs on PCs. RLP can be used to implement processing intensive and time-critical routines.

This is available through GHI NETMF library.

9.21. Watchdog

Watchdog is used to reset the system if it enters an erroneous state. The Watchdog is enabled with a specified timeout. The user must keep resetting the Watchdog time counter within this timeout interval or otherwise the system will reset.

This is available through GHI NETMF library.

10. Advanced Users

EMX Module is based on the NXP LPC2478 microcontroller. With EMX firmware's register access feature, advanced users familiar with NXP microcontrollers, can manipulate the internal registers. For example, COM4 (UART3) is capable of generating carrier frequencies. This feature is not available by default, but can be enabled using register access.

11. EMX Design Consideration

11.1. Hardware

The following peripherals are recommended to be exposed from the module in any design, possibly hidden from the end user:

- EMX access interface Serial COM1 (pins 5, 6 note: TTL level), USB Device (pins 41, 42) or both.
- Up (pin 7), down (pin 3) and select (pin 53) buttons.
- LMODE (pin J2) can be set to high or low (high if left unconnected). Important in case you need to change the access interface.
- The recommended Ethernet connector is J0011D01BNL.
- Refer to EMX Development System schematic for more details about hardware design.

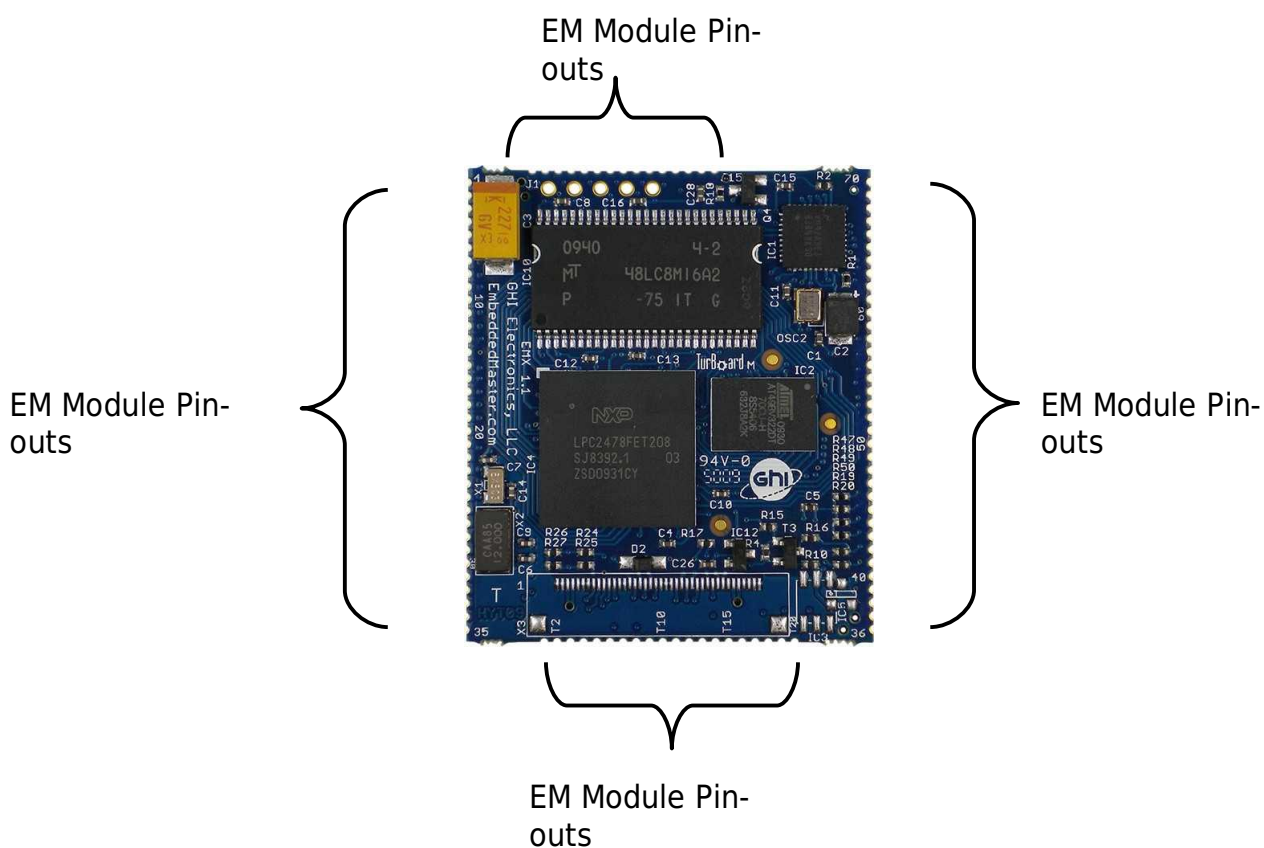
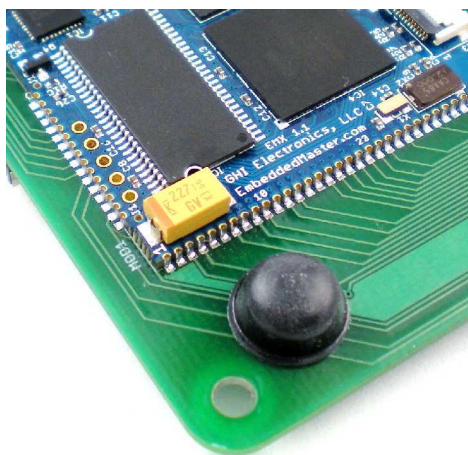
11.2. Software

When EMX module is shipped, do not assume what firmware is included. The firmware has to be downloaded through an EMX access interface using MFDeploy starting with [downloading TinyBooter](#).

If the user is using the in-field firmware update feature (without EMX access interface), it would be a good idea to have an EMX access interface exposed in case of firmware update failure, for example a power loss.

11.3. EMX Placement

EMX Module was designed to be easily placed and soldered, by machine or technician. This image shows a manually-soldered module. Static sensitive precautions should take place when handling the modules.



Machine Placement

When electrical components are machine placed, they are under high temperature for a short time. This is needed to reflow the components. Devices that are not sealed from humidity should be baked before they are used in machine placement. This is a standard procedure and EMX needs to go through this process as well.

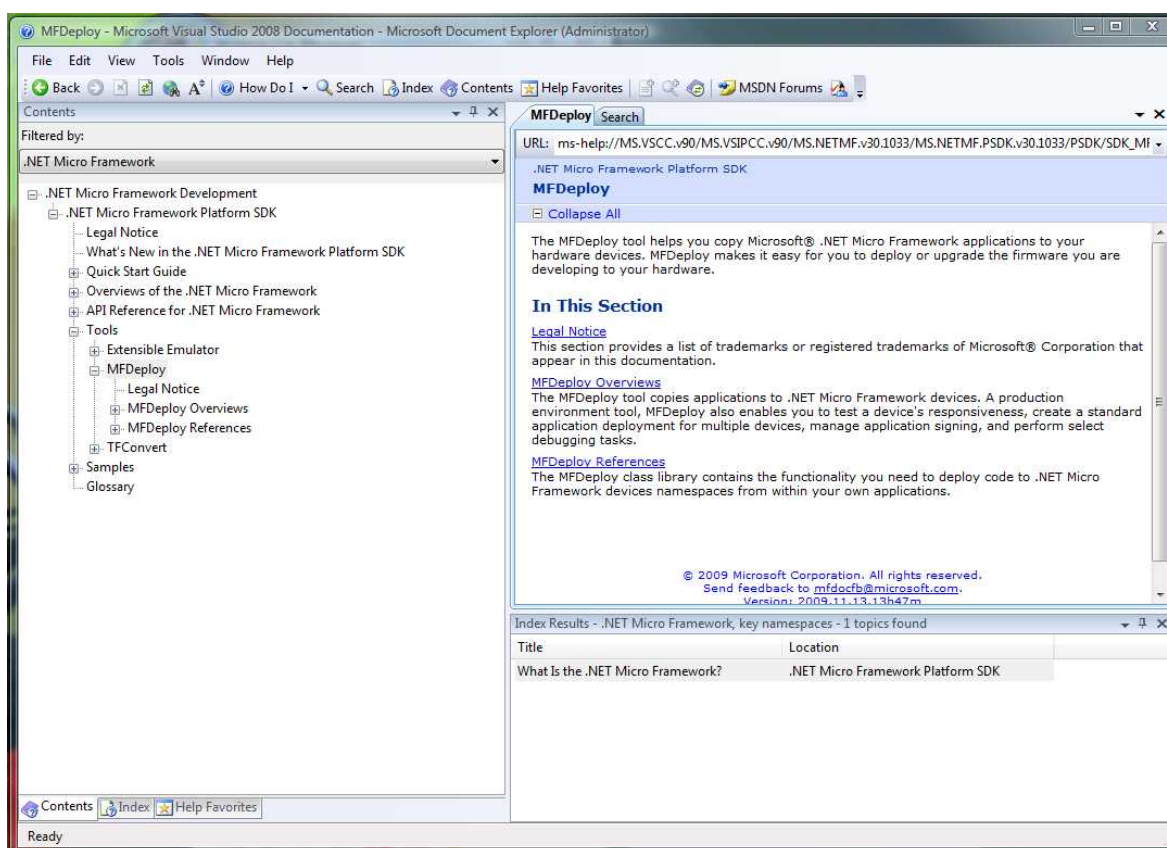
Important Note: The process of reflow can damage EMX if the temperature is too high or exposure is too long. The soldering profile used on the lead-free development system is available for reference.

Appendix A: MFDeploy Tool

MFDeploy is a free tool from Microsoft that helps you deploy Microsoft .NET Micro Framework applications to your hardware devices. MFDeploy makes it easy for you to upgrade the firmware you are developing to your hardware. MFDeploy is available in NETMF SDK. %Microsoft .NET Micro Framework\vx.0 folder%\Tools\MFDeploy.exe

| Name |
|-------------------------|
| MFDeploy.chm |
| MFDeploy.exe |
| MFDeploy.exe.config |
| MFDeployEngine.dll |
| Microsoft.SPOT.Debug... |

Detailed documentation about MFDeploy is available under .NET Micro Framework Help.



One of the great features of MFDeploy is authenticating loaded files. MFDeploy uses public/private keys to verify files. This is a good feature for companies who want to make sure they are the only ones who can load applications on the system. MFDeploy documentation explains this feature in details.

Legal Notice

Licensing

EMX Module is fully licensed for commercial use. The Module price covers the commercial use of EMX Module with .NET Micro Framework.

Disclaimer

IN NO EVENT SHALL GHI ELECTRONICS, LLC. OR ITS CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS PRODUCT, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE. SPECIFICATIONS ARE SUBJECT TO CHANGE WITHOUT ANY NOTICE. GHI ELECTRONICS, LLC LINE OF PRODUCTS ARE NOT DESIGNED FOR LIFE SUPPORT APPLICATIONS.

EMX is a Trademark of GHI Electronics, LLC

.NET Micro Framework, Visual Studio, MFDeploy, Windows Vista, Windows SideShow are registered or unregistered trademarks of Microsoft Corporation.

Other Trademarks and Registered Trademarks are Owned by their Respective Companies.