# 0. 학습 개요

## STM32F103C8T6 (“Blue Pill”) 소개

### 1.1 개요

**STM32F103C8T6**, 일반적으로 **“Blue Pill”**로 알려진 이 보드는 STMicroelectronics에서 제조한 **ARM Cortex-M3 기반 32비트 마이크로컨트롤러**를 중심으로 구성된 저가형 개발 플랫폼이다.
고성능, 저전력, 그리고 풍부한 주변장치 구성을 특징으로 하며, 산업용 제어부터 임베디드 실습 환경까지 폭넓게 활용된다.

본 보드는 단순한 MCU 평가용 보드를 넘어, **센서 허브**, **데이터 로거**, **FreeRTOS 멀티태스킹 학습 플랫폼** 등 다양한 시스템 설계에 적용 가능하다.

---

### 1.2 주요 사양

| 항목 | 내용 |
|---|---|
| **MCU** | STM32F103C8T6 (ARM Cortex-M3, 72 MHz) |
| **Flash Memory** | 64 KB |
| **SRAM** | 20 KB |
| **동작 전압** | 3.3 V (I/O 일부 5 V tolerant) |
| **입출력 핀 수** | 최대 37개 (GPIO) |
| **주요 인터페이스** | USART × 3, SPI × 2, I²C × 2, USB 2.0 FS, CAN, ADC × 10 채널, PWM × 4 |
| **클록 소스** | HSE 8 MHz Crystal, LSE 32.768 kHz (옵션) |
| **전원 입력** | 5 V(USB) 또는 3.3 V(핀 입력) |
| **디버깅 포트** | SWD (SWCLK / SWDIO) |
| **보드 크기** | 약 53 mm × 22 mm |

---

### 1.3 보드 구성 요소

```
┌──────────────────────────────────────────┐
| USB Mini-B 포트 → STM32F103C8T6 MCU       |
| 8 MHz Crystal | Reset Button              |
| BOOT0 Jumper | BOOT1 Pad                  |
| SWD Header | AMS1117-3.3 V Regulator      |
| GPIO Headers: PA0–PA15, PB0–PB15, PC13–PC15|
| LED: PC13 (Active-Low)                   |
└──────────────────────────────────────────┘
```

- **BOOT0/BOOT1 점퍼** : 부트 모드 설정 (Flash / System / SRAM)

- **PC13 LED** : 기본 내장 상태 표시용 LED (Active-Low)
- **AMS1117-3.3 V** : 5 V 입력을 3.3 V로 변환하는 LDO 레귤레이터
- **SWD 헤더** : ST-LINK를 통한 디버깅 및 펌웨어 다운로드
- **USB 포트** : 전원 공급 및 CDC (USB-Serial) 통신 가능

---

## 1.4 부트 모드

STM32 시리즈는 BOOT0, BOOT1 핀 조합에 따라 부팅 경로가 결정된다.

| BOOT1 | BOOT0 | 부팅 대상 | 설명 |
|---|---|---|---|
| 0 | 0 | **Main Flash Memory** | 사용자 펌웨어 실행 |
| 0 | 1 | **System Memory (ISP)** | UART/USB ISP 부트로더 |
| 1 | 0 | **SRAM** | 디버깅 테스트 용도 |

> **참고:**
> USB-to-Serial 어댑터 (CP2102, CH340 등)를 이용하면
> BOOT0 = 1 상태에서 **USART1 (PA9 TX, PA10 RX)** 를 통해 펌웨어를 다운로드할 수 있다.

---

## 1.5 핀맵 요약

| 기능 | 핀 | 비고 |
|---|---|---|
| **LED** | PC13 | Active-Low |
| **UART1** | PA9 (TX), PA10 (RX) | 기본 시리얼 포트 |
| **UART2** | PA2 (TX), PA3 (RX) | 보조 시리얼 |
| **I²C1** | PB6 (SCL), PB7 (SDA) | 센서 통신 |
| **SPI1** | PA5 (SCK), PA6 (MISO), PA7 (MOSI) | 디스플레이 / SD 카드 |
| **ADC1** | PA0 ~ PA7, PB0, PB1 | 10 채널 입력 |
| **Timer PWM** | PA8 ~ PB1 | 모터 / 서보 제어 |
| **SWD** | PA13 (SWDIO), PA14 (SWCLK) | 디버깅 포트 |

---

## 1.6 특성 및 장점

- 저비용 대비 높은 성능 (72 MHz Cortex-M3)
- 32-비트 정밀 타이밍 및 인터럽트 처리 능력
- 다양한 주변장치 통합 지원 (UART, SPI, I²C, ADC 등)
- **STM32CubeIDE + HAL + FreeRTOS** 정식 지원 환경

- HAL API 및 레지스터 직접 제어 방식 병행 가능

---

## 1.7 유의 사항

- **PC13 LED**는 Active-Low 특성으로 `HAL_GPIO_WritePin(PC13, GPIO_PIN_RESET)` 시 켜짐.
- 일부 보드에는 USB D+ 라인 Pull-up 저항 (1.5 kΩ) 미장착 → 필요시 추가 납땜 요망.
- AMS1117 레귤레이터는 발열이 심하므로 고전류 센서 연결 시 별도 전원 모듈 사용 권장.

---

## 1.8 개발 환경

| 항목 | 내용 |
|---|---|
| **IDE** | STM32CubeIDE (Keil, PlatformIO 대체 가능) |
| **디버거** | ST-LINK V2 (SWD 디버깅 지원) |
| **펌웨어 프레임워크** | STM32Cube HAL + FreeRTOS |
| **컴파일러** | GCC ARM Embedded Toolchain |
| **언어** | C / C++ |
| **테스트 통신** | UART printf 리디렉션 또는 USB CDC 전송 |

---

## 1.9 응용 예시

| 응용 분야 | 기능 |
|---|---|
| **Smart Tank** | 초음파 + 수위 + 무게 센서 통합 계측 |
| **FreeRTOS 멀티태스킹** | 센서 / 제어 / 디스플레이 Task 분리 |
| **I²C 센서 허브** | OLED, VL53L0X 등 다중 센서 연동 |
| **저전력 센서 노드** | RTC 기반 Sleep / Wake-up 동작 |
| **IoT 게이트웨이** | ESP32 MQTT 통신 및 데이터 중계 |

---

## 1.10 요약

**STM32F103C8T6 “Blue Pill”**은
경량 하드웨어 환경에서도 정밀한 센서 계측, FreeRTOS 병렬 처리, 그리고 IoT 확장까지 지원하는
**임베디드 시스템 개발의 표준 플랫폼**이다.
저비용 구조와 산업용 MCU 수준의 성능을 동시에 갖추고 있으며,
학습 및 실무 프로토타이핑 모두에 적합하다.

# STM32CubeIDE + HAL + FreeRTOS 개발환경 구성

## 2.1 개요

**STM32CubeIDE**는 STMicroelectronics에서 공식적으로 제공하는 **통합 개발 환경(IDE)**으로,
**CubeMX의 코드 생성 기능**과 **Eclipse 기반 빌드·디버깅 환경**이 결합된 형태이다.
이를 통해 프로젝트 생성, 핀 설정, 클럭 구성, HAL 초기화 코드 자동생성, FreeRTOS 활성화까지
**모든 임베디드 개발 과정을 단일 환경에서 통합 관리**할 수 있다.

본 단원에서는 **STM32CubeIDE + HAL + FreeRTOS** 기반의 표준 펌웨어 개발 환경을 구축하는 절차를 기술한다.

---

## 2.2 개발 환경 구성요소

| 구성요소 | 설명 |
|---|---|
| **STM32CubeIDE** | Eclipse 기반 IDE. 코드 작성, 빌드, 디버깅 통합 |
| **STM32CubeMX** | MCU 설정 및 HAL 코드 자동 생성 도구 |
| **STM32 HAL (Hardware Abstraction Layer)** | 주변장치 제어를 위한 고수준 하드웨어 추상화 계층 |
| **FreeRTOS** | 경량 실시간 운영체제 (RTOS). CubeIDE에서 자동 통합 지원 |
| **ST-LINK Utility / ST-LINK V2** | 펌웨어 다운로드 및 디버깅 인터페이스 |
| **GCC ARM Embedded Toolchain** | ARM Cortex-M용 컴파일러 및 링커 |
| **USB-to-Serial (CH340/CP2102)** | UART 로그 출력용 시리얼 인터페이스 |

---

## 2.3 설치 및 설정 절차

### (1) STM32CubeIDE 설치

- 공식 다운로드 페이지: https://www.st.com/stm32cubeide
- Windows / macOS / Linux 지원
- 설치 시 **ST-LINK 드라이버** 자동 포함 (별도 설치 불필요)
- 설치 후 첫 실행 시 워크스페이스 경로 지정 (`C:\STM32\workspace` 권장)

### (2) ST-LINK 드라이버 확인

- 장치 관리자 → “Universal Serial Bus Devices” → `ST-Link Debug`
- 인식되지 않는 경우, **STM32 ST-LINK Utility**를 별도 설치하여 펌웨어 업데이트 수행

### (3) GCC ARM Toolchain 경로 확인

- 기본 내장되어 있으며, 환경변수 `PATH`에 자동 등록됨
- IDE 메뉴: `Help → About → Installation Details → ARM Toolchain` 확인 가능

---

## 2.4 신규 프로젝트 생성 절차

1. **File** → **New** → **STM32 Project** 선택
2. **Board Selector 탭**에서 `"Blue Pill" (STM32F103C8Tx)` 선택
    - Device: `STM32F103C8T6`
3. **Project Name** 지정
4. **Firmware Package 다운로드** (`STM32Cube FW_F1 v1.x.x`)
5. **Initialization Mode**: “Initialize all peripherals with their default Mode” 선택
6. 생성 완료 후 `.ioc` 파일 자동 생성

---

## 2.5 기본 설정 구성

### (1) System Clock Configuration

- HSE Crystal 8 MHz → PLL ×9 → System Clock 72 MHz
- AHB Prescaler = 1, APB1 = 2, APB2 = 1
- SysTick = 1 ms 주기 (FreeRTOS Tick과 동일)

### (2) GPIO Configuration

- LED (PC13): Output Push-Pull
- UART1 (PA9 TX, PA10 RX): Asynchronous
- I²C1 (PB6 SCL, PB7 SDA): Standard Mode (100 kHz)

### (3) NVIC Configuration

- EXTI0~15, TIMx, ADCx 등 필요한 인터럽트 활성화
- 우선순위 그룹: 4 bits for pre-emption, 0 bits for subpriority

---

## 2.6 HAL 구성

**HAL (Hardware Abstraction Layer)** 은 CubeMX에서 자동으로 생성되는 드라이버 계층으로,
레지스터 접근을 추상화하여 코드 이식성과 유지보수성을 향상시킨다.

### 주요 파일 구조

```
Core/
 ├─ Src/
 |   ├─ main.c
 |   ├─ stm32f1xx_it.c
 |   ├─ syscalls.c, sysmem.c
 |   ├─ system_stm32f1xx.c
 |   └─ freertos.c (FreeRTOS 사용 시 자동 생성)
 ├─ Inc/
 |   ├─ main.h
 |   ├─ stm32f1xx_it.h
```

```
 |   └ freertos.h
Drivers/
 ├ STM32F1xx_HAL_Driver/
 |   ├ Inc/
 |   └ Src/
 └ CMSIS/
     ├ Core/
     └ Device/
```

> HAL 함수 예시
>
> - GPIO: `HAL_GPIO_WritePin(GPIOC, GPIO_PIN_13, GPIO_PIN_RESET);`
> - ADC: `HAL_ADC_GetValue(&hadc1);`
> - I2C: `HAL_I2C_Mem_Read(&hi2c1, addr, reg, I2C_MEMADD_SIZE_8BIT, buf, len, 100);`

---

## 2.7 FreeRTOS 통합 설정

### (1) CubeMX에서 FreeRTOS 활성화

1. **Middleware → FREERTOS → Enable**
2. **Interface:** CMSIS-RTOS v2
3. **Heap Implementation:** `heap_4.c` (가장 일반적이며 동적 메모리 지원)
4. Task 생성
    - SensorTask (우선순위 2)
    - DisplayTask (우선순위 1)
    - ControlTask (우선순위 2)

### (2) 생성된 코드 구조

```
freertos.c
 ├ MX_FREERTOS_Init()
 |   ├ osThreadNew(SensorTask, ...)
 |   ├ osThreadNew(DisplayTask, ...)
 |   └ osKernelStart()
```

### (3) SysTick 및 Tickless 설정

- SysTick: FreeRTOS Tick 주기 (기본 1 ms)
- Tickless Idle 모드: 저전력 시스템 설계 시 활성화 가능

---

## 2.8 UART printf 리디렉션

표준 `printf()` 함수를 UART로 출력하기 위해 `syscalls.c` 내 `_write()` 함수를 다음과 같이 재정의한다.

```c
int _write(int file, char *ptr, int len) {
    HAL_UART_Transmit(&huart1, (uint8_t*)ptr, len, HAL_MAX_DELAY);
    return len;
}
```

이를 통해 디버깅 로그를 CubeIDE 콘솔 또는 시리얼 터미널에서 확인할 수 있다.

---

## 2.9 디버깅 환경 구성

1. **ST-LINK 연결 (SWD)**
   - SWDIO → PA13
   - SWCLK → PA14
   - GND, 3.3V 연결
2. **Run → Debug Configurations → ST-Link GDB Server**
   - Interface: SWD
   - Reset behavior: Connect under reset (필요 시)
3. **Watch / Live Expressions / Memory View** 활성화
   - 변수 실시간 모니터링 가능

---

## 2.10 요약

| 단계 | 구성 요소 | 목적 |
|---|---|---|
| 1 | STM32CubeIDE 설치 | 개발 통합 환경 확보 |
| 2 | CubeMX 설정 | 핀맵 및 클럭 자동화 |
| 3 | HAL 구성 | 하드웨어 제어 코드 추상화 |
| 4 | FreeRTOS 활성화 | 멀티태스킹 구조 구현 |
| 5 | ST-LINK 디버깅 | 실시간 코드 검증 및 분석 |
| 6 | UART printf 리디렉션 | 로그 및 디버깅 메시지 출력 |

---

## 결론

STM32CubeIDE + HAL + FreeRTOS 환경은
임베디드 펌웨어 개발의 **표준화된 워크플로우**를 제공한다.
자동화된 코드 생성, 모듈화된 HAL 구조, 실시간 Task 스케줄링,
그리고 ST-LINK 기반의 정밀 디버깅까지 하나의 IDE에서 통합되어
개발 생산성과 시스템 신뢰성을 동시에 확보할 수 있다.

# 디버깅 환경(ST-LINK, UART, printf 리디렉션)

## 3.1 개요

STM32F103C8T6 (“Blue Pill”) 개발 과정에서 **디버깅(Debugging)** 은 단순한 코드 검증 단계를 넘어
**펌웨어의 동작 흐름, 인터럽트 타이밍, 변수 값, 시스템 부하 분석** 등을 실시간으로 검증하기 위한 핵심 절차이다.

STM32CubeIDE 환경에서는 다음 세 가지 주요 방식으로 디버깅이 수행된다.

| 구분 | 설명 |
| --- | --- |
| **ST-LINK** | 하드웨어 레벨의 실시간 디버깅 (브레이크포인트, 변수 추적, 레지스터 관찰) |
| **UART 로그 출력** | 프로그램 내부 상태를 텍스트 로그로 출력 |
| **printf 리디렉션** | 표준 출력(`printf`)을 UART 또는 SWO로 재전송하여 디버깅 메시지 출력 |

---

## 3.2 ST-LINK 디버깅 환경

### (1) 개요

**ST-LINK**는 STMicroelectronics에서 제공하는 공식 **SWD (Serial Wire Debug)** 프로그래머이자 디버거이다.
STM32CubeIDE는 ST-LINK를 기본적으로 지원하며, Eclipse 기반 **GDB 서버 디버깅 세션**을 통해 MCU 내부 상태를 실시간으로 분석할 수 있다.

### (2) 연결 구성

| ST-LINK 핀 | Blue Pill 핀 | 기능 |
| --- | --- | --- |
| SWDIO | PA13 | 데이터 라인 |
| SWCLK | PA14 | 클럭 라인 |
| GND | GND | 공통 접지 |
| 3.3V | 3.3V | 전원 공급 |
| (NRST, 선택) | RESET | 리셋 제어용 (선택 연결) |

### (3) 설정 절차

1. **Run → Debug Configurations → STM32 Cortex-M C/C++ Application**
2. **Debugger 탭 → ST-LINK (OpenOCD)** 선택
    - Interface: **SWD**
    - Reset Behavior: **Connect under reset** (필요 시)
3. **Startup 탭 → Load image after reset** 활성화
4. **F11 (Debug)** 실행 → 실시간 브레이크포인트 및 스텝 동작 확인

### (4) 주요 기능

- **Breakpoints:** 코드 중단점 설정 및 실행 흐름 분석
- **Watch Expressions:** 변수 및 구조체 실시간 감시
- **Memory View:** 특정 메모리 주소 영역 조회 (Flash, SRAM, Peripheral 등)
- **Peripheral Registers:** HAL 계층과 실제 하드웨어 레지스터 상태 비교

---

## 3.3 UART 디버깅 환경

### (1) 개요

UART(Universal Asynchronous Receiver/Transmitter)는 가장 기본적이면서도 안정적인 디버깅 채널이다.
`printf()`로 출력되는 디버깅 정보를 USB-to-Serial 어댑터를 통해 PC의 시리얼 터미널(예: TeraTerm, PuTTY)로 확인할 수 있다.

### (2) 하드웨어 연결

| Blue Pill 핀 | 어댑터 핀 | 기능 |
|---|---|---|
| PA9 | RXD | UART TX |
| PA10 | TXD | UART RX |
| GND | GND | 공통 접지 |

> ⚠️ 주의
> UART1은 기본적으로 **3.3 V TTL** 레벨을 사용하므로,
> **RS-232C 변환기(±12 V 레벨)** 와 직접 연결해서는 안 된다.

### (3) CubeMX 설정

- Peripherals → USART1 → Mode: Asynchronous
- Baud Rate: 115200
- Word Length: 8 Bits
- Stop Bits: 1
- Parity: None
- Hardware Flow Control: None
- NVIC Interrupt Enable: 활성화

### (4) 코드 예시

```
uint8_t msg[] = "UART Debug Test\r\n";
HAL_UART_Transmit(&huart1, msg, sizeof(msg)-1, HAL_MAX_DELAY);
```

UART 전송 함수는 **Blocking** 모드로 동작하므로, 실시간 시스템에서는 **Interrupt 또는 DMA 전송** 방식이 권장된다.

---

## 3.4 printf 리디렉션

### (1) 개념

C 표준 라이브러리의 `printf()` 함수는 기본적으로 **stdout** 스트림을 사용한다.
임베디드 환경에서는 표준 콘솔 장치가 존재하지 않으므로,
이를 **UART 전송 함수로 재매핑(redirect)** 해야 한다.

### (2) 구현

`syscalls.c` 파일 내 `_write()` 함수를 다음과 같이 재정의한다.

```
int _write(int file, char *ptr, int len) {
    HAL_UART_Transmit(&huart1, (uint8_t*)ptr, len, HAL_MAX_DELAY);
    return len;
}
```

이후 `printf()` 호출 시 자동으로 UART1을 통해 데이터가 출력된다.

```
printf("System initialized.\r\n");
printf("Temperature = %.2f°C\r\n", temp);
```

### (3) 출력 확인

1. UART-USB 어댑터 연결 (PA9-TX, PA10-RX)
2. 시리얼 터미널 실행 (TeraTerm, PuTTY 등)
3. 포트 설정: 115200 8N1
4. 콘솔 출력 예시:

```
System initialized.
Temperature = 24.35°C
Sensor active.
```

---

## 3.5 SWO (Serial Wire Output) 디버깅 (선택)

STM32F103C8T6의 경우 일부 변형 칩에서 **SWO (Serial Wire Output)** 기능이 제공된다.
이를 활용하면 `printf()` 출력을 **ST-LINK의 단일 데이터 라인(SWO)** 을 통해 IDE 콘솔로 직접 송신할 수 있다.
단, Blue Pill 보드 기본형은 SWO 핀이 노출되지 않은 경우가 많다.

CubeIDE에서 SWO 사용 시 설정:

- **Debug Configurations → SWV Trace → Enable SWO ITM Data**
- **Core Clock:** 72 MHz
- **SWO Clock:** 2 MHz

---

## 3.6 디버깅 시나리오 예시

| 구분 | 사용 도구 | 목적 |
|---|---|---|
| **Step Debugging** | ST-LINK | 코드 흐름 추적 및 인터럽트 원인 분석 |
| **Live Variables** | ST-LINK | FreeRTOS Task 변수 실시간 관찰 |
| **UART Logging** | CH340/CP2102 | 센서 출력, 상태 메시지 확인 |
| **printf Redirect** | HAL UART | 텍스트 기반 이벤트 로깅 |

---

## 3.7 요약

| 항목 | 기능 | 비고 |
|---|---|---|
| **ST-LINK 디버깅** | 코드 중단, 변수/레지스터 실시간 확인 | 실시간 GDB 제어 |
| **UART 디버깅** | 센서 데이터, 이벤트 로그 출력 | 저비용, 신뢰성 높음 |
| **printf 리디렉션** | 표준 출력 → UART 송신 | IDE 콘솔 또는 터미널 연동 |
| **SWO 출력 (선택)** | ST-LINK 단일 라인 데이터 송출 | 고속 실시간 로그 지원 |

---

## 결론

STM32CubeIDE는 ST-LINK 하드웨어 디버거와 UART 소프트웨어 로그 시스템을 완벽히 통합한다.
하드웨어 수준의 실시간 분석과 텍스트 기반의 런타임 로깅을 병행함으로써,
**FreeRTOS 기반 멀티태스킹 시스템의 정확도와 안정성을 동시에 확보**할 수 있다.

# 프로젝트 기반 학습 로드맵

- # 센서 계측 → 데이터 처리 → FreeRTOS → 제어 → 최적화

## 4.1 개요

STM32F103C8T6(“Blue Pill”) 학습은 단순한 주변장치 실습을 넘어
**센서 계측 → 데이터 처리 → FreeRTOS 기반 병렬 제어 → 시스템 최적화** 로 이어지는
**완전한 임베디드 시스템 설계 흐름**을 단계적으로 학습하는 것을 목표로 한다.

이 로드맵은 실제 산업용 제어기, IoT 센서 노드, 스마트 수조(Smart Tank) 등의
응용 프로젝트로 확장될 수 있도록 구성되어 있다.

---

## 4.2 학습 흐름

```
[1] 센서 계측 → [2] 데이터 처리 → [3] FreeRTOS → [4] 제어 → [5] 최적화
```

---

## 4.3 단계별 학습 내용

### (1) 센서 계측 (Measurement Layer)

- **목표:**
  MCU와 다양한 센서 간의 인터페이스 구조를 이해하고,
  실제 물리량(거리, 무게, 수위 등)을 디지털 데이터로 변환하는 기술을 익힌다.
- **핵심 주제:**
  - GPIO, ADC, I²C, Timer Input Capture
  - HC-SR04, HX711, VL53L0X, Capacitive Water Sensor
  - HAL 기반 센서 드라이버 구현
- **출력 결과:**
  - 실시간 거리·무게·수위 데이터 획득
  - UART/OLED를 통한 실시간 데이터 표시

---

### (2) 데이터 처리 (Processing Layer)

- **목표:**
  수집된 센서 데이터를 정제하고 보정하여,
  노이즈 제거 및 신뢰도 향상을 위한 데이터 처리 알고리즘을 구현한다.
- **핵심 주제:**
  - 오프셋 및 스케일 보정 (Calibration)
  - 이동평균, 지수평활, 칼만필터 적용
  - 타임아웃 및 오류 복구 처리
  - EEPROM / Flash를 이용한 보정값 저장
- **출력 결과:**
  - 안정화된 측정 데이터 출력
  - 오차율 ±1% 이내 유지

---

### (3) FreeRTOS (Concurrency Layer)

- **목표:**
  단일 루프 방식에서 벗어나 **Task 기반 병렬처리 구조**를 설계하고,
  실시간 시스템의 스케줄링 및 동기화 메커니즘을 이해한다.
- **핵심 주제:**
  - FreeRTOS Task 생성 및 스케줄링

    - Queue, Mutex, Semaphore, Event Group
    - Tickless Idle 및 저전력 설계
    - Task 간 통신 (SensorTask ↔ ControlTask ↔ DisplayTask)
- **출력 결과:**
    - 각 센서·제어·디스플레이 루틴의 병렬 실행
    - 실시간성 및 시스템 안정성 확보

---

## (4) 제어 (Control Layer)

- **목표:**
  센서 데이터를 바탕으로 펌프, 밸브, 모터 등의 액추에이터를 제어하고,
  폐루프(Closed-loop) 피드백 제어를 구현한다.
- **핵심 주제:**
    - PWM 제어, 릴레이/MOSFET 구동
    - PID 및 비례제어 (P-Control)
    - 인터럽트 기반 이벤트 제어
    - RTC 스케줄 기반 자동 동작
- **출력 결과:**
    - 목표 수위/온도 유지 제어
    - 일정 주기별 자동 동작 및 경보 시스템

---

## (5) 최적화 (Optimization Layer)

- **목표:**
  실시간 시스템의 효율을 향상시키고,
  소비전력 및 응답 시간을 최소화하는 펌웨어 최적화를 수행한다.
- **핵심 주제:**
    - DMA 적용 (ADC, I²C, UART)
    - Tickless Idle 및 STOP 모드
    - Static Memory Allocation
    - 실행 시간 측정, 부하 분석
- **출력 결과:**
    - CPU 점유율 및 전력소비 감소
    - 안정적인 장시간 운전 가능 시스템

---

## 4.4 단계 간 연계 구조

```
┌──────────────────────────────────┐
| [1] 센서 계측                    |
| └ 물리량 → 디지털 변환           |
|                                  |
| ↓                                |
| [2] 데이터 처리                  |
| └ 보정 / 필터링 / 저장           |
|                                  |
| ↓                                |
| [3] FreeRTOS                     |
| └ 병렬 Task 스케줄링             |
|                                  |
| ↓                                |
| [4] 제어                         |
| └ 펌프/밸브 구동 및 피드백       |
|                                  |
| ↓                                |
| [5] 최적화                       |
| └ DMA, 저전력, 성능 개선         |
└──────────────────────────────────┘
```

---

## 4.5 최종 학습 결과물

| 구분 | 내용 |
|---|---|
| **통합 프로젝트명** | Smart Tank (스마트 수조 제어 시스템) |
| **핵심 기능** | 초음파·수위·무게 센서 통합 계측, FreeRTOS 병렬 실행, 자동 수위 제어 |
| **핵심 기술** | HAL + FreeRTOS + PID 제어 + DMA 최적화 |
| **출력 장치** | OLED 디스플레이 / UART 로그 / RTC 알람 |
| **확장 기능** | IoT 전송 (MQTT), SD카드 로깅, 전력 최적화 |

---

## 4.6 요약

| 단계 | 학습 주제 | 주요 목표 |
|---|---|---|
| 1 | **센서 계측** | 하드웨어 인터페이스 및 데이터 취득 |
| 2 | **데이터 처리** | 보정·필터링을 통한 신뢰도 향상 |
| 3 | **FreeRTOS** | 실시간 병렬 스케줄링 구현 |
| 4 | **제어** | 폐루프 기반 자동 제어 |
| 5 | **최적화** | 성능·전력 효율 향상 및 안정성 확보 |

## 결론

본 로드맵은 단순한 예제 수준의 MCU 학습이 아니라,
**하드웨어–소프트웨어–제어 알고리즘–실시간 OS–시스템 최적화**를 통합적으로 다루는
**실전형 임베디드 개발 학습 체계**이다.
각 단계는 독립적으로 실습 가능하면서도,
최종적으로 하나의 **완성된 실시간 제어 시스템(Smart Tank)** 으로 귀결되도록 설계되어 있다.