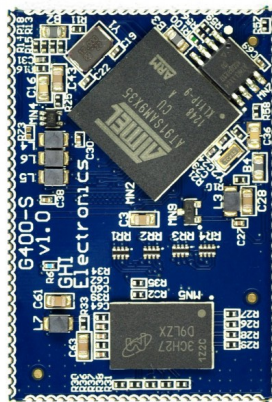


G400 SoM User Manual

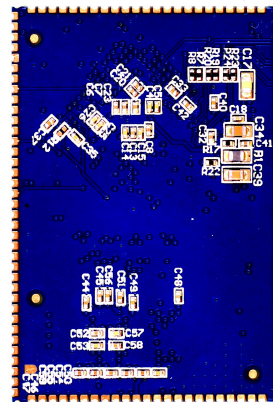
Rev. 0.01

August 27, 2014

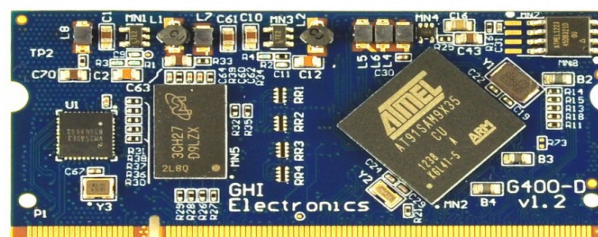
User Manual



G400-S Top



G400-S Bottom



G400-D Top

Revision History		
Rev No.	Date	Modification
Rev. 0.01	08/26/14	Preliminary version

*** This is a preliminary version ***

Table of Contents

1. Introduction.....	4	8.8. Serial Port (UART).....	46
1.1. The .NET Micro Framework.....	4	8.9. SPI.....	47
1.2. GHI Electronics and NETMF.....	5	8.10. I2C.....	48
1.3. Key Features.....	6	8.11. CAN.....	49
1.4. Example Applications.....	6	8.12. One-wire.....	50
2. The Hardware.....	7	8.13. Graphics.....	51
2.1. The SAM9X35 Microcontroller.....	7	Fonts.....	52
2.2. SDRAM.....	7	Glide.....	52
2.3. FLASH.....	7	Touch Screen.....	53
3. Pin-Out Description.....	8	8.14. USB Host.....	54
3.1. G400-S Pin-out Table.....	8	8.15. Accessing Files and Folders.....	55
3.2. G400-D Pin-out Table.....	11	SD/MMC Memory.....	57
4. Booting. Sequence and Control.....	18	USB Mass Storage.....	57
4.1. Boot Mode Pins.....	18	8.16. Secure Networking (TCP/IP).....	57
4.2. GHI Boot Loader vs. TinyBooter vs. G400 Firmware (NETMF/TinyCLR).....	19	The Extensions.....	57
4.3. Physical Bus Used During Programming, "the Debug Access Interface".....	20	MAC address setting.....	57
5. The GHI Boot Loader.....	21	IP address (DHCP or static):.....	58
5.1. Updating TinyBooter.....	21	Ethernet.....	58
5.2. Steps for Using "Update" (SAM-BA):.....	21	Wireless LAN WiFi.....	60
6. TinyBooter.....	24	8.17. PPP.....	61
6.1. TinyCLR (firmware) Update With FEZ Config (Recommended).....	25	8.18. USB Client (Device)	62
6.2. Firmware Update Using MFDeploy (Deprecated).....	26	8.19. Extended Weak References (EWR).....	64
7. NETMF TinyCLR (firmware).....	30	8.20. Real Time Clock.....	64
7.1. Assemblies and Version Matching.....	30	8.21. Watchdog.....	66
7.2. Deploying to the Emulator.....	31	8.22. Power Control.....	66
7.3. Deploying to the G400.....	35	8.23. In-Field Update.....	66
7.4. Targeting Different Versions of the Framework.....	35	8.24. SQLite Database.....	67
8. The Libraries.....	36	9. Advanced use of the Microprocessor	69
8.1. Finding NETMF Library Documentation.....	37	9.1. Register.....	69
8.2. Loading Assemblies.....	37	9.2. AddressSpace.....	69
8.3. Digital Inputs/Outputs.....	39	9.3. Runtime Loadable Procedure.....	69
Interrupt Pins.....	42	10. design Consideration.....	70
8.4. Analog Inputs/Outputs.....	43	10.1. Required Pins.....	70
8.5. PWM.....	43	10.2. SPI Channels.....	70
8.6. Signal Generator.....	44	10.3. Watchdog.....	70
8.7. Signal Capture.....	45	11. Soldering the G400.....	71
		Legal Notice.....	72
		Licensing.....	72
		Disclaimer.....	72

1. Introduction

The G400 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.

A simple two layer circuit, with just power and some connectors, can utilize the G400 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 within 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 G400. GHI Electronics continuously maintains, upgrades and solves any of the issues on the G400's firmware, to provide regular and free releases. Users can simply load the new software on the G400 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. Key Features

- .NET Micro Framework
- 400Mhz ARM 926EJ-S processor
- 128MB RAM
- 4MB MB FLASH
- Embedded LCD controller
- 102 GPIO Pins
- 102 Interrupt Inputs
- 2 SPI
- I2C
- 4 UART
- 2 CAN Channels
- 12 10-Bit Analog Input
- 10-Bit Analog Output
- 4Bit SD/SDHC/MMC Memory card interface
- 4 PWM
- 160 mA max
- 60 mA Hibernate Mode
- -40°C to +85°C Operational
- RoHS Lead Free
- TCP/IP Stack (.NET sockets)
- SSL secure networking
- WiFi
- PPP
- One-wire on all IOs
- USB Host/Device
- Graphics (image, font and controls)
- SQLite database
- File System (SD and USB Sticks)
- Native extensions RLP
- Real Time Clock
- In-Field-update
- Dimensions: (32.05mm x 67.49mm x 4.84mm)

1.4. Example Applications

- Vending machines, POS Terminals
- Measurement tools and testers
- Networked sensors
- Robotics
- Central alarm system
- Smart appliances
- Industrial automation devices

2. The Hardware

The G400 core components includes the processor, 4MB flash, and 128MB RAM.

The small, 38.1 x 26.7 x 3.55 mm, 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 G400's long list of available features.

2.1. The SAM9X35 Microcontroller

The SAM9X35 microcontroller in the G400 is an 400Mhz, 32Bit, 926EJ-S ARM architecture. It incorporates:

- 16KB Data and Instruction Caches
- internal 32KB SRAM, single-cycle at system speed

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

2.2. SDRAM

128MB of SDRAM comes standard with the G400

2.3. FLASH

4MB flash is available on the G400. One of the library extensions provided allows the user application to be updated in field, even remotely (see Runtime Loadable Procedure.)

3. Pin-Out Description

Many signals on the G400 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
- Pins are 3.3V levels
- Advanced details on all pins can be found in the SAM9X35 datasheet from Atmel's website

3.1. G400-S Pin-out Table

G400-S	GPIO	Multiplexed Function(s)	Notes
1	PD0		
2	PD4		
3		DIBP	
4	PA27		
5	PA16	SD CMD	
6	PA8	COM4 RX	
7	PA3	COM2 CTS	
8	PA2	COM2 RTS	
9	PC28	LCD H Sync COM3 CTS	
10	PC23		
11	PC5	LCD Green 0	
12	PC1	LCD Blue 1	
13		VDDIOM1	
14		SHDN	
15		JTAGSEL	
16		GND1	
17	PB3	ERXDV	Do not connect. Leave Floating
18	PB1	ERX1	Do not connect. Leave Floating
19		WKUP	
20	PB18		
21	PB8	AD9	
22	PB14	AD3	
23	PB12	AD1	

G400-S	GPIO	Multiplexed Function(s)	Notes
24	PB6	EMDC	
25	PB15	AD7	Do not connect. Leave Floating
26	PB0	AD4	
27	PB5	ERX0	Do not connect. Leave Floating
28	PC2	EMDIO	Do not connect. Leave Floating
29	PC9	LCD Blue 2	
30	PC11	LCD Green 4	
31	PC12	COM5 RX	
32	PC24	LCD Red 0	
33	PA0	LCD Red 1	
34	PC21		
35	PC19	COM2 TX	
36	PC22	PWM3	
37	PA7	PWM1	
38	PA4	COM4 TX	
39	PB11	LDR1	Boot Mode Control
40	PB13	AD0	
41	PB10	AD2	
42		ETX1	Do not connect. Leave Floating
43		AD11	
44	PB7	VCC 3.3V	
45	PB2	VBAT	
46	PC0	ETXEN	Do not connect. Leave Floating
47	PC6	AD8	Do not connect. Leave Floating
48		ERXER	Do not connect. Leave Floating
49	PC10	LCD Blue 0	
50	PC3	LCD Green 1	
51	PC15	VDDIOM2	
52	PC18	LCD Green 5	
53		LCD Blue 3	
54	PC13	LCD Red 4	
55	PC31	PWM 0	
56		VDDCORE1	
57	PC26	LCD Red 2	
58	PC30		
59	PA12	GND2	
		LCD Clock	
		SPI1 MOSI	Use as SPI Only

G400-S	GPIO	Multiplexed Function(s)	Notes
60	PB16	AD5	
61	PB17	AD6	
62	PB9	ETX0 AD10	Do not connect. Leave Floating
63	PB4	ETXCK	Do not connect. Leave Floating
64	PC4	LCD Blue 4	
65	PC7	LCD Green 2	
66	PC8	LCD Green 3 COM5 TX	
67	PC14	LCD Red 3	
68	PC16	COM6 TX	
69	PC20	PWM2	
70	PC17	COM6 Rx	
71	PC27	LCD V Sync COM3 RTS	
72	PC29	LCD Data Enable	
73	PA5	COM3 TX CAN2 TX	
74	PA1	COM2 RX	
75	PA11	SPI1 MISO	Use as SPI Only.
76	PA10	COM1 TX CAN1 TX	
77	PA9	COM1 RX CAN1 RX	Must be pulled high when booting entering Loader Mode.
78	PA15	SD D0	
79	PA18	SD D1	
80	PA20	SD D3	
81	PA23	SPI2 CLK	
82	PA28		
83	PA31	I2C SCL	Open drain port
84		VCC 3.3V	
85		TCK	
86		USB D+	
87		USB D-	
88		USB H0+	
89		USB H0-	
90		USB H1+	
91		USB H1-	
92		TDI	
93		NRST	

G400-S	GPIO	Multiplexed Function(s)	Notes
94		TMS	
95		BMS	
96		PWR_EN	
97		GND3	
98	PA13	SPI1 SCK	Use as SPI Only
99	PA19	SD D2	
100	PA21	SPI2 MISO	
101	PA24	LDR0	Boot Mode Control
102	PA25	MODE(USB/COM1#)	
103		VDDCORE2	
104		TDO	
105		NTRST	
106	PA6	COM3 RX CAN2 RX	
107	PA17	SD CLK	
108	PA22	SPI2 MOSI	
109	PA26		
110	PA30	I2C SDA	Open drain port
111	PA29		
112		RTCK	
113		DIBN	
114	PD2		
115	PD1		
116	PD7		
117	PD3		
118	PD5		
119	PD6		
120		GND4	

3.2. G400-D Pin-out Table

G400-D	GPIO	Multiplexed Function(s)	Notes
1		Ground	
2		Ethernet TX-	
3			
4		Ethernet TX+	
5		Ground	
6		Ethernet RX-	
7			

G400-D	GPIO	Multiplexed Function(s)	Notes
8		Ethernet RX+spi	
9			
10		Ethernet Speed	
11		Ethernet Link	
12			
13		Ground	
14			
15			
16			
17			
18			
19			
20		VCC 3.3V	
21			
22			
23			
24			
25			
26			
27		Ground	
28			
29			
30			
31			
32		VCC 3.3V	
33			
34			
35	PB3	ERXDV	Do not connect. Leave Floating
36	PB4	ETXCK	Do not connect. Leave Floating
37	PB5	EMDIO	Do not connect. Leave Floating
38	PB6	EMDC AD7	Do not connect. Leave Floating
39	PB7	ETXEN AD8	Do not connect. Leave Floating
40		Ground	
41		Ground	
42		DIBN	
43		DIBP	
44		BMS	
45		JTAGSEL	

G400-D	GPIO	Multiplexed Function(s)	Notes
46		VCC 3.3V	
47	PB2	ERXER	Do not connect. Leave Floating
48	PB9	ETX0 AD10	Do not connect. Leave Floating
49	PB10	ETX1 AD11	Do not connect. Leave Floating
50	PB0	ERX0	Do not connect. Leave Floating
51		Ground	
52	PB1	ERX1	Do not connect. Leave Floating
53			
54			
55			
56			
57	PD18		
58	PD17		
59	PD16		
60		VCC 3.3V	
61	PD15		
62	PD14		
63	PD13		
64	PD12		
65		Ground	
66	PD11		
67	PD10		
68	PD9		
69	PD8		
70			
71			
72		VCC 3.3V	
73			
74			
75			
76			
77			
78			
79		Ground	
80			
81			
82			
83	PA14		Do not connect. Leave Floating

G400-D	GPIO	Multiplexed Function(s)	Notes
84			
85	PC25		
86			
87			
88		VCC 3.3V	
89			
90			
91	PB8	AD9	
92	PD2		
93	PC23		
94	PD0		
95		Ground	
96	PB18		
97	PB11	AD0	
98	PA5	COM3 TX CAN2 TX	
99	PA6	COM3 RX CAN2 RX	
100	PC22		
101	PC31		
102	PA0	COM2 TX	
103	PA1	COM2 RX	
104	PB12	AD1	
105	PC18	PWM0	
106		VCC 3.3V	
107		SPI1 MISO	Use as SPI Only
108		SPI1 MOSI	Use as SPI Only
109		SPI1 CLK	Use as SPI Only
110	PB17	AD6	
111	PA4	LDR1	Boot Mode Control
112	PC19	PWM1	
113		Ground	
114	PB16	AD5	
115	PA30	I2C SDA	Open drain port
116	PA31	I2C SCL	Open drain port
117	PA9	COM1 RX CAN1 RX	Must be pulled high when booting entering Loader Mode.
118	PA10	COM1 TX CAN1 TX	
119	PC24		

G400-D	GPIO	Multiplexed Function(s)	Notes
120	PA2	COM2 RTS	
121	PA3	COM2 CTS	
122	PD7		
123	PA15	SD D0	
124		VCC 3.3V	
125	PA16	SD CMD	
126	PA17	SD CLK	
127	PA18	SD D1	
128	PA19	SD D2	
129	PA20	SD D3	
130	PC21	PWM3	
131		Ground	
132	PC26		
133	PC20	PWM2	
134	PA24	LDR0	Boot Mode Control
135	PA25	MODE(USB/COM1#)	
136	PA26		
137	PA27		
138	PA28		
139	PA29		
140	PC16	COM6 TX	
141	PC17	COM6 RX	
142		VCC 3.3V	
143	PC27	LCD V Sync COM3 RTS	
144	PC28	LCD H Sync COM3 CTS	
145	PC30	LCD Clock	
146	PC29	LCD Data Enable	
147	PD3		
148	PD4		
149	PD5		
150	PD6		
151		Ground	
152	PC0	LCD Blue 0	
153	PC1	LCD Blue 1	
154	PC2	LCD Blue 2	
155	PC3	LCD Blue 3	
156	PC4	LCD Blue 4	
157	PB13	AD2	

G400-D	GPIO	Multiplexed Function(s)	Notes
158	PB14	AD3	
159	PB15	AD4	
160		VCC 3.3V	
161	PC5	LCD Green 0	
162	PC6	LCD Green 1	
163	PC7	LCD Green 2	
164	PC8	LCD Green 3 COM5 TX	
165	PC9	LCD Green 4 COM5 RX	
166	PD1		
167	PA8	COM4 RXD	
168	PC15	LCD Red 4	
169		Ground	
170	PC10	LCD Green 5	
171	PC11	LCD Red 0	
172	PC12	LCD Red 1	
173	PC13	LCD Red 2	
174	PC14	LCD Red 3	
175	PA23	SPI2 CLK	
176	PA21	SPI2 MISO	
177		WKUP	
178	PA22	SPI2 MOSI	
179		SHDN	
180		VCC 3.3V	
181		PWM_EN	
182		USB3 HOST D+	
183		VDDBU	
184		USB3 HOST D-	
185		Ground	
186		Ground	
187		NRST	
188		USB2 HOST D+	
189		RTCK	
190		USB2 HOST D-	
191		TDO	
192		VCC 3.3V	
193		NTRST	
194		USB1 CLIENT D+	
195		TDI	

G400-D	GPIO	Multiplexed Function(s)	Notes
196		USB1 CLIENT D-	
197		TCK	
198		Ground	
199		TMS	
200	PA7	COM4 TXD	

4. Booting. Sequence And Control

Software on the G400 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 and then runs 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, LDR0 and LDR1, are left floating (internal pull up), or pulled high externally, the default boot-up sequence executes as the following:

- The GHI Boot loader initializes Flash and RAM memory and looks for a valid TinyBooter and passes execution to it.
- TinyBooter prepares the G400 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 101, LDR0	Pin 38, LDR1	Effect
(Ignored)	High	Default, execute all levels
High	Low	Execute the Boot Loader and TinyBooter but do not execute NETMF TinyCLR
Low	Low	Execute Boot Loader but do not execute TinyBooter*

* This mode is not currently implemented. It will be added in a future release. Check the release notes in the SDK.

4.2. GHI Boot Loader vs. TinyBooter vs. G400 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 the G400 TinyBooter or for low level G400 flash maintenance.	Used to update the G400 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 in the G400's flash memory. Can't be updated.	Replaceable using the GHI Boot Loader	Replaceable; 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.

* This mode is not currently implemented. It will be added in a future release. Check the release notes in the SDK.

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.

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

4.3. Physical Bus Used During Programming, “the Debug Access Interface”

The communication between a PC and the various loaders is over USB or UART (serial port). This, “Debug Access Interface,” is used for updating, deploying or debugging the software components.

The MODE Pin is used to select USB vs serial. When the pin is high or left floating (internal weak pull up resistor) the system will run in serial mode. If a direct connection between the G400's UART and another UART based device a null modem (cross-over cable) is needed. If connecting to an RS232 device, such as a true COM port on a PC, requires RS232 to/from TTL level converter.

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 the NETMF firmware.

For the G400, if PA25 is connected to ground then COM1 (Pins PA9, PA10) is enabled as the debug interface.

5. The GHI Boot Loader

The G400 boot Loader software is under development. When implemented it supports an easy to use, ASCII based, command processor for managing the flash; including loading TinyBooter.

The Booting. Sequence and Control chapter provides information on how to choose the access interface.

5.1. Updating TinyBooter

TinyBooter is updated by erasing Flash storage, and then loading the new TinyBooter into Flash. The GHI SDK installation kit places all the necessary tools on the development system. They are installed in the folder "C:\Program Files (x86)\GHI Electronics\GHI NETMF v4.3 SDK\Firmwares\G400\Loader". The version number (4.3 here) will match your SDK.

IMPORTANT: Check the tools directory for a "README" file; if there, it contains important information not available when this manual was published.

SAM-BA, the Atmel In-System Programmer, is the application used to load a new TinyBooter onto the G400. To make the use of SAM-BA easier, GHI supplies another program named "Update" (either a .BAT or a .EXE file).

5.2. Steps for Using "Update" (SAM-BA):

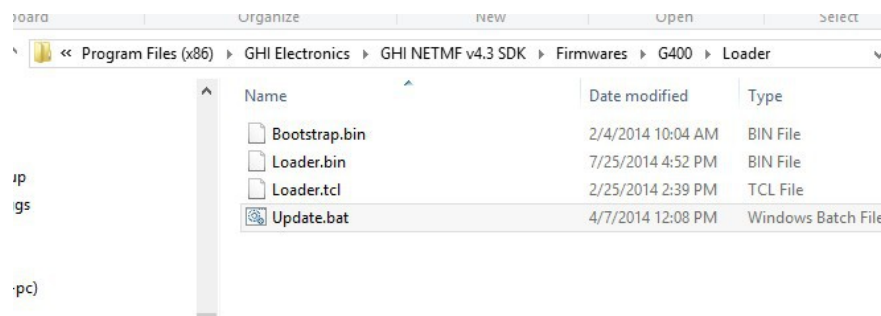
Boot the G400 into SAM-BA Loader mode by grounding PA11 and resetting the module.

For .NET Gadgeteer mainboards based on the G400 check the schematic to find the socket number that has a pin to PA11.

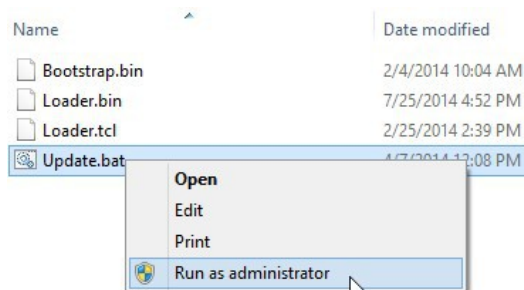
Open the Devices and Printers control panel and find the COM port allocated to the G400:



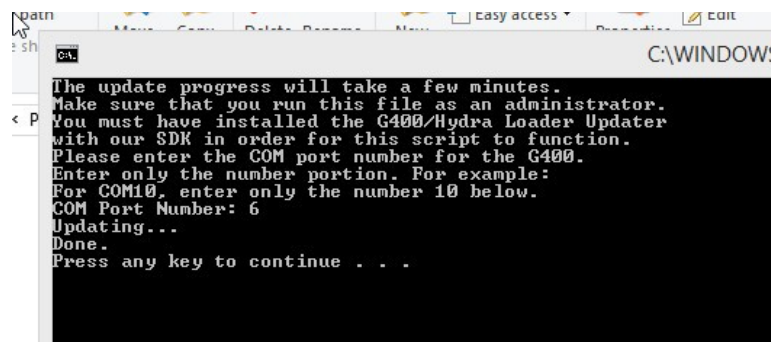
In this case, the G400 was allocated as COM6. Navigate to the folder with the Update utility and TinyBooter image files:



Right click on the Update program. And choose “Run as administrator”



Enter the COM port number, in this example 6. The Update takes a little while, then prints



“Done”

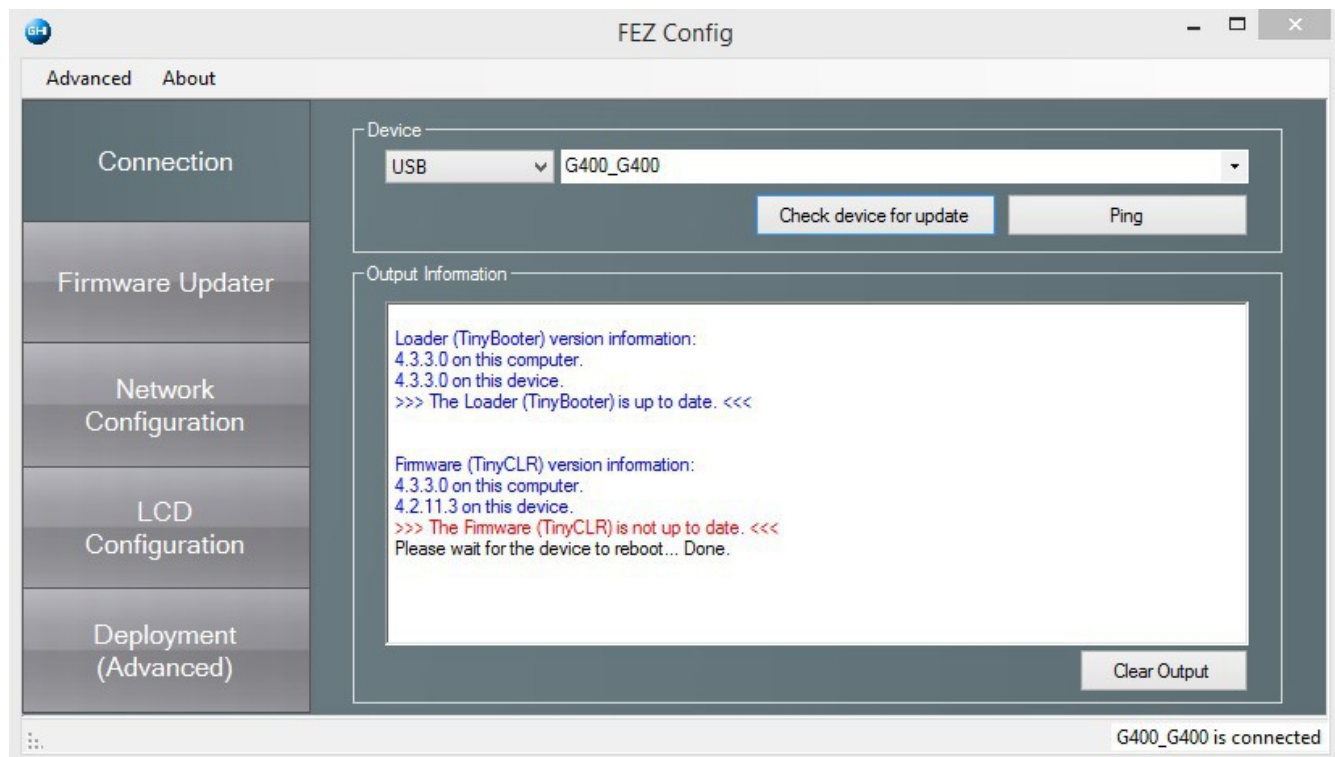
6. TinyBooter

TinyBooter has two functions;

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

The Booting. Sequence and Control 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.



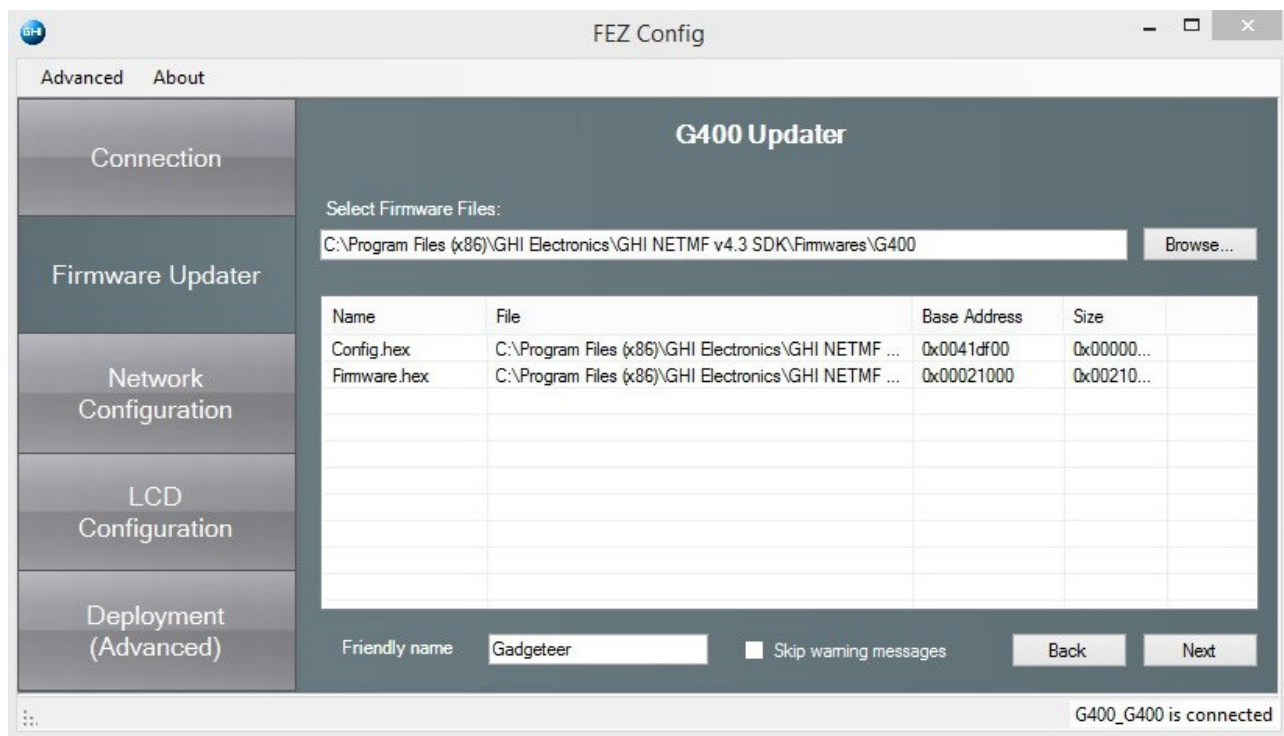
If TinyBooter needs to be updated , do it before updating TinyCLR. In the above example, TinyBooter is up-to-date (for more information, see Updating TinyBooter).

In the picture above, FEZ Config determined that the firmware needs an update. There are two tools available for updating firmware, GHI's FEZ Config or Microsoft's MFDeploy. The recommended method is the user friendly FEZ Config.

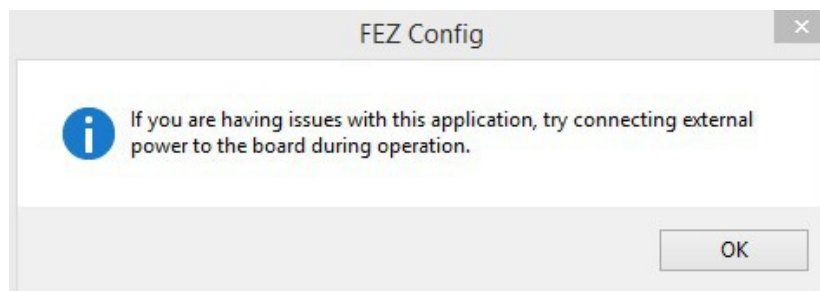
NOTE: Updates can also be performed using In-Field Update.

6.1. TinyCLR (firmware) Update With FEZ Config (Recommended)

1. Connect the G400 to the PC.
2. Launch FEZ Config and click on “Check device for update” button.
3. If the firmware needs updating (as shown by the picture above), click on the “Firmware Updater” tab on the left, click on the “Next button”:
4. After FEZ Config selects the firmware and the default configuration files, click “Next”



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



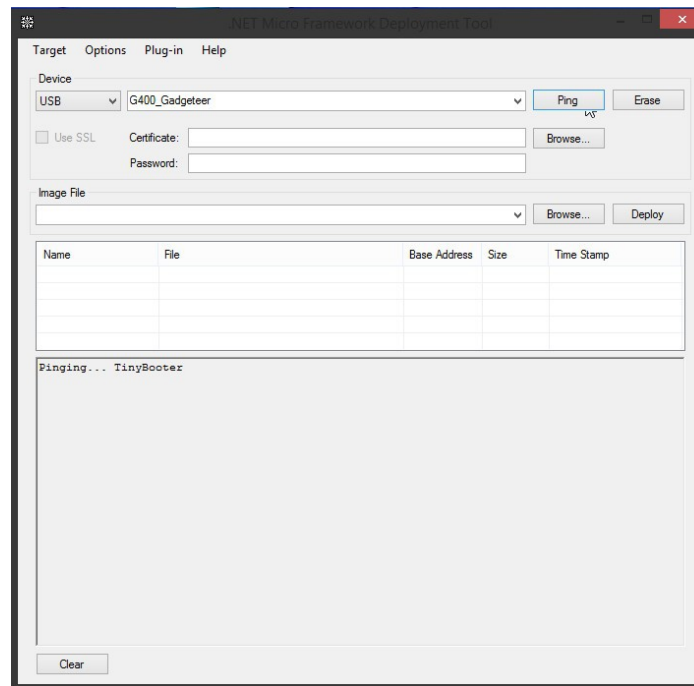
6. As the update occurs, the steps and progress are shown. When it is finished, the module is ready to be flashed with NETMF applications.

6.2. Firmware Update Using MFDeploy (Deprecated)

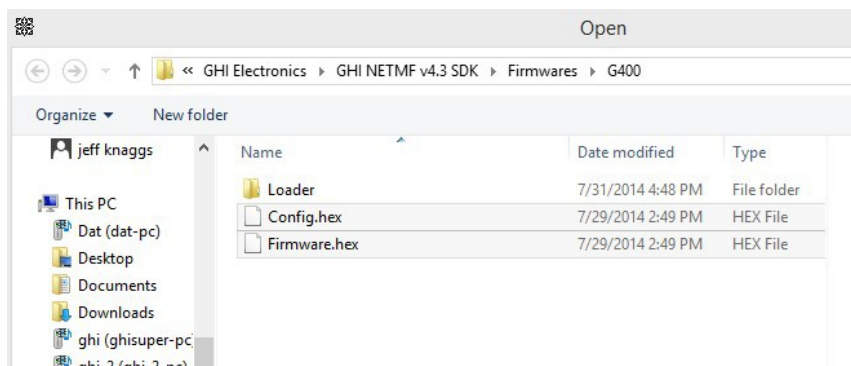
It is *strongly* recommended to use GHI Electronics' FEZ Config tool for updating the G400 (as described in the previous section). This section describes the use of MFDeploy to update the G400's firmware.

1. Set the pins as described in Boot Mode Pins to enter TinyBooter.
2. Run MFDeploy and select “USB” or “Serial” from the Device list, the available devices or COM ports appear in the drop down list.

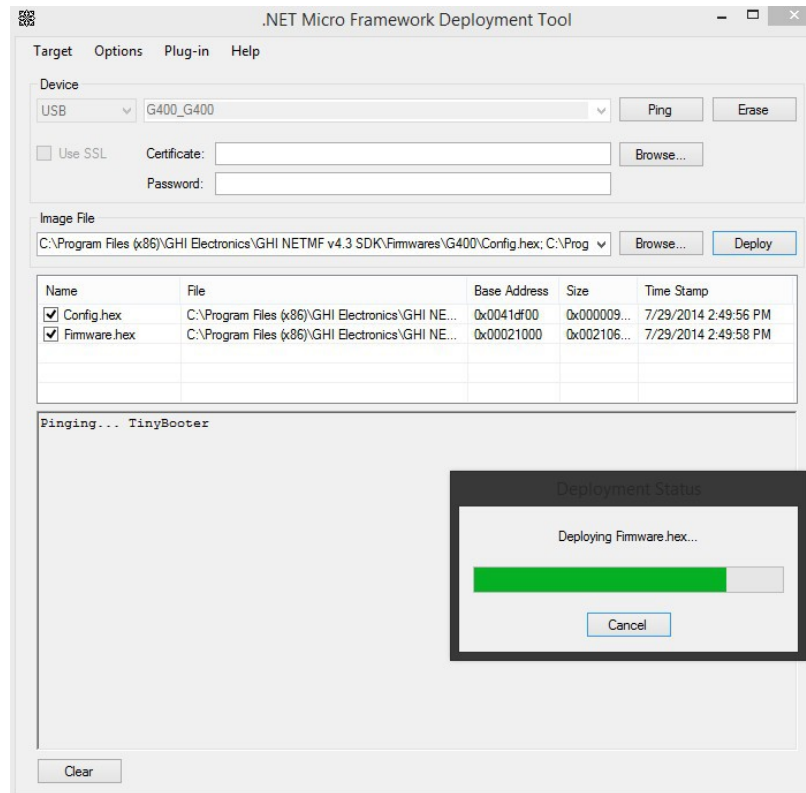
3. Check the communication between MFDeploy by pressing “Ping”; the output window should contain “Pinging... TinyBooter” or “Pinging... TinyCLR”.



4. Add the G400's 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\G400”. Select **all** of the firmware hex files at once.

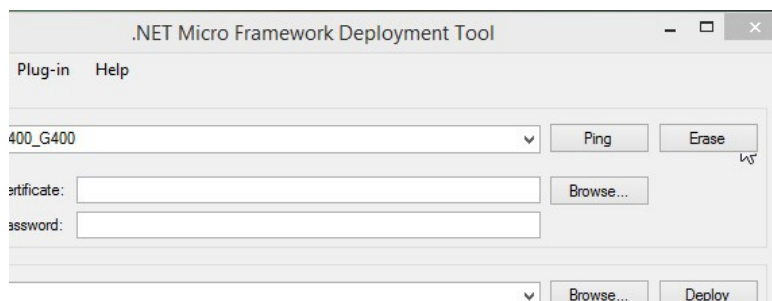


5. Start deploying the firmware by pressing “Deploy”.

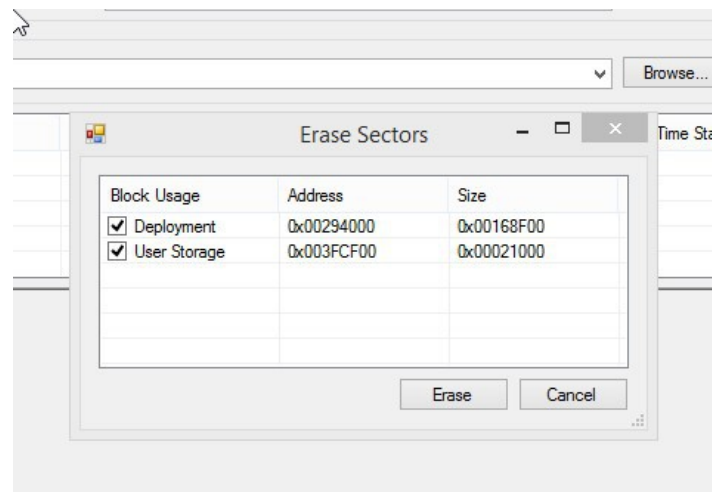


6. Loading the files takes a little while. Upon completion, the firmware (TinyCLR) will execute. Double check the version number to make sure the correct firmware is loaded.

Loading does not erase any resident program; that is the function of the “Erase” button:



MFDeploy will show the regions being erased.



When done, the G400 will be rebooted into TinyCLR

```
GC: 1msec 156492 bytes used, 66949272 bytes available
Type 0F (STRING) ): 24 bytes
Type 15 (FREEBLOCK) ): 66949272 bytes
Type 1E (BINARY_BLOB_HEAD) ): 135216 bytes
Type 34 (APPDOMAIN_HEAD) ): 72 bytes
Ready.
Done.
Waiting for debug commands...
```

7. NETMF TinyCLR (firmware)

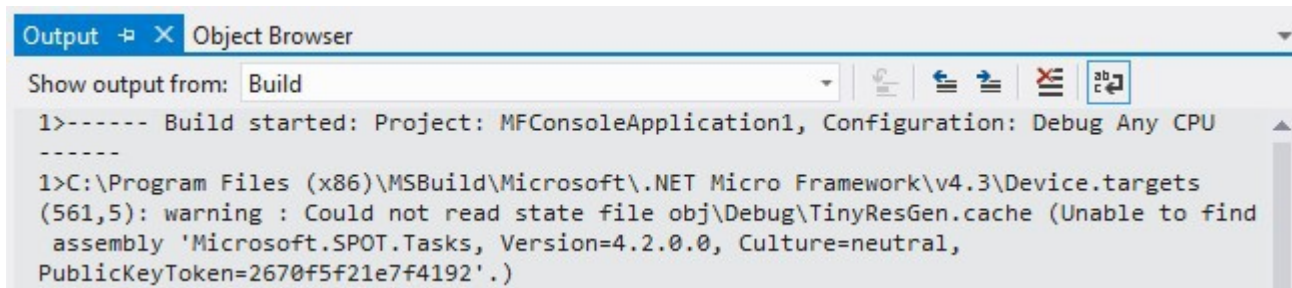
Other than an installed program, the Firmware is the main piece of embedded software running on the G400. The Firmware interprets and runs the user's managed application and it is what Microsoft's Visual Studio uses to deploy, hook-into and debug the managed application. As explained in Physical Bus Used During Programming, “the Debug Access Interface” 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.

7.1. Assemblies and Version Matching

GHI Electronics constantly makes improvements and adds new features to the extensions it supplies for NETMF. When installing a new SDK that *replaces an existing* SDK, there are some common compilation and deployment errors that may occur. These only occur with applications that *were developed with the previous* SDK. The errors, and their quick and easy fixes are described here.

1. Sometimes the improvements made by GHI will affect arguments, names, and other parts of a class. Which can cause compilation errors. This is easily corrected by looking at the current library documentation on GHI's website. Go to <https://www.ghielectronics.com/support/netmf> and follow the “Library Documentation” link.
2. When an application is compiled, it has a list of any assemblies needed. When the executable application is loaded, so are the assemblies. Assembly versions are checked against versions of libraries in the Firmware. For new projects with a new SDK, this is not a problem. But if an old project/solution is used with the new SDK, there *may* be problems with version mismatch (due to the fact that the *existing* Visual Studio projects may include a local copy of the assemblies supplied by the old SDK). During loading, the cached libraries are used and may not match what is found. For example: an application was previously compiled with 4.2, then the 4.3 SDK is installed. The application successfully compiles with 4.3. However, when the deployment process begins to load it uses the local copies of the 4.2 assemblies; causing loading errors. Visual Studio's Output panel will contain something like:



```
Output  Object Browser
Show output from: Build
1>----- Build started: Project: MFConsoleApplication1, Configuration: Debug Any CPU
-----
1>C:\Program Files (x86)\MSBuild\Microsoft\.NET Micro Framework\v4.3\Device.targets
(561,5): warning : Could not read state file obj\Debug\TinyResGen.cache (Unable to find
assembly 'Microsoft.SPOT.Tasks, Version=4.2.0.0, Culture=neutral,
PublicKeyToken=2670f5f21e7f4192'.)
```

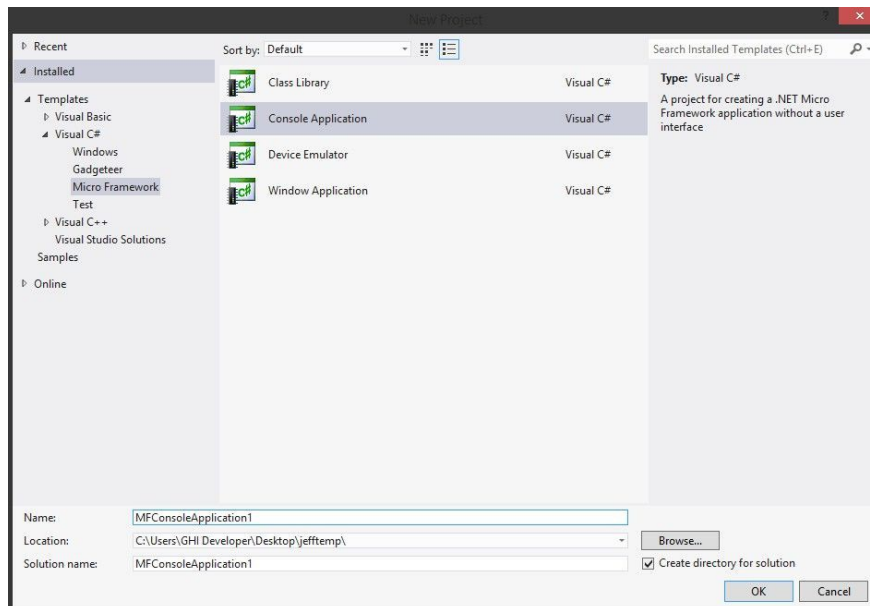
there is further discussions of assemblies in the Loading Assemblies section of chapter 8.

7.2. Deploying to the Emulator

Once the latest SDK is installed and the G400 is loaded with the new firmware, application development can begin.

The NETMF SDK includes an emulator for running applications on the PC. For the first part of this project, the emulator will be used to run a very simple application; after that, the same program is run on the G400 hardware.

Open Visual Studio, select FILE->New Project. There should be a “Micro Framework” option in the left menu. Click on it, select “Console Application”. C# is selected in this example but Visual Basic will be very similar.



Click the “OK” button; a project and program are created. The project has only one C# file, called Program.cs. C# source files are listed in the “Solution Explorer” window. If the Solution Explorer is not opened, use the VIEW->Solution Explorer menu.

Double click on Program.cs to open the file in the editor.

```
using System;
using Microsoft.SPOT;

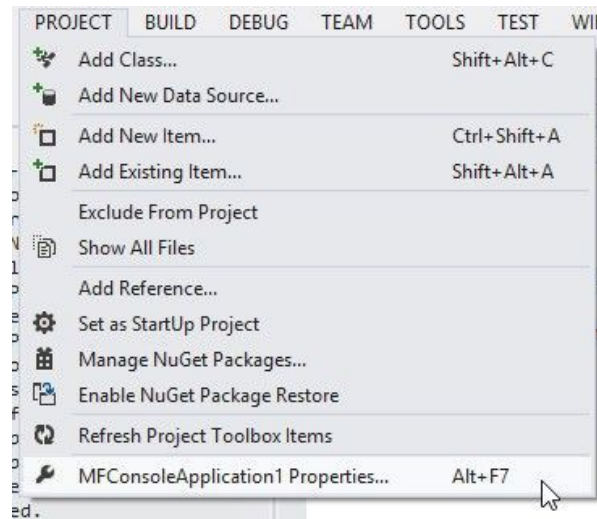
namespace MFConsoleApplication1
{
    public class Program
    {
        public static void Main()
        {
            Debug.Print(
                Resources.GetString(Resources.StringResources.String1));
        }
    }
}
```

For simplicity change the code to match the following.

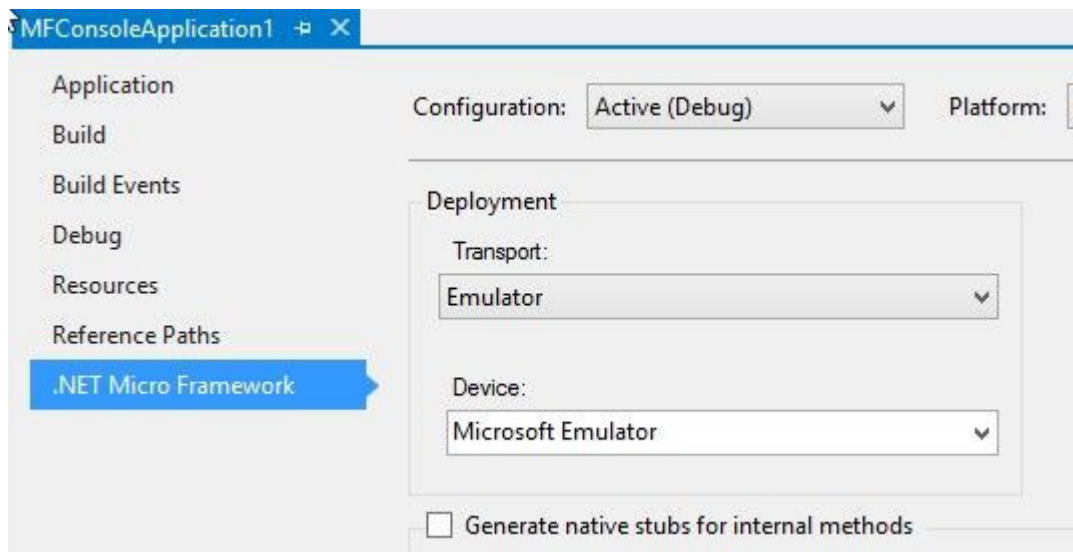
```
using Microsoft.SPOT;

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

When compiling a NETMF Console Project, there are options in Video Studio that control where the program is loaded and run. Select the menu item for the project's properties:

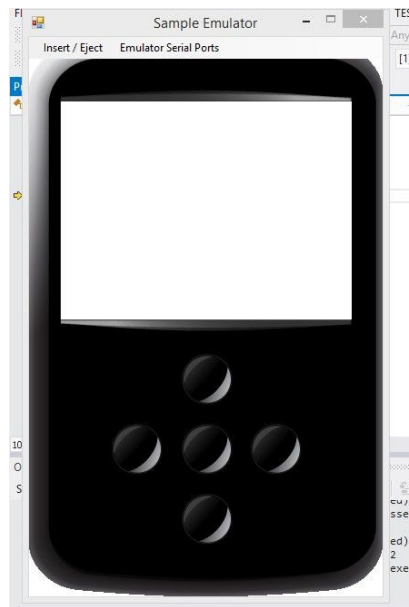


In the Properties window, on the left side tabs, select “.NET Micro Framework” and make sure that "Transport:" is "Emulator."

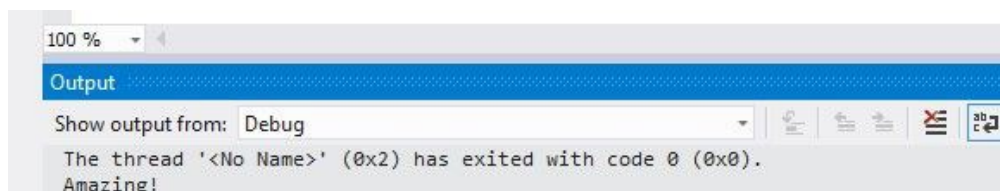


Console applications have their "Debug.Print" statements appear in Visual Studio's "Output" Window. The application in this example uses print statement; so, if the Output window is not visible, use the "VIEW->Output" menu item.

Press F5. Visual Studio will compile the program, load it into the emulator and run it. The emulator displays a hypothetical NETMF device, complete with buttons and a screen. Do not close it.



After a few seconds, the program will stop and the Output window will have "Amazing!" printed (surrounded by other application information).



7.3. Deploying to the G400

This section relies on the work done in the previous section as loading to the actual hardware device is exactly the same as loading to the emulator. The only difference is in selecting the device instead of the emulator. This is done by going to the project properties:

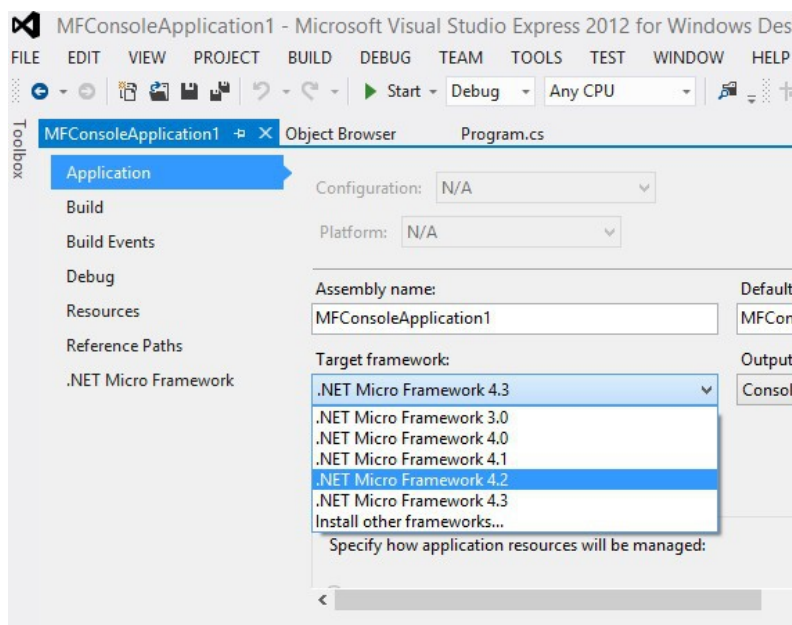
Connect the G400 to the PC using the USB interface (see Physical Bus Used During Programming, “the Debug Access Interface” section). To switch from the emulator to the hardware, select USB for “Transport.”

The first time a processor is connected to the computer, Windows may need to load a driver. Before proceeding, wait until the driver is fully loaded.

The F5 key will load the program to the G400 and then run it.

If necessary, the deployed program can use the full power of the Visual Studio debugger; including, stepping through lines, inspecting variables, setting breakpoints, etc. Programming and debugging the G400 is this easy!

7.4. 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. 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.

8. 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 from the program will look similar to this:

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

8.1. Finding NETMF Library Documentation

While this user manual is not meant to be a tutorial on the use of NETMF, a lot of details are provided to aid newcomers to NETMF. For further details, see the documentation library on the GHI website. Also, [the main support page for NETMF](#) includes links to the library reference documentation (NETMF APIs).

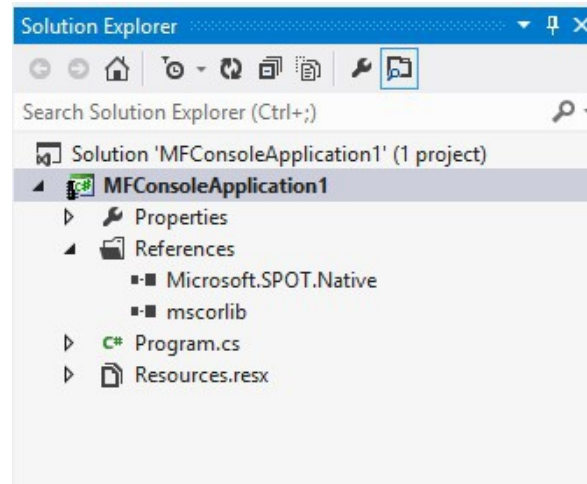
Because NETMF is a subset of the full .NET platform, services such as file input/output and Networking are very close, sometimes identical, between the full .NET Framework and the smaller .NET Micro Framework. The internet is a great source of .NET examples code that often can be used in a NETMF program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

8.2. 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.

8.3. Digital Inputs/Outputs

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. 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.

While it is clear that the earlier example blinks an LED every one second (on for 500ms and off for another 500ms), it is not clear what pin on the G400 will be controlled. Instead of using the generic GPIO_Pin0 name, the actual G400 name can be found in the GHI.Pins assembly. This example now uses an actual G400 pin name from 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(G400.PC20, true);
        while (true)
        {
            LED.Write(true);
            Thread.Sleep(500);
            LED.Write(false);
            Thread.Sleep(500);
        }
    }
}
```

Reading Input pins is also 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(Cpu.Pin.GPIO_Pin0, true);
        InputPort Button = new InputPort(Cpu.Pin.GPIO_Pin1, false, Port.ResistorMode.PullUp);
        while (true)
        {
            if (Button.Read() == true)
            {
                LED.Write(true);
                Thread.Sleep(500);
                LED.Write(false);
                Thread.Sleep(500);
            }
        }
    }
}
```

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 in a lower power state.

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

using GHI.Pins;

public class Program
{
    public static OutputPort LED = new OutputPort(Cpu.Pin.GPIO_Pin0, true);
    public static void Main()
    {
        InterruptPort Button = new InterruptPort(Cpu.Pin.GPIO_Pin1, 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);
    }
}
```

8.4. 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.

8.5. 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.

8.6. 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(Cpu.Pin.GPIO_Pin0, false);

        pin.Set(false, signal);

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

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

This is available through the GHI.Hardware assembly.

8.7. 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(Cpu.Pin.GPIO_Pin0,Port.ResistorMode.Disabled);

        pin.Read(false, signal);

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

This is available through the GHI.Hardware assembly.

8.8. Serial Port (UART)

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

Serial Port	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.

8.9. SPI

The G400 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: SPI1 is shared internally with the flash memory. Using a chip select is required with devices connected using SPI1. Improper use of SPI1 will cause G400 to not boot. The use of SPI2 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.

8.10. I2C

I2C is a two-wire addressable serial interface.

The G400 supports one master I2C port. Refer to the [Pin-Out Description](#) chapter for more information about I2C signals assignments to G400 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.

8.11. 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. There are two CAN channels on the G400.

```
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.

8.12. 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.

```
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(Cpu.Pin.GPIO_Pin0, 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.

8.13. Graphics

The G400 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 G400 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 G400's 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 similarly.

```
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);
            }
        }
    }
}
```

This requires the Microsoft.SPOT.Graphics assembly.

Fonts

Thanks to NETMF, developers can convert true type font files to the tiny font 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 . Glide includes a window designer as well <https://www.ghielectronics.com/glide>. Contained in the GHI.Glide assembly.

Touch Screen

The G400 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 (YU, YD, XL, XR) assignments to G400 hardware pins.

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

8.14. 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.

8.15. 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 G400. The GHI Electronics' online documentation has further examples as well. The only difference from a the full .NET on the PC would be in the need to mount the media and also in the media names. The easiest way to know and handle the media names is by obtaining the root directly name and dynamically using that name.

```
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();
    }
}
```

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.

SD/MMC Memory

SD and MMC memory cards have similar interfaces. The G400 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 G400.

USB Mass Storage

USB mass storage devices such as USB hard drives or memory sticks are directly supported on G400 through the USB Host library.

8.16. 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 networking is implemented for hard-wired Ethernet, WiFi, and via PPP over UART.

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

The Extensions

The way networking works on NETMF is very similar to the full desktop .NET. The networking libraries work on the Microsoft NETMF emulator. This allows for testing and developing right on the desktop. GHI Electronics adds few important extensions to the system to initialize the networking interfaces. For example, applications that do not use networking can still use the G400 as it defaults with networking services disabled. Programmatically, developers can choose to add Ethernet or WiFi. These additions are typically only needed in the initialization and setup stage and they cannot be used on the emulator. When using the emulator, the few initialization lines can be commented out.

MAC address setting

All G400s ship with the same default MAC address. This is good for testing a single device on internal networks (LANs). If using multiple devices, or networking over the internet, a proper MAC address must be set.

To set the MAC address, FEZ Config can be used. The MAC address can also be set in the application.

```
byte[] newMAC = new byte[] { 0x00, 0x1A, 0xF1, 0x01, 0x42, 0xDD };
var enc = new GHI.Networking.EthernetENC28J60(SPI.SPI_module.SPI1,
    Cpu.Pin.GPIO_Pin0, // chip select
    Cpu.Pin.GPIO_Pin1, // external interrupt
    Cpu.Pin.GPIO_Pin2 // reset
); //change to target design
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 G400 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.EthernetENC28J60(SPI.SPI_module.SPI1,
    Cpu.Pin.GPIO_Pin0, // chip select
    Cpu.Pin.GPIO_Pin1, // external interrupt
    Cpu.Pin.GPIO_Pin2 // reset
); //change to target design
enc.EnableDhcp();
enc.EnableDynamicDns();
```

This is available through the GHI.Networking assembly.

Ethernet

The support for Ethernet is available through the ENC28J60 SPI-ethernet chip.

This is available through the GHI.Networking assembly.

```
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 EthernetENC28J60 enc;
    static bool hasAddress = false;
    static bool available = false;

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

        var enc = new GHI.Networking.EthernetENC28J60(SPI.SPI_module.SPI1,
Cpu.Pin.GPIO_Pin0 , // chip select
Cpu.Pin.GPIO_Pin1 , // external interrupt
Cpu.Pin.GPIO_Pin2 // reset
); //change to target design
        enc.Open();
        enc.EnableStaticIP("192.168.1.100", "255.255.255.0", "192.168.1.0");
        enc.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 = enc.IPAddress != "0.0.0.0";
    }
}
```

Wireless LAN WiFi

Any WiFi module with built in TCP/IP stack can be used through the core NETMF libraries, this method has many limitations; GHI Electronics adds support to WiFi internally through 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
var wifi = new GHI.Networking.WiFiRS9110(SPI.SPI_module.SPI1,
Cpu.Pin.GPIO_Pin0, // chip select
Cpu.Pin.GPIO_Pin1, // external interrupt
Cpu.Pin.GPIO_Pin2 // reset
); //change to target design
//Open and configure
wifi.Join("ssid", "password");
//...
```

This is available through the GHI.Networking assembly.

8.17. 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 G400 will enjoy the use of these modems through PPP and the internal NETMF-TCP/IP stack with SSL.

The PPP feature is not currently available but being ported to the G400.

8.18. USB Client (Device)

The USB client interface is typically used as the G400's Debug Access Interface and for application deployment through Microsoft's Visual Studio. 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: USB Host is what devices connect to. This is like a PC where USB devices connect to. A USB device can be a mouse or a memory stick. The G400 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);
        }
    }
}
```

8.19. 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.

8.20. Real Time Clock

The G400 processor includes a real-time clock (RTC) that can operate while the processor is off, through a backup battery or a super capacitor. The G400 Real Time Clock will require a voltage source of between 1.8V ~ 3.6V, typically a button battery, to be connected to the VDDBU/VBAT pin.

NETMF 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());
    }
}
```

8.21. 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();
}
```

8.22. Power Control

The G400 automatically goes into a lower power mode whenever it is not actively executing code, for example, when waiting for an event. Additionally, it can be placed in hibernate mode. When hibernating, the G400 stops executing and draws much less power.

This feature is under development.

8.23. In-Field Update

The In-field update feature allows the G400 to update itself! This powerful feature adds considerable flexibility and is very simple to use. The managed application or even the TinyCLR NETMF firmware can be updated. When needed, this feature 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 a couple of seconds and then the system reboots, running the new software.

The online documentation includes further details.

8.24. 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 G400'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 origin 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.

9. Advanced Use Of The Microprocessor

The G400 is based on the Atmel SAM9X35 microcontroller. There are times when direct programming is needed. GHI has extended NETMF to allow assembly level access from managed code to the SAM9X35. 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.

9.1. Register

This class is used for manipulating the processor registers directly. For example here is a snippet of manipulating a control register of the Real Time Clock

```
// set UPDTIM & UPDCAL  
new Register(0xFFFFFEB0).SetBits(3);
```

9.2. AddressSpace

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

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

9.3. Runtime Loadable Procedure

Similar to code dynamically 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.

10. Design Consideration

10.1. Required Pins

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

- Serial COM1 (PA9, P10), USB Device (see the G400-S Pin-out Table) or both (see also: Physical Bus Used During Programming, “the Debug Access Interface”)
- MODE (PA25) can be set to high or low per the project design (high if left unconnected).
- LDR0 (PA24) LDR1 (PA4).

10.2. SPI Channels

SPI channel 1 is not available as GPIO and is internally shared with the G400's FLASH memory. Only SPI slave devices can be used on this SPI channel and only if the chip select pin is used on the connected slaves. Consider using the other SPI channel instead.

10.3. Watchdog

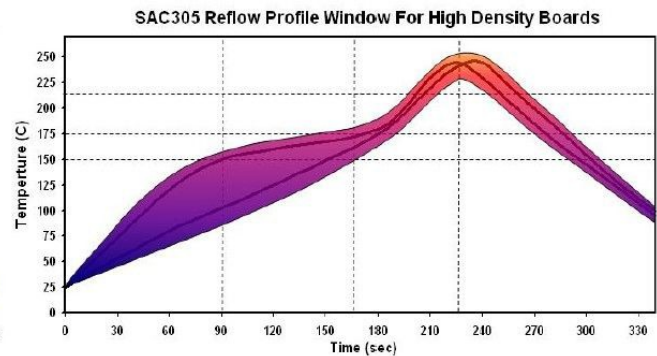
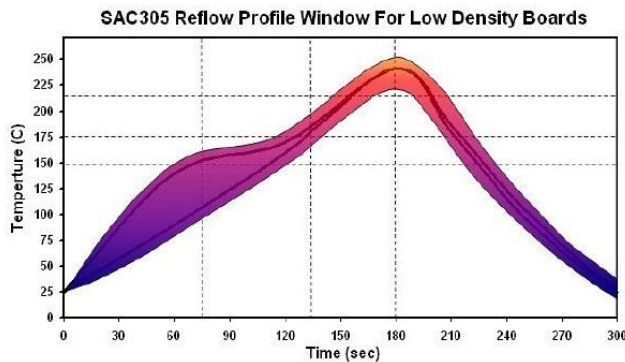
The NETMF LastResetCause flag will always return Normal; even when the watchdog was the cause of the board reset. This is due to the architecture of the SAM9X35.

11. Soldering The G400

The G400 was designed to be easily machine-placed and hand-soldered. Static sensitive precautions should take place when handling the modules.

The G400s are not sealed for moisture, it is recommended to bake the module before reflow. The process of reflow can damage the G400 if the temperature is too high or exposure is too long. This lead-free reflow is used by GHI Electronics when machine-placing the G400.

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.



<i>RATE OF RISE 2°C / SEC MAX</i>	<i>RAMP TO 150°C (302°F)</i>	<i>PROGRESS THROUGH 150°C-175°C (302°F-347°F)</i>	<i>TO PEAK TEMP 230°C- 245°C (445°F- 474°F)</i>	<i>TIME ABOVE 217°C (425°F)</i>	<i>COOLDOWN ≤ 4 °C / SEC</i>	<i>PROFILE LENGTH AMBIENT TO COOL DOWN</i>
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 , 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.

G400 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.