15. 응용 및 실습 예제

15.1 어셈블리 코드 작성 및 디버깅(x86_64 기준)

🧠 어셈블리(Assembly)란?

- **CPU가 직접 해석할 수 있는 기계어(machine code)**와
- **1:1 대응되는 저수준 언어**
- 각 명령어는 CPU 아키텍처(예: x86, ARM 등)별로 다름

> 목적: 부트로더, OS 커널, 성능 최적화, 시스템 호출, 디버깅에 활용됨

🧩 어셈블리 코드 기본 형식 (AT&T vs Intel)

구분	설명
AT&T 문법	GNU/Linux에서 사용 (`.s`, `.S` 파일, `as`)
Intel 문법	Windows/NASM 스타일 (Intel 문서 기반)

```
# AT&T 문법 예
movl $5, %eax     # eax ← 5
addl $3, %eax     # eax ← eax + 3

# Intel 문법 예
mov eax, 5
add eax, 3
```

🧰 툴체인 구성

작업	도구
작성	`.s`, `.S`, `.asm`
어셈블	`as`, `nasm`, `gas`
링크	`ld`, `gcc`
디버깅	`gdb`, `lldb`, `objdump`, `readelf`

📄 간단한 어셈블리 예제 (x86_64, Linux)

```
.section .data
msg:    .asciz "Hello, World!\n"

.section .text
.globl _start
_start:
```

```
    mov $1, %rax          # syscall: write
    mov $1, %rdi          # fd: stdout
    mov $msg, %rsi        # buf
    mov $13, %rdx         # size
    syscall

    mov $60, %rax         # syscall: exit
    xor %rdi, %rdi        # exit code 0
    syscall
```

🔧 컴파일 및 실행

```
as -o hello.o hello.s
ld -o hello hello.o
./hello
```

📦 스택 및 함수 호출 구조

```
push %rbp         # 이전 base pointer 저장
mov %rsp, %rbp    # 새로운 스택 프레임 생성

sub $16, %rsp     # 지역변수 공간 확보
mov $42, -4(%rbp) # 지역변수에 저장

leave             # %rbp → %rsp, 이전 %rbp 복원
ret               # 호출한 곳으로 복귀
```

> 함수 호출 시 스택 기반 프레임을 생성하고
> 인자는 일반적으로 `%rdi`, `%rsi`, `%rdx`, `%rcx`, `%r8`, `%r9`에 저장됨

🪲 gdb 디버깅 기초

```
gdb ./hello
```

주요 명령어

명령	설명
`layout asm`	어셈블리 화면 보기
`start`	프로그램 시작
`si` / `ni`	한 명령어(step) 실행
`disas`	현재 함수 디스어셈블
`info registers`	레지스터 상태 보기
`x/s $rsi`	메모리 주소 값 출력 (e.g., 문자열)
`break _start`	심볼 브레이크포인트

명령	설명
`b *0xaddr`	주소 지정 BP
`r` / `c`	실행 / 계속
`quit`	종료

📉 disassemble 예시 (objdump)

```
objdump -d hello
```

```
0000000000401000 <_start>:
  401000: b8 01 00 00 00       mov    $0x1,%eax
  401005: bf 01 00 00 00       mov    $0x1,%edi
  ...
```

→ 바이너리 코드의 **어셈블리 수준에서 실행 흐름을 확인**할 수 있음

🧪 시스템 콜 예 (write, exit)

시스템 콜	번호 (`%rax`)	인자 (%rdi, %rsi, %rdx 등)
write	1	fd, buf, size
exit	60	exit_code
open	2	pathname, flags, mode
read	0	fd, buf, size

> syscall 시 `rax`에 호출 번호 넣고 `syscall` 실행

📌 요약 정리

항목	내용
어셈블리 목적	하드웨어 직접 제어, 부트 코드, 최적화
주요 명령어	`mov`, `add`, `call`, `ret`, `syscall`
레지스터	x86_64: `%rax`, `%rdi`, `%rsp`, `%rbp` 등
스택 처리	함수 호출 시 `push`, `mov`, `leave`, `ret` 사용
디버깅 도구	`gdb`, `objdump`, `readelf`
시스템 콜	리눅스에서는 syscall 번호 기반 직접 호출 가능

15.2 LED, 스위치, 모터 제어 실습 (GPIO)

🧠 GPIO란?

GPIO (General Purpose Input/Output)는
임베디드 시스템에서 **디지털 신호를 주고받기 위해 사용하는 핀 인터페이스**이다.

핀 설정 모드	동작
Input	스위치, 센서 값 읽기
Output	LED, 모터, 릴레이 등 제어
Alternate	PWM, SPI, UART 등 특수 기능 전환 시 사용

🧩 기본 흐름 (GPIO 제어 절차)

1. 핀 번호 선택 (보드에 따라 다름)
2. 방향 설정 (input/output)
3. 값 설정 또는 읽기 (0 or 1)
4. 모터/PWM 제어는 **주기적 신호 출력**

✅ 1. LED 제어 실습 (Raspberry Pi 예: BCM GPIO 17)

A. Bash(sysfs 방식)

```
echo "17" > /sys/class/gpio/export
echo "out" > /sys/class/gpio/gpio17/direction
echo "1" > /sys/class/gpio/gpio17/value   # ON
sleep 1
echo "0" > /sys/class/gpio/gpio17/value   # OFF
```

B. C 언어 (파일 기반)

```
#include <stdio.h>
#include <unistd.h>
#include <fcntl.h>

int main() {
    int fd = open("/sys/class/gpio/export", O_WRONLY);
    write(fd, "17", 2); close(fd);

    fd = open("/sys/class/gpio/gpio17/direction", O_WRONLY);
    write(fd, "out", 3); close(fd);

    fd = open("/sys/class/gpio/gpio17/value", O_WRONLY);
    write(fd, "1", 1);  // ON
    sleep(1);
    write(fd, "0", 1);  // OFF
    close(fd);
```

```
    return 0;
}
```

✅ 2. 스위치 입력 실습 (GPIO 18)

```
echo "18" > /sys/class/gpio/export
echo "in" > /sys/class/gpio/gpio18/direction

cat /sys/class/gpio/gpio18/value
# 0: 누르지 않음, 1: 눌림
```

> C언어에서는 `poll()`이나 `select()`로 인터럽트 감지 가능

✅ 3. 모터 제어 (단순 ON/OFF or PWM)

A. 릴레이 or 트랜지스터 기반 On/Off

```
echo "22" > /sys/class/gpio/export
echo "out" > /sys/class/gpio/gpio22/direction
echo "1" > /sys/class/gpio/gpio22/value   # 모터 ON
sleep 2
echo "0" > /sys/class/gpio/gpio22/value   # 모터 OFF
```

B. PWM을 이용한 속도 제어 (Raspberry Pi)

```
# enable PWM 핀 (ex. GPIO18)
echo 0 > /sys/class/pwm/pwmchip0/export
echo 1000000 > /sys/class/pwm/pwmchip0/pwm0/period
echo 500000  > /sys/class/pwm/pwmchip0/pwm0/duty_cycle
echo 1 > /sys/class/pwm/pwmchip0/pwm0/enable
```

> 50% 듀티비 = 0.5초 ON / 0.5초 OFF
> duty_cycle 값을 늘리면 속도가 빨라지고, 줄이면 느려짐

⚙️ 직접 레지스터 제어 방식 (STM32, Baremetal)

STM32 예제 (HAL 없이 레지스터 직접 접근)

```
#define RCC_AHB2ENR  (*(volatile unsigned int*)0x4002104C)
#define GPIOA_MODER  (*(volatile unsigned int*)0x48000000)
#define GPIOA_ODR    (*(volatile unsigned int*)0x48000014)

int main() {
    RCC_AHB2ENR |= (1 << 0);        // GPIOA 클럭 인가
    GPIOA_MODER &= ~(3 << (2*5));  // PA5를 출력 모드로
    GPIOA_MODER |=  (1 << (2*5));

    while (1) {
        GPIOA_ODR |= (1 << 5);     // PA5 ON
        delay();
```

```
        GPIOA_ODR &= ~(1 << 5);   // PA5 OFF
        delay();
    }
}
```

📐 보드별 GPIO 핀맵 예 (Raspberry Pi 4)

핀번호	GPIO	기능
11	GPIO17	일반 출력 (LED 연결)
12	GPIO18	PWM 가능
13	GPIO27	일반 입력 (스위치)

🛠️ 디버깅 팁

문제	원인
`permission denied`	`sudo` 안 함
LED 안 켜짐	`direction` 설정 미비, 회로 이상
스위치 안 감지됨	`pull-up/down` 설정 필요
PWM 미동작	커널에 PWM 드라이버 없음 (device tree 확인)

📌 요약 정리

항목	설명
LED 제어	`gpio/export`, `direction`, `value` 통해 on/off
스위치 입력	input 모드 + `value` polling 또는 interrupt
모터 제어	릴레이 ON/OFF or PWM 출력
PWM 제어	`period`, `duty_cycle`, `enable` 설정
직접 제어	STM32/MCU에서는 레지스터 직접 접근
디버깅 도구	`gpio readall`, `gpiodetect`, `dmesg`, `scope` 등 사용 가능

15.3 타이머, 인터럽트, ADC 실습

🔧 1. 타이머(Timer) 실습

🧠 타이머란?

MCU 내부 클럭을 기준으로 일정 시간 후 **인터럽트를 발생**하거나
PWM 신호를 생성하는 장치.

타입	설명
Basic Timer	단순 주기 타이머 (인터럽트용)
General-purpose Timer	카운터 + PWM + 캡처 등
Advanced Timer	Complementary PWM, Dead-Time 등 포함 (서보, BLDC 제어용)

✅ 예: STM32에서 1초마다 인터럽트 발생

A. HAL 방식 (STM32CubeMX로 설정)

```
void HAL_TIM_PeriodElapsedCallback(TIM_HandleTypeDef *htim) {
    if(htim->Instance == TIM2) {
        HAL_GPIO_TogglePin(GPIOA, GPIO_PIN_5);  // LED Toggle
    }
}
```

- CubeMX에서 TIM2 → Interrupt Enabled, Prescaler/Period 조절
- HAL_TIM_Base_Start_IT() 호출 필요

B. 레지스터 방식 예제 (Pseudo C)

```
RCC->APB1ENR |= (1 << 0);          // TIM2 클럭 인가
TIM2->PSC = 7999;                  // 8MHz / (7999+1) = 1kHz
TIM2->ARR = 999;                   // 1kHz → 1초 (1000주기)
TIM2->DIER |= 1;                   // 인터럽트 허용
TIM2->CR1 |= 1;                    // 타이머 시작
NVIC_EnableIRQ(TIM2_IRQn);        // 인터럽트 등록

void TIM2_IRQHandler(void) {
    if (TIM2->SR & 1) {
        TIM2->SR &= ~1;
        GPIOA->ODR ^= (1 << 5);   // LED Toggle
    }
}
```

🚨 2. 인터럽트(Interrupt) 실습

🧠 외부 인터럽트(EXTI)

스위치 같은 외부 신호 입력 시 즉시 MCU에 알려주는 비동기 이벤트 처리.

EXTI 라인	GPIO 연결 가능 핀
EXTI0	PA0, PB0, PC0...

EXTI 라인	GPIO 연결 가능 핀
EXTI13	PC13 (보통 USER 버튼 연결됨)

✅ 예: USER 버튼 누르면 LED Toggle

A. HAL 방식

```
void HAL_GPIO_EXTI_Callback(uint16_t GPIO_Pin) {
    if(GPIO_Pin == GPIO_PIN_13) {
        HAL_GPIO_TogglePin(GPIOA, GPIO_PIN_5);
    }
}
```

- CubeMX에서 PC13 → External Interrupt Mode 설정
- NVIC EXTI Line13 활성화

✅ 인터럽트 처리 흐름 (레지스터 기반)

```
SYSCFG->EXTICR[3] |= (0x02 << 4);  // EXTI13 ← PC13
EXTI->IMR |= (1 << 13);            // 인터럽트 마스크
EXTI->FTSR |= (1 << 13);           // Falling edge 트리거
NVIC_EnableIRQ(EXTI15_10_IRQn);

void EXTI15_10_IRQHandler(void) {
    if (EXTI->PR & (1 << 13)) {
        EXTI->PR |= (1 << 13);     // Pending bit 클리어
        GPIOA->ODR ^= (1 << 5);    // LED Toggle
    }
}
```

🔍 3. ADC (Analog to Digital Converter) 실습

🧠 ADC란?

- **0~Vref 사이의 아날로그 전압을 디지털 값(정수)로 샘플링**
- 10-bit → 0~1023 / 12-bit → 0~4095

✅ 기본 흐름

```
HAL_ADC_Start(&hadc1);                      // ADC 시작
HAL_ADC_PollForConversion(&hadc1, 10);      // 변환 완료 대기
adc_val = HAL_ADC_GetValue(&hadc1);         // 결과 획득
```

- CubeMX에서 ADC1 활성화, 채널 선택 (e.g. IN0 = PA0)

✅ 레지스터 방식 (간단 예)

```
RCC->APB2ENR |= (1 << 9);       // ADC1 클럭
ADC1->SQR3 = 0;                 // Channel 0
ADC1->CR2 |= (1 << 0);          // Enable ADC
ADC1->CR2 |= (1 << 30);         // Start conversion

while (!(ADC1->SR & 2));        // EOC 확인
val = ADC1->DR;                 // 결과 읽기
```

📊 실습 예: 타이머 + ADC + LED 밝기 제어

```
void HAL_TIM_PeriodElapsedCallback(TIM_HandleTypeDef *htim) {
    HAL_ADC_Start(&hadc1);
    HAL_ADC_PollForConversion(&hadc1, 10);
    uint32_t adc_val = HAL_ADC_GetValue(&hadc1);

    __HAL_TIM_SET_COMPARE(&htim3, TIM_CHANNEL_1, adc_val); // PWM duty 변경
}
```

> 아날로그 포텐셔미터 값으로 LED 밝기를 실시간 조절

📌 요약 정리

항목	내용
타이머	주기적 인터럽트, PWM 생성, 카운터 가능
외부 인터럽트	스위치, 센서 등 비동기 입력 처리
ADC	0~Vref 전압 → 디지털 값 (0~4095) 변환
사용 목적	센서 읽기, 모터 제어, 상태 전환 감지
실습 툴	STM32 HAL, 레지스터 직접제어, CubeMX 구성

15.4 UART 통신을 통한 PC 연동

🧠 UART란?

UART (Universal Asynchronous Receiver/Transmitter)는
비동기 방식의 직렬 시리얼 통신 프로토콜로,
MCU ↔ MCU, MCU ↔ PC, MCU ↔ 센서 등의 간단한 데이터 통신에 사용된다.

특징	설명
비동기식	클럭 선 없음, **Start/Stop 비트**로 동기화
Half-Duplex / Full-Duplex	TX, RX 라인
전송 속도	Baud rate (bps) 단위, 예: 9600, 115200

📶 UART 신호 구성

```
[Start] [Data(8-bit)] [Parity(optional)] [Stop]

예: 9600-8-N-1
→ 보레이트: 9600bps, 데이터: 8비트, 패리티 없음, 스탑비트 1개
```

🧩 UART 연결 구성도 (MCU ↔ PC)

```
MCU TX ——— RX PC (USB-UART)
MCU RX ——— TX PC (USB-UART)
MCU GND ——— GND PC
```

※ PC는 일반적으로 **USB to UART 변환 케이블(CH340, CP2102 등)**을 통해 연결

🧪 STM32 UART 기본 실습 (HAL 기반)

A. 송신 코드 (Tx: printf redirect)

```c
#include "usart.h"
int __io_putchar(int ch) {
    HAL_UART_Transmit(&huart2, (uint8_t*)&ch, 1, 100);
    return ch;
}

HAL_UART_Transmit(&huart2, (uint8_t*)"Hello UART\r\n", 12, 100);
```

B. 수신 코드 (Rx polling 방식)

```c
uint8_t rx;
HAL_UART_Receive(&huart2, &rx, 1, HAL_MAX_DELAY);
HAL_UART_Transmit(&huart2, &rx, 1, 100);  // Echo
```

> HAL_UART_Receive_IT() or HAL_UART_Receive_DMA()로 인터럽트 기반 수신도 가능

✅ UART 보레이트 설정 (CubeMX 기준)

항목	설정 예
Baud Rate	115200
Word Length	8 bits
Stop Bits	1
Parity	None
Flow Control	None

→ **PC 측 터미널 프로그램도 동일 설정**이어야 통신 성공

💻 PC 측: 시리얼 터미널 연결

Linux: `minicom`

```
sudo minicom -b 115200 -D /dev/ttyUSB0
```

Windows: `PuTTY`, `TeraTerm`

항목	설정값
포트	COMx (장치 관리자 확인)
Baud Rate	115200
Data	8
Parity	None
Stop Bit	1

🛠️ 송수신 실습 예

```
while (1) {
    uint8_t rx;
    HAL_UART_Receive(&huart2, &rx, 1, HAL_MAX_DELAY);
    if (rx == 'a') {
        HAL_GPIO_TogglePin(GPIOA, GPIO_PIN_5);  // LED 토글
        HAL_UART_Transmit(&huart2, (uint8_t*)"Toggled LED\r\n", 13, 100);
    }
}
```

→ PC에서 'a' 전송 시 LED가 깜빡이고, 메시지 응답

⚙️ 디버깅 팁

문제	원인
화면 출력 없음	보레이트 불일치
이상한 문자	데이터 비트/패리티 설정 불일치
수신 없음	RX 핀 설정 누락, NVIC 설정 확인
printf 안 나옴	`__io_putchar()` 오버라이드 누락

📌 요약 정리

항목	설명
UART 기본 구성	TX, RX, GND 3선

항목	설명
보레이트 설정	9600 ~ 115200 (양쪽 동일해야 함)
MCU 코드	HAL_UART_Transmit(), HAL_UART_Receive()
PC 연결	USB-UART → /dev/ttyUSBx or COMx
터미널 도구	minicom, PuTTY, TeraTerm 등
활용 예	디버깅 메시지, 센서 데이터 송신, 명령 수신

15.5 RTOS와 마이크로프로세서 연동 실습

🧠 RTOS란?

RTOS (실시간 운영체제)는
MCU에서 **여러 작업(Task)을 멀티태스킹 구조로 관리**하고
정해진 시간 내 동작을 보장하기 위한 경량 커널 시스템이다.

특징	설명
선점형 / 비선점형	태스크 우선순위 기반으로 선점 가능
태스크	독립적 실행 단위 (스레드처럼 동작)
커널 객체	큐, 세마포어, 뮤텍스, 타이머 등
ISR 분리	인터럽트 → 태스크로 분리 처리 가능

🛠️ 실습 구성 예 (FreeRTOS + STM32)

- **Task1**: UART 입력 처리
- **Task2**: LED 주기적 점멸
- **ADC Task**: 센서 값 읽기
- **Queue**: UART 수신 데이터 공유
- **Semaphore**: 인터럽트 동기화 처리

✅ 기본 RTOS 초기화 흐름

```c
int main(void) {
    HAL_Init();
    SystemClock_Config();

    MX_GPIO_Init();
    MX_USART2_UART_Init();

    xTaskCreate(Task1, "LED", 128, NULL, 1, NULL);
    xTaskCreate(Task2, "UART", 128, NULL, 2, NULL);

    vTaskStartScheduler();  // RTOS 시작
    while (1);              // 도달하지 않음
}
```

✅ 태스크 작성 예시

```c
void Task1(void *pvParameters) {
    while (1) {
        HAL_GPIO_TogglePin(GPIOA, GPIO_PIN_5);
        vTaskDelay(pdMS_TO_TICKS(500));  // 500ms delay
    }
}

void Task2(void *pvParameters) {
    uint8_t data;
    while (1) {
        if (xQueueReceive(uartQueue, &data, portMAX_DELAY) == pdPASS) {
            HAL_UART_Transmit(&huart2, &data, 1, 100);
        }
    }
}
```

📩 UART + RTOS 연동: 인터럽트 → Queue

1. 인터럽트 콜백

```c
void HAL_UART_RxCpltCallback(UART_HandleTypeDef *huart) {
    BaseType_t xHigherPriorityTaskWoken = pdFALSE;
    xQueueSendFromISR(uartQueue, &rx_byte, &xHigherPriorityTaskWoken);
    portYIELD_FROM_ISR(xHigherPriorityTaskWoken);
    HAL_UART_Receive_IT(&huart2, &rx_byte, 1);  // 재시작
}
```

2. 메인에서 초기화

```
xQueueCreate(10, sizeof(uint8_t));
HAL_UART_Receive_IT(&huart2, &rx_byte, 1);
```

📈 RTOS + ADC 연동 예

```
void ADCTask(void *pvParam) {
    uint32_t adc;
    while (1) {
        HAL_ADC_Start(&hadc1);
        HAL_ADC_PollForConversion(&hadc1, 10);
        adc = HAL_ADC_GetValue(&hadc1);
        vTaskDelay(pdMS_TO_TICKS(200));
    }
}
```

📦 RTOS 커널 객체 정리

객체	설명
Task	독립적인 실행 단위
Queue	태스크 간 메시지 전달
Semaphore	자원 접근 제어, ISR-Task 간 동기화
Mutex	우선순위 반전 보호 포함
Timer	소프트웨어 타이머, 콜백 기반
Event Group	비트 단위 상태 공유 (FSM에 유용)

🧪 실습 구조 요약 예

```
[ UART ISR ] ──▶ [ Queue ] ──▶ [ UART Task ]
                                   ↓
                               printf / LED 반응

[ ADC Task ] ──────────────────▶ 센서값 처리

[ LED Task ] ──────────────────▶ 주기적 토글
```

📌 요약 정리

항목	설명
RTOS 핵심	Task, Queue, Semaphore, ISR-Task 분리
FreeRTOS 주요 API	`xTaskCreate`, `vTaskDelay`, `xQueueSend`, `xSemaphoreGive`

항목	설명
인터럽트 연동	`HAL_xxx_Callback()` → `FromISR()` 계열 사용
디버깅	FreeRTOS+Trace, SEGGER RTT, Uart printf
실전 예제	UART 입력 → LED 제어, ADC 주기 샘플링

15.6 시뮬레이터/에뮬레이터 (QEMU, Keil, Proteus 등)

🧠 시뮬레이터 vs 에뮬레이터

구분	시뮬레이터 (Simulator)	에뮬레이터 (Emulator)
정의	하드웨어의 동작 *모델*을 소프트웨어로 흉내냄	실제 하드웨어의 *바이너리 레벨* 명령어 실행 환경 제공
실행 대상	코드의 논리적 흐름	바이너리, 펌웨어 이미지
정확성	논리/기능 중심	타이밍·하드웨어 중심
예시	Keil MDK Simulator	QEMU, Proteus, Renode, VirtualBox

🔧 1. QEMU (Quick EMUlator)

✅ 개요

항목	설명
용도	OS, 커널, Baremetal 코드 에뮬레이션
아키텍처	ARM, x86, RISC-V, MIPS 등 매우 다양
입력	바이너리(ELF), 디스크 이미지, 커널
특징	가상화 + 디바이스 에뮬레이션 가능

✅ 사용 예 (ARM Cortex-A)

```
qemu-system-arm -M versatilepb -kernel my.elf -nographic
```

✅ 사용 예 (RISC-V)

```
qemu-system-riscv32 -M sifive_e -kernel firmware.elf
```

→ UART, 타이머, Flash 등도 가상 에뮬레이션 가능

🛠️ 2. Keil MDK (uVision + ARM CMSIS 기반)

✅ 개요

항목	설명
타겟	Cortex-M 계열 (STM32, Nuvoton 등)
시뮬레이터	CPU + IO 핀 + 인터럽트 논리적 시뮬레이션
디버깅	레지스터, 변수, 메모리, 파형 추적
디바이스 데이터베이스	칩, Peripheral, 핀 배치 포함

✅ 주요 기능

- `Debug → Start/Stop Simulation`
- `Logic Analyzer`, `Call Stack`, `Watch`
- GPIO, ADC, Timer, NVIC 동작 **시뮬레이션** 가능

🧪 3. Proteus VSM (Virtual System Modeling)

✅ 개요

항목	설명
타겟	AVR, PIC, STM32, 8051 등
특징	**회로 시뮬레이션 + MCU 실행** 통합
입력	HEX, ELF, BIN 펌웨어 이미지

✅ 주요 기능

- LED, LCD, 모터, 센서와 **가상 연결된 하드웨어** 회로
- UART, I2C, SPI, ADC를 포함한 **시각적 시뮬레이션**
- PWM 파형, 오실로스코프 연동 가능
- FSM 분석, 디버깅 가능

🧩 시뮬레이터/에뮬레이터 비교 요약

도구	범용성	하드웨어 정확도	IO 연동	OS 실행	디버깅
QEMU	★★★★★	★★★☆☆	가능 (가상 UART 등)	YES (Linux, RTOS)	GDB 연동
Keil MDK	★★★☆☆	★★★★☆	일부 IO, 내부 시뮬	NO (OS X)	내장 디버거
Proteus	★★☆☆☆	★★★★★	회로도 기반	NO	시각적 디버깅

도구	범용성	하드웨어 정확도	IO 연동	OS 실행	디버깅
Renode	★★★★☆	★★★★☆	UART/GPIO 가상화	YES	CLI 기반
STM32CubeIDE	★★☆☆☆	★★★★☆	디바이스 일치 정확	NO	SWD, GDB 연동

📐 실습 시나리오 예시

시나리오	툴 추천
STM32 GPIO 테스트	Keil, Proteus
RTOS + UART + ADC + LED	QEMU (ARM Cortex-A or RISC-V)
센서 + 모터 회로 시뮬레이션	Proteus
커널/OS 디버깅	QEMU + GDB
STM32Cube HAL 기반 타이머 테스트	STM32CubeIDE + Keil Simulation

📌 요약 정리

항목	설명
QEMU	커널/RTOS 개발용 에뮬레이터, 디바이스 가상화
Keil MDK	STM32 등 Cortex-M MCU 시뮬레이터, 코드 추적에 강함
Proteus	회로 시뮬레이션 + MCU 코드 실행 통합
에뮬레이터 vs 시뮬레이터	타이밍 정확성, 하드웨어 동작 재현 가능성의 차이
실습 추천	센서/모터/LED 연동 테스트 + UART/ADC 코드 디버깅 환경 구축에 적합