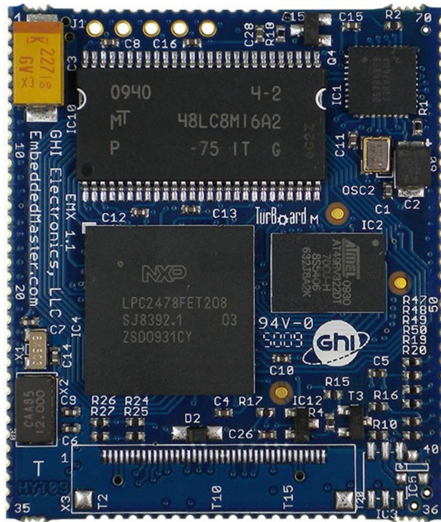
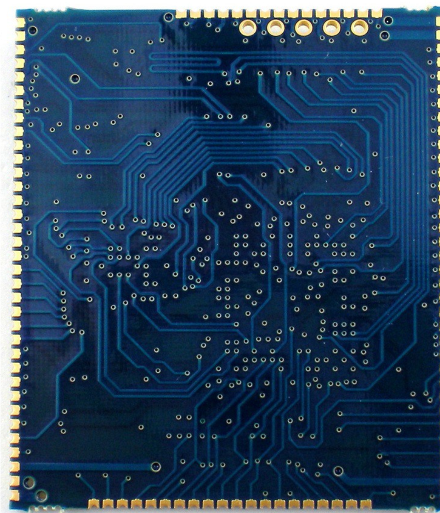
EMX SoM User Manual

Rev. 0.02 | December 16, 2014 | User Manual





EMX Module Top **EMX Module Bottom**

Document Information

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

GHI Electronics

Revision History		
Rev No.	**Date**	**Modification**
Rev. 0.02	12/16/14	Misc typos, etc.
Rev. 0.01	8/27/14	Preliminary version

Table of Contents

1.Introduction....4
1.1.The .NET Micro Framework....4
1.2.GHI Electronics and NETMF....5
1.3.EMX Module Key Features....6
1.4.Example Applications....6
2.The Hardware....7
2.1.LPC2478 Microcontroller....7
2.2.SDRAM....7
2.3.FLASH....7
3.Pin-Out Description....8
3.1.Pin-out Table....8
4.EMX on boot up....13
4.1.Boot Mode Pins....13
4.2.GHI Boot Loader vs. TinyBooter vs. EMX Firmware (NETMF/TinyCLR)....15
4.3.The Loader and Firmware Debug Access Interface....15
5.The GHI Boot Loader....16
5.1.The Commands....16
5.2.Updating TinyBooter....16
Updating TinyBooter using FEZ Config....17
Updating EMX Manually....18
Loading using XMODEM....18
5.3.TinyBooter....21
5.4.TinyCLR (firmware) Update Using FEZ Config....22
5.5.Firmware Update Using MFDeploy....23
6.NETMF TinyCLR (firmware)....26
6.1.Assemblies Version Matching....26
6.2.Deploying to the Emulator....27
6.3.Deploying to the EMX Module....28
6.4.Targeting Different Versions of the Framework....30
7.The Libraries....31
7.1.Finding NETMF Library Documentation....32
7.2.Loading Assemblies....33
7.3.Important Information for the Following Examples....34
7.4.Digital Inputs/Outputs (GPIO)....34
Outputs....35
Inputs....37
Interrupt Pins....38
7.5. Analog Inputs/Outputs....39
7.6.PWM....39
7.7.Signal Generator....40
7.8.Signal Capture....41
7.9.Serial Port (UART)....42
7.10.SPI....43
7.11.I2C....44
7.12.CAN....45
7.13.One-wire....46
7.14.Graphics....47
Fonts....48
Glide....48
Touch Screen....49
7.15.USB Host....50
7.16.Accessing Files and Folders....51
SD/MMC Memory....53
USB Mass Storage....53
7.17.Secure Networking (TCP/IP)....53
The Extensions....53
MAC address setting....53
IP address (DHCP or static):....54
Ethernet....55
Wireless LAN WiFi....56
7.18.PPP....57
7.19.USB Client (Device) 57
7.20.Extended Weak References (EWR)....59
7.21.Real Time Clock....59
7.22.Watchdog....61
7.23.Power Control....61
7.24.In-Field Update....63
7.25.SQLite Database....63
8.Advanced use of the Microprocessor 65
8.1.Register....65
8.2.AddressSpace....65
8.3.Battery RAM....65
8.4.Runtime Loadable Procedure....66
9.design Consideration....67
Required Pins....67
Interrupt Pins....67
....67
10.Soldering EMX....68
Legal Notice....69
Licensing....69
Disclaimer....69

1. Introduction

The EMX Module is a powerful, yet low-cost, surface-mount System on Module (SoM) running the .NET Micro Framework software, which enables the SoM to be programmed from Microsoft's Visual Studio, by simply using a USB cable. Programming in a modern managed language, such as C# and Visual Basic, allows developers to accomplish much more work in less time by taking advantage of the extensive built-in libraries for networking, file systems, graphical interfaces and many peripherals.

With just power and some connectors, developers can utilize the EMX Module to bring the latest technologies to any products. There are no additional licensing or fees and all the development tools and SDKs are provided freely.

1.1. The .NET Micro Framework

Inspired by its full .NET Framework, Microsoft developed a lightweight version called .NET *Micro* Framework (NETMF).

NETMF focuses on the specific requirements of resource-constrained embedded systems. Development, debugging and deployment is conveniently performed using Microsoft's powerful Visual Studio tools, all through standard USB cable.

Programming is done in C# or Visual Basic. This includes libraries to cover sockets for networking, modern memory management with garbage collector and multitasking services. In addition to supporting standard .NET features, NETMF has embedded extensions supporting:

- General Purpose IO (GPIO with interrupt handling
- Analog input/output
- Standard buses such I2C, SPI, USB, Serial (UART)
- PWM
- Networking
- File System
- Display graphics, supporting images, fonts and controls.

1.2. GHI Electronics and NETMF

For years, GHI Electronics has been the lead Microsoft partner on .NET Micro Framework (NETMF). The core NETMF was also extended with new exclusive libraries for an additional functionality, such as database, USB Host and WiFi.

One of the important extensions by GHI Electronics is Runtime Loadable Procedures (RLP), allowing native code (Assembly/C) to be compiled and loaded right from withing managed code (C#/Visual Basic) to handle time critical and processor intensive tasks. IT can also be used to add new native extensions to the system.

As for networking, WiFi and PPP libraries are added by GHI Electronics to the NETMF core. Combined with Ethernet and the other managed services, it is a complete toolbox for the internet of things.

All the mentioned features are loaded and tested on the EMX Module. GHI Electronics continuously maintains, upgrades and solves any of the issues on the EMX Module firmware, to provide regular and free releases. Users can simply load the new software on the EMX Module using USB or Serial, and even use the in-field-update feature. This feature allows the upgrade to be done through any of the available interface, including file system and networking.

1.3. EMX Module Key Features

- .NET Micro Framework
- 72 Mhz 32-bit ARM7
- 16 MB RAM
- 4.5 MB FLASH
- Embedded LCD controller
- 76 GPIO Pins
- 43 Interrupt Inputs
- 2 SPI
- I2C
- 4 UART
- 2 CAN Channels
- 7 10-Bit Analog Input
- 10-Bit Analog Output
- 4Bit SD/MMC Memory card interface
- 6 PWM
- Power 200 / 160 / 40 mA
- -40ºC to +85ºC Operational
- RoHS Lead Free
- 45.75 x 39.4 x 4.4 mm
- TCP/IP Stack (.NET sockets)
- SSL secure networking
- WiFi
- PPP
- USB Host & Client
- Graphics (image, font and controls)
- SQLite database
- File System (SD and USB Sticks)
- Native extensions RLP

1.4. Example Applications

- Vending machines, POS Terminals
- Measurement tools and testers
- Networked sensors
- Robotics
- Central alarm system
- Smart appliances
- Industrial automation devices
- Designs with intensive processing or time-critical routines (using RLP)
- GPS navigation
- Medical instrument (with a color touch screen display).

2. The Hardware

The EMX Module core components includes the processor, 4.5MB flash, and 16MB RAM.

The small, 38.1 x 26.7 x 3.55 mm (only 1 x 1.5 inches), module contains everything needed to run a complex embedded-system in a cost-effective and flexible solution. All needed is a 3.3V power source and some connections to take advantage of the EMX Module's long list of available features.

2.1. LPC2478 Microcontroller

The LPC2478 microcontroller in the EMX Module is an 72Mhz, 32Bit, ARM7. 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 while the processor is off. The 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.

The 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.

The NETMF core libraries, combined with the GHI Electronics extensions, provide a long list of methods to access the available peripherals.

2.2. SDRAM

16MB of SDRAM comes standard with the EMX Module.

2.3. FLASH

4.5MB of external flash is available on the EMX Module. The fast zero-wait-state internal 0.5MB is used to execute the core services of the system to achieve the highest possible performance. The remaining 4MB of external memory is used to hold more extensions and to store the user's end application. 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 being 128KB and one of them is reserved for CLR use.

One of GHI's extensions allows applications to be updated in field, even remotely.

3. Pin-Out Description

Many signals on the EMX Module are multiplexed to offer multiple functions on a single pin. Developers can decide on the pin functionality through the provided libraries. These are some important facts pertaining to the available pins:

- Pins with GPIO feature default to inputs with internal weak pull-up resistors
- GPIO pins are 3.3V levels but 5V tolerant
- Pins with analog feature are not 5V tolerant when the analog function is used
- Only GPIO pins on ports 0 and 2 are interrupt capable
- Advanced details on all pins can be found in the LPC2478 datasheet

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 IO7.

3.1. Pin-out Table

No.	LPC2478 H/W Name	EMX IO	2nd Feature	EMX Module Pin Description
1	3.3V			Connect to 3.3 volt source.
2	GND			Connect to Ground.
3	P0.4	IO0*	CAN2/ Boot control	RD CAN Channel 2 Data Receive pin (In). TinyBooter/Firmware Control (with 7, and 53)
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	P2.5	IO4*	Boot Control	General purpose digital I/O. TinyBooter/Firmware Control (with 3, and 53)
8	P0.24	IO5*	ADC1/ Touch_Y_UP	ADC1 (10Bit Analog to Digital Input) or Touch Screen Y-axis Up analog signal.

No.	LPC2478 H/W Name	EMX IO	2nd Feature	EMX Module Pin Description
1	3.3V			Connect to 3.3 volt source.
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		
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	USBD- USB Host Feature			USB negative data line of the USB hosting feature.
30	USBD+ 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.

<table>
<tr><th>No.</th><th>LPC2478
H/W Name</th><th>EMX IO</th><th>2nd Feature</th><th colspan="2">EMX Module
Pin Description</th></tr>
<tr><td>1</td><td colspan="3">3.3V</td><td colspan="2">Connect to 3.3 volt source.</td></tr>
<tr><td>38</td><td colspan="3">N/C</td><td colspan="2">Not Connected.</td></tr>
<tr><td>39</td><td>P3.26</td><td>IO49</td><td>PWM3</td><td colspan="2">PWM3 (Pulse Width Modulation Output) LPC2478 PWM Timer 1.</td></tr>
<tr><td>40</td><td>P3.17</td><td>IO50</td><td>PWM2</td><td colspan="2">PWM2 (Pulse Width Modulation Output) LPC2478 PWM Timer 0.</td></tr>
<tr><td>41</td><td colspan="3">USBD- device</td><td colspan="2">USB negative data line of the USB debugging interface and for the USB client feature.</td></tr>
<tr><td>42</td><td colspan="3">USBD+ device</td><td colspan="2">USB positive data line of the USB debugging interface and for the USB client feature.</td></tr>
<tr><td>43</td><td colspan="3">Ethernet RD-</td><td>Ethernet receive data minus.</td><td rowspan="4">Recommended Ethernet connector is J0011D01BNL.
Ethernet PHY is not needed since it is embedded in EMX hardware.</td></tr>
<tr><td>44</td><td colspan="3">Ethernet RD+</td><td>Ethernet receive data plus.</td></tr>
<tr><td>45</td><td colspan="3">Ethernet TD-</td><td>Ethernet transmit data minus.</td></tr>
<tr><td>46</td><td colspan="3">Ethernet TD+</td><td>Ethernet transmit data plus.</td></tr>
<tr><td>47</td><td>P0.18</td><td>IO24*</td><td>SPI1</td><td colspan="2">SPI master bus interface MOSI signal (Master Out Slave In) for SPI1.</td></tr>
<tr><td>48</td><td>P0.17</td><td>IO25*</td><td>SPI1</td><td colspan="2">SPI master bus interface MISO signal (Master In Slave Out) for SPI1.</td></tr>
<tr><td>49</td><td>P0.16</td><td>IO26*</td><td>N/A</td><td colspan="2">General purpose digital I/O.</td></tr>
<tr><td>50</td><td>P0.15</td><td>IO27*</td><td>SPI1</td><td colspan="2">SPI master bus interface SCK signal (Clock)for SPI1.</td></tr>
<tr><td>51</td><td>P4.23</td><td>IO28</td><td>COM3</td><td colspan="2">Serial port (UART) RXD receive signal (In) for COM3.</td></tr>
<tr><td>52</td><td>P4.22</td><td>IO29</td><td>COM3</td><td colspan="2">Serial port (UART) TXD transmit signal (Out) for COM3.</td></tr>
<tr><td>53</td><td>P2.11</td><td>IO30*</td><td>Boot Control</td><td colspan="2">General purpose digital I/O. TinyBooter/Firmware Control (with 7, and 3)</td></tr>
<tr><td>54</td><td>P3.30</td><td>IO31</td><td>COM2</td><td colspan="2">Serial port (UART) RTS hardware handshaking signal for COM2.</td></tr>
<tr><td>55</td><td>P2.1</td><td>IO32*</td><td>COM2</td><td colspan="2">Serial port (UART) RXD receive signal (IN) for COM2.</td></tr>
<tr><td>56</td><td>P0.6</td><td>IO33*</td><td>N/A</td><td colspan="2">General purpose digital I/O.</td></tr>
<tr><td>57</td><td>P3.18</td><td>IO34</td><td>COM2</td><td colspan="2">Serial port (UART) CTS hardware handshaking signal for COM2.</td></tr>
<tr><td>58</td><td>P0.7</td><td>IO35*</td><td>SPI2</td><td colspan="2">SPI master bus interface SCK signal (Clock)for SPI2.</td></tr>
<tr><td>59</td><td>P0.9</td><td>IO36*</td><td>SPI2</td><td colspan="2">SPI master bus interface MOSI signal (Master Out Slave In) for SPI2.</td></tr>
<tr><td>60</td><td>P2.0</td><td>IO37*</td><td>COM2</td><td colspan="2">Serial port (UART) TXD transmit signal (Out) for COM2.</td></tr>
<tr><td>61</td><td>P0.8</td><td>IO38*</td><td>SPI2</td><td colspan="2">SPI master bus interface MISO signal (Master In Slave Out) for SPI2.</td></tr>
<tr><td>62</td><td>P1.12</td><td>IO39</td><td>SD_DAT3</td><td colspan="2">SD card 4Bit data bus, data line no. 3.</td></tr>
<tr><td>63</td><td>P1.11</td><td>IO40</td><td>SD_DAT2</td><td colspan="2">SD card 4Bit data bus, data line no. 2.</td></tr>
<tr><td>64</td><td>P1.7</td><td>IO41</td><td>SD_DAT1</td><td colspan="2">SD card 4Bit data bus, data line no. 1.</td></tr>
<tr><td>65</td><td>P1.2</td><td>IO42</td><td>SD_CLK</td><td colspan="2">SD card 4Bit data bus, clock line.</td></tr>
<tr><td>66</td><td>P1.6</td><td>IO43</td><td>SD_DAT0</td><td colspan="2">SD card 4Bit data bus, data line no. 0.</td></tr>
<tr><td>67</td><td>P1.3</td><td>IO44</td><td>SD_CMD</td><td colspan="2">SD card 4Bit data bus, command line.</td></tr>
<tr><td>68</td><td colspan="3">SD_PWR</td><td colspan="2">SD memory power (connect directly to SD socket power pin).</td></tr>
<tr><td>69</td><td colspan="3">GND</td><td colspan="2">Connect to Ground.</td></tr>
</table>

No.	LPC2478 H/W Name	EMX IO	2nd Feature	EMX Module Pin Description
1	3.3V			Connect to 3.3 volt source.
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.
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.

No.	LPC2478 H/W Name	EMX IO	2nd Feature	EMX Module Pin Description
1	3.3V			Connect to 3.3 volt source.
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.

4. EMX On Boot Up

Software run on the EMX Module is divided into different components:

- The *GHI Boot loader* – initializes memories and executes TinyBooter. It is also used to update TinyBooter.
- *TinyBooter* – does set-up for, and then, loads the firmware (TinyCLR, NETMF core, and GHI extensions). It is also used to update the NETMF firmware and its system configurations.
- *TinyCLR and NETMF* (the firmware) -- interprets and executes the *managed application*. It is used for other functions such as loading and/or debugging the *managed application*.
- The *managed application* (C# - Visual Basic) – developed by customers.
- Optional: Native *RLP routines* (C and/or assembly, described in the Runtime Loadable Procedure section); developed by customers.

If the boot mode pins, 7 and 3, are left floating (internal pull up), or pulled high externally, the default boot-up sequence executes as the following (see also, the flow-chart further below):

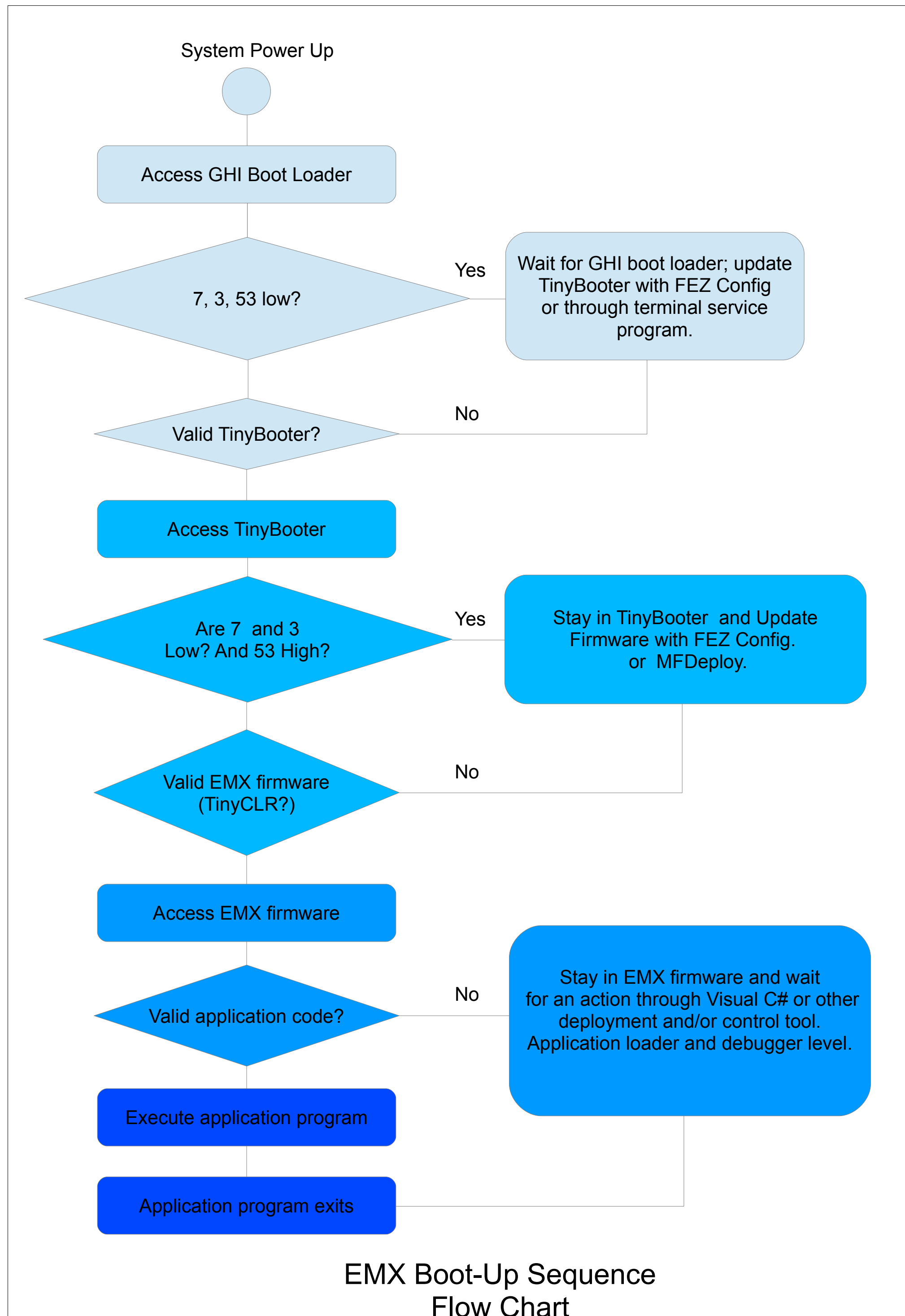
- The GHI boot loader initializes Flash and RAM memory and looks for a valid TinyBooter and passes execution to it.
- TinyBooter prepares the EMX hardware resources required by the NETMF Core environment and passes execution to NETMF TinyCLR.
- If a valid end-user embedded application exists, it gets executed.

4.1. Boot Mode Pins

Default start-up execution can be changed using two control pins, they are active low and have internal weak pull up resistors:

Pin 7	Pin 3	Pin 53	Effect
High or unconnected	High or unconnected	High or unconnected	Default, execute all levels
Low	Low	Low	Execute Boot Loader but do not execute TinyBooter
Low	Low	High or unconnected	Execute the Boot Loader and TinyBooter but do not execute NETMF TinyCLR

The following flow chart clearly explains the boot up sequence:



EMX Boot-Up Sequence
Flow Chart

4.2. GHI Boot Loader vs. TinyBooter vs. EMX Firmware (NETMF/TinyCLR)

The table below gives greater detail of the characteristics of each level of execution.

GHI Boot Loader	TinyBooter	NETMF TinyCLR (firmware)
Used to update EMX TinyBooter or for low level EMX flash maintenance.	Used to update the EMX firmware (NETMF TinyCLR) 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.	Sometimes used.	Always used.
Pre-burned into the EMX Module's flash memory. Can't be updated.	Replaceable using FEZ Config or the GHI Boot Loader	Replaceable; for example, under control of TinyBooter
Controlled through simple text commands and X-modem. Any terminal, such as teraterm or hyper terminal, can be used.	Controlled through MFDeploy or FEZ Config tools.	Runs the user application and accepts commands from Visual Studio for debugging purposes.

When applying updates, the lowest level software should be updated first. For example, if both the firmware (TinyCLR) and TinyBooter need updates, TinyBooter should be updated first.

4.3. The Loader and Firmware Debug Access Interface

The communication between a PC and the EMX Module can be done using a USB port or a serial port (COM1). This interface can be used for updating, deploying or debugging the software components.

The LMODE Pin is used to select USB vs serial (UART). When LMODE is high or left floating (internal weak pull up resistor) the system will run in serial mode using COM1; when pulled, low (10 K resister), USB is used.

When USB is selected, the drivers needed on the PC are included in the GHI SDK. Two different drivers are available. The first one is a virtual COM driver used by the GHI loader. The second one is used by TinyBooter and NETMF TinyCLR.

5. The GHI Boot Loader

The EMX Boot Loader software is pre-loaded and locked on the EMX Module. It is used to update TinyBooter and can be used to do a complete erase all flash memory. The GHI boot loader is rarely needed but it is recommended to keep access available in all project designs.

The GHI boot loader accepts simple commands sent with the help of a terminal service software, such as TeraTerm or Hyper Terminal. A command character is sent and the boot loader performs an action; results are returned in a human friendly format followed by a "BL" indicating that the boot loader is ready for the next command. All commands and responses use ASCII encoded characters.

The EMX on boot up section provides the required information on how to choose the access interface and how to access the GHI boot loader.

5.1. The Commands

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

Notes:

- Commands are not followed by pressing the “ENTER” key. The single command letter is sent to the EMX Module; which immediately begins executing the command.
- The Boot loader commands are case sensitive.

5.2. Updating TinyBooter

GHI Electronics' SDK includes FEZ Config, a Windows program that can be used to update all software components of the EMX Module; as well as, system settings (configuration parameters). Although FEZ Config **is recommended** as the tool to use, the same process can be performed manually using an ASCII terminal program. The next section shows how

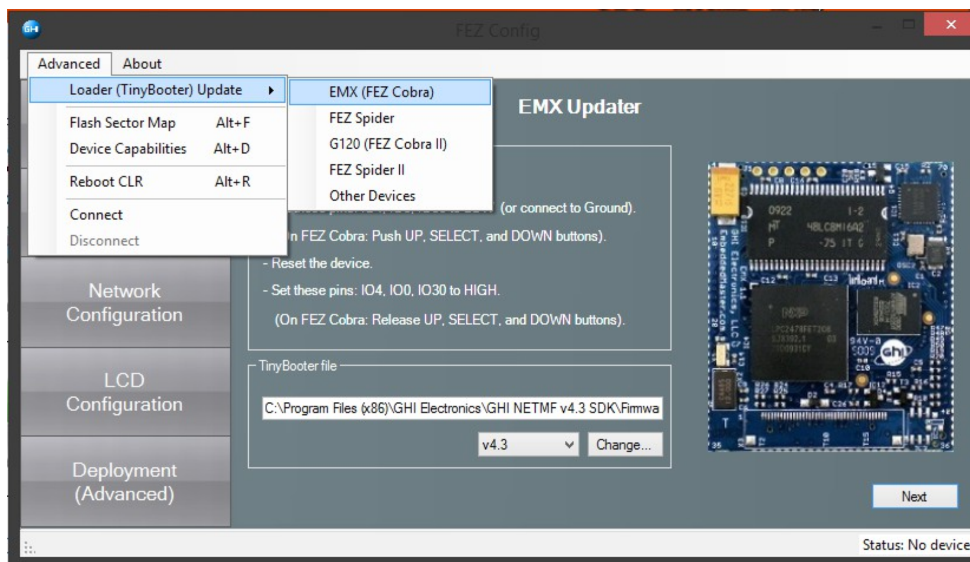
use GHI's FEZ Config. The manual procedure is described in the section titled “Updating EMX Manually.”

Updating TinyBooter using FEZ Config.

The following procedures' images are from version 2.0.2.0 of FEZ Config running on Windows 8.01. The images may vary with other versions.

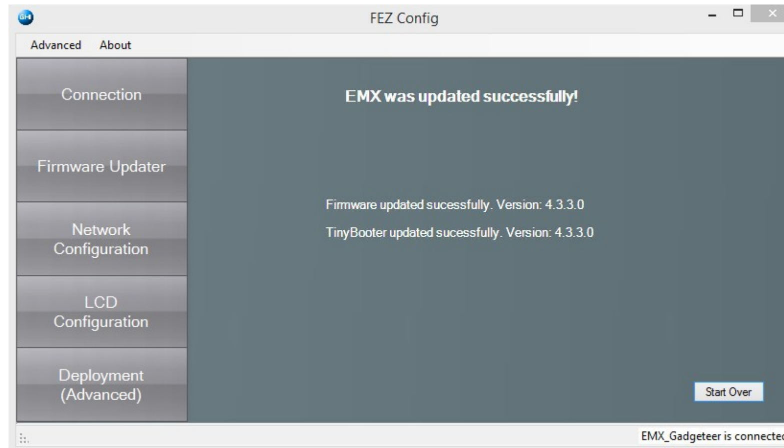
The first step is to interface the EMX with the PC. This is best done using the USB pins, and the appropriate cable.

Launch FEZ Config. Under “Device” make sure “USB” is selected and EMX is the device.



Now follow the menu: `Advanced>Loader (TinyBooter) Update>EMX Module`

Make sure the version shown is correct. Then leave other fields with their defaults and click on “`Next`” Answer any dialog questions. The program will flash the module and display it's progress.



NOTE: When finished updating tinyBooter, FEZ Config will automatically update TinyCLR as needed.

Updating EMX Manually

FEZ Config is the recommended tool for updating TinyBooter on the module. This section describes a manual process. The only advantage of performing flashing this way is that it allows updates from any device with a serial connection; independent of operating system.

Depending on the state of the LMODE Pin (see The Loader and Firmware Debug Access Interface section), the GHI Boot loader will use the USB or the serial interface. When using USB, it requires a virtual serial interface driver on the USB Host. For Windows, this means the GHI boot loader on the will appear as a serial device. It should be noted that the serial pins on are TTL level, a RS232 level converter is needed to wire the serial pins to a PC's serial port.

Serial communications parameters are: 152000 baud, 8-bit data, with one stop bit and no parity.

Loading using XMODEM

The GHI boot loader uses the XMODEM protocol to receive the TinyBooter file.

In this example, a serial program running on a PC is used to show the transfer.

Once the appropriate serial port on the PC is opened, a user can start sending commands, such as entering V to see the boot loader version number.

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,
- 16Bit CRC error checking.

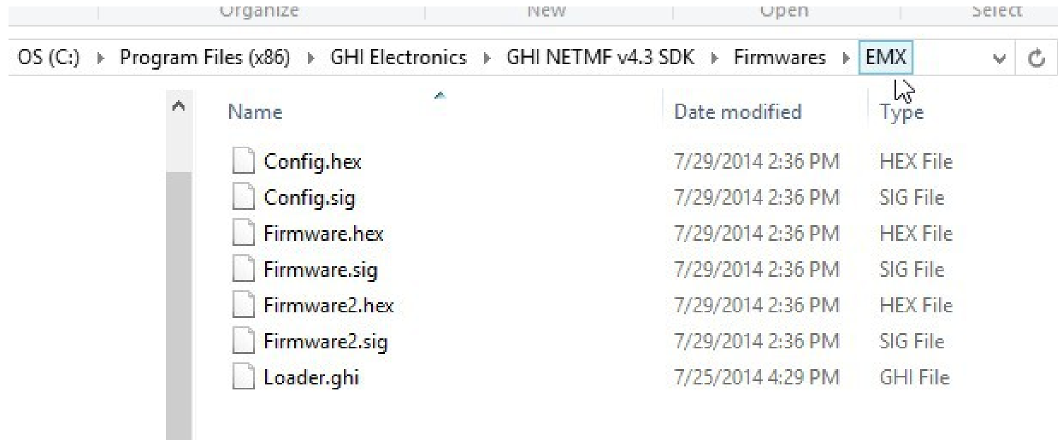
Instructions for updating TinyBooter (commands to the GHI Boot loader are case sensitive):

1. Boot the EMX using the Pin configuration described in the Boot Mode Pins section to start the Boot loader's command/control interface. Access the boot loader using TeraTerm (or other terminal program). With Tera Term use the ***Setup>serial port...*** menu to adjust UART parameters.
2. To confirm the GHI boot loader is active and responding enter V to see the version number. The *data sent to the GHI boot loade*r is not echoed to the sender. This means that no V will appear on the terminal; the boot loader will transmit the version number, followed by "BL" (which is an indication it is ready to receive another command.
3. Erase the flash memory using E command then press Y to confirm (this will take several seconds).
4. Initiate transfer by typing X; this tells the Boot loader to wait for the transmission of data from the terminal program. 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 the "C" character appears on the terminal window, select XMODEM transfer and point the software to the firmware file "Loader.ghi" as shown below:

```
BL
BL
BL
BL
1.01
BL
Start File Transfer
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
```

If using TeraTerm: In the menu, select **File > Transfer > XMODEM > Send...**

Next, select the file containing TinyBooter, "Loader.ghi," from the "C:\Program Files(x86)\GHI Electronics" folder; make sure to use the "Loader.ghi" file specified under an "EMX" subfolder, and make sure to select the 1K option in TeraTerm.



Updating the firmware may take a few seconds. Once loading has finished, Exit the terminal program and reset the EMX Module.

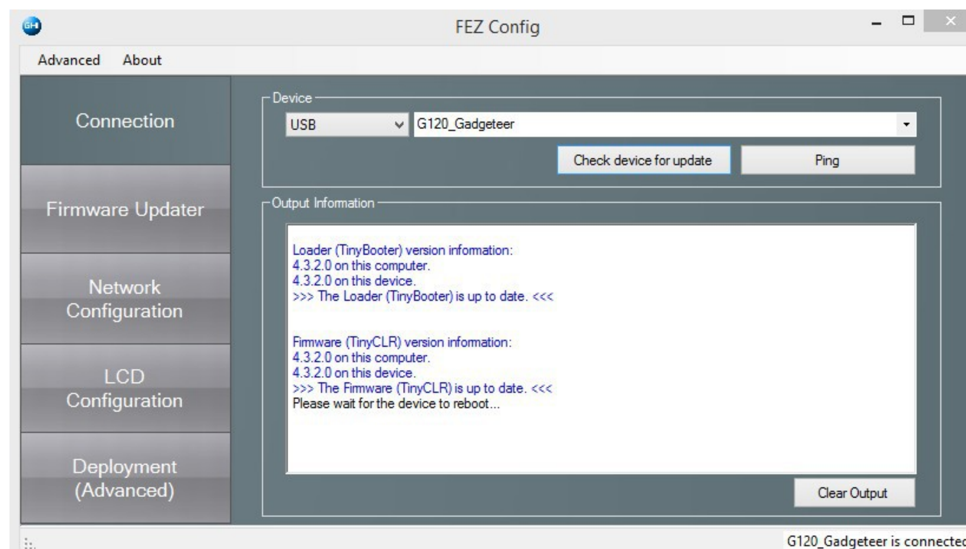
5.3. TinyBooter

TinyBooter has two functions;

1. by default it executes the NETMF TinyCLR firmware,
2. the other function of TinyBooter is to load new firmware (TinyCLR). Whenever GHI Electronics releases new firmware TinyBooter is used to load it.

The EMX on boot up chapter provides the pin configuration required to choose an access interface and how to invoke TinyBooter.

When preparing to install new firmware, it is important to make sure that the version of TinyBooter supports the new TinyCLR. This is done using the "Check for device update" in FEZ Config. Version numbers of both TinyBooter and TinyCLR are always listed in the current GHI Electronics' SDK Release Notes. Release Notes are installed with the SDK; they may also be viewed online by following the GHI SDK Library links to the SDK.

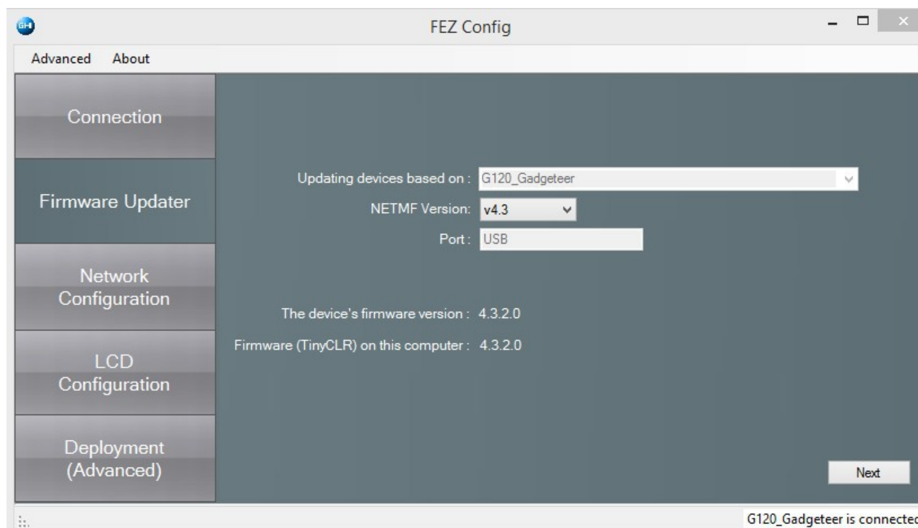


If TinyBooter needs to be updated , do it before updating TinyCLR. See Updating TinyBooter using FEZ Config. (***NOTE:*** if tinyBooter was just used to update with FEZ Config then TinyCLR was updated, and this chapter can be skipped.

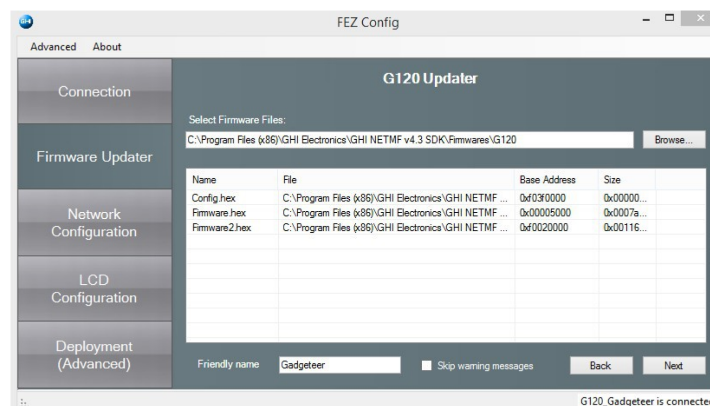
NOTE: TinyCLR can also be updated using In-Field Update.

5.4. TinyCLR (firmware) Update Using FEZ Config

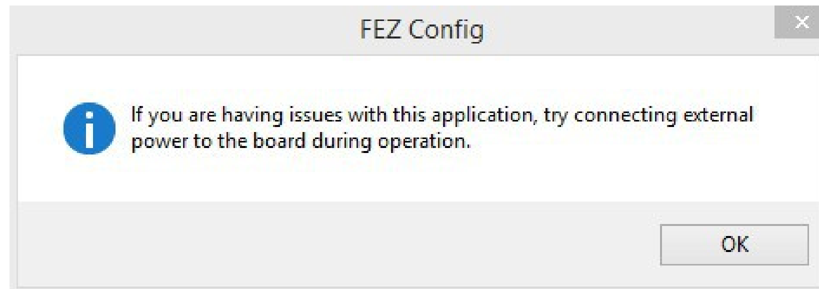
1. Connect the EMX Module to the PC.
2. Launch FEZ Config and click on “Check device for update” button. This will show the version numbers on the PC and what is loaded on the EMX Module. (see picture above)
3. To proceed with updating TinyCLR, click on the “Firmware Updater” tab on the left and follow the instructions.

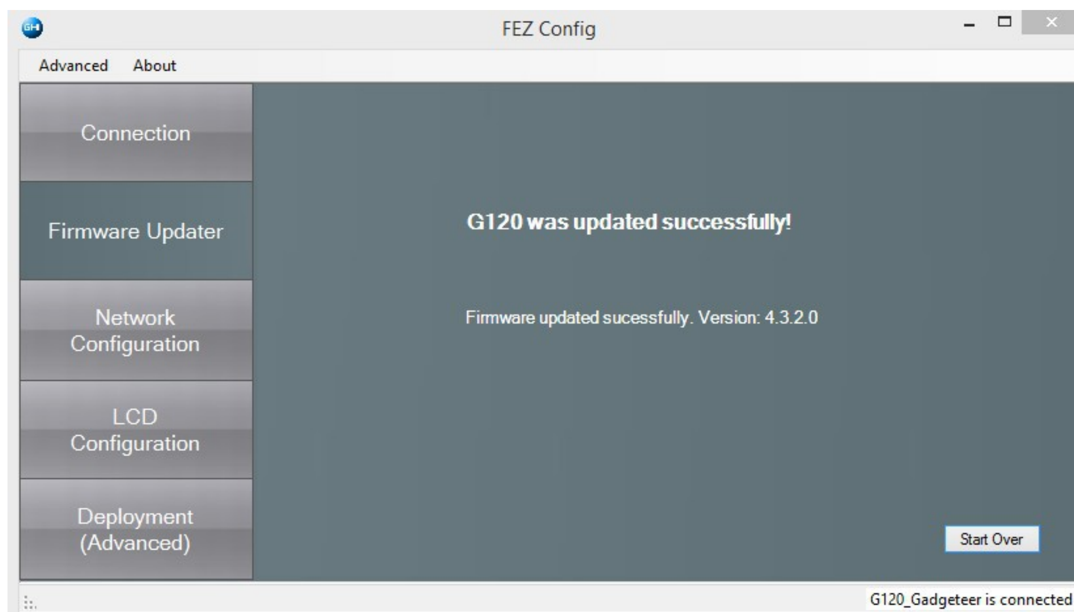


4. After FEZ Config selects the firmware and the default configuration files, click “Next”



5. If this dialog appears, Click “OK” to proceed





6. As the update occurs, the steps and progress are shown. When it is finished, the module is ready to be flashed with NETMF applications.

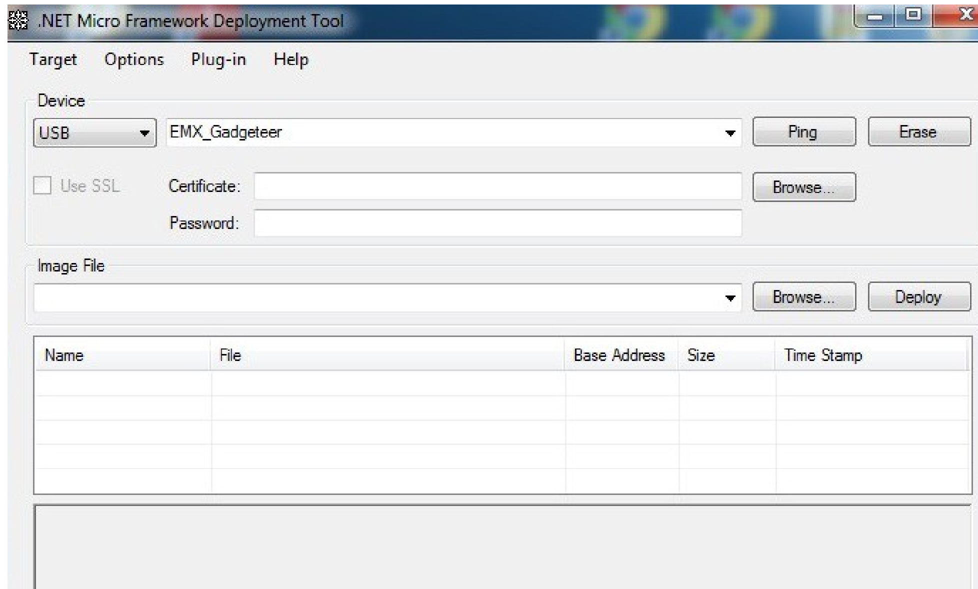
5.5. Firmware Update Using MFDeploy

It is *strongly* recommended to use GHI Electronics' FEZ Config tool for updating the EMX Module. This section describes the use of MFDeploy to update the EMX firmware.

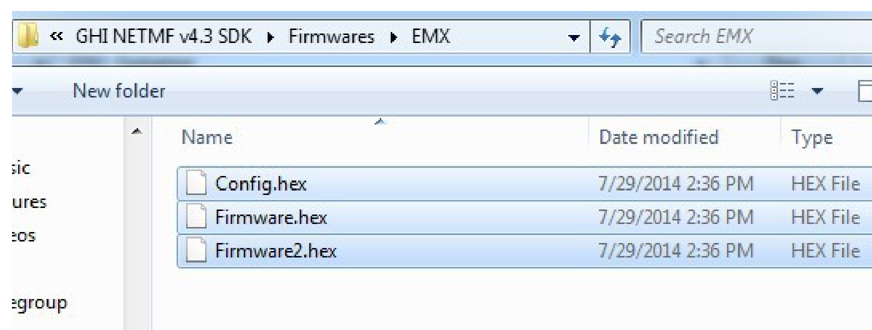
1. Set the pins as described in Boot Mode Pins to enter TinyBooter. This is optional as

EMX will automatically switch to TinyBooter as necessary.

2. Run MFDeploy and select “USB” or “Serial” from the Device list, the available devices or COM ports appear in the drop down list.



3. Add the EMX firmware files to MFDeploy's “**Image File**” by clicking on “Browse” (next to the “Image File” text box). The firmware files can be found under “GHI Electronics” in the “Programs (x86)” folder; inside the directory that corresponds to the SDK being



used, go to “firmwares\EMX”. Select **all** of the firmware hex files at once.

4. Start deploying the firmware by pressing “Deploy”.

5. Loading the files takes a little while. Upon completion, the firmware (TinyCLR) will execute. Double check the version number to make sure the correct firmware is loaded.
6. Loading new firmware will not erase the deployed managed application. To erase the managed application click Erase.

6. NETMF TinyCLR (firmware)

The Firmware is the main piece of embedded software running on the EMX Module. It is what interprets and runs the user's managed application and it is what Microsoft's Visual Studio use to deploy, hook-into and debug the managed application. As explained in Boot Mode Pins section, hardware interfaces between TinyCLR and the host development system is either USB or Serial. In this chapter the examples use the USB interface.

If necessary, the module's firmware can be updated as described in the TinyBooter chapter.

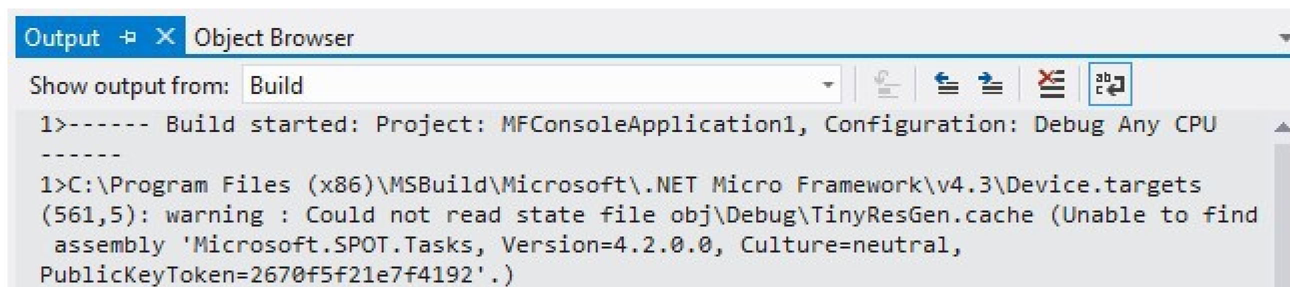
6.1. Assemblies Version Matching

The firmware includes extensions added by GHI Electronics. These extensions are often improved and further extended. If the managed application (C# or Visual Basic) uses any of the GHI specific extensions, care must be taken when a new SDK is installed.

This is due to the fact that the existing Visual Studio projects will include a local copy of the assemblies supplied by the old SDK; during compilation of the application, the extensions may not match what is found.

Additionally, the assemblies themselves are compiled for use with specific SDK versions.

For example where an application was previously compiled with 4.2, then the 4.3 SDK is installed; even if a successful compilation occurs (no extension conflicts were found), then the deployment process will begin to load 4.2 assemblies; this will cause loading errors when compiling for 4.3. This will not harm the EMX Module. Visual Studio's Output panel will contain something like:

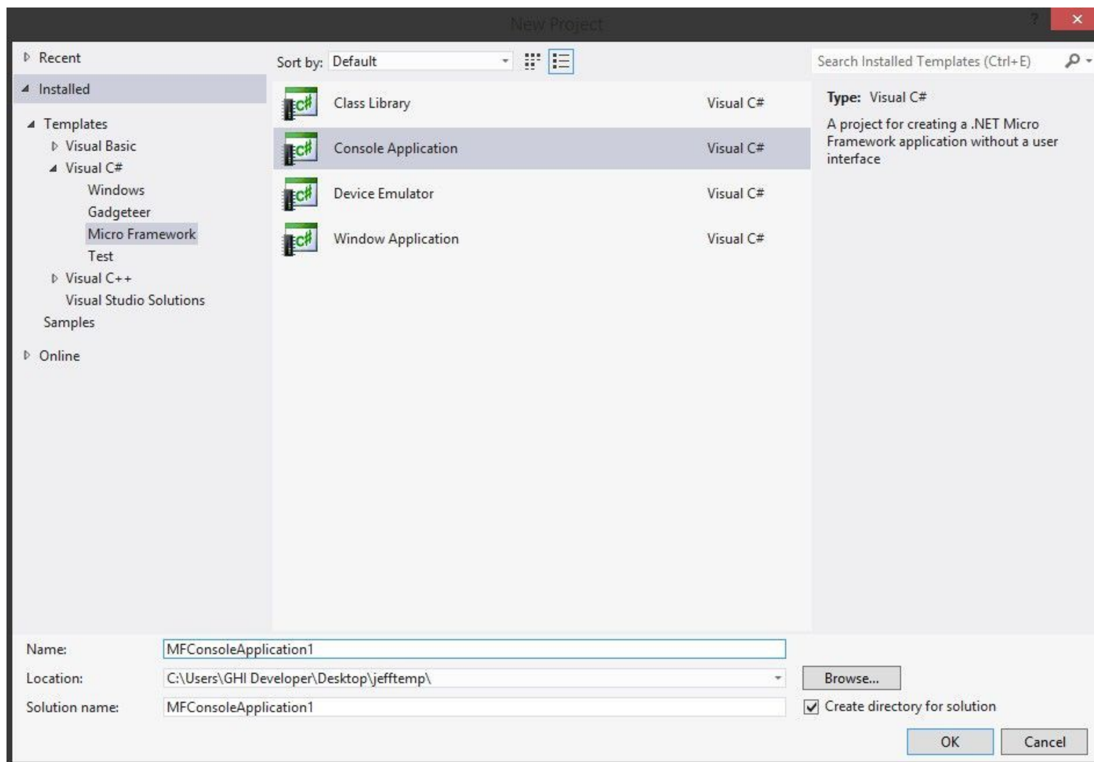


there is further discussions of assemblies in the Loading Assemblies section of chapter 7.

6.2. Deploying to the Emulator

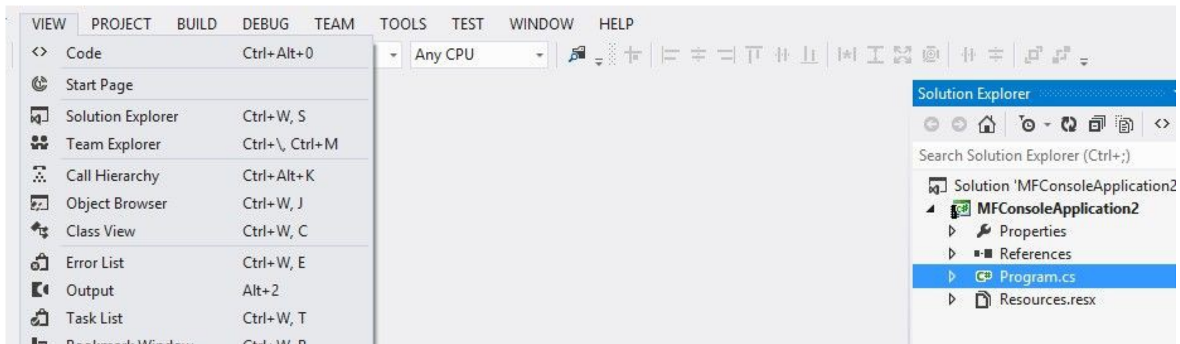
Once the latest SDK is installed and the EMX Module is loaded with the latest TinyBooter and NETMF TinyCLR then using Visual Studio to load/debug C# and Visual Basic applications is very easy. If not installed yet, the latest SDK should be downloaded and installed on the development machine. See www.ghielectronics.com/support/netmf to install the latest SDK.

When done, Visual Studio can be started to create a new ***Micro Framework*** project ***Console Application***.



C# is selected in this example but Visual Basic would be very similar. Run the code as is by pressing F5 or clicking the start button. This should open up the emulator and run the program. This program prints “Hello World” on the output window, not on the screen. If the output window is not visible, it can be opened from the “VIEW” top menu. When running, the emulator shows a simulated device. The output window of Visual Studio will be full of messages... from loading assemblies (libraries) on power up to loading the application, to the actual “Hello World!.”

Open the program Program.cs (it was created automatically by Visual Studio):



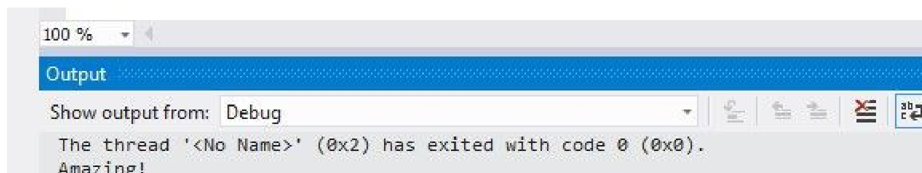
Change the program to the following.

```
using System;
using Microsoft.SPOT;

public class Program
{
    public static void Main()
    {
        Debug.Print("* Amazing! *");
    }

}
```

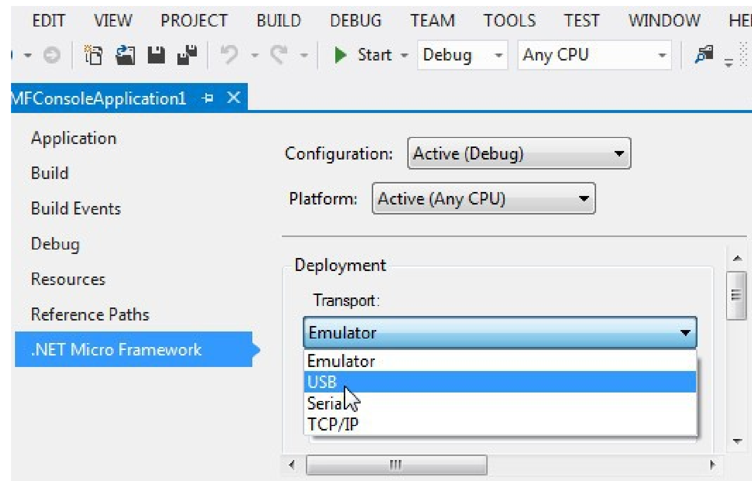
Press F5. The output window will now show something similar to:



6.3. Deploying to the EMX Module

Loading to the actual hardware device is exactly the same as loading to the emulator. The only difference is in targeting the EMX instead of the emulator. This is done by going to the project properties:

In this example, the EMX Module is connected to the PC using the USB interface (see The Loader and Firmware Debug Access Interface section). To deploy the application to the



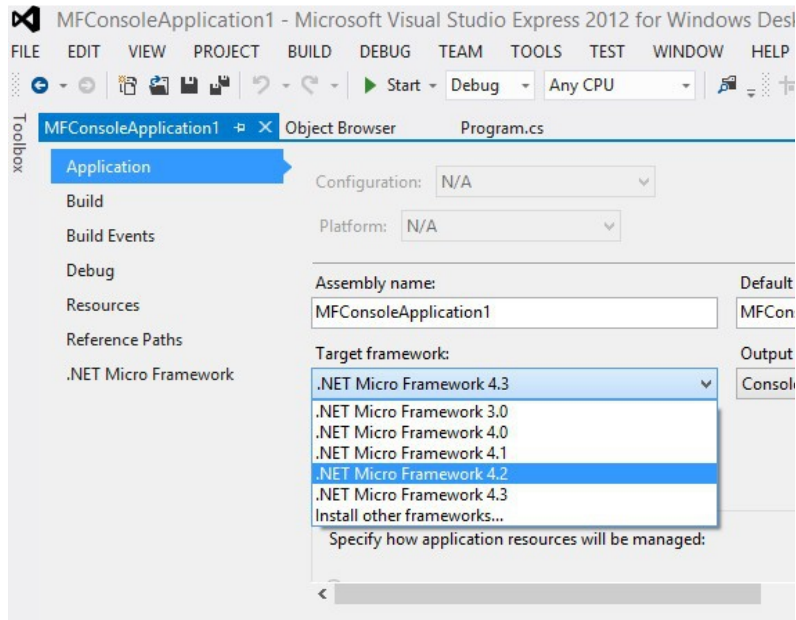
module select USB for “Transport.”

Running the application (F5 key) will load the program on the EMX Module and run it. The output window of Visual Studio will still show very similar messages but they are now coming from the EMX Module directly.

If necessary, the deployed program can use the full power of the Visual Studio debugger; including, stepping through lines, inspecting variables, setting breakpoints, etc.

6.4. Targeting Different Versions of the Framework

There are times when it may be useful to compile and deploy applications for an older version of the SDK. For example, if there is a module with older firmware and there is an older application that needs to be deployed. GHI Electronics and Microsoft makes this easy by shipping the previous version of the framework as part of the current package. Under Project Properties, use the Application panel to target the desired version:



7. The Libraries

Similar to the full desktop .NET, NETMF includes many services to help in modern application development. One example would be threading. This is typically very difficult to deal with on embedded systems, but thanks to NETMF, this is very easy and works as well as it does on a desktop application.

```
using System;
using System.Threading;
using Microsoft.SPOT;

public class Program
{
    // We will print a counter every 1 second
    static int Count=0;
    static void CounterThread()
    {
        while (true)// Infinite loop
        {
            Thread.Sleep(1000);// Wait for 1 second
            Count++;// Increment the count
            Debug.Print("Count = " + Count);// Print the count
        }
    }
    // ************************************************
    static void Main()
    {
        //Create a second thread, main is automatically a thread
        Thread EasyThread = new Thread(CounterThread);
        EasyThread.Start();// Run the Counter Thread

        // We can now do anything we like
        // We will print Hi once every 2 seconds
        while (true)// Infinite loop
        {
            Debug.Print("Hi");
            Thread.Sleep(2000);
        }
    }
}
```

The output will look similar to this:

```
Hi
Count = 1
Hi
Count = 2
Count = 3
Hi
Count = 4
Count = 5
Hi
Count = 6
```

7.1. Finding NETMF Library Documentation

This chapter is not meant to be a full tutorial on the use of NETMF. It contains a survey of the main facilities to aid newcomers to NETMF. The main support page for NETMF includes links to the library reference documentation (NETMF APIs).

Because NETMF is a subset of the full .NET platform, services such as file input/output and Networking are very close, and sometimes identical, to the full .NET Framework. The internet is a great source of .NET examples code that often can be used in a NETMF program with no changes!

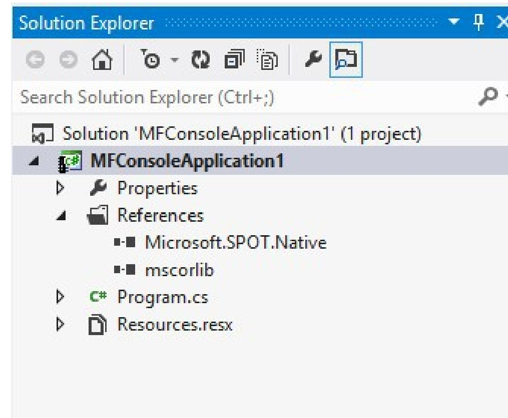
7.2. Loading Assemblies

In an earlier example, the threading libraries were used. This was done by identifying the `namespace` via the statement:

```
using System.Threading;
```

The compiled code for classes in the Threading library are part of the mscorlib assembly (DLL).

To use other libraries, the proper assembly file (DLL) must be added to the project. Such as in adding the Microsoft.SPOT.Hardware to use a GPIO pin. Assembly files used by a project are managed as “References” in Visual Studio:



Important note: The emulator will only work with the Microsoft assemblies. GHI Electronics' libraries will not run on the emulator.

7.3. Important Information for the Following Examples

- To aid in portability, a number of NETMF embedded interfaces can be referenced with generic names. Those names are mapped internally to the specific processor. Alternatively, and probably more convenient, processor specific names can be used. The convenience comes when matching names to processor pins and schematics. For instance, the generic NETMF name `Cpu.Pin.GPIO_Pin0`, may be referenced as `EMX.P0_6` .

  Other examples are sited.
- The examples are not meant to refer to a particular EMX circuit.

7.4. Digital Inputs/Outputs (GPIO)

GPIO (General Purpose Input Output) are used to set a specific pin high or low states when the pin is used as an output. On the other hand, when the pin is an input, the pin can be used to detect a high or low state on the pin. High means there is voltage on the pin, which is referred to as “true” in programming. Low means there is no voltage on the pin, which is referred to as “false”. Pins can also be enabled with an internal weak pull-up or pull-down resistor.

Outputs

Here is a blink LED example.

```
using System;
using System.Threading;
using Microsoft.SPOT.Hardware;

public class Program
{
    public static void Main()
    {
        OutputPort LED = new OutputPort(Cpu.Pin.GPIO_Pin0, true);
        while (true)
        {
            LED.Write(true);
            Thread.Sleep(500);
            LED.Write(false);
            Thread.Sleep(500);

        }
    }
}
```

This is available through the Microsoft.SPOT.Hardware assembly.

The example above blinks an LED every one second (on for 500ms and off for another 500ms); however, it is not clear what pin on EMX will be controlled. Instead of using the generic `Cpu.Pin.GPIO_Pin0` name, the actual EMX name can be found in the GHI.Pins assembly. This example uses the actual EMX pin name using the GHI.Pins assembly.

```
using System;
using System.Threading;
using Microsoft.SPOT;
using Microsoft.SPOT.Hardware;

using GHI.Pins;

public class Program
{
    public static void Main()
    {
        OutputPort LED = new OutputPort(EMX.P0_6, true);
        while (true)
        {
            LED.Write(true);
            Thread.Sleep(500);
            LED.Write(false);
            Thread.Sleep(500);

        }
    }
}
```

Inputs

Reading Input pins is as simple! This example will blink an LED only when the button is pressed.

```
using System;
using System.Threading;
using Microsoft.SPOT;
using Microsoft.SPOT.Hardware;

using GHI.Pins;

public class Program
{
    public static void Main()
    {
        OutputPort LED = new OutputPort(EMX.P0_6, true);
        InputPort Button = new InputPort(EMX.P2_21,false, Port.ResistorMode.PullUp);
        while (true)
        {
            if (Button.Read() == true)
            {
                LED.Write(true);
                Thread.Sleep(500);
                LED.Write(false);
                Thread.Sleep(500);
            }
        }
    }
}
```

Note: The glitch filter feature is only available on interrupt-capable pins, any pin on port 0 or port 2.

Interrupt Pins

The beauty of modern and managed language shines with the use of events and threading. This example will set a pin high when a button is pressed. It should be noted here that the system in this example spends most its time a in a lower power state.

Note: Only pins on port 0 and port 2 are interrupt capable.

```
using System;
using System.Threading;
using Microsoft.SPOT;
using Microsoft.SPOT.Hardware;

using GHI.Pins;

public class Program
{
    public static OutputPort LED = new OutputPort(EMX.P0_6, true);
    public static void Main()
    {
        InterruptPort Button = new InterruptPort(EMX.P2_21,true, Port.ResistorMode.PullUp,
            Port.InterruptMode.InterruptEdgeBoth);

        Button.OnInterrupt += Button_OnInterrupt;
        // The system can do anything here, even sleep!
        Thread.Sleep(Timeout.Infinite);
    }

    static void Button_OnInterrupt(uint port, uint state, DateTime time)
    {
        LED.Write(state > 0);
    }
}
```

7.5. Analog Inputs/Outputs

Analog inputs can read voltages from 0V to 3.3V with a 10-Bit resolution. Similarly, the analog output can set the pin voltage from 0V to 3.3V (VCC to be exact) with 10-Bit resolution. These built in analog circuitry are not designed to be very accurate. For high accuracy, an external ADC can be added, using the SPI bus perhaps.

```
using System;
using Microsoft.SPOT;
using Microsoft.SPOT.Hardware;

public class Program
{
    public static void Main()
    {
        AnalogInput ain = new AnalogInput(Cpu.AnalogChannel.ANALOG_1);
        Debug.Print("Analog Pin =" + ain.Read());
    }
}
```

This is available through the Microsoft.SPOT.Hardware assembly.

7.6. PWM

The available PWM pins have a built-in hardware to control the ration of the pin being high vs low, duty cycle. A pin with duty cycle 0.5 will be high half the time and low the other half. This is used to control how much energy is transferred out from a pin. An example would be to dim an LED. With output pins, the LED can be on or off but with PWM, it can be set to 0.1 duty cycle to give the LED only 10% of the energy.

```
using System;
using Microsoft.SPOT;
using Microsoft.SPOT.Hardware;

public class Program
{
    public static void Main()
    {
        PWM LED = new PWM(Cpu.PWMChannel.PWM_1, 10000, 0.10, false);
        LED.Start();
    }
}
```

This is available through the the Microsoft.SPOT.Hardware.PWM assembly.

Some PWM pin channels can exceed the available enumerations in NETMF. Casting can be done to set the channel number is shown.

```
using System;
using Microsoft.SPOT;
using Microsoft.SPOT.Hardware;

public class Program
{
    public static void Main()
    {
        PWM LED = new PWM((Cpu.PWMChannel)9, 10000, 0.10, false);
        LED.Start();
    }
}
```

Another use of PWM is to generate tones. In this case, the duty cycle is typically set to 0.5 but then the frequency will be changed as desired.

In the case of servo motor control, or when there is a need to generate a pulse at a very specific timing, PWM provides a way to set the high and low pulse with.

7.7. Signal Generator

Using Signal Generator, developers can produce different waveforms. This is available on any digital output pin.

```
using System;
using System.Threading;
using Microsoft.SPOT;
using GHI.Pins;
using GHI.IO;

public class Program
{
    public static void Main()
    {
        uint[] signal = new uint[4] {1000,2000,3000,4000};
        SignalGenerator pin = new SignalGenerator(EMX.P2_21, false);

        pin.Set(false, signal);

        Thread.Sleep(Timeout.Infinite);
    }
}
```

While handeled in software, the SignalGenerator runs through internal interrupts in the background and so is not blocking to the system. Another Blocking methd is also provided for higher accuracy. For example, the blocking method can generate acarrier frequency. This is very useful for infrared remote control applications.

This is available through the GHI.Hardware assembly.

7.8. Signal Capture

Signal Capture monitors a pin and records any changes of the pin into an array. The recorded values are the times taken between each signal change.

```
using System;
using System.Threading;
using Microsoft.SPOT;
using Microsoft.SPOT.Hardware;
using GHI.Pins;
using GHI.IO;

public class Program
{
    public static void Main()
    {
        uint[] signal = new uint[100];
        SignalCapture pin = new SignalCapture(EMX.P2_21,Port.ResistorMode.Disabled);

        pin.Read(false, signal);

        Thread.Sleep(Timeout.Infinite);
    }
}
```

This is available through the GHI.Hardware assembly.

7.9. Serial Port (UART)

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

Serial Port	EMX Module 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.

```
using System;
using System.IO.Ports;
using System.Threading;
using Microsoft.SPOT;

public class Program
{
    public static void Main()
    {
        SerialPort COM1 = new SerialPort("COM1");
        int c = COM1.ReadByte();

        // ...
    }
}
```

This is available through the Microsoft.SPOT.Hardware.SerialPort assembly.

7.10. SPI

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

Important note: SPI2 is shared internally with the flash memory EMX uses. Using a chip select is required with devices connected using SPI2. Improper use of SPI2 will cause EMX to not boot or not work properly. The use of SPI1 is recommended.

```
using System.Threading;
using Microsoft.SPOT.Hardware;

public class Program
{
    public static void Main()
    {
        SPI.Configuration MyConfig =
             new SPI.Configuration(Cpu.Pin.GPIO_Pin1,
             false, 0, 0, false, true, 1000, SPI.SPI_module.SPI1);
        SPI MySPI = new SPI(MyConfig);

        byte[] tx_data = new byte[10];
        byte[] rx_data = new byte[10];

        MySPI.WriteRead(tx_data, rx_data);

        Thread.Sleep(Timeout.Infinite);
    }
}
```

This is available through the Microsoft.SPOT.Hardware assembly.

7.11. I2C

I2C is a two-wire addressable serial interface.

The EMX supports one master I2C port. Refer to the Pin-Out Description chapter for more information about I2C signals assignments to EMX hardware pins.

```
// Setup the I2C bus
I2CDevice.Configuration con =
    new I2CDevice.Configuration(0x38, 400);
I2CDevice MyI2C = new I2CDevice(con);
// Start a transaction
I2CDevice.I2CTransaction[] xActions =
    new I2CDevice.I2CTransaction[2];
byte[] RegisterNum = new byte[1] { 2 };
xActions[0] = I2CDevice.CreateWriteTransaction(RegisterNum);
```

This is available through the Microsoft.SPOT.Hardware assembly.

7.12. 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.

```
using System.Threading;
using Microsoft.SPOT.Hardware;
using GHI.IO;

public class Program
{
    public static void Main()
    {
        ControllerAreaNetwork.Message msg = new ControllerAreaNetwork.Message();
        msg.ArbitrationId = 0x123;
        msg.Data[0]= 1;
        msg.Length =1;
        msg.IsExtendedId = false;
        GHI.IO.ControllerAreaNetwork can = new ControllerAreaNetwork(
            ControllerAreaNetwork.Channel.One,
            ControllerAreaNetwork.Speed.Kbps500);

        can.SendMessage(msg);
        // ...
    }
}
```

This is available through the GHI.Hardware assembly.

There are two CAN channels on the EMX Module.

7.13. 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.

```
using System.Threading;
using Microsoft.SPOT;
using Microsoft.SPOT.Hardware;

public class Program
{
    public static void Main()
    {
        // Change this to correct GPI pin for the onewire used in the project!
        OutputPort myPin = new OutputPort(GHI.Pins.EMX.P2_22, false);

        OneWire ow = new OneWire(myPin);

        while (true)
        {
            if (ow.TouchReset() > 0)
            {
                Debug.Print("Device is detected.");
            }
            else
            {
                Debug.Print("Device is not detected.");
            }

            Thread.Sleep(10000);
        }
    }
}
```

This is available through the Microsoft.SPOT.Hardware.OneWire.

7.14. Graphics

The EMX Module supports 16-Bit color TFT displays. Developers can use almost any digital TFT display, up to 800x600. 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.

SPI-based displays can be utilized as well but using the native TFT interface allows for a better user experience, especially when displays are 320x240 (QVGA) or larger.

For developers wanting to connect VGA or HDMI monitors, a simple circuit is still needed to convert the 16Bit digital signals to analog RGB colors, such as Chrontel's CH7025.

With the EMX graphics support, users can leverage the NETMF graphics features such as:

- Windows Presentation Foundation (WPF)
- BMP, GIF (still) and JPEG image files.
- Fonts
- Simple Shapes

This simple example will run on the emulator and on the EMX Module similarly. It requires the Microsoft.SPOT.Graphics and Microsoft.SPOT.TinyCore.

```
using System.Threading;
using Microsoft.SPOT;
using Microsoft.SPOT.Hardware;
using Microsoft.SPOT.Presentation;
using Microsoft.SPOT.Presentation.Media;

public class Program
{
    public static void Main()
    {
        Bitmap LCD = new Bitmap(SystemMetrics.ScreenWidth, SystemMetrics.ScreenHeight);
        byte red = 0;
        int x = 0;
        while (true)
        {
            for (x = 30; x < SystemMetrics.ScreenWidth - 30; x += 10)
            {
                LCD.DrawEllipse(ColorUtility.ColorFromRGB(red, 10, 10), x, 100, 30, 40);
                LCD.Flush();
                red += 3;
                Thread.Sleep(10);
            }

        }
    }
}
```

Fonts

Thanks to NETMF, developers can convert TrueType font files to the TinyFNT format used on NETMF. The end results will look professionally stunning.

Glide

GHI Electronics has developed a high speed, lightweight full featured graphics/GUI framework called "Glide." The open-source code is available, with the Apache 2 license. This allows for a commercial and non-commercial use. For convenience the compiled libraries are included with GHI's SDK. This is the recommended method for graphics on the EMX Module. Glide includes a window designer as well https://www.ghielectronics.com/glide.

Contained in the GHI.Glide assembly.

Touch Screen

EMX Module supports displays with a four-wire resistive touch screen without the need for any additional hardware. However, the use of other touch displays can be supported by adding the appropriate controller, typically on the SPI bus.

Refer to the Pin-Out Description chapter for more information about touch screen signals (Touch_X_Left, T_X_Right, T_Y_Down, Touch_Y_UP) assignments to EMX hardware pins.

GHI Electronics' open-source graphics-library (Glide) is a good place to learn about touch handling if direct handling is necessary.

7.15. USB Host

The USB Host allows the use of USB Hubs, USB storage devices, joysticks, keyboards, mice, printers and more. Additionally, for USB devices that do not have a standard class, low level raw USB access is provided for bulk transfers.

```
using System.Threading;
using GHI.Usb.Host;
using GHI.Usb;
using Microsoft.SPOT;


public class Program
{
    static Mouse mouse;

    public static void Main()
    {

        // Subscribe to USBH event.
        Controller.DeviceConnected += Controller_DeviceConnected;

        // Sleep forever
        Thread.Sleep(Timeout.Infinite);
    }

    static void Controller_DeviceConnected(object sender, Controller.DeviceConnectedEventArgs
e)
    {
        if (e.Device.Type == Device.DeviceType.Mouse)
        {
            Debug.Print("Mouse Connected");
            mouse = new Mouse(e.Device);
            mouse.CursorMoved += mouse_CursorMoved;
            mouse.ButtonChanged += mouse_ButtonChanged;
        }

    }

    static void mouse_CursorMoved(Mouse sender, Mouse.CursorMovedEventArgs e)
    {
        Debug.Print("(x, y) = (" + e.NewPosition.X + ", " +
                    e.NewPosition.Y + ")");
    }
}
```

This is available through the GHI.Usb and GHI.Hardware assemblies.

7.16. Accessing Files and Folders

The File System feature in NETMF is near very similar to the full .NET and can be tested from within the Microsoft NETMF emulator with minor changes. Changes include removing any of the GHI library dependencies. There are no limits on file sizes and counts, beside the limits of the FAT file system itself. NETMF supports FAT16 and FAT32.
Files are made accessible on SD cards and on USB memory devices through the USB Host library.

Most online examples on how to use .NET to access files on PCs can be used to read and write files on the EMX Module. The GHI Electronics' online documentation has further examples as well. The only differences from the full .NET on the PC is the need to mount the media and differences in media names. The easiest way to know and handle the media names is by obtaining the root directly name and dynamically using that name.

This is available through the GHI.Hardware assembly for SD (or USB media); also required: assemblies for the file system functions System.IO and Microsoft.SPOT.IO.

```
using System;
using System.IO;
using Microsoft.SPOT;
using Microsoft.SPOT.IO;

using GHI.IO.Storage;

class Program
{
    public static void Main()
    {
        // ...
        // SD Card is inserted
        // Create a new storage device

        SD sdPS = new SDCard();

        // Mount the file system
        sdPS.Mount();

        // Assume one storage device is available, access it through
        // NETMF and display the available files and folders:
        Debug.Print("Getting files and folders:");
        if (VolumeInfo.GetVolumes()[0].IsFormatted)
        {
            string rootDirectory =
                VolumeInfo.GetVolumes()[0].RootDirectory;
            string[] files = Directory.GetFiles(rootDirectory);
            string[] folders = Directory.GetDirectories(rootDirectory);

            Debug.Print("Files available on " + rootDirectory + ":");
            for (int i = 0; i < files.Length; i++)
                Debug.Print(files[i]);

            Debug.Print("Folders available on " + rootDirectory + ":");
            for (int i = 0; i < folders.Length; i++)
                Debug.Print(folders[i]);
        }
        else
        {
            Debug.Print("Storage is not formatted. " +
                "Format on PC with FAT32/FAT16 first!");
        }
        // Unmount when done
        sdPS.Unmount();
    }
}
```

SD/MMC Memory

SD and MMC memory cards have similar interfaces. EMX supports both cards and also supports SDHC/SDXC cards. The interface runs through a true 4-bit SD interface. SD cards are available in different sizes but they are all of an identical function making them all supported on the EMX Module.

USB Mass Storage

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

7.17. Secure Networking (TCP/IP)

Networking is a crucial part of today's embedded devices and the internet of things IoT. NETMF includes a full TCP/IP stack with socket support and high level protocols, such as HTTP. The EMX Module networking functions works with media such as Ethernet and WiFi. PPP adds networking to a large range serial media/devices.

Secure networking is accomplished with ease, thanks to SSL support.

The Extensions

The way networking works on NETMF is very similar to the full desktop .NET. Also, the networking libraries work on the Microsoft NETMF emulator. This allows for testing and developing right on the desktop, through the emulator. GHI Electronics adds few important extensions to the system to initialize the networking interfaces. Developers can choose to add Ethernet or WiFi; these additions are typically only needed in the initialization and setup stage; they cannot be used on the emulator. If using the emulator, the few initialization lines can be commented out.

MAC address setting

All EMX Modules ship with the **same** default MAC address. This is good for testing a single device on internal networks. If using multiple devices or reaching the internet, a proper MAC address must be set.

To set the MAC address, FEZ Config can be used. Also, the EMX Module can set its own MAC through software.

```
byte[] newMAC = new byte[] { 0x00, 0x1A, 0xF1, 0x01, 0x42, 0xDD };
var enc = new GHI.Networking.EthernetBuiltIn();
enc.PhysicalAddress = newMAC;
```

This is available through the GHI.Networking assembly.

There is no need to set the MAC address when using WiFi as the system obtains the MAC from the WiFi module itself.

Tip: Some MAC addresses are not legal. The internet includes MAC address generators that can be used in testing.

IP address (DHCP or static):

DHCP (dynamic) IP and Static IP are both supported. If using dynamic IP, the EMX will not obtain an IP lease at power up -- DHCP can only be enabled from software. FEZ Config has a DHCP enable option but it has no effect on getting the IP lease on start-up.

```
var enc = new GHI.Networking.EthernetBuiltIn();
enc.EnableDhcp();
enc.EnableDynamicDns();
```

This is available through the GHI.Networking assembly.

Ethernet

The support for Ethernet is Builtin.

```
using System;
using Microsoft.SPOT.Hardware;
using Microsoft.SPOT;
using Microsoft.SPOT.Net;
using Microsoft.SPOT.Net.NetworkInformation;
using GHI.Pins;
using GHI.Networking;

public class Program
{
    static EthernetBuiltIn  ebi;
    static bool hasAddress = false;
    static bool available = false;

    public static void Main()
    {
        NetworkChange.NetworkAvailabilityChanged += NetworkChange_NetworkAvailabilityChanged;
        NetworkChange.NetworkAddressChanged += NetworkChange_NetworkAddressChanged;

        var ebi = new GHI.Networking.EthernetBuiltIn);
        ebi.Open();
        ebi.EnableStaticIP("192.168.1.100", "255.255.255.0", "192.168.1.0");
        ebi.EnableStaticDns(new string[] { "192.168.1.0" });

        while (!hasAddress || !available)
        {
            Debug.Print("Initializing");
            System.Threading.Thread.Sleep(100);
        }
        //Network ready now.
    }
    static void NetworkChange_NetworkAvailabilityChanged(object sender,
        NetworkAvailabilityEventArgs e)
    {
        Debug.Print("Network available: " + e.IsAvailable.ToString());
        available = e.IsAvailable;
    }
    static void NetworkChange_NetworkAddressChanged(object sender, EventArgs e)
    {
        Debug.Print("The network address has changed.");
        hasAddress = ebi.IPAddress != "0.0.0.0";
    }
}
```

This is available through the GHI.Networking assembly.

Wireless LAN WiFi

Any WiFi module with built in TCP/IP stack can be used through NETMF, which has many limitations; however, GHI Electronics adds support to WiFi internally through the NETMF's TCP/IP and SSL stacks. To utilize these libraries, Redpine's RS9110-N-11-22-04 (chip antenna) or RS9110-N-11-22-05 (uFL connector) must be used. The GHI Electronics' drivers for this module allows for real "Socket" connection over WiFi. This is not a simple WiFi-Serial bridge commonly used on embedded systems.

RS9110-N-11-21-01 WiFi module

This module from Redpine's Connect-io-n™ family is a complete IEEE 802.11bgn WiFi client device with a standard SPI interface to a host processor or data source. It integrates a MAC, baseband processor, RF transceiver with power amplifier, a frequency reference, an antenna, and all WLAN protocol and configuration functionality in embedded firmware to provide a self-contained 802.11bgn WLAN solution for a variety of applications. It supports WPA, WPA2, and WEP security modes in addition to open networks.

```
//Create just like the ENC28 or Builtin
var wifi = new GHI.Networking.WiFiRS9110(SPI.SPI_module.SPI1,
                 EMX.P0_6 // chip select
                 EMX.P2_21, // external interrupt
                 EMX.P2_22 // reset
); //change to target design
//Open and configure
wifi.Join("ssid", "password");
//...
```

This is available through the GHI.Networking assembly.

7.18. PPP

Point to Point (PPP) protocol is essential for devices needing to connect to mobile networks. While typical embedded devices use the mobile modem's built-in and very limited TCP/IP stack, systems with the EMX Module will enjoy the use of these modems through PPP and the internal NETMF-TCP/IP stack with SSL.

7.19. USB Client (Device)

The USB client interface is typically used as the EMX Debug Access Interface and for application deployment through Microsoft's Visual Studio (see The Loader and Firmware Debug Access Interface section). However, developers have control over the USB client interface. For example, the USB client can be made to simulate a USB keyboard or USB mass storage.

Tip: The USB Host is what *devices* connect to. For example a PC is a Host to *devices* like a mouse or a memory stick. The EMX has both interfaces, USB Host and USB Client (*device*).

This is available through the GHI.Usb along with the Microsoft.SPOT.Hardware.Usb

assemblies.

```
using System.Threading;
using GHI.Usb;
using GHI.Usb.Client;
using Microsoft.SPOT;
using Microsoft.SPOT.Hardware.UsbClient;
public class Program
{
public static void Main()
{
// Start keyboard
Keyboard kb = new Keyboard();
Debug.Print("Waiting to connect to PC...");
// Send "Hello world!" every second
while (true)
{
// Check if connected to PC
if (Controller.State ==
UsbController.PortState.Running)
{
// We need shift down for capital "H"
kb.Press(Key.LeftShift);
kb.Stroke(Key.H);
kb.Release(Key.LeftShift);
// Now "ello world"
kb.Stroke(Key.E);
kb.Stroke(Key.L);
kb.Stroke(Key.L);
kb.Stroke(Key.O);
kb.Stroke(Key.Space);
kb.Stroke(Key.W);
kb.Stroke(Key.O);
kb.Stroke(Key.R);
kb.Stroke(Key.L);
kb.Stroke(Key.D);
// The "!"
kb.Press(Key.LeftShift);
kb.Stroke(Key.D1);
kb.Release(Key.LeftShift);
// Send an enter key
kb.Stroke(Key.Enter);
}
Thread.Sleep(1000);
}
```

7.20. Extended Weak References (EWR)

EWR is a way for managed applications to store data on non-volatile memory. This is meant to be used as a configuration holder that does not change frequently. The NETMF documentation includes further details. A good example is included with the Microsoft .NET Micro Framework SDK.

See also the Battery RAM section.

7.21. Real Time Clock

The LPC2478 processor includes a real-time clock (RTC) that can operate while the processor is off, through a backup battery or a super capacitor. Appropriate circuitry is added to EMX so that the clock only needs power (3.3V VBAT) to operate. The NETMF system has its own time keeping that is independent from the real time clock. If actual time is need, the software should read the RTC and set the system's time.

Tip: The system time can also be set using time services through the internet.

```
using System;
using GHI.Processor;
using Microsoft.SPOT;

public class Program
{
    public static void Main()
    {
        DateTime DT;
        try
        {
            DT = RealTimeClock.GetDateTime();
            Debug.Print("Current Real-time Clock " + DT.ToString());
        }
        catch
        {
            // If the time is not good due to powerloss
            // an exception will be thrown and a new time will need to be set
            Debug.Print("The date was bad and caused a bad time");
            // This will set a time for the Real-time Clock clock to 1:01:01 on 1/1/2012
            DT = new DateTime(2012, 1, 1, 1, 1, 1);
            RealTimeClock.SetDateTime(DT);
        }

        if (DT.Year < 2011)
        {
            Debug.Print("Time is not resonable");
        }

        Debug.Print("Current Real-time Clock " + RealTimeClock.GetDateTime().ToString());
        // This will set the clock to 9:30:00 on 9/15/2011
        DT = new DateTime(2011, 9, 15, 7, 30, 0);
        RealTimeClock.SetDateTime(DT);
        Debug.Print("New Real-time Clock " + RealTimeClock.GetDateTime().ToString());
    }
}
```

7.22. Watchdog

Watchdog is used to reset the system if it enters an erroneous state. The error can be due to internal fault or the user's managed code. When the Watchdog is enabled with a specified timeout, the user must keep resetting the Watchdog counter within this timeout interval or otherwise the system will reset.

```
// Enable with 10 second timeout
GHI.Processor.Watchdog.Enable(10 * 1000);
while (true)
{
    // Do some work
    GHI.Processor.Watchdog.ResetCounter();
}
```

7.23. Power Control

Embedded devices often must limit power usage as much as possible. Devices may lower their power consumption in many ways:

1. Reduce the processor clock
2. Shutdown the processor when system is idle (keep peripherals and interrupts running)
3. Shutdown specific peripherals
4. Hibernate the system

A common way to wake a device is using the RTC alarm. Whenever the alarm goes off, it will wake the device. These examples require the GHI.Hardware and Microsoft.SPOT.Hardware assemblies.

Use Microsoft.SPOT.Hardware.HardwareEvent.OEMReserved2 for RTC alarm. When the program starts, it will set an RTC alarm for 30 seconds in the future and then hibernate until then.

```
using GHI.Processor;
using Microsoft.SPOT.Hardware;
using System;

public class Program
{
    public static void Main()
    {
        RealTimeClock.SetAlarm(DateTime.Now.AddSeconds(30));

        PowerState.Sleep(SleepLevel.DeepSleep, HardwareEvent.OEMReserved2);

        ///Continue on with your program here
    }
}
```

The device will awaken whenever an interrupt port is triggered. Some devices can use interrupts internally that can cause spurious wakeups if not disabled. Use Microsoft.SPOT.Hardware.HardwareEvent.OEMReserved1 for interrupts.

NETMF's interrupt ports only function when their glitch filter is enabled or they have an event handler subscribed.

```
using Microsoft.SPOT.Hardware;
using System;

public class Program
{
    public static void Main()
    {
        var interrupt = new InterruptPort(Cpu.Pin.GPIO_Pin0, true, Port.ResistorMode.PullUp,
Port.InterruptMode.InterruptEdgeHigh);
        interrupt.OnInterrupt += interrupt_OnInterrupt;

        PowerState.Sleep(SleepLevel.DeepSleep, HardwareEvent.OEMReserved1);

        ///Continue on with your program here
    }

    private static void interrupt_OnInterrupt(uint data1, uint data2, DateTime time)
    {
        //Interrupted
    }
}
```

7.24. In-Field Update

The In-field update feature allows the EMX Module to update itself! This powerful and flexible feature is very simple to use. The managed application or even the TinyCLR NETMF firmware can be updated. When need, this feature basically allocates a large memory buffer to hold the new software in RAM. The new software can be obtained from any source. It can be loaded from the network, from a file on a USB memory or SD card, and even from a serial port. There is no need to receive the entire new software at once. It can be received in chunks and the update process can be aborted at anytime. Power loss is safe during the new software load process as everything is done in RAM.

Once the entire new software is received and buffered in RAM, the already running software (old software) can execute a method in the in-field update library to copy all ram to flash. This happens in couple seconds and then the system reboots, running the new software.

The online documentation includes further details.

7.25. SQLite Database

SQLite is a software library that implements a self-contained server-less SQL database engine. SQLite is the most widely deployed SQL database engine in the world. Thanks to GHI Electronics' efforts, SQLite is part of the EMX Module's built in libraries. The SQLite website include further details and documentation. http://www.sqlite.org

```
using System;
using System.Collections;
using Microsoft.SPOT;
using GHI.SQLite;

public class Program
{
    public static void Main()
    {
        // Create a database in memory,
        // file system is possible however!
        Database myDatabase = new GHI.SQLite.Database();

        myDatabase.ExecuteNonQuery("CREATE Table Temperature"+
            " (Room TEXT, Time INTEGER, Value DOUBLE)");

        //add rows to table
        myDatabase.ExecuteNonQuery("INSERT INTO Temperature (Room, Time, Value)"+
            " VALUES ('Kitchen', 010000, 4423)");
        myDatabase.ExecuteNonQuery("INSERT INTO Temperature (Room, Time, Value)"+
            " VALUES ('Living Room', 053000, 9300)");
```

```
        myDatabase.ExecuteNonQuery("INSERT INTO Temperature (Room, Time, Value)"+
            " VALUES ('bed room', 060701, 7200)");

        // Process SQL query and save returned records in SQLiteDataTable
        ResultSet result = myDatabase.ExecuteQuery("SELECT * FROM Temperature");

        // Get a copy of columns orign names example
        String[] origin_names = result.ColumnNames;

        // Get a copy of table data example
        ArrayList tabledata = result.Data;


        String fields = "Fields: ";
        for (int i = 0; i < result.RowCount; i++)
        {
            fields += result.ColumnNames[i] + " |";
        }
        Debug.Print(fields);

        object obj;
        String row = "";
        for (int j = 0; j < result.RowCount; j++)
        {
            row = j.ToString() + " ";
            for (int i = 0; i < result.ColumnCount; i++)
            {
                obj = result[j, i];

                if (obj == null)
                    row += "N/A";
                else
                    row += obj.ToString();

                row += " |";
            }
            Debug.Print(row);
        }

        myDatabase.Dispose();
    }
}
```

This requires the GHI.SQLite assembly.

8. Advanced Use Of The Microprocessor

The EMX Module is based on the LPC2478 microcontroller. There are times when direct programming is needed. GHI has extended NETMF to allow assembly level access from managed code to the LPC2478. This chapter describes those features.

Important: All examples in this chapter use the GHI.Hardware assembly; add it to “References” in Visual Studio, see the Loading Assemblies section.

8.1. Register

This class is used for manipulating the processor registers directly. This example demonstrates access to the battery backed RAM:

```
        // write a 32 bit value into virtual uint[512] array mapped over RTC battery RAM
        // offset is the index into the array.
        void WriteBatteryRam(uint offset, uint a_value)
        {

            BatteryRam = new Register(0xE0084000 + (offset << 2));
            BatteryRam.Write(a_value);
        }
```

8.2. AddressSpace

Allows applications to read and write memory directly. This code reads a byte from address 0xA0000000.

```
GHI.Processor.AddressSpace.Read(0xA0000000);
```

8.3. Battery RAM

On the same power domain as the RTC (Real Time Clock) there is 2KB of RAM. The data on this RAM is retained on power loss as it is powered through the backup battery used by the RTC.

The Battery RAM's address range is 0xE0084000 to 0xE00847FF. The SRAM can be accessed word-wise (32-bit) only. the GHI Register class can be used to read and write to battery RAM.

The LPC2478 datasheet includes full details.

8.4. Runtime Loadable Procedure

Similar to code loaded by using a DLL (Dynamic Link Library), RLP (Runtime Loadable Procedure) allows compiled assembly code, perhaps from C/C++, to be loaded and run from the embedded application. For example a checksum or crypto procedure. While it can be done on the managed side, a native procedure will run a much faster. RLP also provides extensions that allow the native procedure to hook back into some of the services available in the managed side, like memory allocation.

RLP is very advanced and requires understanding of compilers and C/Assembly programming. It is documented in detail in the online documents and the reference guides.

9. Design Consideration

Required Pins

The following pins are recommended to be exposed or noted in any design:

- EMX access interface Serial COM1 (pins 5, 6), USB Device (pins 41, 42) or both.
- LMODE (Pin 16) can be set to high or low (high if left unconnected).
- Boot control (pins 3, 7, 53).

Interrupt Pins

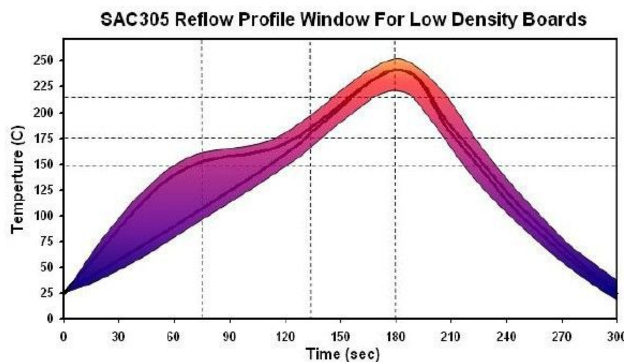
Only pins on ports 0 and 2 support interrupts.

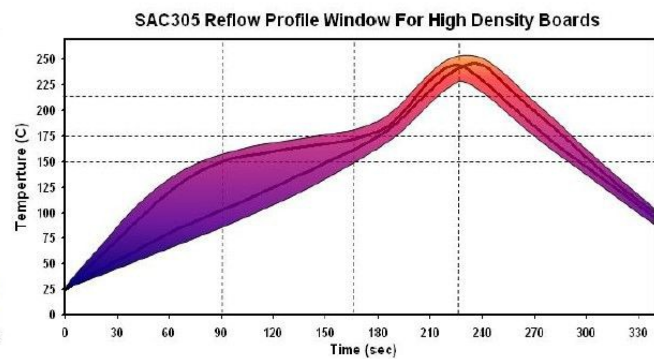
10. Soldering EMX

The EMX Module was designed to be easily machine-placed and hand-soldered. Static sensitive precautions should take place when handling the modules.

EMX Modules are not sealed for moisture, it is recommended to bake the module before reflow. The process of reflow can damage the EMX Module if the temperature is too high or exposure is too long. This lead-free reflow is used by GHI Electronics when machine-placing the EMX Module.

NOTE: The profiles shown are based on SAC 305 solder (3% silver, 0.5% copper). The thermal mass of the assembled board and the sensitivity of the components on it affect the total dwell time. Differences in the two profiles are where they reach their respective peak temperatures, as well as the time above liquids (TAL). The shorter profile of the two would apply to smaller assemblies, where as the longer profile would apply to larger assemblies, such as back-planes or high-density boards. The process window is described by the shaded area. These profiles are starting-points (mainly guidance), the particulars of an oven and the assembly will determine the final process.





RATE OF RISE 2°C / SEC MAX	*RAMP TO 150°C (302°F)*	*PROGRESS THROUGH 150°C-175°C (302°F-347°F)*	*TO PEAK TEMP 230°C-245°C (445°F-474°F)*	*TIME ABOVE 217°C (425°F)*	*COOLDOWN ≤ 4 °C / SEC*	*PROFILE LENGTH AMBIENT TO COOL DOWN*
Short Profiles	≤ 75 Sec	30-60 Sec	45-75 Sec	30-60 Sec	45± 15 Sec	2.75-3.5 Min
Long Profiles	≤ 90 Sec	60-90 Sec	45-75 Sec	60-90 Sec	45± 15 Sec	4.5-5.0 Min

Legal Notice

Licensing

The EMX Module, with all its built in software components, is licensed for commercial and non-commercial use. No additional fee or licensing is required.

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, and Windows are registered or unregistered trademarks of Microsoft Corporation.

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