Gestion des graphiques pour les afficheurs non TFT

[Mise à jour le 31/12/2018]

Source : forum GHI

Introduction

Les cartes **G120**, **G400** et **UC5550** sous TinyCLR OS disposent d'une bibliothèque graphique ciblant les écrans TFT. **GHI** a activé partiellement cette bibliothèque sur des périphériques à faible mémoire, comme les cartes **FEZ** et **BrainPad**. Pour cela, ils ont ajouté une couche d'indirection dans l'implémentation des APIs de l'espace de nom **System.Drawing** afin de les faire fonctionner avec les périphériques GPIO, SPI et autres.

Accès aux opérations graphiques

Pour que cela fonctionne, il suffit d'appeler la méthode **GraphicsManager.RegisterDrawTarget** après avoir installé le nuget TinyCLR.Drawing et déclaré l'espace de noms **GHIElectronics.TinyCLR.Drawing**. Cette méthode prend un seul paramètre de type **IDrawTarget** et retourne un **IntPtr**. La valeur de retour représente un descripteur de contexte du périphérique qu'il sufit de transmettre à la méthode **Graphics.FromHdc** comme avec les écrans TFT. À partir de là, on peut utiliser plusieurs méthodes d'affichage pour effectuer des opérations graphiques telles que **DrawEllipse** et **DrawString**.

Exemple

```
var hdc = GraphicsManager.RegisterDrawTarget(new DrawTarget(disp));
var screen = Graphics.FromHdc(hdc);

screen.DrawEllipse(Colors.Blue, 10, 10, 10, 10);
screen.Flush();
```

Implémentation de la classe DrawTarget

- **Présentation**

La manière dont on implémente la classe **DrawTarget** (héritant de l'interface **IDrawTarget**) dépend du type d'afficheur ciblé. Les principales fonctionnalité d'IDrawTarget sont **SetPixel** et **Flush**. Tous les algorithmes de dessin implémentés dans la bibliothèque fournie finissent par appeler **SetPixel**

pour mettre à jour la structure de données interne, puis **Flush** pour mettre à jour l'affichage.

> Nous devons donc implémenter une version de **SetPixel** et **Flush** adaptée à l'afficheur ciblé pour la classe **DrawTarget**.

Prenons l'exemple d'un écran **SPI** à **ST7735** comme celui qui est connecté à une **carte de développement G80**. Il présente une zone d'affichage de 1**60 x 128 pixels**, orientée de gauche à droite, et attend des données au format **RGB565**. Cela signifie que chaque pixel est composé de deux octets: **5 bits rouges**, **6 bits verts** et **5 bits bleus**, dans cet ordre. Pour stocker la totalité des pixels de l'afficheur, nous devons créer un **tableau d'octets** de **largeur x hauteur x nb octets par pixel**. Donc, pour cet affichage, cela donne 128 * 160 * 2 = **40 960** octets. Dans l'image ci-dessous, chaque case soulignée en noir représente un pixel avec ses octets constitutifs et chaque nombre représente l'octet dans l'ordre dans lequel il sera envoyé à l'écran. Notez qu'il y a 320 octets sur la première ligne, car chacun des 160 pixels a deux octets.

0	1	2	3	...	...	316	317	318	319
320	321	322	323	...	...	636	637	638	639
...	...	...	...	...	...	...	...	...	...
40640	40641	40642	40643	...	...	40956	40957	40958	40959

- **Stockage des pixels**

Les octets représentant les pixels sont stockés dans un tableau (mémoire tampon). On crée un champ privé dans la classe qui implémente **IDrawTarget**.

```
private readonly byte[] buffer;
```

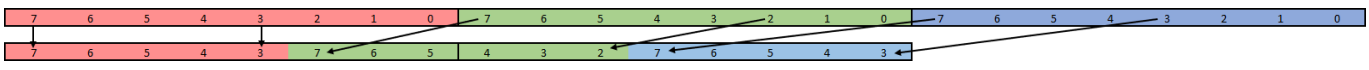
- **Implémentation de la méthode SetPixel**

Dans la fonction SetPixel, on détermine les coordonnées x et y du pixel et la couleur à afficher. On appelle **ToArgb** sur cette couleur pour obtenir la couleur au format **RGB888** soit 8 bits pour chacune des couleurs rouge, vert et bleu.

> Comment comvertir le code d'une couleur au format RGB888 en **RGB565** et le stocker dans le tableau d'octets ?

Pour cela, on sélectionne les bits d'ordre le plus élevé dans le format RGB888, car ce sont eux qui définissent le plus la couleur, puis on les réduit à une matrice RGB565. Pour savoir quel octet du tableau contient quel pixel , on doit effectuer l'opération **(y * largeur + x) * 2**, de sorte que pour le pixel situé en (159, 1), le résultat soit (1 * 160 + 159) * 2 = 638 comme cela est représenté dans l'image ci-dessus.

L'image ci-dessous montre où chaque bit d'un pixel place avec cette conversion. Chaque case en gras représente un octet. Chaque cellule numérotée est un bit dans la couleur donnée.



Vous pouvez observer comment le composant vert est partagé entre les deux octets. Le premier octet est composé des cinq bits de poids fort du rouge et des trois bits de poids fort du vert. Le deuxième

octet est composé des trois bits de poids faible du vert et des cinq bits de poids fort du bleu.

Le code ci-dessous montre comment implémenter cette conversion. Nous extrayons uniquement les bits souhaités de la couleur d'origine, puis nous les décalons afin d'avoir les couleurs dans des variables car la couleur d'origine est codée sur un entier long. Nous devons ensuite sélectionner à nouveau les éléments souhaités, à savoir cinq éléments parmi le rouge et le bleu, et six autres dans le vert, puis les déplacer à l'endroit où nous en avons besoin dans les octets codant le résultat, enfin nous les combinons. Pour en savoir plus, voir l'article de Wikipedia si vous ne connaissez pas les opérations au niveau des bits.

```
public void SetPixel(int x, int y, Color color) {
    if (x < 0 || y < 0 || x >= this.Width || y >= this.Height) return;

    var idx = (y * this.Width + x) * 2;
    var clr = color.ToArgb();
    var red = (clr & 0b0000_0000_1111_1111_0000_0000_0000_0000) >> 16;
    var green = (clr & 0b0000_0000_0000_0000_1111_1111_0000_0000) >> 8;
    var blue = (clr & 0b0000_0000_0000_0000_0000_0000_1111_1111) >> 0;

    this.buffer[idx] = (byte)((red & 0b1111_1000) | ((green & 0b1110_0000)
>> 5));
    this.buffer[idx + 1] = (byte)(((green & 0b0001_1100) << 3) | ((blue &
0b1111_1000) >> 3));
}
```

> Le tableau contient maintenant le code des pixels mais comment pouvons-nous l'afficher ?

- **Implémentation de la méthode Flush**

Une solution consiste à transmettre un **DisplayController** (qui implémente **IDisplayControllerProvider**) à la classe, puis à appeler **DrawBuffer** dessus. Vous vous assurerez que le contrôleur d'affichage attend les données dans le format dans lequel nous les convertissons. Comme nous stockons les données au format RGB565, c'est ce que l'affichage sur la carte de développement attend sans aucun problème.

```
public void Flush() => this.parent.DrawBuffer(0, 0, this.Width, this.Height,
this.buffer, 0);
```

DrawBuffer permet de spécifier un emplacement et une taille personnalisés, mais comme nous dessinons tout l'écran, nous commençons à (0, 0).

Implémentation de la classe ST7735Controller

La prochaine étape consiste à contrôler la carte graphique. Pour cela on créera une classe implémentant **IDisplayControllerProvider**. La principale fonction est **DrawBuffer** appelée ci-dessus. Elle envoie le tampon à l'affichage. Le contrôle d'un afficheur à ST7735 se fait avec une liaison SPI, la classe doit donc intégrer un périphérique SPI sur lequel elle pourra écrire. Cette classe doit également configurer l'affichage.

Le code ci-dessous implémente cette fonction et effectue les étapes nécessaires pour transférer les

données à l'afficheur ST7735. Dans cet exemple, certains paramètres sont ignorés, bien que des applications plus avancées puissent les utiliser.

```
void IDisplayControllerProvider.DrawBuffer(int x, int y, int width, int
height, byte[] data, int offset) {
    this.SendCommand(ST7735CommandId.RAMWR);
    this.control.Write(GpioPinValue.High);
    this.spi.Write(data, offset, data.Length);
}
```

On trouvera ci-dessous un exemple complet pouvant être exécuté tel quel sur la carte de développement G80 ou sur une carte **Panda III** munie d'un shield **Adafruit 1.8"TFT** à l'aide de la dernière version de TinyCLR OS. Il dessine une petite boule blanche qui rebondit sur l'écran. Il crée toutes les broches nécessaires et effectue l'initialisation de l'écran.

> Ce driver est disponible dans le package ST7735 NuGet.

GHI a également créé des buffer pour RGB565, RGB444 et VerticalByteStrip1Bpp disponibles dans l'espace de noms GHIElectronics.TinyCLR.Drawing. sous la forme BufferDrawTargetRgb444 (et connexe). Vous devez simplement dériver l'une de ces classes et implémenter votre logique Flush.

Vous pouvez voir un exemple complet pour le module d'affichage Adafruit sur GitHub.

Code de l'exemple G80DevBoardDisplay

G80DevBoardDisplay.cs

```
using GHIElectronics.TinyCLR.Devices.Display;
using GHIElectronics.TinyCLR.Devices.Display.Provider;
using GHIElectronics.TinyCLR.Devices.Gpio;
using GHIElectronics.TinyCLR.Devices.Spi;
using GHIElectronics.TinyCLR.Drawing;
using GHIElectronics.TinyCLR.Pins;
using GHIElectronics.TinyCLR.Drivers.Sitronix.ST7735;
using System;
using System.Drawing;
using System.Threading;

namespace G80DevBoardDisplay {
    public static class Program {
        public static void Main() {
             //var spi = SpiController.FromName(G80.SpiBus.Spi2);
            var spi = SpiController.FromName(G80.SpiBus.Spi1);
            var gpio = GpioController.GetDefault();
            //var st7735 = new
ST7735Controller(spi.GetDevice(ST7735Controller.GetConnectionSettings(S
piChipSelectType.Gpio, G80.GpioPin.PD10)),
gpio.OpenPin(G80.GpioPin.PE10), gpio.OpenPin(G80.GpioPin.PE12));
            var st7735 = new
```

```
ST7735Controller(spi.GetDevice(ST7735Controller.GetConnectionSettings(S
piChipSelectType.Gpio, FEZPandaIII.GpioPin.D10)),
gpio.OpenPin(FEZPandaIII.GpioPin.D8),
gpio.OpenPin(FEZPandaIII.GpioPin.D9));

            var disp = DisplayController.FromProvider(st7735);
            disp.SetConfiguration(new SpiDisplayControllerSettings {
Width = 160, Height = 128 });

            var bl = gpio.OpenPin(G80.GpioPin.PC7);
            bl.Write(GpioPinValue.High);
            bl.SetDriveMode(GpioPinDriveMode.Output);

            var hdc = GraphicsManager.RegisterDrawTarget(new
DrawTarget(disp));
            var screen = Graphics.FromHdc(hdc);

            var rnd = new Random();
            var x = rnd.Next(160);
            var y = rnd.Next(128);
            var vx = rnd.Next(20) - 10;
            var vy = rnd.Next(20) - 10;
            var color = new Pen(Color.White);

            while (true) {
                x += vx;
                y += vy;

                if (x >= 160 || x < 0) vx *= -1;
                if (y >= 128 || y < 0) vy *= -1;

                screen.Clear(Color.Black);
                screen.DrawEllipse(color, x, y, 10, 10);
                screen.Flush();

                Thread.Sleep(10);
            }
        }
    }

    public sealed class DrawTarget : IDrawTarget {
        private readonly DisplayController parent;
        private readonly byte[] buffer;

        public DrawTarget(DisplayController parent) {
            this.parent = parent;

            this.Width = parent.ActiveConfiguration.Width;
            this.Height = parent.ActiveConfiguration.Height;

            this.buffer = new byte[this.Width * this.Height * 2];
```

```
        }

        public int Width { get; }
        public int Height { get; }

        public void Dispose() { }
        public byte[] GetData() => this.buffer;
        public Color GetPixel(int x, int y) => throw new
NotSupportedException();

        public void Clear(Color color) => Array.Clear(this.buffer, 0,
this.buffer.Length);

        public void Flush() => this.parent.DrawBuffer(0, 0, this.Width,
this.Height, this.buffer, 0);

        public void SetPixel(int x, int y, Color color) {
            if (x < 0 || y < 0 || x >= this.Width || y >= this.Height)
return;

            var idx = (y * this.Width + x) * 2;
            var clr = color.ToArgb();
            var red = (clr & 0b0000_0000_1111_1111_0000_0000_0000_0000)
>> 16;
            var green = (clr &
0b0000_0000_0000_0000_1111_1111_0000_0000) >> 8;
            var blue = (clr &
0b0000_0000_0000_0000_0000_0000_1111_1111) >> 0;

            this.buffer[idx] = (byte)((red & 0b1111_1000) | ((green &
0b1110_0000) >> 5));
            this.buffer[idx + 1] = (byte)(((green & 0b0001_1100) << 3)
| ((blue & 0b1111_1000) >> 3));
        }
    }

    public class ST7735Controller : IDisplayControllerProvider {
        private readonly byte[] buffer1 = new byte[1];
        private readonly byte[] buffer4 = new byte[4];
        private readonly SpiDevice spi;
        private readonly GpioPin control;
        private readonly GpioPin reset;

        public static SpiConnectionSettings
GetConnectionSettings(SpiChipSelectType chipSelectType, int
chipSelectLine) => new SpiConnectionSettings {
            Mode = SpiMode.Mode3,
            ClockFrequency = 12_000_000,
            DataBitLength = 8,
            ChipSelectType = chipSelectType,
            ChipSelectLine = chipSelectLine
```

```
        };

        public ST7735Controller(SpiDevice spi, GpioPin control, GpioPin
reset) {
            this.spi = spi;

            this.control = control;
            this.control.SetDriveMode(GpioPinDriveMode.Output);

            this.reset = reset;
            this.reset.SetDriveMode(GpioPinDriveMode.Output);

            this.reset.Write(GpioPinValue.Low);
            Thread.Sleep(50);

            this.reset.Write(GpioPinValue.High);
            Thread.Sleep(200);

            this.SendCommand(ST7735CommandId.SWRESET);
            Thread.Sleep(120);

            this.SendCommand(ST7735CommandId.SLPOUT);
            Thread.Sleep(120);

            this.SendCommand(ST7735CommandId.FRMCTR1);
            this.SendData(0x01);
            this.SendData(0x2C);
            this.SendData(0x2D);

            this.SendCommand(ST7735CommandId.FRMCTR2);
            this.SendData(0x01);
            this.SendData(0x2C);
            this.SendData(0x2D);

            this.SendCommand(ST7735CommandId.FRMCTR3);
            this.SendData(0x01);
            this.SendData(0x2C);
            this.SendData(0x2D);
            this.SendData(0x01);
            this.SendData(0x2C);
            this.SendData(0x2D);

            this.SendCommand(ST7735CommandId.INVCTR);
            this.SendData(0x07);

            this.SendCommand(ST7735CommandId.PWCTR1);
            this.SendData(0xA2);
            this.SendData(0x02);
            this.SendData(0x84);

            this.SendCommand(ST7735CommandId.PWCTR2);
```

```
            this.SendData(0xC5);

            this.SendCommand(ST7735CommandId.PWCTR3);
            this.SendData(0x0A);
            this.SendData(0x00);

            this.SendCommand(ST7735CommandId.PWCTR4);
            this.SendData(0x8A);
            this.SendData(0x2A);

            this.SendCommand(ST7735CommandId.PWCTR5);
            this.SendData(0x8A);
            this.SendData(0xEE);

            this.SendCommand(ST7735CommandId.VMCTR1);
            this.SendData(0x0E);

            this.SendCommand(ST7735CommandId.GAMCTRP1);
            this.SendData(0x0F);
            this.SendData(0x1A);
            this.SendData(0x0F);
            this.SendData(0x18);
            this.SendData(0x2F);
            this.SendData(0x28);
            this.SendData(0x20);
            this.SendData(0x22);
            this.SendData(0x1F);
            this.SendData(0x1B);
            this.SendData(0x23);
            this.SendData(0x37);
            this.SendData(0x00);
            this.SendData(0x07);
            this.SendData(0x02);
            this.SendData(0x10);

            this.SendCommand(ST7735CommandId.GAMCTRN1);
            this.SendData(0x0F);
            this.SendData(0x1B);
            this.SendData(0x0F);
            this.SendData(0x17);
            this.SendData(0x33);
            this.SendData(0x2C);
            this.SendData(0x29);
            this.SendData(0x2E);
            this.SendData(0x30);
            this.SendData(0x30);
            this.SendData(0x39);
            this.SendData(0x3F);
            this.SendData(0x00);
            this.SendData(0x07);
            this.SendData(0x03);
```

```
            this.SendData(0x10);

            this.SendCommand(ST7735CommandId.COLMOD);
            this.SendData(0x05);

            this.SendCommand(ST7735CommandId.MADCTL);
            this.SendData(0b1010_0000);

            this.buffer4[1] = 0;
            this.buffer4[3] = 159;
            this.SendCommand(ST7735CommandId.CASET);
            this.SendData(this.buffer4);

            this.buffer4[1] = 0;
            this.buffer4[3] = 127;
            this.SendCommand(ST7735CommandId.RASET);
            this.SendData(this.buffer4);

            this.SendCommand(ST7735CommandId.DISPON);
        }

        private void SendCommand(ST7735CommandId command) {
            this.buffer1[0] = (byte)command;
            this.control.Write(GpioPinValue.Low);
            this.spi.Write(this.buffer1);
        }

        private void SendData(byte data) {
            this.buffer1[0] = data;
            this.control.Write(GpioPinValue.High);
            this.spi.Write(this.buffer1);
        }

        private void SendData(byte[] data) {
            this.control.Write(GpioPinValue.High);
            this.spi.Write(data);
        }

        void IDisplayControllerProvider.DrawBuffer(int x, int y, int
width, int height, byte[] data, int offset) {
            this.SendCommand(ST7735CommandId.RAMWR);
            this.control.Write(GpioPinValue.High);
            this.spi.Write(data, offset, data.Length);
        }

        DisplayInterface IDisplayControllerProvider.Interface =>
DisplayInterface.Spi;
        DisplayDataFormat[]
IDisplayControllerProvider.SupportedDataFormats => new[] {
DisplayDataFormat.Rgb565 };
```

```
        void IDisposable.Dispose() { }
        void IDisplayControllerProvider.Enable() { }
        void IDisplayControllerProvider.Disable() { }
        void
IDisplayControllerProvider.SetConfiguration(DisplayControllerSettings
configuration) { }
        void IDisplayControllerProvider.DrawString(string value) { }
        void IDisplayControllerProvider.DrawPixel(int x, int y, long
color) { }
    }

    public enum ST7735CommandId : byte {
        //System
        NOP = 0x00,
        SWRESET = 0x01,
        RDDID = 0x04,
        RDDST = 0x09,
        RDDPM = 0x0A,
        RDDMADCTL = 0x0B,
        RDDCOLMOD = 0x0C,
        RDDIM = 0x0D,
        RDDSM = 0x0E,
        SLPIN = 0x10,
        SLPOUT = 0x11,
        PTLON = 0x12,
        NORON = 0x13,
        INVOFF = 0x20,
        INVON = 0x21,
        GAMSET = 0x26,
        DISPOFF = 0x28,
        DISPON = 0x29,
        CASET = 0x2A,
        RASET = 0x2B,
        RAMWR = 0x2C,
        RAMRD = 0x2E,
        PTLAR = 0x30,
        TEOFF = 0x34,
        TEON = 0x35,
        MADCTL = 0x36,
        IDMOFF = 0x38,
        IDMON = 0x39,
        COLMOD = 0x3A,
        RDID1 = 0xDA,
        RDID2 = 0xDB,
        RDID3 = 0xDC,

        //Panel
        FRMCTR1 = 0xB1,
        FRMCTR2 = 0xB2,
        FRMCTR3 = 0xB3,
        INVCTR = 0xB4,
```

```
        DISSET5 = 0xB6,
        PWCTR1 = 0xC0,
        PWCTR2 = 0xC1,
        PWCTR3 = 0xC2,
        PWCTR4 = 0xC3,
        PWCTR5 = 0xC4,
        VMCTR1 = 0xC5,
        VMOFCTR = 0xC7,
        WRID2 = 0xD1,
        WRID3 = 0xD2,
        NVCTR1 = 0xD9,
        NVCTR2 = 0xDE,
        NVCTR3 = 0xDF,
        GAMCTRP1 = 0xE0,
        GAMCTRN1 = 0xE1,
    }
}
```