BÁO CÁO CHI TIẾT CODE VÀ CÁCH TRIỂN KHAI
Module quản lý cảm biến đa năng (MSMS)

Mục lục

Chương 1: Tổng quan cấu trúc và các module firmware

1.1 Tổng quan cấu trúc mã nguồn

Firmware MSMS được tổ chức theo mô hình **component** của ESP-IDF: mã nguồn nằm trong `main/` (điểm vào và khởi tạo toàn cục) và `component/` (các module chức năng độc lập, mỗi component có thể có `.c`, `.h` và `CMakeLists.txt` riêng).

1.2 Cây thư mục và file nguồn

- **main/**
  - `main.c`, `main.h` – Điểm vào `app_main()`, khởi tạo GPIO (trigger/echo/analog), NVS, LED, nút bấm, I2C, màn hình OLED, menu và các task.
  - `CMakeLists.txt`, `Kconfig.projbuild` – Cấu hình build và pin (UART, SPI, IO port) theo từng target ESP32/ESP32-C6.
- **component/core/**
  - **DataManager/** – `Common.c`, `Common.h`: Định nghĩa các kiểu dữ liệu dùng chung (`DataManager_t`, `menu_item_t`, `menu_list_t`, `objectInfoManager_t`, `SelectionParam_t`, ...). (Các kiểu khác xem `Common.h`.)
  - **FunctionManager/** – `FunctionManager.c`, `FunctionManager.h`: Triển khai các callback của menu (cấu hình Wi-Fi, chọn cảm biến, reset port, trạng thái pin, điều khiển actuator ON/OFF).
  - **BatteryManager/** – `BatteryManager.c`, `BatteryManager.h`: Đọc ADC pin, cập nhật mức pin và thông tin hiển thị.
- **component/sensors/**
  - **SensorTypes/** – `SensorTypes.h`: Định nghĩa `SensorType_t`, `PortId_t`, `TypeCommunication_t`, `SensorData_t`, `sensor_driver_t`. (Chi tiết ở mục 1.6.1.)
  - **SensorRegistry/** – `SensorRegistry.c`, `SensorRegistry.h`: Bảng đăng ký driver cảm biến (BME280, MH-Z14A, PMS7003, DHT22, MQ-2 đến MQ-135), lọc theo giao tiếp (UART, I2C, SPI, ANALOG, PULSE).
  - **SensorConfig/** – `SensorConfig.c`, `SensorConfig.h`: Khởi tạo/đọc/giải phóng cảm biến (wrapper BME280 và mở rộng sau).
- **component/ui/**
  - **MenuSystem/** – `MenuSystem.c`, `MenuSystem.h`: Cây menu Root → WiFi Config, Sensors, Actuators, Battery Status, Information; Sensors → Port 1/2/3 → UART/I2C/SPI/ANALOG/PULSE → danh sách cảm biến; task `MenuNavigation_Task`.

- **ScreenManager/** – `ScreenManager.c`, `ScreenManager.h`: Vẽ menu lên OLED (`MenuRender`), màn hình Wi-Fi đang kết nối, tin nhắn, dữ liệu cảm biến; dùng mutex để tránh tranh chấp I2C.
- **ButtonManager/** – `ButtonManager.c`, `ButtonManager.h`: Đọc 4 nút (UP, DOWN, SEL, BACK), debounce, trả về `ReadButtonStatus()`.

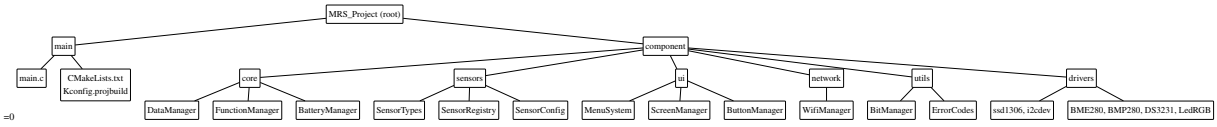
- **component/network/**
  - **WifiManager/** – `WifiManager.c`, `WifiManager.h`: Khởi tạo Wi-Fi AP/STA, captive portal cấu hình SSID/password, cập nhật trạng thái kết nối.

- **component/utils/**
  - **BitManager/** – `BitManager.c`, `BitManager.h`: Tiện ích bit/image cho menu.
  - **ErrorCodes/** – `ErrorCodes.c`, `ErrorCodes.h`: Mã lỗi hệ thống (`system_err_t`) và chuỗi mô tả.

- **component/drivers/**
  - **ssd1306/** – Driver màn hình OLED I2C (SSD1306).
  - **i2cdev/** – Lớp I2C dùng chung (mutex, timeout), cấu hình qua Kconfig (SDA/SCL theo ESP32/ESP32-C6).
  - **BME280/**, **BMP280/** – Driver cảm biến nhiệt độ/độ ẩm/áp suất.
  - **DS3231/**, **DS3231Time/** – RTC.
  - **LedRGB/** – LED RGB.

1.3 Sơ đồ cây thư mục dự án

Hình 1 minh họa cấu trúc thư mục chính của dự án (main + component và các component con).



Hình 1: Cây thư mục dự án (main và component).

1.4 Điểm vào và luồng khởi tạo

Điểm vào duy nhất của firmware là `app_main()` trong `main/main.c`. Thứ tự khởi tạo:

1. **GPIO (InitGPIO):** Cấu hình chân trigger (output), echo (input) và các chân analog (ADC) theo `Kconfig` (`CONFIG_IO_1_PORT_2`, `CONFIG_IO_2_PORT_2`, `CONFIG_IO_1_PORT_1`, `CONFIG_IO_3_PORT_2`, `CONFIG_IO_1_PORT_3`). Trên ESP32 dùng ADC1 (GPIO 32, 33, 35, ...); trên ESP32-C6 chỉ cấu hình GPIO input cho các chân analog (ADC dùng API mới sau).

2. **NVS:** `nvs_flash_init()`; nếu cần thì xóa và init lại.

3. **LED RGB:** `LedRGB_Init()`.

4. **Nút bấm:** `ButtonManagerInit()`.

5. **I2C chung:** `i2cInitDevCommon()` (cấu hình SDA/SCL từ Kconfig).

6. **Màn hình OLED:** `ssd1306_create(I2C_NUM_0, ...)`, `ScreenManagerInit(&MainScreen)`.

7. **Dữ liệu Port:** Gán `DataManager.selectedSensor[i] = SENSOR_NONE` cho mọi port `i` trước khi gọi `MenuSystemInit`, để menu Sensors lúc khởi động hiển thị "Port 1", "Port 2", "Port 3" (tránh zero-init hiển thị nhầm "Port 1 - BME280").

8. **Menu:** `MenuSystemInit(&DataManager)` – gán `Data->screen.current = &Root_Menu`, khởi tạo menu Sensors theo Port 1/2/3 → giao tiếp → cảm biến và `selectedSensorName` (phục vụ callback, màn hình dữ liệu cảm biến).

9. **Task:** `xTaskCreate(wifi_init_sta, ...)`, `xTaskCreate(MenuNavigation_Task, ...)`. (BatteryManager init/task có thể bật lại nếu dùng.)

Sau đó vòng lặp chính chỉ `vTaskDelay`; toàn bộ tương tác người dùng và cập nhật màn hình diễn ra trong `MenuNavigation_Task` và các callback.

1.5 DataManager (Common) – Dữ liệu dùng chung

1.5.1 Vai trò

`Common.h` (và `Common.c` tối thiểu) là nơi định nghĩa các kiểu và hằng dùng chung cho toàn bộ firmware, đảm bảo UI, Sensor và Core dùng chung một mô hình dữ liệu.

1.5.2 Cấu trúc chính

- **Nút bấm:** `button_type_t` (BTN_UP, BTN_DOWN, BTN_SEL, BTN_BACK, BTN_NONE); `ButtonManager_t` (trạng thái và thời gian debounce).
- **Menu:** `menu_item_type_t` (MENU_ACTION, MENU_SUBMENU, ...); `menu_item_t` (name, type, callback, ctx, children); `menu_list_t` (items, text, image, count, object, parent, `port_index`). Trường `port_index` dùng cho cấu trúc menu (0..NUM_PORTS-1 cho menu từng port; $\geq$ NUM_PORTS cho menu hiển thị nhiều port, ví dụ Actuators).
- **Màn hình:** `ScreenManager_t` (current menu, selected index, prev_selected).
- **Thông tin đối tượng:** `objectInfoManager_t` (batteryInfo, wifiInfo, `selectedSensorName[NUM_PORTS]`).
  `selectedSensorName` do MenuSystem cập nhật từ `DataManager->selectedSensor`, dùng trong callback và màn hình dữ liệu cảm biến.

- **DataManager_t:** `sensor`, `button`, `screen`, `objectInfo`, `MenuReturn[10]`, `selectedSensor[NUM_PORTS]`; cùng hai callback: `on_sensor_selected(data, port)` (cập nhật tên menu, quay về menu Sensors), `on_ports_reset(data)` (đưa tên Port về "Port 1", "Port 2", "Port 3").
- **SelectionParam_t**, **ShowDataSensorParam_t:** Tham số truyền vào callback chọn cảm biến và hiển thị dữ liệu theo port.

Các GPIO nút bấm (BTN_UP_GPIO, . . .) và `MAX_VISIBLE_ITEMS` cũng được định nghĩa tại đây.

1.6 SensorTypes và SensorRegistry

1.6.1 SensorTypes.h

Định nghĩa:

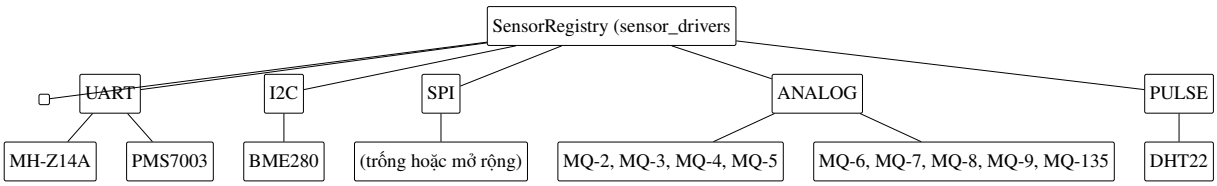
- `PortId_t` (PORT_1, PORT_2, PORT_3), `NUM_PORTS = 3`, `SensorType_t` (SENSOR_BME280, SENSOR_MHZ14A, . . .).
- `TypeCommunication_t`: COMMUNICATION_UART, I2C, SPI, ANALOG, PULSE (nhóm menu theo giao tiếp).
- `SensorData_t`: Mảng `data_fl`, `data_uint32`, . . . để lưu giá trị đọc được.
- `sensor_driver_t`: name, description, unit, unit_count, `interface` (TypeCommunication_t), is_init, con trỏ hàm init/read/deinit.

1.6.2 SensorRegistry

- **Mảng tĩnh** `sensor_drivers[]`: Mỗi phần tử là một `sensor_driver_t` tương ứng một loại cảm biến (BME280, MH-Z14A, PMS7003, DHT22, MQ-2 . . . MQ-135), kèm `interface` (I2C, UART, PULSE, ANALOG).
- **API:** `sensor_registry_get_drivers()`, `sensor_registry_get_count()`, `sensor_registry_get_driver(sensor_type)`, `sensor_registry_get_count_by_interface(iface)`, `sensor_registry_get_driver_at_interface(iface, index, out_sensor_type)`. Hai hàm cuối dùng để xây menu theo giao tiếp (UART, I2C, SPI, ANALOG, PULSE).

1.6.3 Sơ đồ cây cảm biến theo giao tiếp

Menu Sensors hiển thị cảm biến theo từng giao tiếp; nguồn dữ liệu là `sensor_drivers[]` lọc bởi `interface`. Hình 2 minh họa nhóm cảm biến theo `TypeCommunication_t`.



Hình 2: Cây cảm biến theo giao tiếp (interface). Menu Port 1/2/3 → UART/I2C/... lấy danh sách từ `sensor_registry_get_count_by_interface` và `sensor_registry_get_driver_at_interface`.

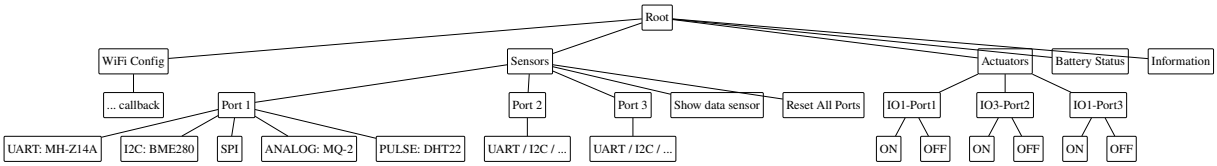
1.7 MenuSystem – Cấu trúc menu và điều hướng

1.7.1 Cây menu

- **Root:** WiFi Config, Sensors, Actuators, Battery Status, Information.
- **Sensors:** Port 1, Port 2, Port 3, Show data sensor, Reset All Ports. Mỗi Port 1/2/3 mở ra menu UART, I2C, SPI, ANALOG, PULSE; mỗi giao tiếp mở ra danh sách cảm biến tương ứng (driver có `interface` trùng). Chọn cảm biến → gọi `select_sensor_cb`: chỉ khi driver init thành công mới gán `Data->selectedSensor[port]`; nếu thất bại thì `selectedSensor[port] = SENSOR_NONE`, menu hiển thị "Port X". Callback `on_sensor_selected` (MenuSystem) cập nhật `Sensor_Menu_Items[0..2].name` và quay màn hình về menu Sensors, highlight port vừa chọn.
- **Actuators:** IO1-Port1, IO3-Port2, IO1-Port3. Mỗi mục có submenu **ON / OFF** để gọi `actuator_on_cb` / `actuator_off_cb` (FunctionManager) đặt mức GPIO tương ứng (CONFIG_IO_1_PORT_1, CONFIG_IO_3_PORT_2, CONFIG_IO_1_PORT_3).
- **WiFi Config / Battery Status:** Callback do FunctionManager đảm nhiệm.
- **Information:** `information_callback` (FunctionManager) gọi `ScreenShowInformation` hiển thị thông tin project (MRS Project, module quản lý cảm biến đa năng, tác giả MrKoi); sau vài giây tự vẽ lại Root menu.

1.7.2 Sơ đồ cây menu (Menu Tree)

Hình 3 mô tả cây menu từ Root đến các mục lá (cảm biến theo giao tiếp, ON/OFF của actuator).



Hình 3: Cây menu: Root → Sensors (Port → giao tiếp → cảm biến), Actuators (chân → ON/OFF), và các mục khác.

1.7.3 Khởi tạo menu động theo giao tiếp

Hàm `init_sensor_interface_port_menu_items(Data)`:

- Với mỗi cặp (port, interface): lấy `count = sensor_registry_get_count_by_interface(iface)`, cấp phát `SelectionParam_t` và `menu_item_t` cho từng cảm biến, điền `PortInterfaceMenus[port][i]`, `PortMenus[port].port_index = port`.
- Gán `PortMenus[port].items` = [UART, I2C, SPI, ANALOG, PULSE]; `Sensor_Menu_Items` = [Port 1, Port 2, Port 3, Show data sensor, Reset All Ports].

1.7.4 Cập nhật tên Port và quay về menu Sensors

MenuSystem dùng hàm nội bộ `update_port_names(Data)`: dựa trên `data->selectedSensor[0..2]` gán `Sensor_Menu_Items[i].name` là "Port $i+1$" nếu `selectedSensor[i] == SENSOR_NONE`, hoặc "Port $i+1$ - tên cảm biến" nếu đã chọn và init thành công (FunctionManager chỉ gán `selectedSensor[port]` trong trường hợp đó).

Hàm `go_to_sensor_menu(Data, selected_index)` đặt `data->screen.current = &Sensor_Menu`, `data->screen.selected = selected_index` và gọi `MenuRender`. Trong `MenuSystemInit`, DataManager được gán `on_sensor_selected` và `on_ports_reset`; `Root_Items[4].ctx = Data` để `information_callback` nhận DataManager. Khi người dùng chọn xong cảm biến, FunctionManager gọi `on_sensor_selected` $\rightarrow$ MenuSystem cập nhật tên và quay về menu Sensors; khi chọn "Reset All Ports", FunctionManager gọi `on_ports_reset` $\rightarrow$ MenuSystem đưa tên ba mục Port về "Port 1", "Port 2", "Port 3".

1.7.5 Task điều hướng

`MenuNavigation_Task` lặp vô hạn: đọc `ReadButtonStatus()`; nếu UP/DOWN thì thay đổi `selected` và gọi `MenuRender`; nếu SEL thì gọi callback của mục đang chọn hoặc chuyển vào `children`; nếu BACK thì chuyển về `parent`.

Trước mỗi lần xử lý, cập nhật `data->objectInfo.selectedSensorName[p] = sensor_type_to_name(data->selectedSensor[p])` cho mọi port `p` (phục vụ callback và màn hình dữ liệu cảm biến).

1.8 ScreenManager – Hiển thị OLED

1.8.1 Chức năng chính

- `ScreenManagerInit(&MainScreen)`: Lưu handle OLED, tạo mutex `oled_mutex`, gọi `initUIState()` (màn hình splash).
- `MenuRender(menu, selected, objectInfo)`: Lấy mutex; nếu menu có image/text (WiFi, Battery) thì vẽ bitmap và text theo `objectInfo`; sau đó vẽ danh sách `menu->items` với phân trang (MAX_VISIBLE_ITEMS); `ssd1306_refresh_gram`; trả mutex.
- `ScreenShowInformation(lines, n_lines)`: Hiển thị nhiều dòng text lên OLED (mỗi dòng cách nhau 12 px), dùng cho màn **Information** (thông tin project, tác giả MrKoi).

Dùng chung mutex và ssd1306_refresh_gram.

- ScreenWifiConnecting, ScreenShowMessage, ScreenShowDataSensor: Các màn hình đặc biệt khác, đều dùng chung mutex và ssd1306_refresh_gram.

1.8.2 Mutex và timeout I2C

Để tránh lỗi ESP_ERR_TIMEOUT khi nhiều task cùng dùng I2C (OLED và cảm biến): (1) Trong driver SSD1306 tăng timeout i2c_master_cmd_begin lên 2000 ms; (2) Trong ScreenManager mọi thao tác vẽ/refresh đều nằm trong xSemaphoreTake(oled_mutex) / xSemaphoreGive(oled_mutex).

1.9 FunctionManager – Callback menu

FunctionManager triển khai các callback được gán vào menu_item_t:

- **wifi_config_callback:** Tạo task wifi_config_task (WifiManager AP, cập nhật trạng thái, ScreenWifiConnecting, khi kết nối xong thì MenuRender(MenuReturn[0]) và xóa task).
- **select_sensor_cb:** Nhận SelectionParam_t (data, port, sensor). Chỉ gán data->selectedSensor[port] = sensor khi driver init thành công (hoặc port đã chọn/sensor đã init); nếu không có init hoặc init thất bại thì gán selectedSensor[port] = SENSOR_NONE để menu hiển thị "Port X". Sau đó gọi data->on_sensor_selected(data, port) (MenuSystem) để cập nhật tên Sensor_Menu_Items[0..2] và quay màn hình về menu Sensors, highlight port vừa chọn.
- **show_data_sensor_cb:** Bật flag hiển thị màn hình dữ liệu cảm biến cho port tương ứng.
- **reset_all_ports_callback:** Đặt selectedSensor[i] = SENSOR_NONE cho mọi port i; gọi data->on_ports_reset(data) để MenuSystem đưa Sensor_Menu_Items[0..2].name về "Port 1", "Port 2", "Port 3"; dọn task đọc cảm biến.
- **battery_status_callback:** BatteryManager_UpdateInfo(&objectInfo->batteryInfo); MenuRender(MenuReturn[1], ...).
- **information_callback:** Nhận ctx = DataManager. Gọi ScreenShowInformation với các dòng (MRS Project, module quản lý cảm biến đa năng, tác giả MrKoi); delay 4 s; MenuRender lại Root menu.
- **actuator_on_cb / actuator_off_cb:** Nhận ctx = số GPIO; cấu hình GPIO output và gpio_set_level 1 hoặc 0.

1.10 Cấu hình pin và Kconfig

1.10.1 main/Kconfig.projbuild

Các option pin được chọn theo target:

- **ESP32:** TX=17, RX=16; MISO=19, SCK=18, MOSI=23, CS=4; IO_1_PORT_1=32, IO_1_PORT_2=26, IO_2_PORT_2=27, IO_3_PORT_2=35, IO_1_PORT_3=33.
- **ESP32-C6:** TX=5, RX=4; MOSI=2, MISO=3, SCK=23, CS=22; IO_1_PORT_1=0, IO_1_PORT_2=1, IO_2_PORT_2=10, IO_3_PORT_2=11, IO_1_PORT_3=18.

1.10.2 component/drivers/i2cdev/Kconfig.projbuild

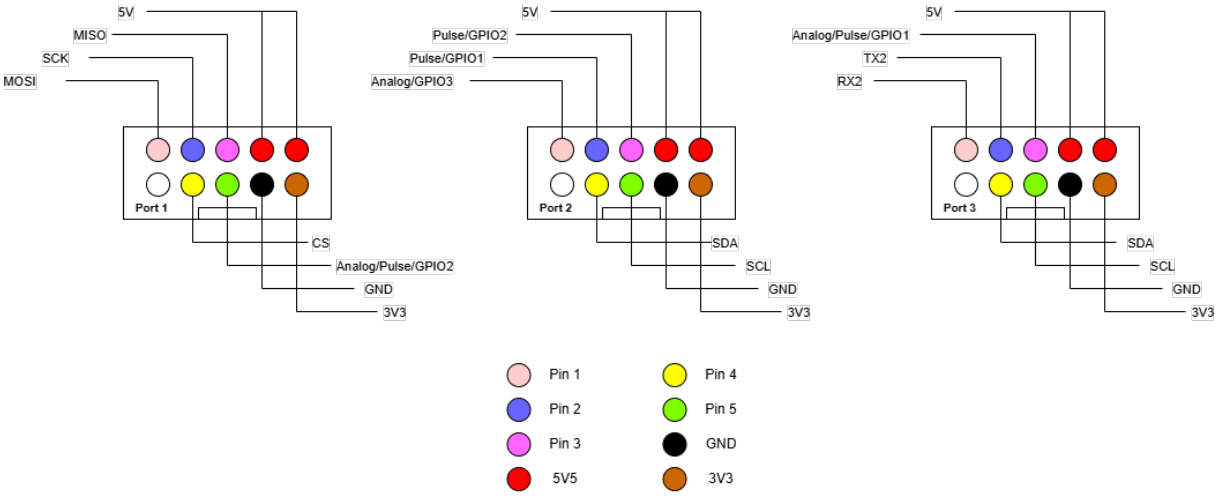
SDA/SCL cho I2C chung:

- **ESP32:** SDA=21, SCL=22.
- **ESP32-C6:** SDA=6, SCL=7.

Các driver BME280, DS3231 có thể có `Kconfig.projbuild` riêng (trong dự án có thể bị comment để dùng chung I2C từ main).

1.10.3 Cấu hình và tác dụng của chân trên từng Port

Hình 4 mô tả sơ đồ chân (pinout) của Port 1, Port 2 và Port 3: mỗi port có 10 chân với nguồn 5V, 3V3, GND; Port 1 gồm MISO, SCK, MOSI, CS (SPI) và Analog/Pulse/GPIO; Port 2 gồm Pulse/GPIO, Analog/GPIO, SDA/SCL (I2C); Port 3 gồm TX2/RX2 (UART), Analog/Pulse/GPIO, SDA/SCL (I2C). Các chân được khai báo trong menuconfig và dùng trong firmware qua `CONFIG_*` tương ứng.



Hình 4: Cấu hình và tác dụng chân trên Port 1, Port 2, Port 3 (5V, 3V3, GND, SPI/I2C/UART/Analog/Pulse/GPIO).

Giải thích chân đa năng:

- **Analog/Pulse/GPIO:** Chân này có thể dùng để đọc input Analog, GPIO và output: PWM, GPIO.
- **Pulse/GPIO:** Chỉ là output và input GPIO, không có Analog. Hai chân Pulse/GPIO (trên sơ đồ) dùng cho giao tiếp Pulse 2 chân (DHT11, HC-SR04, . . .).

Chương 2: Triển khai build, chạy và hướng dẫn mở rộng

Lưu ý (menuconfig): Các chân GPIO (UART, SPI, I2C, IO port) và tùy chọn build được **khai báo trong menuconfig**: chạy `idf.py menuconfig`, vào menu "Pin config" (main) và "I2C common config" (driver i2cdev) để đặt SDA/SCL, TX/RX, IO_1_PORT_1, IO_3_PORT_2, ... theo đúng phần cứng và target (ESP32/ESP32-C6). Giá trị này được dùng trong `main.c` và các driver qua `CONFIG_*`.

2.1 Build

Listing 1: Build firmware với ESP-IDF

```
idf.py set-target esp32    # hoc esp32c6
idf.py build
```

Cấu hình pin và tùy chọn (I2C timeout, ...) qua `idf.py menuconfig` (menu "Pin config", "I2C common config").

2.2 Flash và monitor

Listing 2: Nạp firmware và mở serial

```
idf.py -p COMx flash monitor
```

2.3 Luồng dữ liệu điển hình

1. Người dùng mở Sensors → Port 1 → I2C → chọn BME280. `select_sensor_cb` gọi `driver->init()`; chỉ khi init thành công mới gán `selectedSensor[PORT_1] = SENSOR_BME280`, rồi gọi `on_sensor_selected` → màn hình quay về menu Sensors, mục đầu tiên hiển thị "Port 1 - BME280". Nếu init thất bại hoặc không có init, `selectedSensor[PORT_1]` giữ `SENSOR_NONE` nên menu vẫn hiển thị "Port 1".

2. Task đọc cảm biến gọi `sensor->read(&data)` định kỳ; có thể gọi `ScreenShowDataSensor` hoặc cập nhật `DataManager` để menu hiển thị dữ liệu.

3. `MenuNavigation_Task` cập nhật `objectInfo.selectedSensorName[p]` từ `selectedSensor[p]` cho mọi port, phục vụ màn hình "Show data sensor" và các callback.

4. Khi chọn "Reset All Ports", `on_ports_reset` được gọi → tên ba mục Port trong menu Sensors trở lại "Port 1", "Port 2", "Port 3".

5. Người dùng mở Actuators → IO1-Port1 → ON: `actuator_on_cb` đặt GPIO `CONFIG_IO_1_PORT_1` lên 1.

2.4 Hướng dẫn chi tiết cách thêm cảm biến mới

Để thêm một loại cảm biến mới vào firmware MSMS, làm lần lượt các bước dưới đây. **Thứ tự quan trọng:** triển khai `init/read/deinit` trước; sau đó mới đăng ký driver trong `SensorRegistry.c` (gán con trỏ hàm vào `sensor_drivers[]`). Sau khi hoàn tất, cảm biến sẽ tự xuất hiện trong menu Sensors → Port 1/2/3, nhánh UART / I2C / SPI / ANALOG / PULSE, tương ứng với `interface` đã gán (không cần sửa MenuSystem).

2.4.1 Bước 1: Thêm định danh cảm biến trong `SensorTypes.h`

Thêm một phần tử mới vào enum `SensorType_t`, đặt ngay trước `SENSOR_NONE` nếu có, hoặc theo thứ tự mong muốn (thứ tự enum phải trùng với thứ tự phần tử trong mảng `sensor_drivers[]` ở Bước 3).

Listing 3: Ví dụ thêm SENSOR_XYZ vào SensorTypes.h

```
// Trong SensorTypes.h, enum SensorType_t
typedef enum {
  SENSOR_NONE = -1,
  SENSOR_BME280 = 0,
  // ... c c sensor h i n c ...
  SENSOR_MQ135 = 12,
  SENSOR_XYZ = 13,   // th m m i
} SensorType_t;
```

2.4.2 Bước 2: Triển khai init/read/deinit

Triển khai trước các hàm `init`, `read`, `deinit` của cảm biến. Nếu cảm biến dùng driver riêng (ví dụ component `component/drivers/XYZ/`), triển khai `system_err_t xyzInitialize(void)`, `xyzReadData(SensorData_t *)`, `xyzDeinitialize(void)`. Nếu logic đọc/init nằm trong `SensorConfig` (như BME280), có thể gọi từ `SensorConfig`; khi đăng ký vào registry sẽ gán con trỏ tới các hàm đó. Đảm bảo `read` ghi dữ liệu vào `data->data_fl[]` hoặc `data->data_uint32[]` theo `unit_count` và `description` sẽ khai báo ở bước đăng ký.

2.4.3 Bước 3: Đăng ký driver trong `SensorRegistry.c`

Sau khi đã có `init/read/deinit`,
thêm một phần tử `sensor_driver_t` vào mảng `sensor_drivers[]` (trong `SensorRegistry.c`).

Các trường bắt buộc: `name`, `description`, `unit`, `unit_count`, `interface` (UART, I2C, SPI, ANALOG, PULSE), và các con trỏ hàm `init`, `read`, `deinit` trỏ tới các hàm đã triển khai ở Bước 2.

Listing 4:]Ví dụ thêm driver XYZ (I2C) vào sensor_drivers[]

```
// Trong SensorRegistry.c, m ng sensor_drivers[]
{
    .name = "XYZ",
    .init = xyzInitialize,
```

```
    .read = xyzReadData,
    .deinit = xyzDeinitialize,
    .description = {"Temp", "Humidity"},
    .unit = {"C", "%"},
    .unit_count = 2,
    .is_init = false,
    .interface = COMMUNICATION_I2C,
},
```

Lưu ý: Thứ tự phần tử trong `sensor_drivers[]` phải tương ứng với giá trị số trong `SensorType_t` (phần tử đầu tiên = 0, tiếp theo = 1, ...). Nếu thêm enum `SENSOR_XYZ = 13`, phần tử mới phải là phần tử thứ 14 trong mảng (index 13).

2.4.4 *Bước 4: Cập nhật `sensor_type_to_name()` trong `SensorRegistry.c`*

Hàm `sensor_type_to_name(SensorType_t t)` dùng để hiển thị tên cảm biến trong menu (vd trong Show data sensor, label port). Thêm một `case` cho `SENSOR_XYZ`:

Listing 5: Thêm case trong sensor_type_to_name()

```
const char *sensor_type_to_name(SensorType_t t) {
  switch (t) {
  // ... cac case hien co ...
  case SENSOR_XYZ:
    return "XYZ";
  case SENSOR_NONE:
    return "None";
  default:
    return "Unknown";
  }
}
```

2.4.5 *Bước 5: (Tùy chọn) Cấu hình Kconfig và GPIO*

Nếu cảm biến cần pin riêng (UART, SPI, ADC), thêm option trong `main/Kconfig.projbuild` hoặc `component/drivers/.../Kconfig.projbuild` và đọc trong `main.c` hoặc trong hàm `init` của driver. Cảm biến ANALOG thường dùng chân đã cấu hình sẵn (CONFIG_IO_*_PORT_*); cảm biến UART dùng UART port tương ứng (CONFIG_UART_TX/RX).

2.4.6 *Tóm tắt quy trình thêm cảm biến*

1. Thêm enum trong `SensorTypes.h` (`SensorType_t`).
2. Triển khai `init/read/deinit` (driver riêng hoặc SensorConfig).
3. Đăng ký driver: thêm phần tử `sensor_driver_t` vào `sensor_drivers[]` trong `SensorRegistry.c`, gán `interface` và con trỏ `init/read/deinit`.

4. Cập nhật `sensor_type_to_name()` trong `SensorRegistry.c`.

5. (Tùy chọn) Thêm Kconfig/GPIO nếu cần.

Sau các bước trên, menu Sensors → Port 1/2/3 → [giao tiếp tương ứng] sẽ tự hiển thị cảm biến mới nhờ `init_sensor_interface_port_menu_items()` đã xây menu động từ `sensor_registry_get_count_by_interface()` và `sensor_registry_get_driver_at_interface()`.