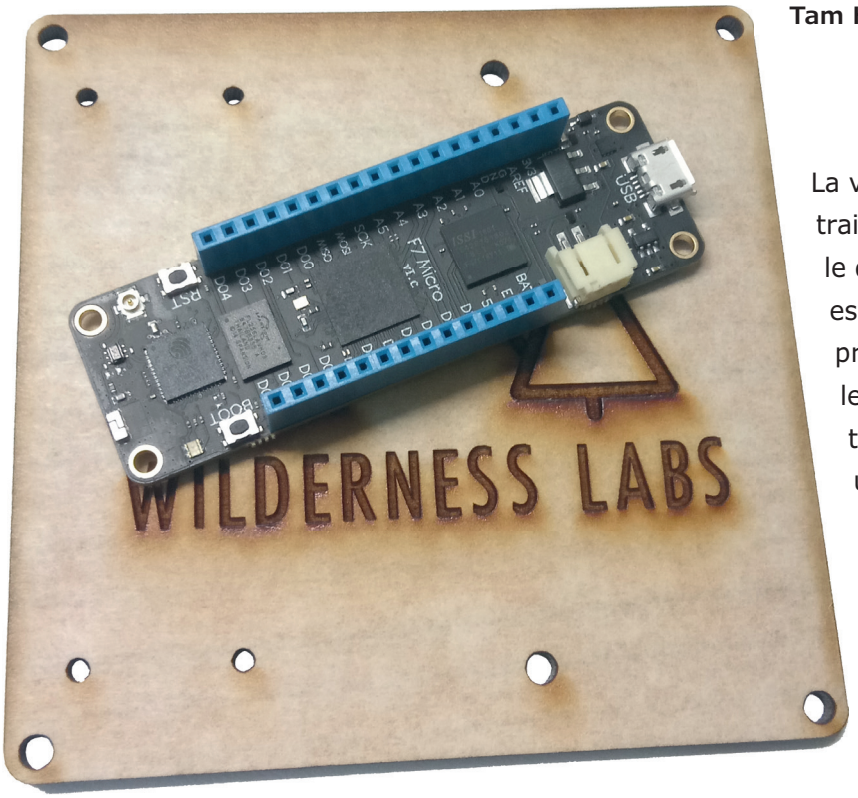
carte Meadow F7

Une carte pour développeurs .NET

Tam Hanna (Allemagne)



(Source : Wilderness Labs)

La variété des cartes de calculateurs de traitement (ou de *process*) disponibles pour le développement de nouvelles applications est étonnante, mais si vous voulez programmer en *Visual Basic* ou en C#, le choix se réduit. C'est comme ça qu'on tombe sur la nouvelle carte *Meadow* F7 qui utilise un STM32F777 pour l'exécution du *runtime* .NET, tandis qu'un module ESP32 se charge du Wi-Fi.

Pendant son mandat de vice-président chez *Xamarin*, Bryan Costanich était occupé à porter l'environnement .NET sur Android et iOS. Il a ensuite acheté les droits de propriété intellectuelle à une société dirigée par Chris Walker et a créé *Wilderness Labs*. Leur dernier produit est l'ordinateur de traitement *Meadow F7*. Son ambition est de fournir aux développeurs .NET un accès de *première classe* à l'écosystème de l'IdO.
Du point de vue de l'architecture, ça ressemble à ce qu'on obtenait jusqu'à présent en combinant Raspberry Pi ou Orange Pi et un cœur de processeur en temps réel. Un STM32F777 cadencé à 216 MHz se charge du *runtime* .NET, un module ESP32 de la connexion Wi-Fi.

Allons-y, le bonheur est dans le pré

Telle qu'elle est livrée, la carte *Meadow* F7 (*meadow* signifie pré ou pâturage en anglais) se présente avec un système d'exploitation soit obsolète, soit pas installé du tout. L'installation se fait avec le chargeur d'amorce STM. Téléchargez [**1**] le système d'exploitation composé de deux fichiers. Il vous faut ensuite des DFU-utils [**2**].
La documentation de la carte indique que pour lancer le chargeur de démarrage il faut appuyer sur le bouton de r-à-z. Maintenez ce bouton enfoncé pendant que vous connectez la carte à votre poste de travail *Windows* à l'aide d'un câble micro USB. Nous utilisons `list` pour obtenir le numéro de série de l'ordinateur de traitement :

```
C:\dfu-util-0.9-win64>dfu-util --list
dfu-util 0,9

. . .
Cannot open DFU device 0483:df11
```

Oups - ça fonctionne sous d'anciennes versions de Windows, mais pas sous Windows 10 (voir l'explication [**3**]). Pour contourner le problème, suivre les instructions d'initialisation du pilote. Après cela, le numéro de série peut être déterminé ainsi :

```
C:\dfu-util-0.9-win64>dfu-util --list
dfu-util 0.9
. . .
Found DFU: [0483:df11] ver=2200, devnum=4,
   cfg=1, intf=0, path="5-3", alt=3, name="@
   Device Feature/0xFFFF0000/01*004 e",
   serial="346B38733536"
```

Dans la liste de *Windows, l*es cartes *Meadow* reconnues n'apparaîtront pas en un seul exemplaire, mais comme quatre périphériques. Pour nous, seul le numéro de série importe, nous en avons besoin dans l'étape suivante pour charger noyau et *runtime*. Pensez à modifier les lignes de commande présentées ici en accord avec votre propre situation. Assurez-vous

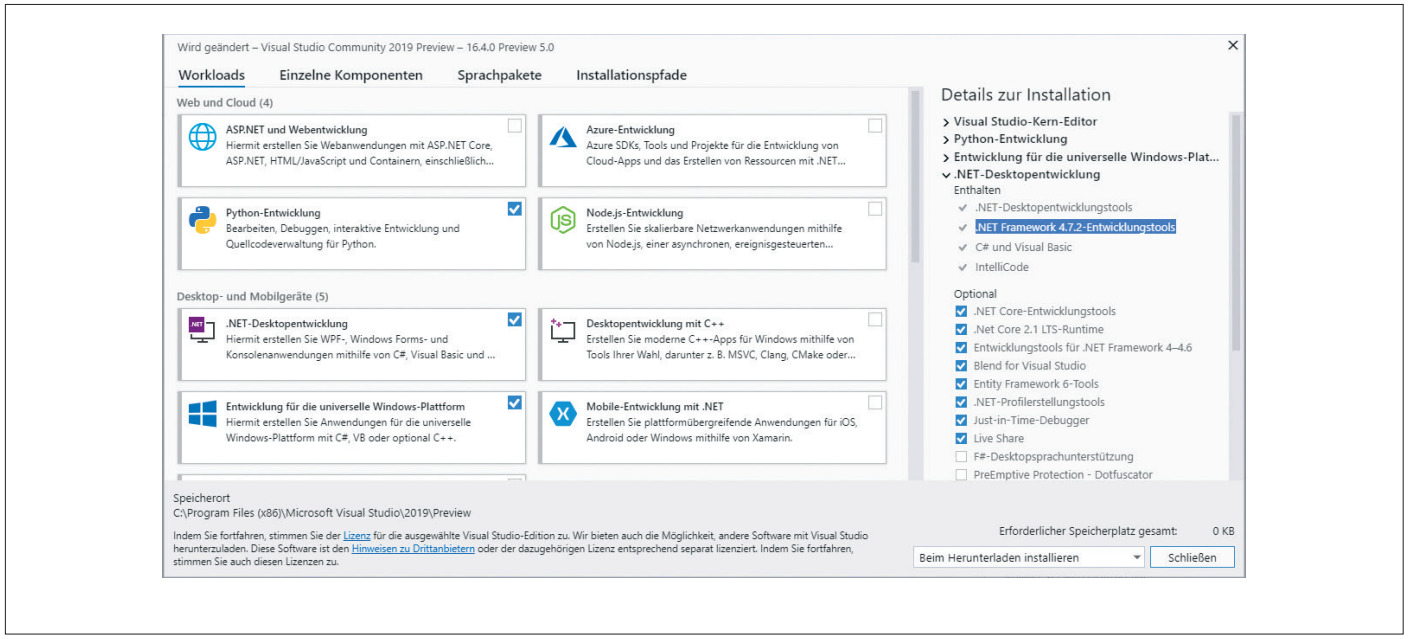


Figure 1. Meadow n'est guère amusant sans les outils de développement du SDK de .NET Framework 4.7.2.

de saisir les adresses hexadécimales sans faute de frappe et placez dans le bon dossier les deux fichiers `Meadow.OS_Kernel.bin` et `Meadow.OS_Runtime.bin` :

```
C:\dfu-util-0.9-win64> dfu-util -a 0 -S 346B38733536
   -D Meadow.OS_Kernel.bin -s 0x08000000
dfu-util 0,9
C:\dfu-util-0.9-win64> dfu-util -a 0 -S 346B38733536
   -D Meadow.OS_Runtime.bin -s 0x08040000
dfu-util 0,9
```

Pour garantir un bon démarrage, appuyez sur le bouton RST : la LED clignotera un instant. Comme environnement de développement, j'ai utilisé *Visual Studio* dans sa version communauté libre 2019.8. Pour l'installation, les composants mentionnés sur la **fig. 1** sont sélectionnés et téléchargés par l'assistant d'installation, disponible dans le menu de démarrage sous *Visual Studio Installer*.

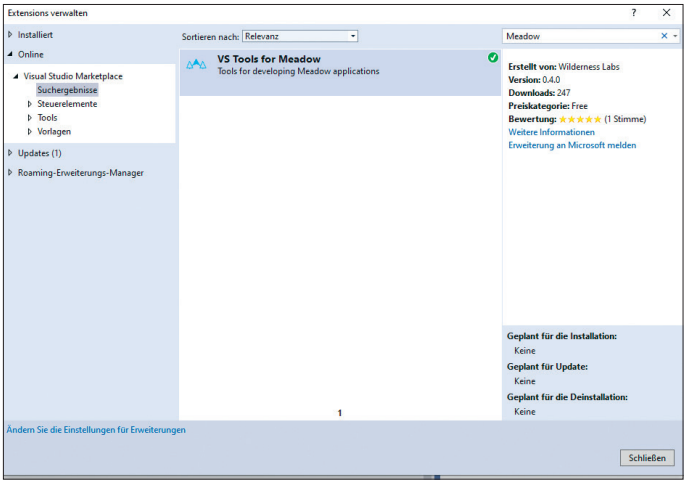


Figure 2. Sur demande, *Visual Studio* obtient automatiquement les mises à jour sur l'internet.

Ensuite, démarrez *Visual Studio* comme d'habitude. Si l'EDI ne démarre pas avec un projet déjà ouvert, le nouvel assistant de démarrage peut être désactivé en cliquant sur l'option *Continuer sans code*.

Wilderness Labs fournit le SDK actuel sous la forme d'un *plug-in Visual Studio*. Cliquez sur *Extensions Manage Extensions* pour charger l'assistant du *plug-in*. Nous passons ensuite à la section *Online* (**fig. 2**), et cherchons la chaîne *Meadow*.

Après un redémarrage obligatoire, il y aura un nouveau modèle appelé *Meadow Application,* qui vous servira de base pour vos propres expériences. Créez un nouveau programme appelé *ElektorSample* pour visualiser le code de l'exemple de LED clignotante fourni avec le chargeur de démarrage. Le code dans le fichier `MeadowApp.cs` devrait être explicite.

Les développeurs venus d'Arduino doivent tenir compte de l'absence de structure en boucle du *framework* lui-même. L'exemple ci-dessus initialise simplement les deux méthodes à partir du constructeur : une boucle sans fin est implémentée dans les `BlinkLeds` :

```
public MeadowApp()
{
    ConfigurePorts() ;
    BlinkLeds() ;
}
```

Tests : démarrage et ping

Si un ordinateur de traitement doit assurer des tâches difficiles en temps réel, le système d'exploitation doit pouvoir garantir des temps de réponse. Les systèmes basés sur des langages gérés tels Java ou C# sont généralement en peine à cet égard. Pas seulement à cause de faiblesses dans l'exécution, mais toutes les autres tâches sont bloquées tant que dure le

nettoyage de la mémoire (*Garbage Collector*). C'est pourquoi notre petit programme de test se passe presque entièrement d'allocation dynamique de mémoire. Cependant, si une tâche de commande alloue et libère beaucoup de mémoire, le risque de mise en marche du nettoyage de la mémoire augmente.
Pour tester le comportement de la carte *Meadow* F7, nous pouvons écrire une routine qui bascule simplement une broche de sortie pour produire une certaine forme d'onde de sortie. Pour cela, les `ConfigurePorts` et les `BlinkLeds` sont utilisés comme suit :

```
public class MeadowApp : App<F7Micro, MeadowApp> {
        IDigitalOutputPort myOut;

        public MeadowApp()
. . .
        public void ConfigurePorts() {
            Console.WriteLine("Creating Outputs...");
            myOut= Device.
   CreateDigitalOutputPort(Device.Pins.D05);
        }

        public void BlinkLeds() {
            var state = false;

            while (true) {
                myOut.State = true;
                myOut.State = false;
                myOut.State = true;
                myOut.State = false;
            }
        }
    }
```

Pour interagir avec des éléments physiques, le logiciel *Meadow* utilise des classes d'abstraction. Notre port numérique est créé, p. ex. par une interface du type `IDigitalOutputPort` — si vous deviez proposer une extension qui expose également les broches GPIO, vous pourriez (avec un pilote approprié) déplacer le code entre le périphérique «ordinaire» et le nouveau périphérique.
Pour examiner la forme d'onde, on a utilisé un oscilloscope numérique à mémoire Tektronix TDS754D modifié pour obtenir une bande passante de 1 GHz. C'est du matériel importé des États-Unis, remis à neuf, fourni avec un LCD en option, à un prix raisonnable.

Puisque *Meadow* F7 est vu par *Windows* comme un simple port COM, il faudra donner un coup de pouce à *Visual Studio* pour l'installation. Dans la première étape, cliquez sur *View Other Windows Meadow* (ou *Ctrl+Maj+M*) pour activer la fenêtre de sélection de l'appareil appelée *Meadow Device Explorer*. Sélectionnez votre *Meadow* pour lancer un débogage. Chaque fois que le programme est lancé, la fenêtre de la **fig. 3** s'affiche, que vous fermerez en appuyant sur la touche Entrée.
Après l'installation, connectez l'oscillo à la broche D05 et observez la forme d'onde illustrée de la **fig. 4**. Il est normal que pendant l'exécution le *runtime* donne des erreurs de classes de donnée.

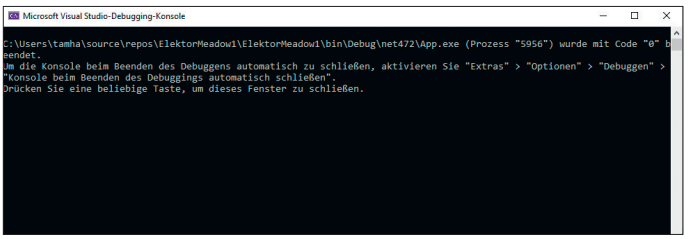


Figure 3. Pour fermer cette fenêtre, frappez la touche Entrée.

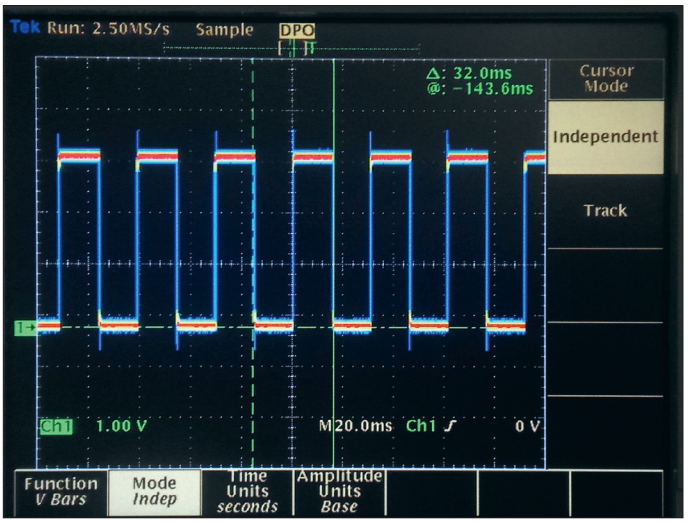


Figure 4. La commutation d'une broche de sortie prend environ 32 ms.

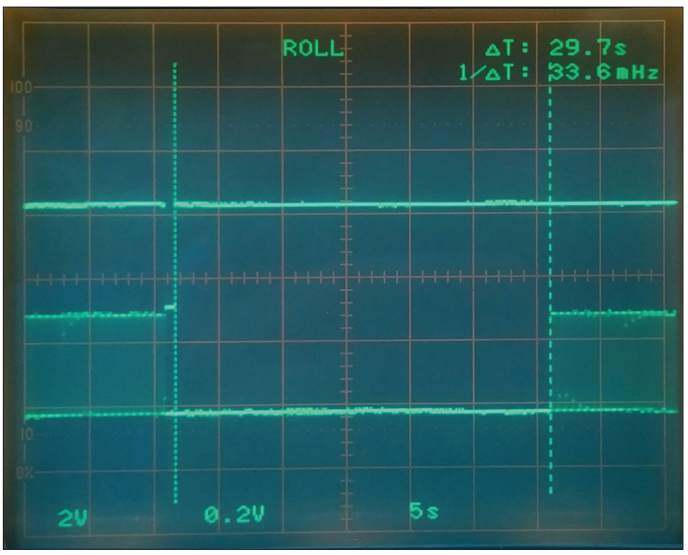


Figure 5. Après une remise à zéro (trace supérieure), il faut presque 3 s pour repartir.

Les différences de temps entre les bords de la forme d'onde sont très faibles. Cela indique que ce qui prend le plus de temps, c'est de changer l'état de la broche – un point d'optimisation possible.
La ligne de r-à-z n'est pas reliée seulement à un bouton-poussoir, mais aussi à une broche du long connecteur (la première connexion à côté du convertisseur de tension). Si votre oscilloscope a une option Roll-Mode et que vous avez de

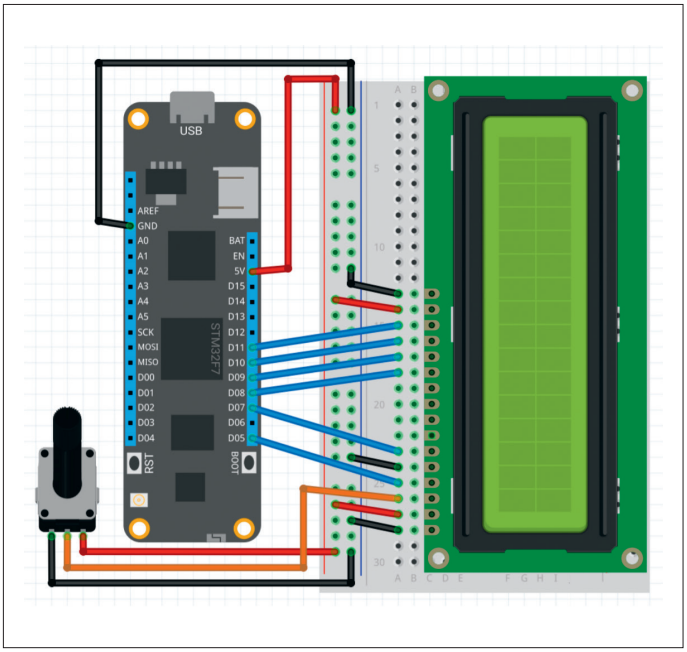


Figure 6. Branchez l'afficheur LCD sur une carte enfichable (Source : Wilderness Labs [5]).

la patience, vous pouvez utiliser l'initialisation pour voir que la carte met presque 30 s avant d'exécuter le programme (**fig. 5**) ! Pour les premiers tests, l'idéal est de connecter un simple afficheur (**fig. 6**) à *Meadow* F7 pour émettre quelques messages. Le paradigme de conception des classes d'abstraction décrites ci-dessus se poursuit également avec `CharacterDisplay.` Nous devons tout d'abord ajouter une autre instance à la classe `MeadowApp` qui sera chargée de communiquer avec l'afficheur :

```
public class MeadowApp : App<F7Micro, MeadowApp>  {
        CharacterDisplay myDisplay;
```

La structure des pilotes matériels est toujours la même sous *Meadow*. Le constructeur accepte d'abord une référence à un objet de périphérique – celui qui transfère le périphérique demande au pilote d'utiliser le matériel de sortie de l'ordinateur de traitement. Comme le Kinect SDK de *Microsoft*, c'est là une façon d'augmenter la flexibilité (théoriquement, on pourrait aussi procéder par extension de GPIO).

À l'étape suivante apparaît un groupe de paramètres nommés pour décrire les broches de sortie utilisées. Les valeurs du `Enum Device.Pins` y sont transférées – dans ce groupe, un champ de bits spécifique correspond à chaque périphérique matériel du processeur STM32, ce qui facilite la commande :

```
public void ConfigurePorts()
{
    Console.WriteLine("Creating Outputs...");
    myDisplay = new CharacterDisplay(
        Device,
        pinRS: Device.Pins.D05,
        pinE: Device.Pins.D07,
        pinD4: Device.Pins.D08,
        pinD5: Device.Pins.D09,
        pinD6: Device.Pins.D10,
        pinD7: Device.Pins.D11,
        rows: 4, columns: 20
        );
}
```

À ce stade, remarquons que *Visual Studio* ne peut pas résoudre la référence à la classe `CharacterDisplay.` Ceci est dû à la livraison modulaire de la structure (*framework*) – cliquez avec le bouton droit sur *Search* (*Rechercher)* dans l'explorateur de solutions et choisissez le gestionnaire de paquets NuGet (*package manager*). Ensuite, cherchez la chaîne *Meadow*Character* – l'astérisque se substitue à n'importe quel signe. Le paquet `Meadow.Foundation.Displays.LCD.CharacterDisplay` vaut la peine d'être installé, ce qui peut être fait comme une installation normale de *NuGet-Package*.

Il ne manque que la fonction d'édition réelle qui envoie le texte à l'afficheur. *Wilderness Labs* utilise l'infrastructure .NET ; la syntaxe de l'instruction `WriteLine` peut vous sembler familière. Le paramètre numérique supplémentaire définit la ligne sur laquelle la chaîne livrée doit être affichée. Si vous passez la valeur '1', elle apparaît sur la deuxième ligne en partant du haut :

```
public void BlinkLeds()
{
    var state = false;

    while (true)
    {
        myDisplay.WriteLine("Hello Elektor", 1);
        System.Threading.Thread.Sleep(1000);
    }
}
```

Téléchargez à nouveau le programme dans *Meadow* et voyez comment il apparaît à l'écran. Si vous utilisez le module inclus dans le kit de modification et que l'écran reste vide, modifiez le contraste de l'afficheur ; s'il est trop faible, vous ne voyez rien ! Remarquez aussi comme l'écriture à l'écran est lente – à l'heure de mettre sous presse cet article, il restait de la marge pour optimiser le code.

Gadget est mort… vive le gadget !

Quand *Microsoft* lançait en 2011 sa plate-forme .NET *Gadgeteer* (aujourd'hui disparue), son intention était de faciliter le prototypage aux développeurs mal à l'aise avec le matériel. Le système se composait d'une carte mère à microprocesseur prête à l'emploi et d'une gamme de modules de capteurs et d'E/S qui s'y connectaient à l'aide de câbles plats.
L'esprit est le même chez *Wilderness Labs*. L'offre *Meadow F7 Micro Development Kit w/Hack Kit Pro* (**fig. 7**), disponible en précommande [**6**], comprend une carte *Meadow* F7, deux cartes de prototypage, un afficheur LCD alphanumérique 4x20 de grande qualité, un assortiment varié de composants actifs et passifs, d'actionneurs et de capteurs ainsi qu'une carte de développement en bois MDF (reconstitué).
En plus de cela, il y a une bibliothèque de pilotes très étendue.

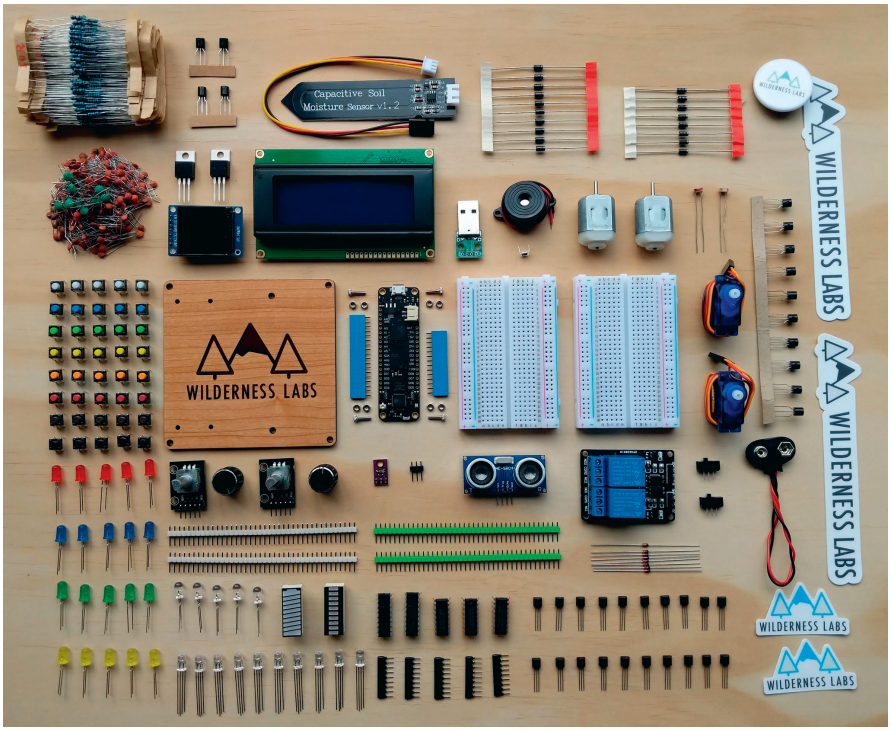


Figure 7. Le kit de développement est farci de petits cadeaux (Source : Wilderness Labs [6]).

Au moment où je boucle cet article, le pilote pour l'afficheur LCD en couleur (de qualité) n'est pas prêt, mais le kit lui-même devrait être disponible dès mars 2020. La gamme de capteurs apparaît dans la liste du matériel de *Wilderness Labs* [**4**].

Un mendiant n'a pas le choix, il prend ce qu'on lui donne. Ceux qui veulent utiliser .NET dans une application embarquée n'ont le choix pour l'instant qu'entre la carte désuète *NetDuino* et la carte *Meadow* F7. Pour les applications compatibles avec les performances d'entrées/sorties de la carte, on trouve une vaste bibliothèque de pilotes qui facilitent l'assemblage et les tests rapides des prototypes - l'avenir du gadget s'annonce radieux ! ⏮ (191190-03)

Liens

[1] Meadow F7 : http://beta-developer.wildernesslabs.co/Meadow/Getting_Started/Deploying_Meadow/

[2] DFU-utils : http://dfu-util.sourceforge.net/releases/dfu-util-0.9-win64.zip

[3] Bogue de Windows:
www.hanselman.com/blog/HowToFixDfuutilSTMWinUSBZadigBootloadersAndOtherFirmwareFlashingIssuesOnWindows.aspx

[4] Périphérie : http://developer.wildernesslabs.co/Meadow/Meadow.Foundation/Peripherals/

[5] Character Display :
http://beta-developer.wildernesslabs.co/docs/api/Meadow.Foundation/Meadow.Foundation.Displays.Lcd.CharacterDisplay.html

[6] Meadow Kit /w Hack :
https://store.wildernesslabs.co/collections/frontpage/products/meadow-f7-micro-development-board-w-hack-kit-pro