Guia para iniciantes em C#

e

.NET Micro Framework

11 de Agosto de 2010

Rev 1.04

www.GHIElectronics.com

www.TinyCLR.com

Por: Gus Issa

Versão Portuguêsa: Miguel Alexandre Wisintainer

Sumário

1. Histórico de mudanças

Versão 1.04

- Corrigidos mais alguns erros de digitação
- Trocado texto VS2008 para VS2010
- Adicionada uma nota sobre uma expansão de I/O chamada IO40
- Atualizada a seção do watchdog, desabilitação não é permitido

Versão 1.03

- Corrigido mais erros de digitação
- Revisada as fontes dos exemplos
- Revisado o parágrafo 16.1, Byte
- Reescrito “tópicos faltantes”
- Adicionada seção de Watch Dog
- Adicionada seção de Baixo Consumo
- Adicionada nota importantes sobre Coletor de Lixo
- Adicionada subseção do Coletor de Lixo, sobre mensagens de saída
- Adicionado outros dispositivos na seção Selecionando um Dispositivo
- Trocada a seção Display gráficos para para apontar para novos displays SPI na www.TinyCLR.com
- O índice agora tem 2 colunas

Versão 1.02

- Corrigidos erros de digitação
- Atualizado o uso do VS2010 em vez do VS2008. Atualizado 4.0 para 4.1 e colocando novo link para para download da versão 4.1
- Adicionado nota sobre filtros de Glitch com pinos capazes de gerar interrupção
- Reescrita seção sobre relógio em tempo real
- Adicionada seção sobre cliente USB

Versão 1.01

- Adicionada seção para tradução do livro
- Consertado um pequeno bug no exemplo de matriz de botões

Version 1.00

- Primeira versão oficial

2. Sobre o livro

2.1. Público Alvo

Este livro é para iniciantes que querem aprender .NET Micro Framework. Nenhum conhecimento prévio é necessário. O livro cobre .NET Micro Framework, Visual C# e C#!

Se você é um programador, um hobbista ou um engenheiro, você encontrará uma boa quantidade de informações neste livro. Este livro não faz nenhuma suposição sobre o que você, leitor sabe, é explicado detalhadamente.

Eu usei o meu tempo livre pessoal (se é que já está disponível!) para escrever este livro. Esperamos um grande número de erros ortográficos e gramaticais, mas por favor informe nos pelo fórum para que eu possa melhorar este livro.

2.2. Traduzindo o livro

Este livro é dado para a comunidade em um esforço para fazer NETMF mais fácil para todos os usuários. Se você acha que pode traduzir o livro para outros idiomas, então nós adoraríamos ver sua contribuição. As fontes originais do livro e as regras estão nesta página:

http://www.microframeworkprojects.com/index.php?title=Free_eBook

3. Introdução

Alguma vez você já pensou em alguma grande idéia para um produto, mas você não pode trazê-lo para o mundo real porque a tecnologia não estava do seu lado? Ou talvez pensou: "tem que ser uma maneira mais fácil de fazer!". Talvez você seja um programador que queria fazer um sistema de segurança, mas depois pensou que utilizar os PCs é muito caro para executar um sistema simples? A resposta é : Microsoft .NET Micro Framework!

Aqui está um cenário, você quer fazer um data logger de GPS de bolso, que salve as posições, aceleração e temperatura em um cartão de memória e os exiba em uma tela pequena, dispositivos GPS pode enviar dados de posição através de uma porta serial para que você possa escrever algum código em um PC para ler os dados de GPS e salvá-lo em um arquivo. Mas um PC não pode caber no seu bolso! Outro problema é como você vai medir a temperatura e aceleração em um PC? Se você fizer este projeto usando microcontroladores clássicos, como AVR, PIC, tudo isso pode ser feito, mas então você precisa de um compilador para o micro que você escolher (provavelmente não livre), uma semana para aprender o processador, uma semana para escrever o driver da serial, um mês ou mais para descobrir o sistema de arquivos FAT e mais tempo para cartões de memória, etc ... Basicamente, isso pode ser feito em poucas semanas de trabalho.

Outra opção é utilizar métodos mais simples (Basic Stamp, PICAXE, Arduino, etc). Todos estes produtos simplificam um projeto, mas cada um tem a sua limitação. Somente alguns deles têm capacidade de depuração. Finalmente, esses dispositivos não são normalmente adequados para produção em massa.

3.1. Vantagens

Se você está utilizando o .NET Micro Framework, então há muitas vantagens:

1. Ele roda no Microsoft Visual C# express que é grátis e tem uma IDE de alta performance.
2. .NET Micro Framework é gratuíto e open-source.
3. Seu código irá rodar em todos os dispositivos com poucas modificações.
4. Capacidade de debug completo (Breakpoints, execução passo a passo, variáveis...etc.).
5. Tem sido testado em muitos produtos comerciais de forma que a qualidade foi garantida.
6. Inclui muitos drivers (SPI, UART , I2C...etc.).
7. Não há necessidade de datasheets do processador porque graças ao padrão do Framework.

8. Se você já é um programador C# para PC, então você esta pronto para ser um desenvolvedor de sistemas embarcados com NETMF!

   Ao longo deste documento, NET Micro Framework será referenciado como NETMF.

4. Portabilidade

Existem duas faces de trabalhar com NETMF, portando e usando. Por exemplo, escrevendo um jogo JAVA em um celular é muito mais fácil do que colocar a máquina virtual Java (JVM) em um telefone. O fabricante já fez fez todo o trabalho de portar JAVA para o seu telefone para que programadores de jogos possa usá-lo com menos esforço. NETMF funciona da mesma maneira, portar não é muito fácil mas usá-lo é muito fácil.

NETMF pode ser dividida em dois componentes principais: o núcleo e HAL (Hardware Access Layer). As principais bibliotecas do nucleo são feitos para que sejam independentes do hardware. Normalmente não são necessárias modificações nas bibliotecas do núcleo. Os desenvolvedores precisam para fazer seu próprio HAL para que o NETMF funcione em um dispositivo.

4.1. Ofertas do padrão GHI

Nós trabalhamos muito próximos de nossos clientes para se certificar que tudo está correndo conforme o necessário. GHI oferece muitos recursos exclusivos que são padrões e grátis com os nossos produtos (USB host, database, PPP, RLP, Wi-Fi, OneWIRE, etc...).

Suporte dedicado está disponível gratuitamente através de e-mail, telefone e fórum.

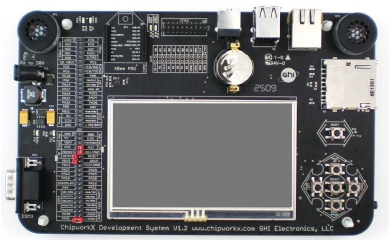
5. Selecionando um Dispositivo

GHI Electronics oferece uma variedade de produtos do básico ao avançado.

- Família X
- Família EMX
- USBizi
- Família FEZ
  - FEZ Domino e FEZ Mini
- FEZ Cobra

5. 1 ChipworkX

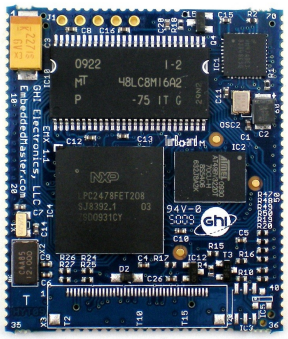
Se o poder de processamento e customizacação são necessários, então este é a escolha certa.
ChipworkX roda um processador ARM de 200Mhz com 64MB de memória SDRAM de 32 bits e 8MB para aplicações do usuário. Também contém 256MB de flash interna para armazenamento do sistema. Ele inclui todas os grandes benifícios e também exclusivos benefícios como WiFi e suporte a USB host.

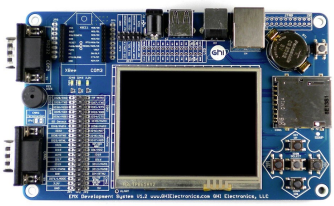


ChipworkX também adiciona suporte de banco de dados SQLite e permite aos usuários carregar o seu próprio código nativo (C / assembly) no dispositivo usando RLP (Runtime Loadable Procedures). RLP permite a programação de aplicações intensivas e em tempo real.

5.2 EMX

Este pequeno módulo inclui todas as grandes funcionalidades do NETMF e adiciona muitas características exclusivas da GHI. No lado do software: o sistema de arquivos, TCP / IP, SSL, gráficos, depuração e mais recursos NETMF estão incluídos. GHI também acrescenta: WiFi, PPP, USB host, construtor de dispositivo USB, CAN, entrada / saída analógica, PWM e muito mais. Quanto ao hardware: É um processador de ARM de com 8MB SDRAM e FLASH 4.5MB.

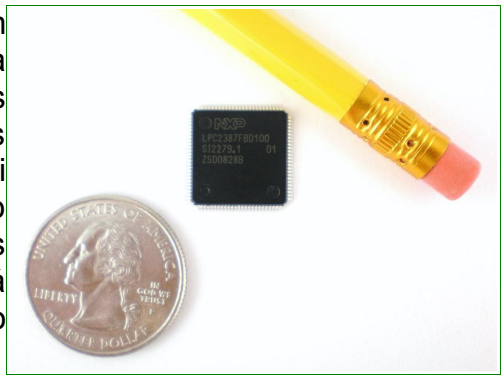




O processador de EMX contém nativamente o Ethernet MAC com transferências via DMA, que lhe dá um grande impulso quando comparado com o chipset Ethernet clássica baseado no SPI, utilizado por outros.

5.3USBizi Chipset

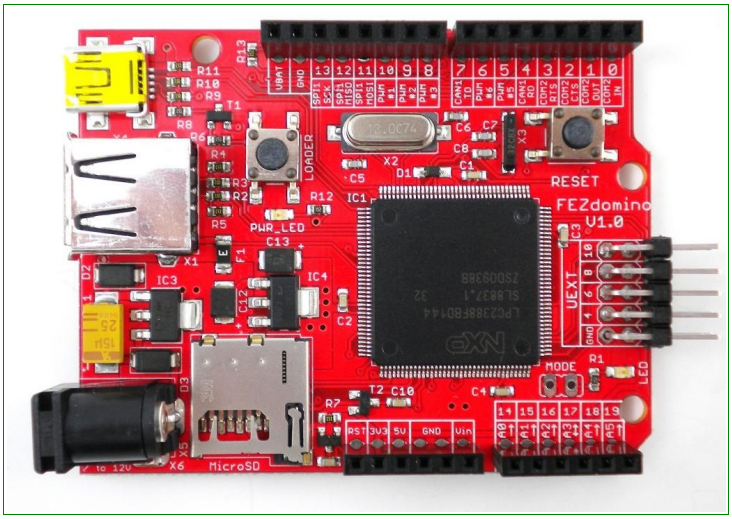
USBizi é o menor e único single-chip NETMF em execução no mundo. O software é executado em uma versão reduzida do Embedded Master. Ele inclui todos os recursos, exceto em rede (TCP / IP e PPP) e gráficos nativa. Mesmo que esses recursos estão faltando, USBizi pode ser conectado a uma rede usando TCP / IP como chipsets WIZnet e pode ser executado com alguns displays gráficos simples. Há exemplos de projetos já providos demonstrando como USBizi pode ser colocado em rede e pode exibir gráficos em displays.

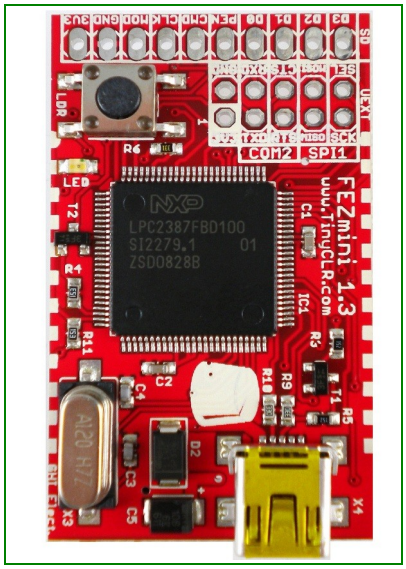


5.4 Família FEZ

FEZ Domino e FEZ Mini

FEZ FEZ Domino e Mini são placas muito pequenos (open source) específicas para iniciantes. Eles são baseados no chipset USBizi. FEZ oferece vários periféricos, como USB host e interface SD, não está disponível em muitas placas destinadas a hobbista. Mesmo FEZ sendo direcionado para iniciantes, é também ponto de partidade de baixo custo profissionais que querem explorar NETMF.



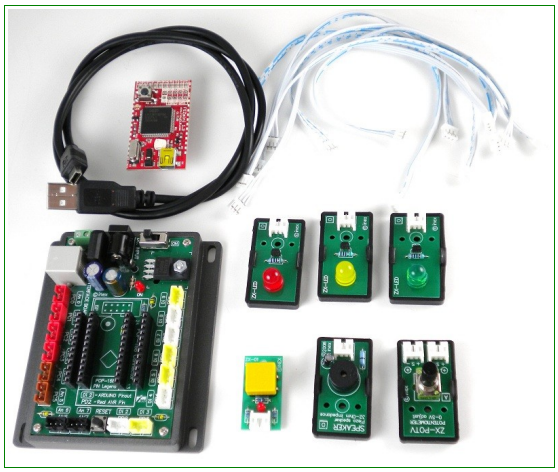


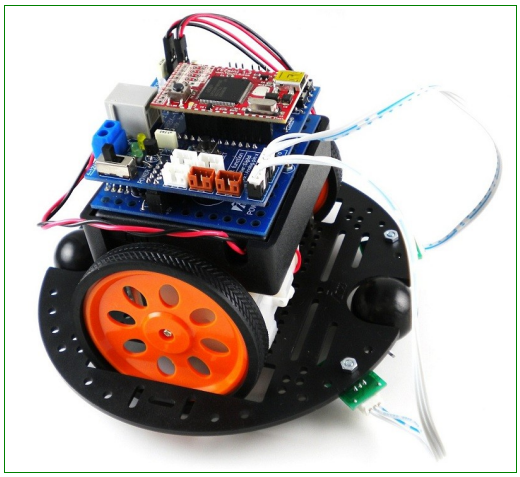
FEZ significa “Freakin' Easy!”

FEZ oferece muitas características não encontradas em Arduino, Basic Stamp e outros:

- Baseado no .NET Framework da Microsoft.
- Executa em um processador ARM da NXP de 72Mhz.
- Suporte a debug em tempo real (breakpoints, inspeção de variáveis, execução passo a passo, etc.).
- Utiliza Visual Studio 2008 C# Express Edition para desenvolvimento.
- Capacidades avançadas como FAT, dispositivo USB e USB host.
- Fácil upgrade para Hardware como o Embedded Master.
- Arquivos de Design de Hardware grátis.
- Compatível com shields existentes.
- Baseado no chipset USBizi(ideal para uso comercial).
- Pinagem do FEZ Mini compatível com BS2.
- Pinagem do FEZ Domino compatível com Arduino.

Quando usando FEZ, as possibilidades são infinitas...



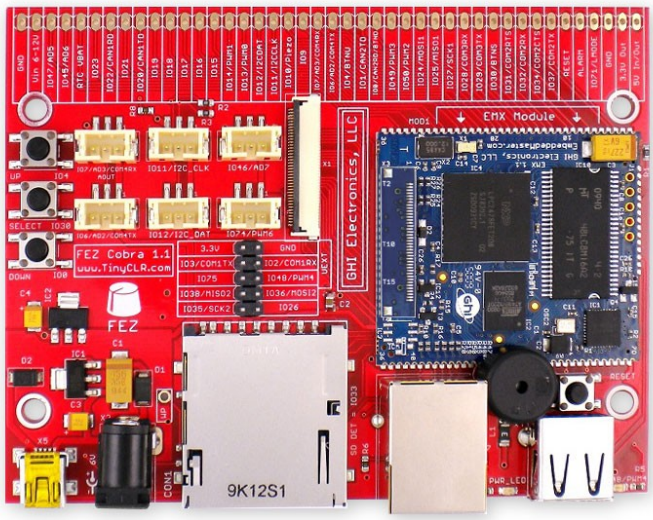


Há uma quantidade numerosa de sensores que estão prontos para ligar diretamente nos kits . De simples LEDs e botões, bem como sensores de temperatura ou distância.

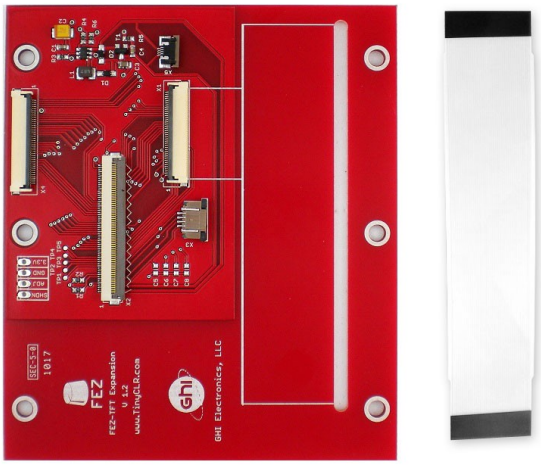
Estes exemplos do livro são feitos para dispositivos FEZ. Em geral, os exemplos são ainda para. NET Micro Framework, para modificá-lo para rodar em qualquer sistema NETMF deve ser uma tarefa fácil.

FEZ Cobra

FEZ Cobra é uma placa de circuito open source baseada em módulos EMX. Com múltiplos opcionais mostradores, gabinete opcional, suporte nativo de gráficos, suporte a Ethernet nativo com SSL, e megabytes de memória, FEZ Cobra é ideal para aqueles projetos high-end. Não esquecer que este ainda é FEZ, ou seja, fácil de usar!



Uma expansão LCD é disponível para prender tanto displays de 3.5” (320x240) ou 4.3”



(480x272)

Outros dispositivos

GHI trabalha para oferecer as mais recentes tecnologias de forma dinâmica para o que seu cliente necessita. Há sempre algo de novo no GHI, por favor visite www.GHIElectronics.com e www.TinyCLR.com para as últimas ofertas.

6. Primeiros Passos

Nota Importante 1: **Se você recebeu o seu equipamento ou se você não tiver certeza do firmware carregado nele, você deve atualizar o firmware. A documentação (manual ou tutorial) de seu equipamento mostra como a atualização. Este é um passo necessário. Além disso, verifique se você leu o título "Coerência Firmware / Assembly" deste livro.**

6.1. Configuração do Sistema

Antes de tentar qualquer coisa, temos de assegurar que o PC está configurado com o software necessário. Para isso, devemos primeiro fazer o download e instalar o **Visual C# Express 2010** (VS2008 ou mais antigo não irá funcionar)

http://www.microsoft.com/express/vcsharp/

Então faça o download e instale NET Micro Framework 4.1 SDK (não o kit de portabilidade).

http://www.microsoft.com/downloads/details.aspx?displaylang=en&FamilyID=cff5a7b7-c21c-4127-ac65-5516384da3a0

Se o link acima não funcionar, procurar ".NET Micro Framework 4.1 SDK"

Por fim, instale o SDK GHI NETMF. Você pode obtê-lo aqui:

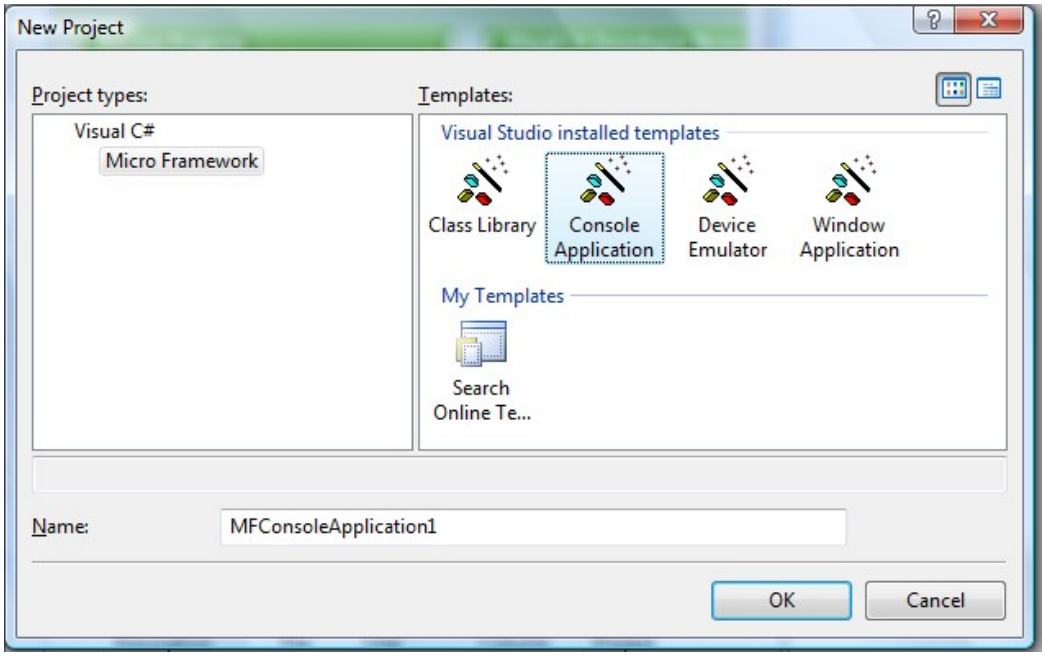
http://www.tinyclr.com/dl/

6.2. O emulador

NETMF inclui um emulador que permite a execução de aplicação NETMF diretamente no PC. Para o nosso primeiro projeto, vamos usar um emulador para rodar uma aplicação simples.

Criando um Projeto

Abra o Visual C# Express, e, no menu, escolha **File -> New Project**. O assistente deve mencionar "Micro Framework" no menu à esquerda. Clique nele e, nos modelos, escolha "Console Application".



Clique no botão "OK" e você terá um novo projeto que está pronto para executar. O projeto tem somente o arquivo C#, chamado Program.cs, que contém poucas linhas de código. O arquivo é mostrado na janela "Solution Explorer". Se o Windows não está mostrando isso, então você pode abri-lo clicando em "View> Solution Explorer" no menu.

```
using System;
using Microsoft.SPOT;

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

    }
```

```
}
```

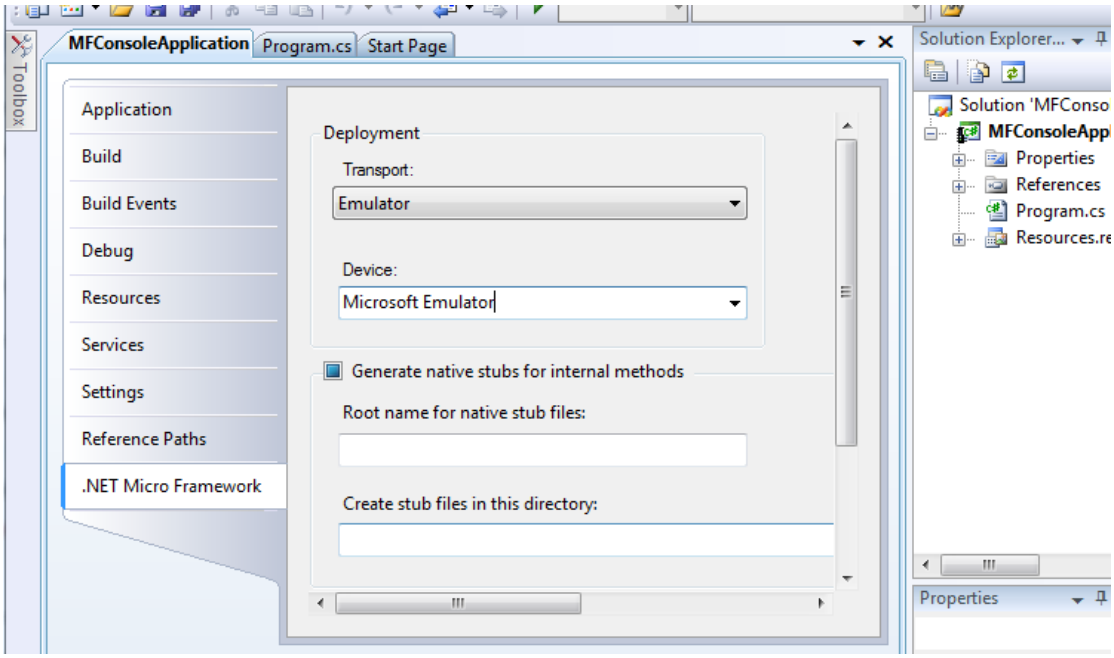
Para simplificar, altere o código, deve ser idêntico ao abaixo:

```
using System;
using Microsoft.SPOT;

namespace MFConsoleApplication1
{
    public class Program
    {
        public static void Main()
        {
            Debug.Print("Amazing!");
        }

    }
}
```

Selecionando Transporte

Não se desespere se você não entender o código. Vou explicar mais tarde. Por agora, queremos executar no emulador. Vamos garantir que temos tudo configurado corretamente. Clique em "Project-> Properties" no menu. Na janela que aparecer, você irá selecionar o emulador. Na esquerda, selecione ".NET Microframework" e certifique-se da janela parecida com a abaixo.

Transport: emulator



Device: emulador Microsoft

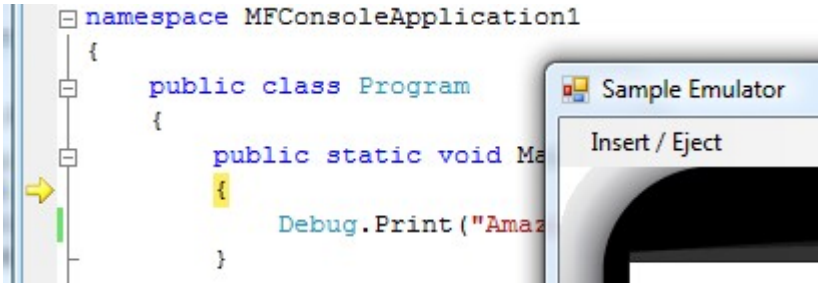
Certifique-se a janela de saída está visível, clique em "View->Output"

Executando

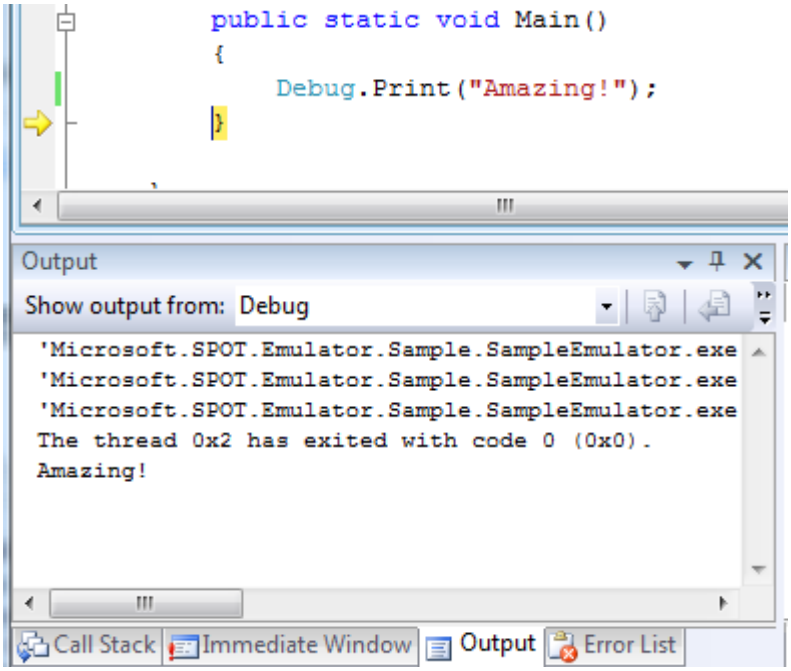
Estamos finalmente prontos para rodar a nossa primeira aplicação. Pressione a tecla F5. É um atalho muito útil e você vai usá-lo muitas vezes para executar seus aplicativos. Depois de pressionar F5, o aplicativo será compilado e carregado no emulador e alguns segundos depois pára, tudo! Porque o nosso programa terminou tão rápido que não vimos muito.

Nós queremos depurar o código agora. Depurar significa que você pode ver o passo-a-passo através do código do programa. Esta é uma das principais características do NETMF.

Desta vez, utilizando a tecla F11 em vez de F5, que vai acompanhar as instruções, ao invés de apenas executar o programa. Isto irá disparar o aplicativo no emulador e parar na primeira linha de código. Isto é indicado pela seta amarela.



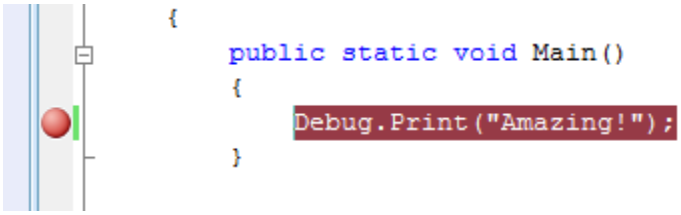
Aplicações C# começam sempre a partir de um método chamado Main e este é o lugar onde a seta tem parado. Pressione F11 novamente e o depurador irá executar a seguinte linha de código que você modificou anteriormente. Você provavelmente já adivinhou, esta linha vai escrever "Amazing!" na janela de depuração. A janela de depuração é a janela de saída do Visual C# Express. Certifique-se a janela de saída é visível, como explicado acima e pressione F11 novamente. Quando você passar desta linha, a palavra "Amazing" aparecerá na janela de saída.



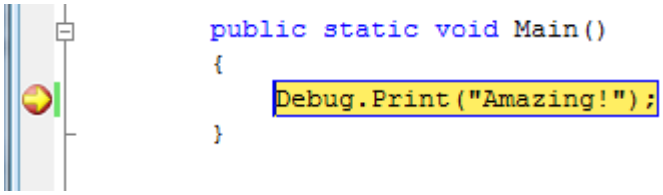
Se você pressionar F11 novamente, o programa irá terminar e emulador também.

Breakpoints

Breakpoints é outra característica útil quando a depuração de código. Enquanto o aplicativo é executado, o depurador verifica se a execução chegou a um ponto de interrupção. Se assim for, a execução será interrompida. Clique na barra esquerda, à direita da linha que imprime "Amazing!". Isto irá mostrar um ponto vermelho que é o ponto de interrupção.



Agora pressione F5 para executar o software e quando o aplicativo chega ao ponto de interrupção do depurador, ele pausa mostrando como na imagem abaixo



Neste momento, você pode executar o passo-a-passo com F11, ou F5 para continuar a execução

6.3. Executando no Hardware

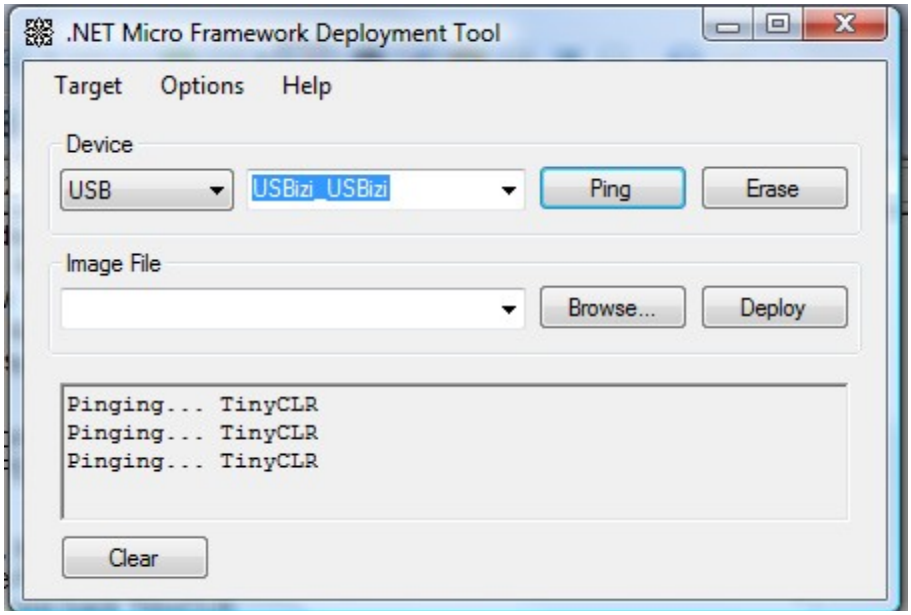
Executando aplicações em equipamentos NETMF é muito simples. Instruções podem diferir dependendo do hardware. Este livro utiliza FEZ para demonstração, mas qualquer outro hardware pode funcionar similarmente

MFDeploy pode enviar um ping!

Antes de utilizar o Hardware, podemos garantir que esteja conectado corretamente. O SDK NETMF vem com um programa chamado Microsoft MFDeploy. Existem diversas maneiras de usar MFDeploy mas por agora só precisamos "pingar" com o Hardware. Para simplificar, MFDeploy vai dizer "Hi!" para verificar se ele também irá responder com "Hi!". Isso é útil para verificar se o Hardware está ligado e não há problemas de comunicação.

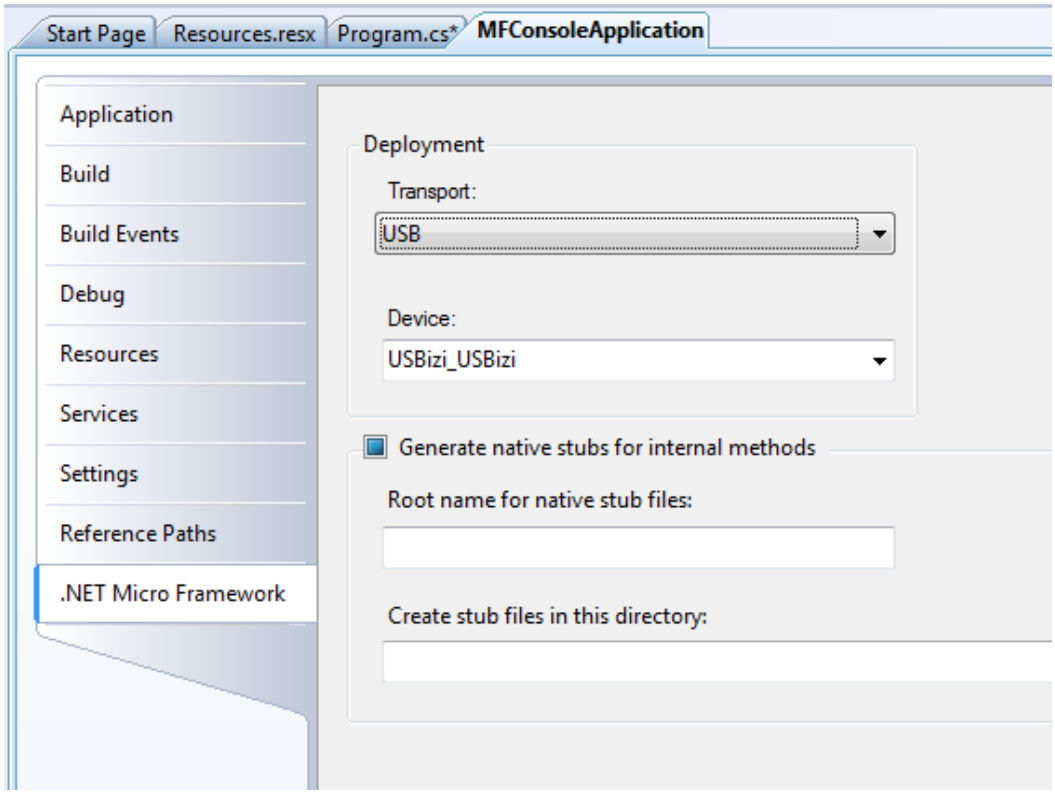
Abra o MFDeploy e conecte o FEZ ao PC através do cabo USB fornecido. Se esta foi a primeira vez que você conectou o FEZ, o Windows irá pedir por drivers. Aponte para o diretório de instalação do SDK e aguarde até que o Windows conclua a sua instalação.

Na lista, selecione USB. Na lista de equipamentos, você deve ver USBizi. Você verá USBizi porque FEZ é baseado no chipset USBizi. Escolha USBizi e clique no botão "Ping". Você deve ver agora de volta "TinyCLR".



Implantação ao Hardware

Agora sabemos que o equipamento é conectado através de MFDeploy, devemos voltar no Visual C# Express. Nas propriedades do projeto, selecione USB para o Transport e USBizi para o Hardware. Verifique se o seu setup parece com que está na tela abaixo.



Pressionando F5 irá enviar nossa pequena aplicação para o FEZ e executar em Hardware real. Mudar do emulador para hardware real é tão simples!

Tente os passos que nós seguimos com o emulador, como a definição de breakpoints e usar F11 para execucação passo-a-passo. Note-se que "Debug.Print" continuará a enviar mensagens de depuração do equipamento para a janela de saída do Visual C# Express.

7. Drivers de Componentes

Os componentes FEZ (LEDs, botões, sensores de temperatura, relés, controladores de servo ... etc.) e interfaces shield (Ethernet, controladores de motores de LCD ... etc.) são fornecidos com exemplos com seus drivers. Esses drivers supoe que você não conhece nada sobrem Hardware. Por exemplo, para piscar um LED, basta pedir para o driver para fazê-lo. Não há menção de pinos do processador e como alterar o status do pino,etc …Por outro lado, este livro ensina os princípios. Use os drivers dos componentes para começar a utilizar este livro para entender o que realmente esses drivers fazem.

8. C# Nível 1

Este livro não se destina a ensinar C#, mas irá abranger a maioria das bases necessárias para começar.

Para a aprendizagem de C# não ser tediosa, eu vou dividir em vários níveis, para que possamos fazer coisas mais interessantes com NETMF e voltar ao C#, quando necessário.

8.1. O que é .NET?

Microsoft desenvolveu .NET Framework para padronizar programação. (Note que eu estou falando sobre NET Framework e não o Micro Framework.) Há um conjunto de bibliotecas que desenvolvedores podem usar de muitas linguagens de programação. O .NET Framework roda em PCs e não em dispositivos menores, porque é um é um framework muito amplo. Além disso, a estrutura completa tem muitas coisas que seriam muito úteis em dispositivos menores. Para isto NET Compact Framework nasceu. O compact framework removeu bibliotecas desnecessárias para diminuir o tamanho. Esta versão menor roda no Windows CE e smart phones. O compact framework menor que o framework completo mas ainda é demasiado grande para rodar pequenos dispositivos por causa do tamanho e porque exigem um sistema operacional para rodá-lo.

. NET Micro Framework é a versão menor desses frameworks. Removeu mais bibliotecas e tornou-se um sistema operacional independente. Devido à semelhança entre estes 3 frameworks, quase todo código rodar em Pcs, pode rodar em dispositivos pequenos, com pouca ou nenhuma modificação.

Por exemplo, usando a porta serial de um PC, dispositivo WinCE ou FEZ (USBizi) funciona da mesma maneira, quando se utiliza. NET.

8.2. O que é C#?

C e C++ são linguagens de programação mais populares. C# é uma melhoria e mais moderno que C e C++. Ele inclui tudo que você esperaria de uma linguagem moderna, como coleta de lixo e validação em tempo de execução. Ele também é orientado a objeto, o que torna os programas portáteis e mais fáceis de depuração. C# usa um monte de regras de programação para reduzir a possibilidade de erros, provê da maioria dos recursos do C/C++.

“Main” é o ponto de partida

Como vimos antes, os programas sempre iniciam em um método chamado Main.Um método é um pequeno pedaço de código que executa uma tarefa específica. Os métodos começam e

terminar com chaves “{“ e “}”. Em nosso primeiro programa, tínhamos apenas uma linha de código entre as chaves.

A linha foi Debug.Print ("Amazing!");

Pode ver que a linha termina com um ponto e vírgula. Todas as linhas terminam assim. Esta linha chama o método Print que existe no objeto Debug. Chama ele passando o parâmetro "Amazing".

Confuso? Vamos esclarecer um pouco. Suponha que você é um objeto. Você terá vários métodos para controlá-lo. Um método pode ser "sente" e outro seria "Corra”. Agora, como você faria para "dizer" Amazing? Vai chamar o seu método "dizer" com a frase (string) "Amazing!!". E o código seria o seguinte:

Voce.diz(“Amazing!”);

Por que precisamos de aspas de depois da palavra “Amazing!!” ? Porque C# não sabe que o texto que você está escrevendo é um comando ou apenas um texto real. Você pode ver que está colorido em vermelho quando adiciona aspas, o que faz a a leitura do código muito mais fácil.

Comentários

Como adicionar notas, comentários, avisos em seu código? Estas opiniões vão ajudá-lo a entender o que o código. C# ignora completamente estes comentários. Há duas maneiras para postar comentários: linha ou bloco. Os comentários são escritos em verde. Para comentar uma linha ou uma parte da linha, adicione / / antes que o texto do comentário. A cor do texto fica verde, indicando que o texto é um comentário e é ignorado pelo C#.

```
using System;
using Microsoft.SPOT;

namespace MFConsoleApplication1
{
    public class Program
    {
        public static void Main()
        {
            // Isto é um comentário
            Debug.Print("Amazing!");//Isto é um comentário também!
        }

    }
}
```

Você pode comentar um bloco completo. Começe o comentário com o símbolo /* e termine com o símbolo */ .

```
using System;
using Microsoft.SPOT;

namespace MFConsoleApplication1
{
    public class Program
    {
        public static void Main()
        {
            /* Isto é um comentário
            isto ainda é um comentário
            o bloco de comentário termina aqui*/
            Debug.Print("Amazing!");
        }

    }
}
```

While-loop

Este é o momento de nossa primeira palavra-chave "while". Um while começa e termina com chaves com o código entre os dois. Tudo entre as chaves será executado enquanto uma condição é verdadeira. Por exemplo, eu peço que você continue a ler este livro, enquanto você está acordado!

Por isso, vamos garantir que o programa exibe continuamente "Amazing!!". Este ciclo é sem fim, a condição será sempre "true" (verdadeiro).

```
using System;
using Microsoft.SPOT;

namespace MFConsoleApplication1
{
    public class Program
    {
        public static void Main()
        {
            while(true)
            {
                Debug.Print("Amazing!");
            }
        }

    }
}
```

No código acima, a execução terá início no método “Main”, como de costume, em seguida,

irá para a próxima linha que é o loop while. O laço diz que o código dentro das chaves deve ser executado enquanto que a condição é "verdadeira". Na verdade, nós não temos nenhuma sentença, mas temos diretamente "True", que indica que o loop será executado indefinidamente.

Não tecle F5 senão você irá preencher a janela com a palavra "Amazing!", em vez disso, pressione a tecla F11 e ir passo a passo ao longo do circuito para ver como ele funciona. Note que este programa nunca vai parar, então você precisa pressionar Shift-F5 para forçar o desligamento.Nota: Você pode acessar todos esses atalhos do menu a opção "Debug".

Variáveis

As variáveis são porções de memória reservado para seu uso. A quantidade de memória reservada depende do tipo da variável. Eu não vou cobrir cada tipo aqui, mas qualquer livro sobre C # irá explicar em detalhes.Vamos usar variáveis "int". Este tipo de variável é usado para armazenar números inteiros.

Vamos usar variáveis "int". Este tipo de variável é usado para armazenar números inteiros.

Explanação:

Int MyVar nt;

Informar ao sistema para reservar memória. Esta memória será referenciado por "MyVar. Você pode dar qualquer nome, desde que este nome não contenha espaços. Agora você pode colocar qualquer número inteiro na memória:

MyVar = 1234;

Você também pode usar funções matemáticas para calcular números:

MyVar = 123 + 456;

ou incrementar o número de 1

MyVar++;

ou diminuição de 1:

MyVar -;

Com isso, podemos escrever um programa que exibe três vezes a palavra "Amazing!"? Aqui está o código:

```
using System;
using Microsoft.SPOT;

namespace MFConsoleApplication1
{
    public class Program
```

```
    {
        public static void Main()
        {
            int MyVar;
            MyVar = 3;
            while(MyVar>0)
            {
                MyVar--;
                Debug.Print("Amazing!");
            }
        }

    }
}
```

Note-se que o loop While não é verdade o tempo todo, mas enquanto MyVar> 0. Isso quer dizer, continua a loop até o valor de myVar é maior que 0.

No primeiro ciclo, MyVar é igual a 3. Em cada ciclo é aplicado um descescimo de 1 em Myvar. O laço será executado exatamente três vezes e em seguida, exibir três vezes a palavra "Amazing!"

Vamos fazer algo mais interessante. Eu quero mostrar os números de 1 a 10. Ok, sabemos como criar uma variável e como o incremento, mas como mostrar a janela de saída? Fornecer MyVar para Debug.Print vai dar um erro porque só aceita strings, não números inteiros. Como converter um inteiro para uma string? Muito simples: use MyVar.ToString(). Fácil!

```
using System;
using Microsoft.SPOT;

namespace MFConsoleApplication1
{
    public class Program
    {
        public static void Main()
        {
            int MyVar;
            MyVar = 0;
            while(MyVar<10)
            {
                MyVar++;
                Debug.Print(MyVar.ToString());
            }
        }

    }
}
```


A última coisa que vamos acrescentar: queremos que o programa escreva:

Count:: 1

Count: 2

...

...

Count: 9

Count:10

Isto pode ser facilmente feito adicionando strings. Strings são adicionadas usando o símbolo + como se adicionasse qualquer número.

Tente o código seguinte

```
using System;
using Microsoft.SPOT;

namespace MFConsoleApplication1
{
    public class Program
    {
        public static void Main()
        {
            int MyVar;
            MyVar = 0;
            while(MyVar<10)
            {
                MyVar++;
                Debug.Print("Count: " + MyVar.ToString());
            }
        }

    }
}
```

Assemblies

São arquivos que contém o código compilado (montados). Isto permite ao desenvolvedor usar o código embora não tenha acesso ao código fonte. Nós já usamos Debug.Print, por exemplo. Mas quem criou a classe de depuração e escreveu o método de impressão no interior? Estes foram criados pela equipe NETMF Microsoft. Eles compilam o código e fornecem-no com um conjunto para o uso. Assim você não precisa se preocupar com detalhes internos para utilizá-lo.

No início do código usado anteriormente, podemos ver "using Microsoft.SPOT";

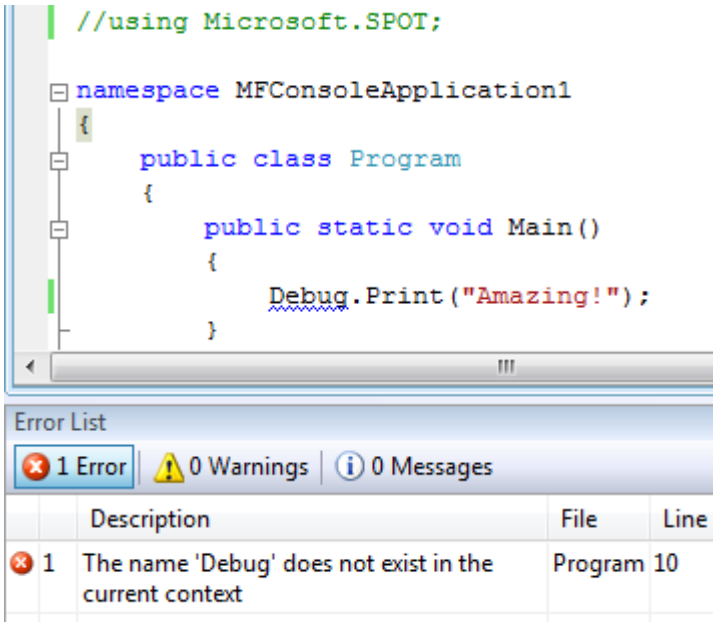
Isso diz a C# que queremos usar o namespace Microsoft.SPOT. Uh ... afinal o que é um namespace ? Os programas são divididos em regiões "espaços". É muito importante quando os programas se tornam mais longos. Cada pedaço de código ou a biblioteca é atribuído um "nome" no seu "espaço". Os programas com o mesmo namespace podem ver um ao outro, mas se for diferente, podemos dizer para o C# para usar (use) o espaço de outros.

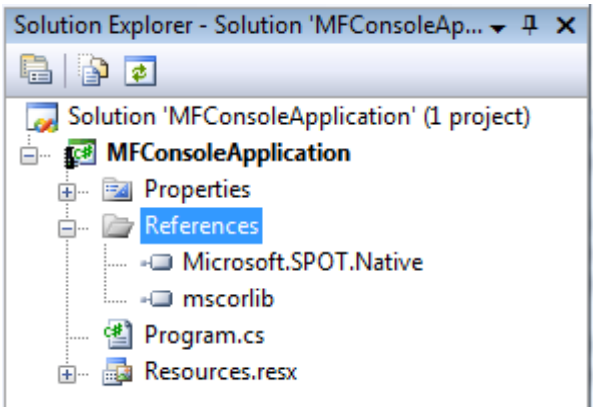
O nome para o espaço do nosso programa é namespace MFConsoleApplication1 Para usar um espaço de nomes diferentes como "Microsoft.SPOT você deve adicionar “using Microsoft.SPOT”;

O que é o SPOT, afinal? Aqui está a história: alguns anos atrás, a Microsoft iniciou um projeto chamado internamente SPOT. Eles perceberam que este projeto foi uma boa idéia e queria oferecer a outros desenvolvedores. Eles então decidiram mudar o nome do produto para .NET Micro Framework, mas mantiveram o mesmo código para compatibilidade com sistemas existentes. Então, basicamente, é NETMF SPOT!

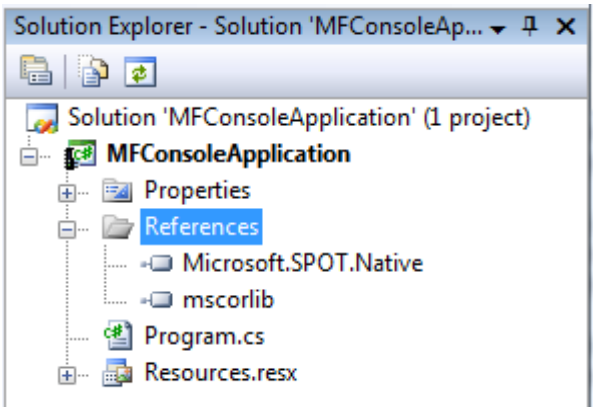
Volte para o código. Tente apagar (ou comentar) “using Microsoft.SPOT”e seu código não vai funcionar.

Aqui está a mensagem de erro recebida após comentar linha “using Microsoft.SPOT”;



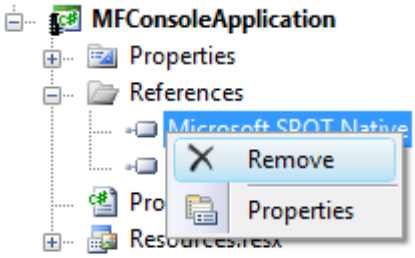


Nós usamos “assemblies”, mas para onde eles foram adicionados?

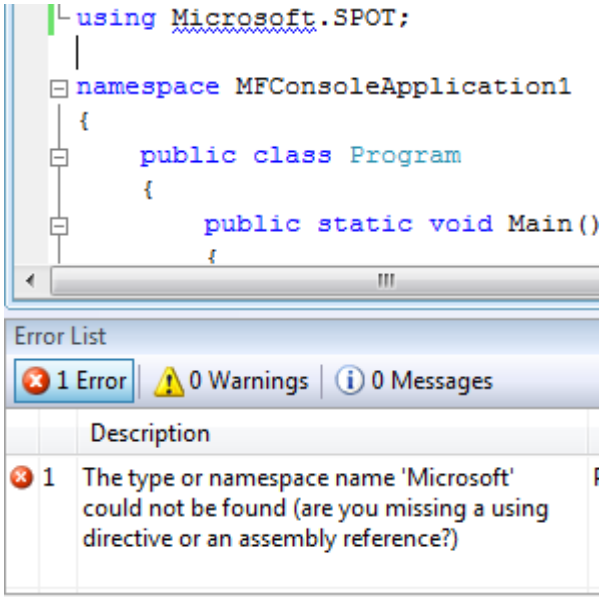


Remova o comentário e certifique-se o programa funciona de novo. Agora olhe para a janela "Solution Explorer" e clique no sinal de adição antes da palavra "References". Você deverá ver dois “assemblies”.

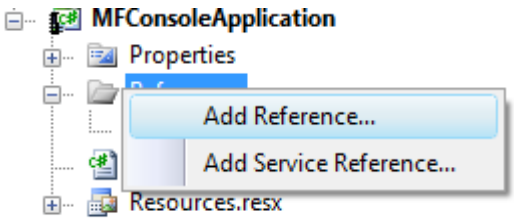
Agora, faça um clique com o botão direito em "Microsoft.SPOT.Native" e "Remove”



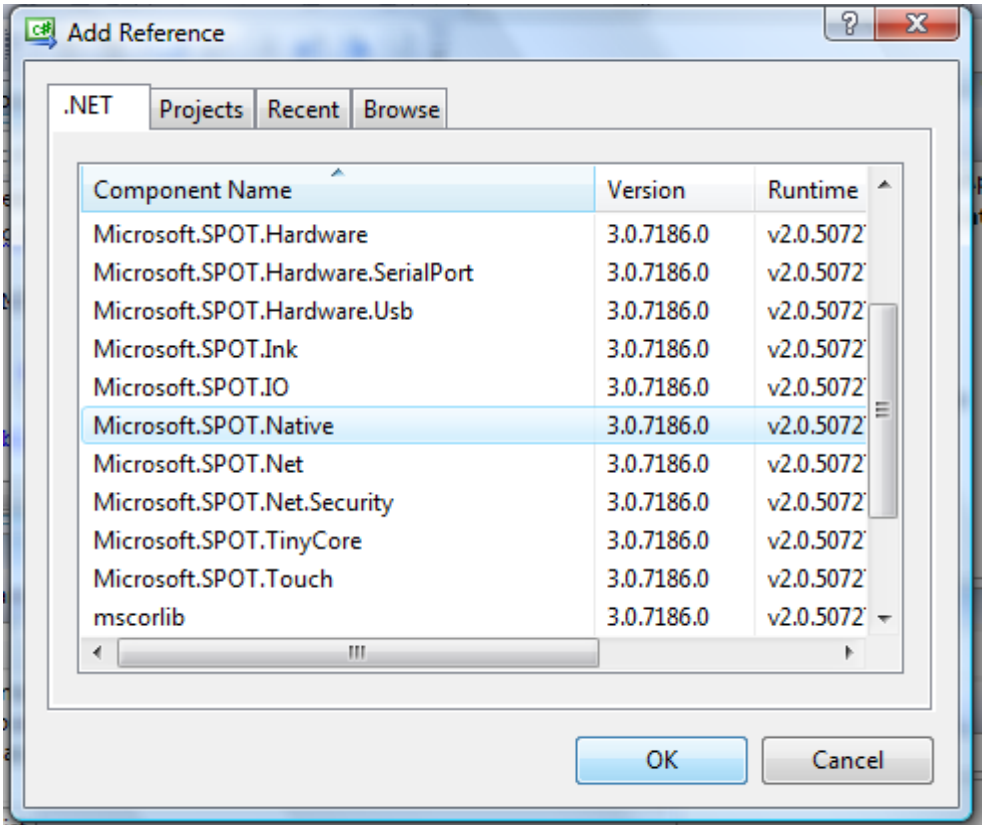
Nosso programa é rigorosamente a mesma de antes, exceto que agora esta faltando um assembly muito importante. Tente lançá-lo e você terá algo como isto:



Volte a adicioná-lo e verifique se o programa ainda funciona. Clock com o botão direito do mouse sobre a palavra "Reference" e selecione "Add Reference ..."



Na nova janela, selecione o guia. NET e escolha "Microsoft.SPOT.Native" e clique em OK.



Experimente o programa, ele deve funcionar. Se você receber erros, volte leia mais e tenha certeza de que seguiu corretamente os passos mencionados antes de prosseguir

Quais Assemblies adicionar ?

Neste livro, ofereço vários exemplos, mas eu não lhe disse que assemblies eu estou utilizando. É realmente fácil de resolver através da documentação mas pode ter dificuldades algumas vezes. Por que não juntar tudo de uma vez? Como iniciante, sua aplicação é ainda muito pequena e você vai ter muita memória, se você pode adicionar todos os assemblies, mesmo que você não usar todos eles.

Os assemblies abaixo são os mais utilizados. Adicione em todos os seus projetos para o momento. Depois de saber o que vai realmente necessitar, pode apagar o que não interessa.

GHIElectronics.NETMF.Hardware

GHIElectronics.NETMF.IO

GHIElectronics.NETMF.System

Microsoft.SPOT.Hardware

Microsoft.SPOT.Native

Microsoft.SPOT.Hardware.SeriaPort

Microsoft.SPOT.IO

mscorlib

System

System.IO

Lembre-se de utilizar um dos seguintes assemblies, dependendo do que hardware você tem. Eles contêm as atribuições dos pinos do processador.

FEZMini_GHIElectronics.NETMF.FEZ
FEZDomino_GHIElectronics.NETMF.FEZ.

Threading

É um assunto muito difícil. Somente as informações básicas serão abordados aqui.

Processadores / programas executam uma instrução por vez. Lembre-se de como nós estávamos indo passo a passo através do código? Uma única instrução foi executada e o fluxo de então passou para a próxima. Então, como é possível que seu computador pode rodar vários programas ao mesmo tempo? Na verdade, o PC não estão executando ao mesmo tempo.! Ele executa cada programa em um curto período de tempo, pára e executa um outro programa após um curto período de tempo.

Em geral, o multi-tasking não é recomendado para iniciantes, mas algumas coisas são mais fáceis utilizando threads. Por exemplo, você quer um flash LED. Seria bom fazer isso em uma thread e você não precisa se preocupar em seu programa principal.

Além disso, para adicionar atrasos no seu programa você precisa do “namespace threading. Você entenderá melhor nos exemplos a seguir.

Aliás, LED significa Light Emitting Diodes. Há LEDs em torno de você. Pegue qualquer TV, DVD ou dispositivo eletrônico e você verá uma pequena luz vermelha (ou outras). Eles são LEDs.

FEZ fornece uma biblioteca dedicada LED para simplificar ao máximo. Este livro explica como controlar diretamente os pinos e hardware.

Adicione "using System.Threading" em seu programa:

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

Isso é tudo que nós precisamos usar threads! É importante saber que o nosso programa em si é uma thread. No início da execução, C# busca por “Main” e executá-lo em uma thread. Nós vamos adicionar um delay em nossa thread (o nosso programa) para escrever

"Amazing!" a cada segundo. Para dar um delay na “thread” nós a colocamos em “Sleep”. Note que o “Sleep” não é para todo o sistema, mas apenas para o “Sleep” na “thread” em questão.

Adicione "Thread.Sleep (1000);

O método "Sleep" usa o tempo em milissegundos. Assim, por um segundo, teremos em 1000 milissegundos.

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

namespace MFConsoleApplication1
{
    public class Program
    {
        public static void Main()
        {
            while (true)
            {
                Debug.Print("Amazing!");
                Thread.Sleep(1000);
            }
        }

    }
}
```

Inicie o programa e olhe a janela de saída. Se você já tentou o emulador e que não deu exatamente 1 segundo o intervalo, não se preocupe. Tente em hardware real (FEZ) e você estará muito próximo de 1 seg.

Vamos criar uma segunda thread (o primeiro foi criado automaticamente, lembra?) Vamos ter que criar um novo handler para a thread e dar-lhe um nome de fantasia como MyThreadHandler, em seguida, criar um novo método local e o chame de MyThread, então rode a nova thread.

Nós não estamos usando a thread "main" (principal) nunca mais, então vamos deixá-la num loop infinito.

Aqui está a listagem do código. Se você não entender tudo, não se preocupe. O objetivo aqui é saber como por em “Sleep” uma thread.

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

```
namespace MFConsoleApplication1
{
    public class Program
    {
        public static void MyThread()
        {
            while (true)
            {
                Debug.Print("Amazing!");
                // Pause esta thread durante 1 segundo
                Thread.Sleep(1000);
            }
        }
        public static void Main()
        {
            // Cria um tratador de Thread
            Thread MyThreadHandler;
            // Cria um novo objeto thread
            // e atribua para meu Handler
            MyThreadHandler = new Thread(MyThread);
            // Inicia minha nova Thread
            MyThreadHandler.Start();

            ////////////////////////////////
            // Faça alguma coisa que você gostaria aqui
            Thread.Sleep(Timeout.Infinite);
        }

    }
}
```

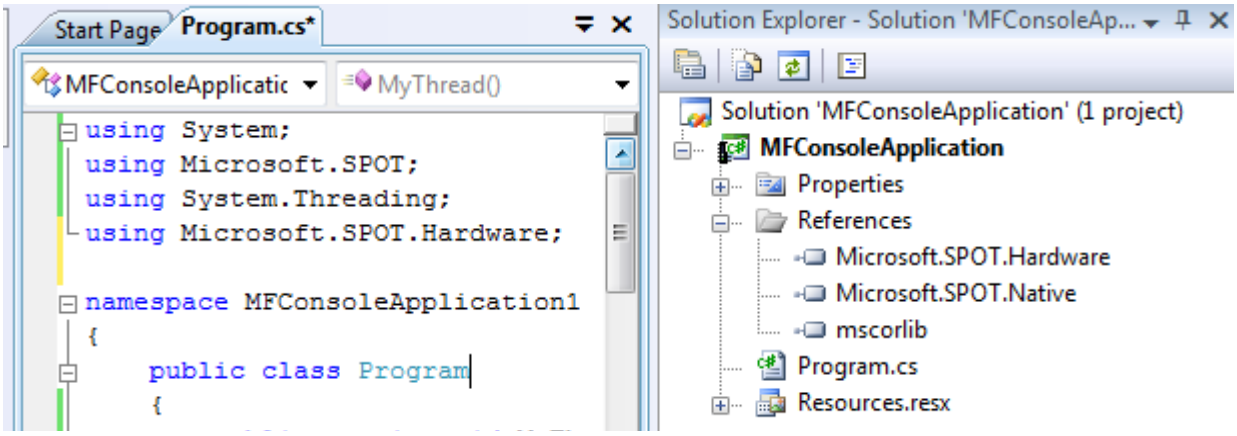
9. Entradas e saídas digitais

Nos processadores há vários pinos 'digitais' que pode ser usados como entrada ou saída. Quando dizemos "digital", isso significa que o estado do pino pode ser 0 ou 1.

Nota Importante: A eletricidade estática (mesmo que do corpo humano) pode danificar o processador. Você já experimentou um choque ao tocar em alguém, às vezes. Este pequeno choque é poderoso e suficiente para queimar circuitos eletrônicos. Os profissionais utilizam equipamentos especiais e adaptados para previnir este problema (aterramento). Você pode não ter este equipamento, de modo que tem que evitar tocar no circuito. Você também pode usar no pulso uma pulseira anti estática.

NETMF suporta E/S digitais através do assembly e namespace "Microsoft.SPOT.Hardware".

Vá em frente e acrescente o assembly e namespace como aprendido anteriormente.



Nós estamos agora prontos para use E/S digitais.

9.1. Saídas Digitais

Sabemos que uma saída digital pode ser definido como 0 ou 1. Note que este não é um volt mas indica que o pino esta suprindo tensão. Se o processador é alimentado por 3.3V, assim, o estado em um pino indica uma tensão de 3,3 V neste pino. Não será exatamente 3,3 V, mas muito perto. Quando o estado do pino é definido como 0, a tensão será muito próximo de 0V.

A corrente de saída destes pinos é muito baixa! Eles não podem alimentar dispositivos que

requerem muita corrente. Por exemplo, um motor pode também alimentado por 3,3V, mas você não pode conectar diretamente a saída digital do processador. Como o processador fornece 3,3 V, mas com muito pouca corrente. O melhor que você pode fazer é alimentar um LED ou sinalizar "1" ou "0" para outro pino.

Todos as placas FEZ têm um LED conectado a uma saída digital. Estamos vamos querer fazer este LED piscar.

As saídas digitais são controladas por um objeto OutputPort. Primeiro criamos uma referência ao objeto (handler) e então criamos um novo objeto OutputPort e o atribuimos para o nosso handler. Quando você cria um novo objeto OutputPort, você deve especificar o estado inicial do pino: 0 ou 1. Zero e um podem ser referidos respectivamente como alto e baixo and tambem pode ser **True para alto e False para baixo**. Nós vamos colocar o pino no estado (true alta) neste exemplo para ligar o LED na inicialização.

Aqui está o código que usa o pino 4 ligado ao LED no mapa FEZ Domino.

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

namespace MFConsoleApplication1
{
    public class Program
    {
        public static void Main()
        {
            OutputPort LED;
            LED = new OutputPort((Cpu.Pin)4, true);

            Thread.Sleep(Timeout.Infinite);
        }

    }
}
```

O SDK FEZ é fornecido com os assemblies "FEZMini_GHIElectronics.NETMF.FEZ" e "FEZDomino_GHIElectronics.NETMF.FEZ". Adicione o assembly apropriado para seu programa e tambem adicione "FEZ_GHIElectronics.NETMF.System".

Agora, modifique o código adicionando "using GHIElectronics.NETMF.FEZ" no início do seu código.

Aqui está o código, desta vez usando a lista de pinos presentes no assembly (pin enumeration class).


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

using Microsoft.SPOT.Hardware;
using GHIElectronics.NETMF.FEZ;

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

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

Viu como é muito fácil ? Nós não temos que se preocupar com o pino onde o LED está ligado.

Inicie o programa e observe o LED. Deve estar aceso agora. Agora as coisas estão ficando interessantes!!!

Piscando um LED

Para fazer um flash no LED, temos que colocar o pino como alto, forçar um delay e colocar o pino para baixo e repetir o delay. É importante incluir um delay ? Por quê? Porque nossos olhos não são rápidos o suficiente para detectar a mudança se o LED é aceso e apagado imediatamente.
O que necessitamos para piscar um LED ? ... aprendemos while, como inserir um atraso, resta-nos saber como colocar o pino no estado Alto/Baixo. Isso é feito chamando o método Write do objeto OutputPort. Observe que você não pode usar diretamente "OutputPort.Write". Isto é errado, que porta de saída você está se referenciando ? Em vez disso, use LED.Write, que faz mais sentido.

Aqui está o código para o flash LED para a placa FEZ Domino / Mini

```
using System;
using Microsoft.SPOT;
using System.Threading;
using Microsoft.SPOT.Hardware;
using GHIElectronics.NETMF.FEZ;
```

```
namespace MFConsoleApplication1
{
    public class Program
    {
        public static void Main()
        {
            OutputPort LED;
            LED = new OutputPort((Cpu.Pin)FEZ_Pin.Digital.LED, true);
            while (true)
            {
                LED.Write(!LED.Read());

                Thread.Sleep(200);
            }
        }
    }
}
```

Aqui tem outra forma, e simples, de se fazer um LED piscar.

```
using System;
using Microsoft.SPOT;
using System.Threading;
using Microsoft.SPOT.Hardware;
using GHIElectronics.NETMF.FEZ;

namespace MFConsoleApplication1
{
    public class Program
    {
        public static void Main()
        {
            OutputPort LED;
            LED = new OutputPort((Cpu.Pin)FEZ_Pin.Digital.LED, true);
            while (true)
            {
                LED.Write(true);
                Thread.Sleep(200);

                LED.Write(false);
                Thread.Sleep(200);
            }
        }
    }
}
```

Tente mudar o valor do Sleep para fazer o LED piscar mais rápido ou mais lento. Tente valores diferentes.

Nota Importante: Nunca ligue duas saídas juntos. Se eles estão ligados e se um está em estado de alta e o outro para o estado baixo, você irá danificar o processador. Sempre ligar uma saída para uma entrada, jogando um sinal ou ligue num LED.

9.2. Entradas Digitais

As entradas digitais detectam se o estado de um pino está alto ou baixo. Há algumas limitações nestes pinos de entradas. Por exemplo, a tensão mínima é de 0V. A tensão negativa pode danificar os pinos do processador. Além disso, a tensão máxima deve ser menor do que a alimentação do processador. Todas as placas da GHI Electronics trabalham com 3.3V, logo a tensão máxima em um pino deve ser 3.3V. Isso é verdade para ChipworkX, mas para Embedded Master e USBizi, são processadores tolerantes a 5V. Isto significa que mesmo se o processador é alimentado por 3,3 V, é capaz de receber entradas de 5V em suas entradas. A maioria dos circuitos digitais são de 5V, logo você pode usar para se comunicar com o processador.

Nota: FEZ baseia-se no chip USBizi e que, portanto, tolera os 5V nas entradas.

Nota Importante: Compatível com 5V, não significa que o processador pode ser alimentado por 5V.

Sempre alimente o processador com 3,3 V. Apenas pinos são tolerantes a 5V.

O objeto InputPort é usado para gerenciar a entrada digital. Todos os pinos do processador GHI podem ser entradas ou saídas, mas, obviamente, não ao mesmo tempo! Entradas que não estão ligadas são chamados de "flutuante". Você poderia pensar que uma entrada que não está ligada equivale ao estado baixo, mas este não é o caso. Quando uma entrada não estiver conectado, ele está aberto a qualquer ruído que possa representar alto/baixo. Para superar isso, os processadores modernos incluem internamente resistores pull-up ou pull-down que geralmente são controladas pelo software. Resistor de pull-up vai puxar o pino para o estado alto. Note que isso não coloca o pino no estado alto, mas sim a força de um estado perto do topo. Se não estiver conectado, o estado irá ler sinal alto.

Há uma abundância de utilizações de portas de entrada, mas o clássico é para ligar um botão ou switch. FEZ já inclui um botão ligado ao pino "Loader". Este pino é usado para ativar o download, mas podemos usar esse pino livremente então. O botão chamado "LDR" ou "Loader".

O botão liga ao terra o pino de entrada. Também vamos conectar o resistor de pull-up. Isso significa que o pino é "alto" (pull up) quando o botão é liberado e "low" (ligados à terra) quando o botão for pressionado.

Vamos ler o estado do botão e encaminhar para o LED. O LED será desligado quando o botão for pressionado.

Código:


```
using System;
using Microsoft.SPOT;
using System.Threading;
using Microsoft.SPOT.Hardware;
using GHIElectronics.NETMF.FEZ;

namespace MFConsoleApplication1
{
    public class Program
    {
        public static void Main()
        {
            OutputPort LED;
            InputPort Button;
            LED = new OutputPort((Cpu.Pin)FEZ_Pin.Digital.LED, true);
            Button = new InputPort((Cpu.Pin)FEZ_Pin.Digital.LDR, false,
Port.ResistorMode.PullUp);
            while (true)
            {
                LED.Write(Button.Read());
                Thread.Sleep(10);
            }
        }
    }
}
```

Criar um objeto InputPort requer um parâmetro adicional para filtro “Glitch”. Isso será explicado mais tarde. Além disso, a forma como passamos o status de uma porta de entrada para setar uma porta como saída pode parecer confuso. Voltaremos na próxima seção.

9.3. Porta de Interrupção

Se quisermos saber o status de um pino, é preciso verificar seu status periodicamente. Esse desperdício de tempo de CPU para uma tarefa não é interessante. Você estará checando o pino, talvez um milhão de vezes antes que o botão seja pressionado! Porta de interrupção permitem-nos escrever um método que será chamado apenas quando o botão for pressionado (pino nível baixo, por exemplo).

Podemos provocar a interrupção na mudanças de estado do pino, de “alto” ou "baixo", por exemplo. O uso comum é "na troca de estado" (cada alteração). A mudança de alto para baixo ou alto par baixo cria um sinal de borda. A borda para alta quando o sinal muda de baixo para alto e a borda de descida quando o sinal muda de alto para baixo.

No exemplo abaixo, eu uso ambas bordas, por isso o nosso método "IntButton_OnInterrupt" será chamado sempre que o estado das mudanças pino.


```
using System;
using Microsoft.SPOT;
using System.Threading;
using Microsoft.SPOT.Hardware;
using GHIElectronics.NETMF.FEZ;

namespace MFConsoleApplication1
{
    public class Program
    {
        static OutputPort LED; // Isto foi movido fora daqui para que pode
ser utilizado por outros  métodos

        public static void Main()
        {
            LED = new OutputPort((Cpu.Pin)FEZ_Pin.Digital.LED, true);
            // Este pino vai gerar uma interrupção na borda de subida e
descida
            InterruptPort IntButton = new
InterruptPort((Cpu.Pin)FEZ_Pin.Interrupt.LDR, true,
                Port.ResistorMode.PullUp,
Port.InterruptMode.InterruptEdgeBoth);

            // Adiciona um tratador de interrupção para o pino
            IntButton.OnInterrupt += new
NativeEventHandler(IntButton_OnInterrupt);

            //Faça alguma coisa que você gostaria aqui
            Thread.Sleep(Timeout.Infinite);
        }

        static void IntButton_OnInterrupt(uint port, uint state, DateTime
time)
        {
            // Muda o estado do led
            LED.Write(state == 0);
        }
    }
}
```

Nota: Nem todos os pinos do processador suportar interrupções, mas a maioria deles sim. Para uma melhor identificação dos pinos de interrupção, use a lista (enumeração) para "interrupt" em vez de "Digital", como consta nos códigos acima.

9.4. Porta Tristate

E se quisermos que um um pino seja entrada e saída, o que podemos fazer? Um pino nunca pode ser ambos simultaneamente, mas nós podemos fazê-lo como saida para escrever alguma coisa e fazê-lo como entrada para ler a resposta de volta.

Uma maneira de fazer seria "Dispor" o pino. Nós fazemos ele como porta de saída, usamos ele e então “dispose” (libere) ele, então fazemos como entrada para lê-lo.

NETMF oferece as melhores opções para isto, através dos portas Tristate. Tristate significa três pinos de estados possíveis: entrada, saída baixa e alta saída. O único problema com estes três pinos de estado é que se um pino está setado como saida e você nomeá-lo novamente como saída, então você receberá uma exceção. A possibilidade de evitar isso é para verificar a direção do pino antes da mudança. A direção do pino é de sua propriedade "Active" onde falso significa entrada e verdadeiro significa saída. Pessoalmente, eu não recomendo o uso de portas tristate, a não ser que seja absolutamente necessário.

```
using System;
using Microsoft.SPOT;
using System.Threading;
using Microsoft.SPOT.Hardware;
using GHIElectronics.NETMF.FEZ;

namespace MFConsoleApplication1
{
    public class Program
    {
        static void MakePinOutput(TristatePort port)
        {
            if (port.Active == false)
                port.Active = true;
        }
        static void MakePinInput(TristatePort port)
        {
            if (port.Active == true)
                port.Active = false;
        }
        public static void Main()
        {
               TristatePort TriPin = new
TristatePort((Cpu.Pin)FEZ_Pin.Interrupt.LDR, false, false,
Port.ResistorMode.PullUp);
            MakePinOutput(TriPin);// faça o pino como saída
            TriPin.Write(true);
            MakePinInput(TriPin);// faça o pino como entrada
            Debug.Print(TriPin.Read().ToString());
        }
    }
}
```

Nota: Por padrão, TristatePort só funcionam com o pinos que podem gerar interrupções.

Nota Importante: Tenha cuidado para não ter um pino ligado a uma chave e a porta estiver configurada para saida. Isso irá danificar o processador. Eu diria para iniciantes em

aplicações, você não precisa da porta tristate, portanto, não a use se você se sente confortável com circuitos digitais.

10. C# Nível 2

10.1. Variáveis Booleanas

Nós aprendemos como variáveis inteiras podem conter números. Em contraste, as variáveis Boolean podem somente ser true ou false (verdadeiro ou falso). A luz pode ser estar acesa ou apagada, representando isto com um tipo inteiro não faz sentido, mas usando boolean é o estado verdadeiro para ligado e estado falso para desligado. Nós já usamos esta variável para alterar o estado do pino de saída digital como alto ou baixo, LED.Write(true);

Para armazenar o valor de um botão em uma variável, usamos:

```
bool button_state;
button_state = Button.Read();
```

Nós também usamos o laço while e solicitamos que entrasse em um loop infinito, quando nós usamos True no parâmetro.

```
while (true)
{
    //código aqui
}
```

Veja as últimas linhas de código que escrevi e modifique-o para usar um boolean para torná-lo mais fácil de ler. Ao invés de passar o estado do botão diretamente para o LED, nós lemos o estado de botão em button_state (boolean) então nós passamos essa variável para o LED.

```
using System;
using Microsoft.SPOT;
using System.Threading;
using Microsoft.SPOT.Hardware;
using GHIElectronics.NETMF.FEZ;

namespace MFConsoleApplication1
{
    public class Program
    {
        public static void Main()
        {
            OutputPort LED;
            InputPort Button;
```

```
            bool button_state;
            LED = new OutputPort((Cpu.Pin)FEZ_Pin.Digital.LED, true);
            Button = new InputPort((Cpu.Pin)FEZ_Pin.Digital.LDR, false,
Port.ResistorMode.PullUp);
            while (true)
            {
                button_state = Button.Read();
                LED.Write(button_state);
                Thread.Sleep(10);
            }
        }
    }
}
```

Pode você fazer um LED piscar no momento que uma tecla é pressionada ?

```
using System;
using Microsoft.SPOT;
using System.Threading;
using Microsoft.SPOT.Hardware;
using GHIElectronics.NETMF.FEZ;

namespace MFConsoleApplication1
{
    public class Program
    {
        public static void Main()
        {
            OutputPort LED;
            InputPort Button;

            LED = new OutputPort((Cpu.Pin)FEZ_Pin.Digital.LED, true);
            Button = new InputPort((Cpu.Pin)FEZ_Pin.Digital.LDR, false,
Port.ResistorMode.PullUp);
            while (true)
            {
                while (Button.Read() == false) //Botão é falso quando
pressionado
                {
                    LED.Write(true);
                    Thread.Sleep(300);
                    LED.Write(false);
                    Thread.Sleep(300);
                }
            }
        }
    }
}
```

Nota Importante: O símbolo "==" é usado em C# para testar igualdade, enquanto o sinal "=" é usado para atribuir um valor.

10.2. Comando if

Uma parte importante do programa é verificar algum estado e tomar uma ação de acordo. Por exemplo, "se a temperatura ultrapassar 35 graus, ligue o ventilador".

Para testar o If com nosso programa, vamos ligar o LED "If" o botão for pressionado. Note que, ao contrário do que nós tinhamos antes. No programa, o estado do botão é low quando pressionado. Então, para isso, vamos inverter o estado do LED em comparação com o estado do botão. Se o botão for pressionado (baixo), então a luz do LED acende (alta). Isto deve ser verificado regularmente, por isso vamos fazê-lo cada 10ms.

```
using System;
using Microsoft.SPOT;
using System.Threading;
using Microsoft.SPOT.Hardware;
using GHIElectronics.NETMF.FEZ;

namespace MFConsoleApplication1
{
    public class Program
    {
        public static void Main()
        {
            OutputPort LED;
            InputPort Button;
            bool button_state;
            LED = new OutputPort((Cpu.Pin)FEZ_Pin.Digital.LED, true);
            Button = new InputPort((Cpu.Pin)FEZ_Pin.Digital.LDR, false,
Port.ResistorMode.PullUp);
            while (true)
            {
                button_state = Button.Read();

                if (button_state == true)
                {
                    LED.Write(false);
                }

                if (button_state == false)
                {
                    LED.Write(true);
                }

                Thread.Sleep(10);
            }
```

```
        }
    }
}
```

10.3. Comando if-else

Nos vimos como o "if" funciona. Agora vamos usar a palavra chave "else". Simplesmente, se um "if" é verdade, então o código associado é executado ou "else" (caso contrário) o código do "else" é executada. Com esta nova palavra-chave, podemos otimizar nosso código:

```
using System;
using Microsoft.SPOT;
using System.Threading;
using Microsoft.SPOT.Hardware;
using GHIElectronics.NETMF.FEZ;

namespace MFConsoleApplication1
{
    public class Program
    {
        public static void Main()
        {
            OutputPort LED;
            InputPort Button;
            bool button_state;
            LED = new OutputPort((Cpu.Pin)FEZ_Pin.Digital.LED, true);
            Button = new InputPort((Cpu.Pin)FEZ_Pin.Digital.LDR, false,
Port.ResistorMode.PullUp);
            while (true)
            {
                button_state = Button.Read();

                if (button_state == true)
                {
                    LED.Write(false);
                }
                else
                {
                    LED.Write(true);
                }

                Thread.Sleep(10);
            }
        }
    }
}
```

Vou lhe contar um segredo! Foram utilizados os comandos if e if-else neste exemplo para

propósitos de demonstração. Nós podemos escrever o mesmo código como este:

```
using System;
using Microsoft.SPOT;
using System.Threading;
using Microsoft.SPOT.Hardware;
using GHIElectronics.NETMF.FEZ;

namespace MFConsoleApplication1
{
    public class Program
    {
        public static void Main()
        {
            OutputPort LED;
            InputPort Button;

            LED = new OutputPort((Cpu.Pin)FEZ_Pin.Digital.LED, true);
            Button = new InputPort((Cpu.Pin)FEZ_Pin.Digital.LDR, false,
Port.ResistorMode.PullUp);
            while (true)
            {
                LED.Write(Button.Read() == false);
                Thread.Sleep(10);
            }
        }
    }
}
```

Ou mesmo assim!

```
using System;
using Microsoft.SPOT;
using System.Threading;
using Microsoft.SPOT.Hardware;
using GHIElectronics.NETMF.FEZ;

namespace MFConsoleApplication1
{
    public class Program
    {
        public static void Main()
        {
            OutputPort LED;
            InputPort Button;

            LED = new OutputPort((Cpu.Pin)FEZ_Pin.Digital.LED, true);
            Button = new InputPort((Cpu.Pin)FEZ_Pin.Digital.LDR, false,
Port.ResistorMode.PullUp);
```

```
            while (true)
            {
                LED.Write(!Button.Read());
                Thread.Sleep(10);
            }
        }
    }
}
```

Em geral, existem várias maneiras de escrever código. Use o que você acha que é mais fácil de começar, e com a experiência você aprenderá a otimizar seu código..

10.4. Métodos e Argumentos

Os métodos são as ações executadas por um objeto. Eles também são chamados de "funções" de um objeto. Nós já vimos e utilizamos. Lembre-se o objeto Debug com o seu método Print. Temos usado frequentemente Debug.Print, a qual temos dado uma string para ele mostrar. Esta "cadeia" passada é chamado de "argumento"

Os métodos podem ter um ou mais argumentos opcionais, mas podem retornar apenas um valor opcional.

O método Print do objeto Debug recebe um único argumento. Outros métodos não necessitam de argumentos ou tem vários. Por exemplo, um método que poderia desenhar um círculo poderia ter quatro argumentos: Circle (X , Y, Diâmetro, Cor). Um exemplo de um método que retorna um valor é um método retorna uma temperatura.

Até agora, nós aprendemos três tipos de variáveis: int, string e boolen. Veremos outros tipos mais tarde, mas lembre-se que tudo o que dizemos aqui também se aplica aos outros tipos.

O valor de retorno é opcional. Se nenhum valor é retornado quando você usa o tipo "void". A palavra-chave "return" é usado para retornar valores no fim de um método.

Aqui está um método simples que retorna a soma de dois números inteiros:

```
int Add(int var1, int var2)
{
    int var3;
    var3 = var1 + var2;
    return var3;
}
```

Nós começamos o método indicando o tipo do valor de retorno (int), seguido do nome do método. Então nós temos a lista de argumentos. Os argumentos são agrupadas entre parênteses e separados por vírgula.

Dentro do método Add, uma variável inteira local foi declarado: var3. As variáveis locais só existem no método no qual eles são declarados. Em seguida, some os nossos dois argumentos e, finalmente, retorne o resultado.

Como retornar uma string representando a soma de dois números? Lembre-se que uma string contendo "123" não é o mesmo como um inteiro 123. Um número inteiro é um número e string é uma seqüência representa um texto. Para os seres humanos é a mesma de um computador são duas coisas totalmente diferentes

Aqui está o código para retornar uma string:

```
string Add(int var1, int var2)
{
    int var3;
    var3 = var1 + var2;

    string MyString;
    MyString = var3.ToString();

    return MyString;
}
```

Podemos ver como o tipo retornado troca para string. Nós não podemos retornar var3 porque é uma variável inteira e então tivemos de converter para uma string. Para isso, criamos uma nova variável objeto chamada MyString. Então converte var3 “ToString()” e ponha a nova string em MyString.

A pergunta agora é como é possível chamar um método "ToString()" em uma variável inteira? Na verdade, em C#, tudo é representado por um objeto, mesmo os tipos de variáveis. Este livro não entrará nesses detalhes porque é destinado para os iniciantes.

Tudo o que foi escrito antes foi detalhado para melhor poder explicar, mas você pode escrever do seu jeito e compactar.

```
string Add(int var1, int var2)
{
    return (var1+var2).ToString();
}
```

Eu não aconselho você a escrever código de forma compacta se você não está familiarizado com a linguagem. E mesmo que haja certos limites devem ser respeitados, se queremos que o código permaneça legível ou é re-utilizado por outra pessoa facilmente.

10.5. Classes

Todas as coisas que nós discutimos até agora são feitos de "classes" em C #. Em modernas

linguagens orientadas a objetos, tudo é um objeto e os métodos pertencem a um objeto. Isto permite ter métodos com nomes idênticos, mas cujas acções são totalmente diferentes. O homem pode "andar" e um gato também pode "andar", só que não da mesma maneira. Se você escreve um método “ande" em C#, então não é muito claro se o homem ou gato. A menos que você está usando: homem.ande ou gato.ande.

A criação de classes não é o propósito deste livro, mas aqui é uma classe simples para começar.

```
class MyClass
{
    int Add(int a, int b)
    {
        return a + b;
    }
}
```

10.6. Publico e Privado

Os métodos podem ser privados a uma classe ou publicamente acessível. Isso é útil simplesmente para forçar os objetos mais robustos. Se você criar um objeto (classe) que tem métodos que você não quer que o programador use externamente, basta adicionar a palavra "private" antes do tipo de retorno do método. Caso contrário, coloque a palavra "public".

Aqui está um exemplo:

```
class MyClass
{
    public int Add(int a, int b)
    {
        // Este objeto pode usar métodos privados
        // somente dentro da classe
        DoSomething();
        return a + b;
    }
    private void DoSomething()
    {
    }
}
```

10.7. Estático e não estático

Alguns objetos na vida têm várias instâncias, enquanto outros existem apenas uma vez. Por

exemplo, um objeto que represente um ser humano não significa muito. Você precisa criar uma "instância" do objeto para representar um ser humano. Você terá algo como

humano Mike;

Agora temos uma referência chamado Mike, do tipo humano. É importante notar que esta referência neste momento não está refenciando objeto algum (sem instancia atribuida), então a referência é NULL

Para realmente criar o exemplo, usando a palavra "new" na referência especificando a classe do objeto:

Mike = new humano();

Podemos agora usar qualquer um dos métodos "humano" da instancia de Mike

Mike.Corra (distancia);

Mike.Coma();

Hungry bool Mike.EstaFaminto();

Nós já usamos esses métodos não estáticos quando verificamos os pinos de E/S.

Quando criamos um novo objeto não estático, a palavra "new" é usado para chamar o construtor do objeto. O construtor é um tipo especial de método que não retorna nenhum valor e exclusivamente durante a construção /criação de novos objetos.

Os métodos estáticos são mais fáceis de gerir, porque se referem a um único objeto sem a criação de uma instância. Um exemplo típico é o nosso objeto Debug: não há um único objeto no Debug NETMF, assim, para utilizar os seus métodos são chamados diretamente do objeto:

Debug.Print ("string");

Eu talvez não usei as definições exatas para "estático" e "instância", mas eu queria descrever a mais simples possível.

10.8. Constantes

Algumas variáveis podem ter valores que nunca mudam. Para se protegerem de um possível erro de programação que iria mudar esses valores, basta adicionar a palavra-chave "const" para o tipo da variável na sua declaração:

```
const int hours_in_one_day = 24;
```

10.9. Enumeration (Enumeração)

Listas são similares às constantes. Suponha que temos um dispositivo que aceita quatro comandos: MOVE, STOP, LEFT e RIGHT. Este dispositivo não é um ser humano, portanto, esses comandos são realmente números. Nós podemos criar constantes (variáveis) para estes quatro comandos:

```
const int MOVE = 1;
const int STOP = 2;
const int RIGHT = 3;
const int LEFT = 4;
//Agora nós podemos enviar um comando
SendCommand(MOVE);
SendCommand(STOP);
```

Os nomes de constantes são maiúsculas por convenção. Qualquer programador vendo uma variável maiúscula sabe que se trata de uma constante.

O código acima esta OK e irá funcionar, mas seria ainda melhor se nós agrupassemos estes comandos:

```
enum Command
{
    MOVE = 1,
    STOP = 2,
    RIGHT = 3,
    LEFT = 4,
}
//Agora nós podemos enviar um comando
SendCommand(Command.LEFT);
SendCommand(Command.STOP);
```

Com esta nova abordagem, não há necessidade de lembrar quais comandos existem e quais os números dos comandos. Uma vez a palavra “Command” é digitado, Visual Studio dará a você uma lista de comandos disponíveis.

C# também é esperto suficiente para incrementar os números das enumerações a cada nova ordem. Assim, o código abaixo funciona exatamente da mesma maneira.

```
enum Command
{
    MOVE = 1,
    STOP ,
    RIGHT,
    LEFT ,
}
```



```
//Agora nós podemos enviar um comando
SendCommand(Command.LEFT);
SendCommand(Command.STOP);
```

11. Assembly/Firmware Correspondente

Os dispositivos NETMF incluem funções geralmente extendidas. Essas funções extendidas exigem assembly/biblioteca adicional para que um projeto de C# possa usá-los. Por exemplo, NETMF não suporta pinos analógicos, mas FEZ e outros dispositivos da GHI suportam a utilização dos pinos analógicos, de modo que você deve adicionar um assembly/library fornecido pelo GHI e ter acesso a essas novas oportunidades.

Nota Importante: O firmware não vai funcionar se o assembly/library não coincide com a versão do firmware instalada.

É um problema muito comum que os usuários encontram quando ele não atualiza o firmware e o aplicativo não está funcionando.

Veja o que acontece:

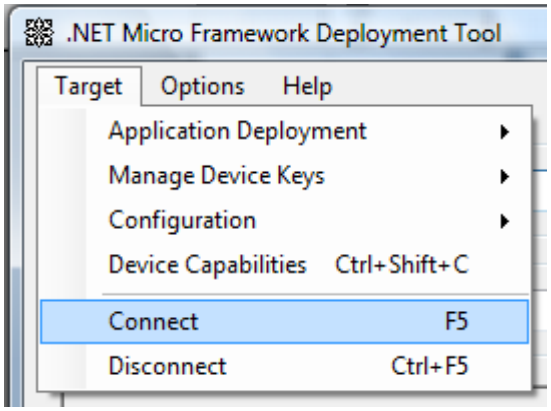
Cenário # 1: Um desenvolvedor recebe novos dispositivo. Tem a versão do firmware 1.0. Então o desenvolvedor baixa do site da GHI a ultima versão doSDK, que é a versão 1.1. Quando você tenta carregar o programa, VS2010 não consegue conectar-se ao dispositivo e não diz porquê! O desenvolvedor então acha que seu equipamento não funciona. Na verdade, o produto funciona bem, mas neste caso, o firmware 1.0 e o conjunto é 1.1, então não vai. Para resolver esse problema, é preciso atualizar o firmware.

Cenário # 2: Um desenvolvedor tem um material que funciona bem com o firmware 2.1. Um novo SDK 2.2 liberado, então o usuário faz a atualizações deste SDK e instala a nova versão do Firmware na placa FEZ. Quando ele reinicia a sua máquina, ele não funciona! Porque o programa no processador ainda está na versão 2.1. Para resolver esse problema, abra o projeto e retire os assembly/library específicos do Hardware e adicioná-los novamente. Assim, os novos arquivos serão buscamos do SDK.

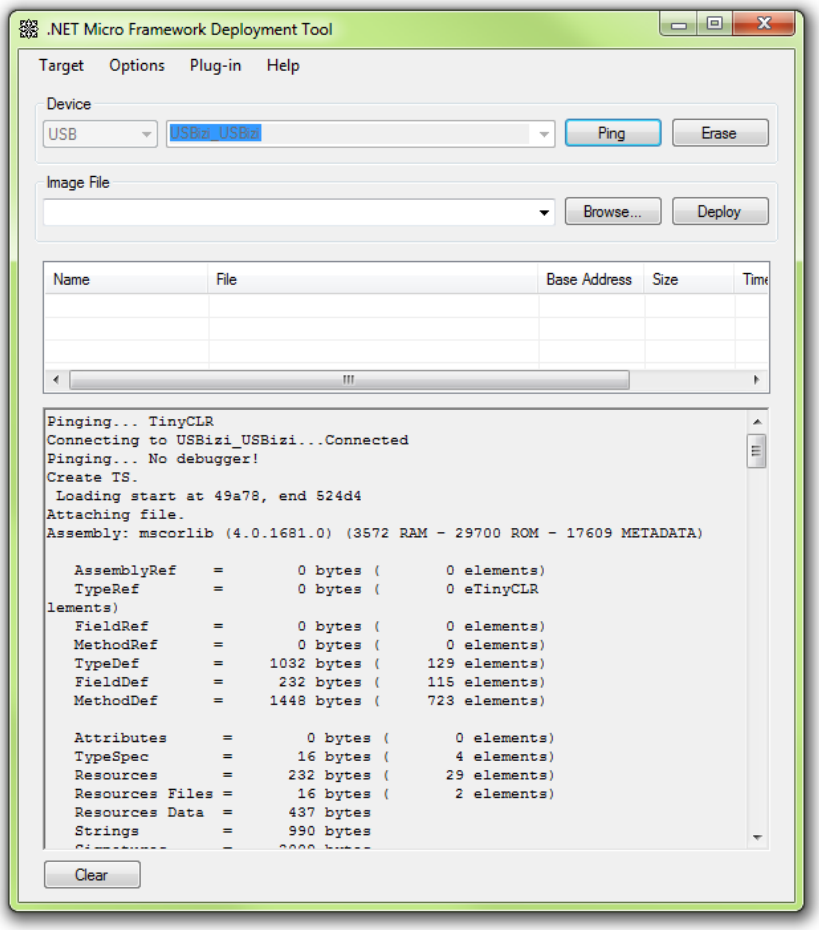
Mensagens durante o Boot-up

Nós podemos facilmente ver porque o sistema não está rodando usando o MFDeploy. NETMF apresenta uma série de mensagens úteis durante a inicialização do sistema. Pode o sistema travar, falhar na execução ou no debug, nós podemos usar MFDeploy para mostrar estas mensagens durante a inicialização. Também todas mensagens “Debug.Print” que são normalmente vistas na janela de saida, podem ser vistos no MFDeploy

Para ver estas mensagens, clique em "Target-> Connect" do menu e pressione o botão reset do Hardware. Imediatamente click em "ping". MFDeploy vai travar por um momento e mostrar uma longa lista de mensagens.



Observe que no FEZ Domino, o botão de reset está disponivel na placa. Para FEZ Mini, você tem que conectar uma chave de reset ou você tem um FEZ Mini no starter kit ou kit de robo, você pode usar o botão de reset dele.



12. Modulação por largura de Pulso(PWM)

PWM é uma forma de controlar a potência fornecida para um dispositivo. Para mudar o brilho de um LED, ou diminuir ou aumentar a velocidade de um motor é mais fácil com a utilização de um sinal PWM. Se aplicarmos uma tensão a um LED, que acende-se completamente e se cortarmos, o LED se apaga. Mas o que poderia acontecer se nós acendermos o LED por durante 1ms e apagarmos durante 1ms ? Será tão rápido que nossos olhos não conseguirão capturar o piscar do LED, por causa da frequência.

Podemos calculá-la muito facilmente: (on / (off + a)) x100 = 50%. Vemos assim que o LED fica apenas metade da potência.

E se ligarmos o LED por 1ms e desligá-lo por 9ms, então ele recebe (1 / (1 +9)) x100 = 10% de energia, ficando com brilho menor.

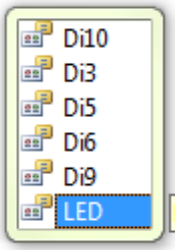
Um sinal PWM é muito simples de se gerar, mas se fosse para mudar o status de um pino várias centenas ou milhares de vezes por segundo, se perderia muito tempo para o processador. Existem placas que geram sinais PWM de forma mais eficiente e muitos processadores incluem circuitos capazes de gerar tais sinais PWM em hardware. Isso significa que podemos configurar nossa placa gerar automaticamente um sinal PWM sem termos que nos preocupar depois.

FEZ dispõe de vários pinos que podem ser usados para gerar um sinal PWM. FEZ inclui bibliotecas que ativam PWM nestes pinos.Aqui está um exemplo que cria um objeto PWM com uma freqüência de 10 KHz (10.000 pulsos / seg) e um “duty cicle” de 50 (50% energia).

```
PWM pwm = new PWM((PWM.Pin) FEZ_Pin.PWM.LED);
pwm.Set(10000, 50);
```

FEZ inclui uma lista para saber quais os pinos podem gerar um sinal PWM. Usando essa enumeração PWM pode facilmente achar esta lista, o Visual Studio dá-lhe a lista enquanto você está digitando o código, conforme imagem.


```
{
    PWM pwm = new PWM((byte)FEZ_Pin.PWM.);
    pwm.Set(10000, 50);

}
```



Na listagem abaixo mostra que o LED da placa é conectada a um pino PWM. E se em vez de fazer flash no LED, que tal aumentar e diminuir o brilho ? Seria muito bom! Aqui está o código:

```
using System;
using Microsoft.SPOT;
using System.Threading;
using Microsoft.SPOT.Hardware;
using GHIElectronics.NETMF.Hardware;
using GHIElectronics.NETMF.FEZ;

namespace MFConsoleApplication1
{
    public class Program
    {
        public static void Main()
        {
            sbyte dirr = 1;
            byte duty = 40;
            PWM pwm = new PWM((PWM.Pin)FEZ_Pin.PWM.LED);
            while (true)
            {
                pwm.Set(10000, duty);
                duty = (byte)(duty + dirr);
                if (duty > 90 || duty < 10)
                {
                    dirr *= -1;
                }

                Thread.Sleep(10);
            }

        }
    }
}
```

Este código aumenta a potência fornecida ao LED até 90% e diminui para 10%, e novamente em direção oposta.

I**mportante**: Um sinal PWM é perfeito para controlar a velocidade do movimento de um robô ou um ventilador girando. Um pino de PWM pode facilmente alimentar um LED. Este não é o caso quando se quer controlar um dispositivo que necessite de corrente, como um motor ou uma lâmpada normal. Neste caso, use o sinal de PWM para controlar outro circuito, que vai gerar a potência fornecida ao dispositivo. Por exemplo, ponte H é frequentemente utilizado para controlar a direção e velocidade dos motores.

Importante: Todos os sinais PWM material compartilham o mesmo clock. A mudança na freqüência em um pino vai mudar a freqüência em todos os outros. Mas o “duty cicle” é independente.

Simulando um sinal PWM

Os pinos PWM são controlados dentro do processador através de circuitos dedicados. Isso significa que esses circuitos irão chavear os pinos, não você. O processador requer apenas setar alguns registros de modo que nenhuma interação é necessária para gerar o sinal PWM.

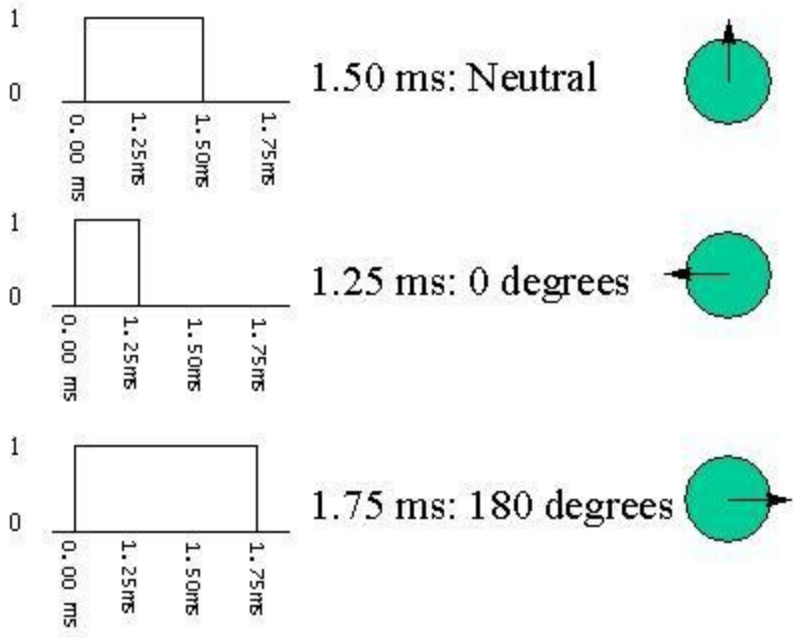
As placas da GHI oferecem classe NETMF OutputCompare. Através desta classe, o usuário pode gerar praticamente qualquer sinal, incluind PWM. Note que isto é feito em software, por isso é melhor usar a classe PWM, se possível. Por quê? Quando usamos a classe PWM, o sinal é gerado pelo hardware do processador e não tem nada a fazer. Por outro lado, quando você usa OutputCompare para gerar o PWM, o processador deve manter o controle do passado e alterar o estado dos pinos em intervalos constantes.

O exemplo de servo motor fornecido no www.TinyCLR.com usa OutputCompare. A vantagem aqui é que você pode usar qualquer pino de controle do servo. É melhor usar PWM para controlar o servo, mas neste caso você está limitado a especificar o pino que tenha PWM.

Controlando Servo Motor e ajustando PWM

Vimos como controlar a potência com PWM. Há outros bons usos para PWM, como controlar servos. A classe PWM fornece dois métodos, Set e SetPulse. Antes, nós usamos PWM.Set o qual é bom para especificar a energia (potência). Para controlar os servos, SetPulse é mais adequado.

Como o servos funcionam? Veja aqui: www.pc-control.co.uk



Você pode então controlar a posição de um servo através do envio de um pulso. Se o pulso dura 1,25 ms, então o servo vai a 0°. Se este for aumentada para 1,50 ms o servo vai para 90° (posição neutra). Um pulso de 1,75 ms vai enviá-lo para 180°. Bem, é fácil gerar o sinal PWM, mas ainda precisamos de uma informação: a imagem mostra a duração de um pulso "alto" mas sobre o pulso "baixo"? Os servos estão à espera de um pulso a cada 20 a 30ms. Agora nós temos tudo que precisamos. Antes de começar, vou explicar o que o "período": é a soma to tempo alto com o tempo baixo. Agora estamos prontos para codificação.

O método PWM.SetPulse necessita do tempo alto e o período do sinal. Desde que este período low no servo não é crítico (entre 20 e 30ms), vou setar para 20ms. O importante é que o pulso alto deve estar entre 1,25 ms e 1,75 ms.Uma última coisa:

SetPulse aceita valores em nanossegundos, mas temos tempo em milissegundos. 1 ms = 1000000 ns.

Finalmente, aqui está o código:


```
using System;
using System.Threading;
using Microsoft.SPOT;
using GHIElectronics.NETMF.Hardware;
using GHIElectronics.NETMF.FEZ;
class Program
{
    public static void Main()
    {
        PWM servo = new PWM((PWM.Pin)FEZ_Pin.PWM.Di5);
        while (true)
        {
            // 0 graus. Período de  20ms e 1.25ms de pulso alto
            servo.SetPulse(20 * 1000 * 1000, 1250 * 1000);
            Thread.Sleep(1000);//wait for a second

            // 90 graus. Período de  20ms e 1.50ms de pulso alto
            servo.SetPulse(20 * 1000 * 1000, 1500 * 1000);
            Thread.Sleep(1000);//wait for a second

            // 180 graus. Período de  20ms e 1.75ms de pulso alto
            servo.SetPulse(20 * 1000 * 1000, 1750 * 1000);
            Thread.Sleep(1000);//Espera por um segundo
        }
        Thread.Sleep(-1);
    }
}
```

12.1. Piezo

Os alto-falantes piezo elétricos são uma maneira barata de adicionar um som para o seu equipamento. Eles emitem um som quando estão aplicando uma determinada frequência. Um sinal PWM é perfeito para gerar essa frequência e pode facilmente ativar um piezo. Usaremos um ciclo de trabalho de 50%, mas vamos mudar a freqüência para ter diferentes tons.

Quando você conectar um piezo, preste atenção à polaridade: + vai PWM e - para terra.

Olhe para o componente FEZ PIEZO em www.TinyCLR.com

Este é um projeto que decodifica um arquivo MIDI de um cartão SD e toca no Piezo

http://www.microframeworkprojects.com/index.php?title=PWM_MIDI_Player

13. Filtro de Glitch (Ruído)

Quando usamos o botão tempo atrás, nós confirguramos de tal modo que quando pressionado , o pino vai a nível baixo, caso contrário vai a alto. Quando você pressionar o botão, ele pode dar “picos”,em outras palavras, o botão gera alguns estados alto/baixo são gerados antes de se estabilizar. Parece, portanto, que o botão foi pressionado várias vezes. Estes saltos em um tempo muito curto. Para eliminá-los em nível de hardware, podemos colocar um capacitor entre o pino eo terra. Para gerenciar isto por software, podemos medir o tempo entre estes pulsos e se for muito curto, podemos descartar. Um usuário pode pressionar um botão aproximadamente a cada 200ms. Portanto, se a diferença é de 10ms entre pressionadas, sabemos que este não é um ser humano quem fez isso e portanto, podem ser filtrados.

Felizmente, NETMF inclui um filtro interno, então não se preocupe com isso. Quando criamos um pino de entrada, temos a possibilidade de ativar o filtro de Glitch. O segundo argumento da criação de um pino de entrada é um booleano que indica se o filtro deve ser ativado (true) ou não (false). Ao usar os pinos com botões / switches, é melhor ativar.

```
Button = new InputPort((Cpu.Pin)FEZ_Pin.Digital.LDR, true,
Port.ResistorMode.PullUp);
```

Você pode alterar a sensibilidade do filtro de retorno da seguinte forma:

```
TimeSpan ts = new TimeSpan(0, 0, 0, 0, 200);//200ms
Microsoft.SPOT.Hardware.Cpu.GlitchFilterTime = ts;
```

Nota importante: Filtro de Glitch funciona somente nos pinos capazes de gerar interrupção. Se você tentar usar InputPort com filtro setado para true e você ver uma exceção, então muito provavelmente você está usando um pino que não é capaz de gerar interrupção.

14. Entradas e saídas analógicas

Os pinos analógicos são multiplexadas com pinos digitais. Alguns pinos do processador podem ser digitais e analógicos mas não ao mesmo tempo.

14.1. Entradas analógicas

As entradas digitais só podem ler valores baixo ou alto (0 ou 1), mas os pinos analógicos podem medir a tensão analógica. Existem limitações quanto ao tensões aplicadas às entradas analógicas. Por exemplo, as entradas analógicas FEZ pode ler uma tensão entre 0 e 3,3V. Quando ao pino digital, ele tolera 5V, mas se ele estiver configurado como analógico, 3,3V é o máximo. Esta limitação não é realmente um problema porque a maioria dos sinais analógicos são condicionados para trabalhar com esta entrada analógica. Um divisor de tensão ou circuito opamp pode ser usado para obter uma parcela do sinal. Por exemplo, se quisermos medir a tensão da bateria, 6V, ou seja, então vamos cortar pela metade a tensão com resistências por exemplo, para o pino só pega metade da tensão que é 3V. No software, sabemos que a leitura da tensão é dividida por dois, basta multiplicar por 2 para obter a real medida.

Felizmente, a implementação da GHI para entradas analógicas manuseia condicionamento do sinal (escala). Ao criar um novo objeto Analog Input, você pode especificar a opção de escala.

A referência interna é 0V a 3.3V, assim que qualquer medida tem que considerar isto.

A maneira mais fácil é colocar a escala de 0-3300. Podemos, portanto, pensar em milivolts. Se nós lermos 1000, então a tensão de entrada é de 1V.

```
AnalogIn BatteryVoltage = new AnalogIn((AnalogIn.Pin)FEZ_Pin.AnalogIn.An0);
BatteryVoltage.SetLinearScale(0, 3300);
int voltage = BatteryVoltage.Read();
```

Considere usar a lista para ver quais os pinos podem ser analógicos.

Para mostrar a leitura da tensão em volts, converter para Volts e depois converter para uma string.


```
AnalogIn BatteryVoltage = new AnalogIn((AnalogIn.Pin) FEZ_Pin.AnalogIn.An0);
BatteryVoltage.SetLinearScale(0, 3300);
int voltage = BatteryVoltage.Read();
Debug.Print("Voltage = " + (voltage / 1000).ToString() + "." + (voltage %
1000).ToString());
```

Nos dividimos por 1000 para obter a tensão e então nos usamos os módulo para obter a fração.

14.2. Sáidas Analógicas

As saídas analógicas são semelhantes às saídas digitais visto que elas também têm limites de quanta potência pode proporcionar e ainda são menores do que as saídas digitais. Eles são capazes de fornecer um pequeno sinal para controlar outro circuito ou possivelmente um amplificador de potência.

As saídas digitais podem ser alto ou baixo estado, enquanto as saídas analógicas podem fornecer uma tensão entre 0 e 3,3V. A implementação da GHI para saídas analógicas permite o dimensionamento automático: você dá o mínimo, máximo e o valor da tensão.

Um teste simples seria colocar o mínimo para 0 e máximo para 330V (3,3 x 100) e o valor de 100 (1Vx100). Isto vai dar 1V no pino. Vamos ligar este pino em uma outra uma entrada analógica para verificar se realmente tem 1V. Pode não ser exatamente 1V, mas muito perto.

A saída analógica é multiplexado com a entrada analógica 3. Ao utilizar a saída analógica, a entrada analógica não pode ser utilizada neste mesmo pino, mas nós podemos utilizar outras entradas analógicas.

```
using System;
using Microsoft.SPOT;
using System.Threading;
using Microsoft.SPOT.Hardware;
using GHIElectronics.NETMF.Hardware;
using GHIElectronics.NETMF.FEZ;

namespace MFConsoleApplication1
{
    public class Program
    {
        public static void Main()
        {
            AnalogOut VoltLevel = new
AnalogOut((AnalogOut.Pin)FEZ_Pin.AnalogOut.An3);
            VoltLevel.SetLinearScale(0, 3300);
            VoltLevel.Set(1000);

            AnalogIn PinVoltage = new
```

```
AnalogIn((AnalogIn.Pin)FEZ_Pin.AnalogIn.An0);
            PinVoltage.SetLinearScale(0, 3300);

            while (true)
            {
                int voltage = PinVoltage.Read();
                Debug.Print("Voltage = " + (voltage / 1000).ToString() + "."
+ (voltage % 1000).ToString());

                Thread.Sleep(200);
            }
        }
    }
}
```

Ligue um fio entre An0 e AOUT (An3) e execute este programa. Note que, se nenhum fio está ligado, então a entrada analógica é "flutuante" e mostra valores aleatórios. Tente tocar a AIN0 e veja como as mudanças de valores, em seguida, conecte-a em AOUT e retornará bem próximo de 1V.

15. Coletor de Lixo

Quando se programa com linguagens mais antigas, como C ou C++, programadores devem manter o controle dos objetos que eles criam e liberam quando necessário. Se um objeto é criado e não liberado, está usando os recursos do sistema que nunca serão liberados. O sintoma mais comum é o estouro de vazamento de memória (memory leak). Um programa que tem um vazamento de memória vai usar mais e mais memória e mais até que dê estouro de memória. Estes erros são muito difíceis de localizar em um código.

As línguagens modernas têm uma coletor de lixo (Garbage Collector), que se mantém informado dos objetos usados. Quando o sistema ficar sem memória, o coletor de lixo é ativado e procura entre todos os objetos e libera aqueles que não são mais referenciados. Lembra quando nós criamos objetos com a palavra "new", então temos uma referência atribuído ao objeto? Um objeto pode ser referenciado várias vezes o coletor de lixo irá excluir o objeto que não tem referência (NULL).

```
// Novo Objeto
OutputPort Ref1 = new OutputPort(FEZ_Pin.Digital.LED, true);
// Segunda referência para o mesmo objeto
OutputPort Ref2 = Ref1;
// Loose a primeira referência
Ref1 = null;
// Nosso objeto está ainda sendo referencia
// Não será removido ainda
// Agora remova a segunda referência
Ref2 = null;
// A partir deste ponto o objeto está pronto para ser
// removido pelo coletor de lixo
```

Observe que o objeto não for removido imediatamente. Se necessário, o coletor de lixo irá executar e excluí-lo. Isso pode ser inconveniente às vezes, porque o coletor de lixo necessita de um tempo para pesquisar e excluir objetos. Isso só vai durar alguns milisegundos, mas se sua aplicação poder permitir isso? Neste caso, você pode forçar a execução do coletor de lixo quando você quiser.

```
//Força o coletor de lixo
Debug.GC(true);
```


15.1. Perdendo Recursos

O coletor de lixo facilita a alocação de objetos, mas também pode causar problemas se não tivermos cuidado. Um exemplo típico de uma saída digital. Suponha que precisamos de um pino para o estado alto. Criamos um objeto OutputPort e setamos o pino para alto. Mais tarde, perdemos a referência a esse objeto por algum motivo. O pino está ainda alto quando se perdeu a referência. Até então, não se preocupe. Depois de alguns minutos, o coletor de lixo entra em ação a objetos não mais referenciados então os mesmos são removidos. Liberando o OutputPort fará com que o estado do pino passe a ser entrada, então o pino não está mais no estado alto.

```
// Ligue o LED
OutputPort LED = new OutputPort((Cpu.Pin)FEZ_Pin.Digital.LED, true);
LED = null;
// Nos perdemos a referência ao LED mais ainda ela está lá
// Force o coletor de lixo
Debug.GC(true);
// O LED irá agora apagar
```

Uma coisa importante a se notar é que, se criar uma referência a um objeto dentro de um método, a referência será perdida na saída do método. Está aqui um exemplo:

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

namespace Test
{
    public class Program
    {
        static void TurnLEDOn()
        {
            // Ligue o LED
            OutputPort LED = new OutputPort((Cpu.Pin)FEZ_Pin.Digital.LED,
true);
        }
        public static void Main()
        {
            TurnLEDOn();
            // Achamos que tudo está OK, mas não está
            // Execute o GC
            Debug.GC(true);
            // O LED ainda está ligado ?
        }
```

```
    }
}
```

Para superar isso, precisamos de uma referência que está sempre acessível. Aqui está o código correto:

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

namespace Test
{
    public class Program
    {
        static OutputPort LED;
        static void TurnLEDOn()
        {
            // Ligue o LED
            LED = new OutputPort((Cpu.Pin)FEZ_Pin.Digital.LED, true);
        }
        public static void Main()
        {
            TurnLEDOn();
            // Execute o GC
            Debug.GC(true);
            // O LED está ligado ?
        }

    }
}
```

Outro bom exemplo refere-se a temporizadores. NETMF oferece um modo de criar timers que manuseia um serviço após um determinado tempo. Se a referência para o timer é perdido e que o coletor de lixo tenha rodado em seguida, o timer é perdido e não funcionará como esperado. Timers serão explandados mais tarde.

15.2. Dispose

O coletor de lixo irá liberar objetos em determinado ponto, mas se você quiser liberar um objeto em particular imediatamente ? A maioria dos objetos tem um método Dispose. Se um objeto tem de ser liberada a qualquer momento, você pode "Dispose" (liberar) ele.

Organizar os objetos é muito importante com NETMF. Quando criamos um objeto InputPort, o pino é reservado. E se alguém quiser usá-las como saída ? Ou até mesmo como uma entrada analógica? Temos que libertar o pino e, em seguida, basta criar o novo objeto:


```
OutputPort OutPin = new OutputPort((Cpu.Pin)FEZ_Pin.Digital.Di5, true);
OutPin.Dispose();
InputPort InPort = new InputPort((Cpu.Pin)FEZ_Pin.Digital.Di5, true,
Port.ResistorMode.PullUp);
```

15.3. Mensagens de saída do coletor de lixo

Quando o coletor de lixo é executado, ele exibe uma série de informações úteis para a janela de saída. Essas mensagens lhe dão uma idéia de que recursos do sistema está usando.

Embora não seja recomendado, você pode querer desativar essas mensagens para liberar a janela de saída para o seu próprio uso. Isso é facilmente alcançável com esta linha de código

```
Debug.EnableGCMessages(false);
```

16. C# Nível 3

Esta seção irá cobrir todo material C# que nós queremos incluir neste livro. Um bom e livro grátis para continuar aprendendo sobre C# está disponível em

http://www.programmersheaven.com/2/CSharpBook

16.1. Byte

Aprendemos como int são úteis para armazenar números. Eles podem armazenar um número muito grande, mas cada inteiro consome 4 bytes de memória. Você pode pensar no byte como uma simples célula de memória. Um byte pode conter um valor entre 0 e 255. Parece pouco mas é suficiente para muitas coisas. Em C#, bytes são declaradas usando a palavra "byte"

```
byte b = 10;
byte bb = 1000;// Isto não vai funcionar
```

O valor em um byte pode ser no máximo 255 [0.255], então o que acontece nele quando o incremento em 1? O valor ultrapassa a capacidade de armazenamento, retornando a 0.

Você provavelmente vai querer usar inteiros para a maioria das variáveis, mas nós aprendemos mais tarde que os bytes são muito importantes quando você começar a usar arrays (array).

16.2. Char

Para representar uma língua como o Inglês, precisamos de 26 valores para letras minúsculas e 26 para maiúsculas e talvez outros 10 para símbolos. Somando tudo, fica abaixo de 255, então um byte é suficiente. Se criarmos uma tabela que contém letras, números e símbolos, todos podem ser representados por um valor numérico. Na verdade, essa tabela já existe e é chamado tabela ASCII.

Um byte é suficiente para armazenar todos os "caracteres" que usamos em Inglês. Os computadores modernos têm sido extendido para incluir outros idiomas, alguns caracteres complexos "não-latino". Esses novos caracteres são chamados de Unicode. Estes podem assumir valores maiores do que 255, então um byte não é suficiente. Mas um número inteiro (4 bytes) é muito longo. Precisamos de um tipo que utilize dois bytes de memória. 2 bytes

são para armazenar valores entre 0 e 65535. Este tipo é chamado de "short", mas não vamos usá-lo neste livro.

Os sistemas podem representar caracteres usando um ou dois bytes. Os programadores decidiram criar um novo tipo chamado de "char" que pode ser 1 ou 2 bytes, dependendo do sistema. Como NETMF é feito para sistemas pequenos, um caractere é 1 byte. Este não é o caso em um PC onde tem dois bytes!

Não se preocupe muito com isso: Não use char, se você não precisa e se você usá-lo, lembre-se que ele ocupa no NETMF 1 byte.

16.3. Array (Vetor/Matriz)

Se lermos 100 vezes uma entrada analógica e queremos passar para as leituras de um método, não é prático usar 100 variáveis em 100 argumentos! Em vez disso, ele cria um array do tipo da nossa variável. Você pode criar array de qualquer objeto. Vamos utilizar principalmente array de bytes. Quando iniciar uma interface de hardware ou acessar arquivos, você irá sempre utilizar array de bytes.A declaração é semelhante ao de objetos:

```
byte[] MyArray;
```

O código acima cria uma “referência"para um objeto do tipo"byte array”. Isto é apenas uma referência mas ainda não inclui qualquer objeto, ele é nulo. Se você esqueceu o que é uma referência por isso volte para o capítulo "C # Nível 2”.

Para criar o objeto, nós utillizamos a palavra reservada "new" e então devemos especificar o tamanho de nosso array. O tamanho é o número de itens que temos em nosso array. Aqui, é o tipo "byte" e, portanto, o número representa a quantidade de bytes alocados na memória.

```
byte[] MyArray;
MyArray = new byte[10];
```

Nós criamos um array de bytes com 10 elementos na mesma. Esta tabela é referenciada por "MyArray".
Agora podemos ler / escrever qualquer um dos 10 valores na tabela simplesmente especificando o índice do valor desejado.

```
byte[] MyArray;
MyArray = new byte[10];
MyArray[0] = 123;// primeiro índice
MyArray[9] = 99;// último índice
MyArray[10] = 1;// isto é ruim...ERRO!!!!
```


Um ponto muito importante a notar aqui é que os índices começam em 0. Assim, para um array de 10 elementos, o índice varia de 0-9. O acesso ao índice 10 não vai funcionar e gerar uma exceção.

Podemos atribuir valores para os elementos da matriz na sua declaração. O exemplo a seguir irá armazenar o número 1-10 no índice de 0-9:

```
byte[] MyArray = new byte[10] { 1, 2, 3, 4, 5, 6, 7, 8, 9, 10 };
```

Para copiar uma array, use a classe Array como segue

```
byte[] MyArray1 = new byte[10] { 1, 2, 3, 4, 5, 6, 7, 8, 9, 10 };
byte[] MyArray2 = new byte[10];
Array.Copy(MyArray1, MyArray2, 5);//copia somente 5 elementos
```

Uma propriedade importante e útil de uma array é a Length. Podemos usá-lo para determinar o número de itens que ela contém

```
byte[] MyArray1 = new byte[10] { 1, 2, 3, 4, 5, 6, 7, 8, 9, 10 };
byte[] MyArray2 = new byte[10];
Array.Copy(MyArray1, MyArray2, MyArray1.Length);//copia a array inteira
```

16.4. String

Nós já usamos strings em vários lugares. Vamos rever o que nós já aprendemos e acrescentar mais alguns detalhes.

Os programas geralmente precisam construir mensagens. Estas mensagem devem ser lidos por um humanos. Por ser útil e muitas vezes usados em programas, C # suporta strings nativamente. C# sabe se o texto em um programa é uma string se estiver entre aspas.

Aqui está um exemplo de strings

```
string MyString = "Some string";
string ExampleString = "string ExampleString";
```

Tudo entre aspas é de cor vermelha e considerada uma string. Note que na segunda linha eu propositadamente coloquei o mesmo texto entre aspas. C # não compilar o que está entre

aspas (texto em vermelho), mas é considerado simplesmente como uma string.

Você pode ainda estar preocupado com a diferença entre uma variável inteira que contém o valor 5 e uma seqüência que contém o número 5. Está aqui um exemplo:

```
string MyString = "5" + "5";
int MyInteger = 5 + 5;
```

Quais são os valores das variáveis? Para o inteiro é 10, porque 5 +5 = 10. Mas para string isto não é verdadeiro. Strings não conhece nada sobre o que há dentro, seja texto ou números, não importa. Quando adicionando 2 strings, uma nova string é construída para combinar ambas. Então "5" + "5" + = "55" e não 10 como inteiros

Quase todos os objetos têm um método toString que converte a informação do objeto para um texto imprimível. Está aqui um exemplo:

```
int MyInteger = 5 + 5;
string MyString = "The value of MyInteger is: " + MyInteger.ToString();
Debug.Print(MyString);
```

Este código irá imprimir

The value of my myInteger is: 10

Strings podem ser convertidos para array de bytes, se necessário. Isto é importante se nós usamos um método que aceita apenas bytes e queremos passar nossa string para ele. Ao fazer isso, todos os caracteres na string serão convertidos em seu valor equivalente em byte e armazenado na array.

```
using System.Text;
.....
.....
byte[] buffer = Encoding.UTF8.GetBytes("Example String");
```

16.5. Laço - For

Usar laço while é suficiente para os nossas necessidades de loops, mas laço for pode ser mais fácil de se utilizar. O exemplo mais simples é faze um programa que conte de 1 até 10. Similarmente nós podemos faze um LED piscar 10 vezes. O laço for tem três argumentos em uma variável. Necessita o valor inicial, como para terminar o ciclo e o que fazer em cada loop.


```
int i;
for (i = 0; i < 10; i++)
{
    //faça alguma coisa
}
```

Inicialmente, devemos declarar a variável que vamos utilizar. Em seguida, no loop for, temos que passar os três argumentos (inicial, regra, ação). No primeiro ciclo de loop, nos setamos a variável i para 0. Em seguida, o loop continuará correndo enquanto a variável i for inferior a 10. Finalmente, o loop for irá incrementar a variável i em cada iteração. Vamos tentar isso:

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

namespace MFConsoleApplication1
{
    public class Program
    {
        public static void Main()
        {
            int i;
            for (i = 0; i < 10; i++)
            {
                Debug.Print("i= " + i.ToString());
            }
        }

    }
}
```

Se executar o programa, vemos que ela apresenta os valores 0-9, mas não o 10. Mas nós queríamos 1-10, 0-9, não! Para começar de 1, não 0, precisamos inicializar i com 1 i no início do loop. Além disso, para rodar até 10, temos de dizer o loop for para executar até 10 e não menos que 10, então vamos mudar a condição de "<" para "<="

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

namespace MFConsoleApplication1
{
    public class Program
    {
        public static void Main()
        {
            int i;
```

```
            for (i = 1; i <= 10; i++)
            {
                Debug.Print("i= " + i.ToString());
            }
        }

    }
}
```

Podemos fazer o loop somente com números pares ?

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

namespace MFConsoleApplication1
{
    public class Program
    {
        public static void Main()
        {
            int i;
            for (i = 2; i <= 10; i = 1 + 2)
            {
                Debug.Print("i= " + i.ToString());
            }
        }
    }
}
```

O melhor meio de se entender o loop for é executando passo a passo o código e vendo como o C# executa ele.

16.6. Comando Switch

Você provavelmente não usou o comando “swtich” em aplicações, mas você verá que ele é muito útil para programas grandes, especialmente quando você tem que trabalhar com máquinas de estados. O comando switch compara uma variável para uma lista de constantes (apenas constantes) e executa uma ação conforme o caso. Neste exemplo, lemos o dia atual ("DayOfWeek") e partir deste se escreve por extenso (string) o dia da semana correspondente. Nós poderíamos fazer isto com o if, mas veja como simplificam as coisas com a opção Switch:

```
using System.Threading;
```

```
using Microsoft.SPOT;
using System;

namespace MFConsoleApplication1
{
    public class Program
    {
        public static void Main()
        {
            DateTime currentTime = DateTime.Now;
            int day = (int)currentTime.DayOfWeek;
            switch (day)
            {
                case 0:
                    Debug.Print("Sunday");
                    break;
                case 1:
                    Debug.Print("Monday");
                    break;
                case 2:
                    Debug.Print("Tuesday");
                    break;
                case 3:
                    Debug.Print("Wednsday");
                    break;
                case 4:
                    Debug.Print("Thursday");
                    break;
                case 5:
                    Debug.Print("Friday");
                    break;
                case 6:
                    Debug.Print("Saturday");
                    break;
                default:
                    Debug.Print("We should never see this");
                    break;
            }
        }
    }
}
```

O ponto importante é que a variável é comparado com uma lista de constantes. Depois de cada ("case") nós temos uma constante e não uma variável.

Podemos modificar esse código para trocar para enumeração como segue.

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

```
using System;

namespace MFConsoleApplication1
{
    public class Program
    {
        public static void Main()
        {
            DateTime currentTime = DateTime.Now;
            switch (currentTime.DayOfWeek)
            {
                case DayOfWeek.Sunday:
                    Debug.Print("Sunday");
                    break;
                case DayOfWeek.Monday:
                    Debug.Print("Monday");
                    break;
                case DayOfWeek.Tuesday:
                    Debug.Print("Tuesday");
                    break;
                case DayOfWeek.Wednesday:
                    Debug.Print("Wednsday");
                    break;
                case DayOfWeek.Thursday:
                    Debug.Print("Thursday");
                    break;
                case DayOfWeek.Friday:
                    Debug.Print("Friday");
                    break;
                case DayOfWeek.Saturday:
                    Debug.Print("Saturday");
                    break;
                default:
                    Debug.Print("We should never see this");
                    break;
            }
        }
    }
}
```

Tente ir Ir passo-a-passo no código e veja a operação em detalhes

17. Interfaces Seriais

Há muitas interfaces seriais disponíveis para transferência de dados entre os processadores. Cada um tem suas vantagens e desvantagens. Vou tentar incluir detalhes suficientes para que você possa usá-los com NETMF.

Apesar de existirem várias interfaces seriais, quando falamos falamos sobre serial, significa UART serial ou RS232. Outras interfaces, como CAN ou SPI, transmitem dados em série, mas não são portas seriais.

Para mais detalhes, consulte a web, particularmente http://www.wikipedia.org/ .

17.1. UART

UART é uma das interfaces mais antigas e mais comuns. Os dados são enviados em um pino TXD com uma seqüência e uma velocidade pré-definidas. Quando o transmissor envia 0 e 1, o receptor verifica os dados de entrada no pino RXD na mesma velocidade que o transmissor envia. Os dados são enviados um byte de cada vez. Esta é uma direção para os dados. Para mover na direção oposta, um circuito similar existe no dispositivo remoto. Transmissão e recepção são realizadas por dois circuitos totalmente separados e podem operar em conjunto ou separadamente. Cada uma das partes pode enviar ou receber dados a qualquer momento.

A velocidade para enviar / receber dados é chamado de taxa de transmissão. Taxa de transmissão é quantos bytes serão enviados em um segundo. Geralmente, um padrão de velocidade é utilizada, como 9600, 119200, 115200, por exemplo.

Através de UART, dois processadores podem ser interligados, ligando o pino TXD um lado no pino RXD do outro e vice-versa. Como o sinal é digital, a tensão no pino TXD UART / RXD irá para 0V (baixo) a 3.3V ou 5V (superior).

Nos sistemas industriais ou quando cabos longos são utilizados, 3.3V 5V não é suficiente para escapar da margem de ruído. Existem normas para definir a forma de transformar o sinal com uma tensão para mais alta para permitir a comunicação mais confiável. O mais comum é RS232. Praticamente todos os computadores possuem uma porta RS232. Em RS232, os dados são os mesmos que UART mas as tensões são convertidas a partir de níveis TTL (0 a 3.3 V) para níveis RS232 (- 12V a +12 V). Um ponto muito importante é que as tensões são invertidas em relação ao que se poderia pensar. Quando o sinal é logicamente "baixo", corresponde a de tensão é 12 V e quando é "alta", a tensão é -12V. Há muitos pequenos circuitos que convertem os níveis TTL para níveis RS232, como MAX232 ou MAX3232.Quando precisamos de int erface de um processador que usa UARTs com um PC usando RS232, então precisamos usar um conversor de sinal.

Aqui está um exemplo de tais conversores:

http://www.nkcelectronics.com/rs232-to-ttl-converter-board-33v232335.html

No mundo do PC, a porta USB é mais comum que as portas seriais. Os novos computadores, especialmente laptops, não têm porta serial, mas tem uma porta USB. Por este motivo, os fabricantes criaram conversores USB-RS232. Um produto interessante é cabo USB com interface UART. Note que este é TTL UART, não RS232, o que significa que você pode ligar diretamente na UART do processador. A referência para este cabo é "TTL-232R-3V3".

Para resumir tudo, você pode ligar dois processadores pelos pinos de conexão UART diretamente. Para a interface do processador com um computador, você precisará de um conversor UART/RS232 ou USB ou utilizando um dos circutos prontos, como MAX232 para RS232 e FT232 para USB.

NETMF suporta portas série UART da mesma maneira que Framework do PC. Para usar uma porta serial, adicione o conjunto "Microsoft.SPOT.Hardware.SerialPort" e "using System.IO.Ports" no início do seu código.

As portas seriais em computadores e NETMF são chamados de COM e começam com a COM1. Não há COM0 em computadores. Isso pode ser irritante quando você quer mapear uma porta COM para uma porta UART no processador, que normalmente começa em UART0 e não UART1. Então COM1 é UART0, COM2 = UART1 etc ....

PCs têm, frequentemente, um programa "Terminal" que abre a porta serial para enviar e receber dados. O que você digitar no terminal é enviado pela serial e o que vir pela serial é impresso. Um exemplo desse tipo de programa é: TeraTerm.

O programa a seguir envia o valor de um contador de 10 vezes por segundo. Os dados são enviados a um taxa de 115.200 baud, portanto, tenha certeza de que o terminal está configurado com esta velocidade. O programa envia os dados no dispositivo COM1 do NETMF. A porta COM não tem nada a ver com a de seu PC. Por exemplo, você pode ter uma conversor USB / Serial que foi mapeado na COM8. Então você deve abrir a porta do seu PC de identificação COM8 e não COM1.

```
using System.Threading;
using System.IO.Ports;
using System.Text;

namespace MFConsoleApplication1
{
    public class Program
    {
        public static void Main()
        {
            SerialPort UART = new SerialPort("COM1", 115200);
            int counter=0;
            UART.Open();
            while (true)
            {
```

```
                // cria uma String
                string counter_string = "Count: " + counter.ToString() +
"\r\n";
                // converte string para bytes
                byte[] buffer = Encoding.UTF8.GetBytes(counter_string);
                // manda os bytes para a porta serial
                UART.Write(buffer, 0, buffer.Length);
                // incrementa o contador
                counter++;
                //aguarde
                Thread.Sleep(100);
            }
        }
    }
}
```

Note que nós terminamos nossa string com "\n\r". The “\r” no código diz ao terminal para voltar ao início da linha e "\n" significa adicionar uma nova linha (pular a linha).

Quando os dados são recebidos na UART, elas são infileirados de forma a não perder qualque dado. Note que há limites para a quantidade de dados que podem ser armazenados, por isso, se você está depurando ou você não está lendo os dados recebidos, feche a porta serial, senão o sistema irá desacelerar significativamente e o debug/execução será instável e lenta. Idealmente, eventos serão utilizados para nós recebermos automaticamente os dados. Eventos serão vistos mais tarde.

Este exemplo irá esperar até receber um byte e mostrar o que você digitou (transmitida, na verdade), pelo terminal.

```
using System.Threading;
using System.IO.Ports;
using System.Text;

namespace MFConsoleApplication1
{
    public class Program
    {
        public static void Main()
        {
            SerialPort UART = new SerialPort("COM1", 115200);
            int read_count = 0;
            byte[] rx_byte = new byte[1];

            UART.Open();
            while (true)
            {

                // Leia um byte
                read_count = UART.Read(rx_byte, 0, 1);
```

```
                if (read_count > 0)// temos dados?
                {
                    // cria uma string
                    string counter_string = "You typted: " +
rx_byte[0].ToString() + "\r\n";
                    // converte string para bytes
                    byte[] buffer = Encoding.UTF8.GetBytes(counter_string);
                    // manda os bytes para serial
                    UART.Write(buffer, 0, buffer.Length);
                    //aguarde
                    Thread.Sleep(10);
                }
            }
        }
    }
}
```

O último exemplo é um loop-back. Ligue um fio do pino TX ao RX na sua placa e envie dados e esteja certo que está recebendo corretamente.

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

namespace MFConsoleApplication1
{
    public class Program
    {
        public static void Main()
        {
            SerialPort UART = new SerialPort("COM1", 115200);
            int read_count = 0;
            byte[] tx_data;
            byte[] rx_data = new byte[10];
            tx_data = Encoding.UTF8.GetBytes("FEZ");
            UART.Open();
            while (true)
            {
                // flush todos os dados
                UART.Flush();
                // Envia algum dado
                UART.Write(tx_data, 0, tx_data.Length);
                // Espere para ter certeza que o dado é transmitido
                Thread.Sleep(100);
                // leia do dado
                read_count = UART.Read(rx_data, 0, rx_data.Length);
                if (read_count != 3)
                {
```

```
                    // Nós enviamos 3 então devemos ter 3 de volta
                    Debug.Print("Wrong size: " + read_count.ToString());
                }
                else
                {
                    // O contador está correto, então check os valores
                    //Eu estou fazendo de uma maneira mais clara
                    if (tx_data[0] == rx_data[0])
                    {
                        if (tx_data[1] == rx_data[1])
                        {
                            if (tx_data[1] == rx_data[1])
                            {
                                Debug.Print("Perfect data!");
                            }
                        }
                    }
                }
                Thread.Sleep(100);
            }
        }
    }
}
```

17.2. SPI

SPI usa 3 a 4 fios para transmitir dados. Em UART, ambos os lados devem usar uma velocidade pré-definidas. Para SPI, é diferente, porque um nó envia um sinal de clock para o outro, ao mesmo tempo que os dados. Este sinal de clock é para determinar o quanto o receptor deve ser rápido para ler os dados. Se você tiver algum conhecimento de eletrônica, há um registrador de deslocamento. O sinal de clock é sempre transmitida a partir do dispositivo mestre. O outro dispositivo é um escravo que não envia, mas recebem sinal de clock do mestre.

Então, o mestre transmite o sinal de clock SCK pino (clock serial) e simultaneamente, transmite os dados no pino MOSI (Master Slave Out In). O escravo lê o sinal de clock em seu pino SCK e, simultaneamente, lê os dados em seu pino MOSI. Até então, ele é uma comunicação em uma direção. Enquanto os dados são transmitidos em uma direção usando MOSI, outro pacote de dados são transmitidos de volta no pino MISO (Master In Slave Out). Tudo isso é feito simultâneamente , enviando e recebendo. Com o SPI, não é possível apenas enviar ou receber. Você sempre receberá um byte para cada byte enviado. Tamanhos diferentes de dados são possíveis, mas o mais comum é o octeto. NETMF suporta transferências de 8 (byte) e 16 bits (short).

Devido a este esquema de mestre/escravo, você pode adicionar mais escravos no mesmo bus e escolhe com quem quer trocar dados. Eu uso o termo de troca, porque você não pode “enviar” ou “receber”, mas “enviar e receber” dados (trocar) . O mestre seleciona o escravo usando seu pino SSEL(slave select) ou CS (Chip Select). Em teoria, um mestre pode ter um

número ilimitado de escravos, mas não pode selecionar mais de um em um única vez. O mestre precisa de apenas três ligações (SCK, MISO, MOSI) para se conectar a todos os escravos, mas precisa de um pino SSEL dedicados a cada escravo.

Alguns hardwares SPI (escravo) pode ter mais de um pino CS como o decodificador MP3 VS1053 que usa um pino para dados e outro para comandos, mas ambos compartilham o mesmo 3 pinos de transferência (SCK, MOSI, MISO) .

SPI exige mais fios que outros “bus” similares, mas permite transferências muito rápidas. Um clock de 50MHz é possível em um barramento SPI, ou seja, 50 milhões de bits por segundo.

Os dispositivos NETMF são sempre mestres SPI. Antes de criar um objeto SPI, precisamos criar um objeto SPI “configuration”. O objeto de configuração é usado para definir o estado de pinos e alguns parâmetros de tempo. Na maioria dos casos, você quer que esteja em repolso (false) com clock na borda de subida (true) e com zero para selecionar o setup e terá o tempo para o estado de baixo com um pulso sobre o estado de alta e de 0 para selecionar o “Setup” e "Hold Time". A única coisa que você deveria setar é o clock do relógio. Alguns dispositivos podem aceitar uma alta freqüência, mas não todos. Ao colocar em 1000KHz (1MHz), deve funcionar para a maioria dos dispositivos escravos.

O exemplo está enviando e recebendo (troca) 10 bytes de pelo SPI, canal 1. Note que no NETMF começa numerando os canais SPI a partir de 1, mas nos processadores, os canais começam a partir de 0. SPI1 no código é o SPI0 no processador.

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

namespace MFConsoleApplication1
{
    public class Program
    {
        public static void Main()
        {
            SPI.Configuration MyConfig =  new
SPI.Configuration((Cpu.Pin)FEZmini.Pin.Di2x,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(100);
        }
    }
```

```
}
```

17.3. I2C

I2C foi desenvolvido pela Philips para permitir que muitos chipsets possam se comunicar em um bus de 2 fios em equipamentos domésticos (TV, geralmente). Como SPI, I2C é um mestre e tem vários escravos no mesmo bus. Ao invés de escolher o escravo através de um pino, I2C utiliza um endereço por software. Antes da transferência de dados, o mestre envia o endereço de 7 bits do escravo com quem deseja comunicar. Ele também envia um bit indicando se o mestre desejar enviar ou receber dados. O endereço do escravo, que se localiza nos bus confirmar sua presença e, em seguida, o mestre pode enviar / receber dados.
O processador irá iniciar seu negócio com o "start", antes de enviar qualquer coisa, e acabar com a condição de "stop".

Os drivers I2C do NETMF são baseados em transações. Se quisermos ler um valor do registro de um sensor, é preciso enviar o número do registo, então vamos ler o registro. Portanto, estas são duas operações: escrita e leitura.

Este exemplo mostra a comunicação com um dispositivo I2C, de endereço 0x38 e então escreve 2 (número do registro) e lê de volta seu valor.

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

namespace Test
{
    public class Program
    {
        public static void Main()
        {
            //cria um objeto I2C
            I2CDevice.Configuration con = new I2CDevice.Configuration(0x38,
400);
            I2CDevice MyI2C = new I2CDevice(con);

            //cria transações (nós necessitamos de 2 neste exemplo)
            I2CDevice.I2CTransaction[] xActions = new
I2CDevice.I2CTransaction[2];

            // crie um buffer de escrita (nós necessitamos 1 byte)
            byte[] RegisterNum = new byte[1] { 2 };
            xActions[0] = MyI2C.CreateWriteTransaction(RegisterNum);
            // cria um buffer de leitura para ler o registrador
            byte[] RegisterValue = new byte[1];
            xActions[1] = MyI2C.CreateReadTransaction(RegisterValue);
```

```
            // Agora nós acessamos o bus I2C e time out de 1 segundos se não
há resposta
            MyI2C.Execute(xActions, 1000);

            Debug.Print("Register value: " + RegisterValue[0].ToString());
        }

    }
}
```

17.4. One Wire

GHI exclusivamente tem suporte para dispositivos 1-Wire sobre NETMF. Dallas Semiconductor é fabricante de dispositivos que conversam 1-Wire, possuindo sensores de temperatura e EEPROMs, utilizando um único fio para transferência de dados. Vários dispositivos podem ser conectados e controlados com um único fio. A classe "one wire" oferece vários métodos para ler e gravar dados em um dispositivo 1-Wire. Ele também inclui um método de cálculo do CRC.
Este exemplo irá ler o sensor de temperatura fornecida pelo DS18B20. Note que este é uma exclusividade da GHI e, portanto, você deve adicionar o GHI assembly para rodar.

```
using System;
using System.Threading;
using Microsoft.SPOT;
using Microsoft.SPOT.Hardware;
using GHIElectronics.NETMF.Hardware;
using GHIElectronics.NETMF.FEZ;

namespace Test
{
    public class Program
    {
        public static void Main()
        {
            // Troque aqui for seu pino correto!
            Cpu.Pin myPin = (Cpu.Pin)FEZ_Pin.Digital.Di4;

            OneWire ow = new OneWire(myPin);
            ushort temperature;

            // leia a cada 1 segundo
            while (true)
            {
                if (ow.Reset())
                {
                    ow.WriteByte(0xCC);     // Pula ROM
                    ow.WriteByte(0x44);     // Inicia a conversão de
```

```
temperatura

                    while (ow.ReadByte() == 0) ;   // aguarde enquanto
ocupado

                    ow.Reset();
                    ow.WriteByte(0xCC);     // Pula ROM
                    ow.WriteByte(0xBE);     // Leia  Scratchpad

                    temperature = ow.ReadByte();       // LSB
                    temperature |= (ushort)(ow.ReadByte() << 8); // MSB

                    Debug.Print("Temperature: " + temperature / 16);
                    Thread.Sleep(1000);
                }
                else
                {
                    Debug.Print("Device is not detected.");
                }

                Thread.Sleep(1000);
            }
        }
    }
}
```

17.5. CAN

Interface CAN (Controller Area Network) é muito comum na indústria e automação. CAN é muito robusto e funciona muito bem em ambientes com ruídos e em altas velocidades. Todos os tratamentos e os metodos de recuperação são feitos no hardware. TD (Transmit Data) e RD (Receive Data) são os únicos dois pinos necessários. Estes pinos transportar o sinal digital para ser convertido para analógico antes de ser enviado sobre a linha usando a camada física. As camadas físicas são às vezes chamados de trasncievers.

Existem vários tipos de camadas físicas, mas a maioria usa HSDW (High Speed Dual-Wire), que utiliza um par trançado para a sua imunidade ao ruído. Esta camada pode se comunicar até 1 Mbps e fazer upload de dados a longas distâncias, se a a taxa de transferência é baixa. Os dados podem ser transferidos entre nós no bus, onde cada nós podem trasnferir a qualquer momento para os outros nós,e todos os outros nós devem ser capazes de receber dados. Com o CAN, não existe master / slave. Além disso, todos “nós” devem ter um critério de sincronização predefinido. Isso é muito mais complicado do que calcular um simples baud rate para UART. Por esta razão, existem muitas calculos de taxa de bits para a CAN.

O dispositivo CAN da placa Embedded Master é idêntico ao popular SJA1000. Um pouco de pesquisa na internet para SJA1000 deve dar-lhe vários opções de cálculo.

Aqui está um exemplo de calculadora dedicado:

http://www.esacademy.com/en/library/calculators/sja1000-timing-calculator.html

Aqui está um exemplo de calcular o “timming”

1.Divida o clock do sistema (72MHz) para obter um clock apropriado para o periférico CAN (isto não é BAUD RATE) Este é a chamado de BRP.

2.Defina quandtos clocks você necessetia para definir um 1. Normalmente, este é de 24 ou 16. Isso é chamado TQ.

3.Atribua os valores para T1 e T2 onde T1 + T2 + 1 = TQ

Vamos supor que precisemos de 250Kbps.

1. A partir de clock 72MHz sistema, eu quero que clock do CAN seja de 6Mhz, então eu preciso dividir por 12 (BRP = 12).
2. 6Mhz/250kbps = 24 TQ (nós queremos geralmente 16 ou 24).
3. T1 = 15, T2 = 8 nos dá 15 + 8 + 1 = 24 e isso é o que precisamos.

Eu obtive os valores de T1 e T2 em http://www.kvaser.com/can/protocol/index.htm

Eu obtive o primeiro valor e subtraí 1 de T1, pois a calculadora incluiu o bit de sincronismo

GHI Electronics está atualizando os driver para o CAN NETMF 4.0 e assim a interface pode mudar. Verifique a documentação para obter mais ajuda.

http://www.esacademy.com/en/library/calculators/sja1000-timing-calculator.html

Aqui está o código com comentários detalhados:

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

namespace MFConsoleApplication1
{
    public class Program
    {
        public static void Main()
        {
            // Estes números foram calculados usando a calculadora deste link
            // http://www.kvaser.com/can/protocol/index.htm
            // Nós usamos o primeiro valor da saída da calculadora
            ////////////////////////////////////////////////////////////////
////////////////////////////
            // Bitrate 250Kbps
            // CLK = 72 Mhz, with BRP = 12 -> 6Mhz CAN clock
            // 6Mhz/250Kbps = 24 TQ
            // T1 = 16 menus 1 para sync = 15
            // T2 = 8
            // 15 + 1 + 8 = 24 TQs que é o que queremos
            ////////////////////////////////////////////////////////////////
```

```
//////////////////////////
            const int BRP = 12;
            const int T1 = 15;
            const int T2 = 8;
            // Para 500Kbps você pode usar BRP=6 e para  1Mbps você pode usar
BRP=3 e para  125Kbps   use BRP=24...e assim por diante
            // Mantenha T1 e T2 o mesmo para manter o sampling do pino o
mesmo (entre 70% e 80%)

            // Inicializa o canal CAN, set o bit rate
            CAN canChannel = new CAN(CAN.CANChannel.Channel_1,
                                                                ((T2 - 1) << 20)
| ((T1 - 1) << 16) | ((BRP - 1) << 0));
            // Faça uma nova mensagem CAN
            CAN.CANMessage message = new CAN.CANMessage();
            // Faça uma mensagem de 8 bytes
            for (int i = 0; i < 8; i++)
                message.data[i] = (byte)i;
            message.DLC = 8; // 8 bytes
            message.ArbID = 0xAB; // ID
            message.isEID = false; // ID não extendido
            message.isRTR = false; // não remoto
            // Envia a mensagem
            canChannel.PostMessage(message);
            // Espera por uma mensagem e a pegue
            while (canChannel.GetRxQueueCount() == 0) ;
            // Pegue a mensagem usando o mesmo objeto message
            canChannel.GetMessage(message);
            // Agora "message" contém o dado, ID, flags da mensagem recebida
        }
    }
}
```


18. Output Compare

Este recurso exclusivo da GHI permite que o desenvolvedor GHI gere um sinal em qualquer pino. Por exemplo, OutputCompare pode ser usado para gerar um sinal UART ou um sinal para controlar um controle remoto infravermelho em 38Khz para simular um controle remoto da TV.

Um bom exemplo é o driver o display serial de 2x16 encontrador em www.TinyCLR.com. O display é controlada pela UART. Ele não retorna dados. Tudo que você precisa é enviar alguns códigos de controle serial. Assim, podemos ligar o LCD em um dos pinos seriais. Então perderemos a porta serial para fazer uma coisa muito simples, além da UART necesssitar de um pino para transmitir e receber cados. Mas, o LCD não manda dados de volta, logo, vamos “perder” um pino. A maneira apropriada de controle deste LCD serial é utilizando o método OutputCompare: você só precisa de um pino e você pode usar qualquer um dos pinos digitais.

Como OutputCompare funciona? Basicamente, você fornece ao método um array de valores de tempos entre cada inversão do pino. OutputCompare vai procurar essa tabela e gerar o sinal no pino selecionado. Portanto, se queremos gerar o UART, devemos primeiro pré-calcular os valores que representam

os dados a transmitir e dar a ele como um parâmetro para o objeto OutputCompare.

Este driver de exemplo mostra para você. É uma cópia do driver do LCD serial encontrado no site da www.TinyCLR.com

```
using System;
using System.Threading;
using Microsoft.SPOT.Hardware;
using GHIElectronics.NETMF.Hardware;
namespace GHIElectronics.NETMF.FEZ
{
    public static partial class FEZ_Components
    {
        public class SerialLCD : IDisposable
        {
            const byte DISP_ON = 0xC;    //Torna o LCD visível
            const byte CLR_DISP = 0x01;      //Apaga display
            const byte CUR_HOME = 2;      //Move cursor para  home e a apaga
a memória de vídeo
            const byte SET_CURSOR = 0x80;   //SET_CURSOR + X : Seta o cursor
para posição X
            const byte Move_CURSOR_LEFT = 0x10;

            OutputCompare oc;
```

```
        const int MAX_TIMINGS_BUFFER_SIZE = 10;
        uint[] buffer = new uint[MAX_TIMINGS_BUFFER_SIZE];
        const int BAUD_RATE = 2400;
        const int BIT_TIME_US = 1 * 1000 * 1000 / BAUD_RATE;
        readonly int BYTE_TIME_MS;
        public void Dispose()
        {
            oc.Dispose();
            buffer = null;
        }
        private void SendByte(byte b)
        {
            bool currentPinState;
            int currentBufferIndex = 0;
            uint currentStateTiming;
            // bit de start
            currentPinState = false;
            currentStateTiming = BIT_TIME_US;
            // bit de dados
            for (int i = 0; i < 8; i++)
            {
                bool neededState = (b & (1 << i)) != 0;

                if (neededState != currentPinState)
                {
                    buffer[currentBufferIndex] = currentStateTiming;
                    currentStateTiming = BIT_TIME_US;
                    currentPinState = neededState;
                    currentBufferIndex++;
                }
                else
                {
                    currentStateTiming += BIT_TIME_US;
                }
            }
            // bit de parada
            if (currentPinState != true)
            {
                buffer[currentBufferIndex] = currentStateTiming;
                currentBufferIndex++;
            }
            oc.Set(false, buffer, 0, currentBufferIndex, false);\
            // espera dado ser enviado
            Thread.Sleep(BYTE_TIME_MS);
        }
        public void PutC(char c)
        {
            SendByte((byte)c);
        }

        private void SendCommand(byte cmd)
        {
```

```
                SendByte(0xFE);
                SendByte(cmd);
            }
            public void Print(string str)
            {
                for (int i = 0; i < str.Length; i++)
                    PutC(str[i]);
            }
            public void ClearScreen()
            {
                SendCommand(CLR_DISP);
            }
            public void CursorHome()
            {
                SendCommand(CUR_HOME);
            }
            public void SetCursor(byte row, byte col)
            {
                SendCommand((byte)(SET_CURSOR | row << 6 | col));
            }
            public void MoveLeft()
            {
                SendCommand(Move_CURSOR_LEFT);
            }
            public SerialLCD(FEZ_Pin.Digital pin)
            {
                BYTE_TIME_MS = (int)Math.Ceiling((double)BIT_TIME_US *
MAX_TIMINGS_BUFFER_SIZE / 1000);
                oc = new OutputCompare((Cpu.Pin)pin, true,
MAX_TIMINGS_BUFFER_SIZE);
                // Inicializa LCD

                SendCommand(DISP_ON);
                SendCommand(CLR_DISP);
            }
        }
    }
}
```

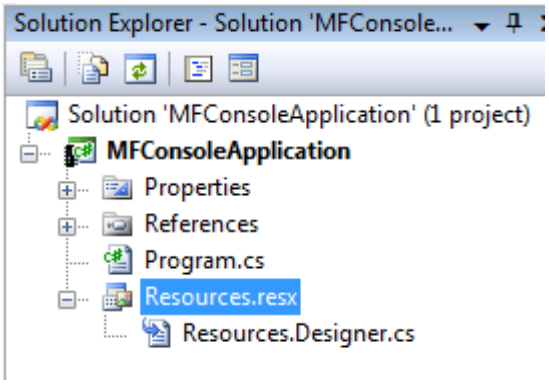
19. Carregando recursos

Um recurso é um arquivo que foi incluído com a imagem da aplicação. Se a sua aplicação depende de um determinado arquivo (imagem, ícone, de som) então nós podemos adicionar este arquivo para a aplicação dos recursos. Um aplicativo pode ler este arquivo do sistema de arquivos, mas se tornaria dependente do sistema de arquivos usado.

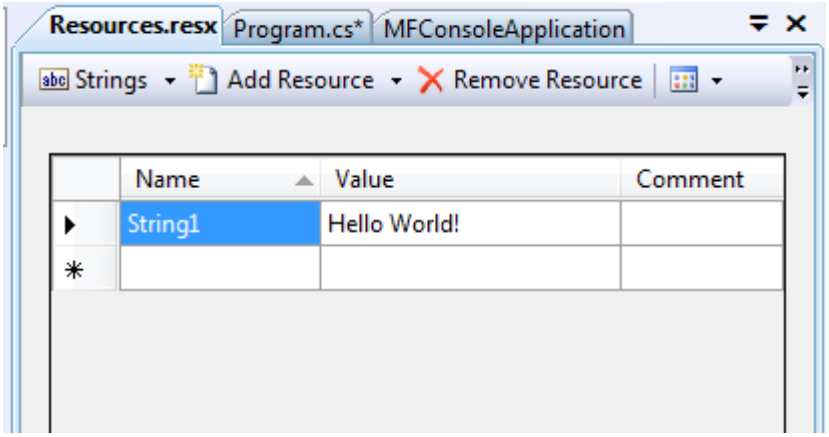
Recursos também podem se strings, como um aviso de direitos autorais. Assim, se necessitamos mudar o texto de copyright, nós deveriamos atualizar os recursos e não fazer nada no código.

Sempre conheça quanto de memória é reservado para espaço de aplicação. Adicionando arquivos muito grandes resultará em erros e o VS2010 não vai lhe dizer porque os arquivos são muito grandes.

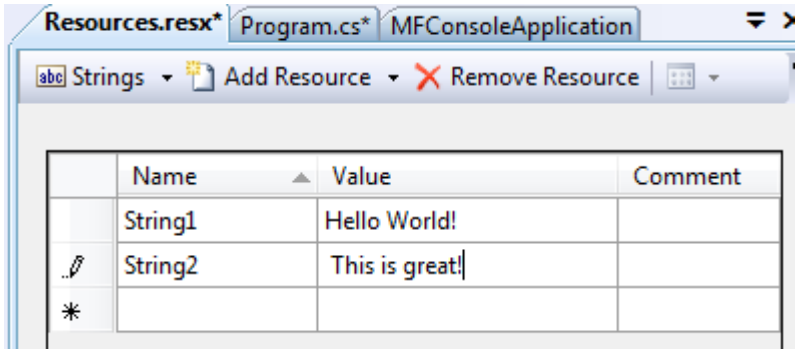
Olhando pela janela "Solution Explorer", você pode ver "Resource.resx" e se nós expandí-lo podemos ver "Resources.Designer.cs".



Visual Studio cria automaticamente um arquivo “resources designer”. Nunca modifique esse arquivo manualmente. Em vez disso, clique duas vezes em "Resources.resx" para abrir uma caixa de ferramentas específica.



Na janela do editor de recursos, o primeiro da lista “drop-down” na esquerda é para selecionar qual tipo de recurso nós queremos editar. Você percebe que já existe um "string resource" chamado “String1” o qual o “value” está setado para “Hello World!” Clique abaixo de “String1” e adicione a um novo recurso de string como mostrado na imagem abaixo.



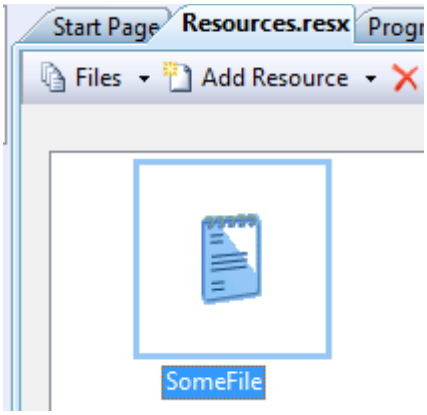
Agora nos temos 2 recursos de strings. Vamos usá-los.

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

namespace MFConsoleApplication1
{
    public class Program
    {
        public static void Main()
        {
```

```
            Debug.Print(Resources.GetString(Resources.StringResources.String1
));
            Debug.Print(Resources.GetString(Resources.StringResources.String2
));
        }
    }
}
```

Tente alterar o texto do recurso e veja como a saída muda.

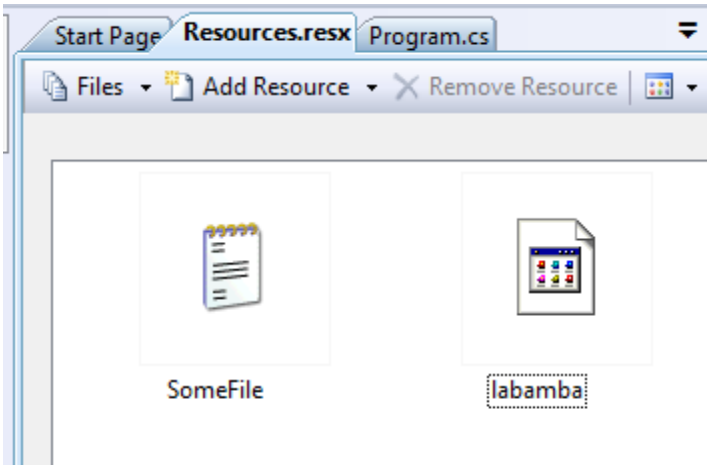
Agora adicione um arquivo. Crie um arquivo de texto no desktop e escrever algumas palavras nele. Escolha "files" do menu drop-down então click no “"Add Resource"



Como um arquivo de texto, o Visual Studio foi experto suficiente para adicionar isto no modo similar as strings. Então para usar este recurso, é da mesma forma como string.

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

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

Neste exemplo, eu adicionei um arquivo midi. Mais tarde, neste livro vemos como adicionar um decodificador e reproduzir o arquivo. Escolha um arquivo midi e adicione-no.



Neste exemplo, o tamanho do arquivo é de cerca de 33Kb. Este é pequeno para o mundo PC, mas não é pequeno para um um sistema embarcado. O exemplo a seguir funcionará em dispositivos com muita memória, como o ChipworkX e Embedded e Master. No USBizi e FEZ, ele pode não funcionar. Nós podemos ainda acessar arquivos grandes com USBIZI e FEZ, mas ao invés de lê-lo de uma única vez (na íntegra), vamos lê-lo em pedaços, decodificar e assim por diante. Vemos isso em detalhes mais tarde.

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

namespace MFConsoleApplication1
{
    public class Program
    {
        public static void Main()
        {
            byte [] file_data =
Resources.GetBytes(Resources.BinaryResources.labamba);
            Debug.Print("File Size is " + file_data.Length.ToString());
        }
    }
}
```

20. Displays

20.1. Display de caracteres

A maioria dos LCDs usa a mesma interface. Em geral, existem duas linhas de 16 caracteres, comumente chamado de 2x16 LCD. O display é controlado com uma interface de 8 bits ou 4 bits. Opção de 4 bits é melhor, porque requer menos pinos.

A interface utiliza RS (dados / instruções), RW (Read / Write), E (Enable) e 4 bits de dados. A documentação do LCD é a melhor fonte de informação, mas aqui está um exemplo para começar rapidamente.

GHI oferece uma alternativa a este LCD. O LCD serial oferecido no site www.TinyCLR.com utiliza um único pino no FEZ. Os drivers incluídos fazem os LCD mais fáceis de usar

Utilizando o driver do LCD

```
using GHIElectronics.NETMF.FEZ;

namespace MFConsoleApplication1
{
    public class Program
    {
        public static void Main()
        {
            FEZ_Components.SerialLCD LCD =
                new FEZ_Components.SerialLCD(FEZ_Pin.Digital.Di5);

            LCD.ClearScreen();
            LCD.CursorHome();
            LCD.Print("FEZ Rocks!");
        }
    }
}
```

20.2. Display gráficos

Suporte Nativo

NETMF, com sua classe bitmap, pode manipilar gráficos muito bem. Essa classe suporta imagens do tipo BMP, JPG e GIF. As imagens podem ser obtidos através do sistema de arquivos ou rede, mas é mais fácil a opção de incluir a imagem como um recurso.

O objeto bitmap pode ser usado para desenhar imagens, formas ou texto (usando uma fonte). NETMF suporta criação de fontes usando a ferramenta TFConvert.

Quando nós desenhamos usando o objeto Bitmap, na verdade estamos desenhando no objeto (na memória) e nada é visível na tela. Para transferir a imagem da memória para a tela, devemos usar o método flush(). Um ponto importante aqui é que flush() só funciona se o tamanho do bitmap é idêntico ao da tela.

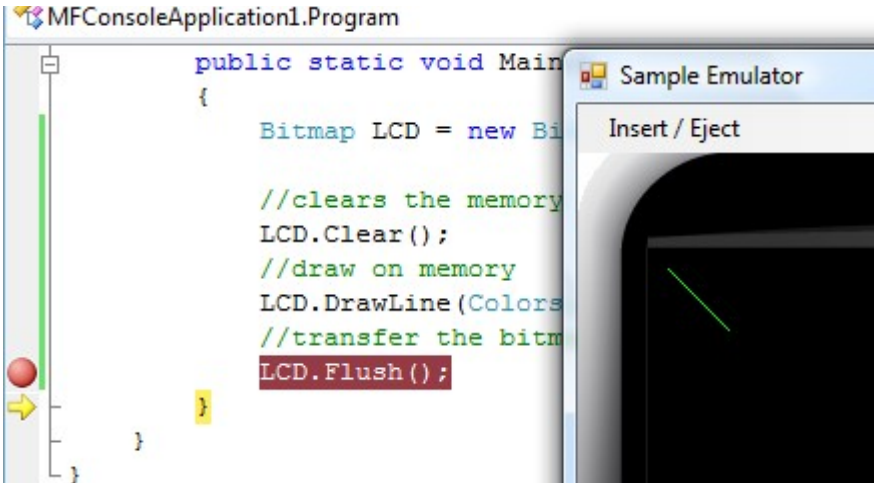
Se temos uma imagem 128x128 e quero mostrar na tela, é preciso primeiro criar um bitmap do tamanho da tela e uma segunda dimensão da imagem. Desenhe a imagem bitmap na tela e use flush () na tela. Para evitar confusão, eu sempre tenho um objeto bitmap chamado LCD e tudo é desenhado para este bitmap.

Vamos fazer os nossos testes no emulador, porque pode seu hardware não suportar nativamente os gráficos.

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

namespace MFConsoleApplication1
{
    public class Program
    {
        public static void Main()
        {
            Bitmap LCD = new Bitmap(SystemMetrics.ScreenWidth,
SystemMetrics.ScreenHeight);

            //Apaga a memória e não o LCD
            LCD.Clear();
            //Desenha na memória
            LCD.DrawLine(Colors.Green, 1, 10, 10, 40, 40);
            //transfere a memória bitmap para o display atual
            LCD.Flush();
        }
    }
}
```

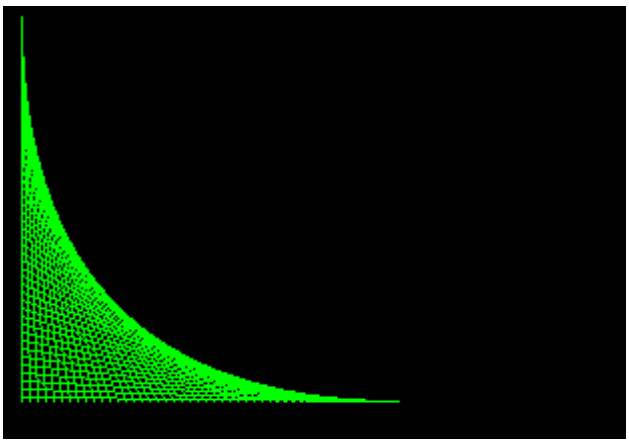

Devemos acrescentar o assembly “Microsoft.SPOT.TinyCore” para rodar. Então nós necessitamos usar o programa e fazer uma referência ao namespace "Presentation" de modo que nós possamos ter o "SystemMetrics”.

Execute o código e você terá uma linha verde na tela do emulador.



Tente usar o que nós aprendemos sobre o o loop for para criar várias linhas.

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

namespace MFConsoleApplication1
{
    public class Program
    {
        public static void Main()
        {
            Bitmap LCD = new Bitmap(SystemMetrics.ScreenWidth,
SystemMetrics.ScreenHeight);

            //Apaga a memória e não o display
            LCD.Clear();
            int i;
            for (i = 10; i < 200; i += 4)
            {
                //Desenha na memória
                LCD.DrawLine(Colors.Green, 1, 10, i, i, 200);
            }
            //transfere a memória bitmap para o display atual
            LCD.Flush();
        }
    }
}
```



Para escrever um texto, devemos primeiro ter o “recurso de fonte". Adicionar um novo recurso para o projeto. Você pode usar qualquer um dos recursos de arquivos enviados com os exemplos do SDK NETMF.

Os exemplos em \Documents\Microsoft. Micro Framework NET 4.0\Samples\

Eu uso o arquivo de fontes "NinaB.tinyfnt. Adicione o arquivo para os recursos como explanado no capítulo anterior. Nós podemos agora executar este programa para escrever no LCD.

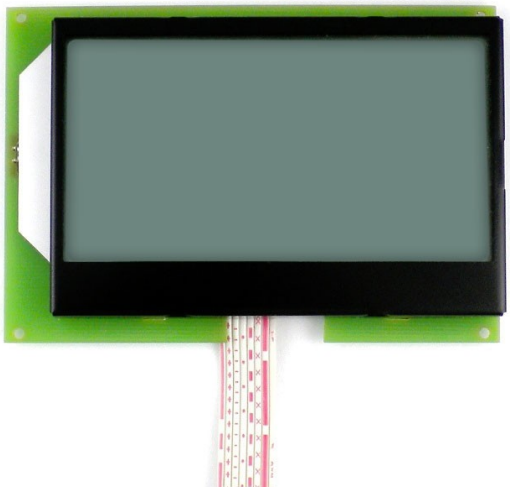
```
using System.Threading;
using Microsoft.SPOT;
using System;
using Microsoft.SPOT.Presentation;
using Microsoft.SPOT.Presentation.Media;

namespace MFConsoleApplication1
{
    public class Program
    {
        public static void Main()
        {
            Bitmap LCD = new Bitmap(SystemMetrics.ScreenWidth,
SystemMetrics.ScreenHeight);
            Font MyFont = Resources.GetFont(Resources.FontResources.NinaB);

            //apaga a memória e não o display
            LCD.Clear();
            int i;
            for (i = 10; i < 200; i += 4)
            {
                //desenha na memória
                LCD.DrawLine(Colors.Green, 1, 10, i, i, 200);
```

```
            }
            // imprime algum texto no bitmap
            LCD.DrawText("Still Amazing!", MyFont, Colors.Red, 100, 20);
            //transfere do bitmap para o display atual
            LCD.Flush();
        }
    }
}
```

Suporte não Nativo

Muitas pequenas telas gráficas usam o bus SPI para receber imagens do host. Dispositivos NETMF normalemente suportam grandes displays TFT que usam um bus especial, como Embedded Master e ChipworkX. Mas mesmo que o sistema não suporte aqueles displays, como USBizi e Fez, o usuário pode conectar um display baseado no SPI e mostrar os gráficos desta forma. É também possível ter 2 displays no sistema que suporte interface TFT nativa. Um grande display rodará na interface TFT e pequenos displays rodarão no bus SPI.

Embora SPI é muito rápido, as telas contêm milhões de pixels, é preferível escolher modelos com um acelerador gráfico. Www.TinyCLR.com tem muitos displays, SPI ou TFT que são ideais para trabalhar com o que a GHI oferece. Abaixo está um exemplo. Além disso, verifique o display padrão baseado no SPI para o FEZ Rhino no site acima mencionado.



Suporte Nativo para displays não padrões

A better option is to use the bitmap class to draw text, draw shapes and then transfer the bitmap to your display. You can only use this if Bitmap (graphics) is supported on your device, USBizi doesn't support graphics.

This code will display some data on the F-51852 128x64 pixel display found on the old non-TFT Embedded Master development system.

```
using System;
using System.Text;
using System.Threading;
using Microsoft.SPOT;
using Microsoft.SPOT.Presentation;
using Microsoft.SPOT.Presentation.Media;
using Microsoft.SPOT.Hardware;
using GHIElectronics.NETMF.System;
```

```
public class Program
{
    static SPI.Configuration conf = new SPI.Configuration((Cpu.Pin)33,
           false, 0, 0, false, true, 1000, SPI.SPI_module.SPI2);
    static SPI SPI_port = new SPI(conf);
    static OutputPort RST = new OutputPort((Cpu.Pin)9, true);
    static OutputPort DC = new OutputPort((Cpu.Pin)15, true);
    static byte[] _ba = new byte[1];
    static void OPTREX_SSD_WriteCmdByte(byte b)
    {
        DC.Write(false);

       Thread.Sleep(1);

        _ba[0] = b;
        SPI_port.Write(_ba);
    }

    static void OPTREX_SSD_WriteByte(byte b)
    {
        DC.Write(true);

        Thread.Sleep(1);

        _ba[0] = b;
        SPI_port.Write(_ba);
    }
    static void OPTREX_Locate(int x, int y)
    {
        if (y > 7)
            y = 7;

        if (x > 127)
            x = 127;

        OPTREX_SSD_WriteCmdByte((byte)(0X10 | (x >> 4)));//col up
        OPTREX_SSD_WriteCmdByte((byte)(0X00 | (x & 0xF)));//col down

        OPTREX_SSD_WriteCmdByte((byte)(0XB0 | y));//page addr

    }
    public static void Main()
    {

        OPTREX_SSD_WriteCmdByte(0XA2);//bias
        OPTREX_SSD_WriteCmdByte(0XA1);//adc   inverse
        OPTREX_SSD_WriteCmdByte(0Xc0);//common dir...normal
        OPTREX_SSD_WriteCmdByte(0X40);//initial line
        OPTREX_SSD_WriteCmdByte(0X81);//evr set
        OPTREX_SSD_WriteCmdByte(0X20);
        OPTREX_SSD_WriteCmdByte(0X29);//2B we have -10V........wait for
```

```
stable voltage
            Thread.Sleep(10);
            OPTREX_SSD_WriteCmdByte(0XA4);//turn all on
            OPTREX_SSD_WriteCmdByte(0XE7);//driver
            OPTREX_SSD_WriteCmdByte(0XAF);//lcd on

            //OPTREX_SSD_WriteCmdByte(0XA7);//inverse
            OPTREX_SSD_WriteCmdByte(0XA6);//no inverse


            OPTREX_SSD_WriteCmdByte(0XB0);//page addr
            OPTREX_SSD_WriteCmdByte(0X10);//col
            OPTREX_SSD_WriteCmdByte(0X00);//col

           int x = 20;
            int y = 50;
            int dx = -2;
            int dy = -3;

            Bitmap bb = new Bitmap(128, 64);
            byte[] bitmapbytes;
            Font fnt = Resources.GetFont(Resources.FontResources.small);
            byte[] vram = new byte[128 * 64 / 8];
            byte[] singleline = new byte[128];
            while (true)
            {
                bb.Clear();
                bb.SetPixel(0, 0, Color.White);
                bb.SetPixel(0, 2, Color.White);
                bb.SetPixel(2, 0, Color.White);
                bb.SetPixel(2, 2, Color.White);
                bb.DrawText("Rocks!", fnt, Color.White, 20, 45);
                bb.DrawEllipse(Color.White, x, y, 5, 3);

                x += dx;
                y += dy;
                if (x < 0 || x > 128)
                    dx = -dx;
                if (y < 0 || y > 64)
                    dy = -dy;

                bitmapbytes = bb.GetBitmap();
                Util.BitmapConvertBPP(bitmapbytes, vram,
Util.BPP_Type.BPP1_x128);

                for (int l = 0; l < 8; l++)
                {
                    OPTREX_Locate(0, l);
                    DC.Write(true);
                    Array.Copy(vram, l * 128, singleline, 0, 128);
                    SPI_port.Write(singleline);
                }
```

```
                Thread.Sleep(1);
            }
        }
    }
```

21. Serviços de Tempo

Em um computador, tempo significa duas coisas. O tempo do sistema, o que representa os ticks do processador , utilizado para controlar a temporização das threads e toda a gestão de termporização do sistema. Por outro lado, existe o relógio de tempo real (RTC), que é usado para refletir o tempo do ser humano, como as horas, minutos ou dias, por exemplo

21.1. Relógio em tempo real (RTC)

Manter um RTC executando enquanto o sistema desligado exige um oscilador 32.768KHZ e uma bateria de 3V (moeda) em um pino específico, geralmente chamado Vbat. Todas as placas e módulos GHI exceto FEZ Mini já possuem um cristal de 32.768KHZ. Apenas uma bateria de 3V é necessário.

É importante compreender que os serviços de tempo em NETMF não estão diretamente ligados ao RTC. Por exemplo, você pode configura a "hora do sistema NETMF", a partir desse ponto em diante, o tempo vai estar precisamente disponível. Isso não significa que o hardware RTC tem o tempo correto. Além disso, um usuário pode ler a hora de um servidor da hora pela rede e ter a hora correta em vez de usar o RTC.

Aqui está um exemplo que configura a hora primeiro e a imprime a cada 100ms. Isto define a hora no NETMF e não no RTC. A falta de energia pode causar o relógio resetar para um número randômico.

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

namespace MFConsoleApplication1
{
    public class Program
    {
        public static void Main()
        {
            // configura a data e hora para  9/9/2010 às 09:09:09
            DateTime time = new DateTime(2010, 9, 9, 9, 9, 9);
            Utility.SetLocalTime(time);
            while (true)
            {
                Debug.Print(DateTime.Now.ToString());
                Thread.Sleep(100);

            }
```

```
        }
```

Para usar o hardware RTC, precisamos primeiro verificar se o hardware RTC tem uma hora válida ou não. Talvez isto acontece porque foi colocada uma nova bateria ou um novo sistema e o RTC não tenha sido configurado. Se a RTC tem uma hora válido, então podemos ler o RTC (hardware) e usar esse tempo para definir a hora do sistema NETMF (software). Se a hora não é válido, então você precisa definir o RTC para a hora correta.

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

    public class Program
    {
        public static void Main()
        {
            // To keep track of time, set it at the beginning of your application
from the RTC.

            // If it was NOT set before and currently running using the battery
(not exhausted), set it to a fixed time.
            if (RealTimeClock.IsTimeValid == false)
                RealTimeClock.SetTime(new DateTime(2010, 03, 01, 12, 0, 0, 0));

            Utility.SetLocalTime(RealTimeClock.GetTime());
        }
  }
```

21.2. Timers

O Micro Framework inclui duas classes de temporizadores: Timer e ExtendedTimer. A classe Timer é idêntica à que está presente full framework onde ExtendedTimer é específico do NETMF com funcionalidade extra. Para aplicações de iniciantes, eu sugiro que você continue utilizando threads antes de partir para os Timers. Eu vou mostrar um exemplo aqui neste livro. Este programa irá criar um timer que vai começar depois de 5 segundos e, então, ativado a cada segundo.

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

namespace MFConsoleApplication1
{
    public class Program
    {
```

```
        static void RunMe(object o)
        {
            Debug.Print("From timer!");
        }
        public static void Main()
        {
            Timer MyTimer = new Timer(new TimerCallback(RunMe), null, 5000,
1000);
            Debug.Print("The timer will fire in 5 seconds and then fire
priodically every 1 second");
            Thread.Sleep(Timeout.Infinite);
        }
    }
}
```

22. USB Host

Há muitas vezes uma confusão entre o "host USB" e "dispositivo USB". O host USB é o hardware que se conecta a vários outros dispositivos USB. Por exemplo, o PC tem um host USB, porque pode conectar diversos dispositivos USB, como mouses, teclados e discos rígidos. A implementação de um dispositivo USB é relativamente simples, então implémenter um hospedeiro é muito mais complicado.

O host USB é uma característica exclusiva da GHI Electronics. Com ele, você pode conectar praticamente qualquer dispositivo USB nos produtos NETMF da GHI (UsBizi, Embedded Master ChipworkX). Ele abre novos horizontes para um sistema embarcado. O produto já pode se conectar a um teclado USB padrão, ou mesmo o acesso a um pendrive USB!

USB é um sistema plugável, o que significa que os dispositivos podem ser conectados e desconectados a qualquer momento. Eventos são gerados quando há conexões/ desconexões. O desenvolvedor deve registrar estes eventos para gerenciar os dispositivos conectados. Como este é um livro para iniciantes, vou assumir que o dispositivo já está conectado ao sistema.

Com hub USB, vários dispositivos podem ser conectados diretamente. Hub usb não pode ser conectado em Hub usb.

Em primeiro lugar, deixe nos detectar quais os dispositivos estão conectados. A primeira coisa a fazer é iniciar o administrador do sistema para nos fornecer a lista de dispositivos disponíveis. Não se esqueça de adicionar o assembly correspondente ao seu projeto.

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

using GHIElectronics.NETMF.USBHost;

namespace Test
{
    class Program
    {
        public static void Main()
        {
            // Inscreve eventos para USBH
            USBHostController.DeviceConnectedEvent += DeviceConnectedEvent;
            USBHostController.DeviceDisconnectedEvent +=
DeviceDisconnectedEvent;

            // Loop para sempre
            Thread.Sleep(Timeout.Infinite);
```

```
        }

        static void DeviceConnectedEvent(USBH_Device device)
        {
            Debug.Print("Device connected...");
            Debug.Print("ID: " + device.ID + ", Interface: " +
device.INTERFACE_INDEX + ", Type: " + device.TYPE);
        }

        static void DeviceDisconnectedEvent(USBH_Device device)
        {
            Debug.Print("Device disconnected...");
            Debug.Print("ID: " + device.ID + ", Interface: " +
device.INTERFACE_INDEX + ", Type: " + device.TYPE);
        }
    }
}
```

Quando nós detectamos um dispositivo, nós podemos comunicar com ele diretamente. Isto requer um monte de conhecimento sobre dispositivos USB. Felizmente, a maioria dos dispositivos abrangidos por classes padrão e GHI já oferecem drivers para eles.

22.1. Dispositivos HID

HID significa Human Interface Devices (dispositivo de interface humana), mouses, teclados e joysticks são suportados diretamente. O uso da HID é através de eventos. Os eventos são métodos que você cria e que você inscreve para certo evento. Quando esse evento ocorre, seu método é executado automaticamente.

```
using System;
using System.Threading;
using Microsoft.SPOT;
using GHIElectronics.NETMF.USBHost;

namespace Test
{
    public class Program
    {
        static USBH_Mouse mouse;
        public static void Main()
        {
            //  Inscreve eventos para USBH
            USBHostController.DeviceConnectedEvent += DeviceConnectedEvent;

            // Loop para sempre
            Thread.Sleep(Timeout.Infinite);
        }
```

```
        static void DeviceConnectedEvent(USBH_Device device)
        {
            if (device.TYPE == USBH_DeviceType.Mouse)
            {
                Debug.Print("Mouse Connected");
                mouse = new USBH_Mouse(device);
                mouse.MouseMove += MouseMove;
                mouse.MouseDown += MouseDown;
            }
        }

        static void MouseMove(USBH_Mouse sender, USBH_MouseEventArgs args)
        {
            Debug.Print("(x, y) = (" + sender.Cursor.X + ", " +
sender.Cursor.Y + ")");
        }

        static void MouseDown(USBH_Mouse sender, USBH_MouseEventArgs args)
        {
            Debug.Print("Button down number: " + args.ChangedButton);
        }
    }
}
```

Acessar o joysticks é muito similar. Aqui o mesmo exemplo, mas modificado para funcionar com um joystick.

```
using System;
using System.Threading;
using Microsoft.SPOT;
using GHIElectronics.NETMF.USBHost;
namespace Test
{
    public class Program
    {
        static USBH_Joystick j;
        public static void Main()
        {
            // Inscreve eventos para USBH
            USBHostController.DeviceConnectedEvent += DeviceConnectedEvent;

            // Loop para sempre
            Thread.Sleep(Timeout.Infinite);
        }

        static void DeviceConnectedEvent(USBH_Device device)
        {
            if (device.TYPE == USBH_DeviceType.Joystick)
            {
```

```
                Debug.Print("Joystick Connected");
                j = new USBH_Joystick(device);
                j.JoystickXYMove += JoystickXYMove;
                j.JoystickButtonDown += JoystickButtonDown;
            }
        }

        static void JoystickButtonDown(USBH_Joystick sender,
USBH_JoystickEventArgs args)
        {
            Debug.Print("Button Pressed: " + args.ChangedButton);
        }

        static void JoystickXYMove(USBH_Joystick sender,
USBH_JoystickEventArgs args)
        {
            Debug.Print("(x, y) = (" + sender.Cursor.X + ", " +
sender.Cursor.Y + ")");
        }
    }
}
```

22.2. Dispositivos Seriais

A interface serial (UART) é muito comum. Há uma abundância de fabricantes que criam chips que convertem USB para serial. GHI suporta os circuitos produzidos pela FTDI e Silabs Prolific.

Além disso, existe uma classe definida pelo USB padrão para comunicação serial: CDC (Communication Device Class). Esta classe também é suportado.

Note aqui que esses circuitos USB são projetados para serem customizados. Por exemplo, uma empresa que usa um chipset FTDI em que seus produtos, ela pode trocar as strings nos descritores do USB de forma que quando plugarem num PC, poderá ver o nome da companhia e não da FTDI. Um bom exemplo é um dispositivo USB de GPS. Quase todos os dispositivos USB GPS usar chip prolifit, que é suportado pela GHI. Muitos dos produtos de interface no mercado de chipset, usam FTDI.

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

using GHIElectronics.NETMF.USBHost;

namespace Test
{
```

```
    class Program
    {
        static USBH_SerialUSB serialUSB;
        static Thread serialUSBThread; // Imprime dados a cada segundo

        public static void Main()
        {
            //  Inscreve eventos para USBH
            USBHostController.DeviceConnectedEvent += DeviceConnectedEvent;

            // Loop para sempre
            Thread.Sleep(Timeout.Infinite);
        }

        static void DeviceConnectedEvent(USBH_Device device)
        {
            Debug.Print("Device connected");

            switch (device.TYPE)
            {
                case USBH_DeviceType.Serial_FTDI: // FTDI conectado
                    serialUSB = new USBH_SerialUSB(device, 9600,
System.IO.Ports.Parity.None, 8, System.IO.Ports.StopBits.One);
                    serialUSB.Open();
                    serialUSBThread = new Thread(SerialUSBThread);
                    serialUSBThread.Start();

                    break;

                case USBH_DeviceType.Unknown: // SiLabs não pode ser
reconhecido
                    // força SiLabs
                    USBH_Device silabs = new USBH_Device(device.ID,
device.INTERFACE_INDEX, USBH_DeviceType.Serial_SiLabs, device.VENDOR_ID,
device.PRODUCT_ID, device.PORT_NUMBER);
                    serialUSB = new USBH_SerialUSB(silabs, 9600,
System.IO.Ports.Parity.None, 8, System.IO.Ports.StopBits.One);
                    serialUSB.Open();
                    serialUSBThread = new Thread(SerialUSBThread);
                    serialUSBThread.Start();

                    break;
            }
        }

        static void SerialUSBThread()
        {
            // Imprime "Hello World!" a cada segundo
            byte[] data = Encoding.UTF8.GetBytes("Hello World!\r\n");
            while (true)
            {
                Thread.Sleep(1000);
```

```
                serialUSB.Write(data, 0, data.Length);
            }
        }
    }
}
```

22.3. Dispositivo de Armazenamento

Os dispositivos de armazenamento como hard drives USB e drives de memória USB usam mesma classe MSC (Mass Storage Class). A biblioteca da GHI suporta diretamente esses dispositivos. USB apenas define como ler / escrever setores na mídia. O sistema operacional lida com o sistema de arquivos. NETMF suporta sistema de arquivos FAT32 e FA16. Para acessar os arquivos em uma mídia USB, precisamos primeiro detectá-lo, então precisamos para montar a mídia.

```
using System;
using System.IO;
using System.Threading;
using Microsoft.SPOT;
using Microsoft.SPOT.IO;
using GHIElectronics.NETMF.IO;
using GHIElectronics.NETMF.USBHost;

namespace Test
{
    class Program
    {
        public static void Main()
        {
            // Inscreve eventos para RemovableMedia
            RemovableMedia.Insert += RemovableMedia_Insert;
            RemovableMedia.Eject += RemovableMedia_Eject;

            // Inscreve para eventos USB
            USBHostController.DeviceConnectedEvent += DeviceConnectedEvent;

            // loop para sempre
            Thread.Sleep(Timeout.Infinite);
        }

        static void DeviceConnectedEvent(USBH_Device device)
        {
            if (device.TYPE == USBH_DeviceType.MassStorage)
            {
                Debug.Print("USB Mass Storage detected...");
```

```
                //....
                //....
            }
        }
    }
}
```

A próxima seção explana como acessar arquivos na memória USB.

23. Sistema de Arquivos

O sistema de arquivos suportado pelo NETMF 4.0 adiciona vários recursos para o suporte padrão. Por exemplo, um cartão SD pode ser montado no sistema de arquivos ou serviço MSC do dispositivo USB. Quando ele está montado no sistema de arquivos, os desenvolvedores podem acessar os arquivos. Se ele é montado sobre um dispositivo MSC USB, um PC conectado à porta USB irá ver um leitor de cartões USB com cartão SD inserido. Isso é útil para a criação de um data logger. O dispositivo irá logar dados no cartão SD e quando o dispositivo for plugado no PC, o dispositivo será um leitor de cartão for o mesmo cartão de memória SD. A classe de armazenamento persistente da GHI é usado para gerenciar a instalação do sistema de arquivos no sistema.

Esta seção irá cobrir somente o uso de dispositivos de armazenamento no sistema de arquivos.

23.1. Cartões SD

Primeiro, precisamos detectar a inserção do cartão SD. Conectores de cartões SD geralmente tem switch interno pequeno, que fecha quando o cartão é inserido. Neste exemplo, vou assumir que o cartão está inserido e que não há necessidade de detectá-lo. O programa irá listar todos os arquivos no diretório raiz.

```
using System;
using System.IO;
using System.Threading;

using Microsoft.SPOT;
using Microsoft.SPOT.IO;

using GHIElectronics.NETMF.IO;

namespace Test
{
    class Program
    {
        public static void Main()
        {
            // ...
            // Cartão SD é inserido
            // Cria um novo dispositivo de armazenamento
            PersistentStorage sdPS = new PersistentStorage("SD");

            // Monta o arquivo de sistemas
```

```
            sdPS.MountFileSystem();

            //Assume que um dispositivo de armazenamento é disponível, acesse
o através do Micro Framework e mostre no display os arquivos e folders
disponíveis
            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.");
            }

            // Desmonte
            sdPS.UnmountFileSystem();
        }
    }
}
```

Há várias maneiras de se abrir arquivos. Vou tratar aqui de objetos FilteStream. Este exemplo abre um arquivo para escrever uma string dentro. Como FileStream aceita apenas arrays de bytes, teremos que converter a nossa string em um array de bytes.

```
using System.Threading;
using System.Text;
using Microsoft.SPOT;
using System.IO;
using Microsoft.SPOT.IO;
using GHIElectronics.NETMF.IO;

namespace MFConsoleApplication1
{
    public class Program
    {
        static void Main()
```

```
        {
            // ... check se o SD está inserido

            // SD Card está inserido
            // Crie um novo dispositivo de armazenamento
            PersistentStorage sdPS = new PersistentStorage("SD");

            // Monte o sistema de arquivos
            sdPS.MountFileSystem();

            // Assume que um dispositivo de armazenameno está disponivel
acesse ele pelo NETMF
            string rootDirectory = VolumeInfo.GetVolumes()[0].RootDirectory;
            FileStream FileHandle = new FileStream(rootDirectory +
@"\hello.txt", FileMode.Create);
            byte[] data = Encoding.UTF8.GetBytes("This string will go in the
file!");
            // escreva dados e feche o arquivo
            FileHandle.Write(data, 0, data.Length);
            FileHandle.Close();

            //se nós necessitarmos desmontar
            sdPS.UnmountFileSystem();

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

        }
    }
}
```

Você pode verificar o arquivo com leitor de cartão do seu PC e verificar se o arquivo está presente. Agora, vamos abrir o arquivo e ler a string escrita anteriormente.

```
using System.Threading;
using System.Text;
using Microsoft.SPOT;
using System.IO;
using Microsoft.SPOT.IO;
using GHIElectronics.NETMF.IO;

namespace MFConsoleApplication1
{
    public class Program
    {
        static void Main()
        {
            // ... check se SD está inserido
```

```
            // SD Card está inserido
            // Cria um dispositivo de armazenamento
            PersistentStorage sdPS = new PersistentStorage("SD");

            // Monte o sistema de arquivos
            sdPS.MountFileSystem();

            // Assume que um dispositivo de armazenameno está disponivel
acesse ele pelo NETMF
            string rootDirectory = VolumeInfo.GetVolumes()[0].RootDirectory;
            FileStream FileHandle = new FileStream(rootDirectory +
@"\hello.txt", FileMode.Open, FileAccess.Read);
            byte[] data = new byte[100];
            // Escreva dados e feche o arquivo
            int read_count = FileHandle.Read(data, 0, data.Length);
            FileHandle.Close();
            Debug.Print("The size of data we read is: " +
read_count.ToString());
            Debug.Print("Data from file:");
            Debug.Print(data.ToString());

            // se nós necessitamos desmontar
            sdPS.UnmountFileSystem();

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

        }
    }
}
```

23.2. Discos USB (Disco de Armazenamento em Massa)

Os arquivos são tratados no USB exatamente da mesma maneira que é feito em cartões SD. A única diferença é como vamos detectar e como montá-lo. Para SD, você pode usar uma entrada para o pino detectá-los. Com um drive USB, usamos eventos para detectar uma nova mídia.

```
using System;
using System.IO;
using System.Threading;
using Microsoft.SPOT;
using Microsoft.SPOT.IO;
using GHIElectronics.NETMF.IO;
using GHIElectronics.NETMF.USBHost;
```

```
namespace Test
{
    class Program
    {
        // Mantém uma referência no caso que o GC cai fora e libere-o
automaticamente, note que somente temos suporte neste exemplo
        static PersistentStorage ps;

        public static void Main()
        {
            // Inscreva para eventos RemovableMedia
            RemovableMedia.Insert += RemovableMedia_Insert;
            RemovableMedia.Eject += RemovableMedia_Eject;

            // Inscreva para eventos USB
            USBHostController.DeviceConnectedEvent += DeviceConnectedEvent;

            // Loop para sempre
            Thread.Sleep(Timeout.Infinite);
        }

        static void DeviceConnectedEvent(USBH_Device device)
        {
            if (device.TYPE == USBH_DeviceType.MassStorage)
            {
                Debug.Print("USB Mass Storage detected...");
                ps = new PersistentStorage(device);
                ps.MountFileSystem();
            }
        }

        static void RemovableMedia_Insert(object sender, MediaEventArgs e)
        {
            Debug.Print("Storage \"" + e.Volume.RootDirectory + "\" is
inserted.");
            Debug.Print("Getting files and folders:");
            if (e.Volume.IsFormatted)
            {
                string[] files = Directory.GetFiles(e.Volume.RootDirectory);
                string[] folders =
Directory.GetDirectories(e.Volume.RootDirectory);

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

                Debug.Print("Folders available on " + e.Volume.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.");
            }
        }

        static void RemovableMedia_Eject(object sender, MediaEventArgs e)
        {
            Debug.Print("Storage \"" + e.Volume.RootDirectory + "\" is
ejected.");
        }
    }
}
```

Podemos ver com o código acima com montar um drive USB para o arquivo de sistemas, tudo funciona exatamente como os cartões SD.

23.3. Considerações nos arquivos de sistemas

NETMF 4.0 suporta sistema de arquivos FAT32 e FAT16. A mídia formatado para FAT12 não irá funcionar.

O sistema de arquivos tem internamente um monte de bufferização para acelerar o acesso a arquivos e aumentar o tempo de vida da mídia. Quando você gravar dados em um arquivo, não está claro se os dados são gravados no cartão. Eles serão gravados, provavelmente, em algum lugar na memória, no buffer interno. Para garantir que os dados sejam gravados na mídia, você deve limpar os buffers usando o método flush(). Fazendo o flush ou fechando o arquivo arquivo é a única maneira de garantir que os dados são realmente gravados na mídia.

Isso é no nível de arquivo. No nível de mídia, há também outras informações que podem não ter efeito imediato. Por exemplo, se você excluir um arquivo e remover o cartão, nada indica que o arquivo é realmente apagado. Para ter certeza, você deve usar o método VolumeInfo.FlushALL.

Idealmente, você desmontar (Unmount) a midia que é removido do sistema. Isto pode nem sempre ser possível, portanto, um Flush ou FlushAll na mídia irá garantir que seus dados tenham sido salvos e sem perda de dados se a mídia for removida em algum momento.

24. Criando Redes (Networking)

A rede é uma parte essencial do nosso trabalho e vida. Quase todas casas está conectadas a rede que tenha Internet e a maioria das empresas pode não funcionar sem uma rede interna (LAN ou WIFI) que é conectado com uma rede externa. Todas as redes tem uma forma de comunicação, eles todos usam o protocolo TCP/IP. Há atualmente vários protocolos que manuseiam diferentes atividades na rede, como DNS, DHCP, IP, ICMP, TCP, UDP, PPP e muito mais.

NETMF suporta redes TCP/IP através dos sockets padrões do .NET. Um socket é uma conexão virtual entre 2 dispositivos em uma rede.

GHI extendeu o suporte TCP/IP para PPP e WIFI. Através do PPP, 2 dispositivos podem se conectar através de uma conexão serial. A conexão serial pode ser um modem de linha telefônica ou um modem 3G/GPRS ou modem analógico. Com PPP, dispositivos NETMF podem conectar a internet usando telefones celulares 3G/GPRS. É Também possível conectar 2 dispositivos NETMF através de fios ou conexão serial sem fio (Xbee / Bluetooth / outros).

Além disso, com suporte a WIFI, dispositivos NETMF podem conectar para redes wireless seguras. NETMF tambem também suporta conexão segura para SSL.

O suporte para rede é padrão e completa (HTTP, SSL, Sockets,etc) no dispositivo EMX, Embedded Master e ChipworkX.

24.1. Suporte a rede com USBizi (FEZ)

Para gerenciar uma rede maior e que ocupa mais memória, USBizi precisa de um apoio externo. Por exemplo, o USBizi gerencia a rede através do chipset Wiznet W5100. Quando USBizi quer criar uma conexão de rede, a única coisa a fazer é enviar a solicitação para Wiznet 5100 e que o chip vai fazer o resto.

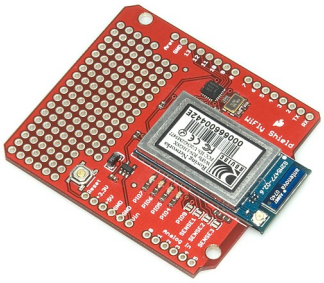
O chip Wiznet W5100 funciona com o barramento SPI. Há o driver da GHI que gerencia o módulo W5100 pelo C# . Este driver é uma versão beta, ou seja, o driver ainda esta em desenvolvimento. O código está disponível para os usuários optimizarem e torná-lo mais eficiente.

24.2. TCP/IP bruto ou Sockets

Esta é uma área onde a maioria dos designers podem cometer erros. Ao selecionar um módulo WiFi ou GPRS/3G, existem duas categorias de módulos para escolher. Módulo com "Sockets" ou "módulos com TCP/IP bruto". Módulos com sockets TCP/IP gerenciam todo o trabalho TCP/IP internamente e dá-lhe o acesso a bases de alto nível. Isso significa que o trabalho mais difícil é feito internamente no módulo. Depois de ter atribuído um endereço e uma porta, você só precisa enviar/receber dados a partir do módulo. Problema: ele é muito limitado. Por exemplo, número de soquetes. Mesmo se seu sistema é muito poderoso, tem muitos MB de RAM, você está limitado pela funcionalidade do módulo. Por esta razão, no entanto, o uso de tais módulos "alto nível" é ideal para pequenos sistemas.

Tomemos um exemplo: Roving Networks oferece um módulo chamado WiFly. Este módulo contém uma rede TCP/IP e um único soquete. Usando este módulo é muito simples, já que você conecta a uma placa de porta serial, em seguida, através de simples comandos em série, você pode ler/escrever dados de um socket disponível. Isso é suficiente para fazer um pequeno servidor web, uma conexão telnet ou transferência de dados. É perfeito para pequenos sistemas e recursos limitados, tais como a memória USBizi (FEZ). O módulo faz tudo para você, basta enviar e receber dados seriais

Se você implementar um servidor web que fornece certos valores, tais como temperatura ou umidade, por isso é tudo que você precisa e pode ser implementado de forma fácil e mais barato com USBizi (FEZ). Um simples protótipo pode ser feito ligando o SHIELD WiFly ao FEZ Domino. Na imagem à direita, veja o SHIELD Wifly da SparkFun.



A desvantagem destes módulos simples é que são em número muito limitado de sockets. E se você precisa de mais? Ou que se pretende implementar uma conexão SSL? Para isso, você terá um módulo que não suporta TCP/IP internamente. O Trabalho do TCP/IP deve ser feito fora do módulo. Aqui que entram os módulos EMX e ChipworkX. Estes módulos são potentes, com muitos recursos. Eles têm uma pilha TCP/IP interrna de SSL / HTTP / DHCP, etc. Conectando o módulo ZEROG ao EMX ou ChipworkX vai aumentar o poder do TCP/IP e permitir também conexões seguras.

Que tal modems GPRS e 3G? É a mesma coisa. Alguns têm socket nativo, como SM5100B enquanto outros trabalham com PPP como um modem de PC (o módulo da Telit, por exemplo). Se você precisar de uma conexão de rede com TCP/IP completo, vá direto para EMX ou ChipworkX com modems padrões PPP, assim como um PC que se conecta à internet com um modem.

Se você precisa de uma conexão barata e fácil, então USBizi (FEZ) pode ser usado com SM5100B. O SHIELD da SparkFun Celular mostrada à direita é montado diretamente sobre o FEZ/ Domino.

24.3. Sockets padrões .NET

O suporte a Socket no NETMF é muito semelhante ao .NET Framework. O NETMF SDK tem muitos exemplos para uso de sockets, cliente e servidor. Também muitos projetos estão disponíveis exibindo as diferentes possibilidades.

Cliente Twitter:

http://www.microframeworkprojects.com/index.php?title=MFTwitter

Google maps:

http://www.microframeworkprojects.com/index.php?title=GoogleMaps

Cliente RSS:

http://www.microframeworkprojects.com/index.php?title=RSS_n_Weather

Rádio Internet MP3:

http://www.microframeworkprojects.com/index.php?title=ShoutcastClient

Servidor Web:

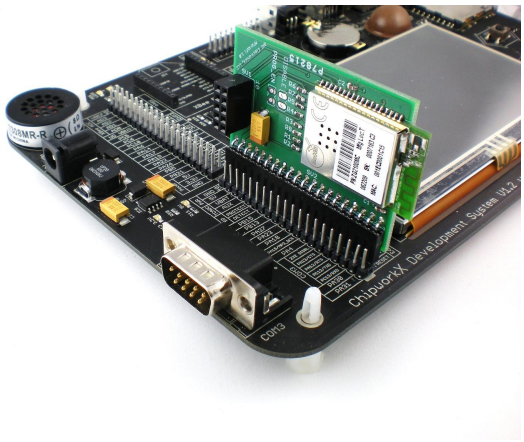
http://www.microframeworkprojects.com/index.php?title=WebServer

24.4. Wi-Fi (802.11)

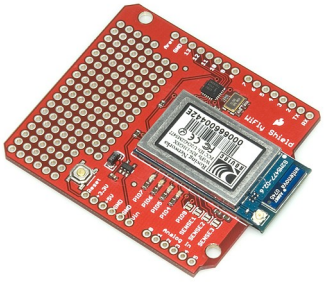
WiFi é muito comum em redes. Ele permite transferências seguras de dados a velocidades altas entre várias conexões. Ele também permite a comunicação entre dois nós (Ad-Hoc). A maneira mais fácil é conectar vários computadores a um ponto de acesso (access point). O acesso não é simples e certamente, não se destina a ser incorporado facilmente. GHI Electronics é a única empresa que oferece a opção de WiFi NETMF em seus dispositivos.

WiFi é projetado para funcionar com pilha TCP/IP (sockets em rede NETMF) e pode ser utilizado em sistemas que já suportam TCP/IP como EMX e ChipworkX.

Wi-Fi apoiado pela GHI usa os chips e ZG2100 (com antena interna e externa respecitivamente) da ZeroG. Para o protótipo, a placa WiFi é um bom começo.



Como observado anteriormente, USBizi (FEZ) pode ser usado com módulos que contêm suporte TCP/IP e socket interno como o Shield Sparkfun que contêm o módulo WiFly da Roving Networks.



24.5. Redes móveis com GPRS e 3G

EMX e ChipworkX tem o stack TCP/IP interno e suporte a PPP internamente. Você pode conectar um modem padrão e usá-lo em alguns passos.

No USBizi, uma conexão a uma rede móvel pode ser estabelecida através de um modem com TCP / IP interno como o SM5100B. Abaixo o Shield com o SM5100B da Sparkfun.

25. Criptografia

A criptografia tem sido uma parte importante da tecnologia por muitos anos. Modernos algoritmos de criptografia podem necessitar de muitos recursos. Já que os cartões NETMF são destinados para dispositivos pequenos, a equipe NETMF teve que escolher quais algoritmos eles devem suportar. Esses algoritmos são XTEA e RSA.

25.1. XTEA

XTEA, com a sua chave de 16 bytes (128 bits) é considerado muito seguro e não requer muito poder de processamento. Originalmente XTEA foi projetado para trabalhar em "pedaços" de 8 bytes apenas. Isto pode ser problemático para a criptografia de dados cujo tamanho não é um múltiplo de 8. A implementação de XTEA em NETMF permite a criptografia de dados de qualquer tamanho.

A criptografia e descriptografia são fáceis de fazer. Está aqui um exemplo:

```
using System;
using System.Text;
using Microsoft.SPOT;
using Microsoft.SPOT.Cryptography;
public class Program
{
    public static void Main()
    {
        // 16-byte 128-bit key (chave)
        byte[] XTEA_key = new byte[] { 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12,
13, 14, 15, 16 };
        Key_TinyEncryptionAlgorithm xtea = new
Key_TinyEncryptionAlgorithm(XTEA_key);
        // Dado que nós queremos incriptart
        string original_string = "FEZ is so easy!";//must be more than 8
bytes
        //Converte para array de bytes
        byte[] original_data = UTF8Encoding.UTF8.GetBytes(original_string);

        //Encripte o dado
        byte[] encrypted_bytes = xtea.Encrypt(original_data, 0,
original_data.Length, null);
        //descripte o dado
        byte[] decrypted_bytes = xtea.Decrypt(encrypted_bytes, 0,
encrypted_bytes.Length, null);
        //imprima o dado descriptado
        string decrypted_string = new
string(Encoding.UTF8.GetChars(decrypted_bytes));
        Debug.Print(original_string);
```



```
            Debug.Print(decrypted_string);
        }
}
```

Os dados criptografados têm o mesmo tamanho que os dados brutos.

XTEA nos PCs

Agora você pode compartilhar dados entre dispositivos NETMF de modo seguro usando XTEA, mas como a troca de dados com um PC ou outro sistema? O livro "expert. NET Micro Framework Second Edition" de Jens Kuhner fornece o código fonte que mostra como implementar XTEA em um PC. Usando o código de Jens, você pode criptografar os dados e enviar-lhes a um dispositvo NETMF e vice-versa. Se você não tiver o livro, o código-fonte é online, capítulo 8:

Você pode baixá-lo aqui: http://apress.com/book/view/9781590599730

25.2. RSA

XTEA é seguro, mas tem uma limitação: a chave deve ser compartilhada. Para compartilhar dados criptografados, os dois sistemas devem primeiro conhecer a chave usada. Se um sistema tenta enviar a chave para o outro, alguém poderia espiar a linha e pegar a chave e, assim, descriptografar os dados. Isto é francamente não mais seguro!

RSA contorna este problema oferecendo uma combinação de chave pública / chave privada. Isso pode parecer absurdo, mas a chave usada para criptografar não pode ser usado para descriptografar. A outra chave deve ser utilizada para descriptografar os dados. Suponha que o sistema de 'A' deve ler sistema de dados seguro 'B'. A primeira coisa que 'A' deve fazer é enviar uma chave pública ao sistema 'B'. O sitema 'B', então irá encriptar os dados com a chave pública e enviar os dados para o PC. Um hacker pode ver os dados codificados e a chave pública, mas sem a chave privada de descriptografia é praticamente impossível. Finalmente, o sistema de 'A' pode descriptografar os dados usando sua chave privada. A propósito, é assim que os sites seguros funcionam.

Os dispositivos NETMF não podem gerar chaves. Estes são gerados em um PC com um programa chamado MetaDataProcessor. Abra uma linha de comando e digite as instruções abaixo

cd “C:\Program Files (x86)\Microsoft .NET Micro Framework\v4.0\Tools”

Isso depende do seu diretório de instalação do NET Micro Framework.

Gere as chaves, fazendo isso:

MetadataProcessor.exe create_key_pair-c: \private.bin \c: public.bin

As teclas são codificadas em binário, mas esta ferramenta pode convertê-los em texto legível:

MetadataProcessor.exe dump_key c:\public.bin >> c:\public.txt

MetadataProcessor.exe dump_key c:\private.bin >> c:\private.txt

Agora copie a chave do nosso programa de exemplo. Note que a chave pública sempre começa com 1,0,1 e o resto são 0. Permite-nos optimizar o nosso código um pouco, conforme visto abaixo:

```
using System;
using System.Text;
using Microsoft.SPOT;
using Microsoft.SPOT.Cryptography;
public class Program
{
    public static void Main()
    {
        //isto é compartilhado entre keys publicas e privadas (chaves)
        byte[] module = new byte[] { 0x17, 0xe5, 0x27, 0x40, 0xa9, 0x15,
0xbd, 0xfa, 0xac, 0x45, 0xb1, 0xb8, 0xe1, 0x7d, 0xf7, 0x8b, 0x6c, 0xbd, 0xd5,
0xe3, 0xf4, 0x08, 0x83, 0xde, 0xd6, 0x4b, 0xd0, 0x6c, 0x76, 0x65, 0x17, 0x52,
0xaf, 0x1c, 0x50, 0xff, 0x10, 0xe8, 0x4f, 0x4f, 0x96, 0x99, 0x5e, 0x24, 0x4c,
0xfd, 0x60, 0xbe, 0x00, 0xdc, 0x07, 0x50, 0x6d, 0xfd, 0xcb, 0x70, 0x9c, 0xe6,
0xb1, 0xaf, 0xdb, 0xca, 0x7e, 0x91, 0x36, 0xd4, 0x2b, 0xf9, 0x51, 0x6c, 0x33,
0xcf, 0xbf, 0xdd, 0x69, 0xfc, 0x49, 0x87, 0x3e, 0x1f, 0x76, 0x20, 0x53, 0xc6,
0x2e, 0x37, 0xfa, 0x83, 0x3d, 0xf0, 0xdc, 0x16, 0x3f, 0x16, 0xe8, 0x0e, 0xa4,
0xcf, 0xcf, 0x2f, 0x77, 0x6c, 0x1b, 0xe1, 0x88, 0xbd, 0x32, 0xbf, 0x95, 0x2f,
0x86, 0xbb, 0xf9, 0xb4, 0x42, 0xcd, 0xae, 0x0b, 0x92, 0x6a, 0x74, 0xa0, 0xaf,
0x5a, 0xf9, 0xb3, 0x75, 0xa3 };
        //A chave privada...para descriptar
        byte[] private_key = new byte[] { 0xb9, 0x1c, 0x24, 0xca, 0xc8, 0xe8,
0x3d, 0x35, 0x60, 0xfc, 0x76, 0xb5, 0x71, 0x49, 0xa5, 0x0e, 0xdd, 0xc8, 0x6b,
0x34, 0x23, 0x94, 0x78, 0x65, 0x48, 0x5a, 0x54, 0x71, 0xd4, 0x1a, 0x35, 0x20,
0x00, 0xc6, 0x0c, 0x04, 0x7e, 0xf0, 0x34, 0x8f, 0x66, 0x7f, 0x8a, 0x29, 0x02,
0x5e, 0xe5, 0x39, 0x60, 0x15, 0x01, 0x58, 0x2b, 0xc0, 0x92, 0xcd, 0x41, 0x75,
0x1b, 0x33, 0x49, 0x78, 0x20, 0x51, 0x19, 0x3b, 0x26, 0xaf, 0x98, 0xa5, 0x4d,
0x14, 0xe7, 0x2f, 0x95, 0x36, 0xd4, 0x0a, 0x3b, 0xcf, 0x95, 0x25, 0xbb, 0x23,
0x43, 0x8f, 0x99, 0xed, 0xb8, 0x35, 0xe4, 0x86, 0x52, 0x95, 0x3a, 0xf5, 0x36,
0xba, 0x48, 0x3c, 0x35, 0x93, 0xac, 0xa8, 0xb0, 0xba, 0xb7, 0x93, 0xf2, 0xfd,
0x7b, 0xfa, 0xa5, 0x72, 0x57, 0x45, 0xc8, 0x45, 0xe7, 0x96, 0x55, 0xf9, 0x56,
0x4f, 0x1a, 0xea, 0x8f, 0x55 };
        //A chave pública sempre começa com 0x01, 0x00, 0x01,... e o resto é
tudo zero
        byte[] public_key = new byte[128];
        public_key[0] = public_key[2] = 1;

        Key_RSA rsa_encrypt = new Key_RSA(module, public_key);
```

```
        Key_RSA rsa_decrypt = new Key_RSA(module, private_key);

        // O dado que queremos incriptar
        string original_string = "FEZ is so easy!";
        //Converte para array de bytes
        byte[] original_data = UTF8Encoding.UTF8.GetBytes(original_string);
        //Encrypt the data
        byte[] encrypted_bytes = rsa_encrypt.Encrypt(original_data, 0,
original_data.Length, null);
        //Descripte os dados
        byte[] decrypted_bytes = rsa_decrypt.Decrypt(encrypted_bytes, 0,
encrypted_bytes.Length, null);
        //Imprima o dado descriptado
        string decrypted_string = new
string(Encoding.UTF8.GetChars(decrypted_bytes));
        Debug.Print("Data Size= " + original_string.Length + " Data= " +
original_string);
        Debug.Print("Encrypted Data size= " + encrypted_bytes.Length + "
Decrypted Data= " + decrypted_string);
    }
}
```

O tamanho dos dados criptografados com a RSA não é o mesmo que os dados originais. Tenha isso em mente quando você planeja carregar ou armazenar dados criptografados. RSA requer muito mais poder de processamento que XTEA, que pode causar problemas em sistemas de pequeno porte. Eu sugiro que você inicie uma sessão RSA para trocar a chave XTEA e alguma informação de segurança e então depois continue com XTEA.

26. XML

26.1. Teoria de XML

eXtensible Markup Language (XML) é uma linguagem padrão para uma organização de dados de computador. Quando você quiser transferir dados entre dois dispositivos, você pode definir regras como os dados podem ser empacotados enviados a partir do dispositivo A. Por outro lado, o dispositivo B recebe e sabe como eles estão organizados. Antes de XML, tinham-se alguns problemas. O que acontece quando você quer enviar dados para um sistema desenvolvido por outra pessoa? Você precisa explicar como você organizou seus registros para que ele possa lê-los corretamente. Agora os desenvolvedores podem usar XML para uma organização comum de dados.

XML é usado diariamente de muitas formas. Por exemplo, se um proprietário de um site de vendas quer saber o custo do transporte de um pacote, ele formata os detalhes de custo em XML e o envia para a FedEx (por exemplo). O site da FedEx irá interpretar o conteúdo e enviar informações sobre a expedição, sempre em formato XML.

A eficácia do XML também pode ser útil em outros casos. Suponha que você tenha programado um conjunto de dados de log. Imagine também que o usuário final pode configurar o data logger para cobrir suas necessidades, então ele precisa salvar a informação em algum lugar. Você pode usar o seu próprio formato, mas exigiria mais código e depuração. A melhor maneira é usar o XML, que é um padrão.Todos os cartões NETMF GHI Electronics ter suporte para leitura / escrita de XML.

Aqui está um exemplo de arquivo XML que será útil para o nosso data logger.

```
<?xml version="1.0" encoding="utf-8" ?>
<NETMF_DataLogger>
  <FileName>Data</FileName>
  <FileExt>txt</FileExt>
  <SampleFreq>10</SampleFreq>
</NETMF_DataLogger>
```

Este exemplo mostra um elemento raíz e 3 filhos. Eu escolhi essa estrutura, mas você poderia muito bem ter colocado apenas elementos pai. XML é muito flexível, às vezes flexível até demais. Voltamos ao nosso exemplo. O elemento raiz "NETMF_DataLogger" contém três elementos de informação que são importantes para o nosso logger. Contém o nome do arquivo, sua extensão e freqüência de backup. Neste exemplo o logger irá criar um arquivo data.txt e preenchê-lo 10 vezes a cada segundo.

Outro uso importante para nós "Desenvolvedores de Aplicações Embarcadas" é a partilha de informação com o sistema maior, como o PC. Como todos os sistemas operacionais de PCs suportam XML, você pode enoyer / receber dados do PC usando XML.

Espaços e layout não são significativas em XML. Estas coisas só são úteis aos seres humanos, por razões de legibilidade. O mesmo exemplo acima poderia muito bem ter sido escritos desta forma:

```
<?xml version="1.0" encoding="utf-8" ?><NETMF_DataLogger>
<FileName>Data</FileName>
<FileExt>txt</FileExt><SampleFreq>10</SampleFreq></NETMF_DataLogger>
```

Você entende porque a apresentação é importante para os seres humanos? Você também pode adicionar comentários em um arquivo XML. Os comentários não representam nada para o XML, apenas facilita a leitura do arquivo por um ser humano.

```
<?xml version="1.0" encoding="utf-8" ?>
<!--Isto é apenas um comentário-->
<NETMF_DataLogger>
  <FileName>Data</FileName>
  <FileExt>txt</FileExt>
  <SampleFreq>10</SampleFreq>
</NETMF_DataLogger>
```

Finalmente, XML suporta atributo. Um atributo é uma informação adicional de um item. Mas por que usar um atributo se você pode adicionar um novo elemento que descreve esta informação extra ? Você não precisa usar estes atributos e eu diria que se você não tiver uma boa razão para usá-los então não use. Eu não estarei explanando sobre atributos neste livro.

26.2. Criando um arquivo XML

Os dispositivos NETMF da GHI Electronics permitem ler e escrever XML. Isto é feito através de "streams", o que significa que qualquer stream que você tenha ou implementou, pode trabalhar com XML. Vamos utilizar os “streams” FileStream e MemoryStream, mas você poderia usar o seu próprio stream, o que não é descrita neste livro.

Este código mostra como criar um documento XML na memória. Esta código representa o exemplo XML recente.


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

public class Program
{
    public static void Main()
    {
        MemoryStream ms = new MemoryStream();

        XmlWriter xmlwrite = XmlWriter.Create(ms);

        xmlwrite.WriteProcessingInstruction("xml", "version=\"1.0\"
encoding=\"utf-8\"");
        xmlwrite.WriteComment("This is just a comment");
        xmlwrite.WriteStartElement("NETMF_DataLogger");//root element
        xmlwrite.WriteStartElement("FileName");//child element
        xmlwrite.WriteString("Data");
        xmlwrite.WriteEndElement();
        xmlwrite.WriteStartElement("FileExt");
        xmlwrite.WriteString("txt");
        xmlwrite.WriteEndElement();
        xmlwrite.WriteStartElement("SampleFeq");
        xmlwrite.WriteString("10");
        xmlwrite.WriteEndElement();
        xmlwrite.WriteEndElement();//end the root element

        xmlwrite.Flush();
        xmlwrite.Close();
        //////// mostra o dado XML ///////////
        byte[] byteArray = ms.ToArray();
        char[] cc = System.Text.UTF8Encoding.UTF8.GetChars(byteArray);
        string str = new string(cc);
        Debug.Print(str);
    }
}
```

Nota importante: com NETMF, bibliotecas de leitura e escrita XML são distintas. O leitura de XML provém do assembly "System.Xml" e a escrita provém do "MFDpwsExtensions"! Para sua comodidade, inclua os assembly quando você for trabalhar com XML.

Nota: Quando você adicionar o assembly, você verá que existem duas assemblies para XML, "System.Xml" e "System.Xml.Legacy". Nunca utilize o driver "legacy", pois é muito lento e consome muita memória. Ele existe apenas para sistemas que não conseguem suportar o padrão XML. Todos os dispositivos NETMF da GHI Electronics tem suporte XML nativo (muito rápido), então você pode sempre utilizar "System.Xml".

Ao executar o programa acima, você verá os dados XML na saída aparecem no final. Estes dados estão corretos, mas não necessariamente legível. Note que nós estamos

escrevendo/lendo arquivos XML em um sistema muito pequeno e, portanto, quanto menos informação (espaços, formato) melhor. Então é melhor não ter a formatação ou espaços, mas, tornam as coisas ainda mais divertidas quando nós adicionamos algumas linhas para melhorar a exibição

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

public class Program
{
    public static void Main()
    {
        MemoryStream ms = new MemoryStream();

        XmlWriter xmlwrite = XmlWriter.Create(ms);

        xmlwrite.WriteProcessingInstruction("xml", "version=\"1.0\"
encoding=\"utf-8\"");
        xmlwrite.WriteComment("This is just a comment");
        xmlwrite.WriteRaw("\r\n");
        xmlwrite.WriteStartElement("NETMF_DataLogger");//elemento raiz
        xmlwrite.WriteString("\r\n\t");
        xmlwrite.WriteStartElement("FileName");//elemento filho
        xmlwrite.WriteString("Data");
        xmlwrite.WriteEndElement();
        xmlwrite.WriteRaw("\r\n\t");
        xmlwrite.WriteStartElement("FileExt");
        xmlwrite.WriteString("txt");
        xmlwrite.WriteEndElement();
        xmlwrite.WriteRaw("\r\n\t");
        xmlwrite.WriteStartElement("SampleFeq");
        xmlwrite.WriteString("10");
        xmlwrite.WriteEndElement();
        xmlwrite.WriteRaw("\r\n");
        xmlwrite.WriteEndElement();//fim do elemento raiz

        xmlwrite.Flush();
        xmlwrite.Close();
        //////// mostra o dado XML ///////////
        byte[] byteArray = ms.ToArray();
        char[] cc = System.Text.UTF8Encoding.UTF8.GetChars(byteArray);
        string str = new string(cc);
        Debug.Print(str);
    }
}
```

26.3. Lendo XML

Criar um arquivo XML que é mais fácil do que ler (analisar). Existem várias maneiras de ler um arquivo XML, mas você pode simplesmente ler em pedaços até ao final do arquivo. Este exemplo cria um dado XML e o lê de volta.

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

public class Program
{
    public static void Main()
    {
        MemoryStream ms = new MemoryStream();

        XmlWriter xmlwrite = XmlWriter.Create(ms);

        xmlwrite.WriteProcessingInstruction("xml", "version=\"1.0\"
encoding=\"utf-8\"");
        xmlwrite.WriteComment("This is just a comment");
        xmlwrite.WriteRaw("\r\n");
        xmlwrite.WriteStartElement("NETMF_DataLogger");//elemento raiz
        xmlwrite.WriteString("\r\n\t");
        xmlwrite.WriteStartElement("FileName");//elemento filho
        xmlwrite.WriteString("Data");
        xmlwrite.WriteEndElement();
        xmlwrite.WriteRaw("\r\n\t");
        xmlwrite.WriteStartElement("FileExt");
        xmlwrite.WriteString("txt");
        xmlwrite.WriteEndElement();
        xmlwrite.WriteRaw("\r\n\t");
        xmlwrite.WriteStartElement("SampleFeq");
        xmlwrite.WriteString("10");
        xmlwrite.WriteEndElement();
        xmlwrite.WriteRaw("\r\n");
        xmlwrite.WriteEndElement();//fim do elemento raiz

        xmlwrite.Flush();
        xmlwrite.Close();
        //////// mostra o dado  XML  ///////////
        byte[] byteArray = ms.ToArray();
        char[] cc = System.Text.UTF8Encoding.UTF8.GetChars(byteArray);
        string str = new string(cc);
        Debug.Print(str);

        ///////////Leia XML
```

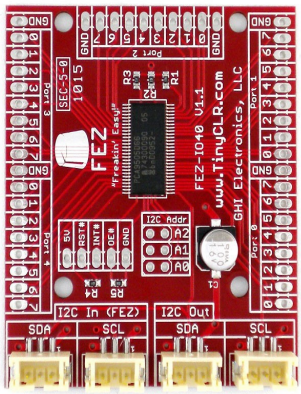
```
        MemoryStream rms = new MemoryStream(byteArray);

        XmlReaderSettings ss = new XmlReaderSettings();
        ss.IgnoreWhitespace = true;
        ss.IgnoreComments = false;
        //XmlException.XmlExceptionErrorCode.
        XmlReader xmlr = XmlReader.Create(rms,ss);
        while (!xmlr.EOF)
        {
            xmlr.Read();
            switch (xmlr.NodeType)
            {
                case XmlNodeType.Element:
                    Debug.Print("element: " + xmlr.Name);
                    break;
                case XmlNodeType.Text:
                    Debug.Print("text: " + xmlr.Value);
                    break;
                case XmlNodeType.XmlDeclaration:
                    Debug.Print("decl: " + xmlr.Name + ", " + xmlr.Value);
                    break;
                case XmlNodeType.Comment:
                    Debug.Print("comment " +xmlr.Value);
                    break;
                case XmlNodeType.EndElement:
                    Debug.Print("end element");
                    break;
                case XmlNodeType.Whitespace:
                    Debug.Print("white space");
                    break;
                case XmlNodeType.None:
                    Debug.Print("none");
                    break;
                default:
                    Debug.Print(xmlr.NodeType.ToString());
                    break;
            }
        }
    }
}
```

27. Expandindo as E/S

Um aplicativo pode exigir mais pinos digitais ou mais entradas analógicas do que o processado tenha disponível. Existem várias maneiras de aumentar este número.

27.1. Digital

A maneira mais fácil de aumentar o número de pinos digitais é utilizando um registrador de deslocamento digital. Este registo irá se conectar ao barramento SPI. Via SPI, que pode enviar/receber status dos pinos. Os registradores de deslocamento podem ser conectados em cascata, que em teoria permite um número ilimitado de pinos digitais Os registradores de deslocamento geralmente tem 8 pinos digital. Se ligarmos 3 no barramento SPI, teremos 24 novos pinos digitais disponíveis e nós só vamos utilizar os pinos SPI do processador.



Outro bom exemplo é a placa de IO40 que roda em um bus I2C. Também, está placa pode ser cascateada para um máximo de 320 I/Os.

Matriz de botões

Aparelhos como fornos de microondas têm frequentemente vários botões na frente. Normalmente, nós nunca pressionamos duas teclas ao mesmo tempo, por isso, pode agrupá-los em uma matriz. Se temos 12 botões em nosso sistema, seria necessário 12 entradas digitais no processador para ler todos eles. Mas ligando estes botões com uma matriz 4x3, temos apenas 7 pinos, em vez de 12. Matrizes de muitos botões existem e podem ser integradas em seu produto desta forma.

Para ligar os botões em uma matriz, deve ser feito o layout de forma a ter linhas e colunas. Cada botão conecta a uma linha e uma coluna. Isso é tudo para a parte de hardware! Note que, se nós não usamos uma matriz, o botão é conectado entre o pino e terra.

Para ler as teclas, faça os pinos do processador se conectar as todas as colunas e todas as linhas da matriz. Coloque uma linha (e única) para o estado alto e baixo para os demais. Nós estamos agora selecionando a linha de botões que nós vamos ler. Agora, lendo as colunas, você terá o estado dos botões .

Quando terminar, coloque esta linha no estado baixo, vá para a próxima colocando em estado alto e começe a ler as colunas. Faça isso para cada linha.

Este exemplo assume as seguintes conexões: 3 linhas conectadas aos pinos (1,2,3) e três

colunas conectado aos pinos (4,5,6)

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

namespace MFConsoleApplication1
{
    public class Program
    {
        static OutputPort[] Rows = new OutputPort[3];
        static InputPort[] Colms = new InputPort[3];

        static bool ReadMatrix(int row, int column)
        {
            bool col_state;

            //selecione uma linha
            Rows[row].Write(true);
            //selecione uma coluna
            col_state = Colms[column].Read();
            // deselecione uma linha
            Rows[row].Write(false);

            return col_state;
        }

        static void Main()
        {
            // inicialize linha como saída e jogue valor baixo
            Rows[0] = new OutputPort((Cpu.Pin)1, false);
            Rows[1] = new OutputPort((Cpu.Pin)2, false);
            Rows[2] = new OutputPort((Cpu.Pin)3, false);

            //Inicialize entradas com PULL DOWN
            Colms[0] = new InputPort((Cpu.Pin)4, true,
Port.ResistorMode.PullDown);
            Colms[1] = new InputPort((Cpu.Pin)5, true,
Port.ResistorMode.PullDown);
            Colms[2] = new InputPort((Cpu.Pin)6, true,
Port.ResistorMode.PullDown);

            while (true)
            {
                bool state;

                // Leia o botão na primeira linha e primeira coluna
                state = ReadMatrix(0, 0);//nós contamos do zero
                Debug.Print("Buton state is: " + state.ToString());
```

```
                //Leia o botão da terceira linha e segunda coluna column
                state = ReadMatrix(2, 1);//nós contaomos do zero
                Debug.Print("Buton state is: " + state.ToString());

                Thread.Sleep(100);
            }
        }
    }
}
```

27.2. Analógico

Existem dezenas (centenas) de chipsets analógicos disponíveis para o barramento SPI, I2C ou 1-Wire. Alguns lêem a partir de 0 a 5V e outros-10V a +10 V. Na verdade, existem muitas maneiras de adicionar entradas analógicas para o circuito! Alguns chips têm funções específicos. Se precisarmos de uma medida de temperatura, podemos conectar um sensor de temperatura para um pino analógico e ler o valor que será convertido em temperatura. É uma possibilidade, mas a melhor solução é usar um sensor de temperatura digital que operam em I2C, 1-Wire ou SPI. Isto lhe dará uma precisão melhor medição e vai salvar o pino analógico para outros usos.

Botões Analógicos

Uma dica para ligar um monte de botões para um único pino é usar uma entrada analógica. Cada botão está ligado a uma resistência diferente e, assim, a tensão na entrada do pino analógico será diferente dependendo do botão pressionado.

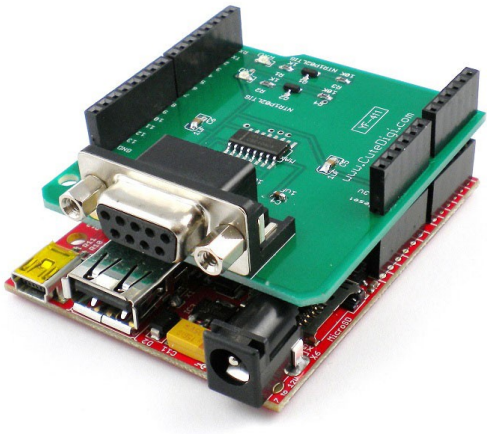
28. Cliente USB

Eu quero começar esta seção, salientando que este é um tema pouco avançada. Salve para ler mais tarde.

28.1. Serial (COM) Debugging

Por padrão, todos os dispositivos NETMF da GHI usam USB para a implantação (deploy) e depuração. Opcionalmente, um desenvolvedor pode querer utilizar o USB cliente (não o Host) para alguma coisa diferente do que a depuração. Isto é atualmente suportado pelo NETMF e GHI acrescentou recursos ficando muito fácil de configurar.

Nota importante: Você não pode usar a porta USB do cliente para ambas depuração e outra coisa. Uma vez que você decidir que quer usar a porta USB do cliente para seu próprio uso, então você não pode usar a porta USB para depuração. A propósito, GHI adcionou suporte a depuração e personalizar da interface USB simultaneamente através da utilização de várias interfaces na mesma interface USB, mas isso causou uma enorme carga de processo de suporte e assim a utilização simultânea foi descartada.



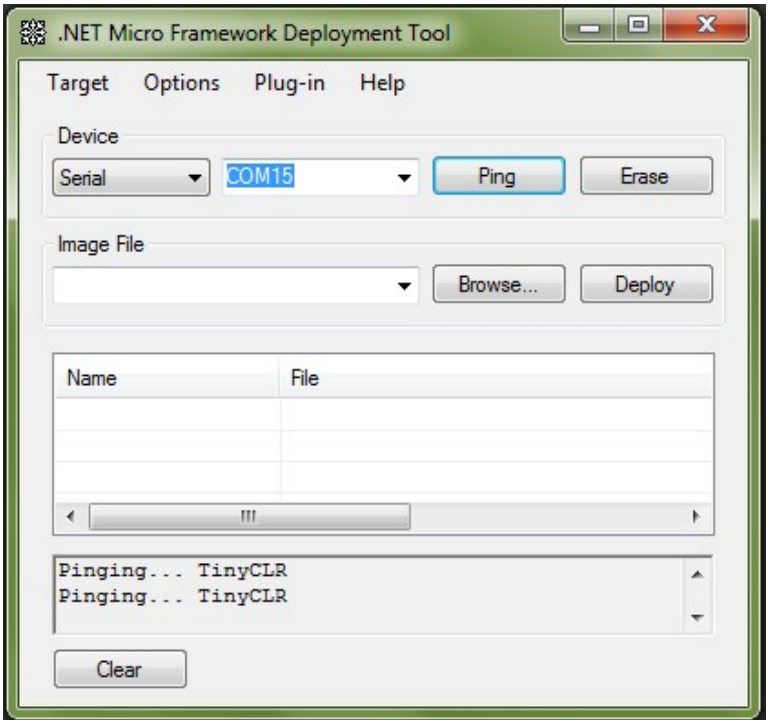
Deixe-me dar um exemplo sobre o uso do cliente USB. Digamos que você está fazendo um dispositivo que lê a temperatura e a umidade, etc ... e registra todos os dados em um cartão SD. Além disso, este dispositivo pode ser configurado, como configurar o tempo ou dar nomes de arquivo etc ... Você deseja que o dispositivo a ser configurado através de USB. Então, quando é ligado o dispositivo a uma porta USB, você quer mostrar como uma porta serial virtual. Desta forma, qualquer um pode abrir um software de terminal (como TeraTerm) para conectar o dispositivo para configurá-lo. Este é o lugar onde cliente USB torna-se muito útil. Não há necessidade de acrescentar custos adicionais ao projeto, mas acrescentando mais portas seriais RS232 ou USB <-> chipsets série. A Porta USB do cliente pode ser configurado para agir como um dispositivo do CDC, a porta COM virtual. Mas, existe uma batalha! Você ainda precisa conectar o PC para o dispositivo serial de depuração e implantação de aplicativos já que a porta cliente USB é usado por sua aplicação final. A boa notícia é que você só precisa da interface serial em fase de desenvolvimento, mas, quando você implanta no produto, a porta serial não é mais necessária. Por exemplo, você pode usar o shield RS232 FEZ Domino em fase de desenvolvimento, mas então você não vai mais necessitar dele quando terminar de depuração.

Agora, você tem COM1 contectado ao seu dispositivo e deseja usá-la para a depuração em vez de USB. O próximo passo é configurar o dispositivo para uso serial de depuração. Este é um dispositivo específico que você precisa verificar no manual do usuário do dispostiivo. Por exemplo, se você estiver usando FEZ Domino, há jumper rotulado MODE que você pode colocar para selecionar a depuração serial. Se você estiver usando FEZ Cobra, então o pino LMODE precisa estar conectado para 3.3V para seleccionar a depuração serial. Lembre-se que uma vez que você ligar o pino em alto ou baixo, então você não pode mais usá-lo em sua aplicação. Por exemplo, o pino MODE no FEZ Domino é o mesmo utilizado para o LED e é também um pino PWM. Uma vez que você curto circuitou o pino terra (através do jumper), você nunca deve tentar usar esse pino, nunca tente usar o LED ou poderá danificar o seu aparelho!

28.2. A configuração (SETUP)

Ok, chega de conversa e vamos iniciar a configuração. Estou usando o shield RS232 FEZ Domino, mas você pode usar qualquer dispositivo de sua escolha. Eu tenho o shield RS232 conectado ao FEZ Domino e também ligado ao meu PC usando um cabo RS232<->USB (sem porta serial no meu PC). Eu também fiz coloquei o jumper no pino MODE e conectei o cabo USB do Domino ao PC. Depois de colocar o jumper MODE, conectando o cabo USB do FEZ Domino a um PC não irá carregar todos os drivers do Windows (você não vai ouvir o som de USB conectado).

Finalmente, queremos ter certeza que podemos ping o dispositivo usando a ferramenta MFDeploy. Nós fizemos isso antes de usar o USB e agora queremos utilizar a serial. Note que apesar de eu ter FEZ Domino + shield RS232 conectado à porta USB do meu PC através do cabo RS232 <-> USBcabo USB, este é a COM e não USB, é uma COM virtual para ser exato.



Então, abra o MFDeploy e selecione COM e então você terá uma lista das portas seriais disponíveis. Selecione o seu dispositivo conectado e clique em Ping. Você deverá ver "TinyCLR ". Se não, então volte e verifique sua configuração.

28.3. Mouse, o plano perfeito

Aqui está o plano de mestre! Você quer fazer uma brincadeira com, gravar um vídeo e enviar ? Aqui está como você pode fazer isto. Configurar seu FEZ para emular um mouse USB, em seguida, fazer o movimento do mouse em um círculo cada poucos minutos. Aposto que ele vai se sentir como se houvesse um fantasma na sua máquina. A boa notícia é que o Windows pode ter vários dispositivos de mouse, logo o FEZ será escondido atrás do PC.

```
using System;
using System.Threading;
using Microsoft.SPOT;
using Microsoft.SPOT.Hardware;
using GHIElectronics.NETMF.USBClient;
using GHIElectronics.NETMF.Hardware;
using GHIElectronics.NETMF.FEZ;

public class Program
{
    public static void Main()
    {
        FEZ_Components.LED led = new FEZ_Components.LED(FEZ_Pin.Digital.Di3);
        led.StartBlinking(100, 200);


        // Checa interface de depuração
        if (Configuration.DebugInterface.GetCurrent() ==
Configuration.DebugInterface.Port.USB1)
            throw new InvalidOperationException("Current debug interface is USB.
It must be changed to something else before proceeding. Refer to your platform
user manual to change the debug interface.");

        // Inicia o Mouse
        USBC_Mouse mouse = USBClientController.StandardDevices.StartMouse();

        // Move ponteiro
        const int ANGLE_STEP_SIZE = 15;
        const int MIN_CIRCLE_DIAMETER = 50;
        const int MAX_CIRCLE_DIAMETER = 200;
        const int CIRCLE_DIAMETER_STEP_SIZE = 1;

        int diameter = MIN_CIRCLE_DIAMETER;
        int diameterIncrease = CIRCLE_DIAMETER_STEP_SIZE;
        int angle = 0;
        int factor;
        Random rnd = new Random();
        int i = 0;

        while (true)
        {
            // Nos queremos fazer isto randomicamente
            i = rnd.Next(5000) + 5000;//entre 5 e 10 segundos
            Debug.Print("Delaying for " + i + " ms");
            Thread.Sleep(i);
```

```
            i = rnd.Next(200) + 100;//faz isto por um curto espaço de tempo
            Debug.Print("Looping " + i + " times!");
            while (i-- > 0)
            {
                // Checa se conectado ao PC
                if (USBClientController.GetState() ==
USBClientController.State.Running)
                {
                    // Nota X, Y são reportados como deslocamentos, não valor
absoluto
                    factor = diameter * ANGLE_STEP_SIZE * (int)System.Math.PI /
180 / 2;
                    int dx = (-1 * factor * (int)Microsoft.SPOT.Math.Sin(angle) /
1000);
                    int dy = (factor * (int)Microsoft.SPOT.Math.Cos(angle) /
1000);

                    angle += ANGLE_STEP_SIZE;
                    diameter += diameterIncrease;

                    if (diameter >= MAX_CIRCLE_DIAMETER ||
                        diameter <= MIN_CIRCLE_DIAMETER
                        )
                        diameterIncrease *= -1;

                    // reporta a posição do mouse
                    mouse.SendData(dx, dy, 0, USBC_Mouse.Buttons.BUTTON_NONE);
                }

                Thread.Sleep(10);
            }
        }
    }
}
```

28.4. Teclado

Emulando um teclado é tão fácil como emular um mouse. O exemplo a seguir criará um teclado USB e enviar "Olá mundo!" para um PC a cada segundo.

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

using GHIElectronics.NETMF.USBClient;
using GHIElectronics.NETMF.Hardware;

public class Program
```

```
{
    public static void Main()
    {
        // Checa interface de debug
        if (Configuration.DebugInterface.GetCurrent() ==
Configuration.DebugInterface.Port.USB1)
            throw new InvalidOperationException("Current debug interface is USB.
It must be changed to something else before proceeding. Refer to your platform
user manual to change the debug interface.");
        // Start keyboard
        USBC_Keyboard kb = USBClientController.StandardDevices.StartKeyboard();
        Debug.Print("Waiting to connect to PC...");
        // Envie "Hello world!" a cada segundo
        while (true)
        {
            // Checa se está conectado ao PC
            if (USBClientController.GetState() ==
USBClientController.State.Running)
            {
                // Temos que pressionar SHIFT para sair o “H” maiúsculo
                kb.KeyDown(USBC_Key.LeftShift);
                kb.KeyTap(USBC_Key.H);
                kb.KeyUp(USBC_Key.LeftShift);
                // Agora "Hello world"
                kb.KeyTap(USBC_Key.E);
                kb.KeyTap(USBC_Key.L);
                kb.KeyTap(USBC_Key.L);
                kb.KeyTap(USBC_Key.O);
                kb.KeyTap(USBC_Key.Space);
                kb.KeyTap(USBC_Key.W);
                kb.KeyTap(USBC_Key.O);
                kb.KeyTap(USBC_Key.R);
                kb.KeyTap(USBC_Key.L);
                kb.KeyTap(USBC_Key.D);
                // Agora "!"
                kb.KeyDown(USBC_Key.LeftShift);
                kb.KeyTap(USBC_Key.D1);
                kb.KeyUp(USBC_Key.LeftShift);
                // Envie um enter (CR)
                kb.KeyTap(USBC_Key.Enter);
            }
            Thread.Sleep(1000);
        }
    }
}
```

28.5. CDC -Virtual Serial

As portas seriais são a interface mais comuns no mundo dos sistemas embarcados, É uma solução ideal para dispositivos para transferir dados entre PCs e dispositivos FEZ. Para combinar a popularidade e utilidade da USB com a facilidade da serial, temos dispositivos USB virtuais. Para aplicações para o Windows ou para dispositivos, uma porta serial virtual funciona como uma porta serial, mas na realidade, é realmente uma porta USB.

Uma coisa importante que eu quero mencionar aqui é que, normalmente, os drivers CDC manuseia uma transação em cada frame. O tamanho máximo do EP no USB é de 64 bytes e há 1000 frames/segundo em uma USB full-speed. Isto significa, a taxa de transferência máxima para os drivers CDC é de 64KB/sec. Eu acho que, a Microsoft percebeu a necessidade de CDC e maior taxa de transferência e realçou esta limitação. A última vez que testei a velocidade de transferência em minha máquina win7, eu era capaz de contastatar 500KB/sec.

O exempo a seguir irá criar um USB CDC e enviar "Olá mundo!" para o PC a cada segundo.

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

using GHIElectronics.NETMF.USBClient;
using GHIElectronics.NETMF.Hardware;

public class Program
{
    public static void Main()
    {
        // Checa a interface de debug
        if (Configuration.DebugInterface.GetCurrent() ==
Configuration.DebugInterface.Port.USB1)
            throw new InvalidOperationException("Current debug interface is USB.
It must be changed to something else before proceeding. Refer to your platform
user manual to change the debug interface.");

        // Incie CDC
        USBC_CDC cdc = USBClientController.StandardDevices.StartCDC();

        // Envie "Hello world!" para o PC a cada segundo (com cr/lf)
        byte[] bytes = System.Text.Encoding.UTF8.GetBytes("Hello world!\r\n");
        while (true)
        {
            // Checa se conectado ao PC
            if (USBClientController.GetState() !=
USBClientController.State.Running)
            {
                Debug.Print("Waiting to connect to PC...");
            }
```

```
            else
            {
                cdc.Write(bytes, 0, bytes.Length);
            }

            Thread.Sleep(1000);
        }
    }
}
```

28.6. Armazenamento em Massa (MSC)

Uma das grandes características exclusivas da GHI sobre cliente USB é o suporte Mass Storage Class (MSC). Este recurso permite o acesso a mídia conectada à USB. Deixe-me explicar isso através de alguns exemplos. A aplicação data logger deve salvar os dados em um cartão SD ou memória USB. Quando o usuário fez a coleta de dados, ele pode ligar o coletor de dados USB no PC e agora o PC pode detectar o dispositivo como um dispositivo de armazenamento em massa. O usuário pode então transferir os arquivos usando o sistema operacional padrão. Pense que o dispositivo é como um leitor de cartão de memória. Podemos até melhorar a nossa logger, onde o cliente USB pode ser CDC para configurar o dispositivo e depois mudar dinamicamente a MSC para transferir arquivos.

Uma pergunta muito comum no suporte da GHI é "Por que não consigo acessar a mídia, enquanto a mídia está sendo acessada externamente (no Windows)?". Vou tentar explicar melhor. A mídia é acessada através de uma classe chamada PersistentStorage. O objeto PersistentStorage pode ser acessada pelo sistema de arquivos interno ou externamente através da MSC USB. Ok, mas porque não os dois? Não pode ser ambos, porque os sistemas de arquivos levanta um monte de informações sobre a mídia para para acelerar o tempo de acesso ao arquivo. Acessando a mídia ao mesmo tempo pode causar dano (corromper) a mídia, por isso o acesso simultâneo não é permitido. Observe que você pode facilmente alternar entre o sistema de arquivos interno e MSC USB.

O exemplo assume que o cartão SD está sempre plugado. Ele habilita MSC exibindo o dispositivo (eu estou usando FEZ Domino) como um cartão de leitura.

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

using GHIElectronics.NETMF.USBClient;
using GHIElectronics.NETMF.IO;
using GHIElectronics.NETMF.Hardware;

namespace USBClient_Example
{
    public class Program
```

```
    {
        public static void Main()
        {
            // Checa interface de debug
            if (Configuration.DebugInterface.GetCurrent() ==
Configuration.DebugInterface.Port.USB1)
                throw new InvalidOperationException("Current debug interface is
USB. It must be changed to something else before proceeding. Refer to your
platform user manual to change the debug interface.");

            // Inicia MS
            USBC_MassStorage ms =
USBClientController.StandardDevices.StartMassStorage();

            // Assume que o cartão SD está conectado
            PersistentStorage sd;
            try
            {
                sd = new PersistentStorage("SD");
            }
            catch
            {
                throw new Exception("SD card not detected");
            }
            ms.AttachLun(0, sd, " ", " ");


            // Habilita acesso ao host
            ms.EnableLun(0);

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

28.7. Dispositivo customizados

O suporte de cliente USB da GHI permite-lhe controlar o cliente USB de qualquer maneira que você gosta. Esta funcionalidade exige um bom conhecimento de USB. Se você não sabe o que é terminal e Pipe então não tente criar dispositivos personalizados. Além disso, é muito importante que você tenha dispositivo configurado corretamente na primeira vez ele está conectado no Windows. Isto é importante porque o Windows armazena um monte de informações no seu registro. Então, se você alterar a configuração do seu aparelho depois que você tinha se conectado no Windows pela primeira vez o Windows pode não pode ver as mudanças, uma vez que estará usando a configuração antiga do seu registo.

Basicamente, ficar longe de USB Client Custom Devices, a menos que você realmente tem uma boa razão para usá-los e você tem um bom conhecimento de USB e os drivers do Windows.

Eu não vou explicar Custom Devices posteriormente, mas só queria deixar claro porque eu

decidi não fazê-lo. O padrão construído em classes, explicado anteriormente, deve ser suficiente para cobrir a maior parte de suas necessidades.

29. Baixo Consumo

Dispositivos alimentados por pilhas deveriam limitar o uso de corrente elétrica, o máximo possível.
Os dispositivos podem reduzir o consumo de corrente de muitas maneiras:

1. Reduza o clock do processador
2. “Paralize” processador quando estive ocioso (mantenha os periféricos e interrupções ativos)
3. “Paralize” alguns periféricos
4. Hiberne o sistema

Um dispositivo NETMF pode opcionalmente suportar qualquer um destes métodos. Consulte o manual do usuário do seu dispositivo para saber mais sobre o que é suportado diretamente. Em geral, todos os dispositivos GHI NETMF “paralizam” o processador quando em estado ocioso. Por exemplo, isto reduzirá o consumo do FEZ Rhino para cerca de 45mA. Você realmente não precisa fazer nada de especial, logo que o sistema estiver ocioso, o processador é “paralizado” automaticamente enquanto as interrupções e periféricos ainda estão ativos.

Além disso, todos os periféricos (ADC, DAC, PWM, SPI, UART ...) estão desativados por padrão, para menor consumo de corrente. Eles são ativados automaticamente quando eles são usados.

Se tudo isso não é suficiente, você pode hibernar completamente o sistema. Basta lembrar que quando o sistema está hibernando, não está executando e periféricos não estão funcionais. Por exemplo, quando dados vindo para a UART não irá acordar o sistema. Você simplesmente vai perder os dados recebidos. Acordando da hibernação é uma característica do sistema depende, mas geralmente o pino específico (s) pode ser alternado para acordar o sistema.

Nota: GHI percebeu que o modo padrão NETMF de alimentação não é adequado para hibernar, assim um método específico da GHI é executado para lidar com a hibernação do sistema.

Nota Importante: Quando o dispositivo está no modo de hibernação, USB para de funcionar. Você não pode ir nó código ou acessar o dispositivo. Se o seu programa sempre coloca o dispositivo em Sleep, então você não será capaz de carregar um novo programa. Basicamente, bloqueia o dispositivo. Você precisa entrar no gerenciador de boot para apagar tudo para desbloquear o aparelho da hibernação!

Nota Importante: Quando você hibernar, o relógio do sistema pára (não RTC) por isso o tempo do NETMF será desativado. Você vai precisar de ler o tempo da RTC e definir o tempo do sistema NETMF depois de acordar da hibernação.

Acordando da hibernação só é possível em alguns eventos. Verifique o manual do usuário do dispositivo ou a documentação da biblioteca para obter mais detalhes. Uma maneira fácil de acordar da hibernação é utilizando o recurso de alarme (quando disponível). Por exemplo, FEZ Rhino inclui o clock de 32KHz necessário para o RTC / alarme, mas o FEZ Mini não. Veja este link para mais detalhes

http://www.tinyclr.com/compare

Neste exemplo, eu ajustei o RTC alguma hora aleatória válida e então eu fiz piscar um LED. A cada três segundos, o dispositivo define o alarme para despertar em 10 segundos e depois coloca o dispositivo em modo de hibernação. O dispositivo estará completamente morto (novamente, sem depuração pela USB) mas você deve ser capaz de ver isso no LED. Quando o dispositivo acorda, ele continua a piscar o LED. No final, o LED irá piscar por 3 segundos e depois pára por 10 segundos.

```
using System;
using System.Threading;
using Microsoft.SPOT;
using GHIElectronics.NETMF.Hardware;
using GHIElectronics.NETMF.Hardware.LowLevel;

using GHIElectronics.NETMF.FEZ;
public class Program
{
    public static void Main()
    {
        //Pisca Led
        FEZ_Components.LED led = new FEZ_Components.LED(FEZ_Pin.Digital.LED);
        led.StartBlinking(100, 100);
        //Seta uma hora válida randomica
        RealTimeClock.SetTime( new DateTime(2010, 1, 1, 1, 1, 1));
        while (true)
        {
            Thread.Sleep(3000);//pisca led por 3 segundos
            RealTimeClock.SetAlarm(RealTimeClock.GetTime().AddSeconds(10));
            Debug.Print("Going to sleep for 10 seonds!");
            // Dorme por 10 segundos
            Power.Hibernate(Power.WakeUpInterrupt.RTCAlarm);

            Debug.Print("Good Morning!");
        }
    }
}
```

Outra opção é acordar na porta de interrupção. Você tem que ter cuidado com isso porque qualquer interrupção em qualquer pino fará com que este acorde. Por exemplo, o módulo Wi-Fi no FEZ Cobra usa internamente um dos pinos de interrupção e de modo que este vai acordar o sistema. Você precisa desativar WiFi antes da hibernação. Este não é o único truque que você precisa estar ciente! Olhe para a código seguinte e tente. Não vai funcionar!


```
using System;
using System.Threading;
using Microsoft.SPOT;
using Microsoft.SPOT.Hardware;
using GHIElectronics.NETMF.Hardware;
using GHIElectronics.NETMF.Hardware.LowLevel;

using GHIElectronics.NETMF.FEZ;
public class Program
{
    public static void Main()
    {
        //Pisca led
        FEZ_Components.LED led = new FEZ_Components.LED(FEZ_Pin.Digital.LED);
        led.StartBlinking(100, 100);
        //Configura o pino de interrupção
        InterruptPort LDR = new InterruptPort((Cpu.Pin)0,false,
Port.ResistorMode.PullUp, Port.InterruptMode.InterruptEdgeLow);
        while (true)
        {
            Thread.Sleep(3000);//Pisca led por 3 segundos
            // dorme
            Power.Hibernate(Power.WakeUpInterrupt.InterruptInputs);
            //chegamos aqui quando sai da hibernação
        }
    }
}
```

Por que o exemplo acima não funciona? Quando você cria uma interrupção (ou entrada) do pino, as interrupções são habilitados somente se o filtro glitch é usado ou se um manipulador de eventos está instalado. Então, para fazer o exemplo acima funcionar, você só precisa ativar o filtro. Aqui está o código que funciona.

```
using System;
using System.Threading;
using Microsoft.SPOT;
using Microsoft.SPOT.Hardware;
using GHIElectronics.NETMF.Hardware;
using GHIElectronics.NETMF.Hardware.LowLevel;

using GHIElectronics.NETMF.FEZ;
public class Program
{
    public static void Main()
    {
        //pisca led
        FEZ_Components.LED led = new FEZ_Components.LED(FEZ_Pin.Digital.LED);
        led.StartBlinking(100, 100);
        //Seta o pino de interrupção com Glitch habilitado
```

```
        InterruptPort LDR = new InterruptPort((Cpu.Pin)0,true,
Port.ResistorMode.PullUp, Port.InterruptMode.InterruptEdgeLow);
        while (true)
        {
            Thread.Sleep(3000);//pisca led por 3 segundos
            // Durma
            Power.Hibernate(Power.WakeUpInterrupt.InterruptInputs);
            //hegamos aqui quando sai da hibernação
        }
    }
}
```

Outra opção é instalar um manipulador de eventos para o botão como segue:

```
using System;
using System.Threading;
using Microsoft.SPOT;
using Microsoft.SPOT.Hardware;
using GHIElectronics.NETMF.Hardware;
using GHIElectronics.NETMF.Hardware.LowLevel;

using GHIElectronics.NETMF.FEZ;
public class Program
{
    public static void Main()
    {
        //pisca led
        FEZ_Components.LED led = new FEZ_Components.LED(FEZ_Pin.Digital.LED);
        led.StartBlinking(100, 100);
        //configura o pino de interrupção
        InterruptPort LDR = new InterruptPort((Cpu.Pin)0,false,
Port.ResistorMode.PullUp, Port.InterruptMode.InterruptEdgeLow);
        LDR.OnInterrupt += new NativeEventHandler(LDR_OnInterrupt);
        while (true)
        {
            Thread.Sleep(3000);//pisca leds por 3 segundos
            // durma
            Power.Hibernate(Power.WakeUpInterrupt.InterruptInputs);
            //chegamos aqui quando sai da hibernação
        }
    }

    static void LDR_OnInterrupt(uint data1, uint data2, DateTime time)
    {
        // vazio no momento
    }
}
```

Por favor, note que o uso InputPort é tão bom quanto usar o InterruptPort uma vez que, internamente, as interrupções são utilizados quando o filtro de Glitech está habilitado. Aqui é

o exemplo, utilizando InputPort em vez de InterruptPort.

```
using System;
using System.Threading;
using Microsoft.SPOT;
using Microsoft.SPOT.Hardware;
using GHIElectronics.NETMF.Hardware;
using GHIElectronics.NETMF.Hardware.LowLevel;

using GHIElectronics.NETMF.FEZ;
public class Program
{
    public static void Main()
    {
        //pisca led
        FEZ_Components.LED led = new FEZ_Components.LED(FEZ_Pin.Digital.LED);
        led.StartBlinking(100, 100);
        //Configura pino de interrupçÃo
        InterruptPort LDR = new InterruptPort((Cpu.Pin)0,false,
Port.ResistorMode.PullUp, Port.InterruptMode.InterruptEdgeLow);
        LDR.OnInterrupt += new NativeEventHandler(LDR_OnInterrupt);
        while (true)
        {
            Thread.Sleep(3000);//pisca led por 3 segundos
            // durma
            Power.Hibernate(Power.WakeUpInterrupt.InterruptInputs);
            //chegamos aqui quando sai da hibernação
        }
    }

    static void LDR_OnInterrupt(uint data1, uint data2, DateTime time)
    {
        // vazio no momento
    }
}
```

Este exemplo irá piscar um LED, quando acordar ou quando o botão LDR botão for pressionado, o sistema vai entrar em sono profundo por 10 segundos.

Este é um exemplo para FEZ Domino (USBizi) que vai entrar em sono profundo e acordar quando eu pressionar um botão ??????????

```
using System;
using System.Threading;
using Microsoft.SPOT;
using Microsoft.SPOT.Hardware;
using GHIElectronics.NETMF.Hardware;
using GHIElectronics.NETMF.Hardware.LowLevel;

using GHIElectronics.NETMF.FEZ;
```

```
public class Program
{
    public static void Main()
    {
        //pisca led
        FEZ_Components.LED led = new FEZ_Components.LED(FEZ_Pin.Digital.LED);
        led.StartBlinking(100, 100);
        //configura pino de interrupção
        InputPort LDR = new  InputPort((Cpu.Pin)0,true,
Port.ResistorMode.PullUp);
        while (true)
        {
            Thread.Sleep(3000);//blink LED for 3 seconds
            // durma
            Power.Hibernate(Power.WakeUpInterrupt.InterruptInputs);
            //chegamos aqui quando sai da hibernação
        }
    }
}
```

Finalmente, você pode acordar em mais um eventos. Por exemplo, você pode acordar se um botão for pressionado ou o alarme é acionado.

```
Power.Hibernate(Power.WakeUpInterrupt.InterruptInputs |
Power.WakeUpInterrupt.RTCAlarm);
```


30. Watchdog

No mundo de sistemas embarcados, os dispositivos são geralmente sempre rodando e sem interação do usuário. Portanto, se algo der errado, seria muito benéfico se tivermos um botão de reset automático. Watchdog é o botão de reset!

Recuperação do Sistema de Execução

Suponha que você está fazendo uma máquina de venda automática inteligente que reporta os relatórios de inventário sobre a rede. Se o seu código gera uma exceção que não foi tratado adequadamente, em seguida, seu programa vai finalizar. Programa terminando significa que a máquina de vendas não funcionam mais. Alguém terá de conduzir até máquina de vendas e dar um reset, ou melhor, é só usar watchdog.

Quando você habilita o cão de guarda (watch dog), você configura o tempo limite para resetar. Em seguida, você manter a redefinição do watchdog timer periodicamente. Se o sistema trava, em seguida, watch dog chega ao limite, então irá resetar o sistema.

Importante: GHI percebeu que o watch dog interno não é o que os consumidores querem, então a GHI implementou a sua própria versão de watch dog. Não use o watch dog da Microsoft.SPOT.Hardware, em vez disso, use o cão de guarda GHIElectronics.NETMF.Hardware.LowLevel.

Se ambos os namespaces forem ser usados, então você terá erros de ambigüidade e por isso é necessário chamar especificamente o caminho completo do cão de guarda que você precisa. Por exemplo, em vez de usar o tempo limite (Watchdog.Enable), use GHIElectronics.NETMF.Hardware.LowLevel.Watchdog.Enable (timeout).

Este exemplo mostra como definir o timeout do watchdog para 5 segundos e criar uma thread para reinicializar o cão de guarda a cada 3 segundos. Se algo der errado, o dispositivo irá reiniciar em 5 segundos.

Nota Importante: Uma vez você ter habilitado o Watchdog, não pode ser mais desabilitado. Então you deve ficar resetando o timeout. Isto é feito para garantir que o corrompimento do sistema não desabilite o watchdog acidentalmente.

```
using System;
using System.Threading;
using GHIElectronics.NETMF.Hardware.LowLevel;

public class Program
{
    public static void Main()
    {
```

```
        // Timeout por 5 segundos
        uint timeout = 1000 * 5;

        // Habilita Watchdog
        Watchdog.Enable(timeout);

        // Da início ao contador do tempo de reset (thread)
        WDTCounterReset = new Thread(WDTCounterResetLoop);
        WDTCounterReset.Start();

        // ....
        // seu programa começa aqui

        // Se nós terminamos o programa, a thread vai parar de trabalhar e o
sistema irá resetar
        Thread.Sleep(Timeout.Infinite);
    }

    static Thread WDTCounterReset;
    static void WDTCounterResetLoop()
    {
        while (true)
        {
            // Conador o tempo de reseet a cada 3 segundos
            Thread.Sleep(3000);

            Watchdog.ResetCounter();
        }
    }
}
```

Está pensando talvez, se o software trava, como seria o código que manipula o watchdog ? Em baixo nível, o watchdog é suportada pelo hardware, não pelo software. Isso significa que o contador e o mecanismo de reset e é feito dentro do processador, sem necessidade de qualquer software.

Limitando tarefas críticas ao tempo

De volta ao nosso exemplo da máquina de venda automática, mas desta vez queremos lidar com um problema diferente. Como NETMF não é em tempo real, as tarefas podem levar mais tempo do que o esperado. Se uma pessoa se aproximou da máquina e selecionou o que eles querem comprar, a máquina vai ligar um motor que, por sua vez irá expulsar o item para o usuário. Digamos que isso foi programada de modo que o motor tem que estar ligado por um segundo. Agora, o que se sucedeu que, ao mesmo tempo o motor está funcionando outro segmento começou a usar o cartão SD. Vamos supor também que o cartão tinha algum problema que causou três segundo atraso na leitura do cartão SD. O atraso de 3 segundos, enquanto o motor estiver funcionando irá resultar em três itens que estão sendo empurrados

para o comprador, mas nós só queriamos um item.

Se utilizado o watch dog e configurá-lo para um segundo, então o usuário terá um item e quando o tempo é superior a um segundo, a máquina de vendas irá resetar, o qual vai parar o motor de mandar outros items para fora.

Aqui um simples exemplo:

```
// ....
// ....
// Timeout de segundo
uint timeout = 1000;

//....usuário compra alguma coisa

// Habilita Watch Dog
Watchdog.Enable(timeout);
//Liga moto
//...
//Desliga motor
//...
Watchdog.Disable();

// ....
// Se programa começa aqui
// ....
// ....
```

Detectando o porque da atuação do Watch Dog

Em alguns casos, você precisa saber se o sistema resetou por causa do watch dog para logar estas informação ou executar algum procedimento de recuperação. Aqui é como funciona:

```
using System;
using System.Threading;
using Microsoft.SPOT;
using GHIElectronics.NETMF.Hardware.LowLevel;


public class Program
{
    public static void Main()
    {
        // você pode ler este flag ***UMA única VEZ*** quando alimentar
        if (Watchdog.LastResetCause == Watchdog.ResetCause.WatchdogReset)
```

```
        {
            Debug.Print("Watchdog did Reset");
        }
        else
        {
            Debug.Print("Reset switch or system power");
        }
    }
}
```

31. Wireless (Conexão sem Fio)

As conexões sem fio estão se tornando muito importante em nossas vidas. Alguns aplicações requerem altas velocidades de transferência, outros de consumo muito baixo. Alguns oferecem ponto-a-ponto, enquanto outros exigem uma rede mesh.
O maior problema quando você cria um dispositivo sem fio é a certificação. Não só isso, deve ser feito de maneira diferente em diferentes países, mas também é muito caro. Isto pode custar cerca de US$ 50.000 para uma certificação. Felizmente, algumas empresas oferecem módulos certificados. Usando esses módulos, você não precisa de certificação.

31.1. Zigbee (802.15.4)

Zigbee foi concebido para uso em sistemas de baixa potência. Vários sensores podem ser conectados a uma rede Zigbee. Ele consome muito pouca corrente, no entando não é muito rápido.

Uma aplicação muito comum do Zigbee é são os módulos Xbee fornecida pela Digi. Existem vários tipos de módulos: de baixo consumo e alta potência de transmissão para cobrir grandes distâncias (20 km!). Estes módulos estão disponíveis com antenas internas ou externas.

A interface com esses módulos é uma UART simples. Com UART, pode ser conectado com qualquer cartão NETMF. Se você criar uma conexão entre dois módulos, é feito automaticamente. Se você deseja se conectar a múltiplos nodos, então você precisa enviar alguns comandos adicionais para configurar a rede. Eu recomendo que você comece com o conexão ponto a ponto.

Conectando-se a Xbee ao FEZ Mini ou FEZ Domino pode ser facílmente utilizando o componente de expansão Xbee

31.2. Bluetooth

Quase todos os telefones celulares têm como se conectar a dispositivos Bluetooth. A tecnologia Bluetooth define vários perfis para conexões. O perfil de áudio é útil para conectar fones de ouvido. Profile (SPP Serial Port Profile) é útil para estabelecer uma comunicação, simulando uma conexão serial. É realmente muito parecido com a conexão Xbee. Enquanto a maioria dos telefones têm Bluetooth, muitos não suportam o perfil SPP e, portanto, a conexão serial com o seu laptop é provavelmente impossível.

No Windows, você pode criar uma conexão Bluetooth em alguns cliques:

1.No painel de controle, procure "Dispositivos Bluetooth"

2.Clique em "Configurações do Bluetooth"

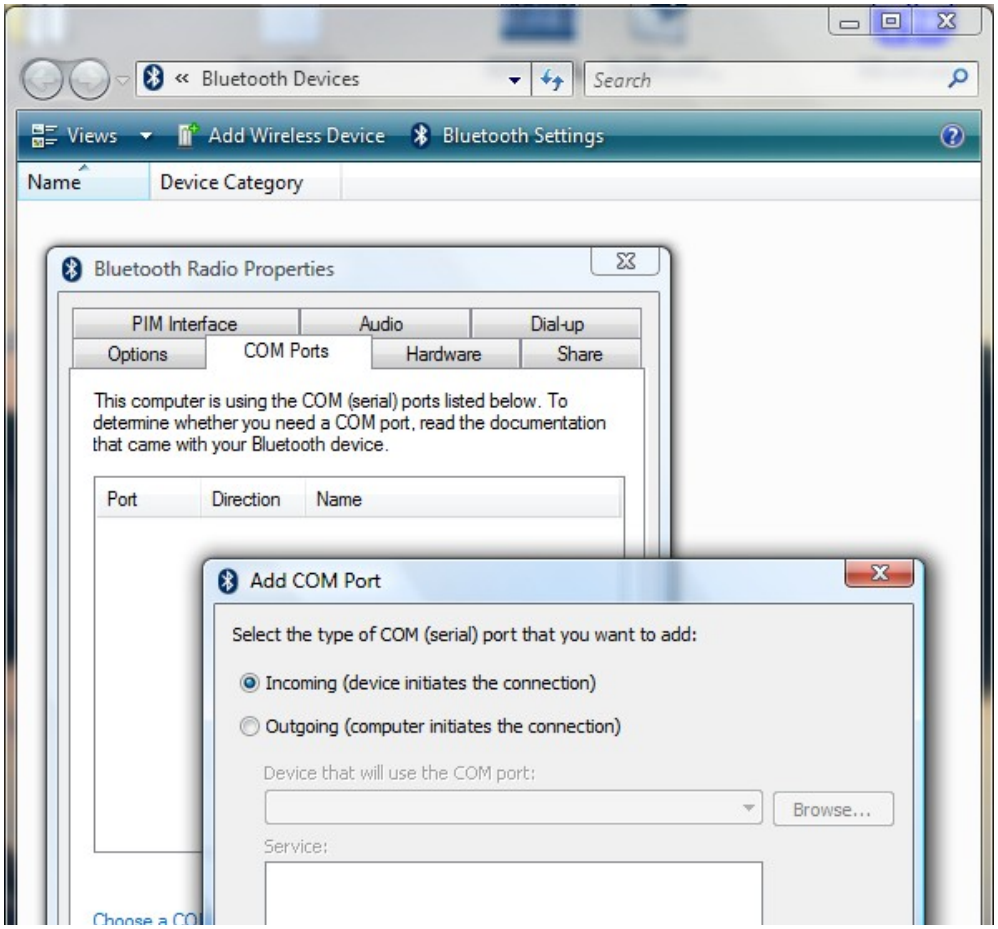
3 Veja o tab “COM ports”

4.Se você tem COMs bluetooth instaladas então você já tem SPP habilitado no seu PC

5.Para adicionar novas portas, clique em "Add …"

6.Crie uma porta pra recebimento de dados e outra para envio de dados

Windows cria duas portas, uma entrada e uma saída. Isso pode confundir o usuário, porque ele não pode usar a mesma porta para enviar ou receber dados.



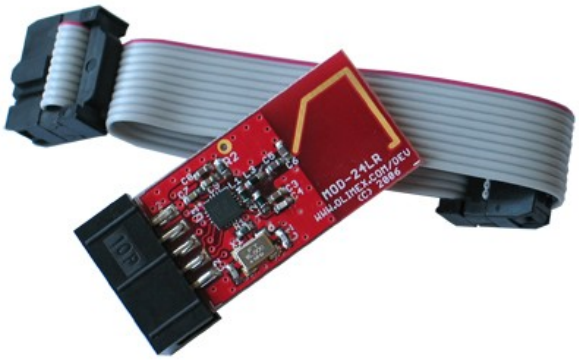
No lado de Hardware, existem muitos módulos seriais Bluetooth que já que incluem Hardware e software. Isto permite facilmente se conectar a uma porta serial do dispositivo NETMF.

A conexão do módulo Bluetooth na placa FEZ ou FEZ Dominio pode facilmente ser realizada utilizando o componente de interface Bluetooth.

31.3. Nordic

Nordic Semiconductor criou seus próprios chips digitais sem fio, nRF24L01. Estes chips de baixa potência, utilizam a banda de 2.4Ghz que é livre em muitos países. Os chips wireless da Nordic permitem conexão ponto a ponto ou multipontos.

Olimex oferece placas com o nRF24L01. Essas placas podem se conectar a maioria das placas NETMF da GHI electronics.



Este é um projeto (e vídeo) exibindo como conectar 2 dispositivos NETMF por meio do NRF24L01

http://www.microframeworkprojects.com/index.php?title=SimpleWireless

32. Objetos em uma pilha customizada

Esta seção aborda sobre alocações de memória grandes e não se aplica ao USBizi.

Sistemas gestores como o NETMF necessitam de um sistema complexo de administração de memória. Para reduzir a sobrecarga em sistemas pequenos, a pilha somente suporta alocação de objetos de até 700KBs. Não é possível usar objetos maiores. A partir do NETMF 4.0, no entanto, grandes buffers podem ser atribuídos usando outra pilha, separada, chamada "**Custom Heap**". Agora, você pode criar buffers ou imagens grandes. Internamente, esses objetos não estão na pilha padrão, mas no "Custom Heap". Isso levanta uma importante questão: quanto de memória é reservada para "Custom Heap" e quanto é reservado para a a pilha ?

GHI oferece uma API que permite que você ajuste o tamanho de cada pilha, "custom heap" e do sistema.

```
using System;
using Microsoft.SPOT.Hardware;
using GHIElectronics.NETMF.Hardware;
class Program
{

    public static void Main()
    {
        // Seta o tamanho da pilha para 4MB
        if (Configuration.Heap.SetCustomHeapSize(4 * 1024 * 1024))
        {
            // Isto terá efeito após resetar o sistema
            PowerState.RebootDevice(false);
        }
        // ...
        // você pode agora usar objetos acima de  4MB
    }
}
```

32.1. Administrando a "Custom Heap"

Ao contrário da pilha normal, que é totalmente gerenciado, a "Custom Heap" é gerenciada, mas há vários pontos a considerar com o uso dela. Para que objetos na pilha sejam apagados automaticamente, você deve preencher 3 requerimentos:

- A referência para o objeto deve ser perdida (isto é tudo que você necessita para objetos regulares)
- O coletor de lixo deve ser executado. Você provavelmente tem que forçá-lo

- O sistema deve estar em repouso que forma que finalize() a execução e “Dispose” o objeto

Não vou detalhar o coletor de lixo, “dispose” ou “finalize”. A única coisa que você precisa saber é que os objetos grandes que estão na “custom heap” não são facilmente apagáveis do sistema de modo que você sempre tem que “dispose” objeto quando tiver terminado de usá-lo.

Aqui está o comentário do “dispose” para objetos grandes.

```
using System;
using Microsoft.SPOT.Hardware;
using GHIElectronics.NETMF.Hardware;
class Program
{

    public static void Main()
    {
        // aloca um buffer de 1MB
        LargeBuffer lb = new LargeBuffer(1024 * 1024);
        // use o buffer
        lb.Bytes[5] = 123;
        // ....
        // Quando pronto, libere (dispose) o objeto para esvaziar a memória
        lb.Dispose();
    }
}
```

A melhor solução é usar a palavra-chave "using". Ela palavra-chave irá automaticamente chamar “dispose” uma vez a execução está deixando as chaves do “using”

```
using System;
using Microsoft.SPOT.Hardware;
using GHIElectronics.NETMF.Hardware;
class Program
{
    public static void Main()
    {
        // aloca um buffer de 1MB
        using (LargeBuffer lb = new LargeBuffer(1024 * 1024))
        {
            // use o buffer
            lb.Bytes[5] = 123;
            // ...
        }
        // Dispose foi chamado automaticamente
```

```
        // ...
    }
}
```

32.2. Grandes bitmaps

As imagens maiores que 750KB/2 são automaticamente alocados na pilha do usuário (custom heap). Por exemplo, uma imagem de 800x480 vai exigir 800x480x4 bytes, 1.5 MB Esta imagem vai estar na “custom heap” e você tem que “dispose" (eliminar) ela no final da sua utilização ou utilizar a palavra-chave “using".

```
using System;
using Microsoft.SPOT;
using Microsoft.SPOT.Presentation.Media;
class Program
{
    public static void Main()
    {
        using(Bitmap largeBitmap = new Bitmap(800,480))
        {
            // assuma que temos um display de 800x480
            // desenhe um circulo
            largeBitmap.DrawEllipse(Colors.Green, 100, 100, 10, 10);
            // desenhe outras coisas
            // ...
            largeBitmap.Flush();//flush o objeto  Bitmap para o display
        }
        // uma vez aqui, o Dispose para largeBitmap é chamado automaticamente
        // ...
    }
}
```

32.3. Buffers grandes

Nós já usamos LargeBuffer no exemplo mais recente. Só use se você realmente precisa alocar um buffer que seja maior que 750KB. Em sistemas embarcados, você pode não necessitar usar buffers grandes. Embora não seja recomendado, você pode usá-lo se necessário.

```
using System;
using Microsoft.SPOT.Hardware;
using GHIElectronics.NETMF.Hardware;
class Program
{
    public static void Main()
```

```
    {
        // aloca 1 Mb de memória
        using (LargeBuffer lb = new LargeBuffer(1024 * 1024))
        {
            // use o buffer
            lb.Bytes[5] = 123;
            byte b = lb.Bytes[5];
        }
        // Dispose foi chamado automaticamente
        // ...
    }
}
```

33. Pensando “pequeno”

Muitos desenvolvedores NETMF vem do mundo do PC. Eles estão acostumados a escrever um código que funciona bem em um PC, mas este mesmo código não será executado de forma ótima para software embarcado. Um PC pode rodar em 4GHz e tem 4GB de memória. cartões NETMF têm menos de 1% dos recursos disponíveis em um PC. Vou falar sobre diferentes áreas onde você tem que pensar "pequeno".

33.1. Utilização da memória

Com uma quantidade limitada de memória, você deve usar apenas o que você realmente precisa. Programadores PC tendem a criar buffers maiores para tratar até mesmo coisas pequenas. Desenvolvedores "embedded" devem estudar suas necessidades e atribuir apenas a memória necessária. Se eu ler dados de um UART, eu posso muito bem usar um buffer de 100 bytes para leitura e 1000 bytes de buffer tambem vai funcionar bem. Mas se eu analisar o meu código, eu percebi que na verdade eu só receberei 40 bytes no meu loop. Então, por que usar um buffer com de mais de 40 bytes?

Em alguns drivers, o NETMF faz uma série de bufferizações internamente. Por exemplo, o sistema de arquivos, UART ou drivers USB têm buffers internos, prontos para serem utilizados no código gerenciado do usuário. Se temos de gerir um arquivo de 1 MB, ele cria uma reserva menor e são enviados em várias partes. Para reproduzir um arquivo MP3 de 5MB, um buffer único byte 100 lerá pequenos pedaços do arquivo para o decodificador MP3.

33.2. Alocação de Objetos

Alocação e liberação de objetos é muito caro. Somente faça alocação de objetos quando você realmente necessita deles. Não se esqueça que você está programando um sistema embarcado e, portanto, os objetos que você esta utilizando sempre são utilizados. Por exemplo, você sempre irá sempre utilizar o LCD ou o SPI.

Tomemos por exemplo o código

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

namespace MFConsoleApplication1
{
    public class Program
    {
```

```
        static void WriteRegister(byte register_num, byte value)
        {
            SPI _spi = new SPI(new
SPI.Configuration(Cpu.Pin.GPIO_NONE,false,0,0,false,
                                                    true,1000,SPI.SPI_module.SPI
1));
            byte[] buffer = new byte[2];
            buffer[0] = register_num;
            buffer[1] = value;
            _spi.Write(buffer);
        }
        public static void Main()
        {
            WriteRegister(5, 100);
        }
    }
}
```

Para escrever um byte em um chip SPI, eu tinha alocado um objeto SPI tenho atribuído um objeto SPI.Configuration e um array de bytes. Três objetos para enviar um único byte! Isso é possível se você fazer algumas vezes no início do programa, mas se você chamar o método WriteRegister então isto não é um bom caminho. Para iniciantes, este método rodará muito lento então não poderá se utilizado continuamente para enviar dados MP3 para um decoder. E o pior, esta função será chamada centenas de vezes por segundo. Um segundo problema é que esses objetos são criados, usados e abandonados para o coletor de lixo remove-los. O coletor de lixo vai ter que disparar e remover todos os objetos não usados da memória o qual fará a CPU parar por alguns milisegundos.

Aqui é uma versão modificada do código para testar o método quando chamado 1000 vezes.

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

namespace MFConsoleApplication1
{
    public class Program
    {
        static void WriteRegister(byte register_num, byte value)
        {
            SPI _spi = new SPI(new
SPI.Configuration(Cpu.Pin.GPIO_NONE,false,0,0,false,
                                                    true,1000,SPI.SPI_module.SPI
1));
            byte[] buffer = new byte[2];
            buffer[0] = register_num;
```

```
            buffer[1] = value;
            _spi.Write(buffer);
            _spi.Dispose();
        }
        public static void Main()
        {
            long ms;
            long ticks = DateTime.Now.Ticks;
            for (int i = 0; i < 1000; i++)
                WriteRegister(5, 100);
            ticks = DateTime.Now.Ticks - ticks;
            ms = ticks / TimeSpan.TicksPerMillisecond;
            Debug.Print("Time = " + ms.ToString());
        }
    }
}
```

Ao executar este código no dispositivo FEZ (USBizi), notamos que o coletor de lixo foi chamado 10 vezes. Ele exibe a sua atividade na janela de saída. O tempo necessário para a execução completa do programa é de 1911ms, quase dois segundos! Agora modifique o código conforme abaixo. Nós criamos o objeto SPI globalmente, e estará sempre disponível. Nós ainda estamos alocando o buffer em cada loop.

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

namespace MFConsoleApplication1
{
    public class Program
    {
        static SPI _spi = new SPI(new SPI.Configuration(
            Cpu.Pin.GPIO_NONE, false, 0, 0, false,
            true, 1000, SPI.SPI_module.SPI1));

        static void WriteRegister(byte register_num, byte value)
        {
            byte[] buffer = new byte[2];
            buffer[0] = register_num;
            buffer[1] = value;
            _spi.Write(buffer);
            _spi.Dispose();
        }
        public static void Main()
        {
            long ms;
            long ticks = DateTime.Now.Ticks;
```

```
            for (int i = 0; i < 1000; i++)
                WriteRegister(5, 100);
            ticks = DateTime.Now.Ticks - ticks;
            ms = ticks / TimeSpan.TicksPerMillisecond;
            Debug.Print("Time = " + ms.ToString());
        }
    }
}
```

Neste exemplo, o coletor de lixo tem sido disparado apenas duas vezes e o código durou 448 milissegundos, menos de meio segundo. Nós apenas movemos uma única linha de código e agora é 4 vezes mais rápido!

Deixe nos mover o buffer globalmente e ver

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

namespace MFConsoleApplication1
{
    public class Program
    {
        static SPI _spi = new SPI(new SPI.Configuration(
            Cpu.Pin.GPIO_NONE, false, 0, 0, false,
            true, 1000, SPI.SPI_module.SPI1));
        static byte[] buffer = new byte[2];
        static void WriteRegister(byte register_num, byte value)
        {
            buffer[0] = register_num;
            buffer[1] = value;
            _spi.Write(buffer);
            _spi.Dispose();
        }
        public static void Main()
        {
            long ms;
            long ticks = DateTime.Now.Ticks;
            for (int i = 0; i < 1000; i++)
                WriteRegister(5, 100);
            ticks = DateTime.Now.Ticks - ticks;
            ms = ticks / TimeSpan.TicksPerMillisecond;
            Debug.Print("Time = " + ms.ToString());
        }
    }
}
```


Nós temos agora 368ms e o coletor de lixo nem sequer foi chamado!

Um teste rápido que você pode fazer no seu dispositivo é checar a saída para ver o quanto frequênte o coletor de lixo é disparado. Nos sistemas como muita memória como ChipworkX não vai ajudar muito, então você deve ainda verificar o seu código manualmente.

34. Tópicos faltantes

Alguns temas não foram abordados neste livro. Aqui está uma pequena lista com um resumo sobre a questão. Talvez eu adicione capítulos destes tópicos posteriormente.

34.1. WPF

Windows Presentation Foundation é uma nova maneira flexível para criar aplicações com interface gráfica. Embedded Master e ChipworkX pode usar WPF, mas não USBizi e FEZ.

34.2. DPWS

Device Profile for Web Services permite que os dispositivos sejam automaticamente detectados e utilizados em uma rede. DPWS requer soquetes. NET para operar e, portanto, pode ser usado com Embedded Master ou ChipworkX.

34.3. Extended Weak Reference

Extensão Weak Reference (EWR) permite que os desenvolvedores possar salvar alguns dados na memória não-volátil. EWR foi utilizado principalmente antes da introdução do sistema de arquivos em NETMF

34.4. Serialização

Serialização é uma maneira de converter um objeto para uma série de bytes que representa um objeto. Um objeto Mike criado como tipo humano pode ser serializado em um array de byte e então estes dados transferidos para outro dispositivo. O outro dispositivo sabe o que é um tipo humano mas não sabe nada sobre Mike Ele vai então pegar este dado para construir um novo objeto baseado nele e vai ter uma cópia do objeto Mike.

34.5. Runtime Loadable Procedures

Runtime Loadable Procedures (PRP) é uma característica exclusiva que permite aos usuários GHI escrever código nativo C/Assembly e usá-lo por meio de código gerenciado (C#). O código nativo é centenas de vezes mais rápido, mas é menos fácil de admnistrar. Tarefas específicas, como cálculos de CRC tem uso intensivo de CPU, são particularmente adequados para tais características. A aplicação é feita usando código gerenciado (C#) mas apenas o método de cálculo do CRC é feito em código nativo (C/Assembly). Atualmente, ChipworkX e EMX são as únicas dispositiivos que dão apoio à RPL.

34.6. Banco de Dados

Um banco de dados armazena os dados em um lugar onde é fácil e rápido para recuperar mais tarde.

Procurar por um item de produto é muito rápido porque tem internamente indexadores do banco de dados.

Atualmente, ChipworkX e EMX são os únicos dispositivos do mundo que tem suporte a bancos de dados usando o SQLite.

34.7. Touch Screen (Tela sensíveis ao toque)

NETMF suporta telas sensíveis ao toque. Estas telas são uma boa combinação com display TFT. Um desenvolvedor pode criar uma aplicação gráfica com o WPF e o usuário pode controlá-lo usando a tela sensível ao toque.

34.8. Eventos

Se você tem um programa que recebe dados de uma porta serial, você deve verificar continuamente essa porta. Pode não ter dados para receber, mas como saber disto sem ficar testando ?Isso seria mais conveniente se pudéssemos ser notificado de que tem dados recebidos. Esta notificação pode vir de um "evento" (Event), que ocorre quando o driver recebeu os dados.

O mesmo raciocínio se aplica para portas de interrupção já relatado antes. Este livro não abrange a criação de eventos, mas nós já vimos como eram utilizados nas portas de interrupções e no mouse com suporte a USB host.

34.9. USB Host Raw

Nós aprendemos como acessar dispositivos USB através do serviço de suporte da GHI. GHI também permite escrever drivers de código gerenciado para a maioria dos dispositivos USB.

Acesso direto a USB é considerada um recurso bastante avançado e não é o objetivo inicial deste livro.

Aqui está um projeto que utiliza “raw” USB para ler um controlador Xbox

http://www.microframeworkprojects.com/index.php?title=Xbox_Controller

Outro projeto interessante é um driver NXT, que permite controlar o LEGO Mindstorms NXT do C# e Visual Studio:

http://www.microframeworkprojects.com/index.php?title=NXT_Mindstorm

35. Palavras finais

Se você achou este livro interessante e poupou-lhe tempo de pesquisa, então eu realizei o que eu tinha em mente. Eu te agredeço muito por baixar este livro e lê-lo.

35.1. Leituras complementares

Este livro cobriu as base do C# e .NET Micro Framework. Aqui temos uma lista de alguns recursos para aprender mais:

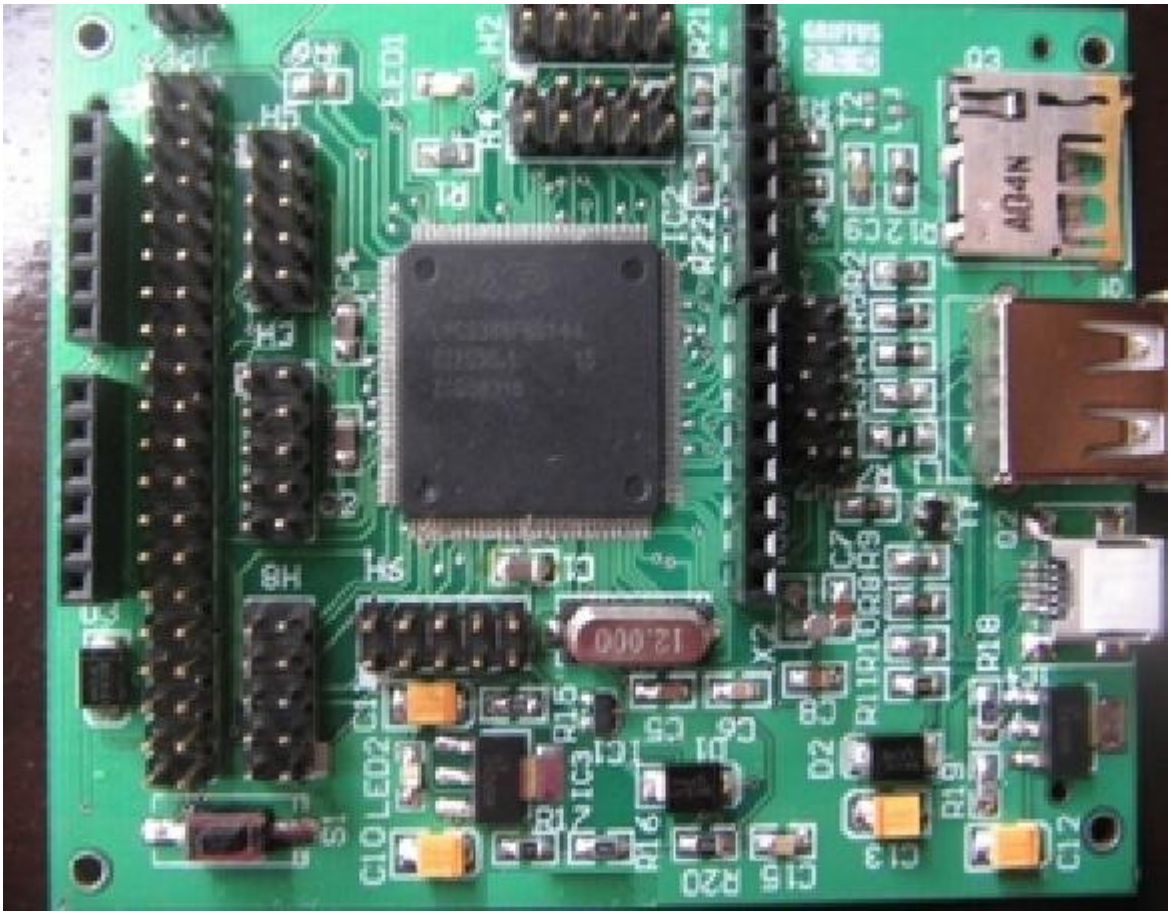
- Meu blog é sempre um bom lugar para visitar.
  http://tinyclr.blogspot.com/
- O site de projetos do Micro Framework Project é um ótimo recurso
  http://www.microframeworkprojects.com/
- Um bom e livro grátis para continuar aprendendo sobre C# está disponível em
  http://www.programmersheaven.com/2/CSharpBook
- Excelente livro de sobre .NET Micro Framework de Jens Kuhner
  http://www.apress.com/book/view/9781430223870
- USB complete é um excelente livro sobre USB
  http://www.lvr.com/usbc.htm
- Wikipedia é meu lugar favorito para por informação para todo mundo
  http://en.wikipedia.org/wiki/.NET_Micro_Framework
- Website da Microsoft sobre .NET Micro Framework
  http://www.microsoft.com/netmf

35.2. Notas

Este livro é grátis somente se você fizer o download direto da GHI Electtronics. Use para seu próprio conhecimento e é seu risco. Nem o escritor ou GHI electronics é responsável for algum dano ou perda causado por este livro ou pela informação suprida por ele. Não há garantia que qualquer informação neste livro é válida.

35.3. Placa Brasileira (BABUINO)

Miguel Alexandre Wisintainer desenvolveu uma placa baseada no USBizi. Quem tiver interesse em comprá-la no Brasil, entre em contato com ele pelo e-mail tcpipchip@hotmail.com



BABUINO