

Beginners Guide to Porting NETMF

January 22nd 2012

Rev 2.0



Copyright © 2012 GHI Electronics, LLC

www.GHIElectronics.com

Community: www.TinyCLR.com

Gus Issa

Table of Contents

1.About the Book.....	3	6.The PK Build System.....	12
1.1.Intended Audience.....	3	6.1.Output Folder.....	14
1.2.Disclaimer.....	3	7.FEZ Hacker Firmware.....	16
2.Introduction.....	4	7.1.Building Steps.....	16
3.Supported Processors.....	5	Build TinyBooterDecompressor.....	16
3.1.Processor Challenges.....	5	7.2.SAM-BA.....	18
Bootstrap.....	5	7.3.MFDeploy.....	22
Essential Peripheral.....	5	8.Memory Layout.....	25
4.DIY NETMF Hardware.....	6	9.Interops and RLP.....	27
Find a suitable Processor/Board.....	6	9.1.Interops.....	27
Select a PCB Design Software.....	6	9.2.RLP.....	29
Finding Component Packages.....	6	10.Open Source Hardware.....	30
PCB Layout.....	6	11.Porting vs GHI's Offers.....	31
Ordering the board.....	7	Features.....	31
Soldering.....	8	Support.....	31
Loading Software.....	8	Maintenance.....	31
4.1.Taking DIY to the Next Level.....	8	Robustness.....	31
5.Software Setup.....	10	Time-to-Market.....	32
5.1.Visual Studio Express.....	10	12.Additional Resources.....	33
5.2.Microsoft NETMF SDK.....	10	Tutorials and Downloads.....	33
5.3.GHI NETMF package installer.....	10	Codeshare.....	33
5.4.Microsoft NETMF Porting Kit.....	10	eBooks.....	33
5.5.GNU GCC Compiler.....	11		

1. About the Book

1.1. Intended Audience

This book is for beginners wanting to understand the .NET Micro Framework porting process. Readers are expected to have good knowledge in C#, C++ and command line compilation. The book doesn't cover the details but it explains how porting NETMF can be simple in some cases and how many low-cost GHI offers can be utilized in this learning process.

1.2. Disclaimer

This is a free book use it for your own knowledge and at your own risk. Neither the writer nor GHI Electronics is responsible for any damage or loss caused by this free eBook or by any information supplied by it. There is no guarantee any information in this book is valid.

2. Introduction

Microsoft offers a technology that, in my opinion, is the best thing that has ever happened to the world of embedded system. This technology is called .NET Micro Framework, NETMF for short. NETMF is not an OS, and at the same time, is not an application. It is a system that sits in between your high-level C# code and low-level C/C++/Assembly. At the end, programmers can develop and debug code with ease through C# (and Visual Basic) and Visual Studio. All the professional features available in Visual Studio are now available to you on very small systems, such as stepping in code and variable inspection.

For programing libraries, NETMF offers a subset of the full .NET Framework. Any existing knowledge with .NET can be leveraged to any system running NETMF. Developers that do not have experience with .NET can still take advantage of any books or online-example-code that is targeting .NET, due to the deep similarities between the full .NET and NETMF.

3. Supported Processors

ARM is the most common processor used with NETMF but other processors can be used. The challenge in using a processor core beside what is already supported in the porting kit (PK for short) is that the you will need to change the build system to support the compiler being used and some of the core functionality needs to be changed as well. I estimate the process of doing so to be about 3 months of work by a PK professional. This book doesn't cover changes to the build system.

If you decided on an architecture that is already supported by the PK, it is time to select a processor. For example, if you selected ARM then there are hundreds of chips to chose from. Even though companies, such as Atmel, NXP and ST, share the same ARM core, the peripherals used are different. The simplest route would be selecting a chip that is already in the PK as the drivers for most (probably not all) peripherals are already available.

In short, if you are new to NETMF PK, start with a processor that is already supported. Even better, start with one of GHI's open-source offers. They are low cost and proven to work by a large community of NETMF fans.

3.1. Processor Challenges

To boot a processor with NETMF, there are couple challengers, Bootstrap and essential peripherals. This also applies to booting other operating systems.

Bootstrap

Most programmers are under the impression that “main” is where execution starts. This is not entirely correct. There are many routines that runs first to initialize the processor, memories and runtime system before “main” is executed. This stage is usually called bootstrapping, which is very processor dependent and very operating-system independent.

The first thing to happen is initializing the processor with a proper clock. Then external memories, if any, need to be initialized. Finally, the stacks and heap need to be initialized.

Essential Peripheral

Any operating system requires at least two things to run bare minimal, timers and interrupts. The timers are needed to keep track of task-time and to invoke timed system-management routines, which also requires interrupts. Out of the two, developing drivers for interrupts will be a lot more challenging than timers.

Assuming we have all above, how do we “see” what is the system doing? I recommend a serial port since NETMF send many useful messages on power up.

4. DIY NETMF Hardware

Although working with porting NETMF can be a little bit complex, you can create your own NETMF device very easily and almost for free (you still need to buy the raw components). Just use one of the processors in the porting kit.

I wanted to test this myself so I started from scratch just to see how long it would take to create my own device, software and hardware. I was actually able to do it in one weekend.

These are the steps I took to create this device, which I call FEZ Hacker:

Find a suitable Processor/Board

I want to make a simple device, no external memories. Currently, the simplest option is SAM7_EK, which is a port for AT91SAM7X-EK Evaluation Board from Atmel.

Here is the User Guide Which also includes full schematics:

<http://www.ghielectronics.com/downloads/FEZ/FEZ%20Hacker/AT91SAM7X-EK.pdf>

Select a PCB Design Software

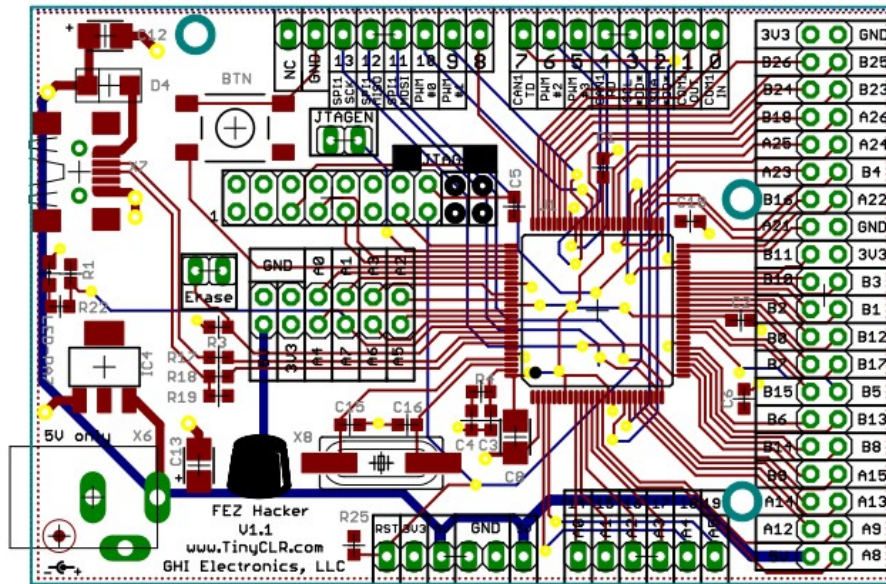
I only need to copy the schematics to create my own board. I will be using the free version of EAGLE, found at <http://www.cadsoft.de/> for schematic capture and PCB layout.

Finding Component Packages

Do not reinvent the wheel. If you look enough, you will find packages for everything you need to create this board, FEZ Hacker. For example, the main component, which is the processor, is already made by someone and it is found on the EAGLE website. Just click on “Libraries” on this page <http://www.cadsoft.de/download.htm>. All other components like resistors and capacitors ship with EAGLE. Do not worry about finding all that, I am giving you the design files.

PCB Layout

Now, what do we want our circuit board to look like? I would say make it Arduino pinout compatible so you can use the available shields. Since this processor has a lot more IOs than what is on the Arduino, we will extend the board slightly on one side and add another header.



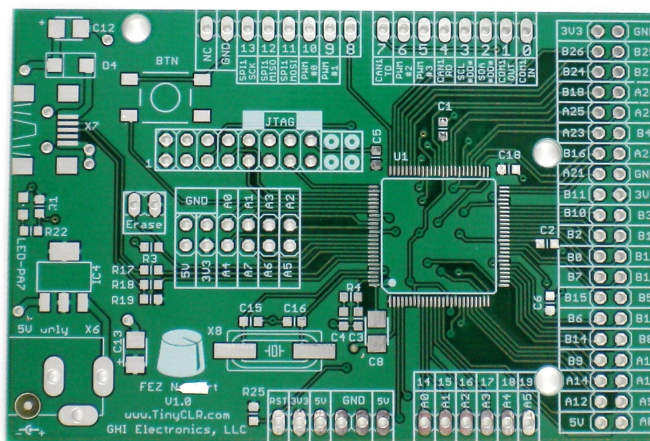
You can see that I even added a JTAG to the board. JTAG is not needed for NETMF usage but it is handy since I can use this board for anything else.

The design files are found at: <http://www.ghielectronics.com/downloads/FEZ/FEZ%20Hacker/FEZ%20Hacker%201.1.zip>

Ordering the board

When running all traces and running some verification scripts, we are ready to order the PCB. There are too many websites that would love to make these boards for you. I used www.my4pcb.com

A few days later I received this board.



Soldering

You can send the board to places like <http://www.screamingcircuits.com/> or you can have more fun and assemble the board yourself. I have a few videos on YouTube on how to solder if you are interested in watching them

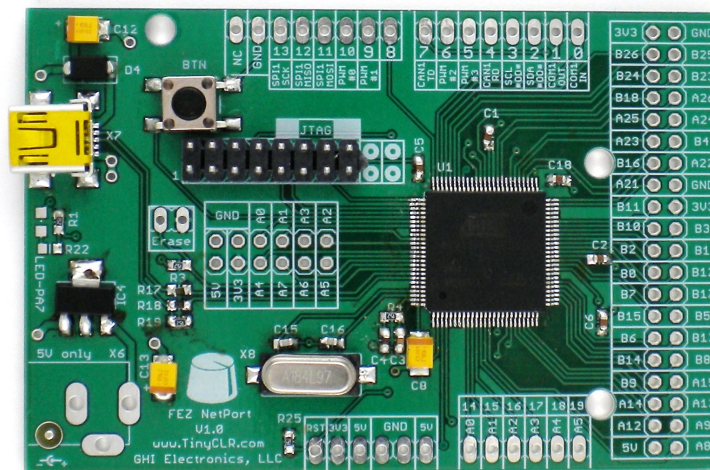
1 of 4: <http://www.youtube.com/watch?v=oUk8uqi7h-A>

2 of 4: <http://www.youtube.com/watch?v=UAAcVVr8sr8>

3 of 4: <http://www.youtube.com/watch?v=9egUh13RHYM>

4 of 4: <http://www.youtube.com/watch?v=wmp5aIWdZLQ>

After some soldering, we get this beautiful FEZ Hacker. Feel free to make your own but please add your name and change the GHI name to “Based on FEZ Hacker”.

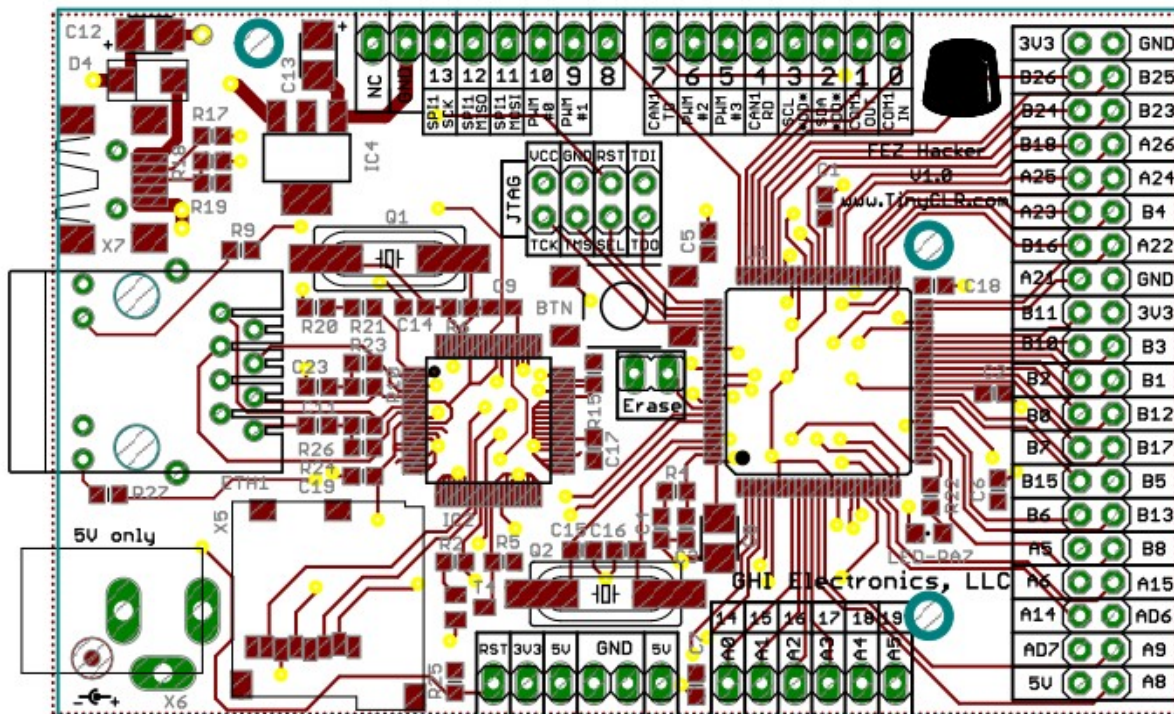


Loading Software

Chapter **FEZ Hacker Firmware** details this step. You need to install a few software components first.

4.1. Taking DIY to the Next Level

We saw how we easily copied schematics to create our own Arduino-compatible NETMF board. Now what about making it even better? For example, take the Ethernet shield design and merge that right into your board. You will end up with on-board Ethernet.



You can find the complete design files at

<http://www.ghielectronics.com/downloads/FEZ/FEZ%20Hacker/FEZ%20Hacker%20Ethernet%201.0.zip>

So why did I use WIZnet W5100 instead of using the NETMF built in LWIP stack? Simply, if you use LWIP, you will end up with very little resources (or no resources) that the system becomes unusable. If you want to use LWIP then you need more RAM/FLASH than this particular chip can offer.

5. Software Setup

Before we do anything, we want to make sure our system can compile one of the example ports with no modifications. In the following steps, we will install all the needed software and compile one of the built-in ports. I will assume the users own no software so we will use free software throughout the book. After all, NETMF is free.

5.1. Visual Studio Express

You will need C# and C++, so install both. Here is where you can find the downloads <http://www.microsoft.com/express/>

Once Visual Studio is installed, try to make a simple windows application, compile and debug...etc. We basically need to make sure all is good before we move on to the next step.

5.2. Microsoft NETMF SDK

The next step is to install the Microsoft NETMF SDK, not the porting kit, just yet. At the time this book was made, the latest version was NETMF 4.1. The download is available at this link

<http://www.microsoft.com/downloads/details.aspx?familyid=CFF5A7B7-C21C-4127-AC65-5516384DA3A0&displaylang=en>

When done installing, run a “Hello World” application using the emulator. If your not sure how, use this book:

<http://www.ghielectronics.com/downloads/FEZ/Beginners%20guide%20to%20NETMF.pdf>

5.3. GHI NETMF package installer

This is an optional download but highly recommended.

<http://www.tinyclr.com/support>

5.4. Microsoft NETMF Porting Kit

This Porting Kit includes all NETMF core source code. At the time this book was made, the latest version was NETMF 4.1. The download is available at this link

<http://www.microsoft.com/downloads/details.aspx?FamilyID=ccdd5eac-04b1-4ecb-bad9-3ac78fb0452b&displaylang=en>

5.5. GNU GCC Compiler

Fortunately, the NETMF team added support to the free GCC compiler.

NETMF is tested to work with GCC package made available by

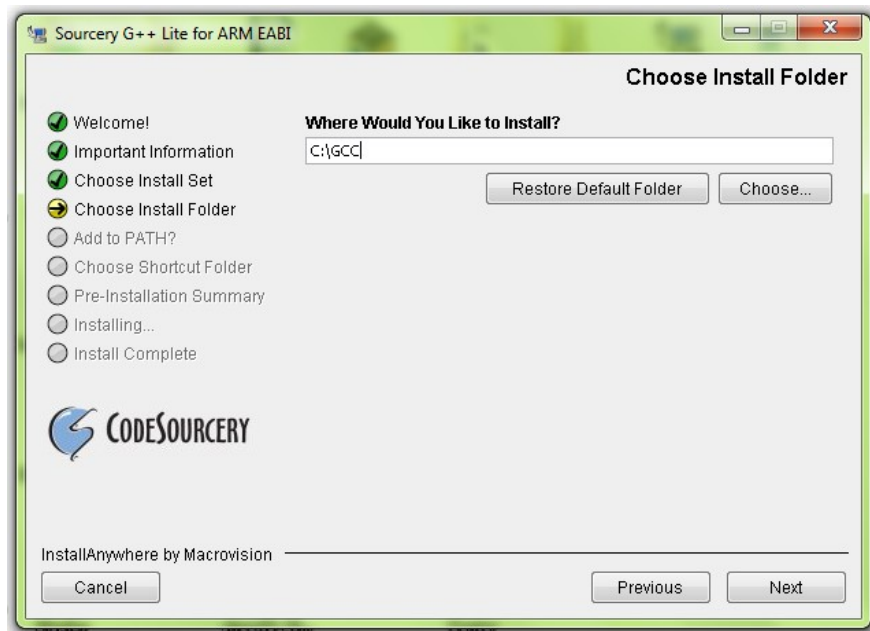
<http://www.codesourcery.com/>

Now, you can't just use the latest version you can find as changes to GCC may cause the NETMF build to break. You will need to use a specific version. For NETMF 4.1, I use arm-2007q3-53-arm-none-eabi.exe found at this link

<http://www.codesourcery.com/sgpp/lite/arm/portal/release316>

You will need to download the windows package, **do not install just yet!**

The NETMF PK build system has a problem compiling GCC builds if the GCC path has spaces. I highly recommend the GCC tool-chain to be installed in the root of your hard drive with the simplest possible name. I simply used "C:\GCC" for the installation location.



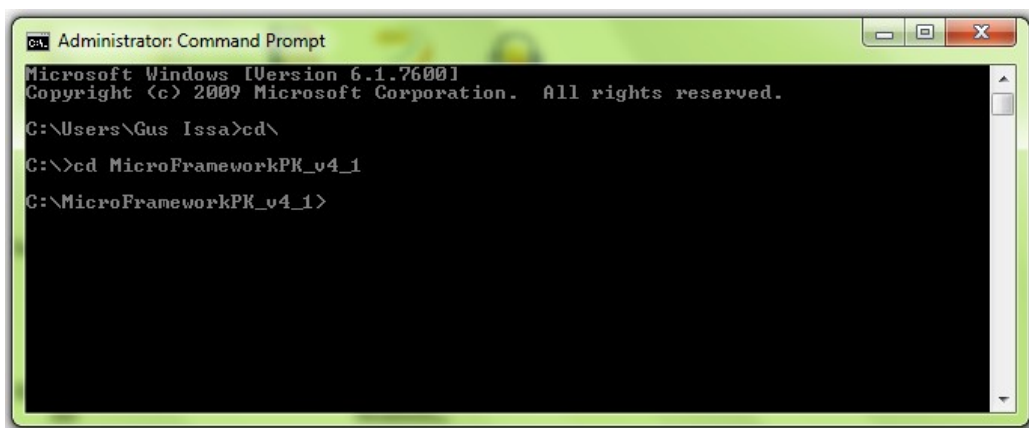
We are now ready to install the GCC tools. Keep all options to the default except for the install location, as shown above.

6. The PK Build System

If you are used to building simple projects using IDEs then you will not be very happy to know that your work with the NETMF source code will be done through command-line. With that said, the build system is really impressive. By running one command, the build system builds some tools (MetadataProcessor) that is used within the build itself!

Before we change a single line of code, we need to make sure we can build one of the ports that ship with the porting kit. I always select “iMXS” as my first test. This is the port that started NETMF and is most used by NETMF core developers. This port is also available in other forms to demonstrate other features such as “iMXS_thumb” which shows how to build using THUMB instructions instead of ARM to reduce firmware footprint.

Start by opening the command prompt, then access the PK folder



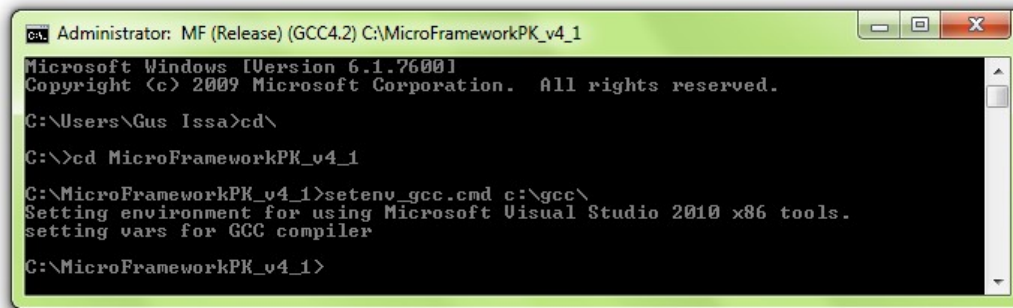
```
Administrator: Command Prompt
Microsoft Windows [Version 6.1.7600]
Copyright (c) 2009 Microsoft Corporation. All rights reserved.

C:\Users\Gus Issa>cd\
C:\>cd MicroFrameworkPK_v4_1
C:\MicroFrameworkPK_v4_1>
```

We now set the environment to your compiler.

```
setenv_gcc.cmd c:\gcc\
```

Note that I am assuming you did install the GCC compiler at c:\gcc as I suggested.



```

Administrator: MF (Release) (GCC4.2) C:\MicroFrameworkPK_v4_1
Microsoft Windows [Version 6.1.7600]
Copyright (c) 2009 Microsoft Corporation. All rights reserved.

C:\Users\Gus Issa>cd\

C:\>cd MicroFrameworkPK_v4_1

C:\MicroFrameworkPK_v4_1>setenv gcc.cmd c:\gcc\
Setting environment for using Microsoft Visual Studio 2010 x86 tools.
setting vars for GCC compiler

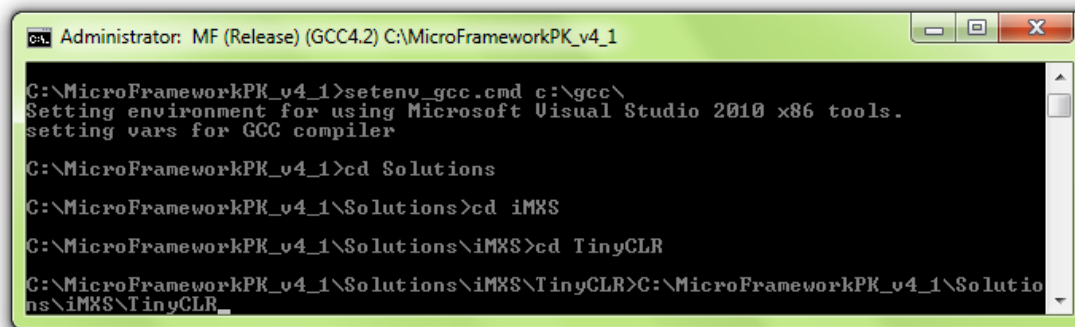
C:\MicroFrameworkPK_v4_1>

```

The next step is to build the iMXS solution to make sure our system is all set. At this point, you still haven't made any modifications to any file in the porting kit. This is important to track down any possible errors in the system setup.

Access the C:\MicroFrameworkPK_v4_1\Solutions\iMXS\TinyCLR folder and run this command

MSBUILD.EXE /t:build /p:flavor=release;memory=flash



```

Administrator: MF (Release) (GCC4.2) C:\MicroFrameworkPK_v4_1

C:\MicroFrameworkPK_v4_1>setenv gcc.cmd c:\gcc\
Setting environment for using Microsoft Visual Studio 2010 x86 tools.
setting vars for GCC compiler

C:\MicroFrameworkPK_v4_1>cd Solutions

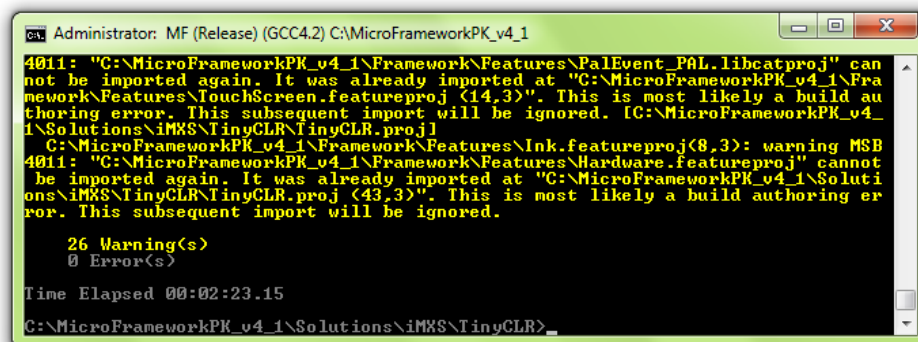
C:\MicroFrameworkPK_v4_1\Solutions>cd iMXS

C:\MicroFrameworkPK_v4_1\Solutions\iMXS>cd TinyCLR

C:\MicroFrameworkPK_v4_1\Solutions\iMXS\TinyCLR>C:\MicroFrameworkPK_v4_1\Solutions\iMXS\TinyCLR_

```

You should see a lot of things happening in the prompt window and then you will have no errors at the end. This can take a while the first time you run this build.



```

Administrator: MF (Release) (GCC4.2) C:\MicroFrameworkPK_v4_1

4011: "C:\MicroFrameworkPK_v4_1\Framework\Features\PALEvent_PAL.libcatproj" can
not be imported again. It was already imported at "C:\MicroFrameworkPK_v4_1\Fra
mework\Features\TouchScreen.featureproj (4.3)". This is most likely a build au
thoring error. This subsequent import will be ignored. [C:\MicroFrameworkPK_v4_
1\Solutions\iMXS\TinyCLR\TinyCLR.proj]
C:\MicroFrameworkPK_v4_1\Framework\Features\Ink.featureproj(8,3): warning MSB
4011: "C:\MicroFrameworkPK_v4_1\Framework\Features\Hardware.featureproj" cannot
be imported again. It was already imported at "C:\MicroFrameworkPK_v4_1\Soluti
ons\iMXS\TinyCLR\TinyCLR.proj (4.3)". This is most likely a build authoring er
ror. This subsequent import will be ignored.

26 Warning(s)
0 Error(s)

Time Elapsed 00:02:23.15

C:\MicroFrameworkPK_v4_1\Solutions\iMXS\TinyCLR>

```

Should you need to a complete rebuild, start by using this clean command .

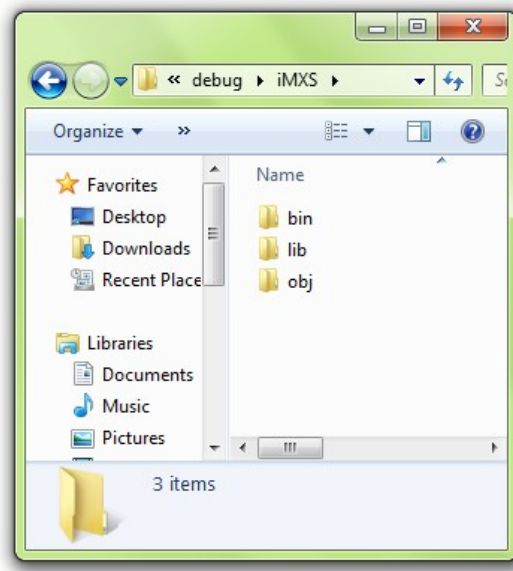
MSBUILD.EXE /t:clean /p:flavor=debug;memory=flash

6.1. Output Folder

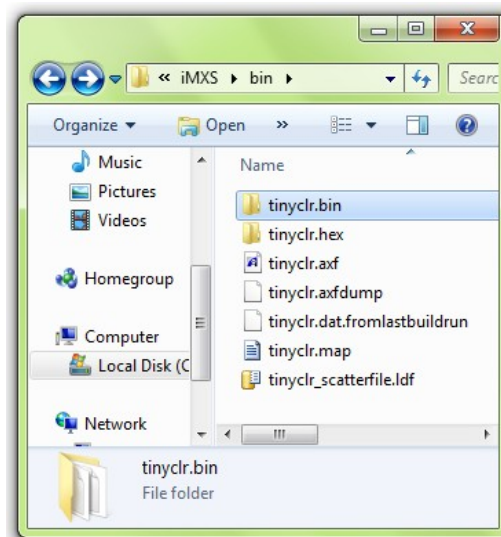
Each solution in the PK is built in a separate folder. For our example, the output folder is found at

C:\MicroFrameworkPK_v4_1\BuildOutput\ARM\GCC4.2\le\FLASH\debug\iMXS

You will find 3 folders, bin, lib and obj



The one we are most interested in is the bin folder



In the folder, `tinyclr.axf` is an ELF file that contains all of the needed debugging info. This is the file you need if you were to use JTAG.

`tinyclr.map` lists where every method lives in memory. You will rarely use this file but when you need it you will be very happy to know it is there for you.

The `tinyclr.scatterfile.ldf` is a very important file when laying out memory. This is the file used by the linker to place an object in the appropriate location.

Finally, and most importantly is the firmware. The firmware is generated in two forms, raw binary image and s-record file. Ideally, you will use the s-record file, which is in `tinyclr.hex` folder, with a chip programmer. Note that this solution generates three output files. All these file are part of the firmware.

7. FEZ Hacker Firmware

If you are using the DIY FEZ Hacker board then you are using the SAM7_EK port. All you need to do is compile it and load it, which should be simple.

Start by compiling the port as we did in the last chapter. Last chapter only compiled TinyCLR but we also want to compile TinyBooter for this port. The is NETMF's boot loader.

7.1. Building Steps

Build TinyBooterDecompressor

MSBUILD TinyBooterDecompressor.proj /t:build /p:flavor=release;memory=flash

If the build completed with an error saying something about "CreateSymdef" then copy CreateSymdef.exe file from

C:\MicroFrameworkPK_v4_1\tools\bin to

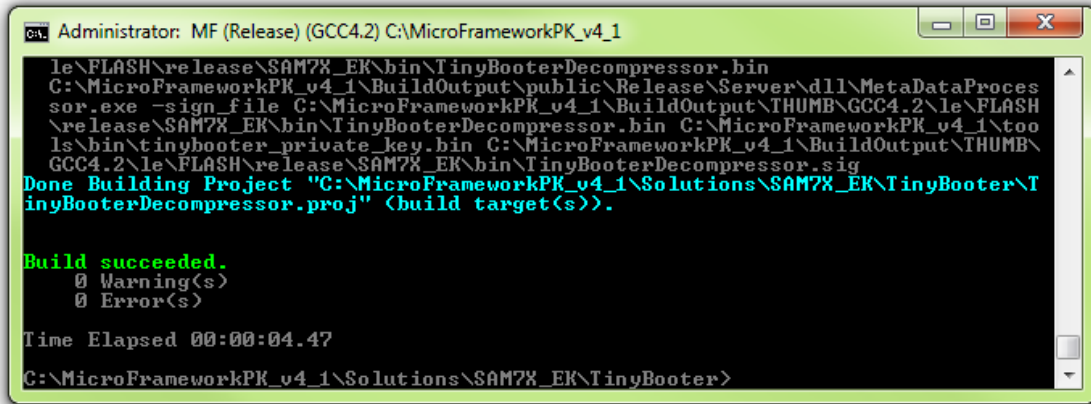
C:\MicroFrameworkPK_v4_1\BuildOutput\public\Release\Server\ddl

This is a sample of the error message

```
Administrator: MF (Release) (GCC4.2) C:\MicroFrameworkPK_v4_1
C:\MicroFrameworkPK_v4_1\BuildOutput\THUMB\GCC4.2\le\FLASH\release\SAM7X_EK\bin\TinyBooter.axf >> C:\MicroFrameworkPK_v4_1\BuildOutput\THUMB\GCC4.2\le\FLASH\release\SAM7X_EK\bin\TinyBooter.dump
C:\MicroFrameworkPK_v4_1\BuildOutput\public\Release\Server\ddl\CreateSymdef C:\MicroFrameworkPK_v4_1\BuildOutput\THUMB\GCC4.2\le\FLASH\release\SAM7X_EK\bin\TinyBooter.dump C:\MicroFrameworkPK_v4_1\BuildOutput\THUMB\GCC4.2\le\FLASH\release\SAM7X_EK\bin\TinyBooter.symdefs
'C:\MicroFrameworkPK_v4_1\BuildOutput\public\Release\Server\ddl\CreateSymdef'
is not recognized as an internal or external command,
operable program or batch file.
C:\MicroFrameworkPK_v4_1\tools\targets\Microsoft.SPOT.System.GCC.targets(378,5)
: error MSB3073: The command "C:\MicroFrameworkPK_v4_1\BuildOutput\public\Release\Server\ddl\CreateSymdef C:\MicroFrameworkPK_v4_1\BuildOutput\THUMB\GCC4.2\le\FLASH\release\SAM7X_EK\bin\TinyBooter.dump C:\MicroFrameworkPK_v4_1\BuildOutput\THUMB\GCC4.2\le\FLASH\release\SAM7X_EK\bin\TinyBooter.symdefs" exited with code 9009. [C:\MicroFrameworkPK_v4_1\Solutions\SAM7X_EK\TinyBooter\TinyBooterDecompressor.proj]
Done Building Project "C:\MicroFrameworkPK_v4_1\Solutions\SAM7X_EK\TinyBooter\TinyBooterDecompressor.proj" (build target(s)) -- FAILED.
Done Building Project "C:\MicroFrameworkPK_v4_1\Solutions\SAM7X_EK\TinyBooter\TinyBooterDecompressor.proj" (build target(s)) -- FAILED.
Build FAILED.
"C:\MicroFrameworkPK_v4_1\Solutions\SAM7X_EK\TinyBooter\TinyBooterDecompressor.proj" (build target) (1) ->
"C:\MicroFrameworkPK_v4_1\Solutions\SAM7X_EK\TinyBooter\TinyBooterDecompressor.proj" (build target) (672) ->
(CompressBin target) ->
C:\MicroFrameworkPK_v4_1\tools\targets\Microsoft.SPOT.System.GCC.targets(378,5): error MSB3073: The command "C:\MicroFrameworkPK_v4_1\BuildOutput\public\Release\Server\ddl\CreateSymdef C:\MicroFrameworkPK_v4_1\BuildOutput\THUMB\GCC4.2\le\FLASH\release\SAM7X_EK\bin\TinyBooter.dump C:\MicroFrameworkPK_v4_1\BuildOutput\THUMB\GCC4.2\le\FLASH\release\SAM7X_EK\bin\TinyBooter.symdefs" exited with code 9009. [C:\MicroFrameworkPK_v4_1\Solutions\SAM7X_EK\TinyBooter\TinyBooterDecompressor.proj]
0 Warning(s)
1 Error(s)
Time Elapsed 00:00:02.98
C:\MicroFrameworkPK_v4_1\Solutions\SAM7X_EK\TinyBooter>
```

Then try to build again if needed and make sure you have no errors at the end.

Here is the build completed with no errors.



```
Administrator: MF (Release) (GCC4.2) C:\MicroFrameworkPK_v4_1
le\FLASH\release\SAM7X_EK\bin\TinyBooterDecompressor.bin
C:\MicroFrameworkPK_v4_1\BuildOutput\public\Release\Server\dl1\MetaDataProces
sor.exe -sign_file C:\MicroFrameworkPK_v4_1\BuildOutput\THUMB\GCC4.2\le\FLASH
\release\SAM7X_EK\bin\TinyBooterDecompressor.bin C:\MicroFrameworkPK_v4_1\too
ls\bin\tinybooter_private_key.bin C:\MicroFrameworkPK_v4_1\BuildOutput\THUMB\
GCC4.2\le\FLASH\release\SAM7X_EK\bin\TinyBooterDecompressor.sig
Done Building Project "C:\MicroFrameworkPK_v4_1\Solutions\SAM7X_EK\TinyBooter\T
inyBooterDecompressor.proj" (build target(s)).

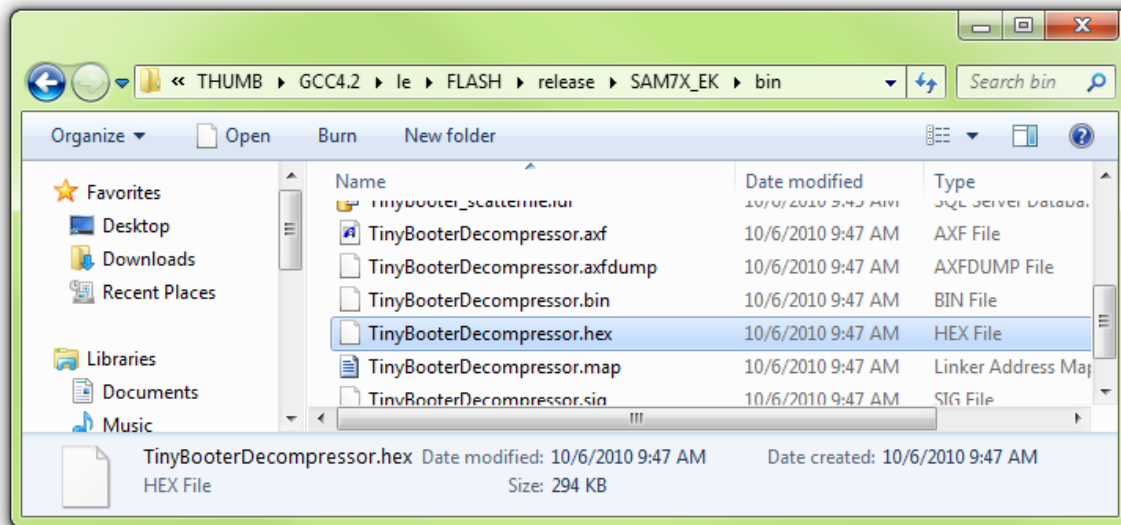
Build succeeded.
    0 Warning(s)
    0 Error(s)

Time Elapsed 00:00:04.47
C:\MicroFrameworkPK_v4_1\Solutions\SAM7X_EK\TinyBooter>
```

The output folder is located at

C:\MicroFrameworkPK_v4_1\BuildOutput\THUMB\GCC4.2\le\FLASH\release\SAM7X_EK\bin

It should have plenty of files but you are specifically interested in
TinyBooterDecompressor.hex and TinyBooterDecompressor.bin



You are now ready to load the firmware

7.2. SAM-BA

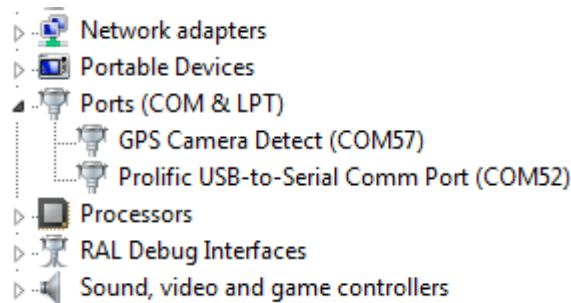
SAM-BA is a free software from Atmel that is used to load firmware on Atmel ARM chips using USB. Download and install SAM-BA from

http://www.atmel.com/dyn/resources/prod_documents/sam-ba_2.10.exe

If the link didn't work then search Atmel website for SAM-BA

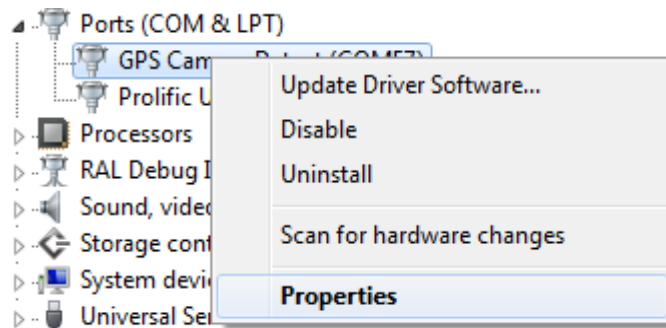
Note: If you are not sure if your chip is erased, short the “Erase” jumper and power the board for 1 second then disconnect power and remove the jumper.

Now, plug in your FEZ Hacker to the PC, windows will ask for drivers. Windows will probably see it as “GPS Camera Detect”. Do not worry about the name and take a note of the com port number, mine is COM57

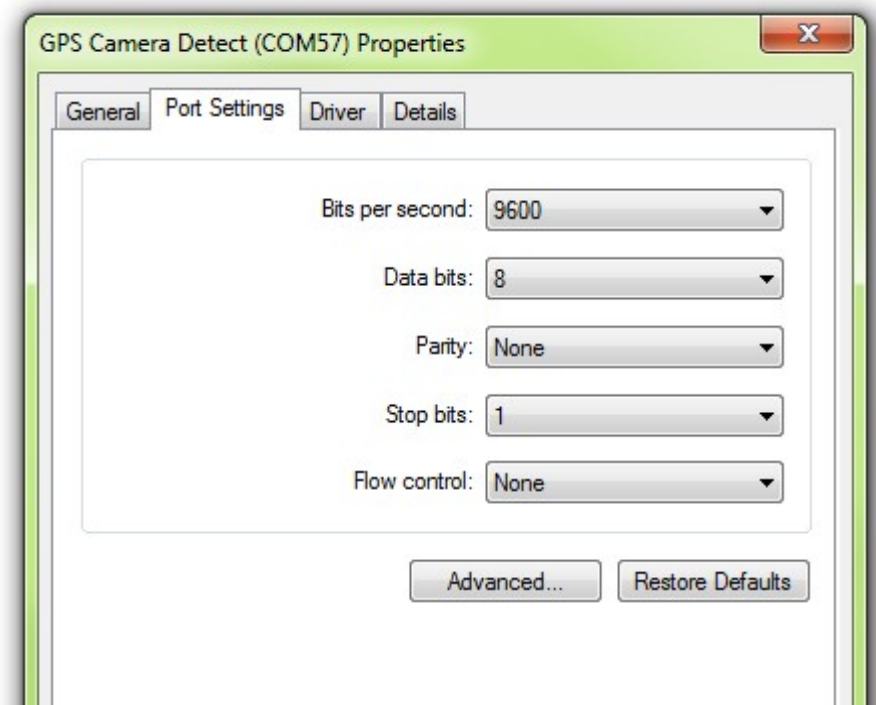


If windows was not able to locate the drivers on its own, direct windows to ChipworkX loader drivers in the GHI SDK found at C:\Program Files (x86)\GHI Electronics\GHI NETMF v4.1 SDK\ChipworkX\Firmware\TinyBooter Updater\USB Tinybooter Updater driver

Loading the driver will create a new virtual serial port on windows. You can now start SAM-BA and select the appropriate board and COM port. Note that running SAM-BA can take minutes, thanks Atmel!! Another problem is that this software doesn't seem to work with high COM port numbers so we need to change our COM port number to something lower than 9. You can use any COM as long as it is not used already. I do not have any under 9 in my list so I will just use COM4. You can change the COM number as follows

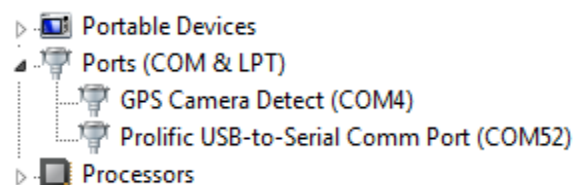


Right click the device and select “Properties” then from the “Port Settings” click “Advanced...”

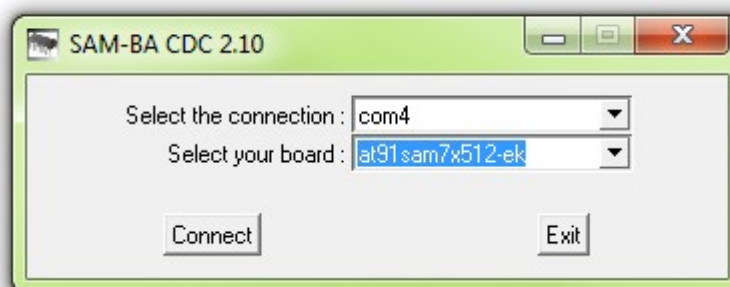


Although windows telling me it is "in use", I know it is not connected right now so I will go ahead and use COM4. Click OK and then disconnect and reconnect your FEZ Hacker board.

My port is now COM4

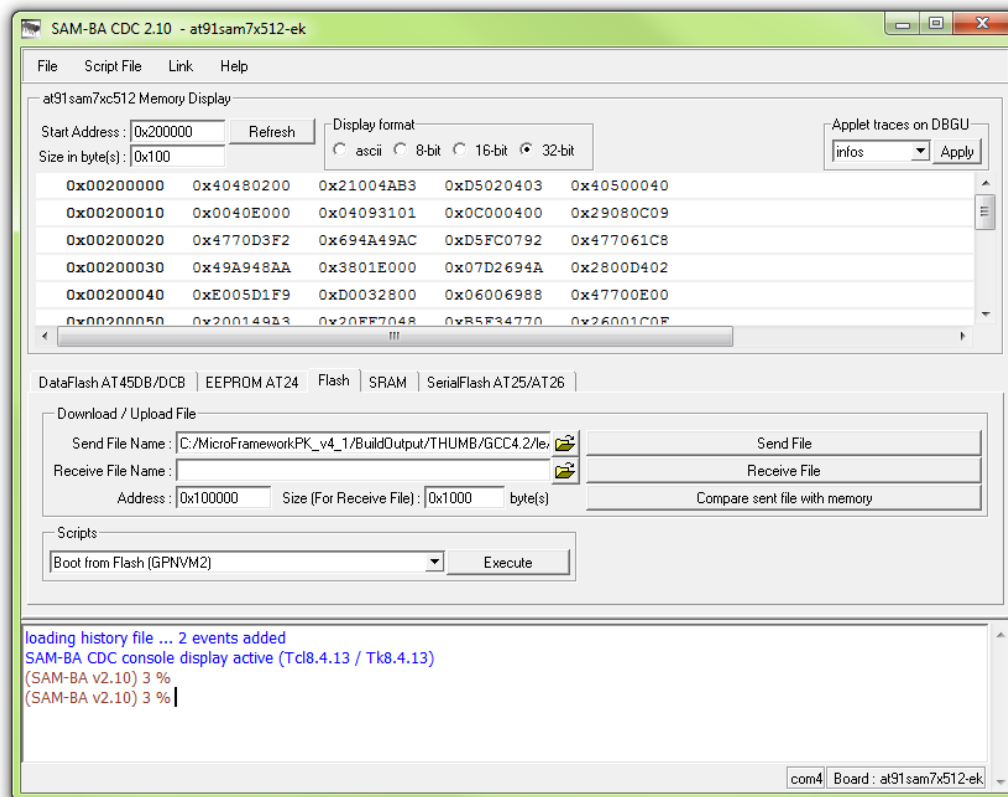


We can now run SAM-BA software. If you had it open then you will have to close it and reopen it.



Remember how our board is compatible with the AT91SAM7X512-EK board? We basically just copied the schematics. So, we will use that option from the menu. Also select the appropriate COM port, mine was forced to COM4 in previous steps.

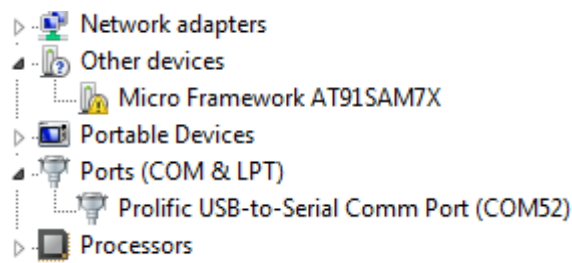
Click connect and you will have a new dialog. From the “Flash” menu, direct SAM-BA to TinyBooterDecompressor.bin file and click “SendFile”



Now that TinyBooter is loaded, we can run the script “Boot from Flash”, hit Execute.



When done, close SAM-BA and reset the board. This time windows will see the device as a NETMF device and will start looking for drivers. Windows will fail to locate drivers at the end since you do not have drivers just yet.



GHI has its own USB VID (Vendor ID) and that's what it uses for its devices. The SAM port you built uses some random VID but it is not GHI's. To make the GHI drivers work, you have 2 options. Either change the VID in source code to the GHI's VID or change the VID in the GHI USB drivers to match what is used in the SAM port. The problem with the second option is that you will lose the digital signing if you make any changes on the USB drivers. So, we will change the VID in the SAM port source code to use GHI's.

Open "usb_config.cpp" file located at
 C:\MicroFrameworkPK_v4_1\Solutions\SAM7X_EK\DeviceCode\USB

Find the VID and PID

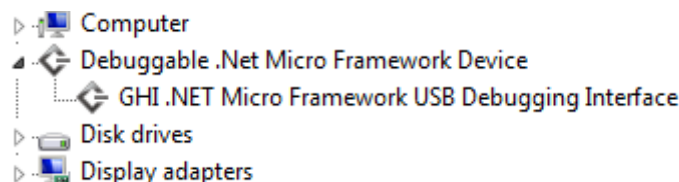
```
// device descriptor
#define PRODUCT_ID      0x6150
#define VENDOR_ID      0x03E8
```

and change them to same ones used by GHI, VID 1B9F and PID 0102

```
// device descriptor
#define PRODUCT_ID      0x0102// GHI's NETMF PID
#define VENDOR_ID      0x1B9F // GHI's VID
```

We can now rebuild and deploy just like we did before, just remember to erase FEZ Hacker first using the "Erase" jumper.

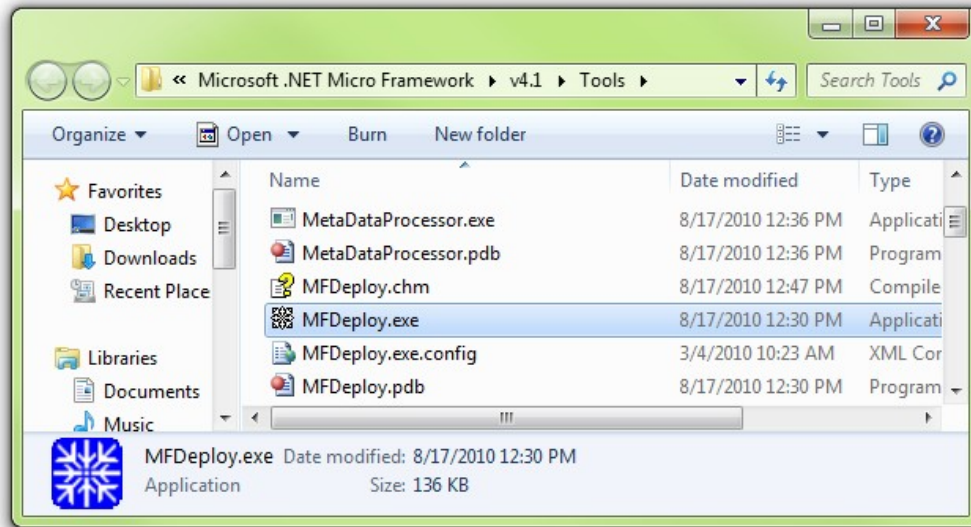
When all is done, you will see this in your "Device Manager"



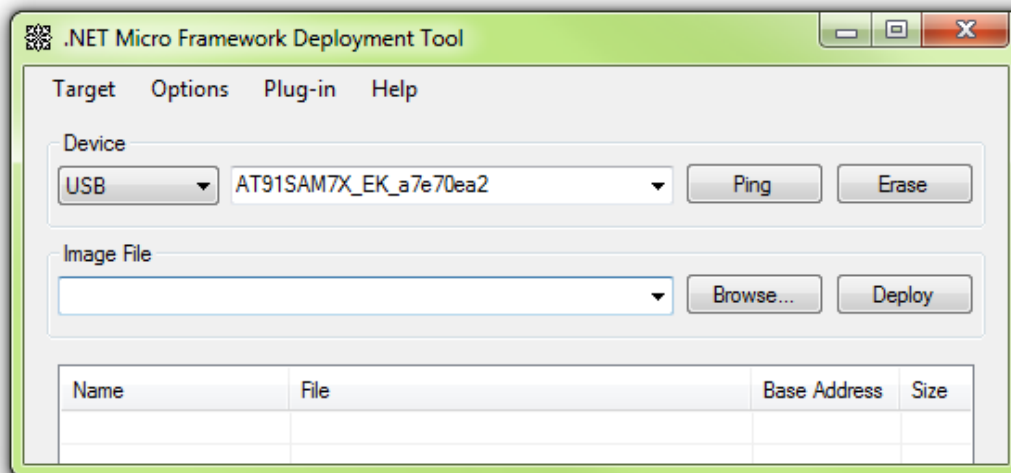
We have now a device that is booted with TinyBooter. I told you NETMF doesn't have to be difficult.

7.3. MFDeploy

MFDeploy has many uses in NETMF. We will use it to load the firmware for now. Locate and run MFDeploy



Select USB and you will see AT91SAM7X_EK showing up in the the device list.



Click "Ping" and the device should respond

```
Pinging... TinyBooter
Bootloader build info: Microsoft Copyright (C) Microsoft
Corporation. All rig
```


We are almost done, just build TinyCLR and load it using MFDeploy.

The build step is simple but first we have to make a little modification.

Locate C:\MicroFrameworkPK_v4_1\Solutions\SAM7X_EK\TinyCLR folder and open TinyCLR.proj

This is an XML file so you can open it with Visual Studio, just drag and drop it in it.

Now find this line and delete it

```
<MultipleOutputSections Condition="'$(MEMORY)'=='FLASH'">true</MultipleOutputSections>
```

Save and close TinyCLR.proj file

Finally, access TinyCLR folder and run

```
C:\MicroFrameworkPK_v4_1\Solutions\SAM7X_EK>
```

```
MSBUILD /t:build /p:flavor=release;memory=flash
```

You may have some warning at the end but need to make sure that we have no errors.

```
Administrator: MF (Release) (GCC4.2) C:\MicroFrameworkPK_v4_1
CLR.proj" <build target(s)>.
Build succeeded.

C:\MicroFrameworkPK_v4_1\Framework\Features\Hardware.featureproj(5,3): warning
MSB4011: "C:\MicroFrameworkPK_v4_1\Framework\Features\Hardware.featureproj" cannot
be imported again. It was already imported at "C:\MicroFrameworkPK_v4_1\Soluti
ons\SAM7X_EK\TinyCLR\TinyCLR.proj (42,3)". This is most likely a build authorin
g error. This subsequent import will be ignored.
C:\MicroFrameworkPK_v4_1\Framework\Features\Network.featureproj(7,3): warning
MSB4011: "C:\MicroFrameworkPK_v4_1\Framework\Features\Network.featureproj" cannot
be imported again. It was already imported at "C:\MicroFrameworkPK_v4_1\Soluti
ons\SAM7X_EK\TinyCLR\TinyCLR.proj (42,3)". This is most likely a build authorin
g error. This subsequent import will be ignored.
C:\MicroFrameworkPK_v4_1\Framework\Features\Network.featureproj(8,3): warning
MSB4011: "C:\MicroFrameworkPK_v4_1\Framework\Features\Network.featureproj" ca
nnot be imported again. It was already imported at "C:\MicroFrameworkPK_v4_1\So
lutions\SAM7X_EK\TinyCLR\TinyCLR.proj (43,3)". This is most likely a build auth
oring error. This subsequent import will be ignored.
C:\MicroFrameworkPK_v4_1\Framework\Features\SerialPort.featureproj(6,3): warn
ing MSB4011: "C:\MicroFrameworkPK_v4_1\Framework\Features\SerialPort.featurepr
oj" cannot be imported again. It was already imported at "C:\MicroFrameworkPK_v
4_1\Solutions\SAM7X_EK\TinyCLR\TinyCLR.proj (47,3)". This is most likely a build
authoring error. This subsequent import will be ignored.
C:\MicroFrameworkPK_v4_1\Framework\Features\SerialPort.featureproj(7,3): warn
ing MSB4011: "C:\MicroFrameworkPK_v4_1\Framework\Features\SerialPort.featurepr
oj" cannot be imported again. It was already imported at "C:\MicroFrameworkPK_v
4_1\Solutions\SAM7X_EK\TinyCLR\TinyCLR.proj (48,3)". This is most likely a build
authoring error. This subsequent import will be ignored.
C:\MicroFrameworkPK_v4_1\Framework\Features\I2C.featureproj(5,3): warning MSB
4011: "C:\MicroFrameworkPK_v4_1\Framework\Features\I2C.featureproj" cannot
be imported again. It was already imported at "C:\MicroFrameworkPK_v4_1\Soluti
ons\SAM7X_EK\TinyCLR\TinyCLR.proj (43,3)". This is most likely a build authorin
g error. This subsequent import will be ignored.

7 Warning(s)
0 Error(s)

Time Elapsed 00:01:34.11
C:\MicroFrameworkPK_v4_1\Solutions\SAM7X_EK\TinyCLR>
```

Back to MFDeploy, select TinyCLR.hex file from the output folder and click Deploy. This will take a while! By the way, this runs much faster on other FEZ boards.

When done, click Ping and you will see TinyCLR instead of TinyBooter

```
Pinging... TinyCLR
```

We can now use Visual Studio to blink an LED or do whatever we like. If not sure how, use the other free “Beginner Guide to NETMF” ebook for details on using NETMF. This book only covers porting. Here is an example on blinking an LED, which is connected to PA7 (that is IO7).

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

namespace MFConsoleApplication1
{
    public class Program
    {
        public static void Main()
        {
            OutputPort LED = new OutputPort(Cpu.Pin.GPIO_Pin7, true);

            while (true)
            {
                Thread.Sleep(500);
                LED.Write(false);
                Thread.Sleep(500);
                LED.Write(true);
            }
        }
    }
}
```

8. Memory Layout

Building a software is usually a two step process, compiling and linking. The compiler is what changes the code to something the processor understand, machine language. A NETMF program will have hundreds of compiled files. The linker is what merge all compiled files together and places them at appropriate address in memory. You may have a 128MB located at address 0x20000000 or 16KB SRAM located at 0x00400000. The linker needs to be aware of these memories and it needs to know what will that memory be used for.

As far as NETMF compilation, there is only one option to change, which is code optimization level. This is automatically done when selecting “debug” vs “release” build. For the linker, we need to provide a file that the linker uses to place program memory sections appropriately. Unfortunately, this file is different from one compile to another. It is even named differently. GCC for example, call this file a linker script where RVDS calls this file a scatter file. To make this easier, NETMF uses an XML file with all info needed to generate the linker file, that is the linker script or scatter file. This XML file is located in the project's folder. For example, C:\MicroFrameworkPK_v4_1\Solutions\iMXS\TinyCLR\scatterfile_tinyclr_rvds.xml

Analyzing the code below, you will see where heap, stack, ROM, RAM is being placed.

```
<?xml version="1.0"?>
<ScatterFile xmlns="http://schemas.microsoft.com/netmf/ScatterfileSchema.xsd">
  <Set Name="Valid" Value="false"/>
  <Set Name="Heap_Begin" Value="0x08100000"/>
  <Set Name="Heap_End" Value="0x084FFFF8"/>
  <Set Name="Custom_Heap_Begin" Value="0x08500000"/>
  <Set Name="Custom_Heap_End" Value="0x088FFFF8"/>
  <Set Name="Stack_Bottom" Value="0x08900000"/>
  <Set Name="Stack_Top" Value="0x08902000"/>

  <If Name="TARGETLOCATION" In="FLASH">
    <!-- iMXS has 8MB of 32-bit FLASH at 0x10000000 -->
    <Set Name="Config_BaseAddress" Value="0x107E0000"/>
    <Set Name="Config_Size" Value="0x00200000"/>
    <Set Name="Code_BaseAddress" Value="0x10020000"/>
    <Set Name="Deploy_BaseAddress" Value="0x10100000"/>
    <Set Name="Code_Size" Value="%Deploy_BaseAddress - Code_BaseAddress%"/>
    <Set Name="Valid" Value="true"/>

    <If Name="TARGETTYPE" In="RELEASE DEBUG">
      <Set Name="Data_BaseAddress" Value="0x100C0000"/>
      <Set Name="Code_Size" Value="%Data_BaseAddress - Code_BaseAddress%"/>
      <Set Name="Data_Size" Value="%Deploy_BaseAddress - Data_BaseAddress%"/>
    </If>
  </If>

  <If Name="Valid" Value="false">
    <Error Message="Configuration not recognized"/>
  </If>

  <LoadRegion Name="LR_%TARGETLOCATION%" Base="%Code_BaseAddress%" Options="ABSOLUTE" Size="%Code_Size%">
    <ExecRegion Name="ER_%TARGETLOCATION%" Base="%Code_BaseAddress%" Options="FIXED" Size="">
      <FileMapping Name="FirstEntry.obj" Options="( +R0, +FIRST)" /> <!-- the entry pointer -->
      <FileMapping Name="*" Options="(SectionForBootstrapOperations)" />
    </ExecRegion>
  </LoadRegion>
</ScatterFile>
```

```

    <FileMapping Name="*" Options="(+RO-CODE)" />
    <FileMapping Name="*" Options="(+RO-DATA)" />
    <IfNotDefined Name="Data_BaseAddress">
        <FileMapping Name="tinyclr_dat.obj" Options="(+RO, +LAST)" />
    </IfNotDefined>
</ExecRegion>

<ExecRegion Name="ER_RAM_RO" Base="0x00000000" Options="ABSOLUTE" Size="0x0005ffe0">
    <FileMapping Name="VectorsTrampolines.obj" Options="(+RO, +FIRST)" />
    <FileMapping Name="*" Options="(SectionForFlashOperations)" />
</ExecRegion>

<ExecRegion Name="ER_RAM_RW" Base="+0" Options="ABSOLUTE" Size="">
    <FileMapping Name="*" Options="(+RW-DATA, +ZI)" />
</ExecRegion>

<ExecRegion Name="ER_HEAP_BEGIN" Base="%Heap_Begin%" Options="ABSOLUTE" Size="UNINIT">
    <FileMapping Name="*" Options="(SectionForHeapBegin)" />
</ExecRegion>

<ExecRegion Name="ER_HEAP_END" Base="%Heap_End%" Options="ABSOLUTE" Size="UNINIT">
    <FileMapping Name="*" Options="(SectionForHeapEnd)" />
</ExecRegion>

<ExecRegion Name="ER_CUSTOM_HEAP_BEGIN" Base="%Custom_Heap_Begin%" Options="ABSOLUTE" Size="UNINIT">
    <FileMapping Name="*" Options="(SectionForCustomHeapBegin)" />
</ExecRegion>

<ExecRegion Name="ER_CUSTOM_HEAP_END" Base="%Custom_Heap_End%" Options="ABSOLUTE" Size="UNINIT">
    <FileMapping Name="*" Options="(SectionForCustomHeapEnd)" />
</ExecRegion>

<ExecRegion Name="ER_STACK_BOTTOM" Base="%Stack_Bottom%" Options="ABSOLUTE" Size="UNINIT">
    <FileMapping Name="*" Options="(SectionForStackBottom)" />
</ExecRegion>

<ExecRegion Name="ER_STACK_TOP" Base="%Stack_Top%" Options="ABSOLUTE" Size="UNINIT">
    <FileMapping Name="*" Options="(SectionForStackTop)" />
</ExecRegion>
</LoadRegion>

<IfDefined Name="Data_BaseAddress">
    <LoadRegion Name="LR_DAT" Base="%Data_BaseAddress%" Options="ABSOLUTE" Size="%Data_Size%">
        <ExecRegion Name="ER_DAT" Base="%Data_BaseAddress%" Options="FIXED" Size="%Data_Size%">
            <FileMapping Name="tinyclr_dat.obj" Options="(+RO)" />
        </ExecRegion>
    </LoadRegion>
</IfDefined>

<IfDefined Name="Config_BaseAddress">
    <LoadRegion Name="LR_CONFIG" Base="%Config_BaseAddress%" Options="ABSOLUTE" Size="%Config_Size%">
        <ExecRegion Name="ER_CONFIG" Base="%Config_BaseAddress%" Options="FIXED" Size="%Config_Size%">
            <FileMapping Name="*" Options="(SectionForConfig)" />
        </ExecRegion>
    </LoadRegion>
</IfDefined>
</ScatterFile>

```

9. Interops and RLP

One of the most important features in NETMF is allowing developers to extend the system with additional native methods. Let us assume you need a method to calculate checksum. This method will have to loop through a 1000 byte array to calculate the checksum. This can take few seconds if it was running in C# (managed). A better alternative is to implement the checksum code in C++ then invoke this C++ method from C#. But in order for C# (managed code) to reach the C++ code (native code) there has to be some sort of connection point. This can either be accomplished through interops or RLP.

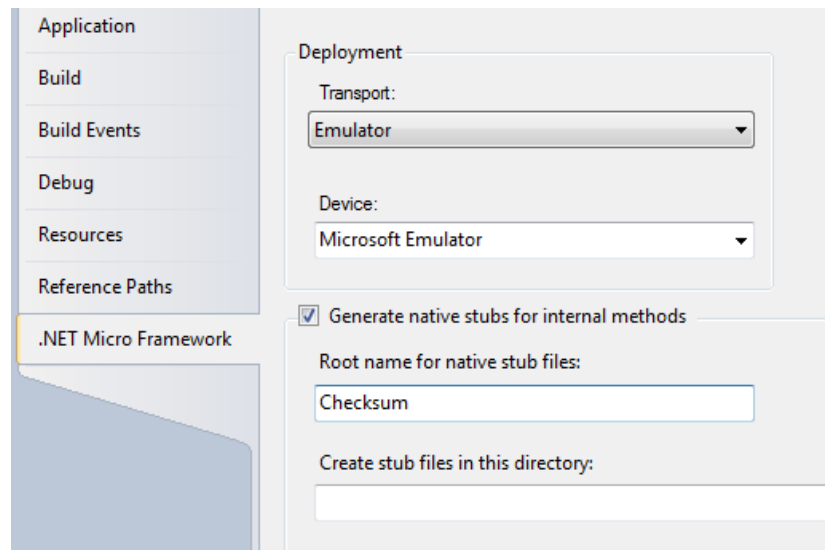
Both these methods are great but each one has its own advantages. Adding interops requiring a complete rebuild of the system and then become part of the system. They can't be dynamically added or removed at runtime. RLP (Runtime Loadable Procedure) is a GHI Electronics invention and it basically works very similarly to DLLs. An RLP is compiled as an entity separate from the firmware and the NETMF build system and can even be compiled with different compiler than what is used to compile the firmware. For example, GHI Electronics uses RVDS compiler for its official firmware (on closed and open source products) but then a developer can implement additional methods that are compiled using other compiler, GCC for example.

9.1. Interops

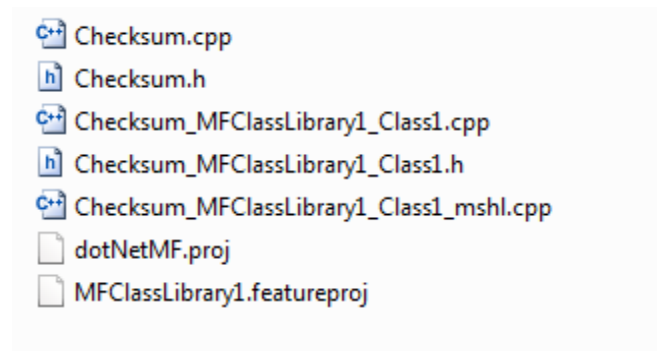
When creating a library that has interops, we need a way to hook these interops into the system. Start by creating a "class library" NETMF project. Add the needed methods wrappers, not the actual code, in your library. For example, here is a method that takes a byte array and returns an int, the checksum. Note how the method is "extern" and it is decorated with "InternalCall"

```
[MethodImplAttribute(MethodImplOptions.InternalCall)]  
extern public static int CalculateChecksum(byte[] data);
```

The next step is to activate the option "Generate native stubs for internal methods" in project properties as showing in image below.



When this option is selected and we have internal methods, visual studio will generate a file called "Stubs". Here is the folder contents



All files except MFClassLibrary1.featureproj need to be added to your solution's compile list in the build system. Those will generate an object that has the checksum code, which we haven't added yet. But even if you added this library to the build system, how would the CLR (the virtual machine) know about the new method? There is where we need MFClassLibrary1.featureproj file. This file needs to be added to TinyCLR.proj project file. I recommend inspecting GHI's open source library and seeing how it is added to the firmware. The library is found here <http://www.ghielectronics.com/downloads/NETMF/OSH%20Library%20Documentation/Index.html> with all instructions on finding and compiling the code found here http://wiki.tinyclr.com/index.php?title=FEZ_Hydra_Developer

So by now we have our checksum library added in the system but we still haven't added the actual checksum code. This is done in Checksum_MFClassLibrary1_Class1.cpp file

```
#include "Checksum.h"
```

```
#include "Checksum_MFClassLibrary1_Class1.h"
using namespace MFClassLibrary1;
INT32 Class1::CalculateChecksum( CLR_RT_TypedArray_UINT8 param0, HRESULT &hr )
{
    INT32 retVal = 0;
    // add your code here
    return retVal;
}
```

9.2. RLP

RLP is quite different that interops but very similar to DLL. The porting kit is not needed at all. Developers do not need to have any experience in porting NETMF or its internals. The RLP library is actually compiled using interops as explained before. This library provides way for users to load methods dynamically at runtime and invoke them.

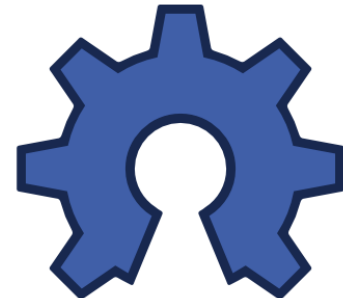
There are two versions of RLP, the complete version found in GHI's standard extended library. And RLPLite found in the open source version. This wiki page explains the differences between the two and shows an example on how it is used http://wiki.tinyclr.com/index.php?title=RLPLite_Demo

RLP is very important to open source projects but it is especially important to users of closed source NETMF boards where developers can extend the system functionality without needing access to the firmware source codes.

10. Open Source Hardware

When a product's hardware design files and related software are made open source then this product is Open Source Hardware, OSH for short. This allows other (community) to improve the product. Depending on the license used, this openness allow other companies/developers to create new products based the original design files and/or source codes.

GHI Electronics released many products with OSH license. This is easily determined by seeing the OSH logo on the catalog page or on the circuit board.

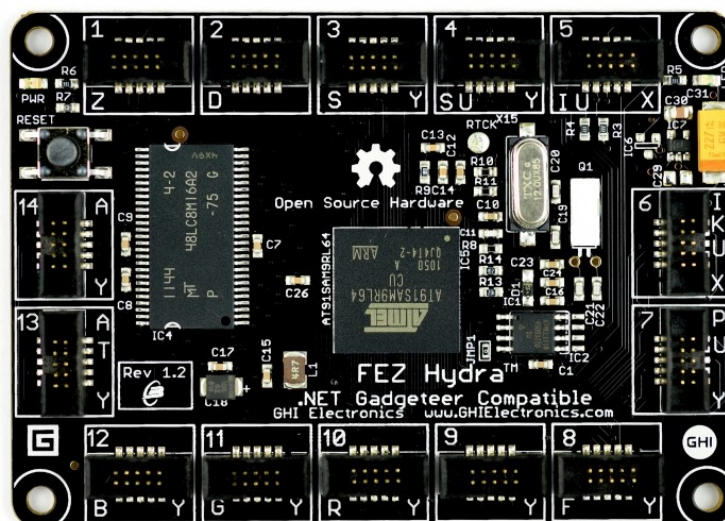


open hardware

Those OSH products running NETMF are ideal for those wanting to get into porting NETMF. They can be used as a very low cost learning platform or the base of a new product. Another advantage of using the sources these products over what is available in the porting kit that that these sources were tested on actual products vs in test labs plus there port for peripheral drivers is complete.

One example would be FEZ Hydra. A 200Mhz ARM9 NETMF board with 16MB of RAM and 4MB of FLASH. It includes drivers for all AT91SAM9R64 processor and even extends NETMF with GHI's OSH extended library. The library references are found here

<http://www.ghielectronics.com/downloads/NETMF/OSH%20Library%20Documentation/Index.html>



By the way, this is also .NET Gadgeteer maiboard so even the Gadgeteer related drivers are open source as well.

11. Porting vs GHI's Offers

There are two sides of working with NETMF, porting it and using it. For example, writing a JAVA game on a smart-phone is much easier than porting the JAVA virtual machine (JVM) to a smart-phone. Usually, the phone manufacturer does all the work of porting JAVA to their phone, then game programmers can use it with less efforts. NETMF works the same way, porting is not easy but using it is very easy.

Most available NETMF devices on the market use a processor that is already found in the porting kit. As far as I know, GHI Electronics is the only company to release public offer based on a complete NETMF port. By the way, this was GHI's very first product as well. GHI strongly believed and NETMF and wanted to master the porting kit, so we selected to do it the hard way. Mastering the PK allowed GHI to add a lot of exclusive features that made NETMF even more powerful.

Before starting the long process of porting, consider GHI Electronics' NETMF products. When using these products, you are not just using a NETMF device but you are receiving **features, support, maintenance, robustness and time-to-market**. Let's cover these in detail.

Features

GHI's NETMF products include many exclusive features, such as USB Host, USB Device, one-wire, CAN, PPP, WiFi, etc. All these are included at no additional cost. GHI continues to add exclusive features via updates free of charge!

Support

Our world-class support is free. The same engineers that invented these devices are monitoring the forums, emails and phone to provide superior support. We're here to assist you every step of the way until your product is on the market as soon as possible. We would love for you to visit our forum and ask other customers how satisfied they are with GHI support.

Maintenance

Every few months, Microsoft releases a new NETMF version. GHI works very closely with Microsoft on any new possible issues and does all the work required to update all GHI's NETMF devices. For GHI customers, this is a five minute FREE firmware update and GHI takes care of the rest.

Robustness

There are thousands of GHI's NETMF devices used around the world in most markets. This

vast usage guarantees quality and stability of GHI Electronics products. You can use any of the GHI Electronics products with ease of mind.

Time-to-Market

Using GHI Electronics' NETMF products will speed up development. Your design is almost done as soon as you add one of the GHI NETMF products. We have seen customers that create full products in a week! You can for example, take a FEZ Rhino starter kit, write a little code over a few days, add your company's logo-sticker on top and you have your own product. You will probably spend most of your time designing/ordering the logo-sticker than you would spend on the hardware design!

GHI Electronics profile <http://www.ghielectronics.com/profile/>

GHI Electronics catalog <http://www.ghielectronics.com/catalog/>

12. Additional Resources

GHI Electronics provides many additional, and free, resources.

Tutorials and Downloads

This page includes plenty of tutorials covering about everything NETMF. It also includes all needed links for downloads, references and any other links needed by NETMF users. You may want to bookmark this page.

<http://www.tinyclr.com/support/>

Codeshare

This website includes hundreds of code snippets and complete projects provided by the community. The source of the files is visible right on the website for those wanting to look for a “cheat sheet” or you can download the complete files as well. We look forward to see your own contributions on codeshare

<http://code.tinyclr.com>

eBooks

Beginners' NETMF Guide:

<http://www.ghielectronics.com/downloads/FEZ/Beginners%20guide%20to%20NETMF.pdf>

Beginners' NETMF Porting Guide:

<http://www.ghielectronics.com/downloads/FEZ/Beginners%20Guide%20to%20Porting%20NETMF.pdf>

Internet of things:

http://www.ghielectronics.com/downloads/FEZ/FEZ_Internet_of_Things_Book.pdf

.NET Gadgeteer Ultimate Guide:

<http://www.ghielectronics.com/downloads/Gadgeteer/.NET%20Gadgeteer%20Ultimate%20Guide.pdf>