

# BÁO CÁO CHI TIẾT CODE VÀ CÁCH TRIỂN KHAI

## Module quản lý cảm biến đa năng (MSMS)

### Mục lục

<b>1</b>	<b>Tổng quan cấu trúc mã nguồn</b>	<b>3</b>
1.1	Cây thư mục và file nguồn . . . . .	3
1.2	Sơ đồ cây thư mục dự án . . . . .	4
1.3	Điểm vào và luồng khởi tạo . . . . .	4
<b>2</b>	<b>DataManager (Common) – Dữ liệu dùng chung</b>	<b>5</b>
2.1	Vai trò . . . . .	5
2.2	Cấu trúc chính . . . . .	5
<b>3</b>	<b>SensorTypes và SensorRegistry</b>	<b>6</b>
3.1	SensorTypes.h . . . . .	6
3.2	SensorRegistry . . . . .	6
3.3	Sơ đồ cây cảm biến theo giao tiếp . . . . .	6
<b>4</b>	<b>MenuSystem – Cấu trúc menu và điều hướng</b>	<b>6</b>
4.1	Cây menu . . . . .	6
4.2	Sơ đồ cây menu (Menu Tree) . . . . .	7
4.3	Khởi tạo menu động theo giao tiếp . . . . .	7
4.4	Task điều hướng . . . . .	7
<b>5</b>	<b>ScreenManager – Hiển thị OLED</b>	<b>7</b>
5.1	Chức năng chính . . . . .	7
5.2	Mutex và timeout I2C . . . . .	8
<b>6</b>	<b>FunctionManager – Callback menu</b>	<b>8</b>
<b>7</b>	<b>Cấu hình pin và Kconfig</b>	<b>8</b>
7.1	main/Kconfig.projbuild . . . . .	8
7.2	component/drivers/i2cdev/Kconfig.projbuild . . . . .	9

<b>8</b>	<b>Triển khai build và chạy</b>	<b>9</b>
8.1	Build . . . . .	9
8.2	Flash và monitor . . . . .	9
8.3	Luồng dữ liệu điển hình . . . . .	9
<b>9</b>	<b>Hướng dẫn chi tiết cách thêm cảm biến mới</b>	<b>9</b>
9.1	Bước 1: Thêm định danh cảm biến trong <code>SensorTypes.h</code> . . . . .	10
9.2	Bước 2: Đăng ký driver trong <code>SensorRegistry.c</code> . . . . .	10
9.3	Bước 3: Cập nhật <code>sensor_type_to_name()</code> trong <code>SensorRegistry.c</code> . . . . .	10
9.4	Bước 4: Triển khai <code>init/read/deinit</code> (nếu chưa có) . . . . .	11
9.5	Bước 5: (Tùy chọn) Cấu hình <code>Kconfig</code> và <code>GPIO</code> . . . . .	11
9.6	Tóm tắt quy trình thêm cảm biến . . . . .	11
<b>10</b>	<b>Tóm tắt</b>	<b>11</b>

# 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).

## 1.1 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`, ...).
- **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`.
- **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).

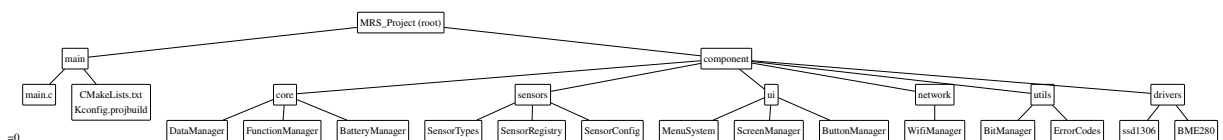
- **component/ui/**

- **MenuSystem/** – `MenuSystem.c`, `MenuSystem.h`: Cây menu (Root → WiFi Config, Sensors, Actuators, Battery Status, Information), cấu trúc Sensors → Port 1/2/3 → UART/I2C/SPI/ANALOG/PULSE → danh sách cảm biến, và task điều hướng `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.

## 1.2 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).

## 1.3 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. **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).
8. **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.

## 2 DataManager (Common) – Dữ liệu dùng chung

### 2.1 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.

### 2.2 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`]).
- **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.

### 3 SensorTypes và SensorRegistry

#### 3.1 SensorTypes.h

Định nghĩa:

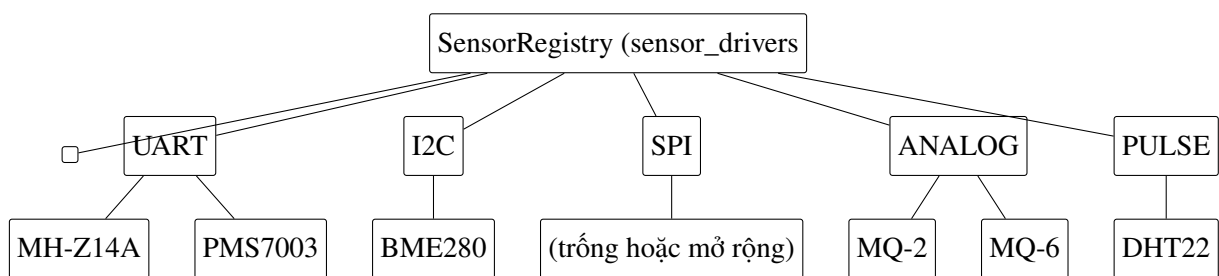
- PortId\_t (PORT\_1, PORT\_2, PORT\_3), NUM\_PORTS = 3, SensorType\_t (SENSOR\_BME280, SENSOR\_MHZ14A, ...).
- TypeCommunication\_t: COMMUNICATION\_UART, I2C, SPI, ANALOG, PULSE (để nhóm menu theo giao tiếp).
- SensorData\_t: Mảng data\_f1, data\_uint32, ... để lưu giá trị đọc được.
- sensor\_driver\_t: name, description, unit, unit\_count, interface (TypeCommunication\_t), is\_init, con trỏ hàm init/read/deinit.

#### 3.2 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\_sensor\_registry\_get\_count\_by\_interface(iface), sensor\_registry\_get\_driver\_at\_index(index, out\_sensor\_type). Hai hàm cuối phục vụ việc xây menu theo giao tiếp (UART/I2C/SPI/ANALOG/PULSE).

#### 3.3 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.

### 4 MenuSystem – Cấu trúc menu và điều hướng

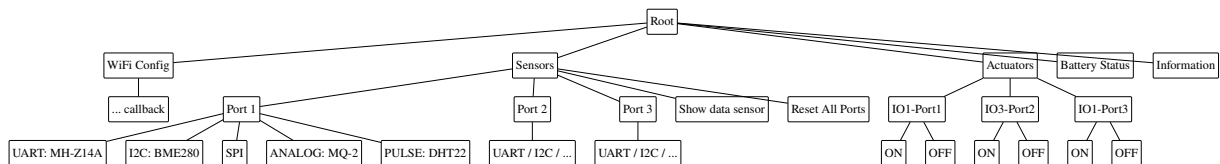
#### 4.1 Cây menu

- **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 (chỉ những driver có interface trùng với giao tiếp đó). Chọn một cảm biến → gọi `select_sensor_cb` (FunctionManager) để gán `Data->selectedSensor[port]`.
- **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 / Information:** Một hoặc vài mục con, callback do FunctionManager đảm nhiệm.

## 4.2 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.

## 4.3 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 thuộc giao tiếp đó, 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]`.

## 4.4 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.selected = sensor_type_to_name(data->selectedSensor[p])` cho mọi port p (phục vụ callback và màn hình dữ liệu cảm biến).

# 5 ScreenManager – Hiển thị OLED

## 5.1 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.
- `ScreenWifiConnecting`, `ScreenShowMessage`, `ScreenShowDataSensor`: Các màn hình đặc biệt, đều dùng chung mutex và `ssd1306_refresh_gram`.

## 5.2 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)`.

## 6 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)`. Gán `data->selectedSensor[port] = sensor`; khởi tạo driver nếu cần; tạo task đọc cảm biến nếu chưa có; có thể gọi `ScreenShowDataSensor` hoặc `MenuRender`.
- **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`; dọn task đọc cảm biến.
- **battery\_status\_callback**: `BatteryManager_UpdateInfo(&objectInfo->batteryInfo); MenuRender(MenuReturn[1], ...)`.
- **actuator\_on\_cb / actuator\_off\_cb**: Nhận `ctx = số GPIO`; cấu hình GPIO output và `gpio_set_level 1` hoặc `0`.

## 7 Cấu hình pin và Kconfig

### 7.1 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=33, 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.

## 7.2 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).

## 8 Triển khai build và chạy

### 8.1 Build

Listing 1: Build firmware với ESP-IDF

```
1 idf.py set-target esp32    # h o c  esp32c6
2 idf.py build
```

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

### 8.2 Flash và monitor

Listing 2: Nạp firmware và mở serial

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

### 8.3 Luồng dữ liệu điển hình

1. Người dùng mở Sensors → Port 1 → I2C → chọn BME280. `select_sensor_cb` gán `selectedSensor[PORT_1] = SENSOR_BME280`, khởi tạo BME280 nếu cần, tạo task đọc cảm biến.
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. Người dùng mở Actuators → IO1-Port1 → ON: `actuator_on_cb` đặt `GPIO_CONFIG_IO_1_PORT_1` lên 1.

## 9 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. Sau khi hoàn tất, cảm biến sẽ tự xuất hiện trong menu Sensors → Port 1/2/3 → [UART|I2C|SPI|ANALOG|PULS] tương ứng với interface đã gán (không cần sửa MenuSystem).

### 9.1 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 2).

Listing 3: Ví dụ thêm `SENSOR_XYZ` vào `SensorTypes.h`

```
1 // Trong SensorTypes.h, enum SensorType_t
2 typedef enum {
3     SENSOR_NONE = -1,
4     SENSOR_BME280 = 0,
5     // ... c c sensor h i n c ...
6     SENSOR_MQ135 = 12,
7     SENSOR_XYZ = 13,    // t h m m i
8 } SensorType_t;
```

### 9.2 Bước 2: Đăng ký driver trong SensorRegistry.c

Thêm một phần tử `sensor_driver_t` vào mảng `sensor_drivers[]`. Các trường bắt buộc: `name`, `description`, `unit`, `unit_count`, `interface` (UART/I2C/SPI/ANALOG/PULSE), và các con trỏ hàm `init`, `read`, `deinit` (có thể `NULL` nếu chưa triển khai).

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

```
1 // Trong SensorRegistry.c, m ng sensor_drivers[]
2 {
3     .name = "XYZ",
4     .init = xyzInitialize,    // hoac NULL
5     .read = xyzReadData,
6     .deinit = xyzDeinitialize, // hoac NULL
7     .description = {"Temp", "Humidity"},
8     .unit = {"C", "%"},
9     .unit_count = 2,
10    .is_init = false,
11    .interface = COMMUNICATION_I2C,
12 },
```

**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).

### 9.3 Bước 3: 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()`

```
1 const char *sensor_type_to_name(SensorType_t t) {
2     switch (t) {
```

```

3 // ... các case hiện có ...
4 case SENSOR_XYZ:
5     return "XYZ";
6 case SENSOR_NONE:
7     return "None";
8 default:
9     return "Unknown";
10 }
11 }

```

#### 9.4 Bước 4: Triển khai init/read/deinit (nếu chưa có)

Nếu cảm biến dùng driver riêng (ví dụ component/component/drivers/XYZ/), triển khai ba hàm `system_err_t xyzInitialize(void)`, `xyzReadData(SensorData_t *)`, `xyzDeinitialize(void)` và khai báo trong `SensorRegistry.c` (include header tương ứng). Nếu logic đọc/init nằm trong `SensorConfig` (như BME280), có thể gọi từ `SensorConfig` và trong registry chỉ cần gán con trỏ tới các hàm đó. Đảm bảo read ghi dữ liệu vào `data->data_f1[]` hoặc `data->data_uint32[]` theo `unit_count` và `description` đã khai báo.

#### 9.5 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`).

#### 9.6 Tóm tắt quy trình thêm cảm biến

1. Thêm enum trong `SensorTypes.h` (`SensorType_t`).
2. Thêm một phần tử `sensor_driver_t` vào `sensor_drivers[]` trong `SensorRegistry.c`, gán interface (UART/I2C/SPI/ANALOG/PULSE).
3. Cập nhật `sensor_type_to_name()` trong `SensorRegistry.c`.
4. Triển khai `init/read/deinit` (driver riêng hoặc `SensorConfig`).
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()`.

## 10 Tóm tắt

Báo cáo này mô tả chi tiết cấu trúc mã nguồn và cách triển khai firmware MSMS: tổ chức theo component (main, core, sensors, ui, network, utils, drivers), vai trò từng module (DataManager, SensorRegistry, MenuSystem, ScreenManager, FunctionManager, WifiManager, ButtonManager, BatteryManager), cấu trúc menu (Sensors với ba port: `Port 1/2/3` → `giao tiếp` →

cảm biến; Actuators ON/OFF; Show data sensor, Reset All Ports), cấu hình pin qua Kconfig theo ESP32/ESP32-C6, và luồng khởi tạo cùng luồng dữ liệu điển hình. Báo cáo kèm các sơ đồ cây: cây thư mục dự án, cây menu (Root → Sensors/Actuators/...), và cây cảm biến theo giao tiếp (SensorRegistry). Phần “Hướng dẫn chi tiết cách thêm cảm biến mới” mô tả từng bước: thêm enum trong SensorTypes.h, đăng ký driver trong SensorRegistry.c, cập nhật `sensor_type_to_name()`, triển khai `init/read/deinit`, và (tùy chọn) `Kconfig/GPIO`; sau đó cảm biến tự xuất hiện trong menu theo `interface`. Firmware được thiết kế để dễ mở rộng và bảo trì nhờ tách lớp rõ ràng.