
Projet WT32 :

Tutoriel de prise en main

Nathanaël DEBREILLY – Eliez LE HUNSEC – Ilan CHANY



Table des matières

Introduction.....	1
1. Installation des outils	2
1.1. Installation de Arduino IDE.....	2
1.2. Configuration de Arduino IDE	4
2. Programmation	7
2.1. Utilisation du moniteur série	7
2.2. Configuration des pins	9
2.3. Programme permettant de paramétrer la bibliothèque	10
2.4. Programme principal.....	11
2.5. Commandes de bases pour LovyanGFX.....	13

Introduction

Dans ce tutoriel, nous détaillerons toutes les étapes nécessaires à la programmation d'une carte [WT32-SC01 Plus](#) à l'aide de Arduino IDE. Nous verrons quels outils sont requis et comment les configurer ainsi que la réalisation d'un programme de base.

Ce tutoriel à une durée estimée à environ 30 minutes et va nécessiter un minimum 1 Go d'espace mémoire, 500 Mo pour le logiciel Arduino IDE et environ 250 Mo pour les outils et configurations.

1. Installation des outils

Dans cette première partie, nous verrons quels outils sont nécessaires à la programmation de la carte WT32-SC01 Plus et comment les installer et les configurer.

1.1. Installation de Arduino IDE

Il existe de nombreux logiciels adaptés à la programmation sur microcontrôleur, dans ce tutoriel, nous privilégierons Arduino IDE (version 2.3.4) qui est gratuit, facile d'accès et intuitif. Commençons donc par installer le logiciel, rendez-vous pour cela sur le [site officiel de Arduino](https://www.arduino.cc/en/software) où vous pourrez le télécharger en suivant les étapes suivantes (nous utiliserons dans notre cas la version Windows 10 - 64 bits) :

Downloads



Arduino IDE 2.3.4

The new major release of the Arduino IDE is faster and even more powerful! In addition to a more modern editor and a more responsive interface it features autocompletion, code navigation, and even a live debugger.

For more details, please refer to the [Arduino IDE 2.0 documentation](#).

Nightly builds with the latest bugfixes are available through the section below.

SOURCE CODE

The Arduino IDE 2.0 is open source and its source code is hosted on [GitHub](#).

DOWNLOAD OPTIONS

- Windows** Win 10 and newer, 64 bits
- Windows** MSI installer
- Windows** ZIP file
- Linux** AppImage 64 bits (X86-64)
- Linux** ZIP file 64 bits (X86-64)
- macOS** Intel, 10.15: "Catalina" or newer, 64 bits
- macOS** Apple Silicon, 11: "Big Sur" or newer, 64 bits

[Release Notes](#)

FIGURE 1 : SITE ARDUINO

Vous pouvez alors exécuter le fichier « arduino-ide_2.3.4_Windows_64bits.exe » que vous venez de télécharger.

On arrive alors aux fenêtres suivantes qui vont permettre l'installation du logiciel :

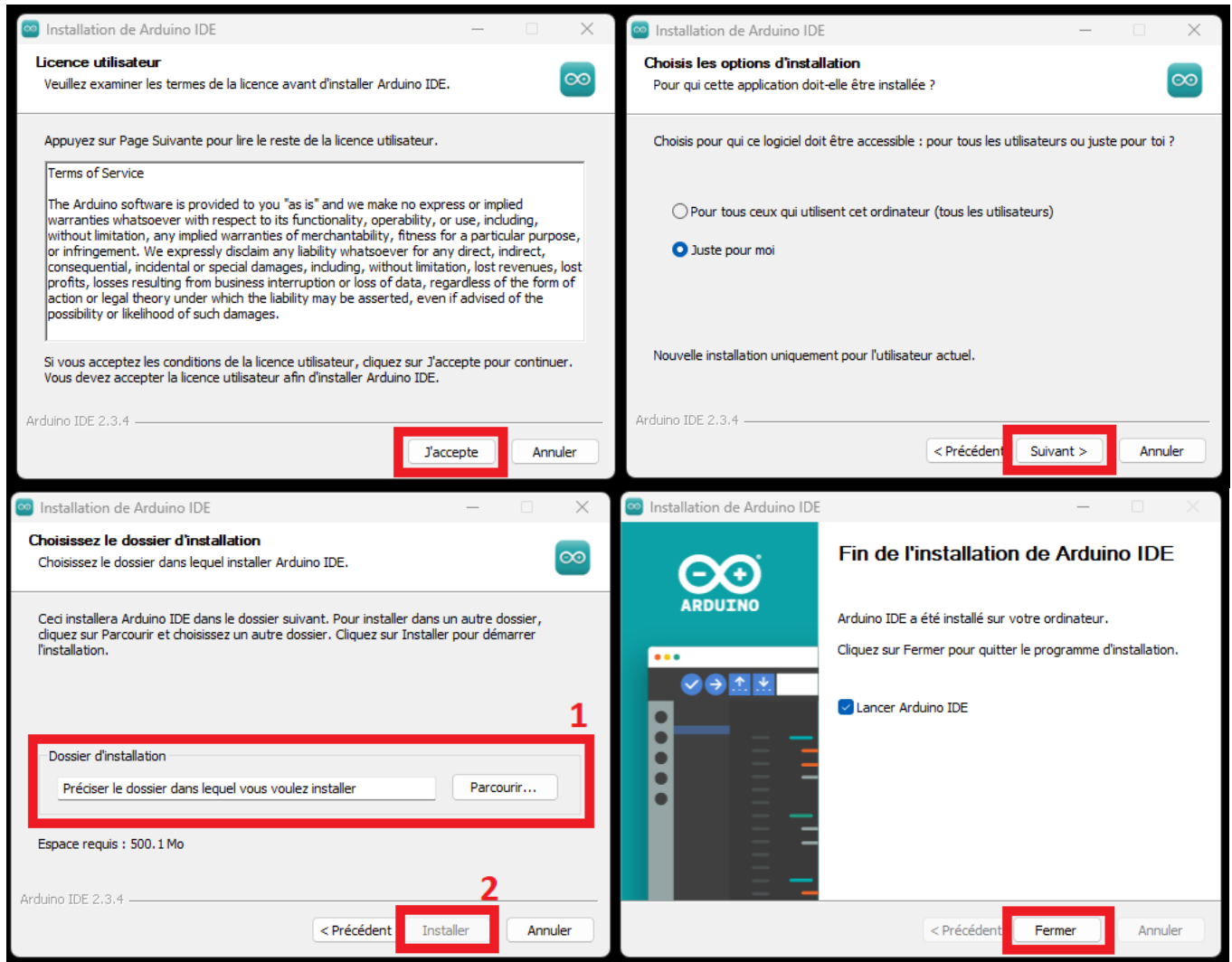


FIGURE 2 : INSTALLATION DE ARDUINO IDE

En fonction de votre ordinateur et des driver qui sont déjà installés, vous pouvez avoir des fenêtres supplémentaires pour l'installation de driver complémentaires, leur installation vous sera proposée directement et il suffira d'appuyer sur installer :

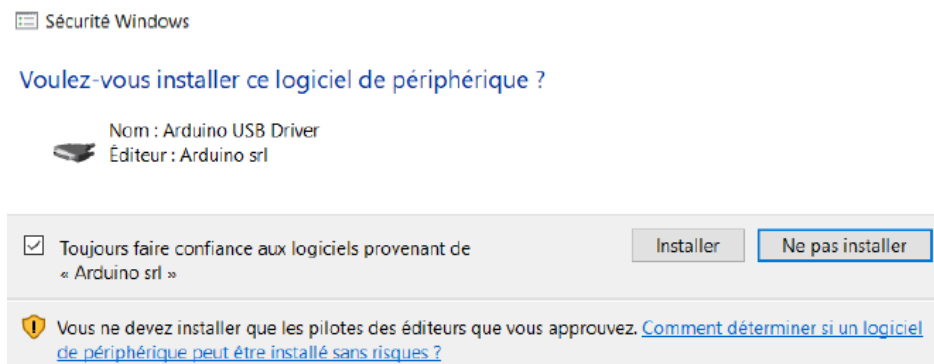


FIGURE 3 : INSTALLATION DES DRIVERS COMPLEMENTAIRES

1.2. Configuration de Arduino IDE

Maintenant que le logiciel est correctement installé, il nous faut le configurer en fonction de la carte que nous souhaitons programmer, dans notre cas une WT32-SC01 Plus qui est une carte utilisant un [microprocesseur ESP32](#). Il nous faut donc installer l'extension correspondante aux ESP32, pour cela, entrez dans le Boards manager puis recherchez « esp32 », il vous suffit alors de télécharger « esp32 by Espressif Systems », nous utilisons la version 3.1.1 mais nous conseillons d'utiliser la dernière version comme indiqué ci-dessous :

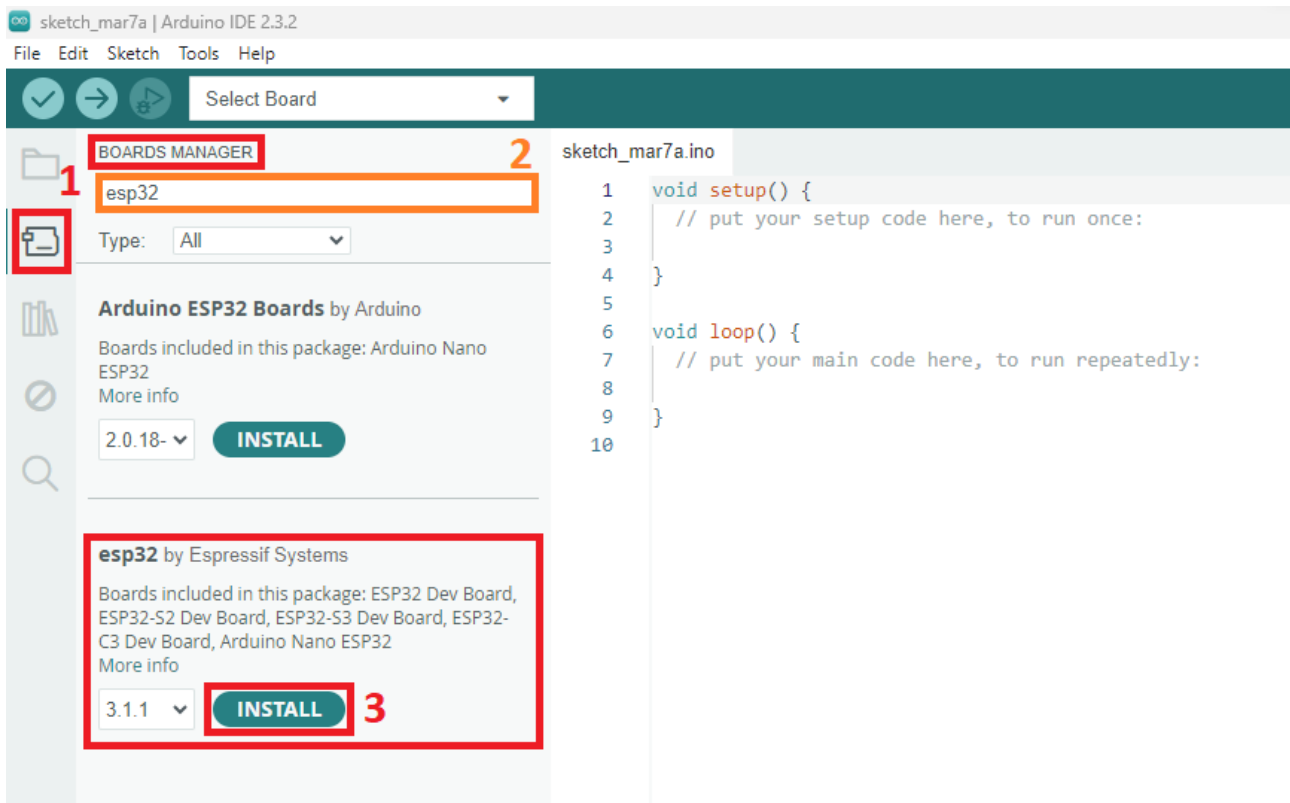


FIGURE 4 : CONFIGURATION DU BOARDS MANAGER

Attention, cette étape peut être un peu longue en fonction de votre connexion internet, attendez bien le message de fin d'installation, dont voici un exemple :

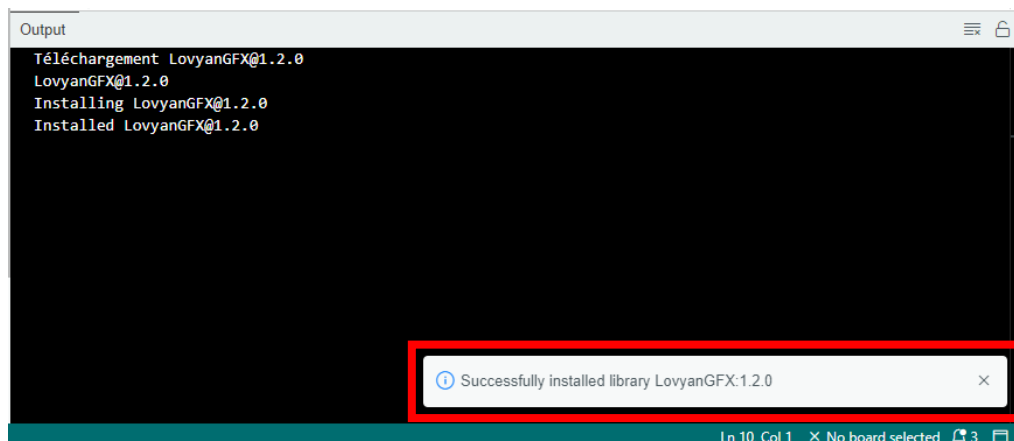


FIGURE 5 : MESSAGE DE FIN D'INSTALLATION

Maintenant que l'extension est correctement installée, nous pouvons configurer le port sur lequel nous allons brancher la carte, cela va permettre à Arduino IDE de la reconnaître et de correctement la traiter. Pour ce faire, commencez par brancher la carte à l'un des ports USB de votre ordinateur, à ce stade, la LED bleue devrait s'allumer. Il est possible qu'un programme de base soit déjà installé sur la carte, si ce n'est pas le cas l'écran restera simplement noir, d'où l'utilité d'observer la LED :

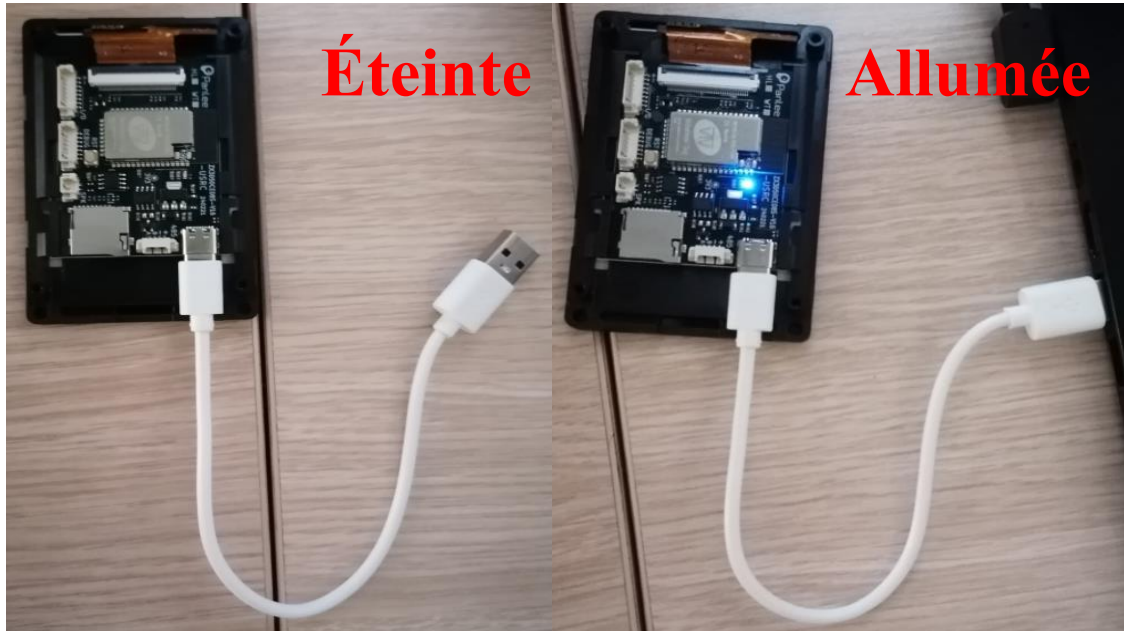


FIGURE 6 : BRANCHEMENT DE LA CARTE

Nous pouvons maintenant revenir sur l'application et accéder à la configuration du port comme indiqué ci-dessous :

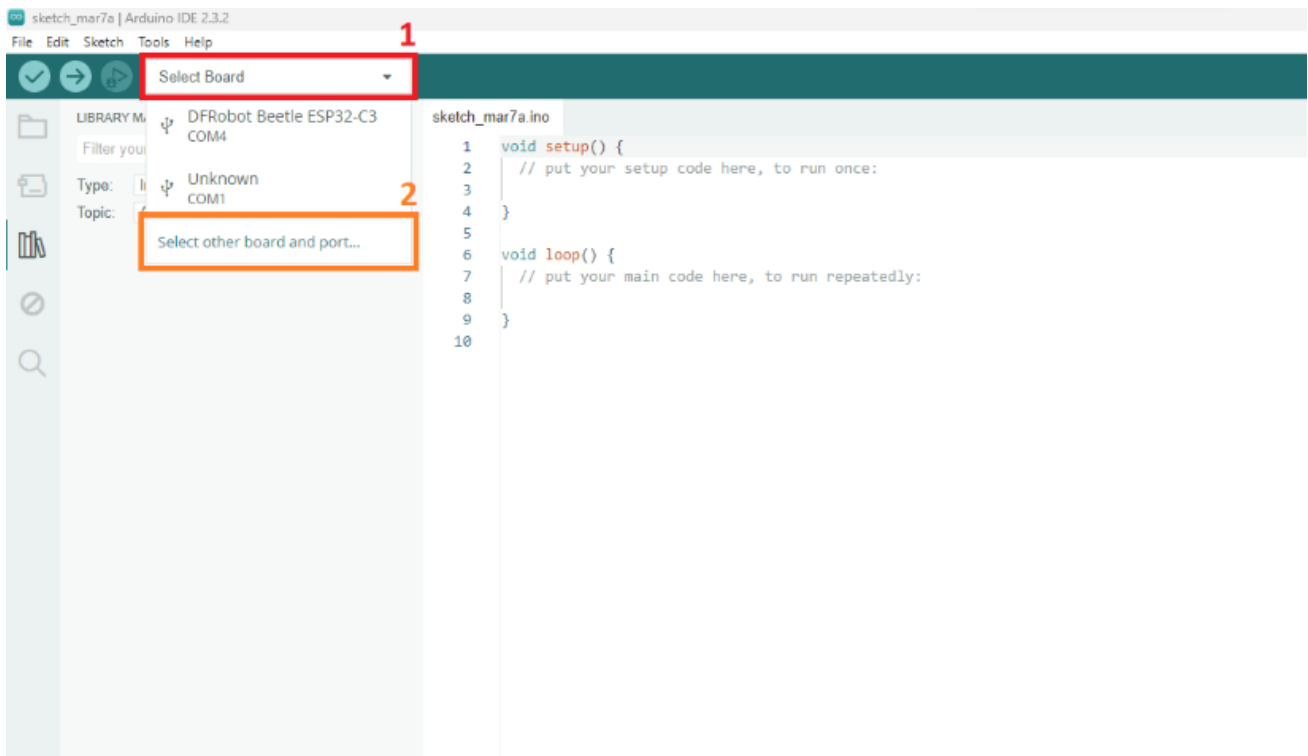


FIGURE 7 : CONFIGURATION DU PORT USB

Une fois dans le menu, il suffit de trouver votre carte dans la barre de recherche, dans notre cas WT32-SC01 Plus puis de sélectionner le bon port. Pour déterminer le port sur lequel est branché la carte, vous pouvez la débrancher avant de la rebrancher pour voir apparaître le port concerné comme indiqué ci-dessous :

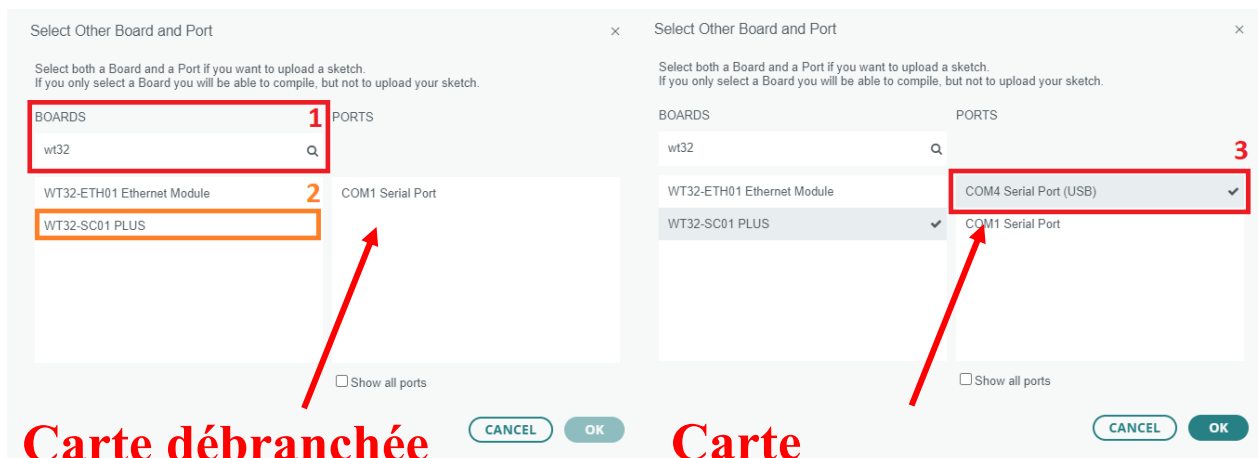


FIGURE 8 : CONFIGURATION DU PORT USB SUITE

Vous pouvez alors valider en appuyant sur « OK » et vous devriez voir apparaître le nom de la carte dans la sélection du port.

Pour finir, il nous reste simplement à installer la bibliothèque permettant d'exploiter l'écran de notre carte, encore une fois, il en existe plusieurs possibles (AdafruitGFX, LovyanGFX, TFT_eSPI, Arduino_GFX) mais nous avons décidé d'utiliser LovyanGFX car elle supporte l'écran du WT32-SC01 Plus et elle permet une bonne gestion des images, couleurs et tactile bien qu'elle soit un peu complexe à configurer.

Pour installer LovyanGFX, il vous suffit de vous rendre dans le « Library Manager » où vous pourrez rechercher « lovyanGFX » avant d'installer LovyanGFX by lovyano3. Nous utilisons la version 1.2.0 pour nos programmes, mais une version plus récente devrait également fonctionner :



FIGURE 9 : INSTALLATION DE LA BIBLIOTHEQUE LOVYANGFX

Bien sûr, il existe de nombreuses autres bibliothèques qui pourraient être utiles pour notre carte comme Wifi.h pour la connexion wifi, mais dans ce tutoriel nous nous concentrerons sur un affichage basique d'une information sur l'écran.

2. Programmation

2.1. Utilisation du moniteur série

Commençons par vérifier que nous avons bien configuré Arduino IDE, pour ce faire, nous pouvons utiliser le moniteur série. Il s'agit d'un outil directement intégré dans l'IDE Arduino (ou d'autres environnements) qui permet d'échanger des messages texte entre carte et ordinateur via le port USB. Ce procédé est très pratique pour vérifier la bonne exécution d'un programme, on pourrait le comparer à un print dans d'autres langages.

Ici, nous l'utiliserons pour vérifier que la carte est bien reconnue et que l'on peut communiquer avec. Commençons par ouvrir le moniteur série qui apparaîtra comme un onglet avec la console :

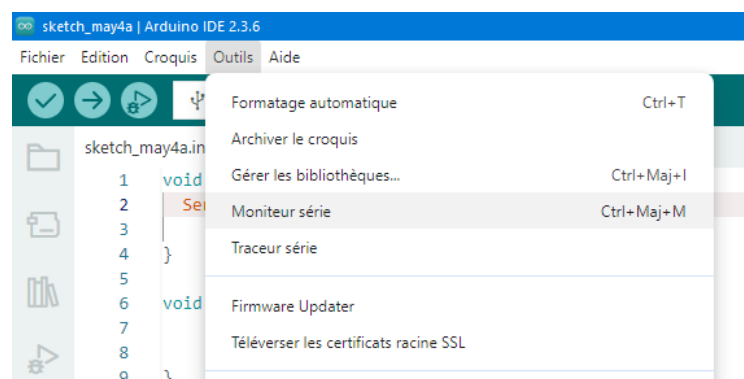


FIGURE 10 : OUVERTURE DU MONITEUR SERIE

Vous devriez alors avoir un nouvel onglet à côté de la console (sortie) appelé moniteur série :

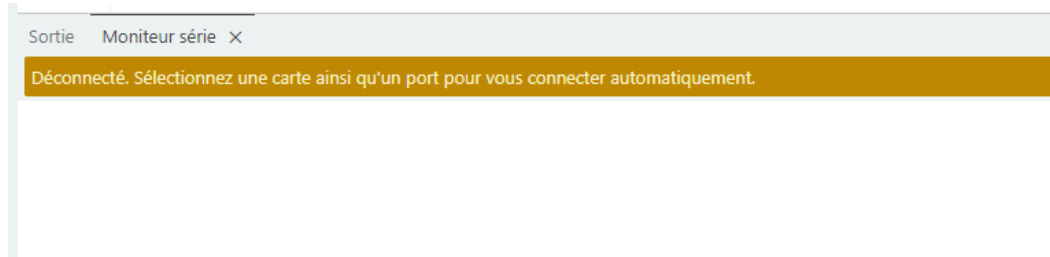


FIGURE 11: ONGLET MONITEUR SERIE

Nous pouvons observer un message en orange, ce dernier apparaît lorsque la carte est débranchée. Une fois la carte branchée, le message devrait disparaître.

À droite de l'onglet moniteur série, nous avons plusieurs options :

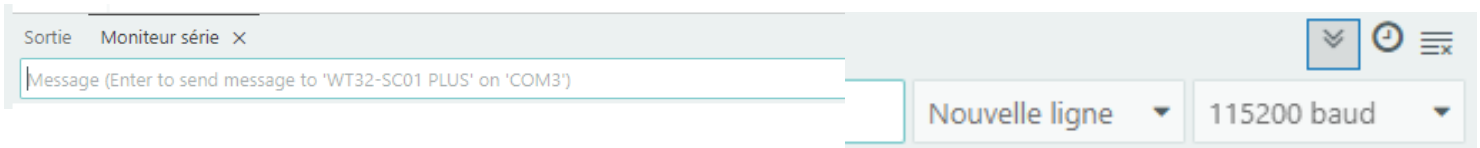


FIGURE 12 : OPTIONS DU MONITEUR SERIE

L'option qui va nous intéresser ici est le menu déroulé 115 200 baud. Il s'agit du débit auquel la communication série va s'effectuer (ici 115 200 bits par seconde). Ce paramètre devrait être sur cette valeur par défaut, si ce n'est pas le cas, vous pouvez la modifier.

Nous pouvons maintenant réaliser un programme d'exemple simple qui va renvoyer des messages au moniteur série. Dans Arduino IDE, le programme est divisé en deux parties, le setup ou initialisation dont le contenu ne sera exécuté qu'une seule fois au lancement du programme et la loop ou boucle dont le contenu sera exécuté en boucle, nous reviendrons plus en détails sur ce sujet dans la partie 2.3 :

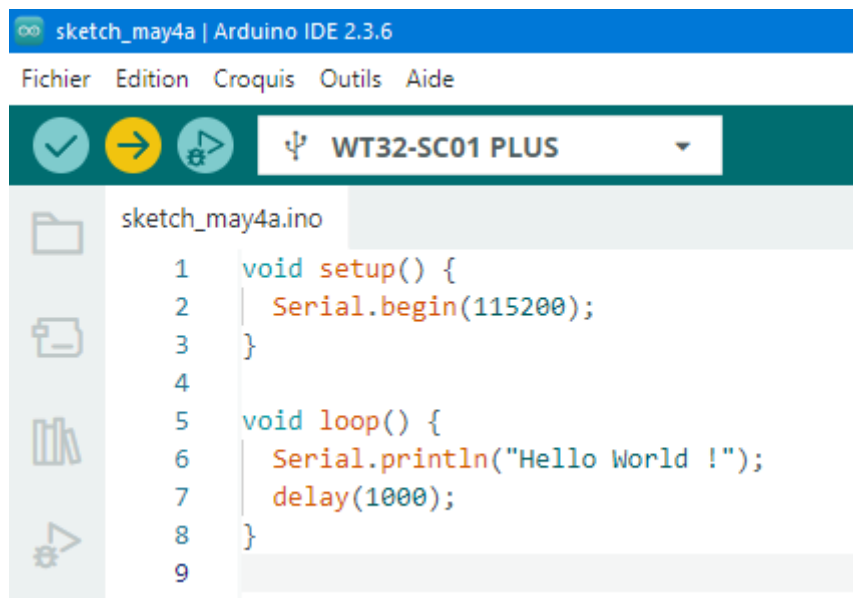


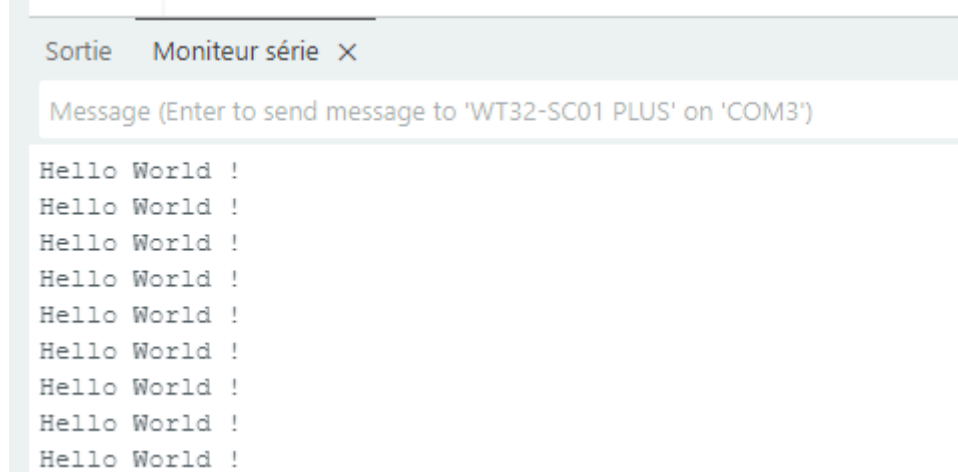
FIGURE 13 : PROGRAMME BASIQUE POUR LE MONITEUR SERIE

Comme dit plus tôt, le programme est divisé en deux parties, dans le setup, nous démarrons la communication avec le moniteur série avec *Serial.begin()* en précisant la valeur 115 200 correspondant à l'option vu précédemment.

Nous pouvons alors réaliser un *Serial.print()* pour afficher une chaîne de caractère dans le moniteur série (la précision *ln* après le print permet simplement un retour à la ligne après l'impression). On ajoute ensuite un délai d'une seconde avec *delay()* (équivalent de sleep ou wait).

Nous pouvons alors téléverser le programme vers la carte à l'aide de la flèche indiquée en jaune en faisant bien attention que le port usb soit bien indiqué comme étant celui de la WT32-SC01 Plus.

Une fois le téléversement terminé, nous obtenons alors le résultat suivant :



```

Sortie  Moniteur série X
Message (Enter to send message to 'WT32-SC01 PLUS' on 'COM3')
Hello World !
Hello World !
Hello World !
Hello World !
Hello World !
Hello World !
Hello World !
Hello World !
Hello World !
Hello World !

```

FIGURE 14 : SORTIE DU MONITEUR SERIE

Nous avons bien le résultat escompté, la carte est donc bien en communication avec notre ordinateur. Nous pourrions ainsi réutiliser ce principe pour vérifier la bonne exécution d'un programme.

2.2. Configuration des pins

Maintenant que Arduino IDE est correctement installée, nous pouvons réaliser un programme basique permettant d'afficher un texte sur l'écran de la carte. Afin que notre programme puisse utiliser correctement l'écran de la carte, il nous faut d'abord déterminer les pins responsables de l'affichage, pour ce faire, il suffit de trouver leur numéro sur la [documentation de la carte](#) dans la partie « interface description » aux pages 5 et 6. Voici dans notre cas les pins dont nous nous servirons pour l'affichage :

[5] LCD Interface (Tab. 5)

Description	Module Pin	Remark
BL_PWM	GPIO 45	Backlight control, active high
LCD_RESET	GPIO 4	LCD reset, multiplexed with touch reset
LCD_RS	GPIO 0	Command/Data selection
LCD_WR	GPIO 47	Write clock
LCD_TE	GPIO 48	Frame sync
LCD_DB0	GPIO 9	LCD data interface, 8bit MCU (8080)
LCD_DB1	GPIO 46	
LCD_DB2	GPIO 3	
LCD_DB3	GPIO 8	
LCD_DB4	GPIO 18	
LCD_DB5	GPIO 17	
LCD_DB6	GPIO 16	
LCD_DB7	GPIO 15	

FIGURE 15 : TABLEAU DES PINS DE L'AFFICHAGE

Attention, ces numéros peuvent varier d'une carte à l'autre, il est donc nécessaire de vérifier directement dans la documentation.

2.3. Programme permettant de paramétrer la bibliothèque

Passons maintenant à la partie programmation, certaines parties du programme resteront toujours les mêmes étant donné qu'il s'agit de l'initialisation de la bibliothèque LovyanGFX choisie précédemment. Dans un premier temps, commençons par ajouter la bibliothèque au programme et créons la classe LGFX qui nous servira à paramétrer l'écran :

```
1  #include <LovyanGFX.hpp>
2
3
4  class LGFX : public lgfx::LGFX_Device {
5      lgfx::Panel_ST7796 _panel;
6      lgfx::Bus_Parallel8 _bus;
```

FIGURE 16 : AJOUT DE LA BIBLIOTHEQUE

Le panel représente l'écran physique que nous utilisons. Il contient toutes les informations spécifiques à l'écran, telles que la résolution, la profondeur de couleur, et les fonctions de contrôle de l'écran.

Le bus quant à lui, représente le moyen de communication entre le microcontrôleur et le panneau d'affichage. Il peut s'agir d'un bus parallèle, d'un bus SPI, d'un bus I2C, etc. Le bus est responsable de l'envoi et de la réception des données entre le microcontrôleur et le panneau.

Nous allons donc configurer le panel et le bus en fonction de notre carte (pour une autre carte, ces paramètres pourront être différents). Nous allons notamment préciser la fréquence d'écriture et les pins indispensables que nous avons vu précédemment pour le bus et la dimension de l'écran pour le panneau d'affichage :

```
4  class LGFX : public lgfx::LGFX_Device {
5      lgfx::Panel_ST7796 _panel;
6      lgfx::Bus_Parallel8 _bus;
7
8      public:
9      LGFX() {
10         // Configuration du bus parallèle 8 bits (mode 8080)
11         auto cfg = _bus.config();
12         cfg.freq_write = 20000000;
13         cfg.pin_wr = 47;
14         cfg.pin_rs = 0;
15         cfg.pin_d0 = 9; cfg.pin_d1 = 46; cfg.pin_d2 = 3; cfg.pin_d3 = 8;
16         cfg.pin_d4 = 18; cfg.pin_d5 = 17; cfg.pin_d6 = 16; cfg.pin_d7 = 15;
17         _bus.config(cfg);
18         _panel.setBus(&_bus);
19
20         // Configuration du panneau LCD ST7796
21         auto panel_cfg = _panel.config();
22         panel_cfg.pin_rst = 4;
23         panel_cfg.panel_width = 320;
24         panel_cfg.panel_height = 480;
25         _panel.config(panel_cfg);
26         setPanel(&_panel);
27     }
28 };
```

FIGURE 17 : CLASSE LGFX COMPLETEE AVEC PINS ET DIMENSIONS

Une fois cette paramétrisation terminée, nous pouvons écrire le cœur de notre programme. La façon dont fonctionne Arduino IDE est simple, comme expliqué plus tôt, les programmes sont divisés en deux parties, l'initialisation (setup) et la boucle principale (loop). Ces deux parties composent le programme complet, l'initialisation n'est exécutée qu'une fois là où la boucle principale restera active tant que la carte sera alimentée :

```

1  void setup() {
2
3      // Programme d'initialisation
4  }
5
6  void loop() {
7
8      // Programme de la boucle principale
9  }

```

FIGURE 18 : DIFFERENTES PARTIES DU PROGRAMME

2.4. Programme principal

Étant donné que nous souhaitons simplement afficher un texte, nous ne nous servons pas de la boucle principale dans cet exemple, mais elle est très utile pour réaliser des programmes interactifs.

Afin d'afficher un texte, il nous faut d'abord commencer par activer le rétroéclairage (encadré en bleu). Une fois cela fait, nous pouvons paramétrer comme nous le souhaitons notre écran en changeant le format (paysage/portrait), le mode des couleurs (RGB) etc. (encadré en vert). Finalement, nous pouvons nous servir des commandes fournies par la bibliothèque LovyanGFX afin de mettre l'écran en blanc et d'afficher un texte surligné en jaune (encadré en rouge) :

```

31  LGFX lcd;                                // Création d'un objet LCD pour contrôler l'écran via la bibliothèque LovyanGFX (LGFX paramétré plus tôt)
32
33  void setup() {
34
35      pinMode(45, OUTPUT);                  // Définit la broche 45 comme sortie
36      digitalWrite(45, HIGH);              // Activation du rétroéclairage
37
38      lcd.init();                           // Initialise l'écran LCD
39      lcd.setRotation(1);                   // Oriente l'écran en format paysage (de 0 à 4 pour 0°, 90°, 180° et 270°)
40      lcd.setColorDepth(16);               // Définit une profondeur de couleur de 16 bits
41      lcd.setSwapBytes(true);              // Inverse l'ordre des octets pour les images (utile selon le format utilisé)
42      lcd.invertDisplay(true);             // Inverse les couleurs de l'affichage car inversés par défaut (blanc devient noir, etc.)
43
44      lcd.fillScreen(TFT_WHITE);            // Remplit l'écran avec du blanc
45
46      lcd.setTextColor(TFT_BLACK, TFT_YELLOW); // Définit la couleur du texte (noir) et du fond du texte (jaune, effet surligné)
47      lcd.setTextSize(3);                 // Définit la taille du texte à 3 (plus grand que la normale)
48      lcd.setCursor(140, 140);            // Positionne le curseur d'écriture à la position (140, 140)
49      lcd.print("Hello World !");         // Affiche "Hello World !" à l'écran
50  }
51
52
53  void loop() {
54      // Vide car aucune action répétée n'est nécessaire
55  }

```

FIGURE 19 : PROGRAMME D'AFFICHAGE DE TEXTE

Maintenant que le programme est réalisé, nous pouvons le compiler et l'envoyer sur notre carte afin qu'il soit exécuté. Pour ce faire, une fois la carte branchée, il suffit de sélectionner le port correspondant et de téléverser le programme comme indiqué ci-dessous :



FIGURE 20 : TELEVERSEMENT DU PROGRAMME SUR LA CARTE

Après compilation et téléversement, nous devrions obtenir le résultat suivant sur la carte :



FIGURE 21 : RESULTAT D'EXECUTION DU PROGRAMME

À ce stade, vous connaissez les bases de la programmation de la carte WT32-SC01 Plus et pouvez maintenant expérimenter en modifiant la partie initialisation et boucle du programme. Vous pouvez utiliser de nombreuses bibliothèques afin de réaliser différents programmes. Attention toutefois, pour pouvoir utiliser certains aspects de l'écran comme le tactile par exemple, il est important de correctement configurer les pins adéquats (nous avons simplement configuré les pins d'affichage).

Vous pourrez trouver plus d'informations et d'exemples de programmes sur le Github du projet de contrôle de base holonome à ce [lien](#).

2.5. Commandes de bases pour LovyanGFX

Pour finir, voici une liste de quelques fonctions de base de la bibliothèque LovyanGFX vous permettant d'expérimenter (vous pourrez retrouver sa version compète sur le Github) :

3. Fonctions Principales

3.1 Initialisation

```
lcd.init();
```

3.2 Orientation

```
lcd.setRotation(0); // 0=Portrait, 1=Paysage, 2=Portrait inversé, 3=Paysage inversé
```

3.3 Effacer l'écran

```
lcd.fillScreen(TFT_BLACK);
```

3.4 Dessiner des formes

```
lcd.drawPixel(10, 10, TFT_RED);  
lcd.drawLine(0, 0, 100, 100, TFT_WHITE);  
lcd.drawRect(50, 50, 100, 50, TFT_GREEN);  
lcd.fillRect(10, 10, 50, 30, TFT_BLUE);  
lcd.drawCircle(120, 120, 40, TFT_YELLOW);  
lcd.fillCircle(200, 200, 30, TFT_CYAN);
```

3.5 Affichage de texte

```
lcd.setTextColor(TFT_WHITE, TFT_BLACK);  
lcd.setTextSize(2);  
lcd.setCursor(50, 50);  
lcd.print("Bonjour !");
```

3.6 Gestion des images

Charger une image BMP depuis une carte SD

```
lcd.drawBmp("/image.bmp", 0, 0);
```

FIGURE 22 : FONCTIONS DE BASE