

Interfaçage d'Écrans LCD avec ESP32

Houssem-eddine LAHMER

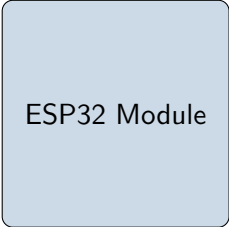
5 mai 2025

Plan du cours

- 1 Introduction aux ESP32 et Écrans LCD
- 2 Principes de Communication
- 3 Programmation et Bibliothèques
- 4 Applications Pratiques
- 5 Optimisation et Bonnes Pratiques

Qu'est-ce que l'ESP32 ?

- Microcontrôleur développé par Espressif Systems
- Successeur de l'ESP8266
- Fonctionnalités principales :
 - Double cœur Xtensa LX6 (jusqu'à 240 MHz)
 - Wi-Fi et Bluetooth intégrés
 - 520 Ko de SRAM
 - 34 GPIOs programmables
 - ADC, DAC, I²C, SPI, UART, etc.

A light blue rounded rectangle with a thin black border, containing the text "ESP32 Module" in a black sans-serif font.

ESP32 Module

- **LCD à caractères (comme HD44780)**
 - Affichage de texte basique
 - Interface parallèle (4 ou 8 bits)
 - Généralement 16x2 ou 20x4 caractères
- **LCD graphiques monochromes**
 - ST7920, KS0108, etc.
 - Résolution plus élevée (128x64 pixels)
- **LCD TFT couleur**
 - SPI : ILI9341, ST7735, etc.
 - I²C : SSD1306 (OLED)
 - Résolutions variées (240x320, 128x160, etc.)

1. Interface Parallèle

- Utilisée avec les LCD à caractères
- Nécessite plusieurs broches GPIO
- Communication directe et simple

3. Interface I²C

- Seulement 2 fils : SDA et SCL
- Économie de broches GPIO
- Plus lent que SPI
- Utilisé avec certains OLED

2. Interface SPI

- Serial Peripheral Interface
- Généralement 4 fils : MOSI, MISO, CLK, CS
- Plus rapide que I²C
- Utilisée avec TFT et certains OLED

4. Communication UART

- Certains modules intelligents
- Utilise RX/TX
- Simplification du code

Brochage et Connexions Typiques



- **Arduino IDE**

- Facile à utiliser
- Grande communauté
- Nombreuses bibliothèques disponibles

- **ESP-IDF (Espressif IoT Development Framework)**

- Framework officiel
- Plus de contrôle, performances optimisées
- Courbe d'apprentissage plus raide

- **PlatformIO**

- Extension pour VSCode
- Gestion des dépendances simplifiée
- Support multi-framework

- **LiquidCrystal_I2C** - Pour LCD à caractères via I²C
- **Adafruit GFX et Adafruit SSD1306** - Pour écrans OLED
- **TFT_eSPI** - Pour écrans TFT couleur
- **LVGL (Light and Versatile Graphics Library)**
 - Bibliothèque graphique avancée
 - Widgets, animations, tactile

Exemple : LCD à Caractères I²C

```
1 #include <Wire.h>
2 #include <LiquidCrystal_I2C.h>
3
4 // Initialisation du LCD      l'adresse 0x27, 16 colonnes et 2
    lignes
5 LiquidCrystal_I2C lcd(0x27, 16, 2);
6
7 void setup() {
8     lcd.init();                // Initialisation du LCD
9     lcd.backlight();           // Allumer le r tro clairage
10
11     // Afficher un message
12     lcd.setCursor(0, 0);       // Positionner le curseur (
        colonne, ligne)
13     lcd.print("Hello, ESP32!");
14     lcd.setCursor(0, 1);
15     lcd.print("LCD I2C Test");
16 }
17
18 void loop() {
19     // Animation simple
```

Exemple : Écran TFT Couleur (SPI)

```
1 #include <SPI.h>
2 #include <TFT_eSPI.h>
3
4 // Configuration dans User_Setup.h
5 TFT_eSPI tft = TFT_eSPI();
6
7 void setup() {
8     tft.init();
9     tft.setRotation(1); // Rotation (0-3)
10    tft.fillScreen(TFT_BLACK);
11
12    // Afficher du texte
13    tft.setCursor(0, 0, 2); // x, y, police
14    tft.setTextColor(TFT_WHITE, TFT_BLACK);
15    tft.println("ESP32 TFT Test");
16
17    // Dessiner des formes
18    tft.drawRect(10, 30, 100, 50, TFT_RED);
19    tft.fillCircle(150, 50, 30, TFT_BLUE);
20 }
21
```

Applications simples

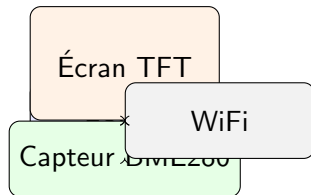
- Affichage de capteurs (température, humidité)
- Horloge et minuteur
- Affichage de statut (WiFi, batterie)
- Menus de navigation simples

Applications avancées

- Interface utilisateur graphique
- Contrôle domotique
- Affichage de données IoT
- Moniteurs de systèmes
- Mini-jeux

Exemple : Station Météo

- Capteur BME280 (température, humidité, pression)
- Écran TFT pour affichage
- ESP32 pour traitement et WiFi
- API météo pour prévisions
- Stockage de données sur carte SD



Avantages :

- Visualisation locale et à distance
- Historique des données
- Interface graphique intuitive

- **Mémoire tampon double**
 - Préparer l'image en mémoire avant affichage
 - Réduire le scintillement
- **Mise à jour partielle**
 - Rafraîchir uniquement les zones modifiées
 - Économiser du temps de traitement
- **Compression de données**
 - Stockage efficace des images et polices
 - Formats spécifiques (RLE, sprites)
- **DMA (Direct Memory Access)**
 - Transfert sans CPU pour SPI
 - Libère le processeur pour d'autres tâches

Stratégies d'économie d'énergie :

- Réduire la luminosité du rétroéclairage
- Utiliser le mode veille de l'écran
- Rafraîchir l'écran moins fréquemment
- Utiliser le mode deep sleep de l'ESP32
- Choisir des écrans à faible consommation (OLED vs LCD)

Exemple de mise en veille

```
1 // teindre l'cran et mettre l'ESP32 en veille profonde
2 display.powerDown(); // Pour certains crans
3 esp_sleep_enable_timer_wakeup(10 * 1000000); // 10 secondes
4 esp_deep_sleep_start();
```

• Structurer le code

- Séparer l'interface graphique de la logique
- Créer des fonctions dédiées pour chaque écran

• Gestion des ressources

- Utiliser des images et polices optimisées
- Convertir les ressources avec des outils dédiés

• Interface utilisateur

- Simplicité et cohérence
- Contraste suffisant
- Texte lisible (taille adaptée)

• Robustesse

- Gérer les erreurs de communication
- Programmer des timeouts

• Documentation

- Documentation ESP32 : <https://docs.espressif.com/>
- Adafruit Learn : <https://learn.adafruit.com/>
- Documentation TFT_eSPI : https://github.com/Bodmer/TFT_eSPI

• Tutoriels recommandés

- Random Nerd Tutorials
- Instructables - Projets ESP32 avec écrans

• Forums et communautés

- Forum ESP32
- Stack Exchange
- Reddit r/esp32

Merci de votre attention !

Questions ?