14. 산업 시스템 통합

14.1 센서퓨전

- ## 초음파 + ToF + 수위센서 융합

1. 개요

정확한 수위 측정을 위해 단일 센서만 사용하는 것은 한계가 있다.
예를 들어, 초음파 센서는 온도·습도·거품 등에 민감하고,
ToF(VL53L0X)는 반사율·오염 등에 영향을 받는다.
정전식 수위센서는 직접 접촉식으로 안정적이지만,
탱크 재질·액체 전도도에 따라 보정이 필요하다.

따라서, **초음파(Ultrasonic)** + **ToF(Time-of-Flight)** + **정전식(Level Capacitance)**
3종 센서를 융합하면 다음과 같은 이점을 얻을 수 있다:

항목	초음파	ToF	정전식	융합 효과
측정 방식	비접촉 음파	IR 비접촉	접촉식	상호 보정 가능
장점	범용성, 저가	고정밀, 근거리	노이즈 저	신뢰도 향상
단점	온도·습도 영향	반사율 영향	탱크 구조 영향	복합 처리 필요
결합 목적	거리 보정	정밀 근거리	절대 수위 기준	전체 범위 커버

---

2. 시스템 구성

```
[Ultrasonic Sensor (HC-SR04)]
       ↓ TRIG/ECHO
[STM32 Timer Input Capture]
       ↓
[VL53L0X ToF Sensor]
       ↓ I²C (0x29)
[Capacitive Water Level Sensor]
       ↓ ADC 입력 (0~3.3V)
[FreeRTOS Task + Fusion Algorithm]
       ↓
[Filtered Level Output → OLED / MQTT]
```

---

3. 데이터 취득 절차

1. **초음파 거리 측정 (HC-SR04)**
   - TRIG 10μs 펄스 발생
   - ECHO 펄스폭 측정 → 거리(cm) 계산
   - `distance_ultra = echo_time * 0.017`

2. **ToF 거리 측정 (VL53L0X)**
    - `distance_tof = VL53L0X_ReadRange()`
    - 내부 시간기반 IR 반사 거리 (정밀도 ±3mm)
3. **정전식 수위 측정**
    - ADC로 0~3.3V 전압 측정
    - `level_cap = (adc_value / 4095.0) * MAX_HEIGHT`
    - 보정: `level_cap = offset + scale * raw_value`

---

4. 융합 알고리즘 (Sensor Fusion Logic)

(1) 가중 평균 방식

가장 단순한 융합 방법으로, 신뢰도(weight)에 따라 합성한다.

```
float fuse_level(float d_ultra, float d_tof, float d_cap)
{
    float w_ultra = 0.3;
    float w_tof   = 0.5;
    float w_cap   = 0.2;
    return (w_ultra*d_ultra + w_tof*d_tof + w_cap*d_cap);
}
```

- ToF: 근거리(0~40cm) 구간에서 높은 가중치
- Ultrasonic: 원거리(>40cm) 구간에서 높은 가중치
- Capacitance: 보정 및 안정화용

(2) 동적 신뢰도 조정

센서의 상태에 따라 가중치를 동적으로 조정.
예를 들어, 초음파가 “Out of Range”를 반환하면 w_ultra = 0 으로 조정.

```
if (ultra_invalid) w_ultra = 0;
if (tof_saturated) w_tof = 0;
if (cap_noise > 0.1) w_cap *= 0.5;
```

(3) Kalman Filter 융합

3개 센서를 모두 상태 추정에 사용.
측정 벡터: $z = [ultra, tof, cap]^T$
상태: 실제 수위 x
측정 방정식: $z = Hx + v$
예측 방정식: $x_k = x_{k-1} + w_k$

Kalman Gain K를 통해 노이즈 공분산에 따른 최적 융합 수행.

---

5. 보정 및 기준 정렬

초음파, ToF는 **탱크 상단 기준 거리**,
정전식은 **하단 기준 수위**를 나타내므로 기준 축을 맞춰야 한다.

```
tank_height = 200 mm

// 상단 거리 센서 → 수위로 변환
level_ultra = tank_height - distance_ultra
level_tof   = tank_height - distance_tof

// 정전식 → 수위 비율
level_cap   = (adc / max_adc) * tank_height
```

최종 융합 값:

```
level_final = fuse_level(level_ultra, level_tof, level_cap)
```

---

6. FreeRTOS Task 구조

Task	기능	주기
`UltrasonicTask`	HC-SR04 Trigger/Echo	100ms
`ToFTask`	VL53L0X 거리 읽기	100ms
`CapTask`	ADC 측정 및 필터링	100ms
`FusionTask`	데이터 융합, 평균, 보정	200ms
`DisplayTask`	OLED 출력	500ms
`CommTask`	MQTT 전송	1s

모든 Task는 Queue 또는 Shared Struct를 통해 `FusionTask`로 데이터를 전달한다.

---

7. 결과 출력 예시

항목	값	비고
초음파 거리	115.2 mm	TRIG/ECHO 측정
ToF 거리	117.8 mm	I²C
정전식	118.0 mm	ADC
최종 융합값	**117.2 mm**	Weighted Average

---

8. 예외 처리 및 보정

상황	조치
초음파 반사 없음	ToF + Capacitance만 사용
VL53L0X Timeout	초음파값 우선
정전식 과노이즈	Moving Average 보정
전체 오차 > ±5mm	재보정 요청 플래그

---

9. 시각화 및 로깅

- OLED 표시:

```
Lvl: 117.2 mm
[U:115][T:118][C:118]
```

- UART / MQTT 로그 전송

```
{"ultra":115.2, "tof":117.8, "cap":118.0, "final":117.2}
```

---

10. 확장

기능	설명
온도센서 연동	초음파 음속 보정 (v = 331.4 + 0.6T)
Self-Diagnosis	센서 이상 검출 및 비활성화
Flash Save	보정값 저장
Grafana 연동	클라우드 모니터링

결론

초음파, ToF, 정전식 수위센서를 병렬로 구성하고
가중 융합 / Kalman Filter 기반 알고리즘을 적용하면
각 센서의 단점을 상호 보완하여,
±1~2 mm 수준의 안정적 수위 측정이 가능하다.

이는 **산업용 수조, 정밀 급수 시스템, IoT 스마트탱크** 등에
적합한 고신뢰도 센서 융합 설계의 핵심 구조이다.

- # 신뢰도 기반 필터링

1. 개요

센서 융합(Fusion) 시스템에서 각 센서는 측정 환경, 노이즈, 하드웨어 특성에 따라
일정한 확률로 오차가 발생한다.
신뢰도 기반 필터링(Reliability-Weighted Filtering) 은
센서별로 실시간 평가된 "신뢰도 지수(Reliability Score)"를 이용해
데이터를 동적으로 가중 처리하는 기법이다.

즉,

> "정확도가 높을 때는 크게 반영하고, 불안정할 때는 영향력을 줄인다."

이는 고정 가중치 평균보다 **환경 적응성**이 뛰어나며,
센서 이상, 반사 오류, 잡음 등의 상황에서 결과 안정성을 보장한다.

---

2. 기본 구조

```
입력:  x₁, x₂, x₃ (센서값)
       r₁, r₂, r₃ (신뢰도)
출력:  x_fused = (r₁x₁ + r₂x₂ + r₃x₃) / (r₁ + r₂ + r₃)
```

- x_n: 각 센서 측정값 (초음파, ToF, 정전식 등)
- r_n: 실시간 계산된 신뢰도 (0.0~1.0)
- `x_fused`: 신뢰도 가중 평균 결과

---

3. 신뢰도 산출 인자

신뢰도는 여러 관찰 항목으로부터 계산된다.
각 센서의 상태를 지속적으로 모니터링하고,
오류나 이상치 발생 시 자동으로 `r` 값을 감소시킨다.

항목	계산식 / 기준	설명
Range Validity	측정값이 허용 범위 내인지	Ex. 0 < x < 200 mm
Temporal Stability	최근 샘플 편차	Δx < threshold
Signal Quality	센서별 내부 품질지표	ToF SPAD, 초음파 ECHO 성공률 등
Noise Level	Moving RMS 값	잡음이 많으면 감점
Update Rate	응답 지연 발생 여부	Timeout 시 0

신뢰도는 다음과 같이 종합할 수 있다.

```
r = w1*validity + w2*stability + w3*signal_quality + w4*rate;
```

4. 예시: 초음파 + ToF + 정전식

```
float r_ultra = 1.0;
float r_tof   = 1.0;
float r_cap   = 1.0;

// 1. 초음파 신뢰도 평가
if (echo_timeout)             r_ultra = 0.0;
else if (fabs(dx_ultra) > 5)  r_ultra *= 0.7;

// 2. ToF 신뢰도 평가
if (tof_status != OK)         r_tof = 0.0;
else if (signal_rate < 0.5)   r_tof *= 0.6;

// 3. 정전식 신뢰도 평가
if (adc_noise > 0.05)         r_cap *= 0.8;

// 4. 가중 융합
float fused = (r_ultra*ultra + r_tof*tof + r_cap*cap) /
              (r_ultra + r_tof + r_cap);
```

5. 안정화 필터 결합

단순 가중 평균만으로는 순간적인 신뢰도 변화로 인한 급격한 변동이 발생할 수 있다.
따라서 다음과 같은 후단 필터를 추가한다.

- **Exponential Smoothing (α=0.2~0.5)**

  ```
  fused_smooth = α*fused + (1-α)*prev;
  ```

- **Median Filter (3~5샘플)**
  잡음이 큰 환경에서 외란 제거.
- **Kalman Filter 결합형**
  신뢰도를 측정 노이즈 공분산 `R` 값에 반영.

  ```
  R = R_base / r
  ```

6. 시각적 개념

```
신뢰도 ↓ → 반영도 ↓
──────────────────────────
r = 1.0   → 100% 반영
r = 0.5   → 절반만 반영
r = 0.0   → 무시 (센서 제외)
```

센서가 일시적으로 이상 신호를 보낼 경우,
fused 값은 갑작스럽게 변하지 않고 이전 값 중심으로 유지된다.

---

7. 실시간 동작 예시

Time	Ultrasonic	ToF	Cap	r(U/T/C)	Fused(mm)
t_0	120.2	119.8	121.0	1.0 / 1.0 / 1.0	120.3
t_1	125.0	120.1	121.2	0.3 / 1.0 / 1.0	121.0
t_2	200.0	120.3	121.5	0.0 / 1.0 / 1.0	120.9
t_3	118.5	118.9	119.2	1.0 / 1.0 / 1.0	118.9

→ 초음파 오동작 시 자동 배제, 결과 안정 유지.

---

8. Fault Detection & Recovery

신뢰도 기반 필터링은 단순 평균이 아니라,
Fault-Tolerant 센서 관리 로직으로 확장할 수 있다.

```
if (r_sensor < 0.2) {
    sensor_state = FAULT;
    // 복구 조건 감시
    if (recover_ok) r_sensor = 1.0;
}
```

센서 하나가 완전히 고장나더라도 나머지 센서들이
정확한 값을 유지하며 시스템은 정상 동작한다.

---

9. FreeRTOS 기반 구현 구조

Task	역할
`SensorTask`	초음파 / ToF / 정전식 데이터 취득
`ReliabilityTask`	신뢰도 평가, r_n 계산
`FusionTask`	r-weighted 평균 + 후단 필터 적용
`DisplayTask`	OLED 출력 및 MQTT 송신

각 센서 데이터와 신뢰도는 구조체로 공유:

```c
typedef struct {
  float value;
  float reliability;
} SensorData_t;
```

---

10. 정리

단계	내용
①	센서별 신뢰도 평가 (r_n 계산)
②	r-weighted 평균 필터 적용
③	신뢰도 급변 시 완화 필터 추가
④	Fault 상태 자동 감지 및 복구
⑤	결과를 OLED, UART, MQTT로 전송

결론

신뢰도 기반 필터링은 단순 평균보다 훨씬 안정적인 센서 융합 방법으로,
실시간 적응형 가중치 조정을 통해 환경 변화나 노이즈에도
측정값의 신뢰성과 일관성을 크게 향상시킨다.

특히 초음파·ToF·정전식 복합 수위 측정 시스템에서
센서 고장, 반사 불량, 전기적 노이즈에 강인한
Fault-Tolerant Measurement Framework를 구성할 수 있다.

- # Adaptive Threshold 적용

1. 개요

센서 데이터나 신호처리 과정에서 고정 임계값(fixed threshold)을 사용하는 경우,
온도 변화, 조도, 전원 노이즈, 환경 변화 등 외부 요인에 따라
성능이 급격히 저하될 수 있다.

Adaptive Threshold (적응형 임계값) 은
입력 데이터의 통계적 특성을 실시간으로 반영하여
임계값을 동적으로 조정하는 방식이다.

즉,

> "환경이 변해도 자동으로 기준선을 조정하여 안정적인 검출을 유지한다."

---

2. 기본 원리

Adaptive Threshold는 다음 개념으로 표현된다.

```
Threshold(t) = μ(t) + k * σ(t)
```

- `μ(t)` : 최근 N개 샘플의 평균 (local mean)
- `σ(t)` : 표준편차 (local deviation)
- `k` : 감도 계수 (0.5 ~ 2.0, 상황에 따라 조정)

센서 노이즈가 커지면 `σ`가 증가하여 임계값이 완화되고,
노이즈가 작을 때는 민감도가 높아진다.

---

3. 적용 예시

(1) 초음파/ToF 거리 측정 안정화

환경 온도나 습도 변화로 측정값이 약간 흔들리는 경우,
Adaptive Threshold를 이용해 급격한 오차를 자동 배제.

```
#define WINDOW_SIZE 20
float window[WINDOW_SIZE];
int idx = 0;

float calcAdaptiveThreshold(float newVal) {
    window[idx++ % WINDOW_SIZE] = newVal;

    // 평균 계산
    float sum = 0.0f;
    for (int i=0; i<WINDOW_SIZE; i++) sum += window[i];
    float mean = sum / WINDOW_SIZE;

    // 표준편차 계산
    float var = 0.0f;
    for (int i=0; i<WINDOW_SIZE; i++) var += pow(window[i] - mean, 2);
    float std = sqrt(var / WINDOW_SIZE);

    // 임계값
    return mean + 1.5f * std;
}
```

→ 현재 측정값이 이 threshold를 초과하면 **비정상치(outlier)** 로 간주.

---

(2) ADC 기반 신호 검출 (예: 진동, 전류, 음향)

```
if (fabs(signal - baseline) > adaptiveThreshold)
    eventDetected = true;
```

이 방식은 노이즈가 큰 환경에서도
불필요한 오탐(False Positive)을 줄이는 효과가 있다.

---

4. Sliding Window 기반 구조

Adaptive Threshold는 **슬라이딩 윈도우(Sliding Window)** 방식으로 구현된다.
즉, 최근 N개의 데이터만 고려하여 평균과 분산을 계산한다.

시간	입력값	평균 μ	표준편차 σ	임계값 (μ + kσ)
t_0	101	101.0	0.0	101.0
t_1	102	101.5	0.7	102.5
t_2	100	101.0	0.8	102.2
t_3	150	113.2	22.4	146.8 ← 이상치 감지

---

5. Adaptive Baseline Tracking

일부 센서에서는 임계값뿐 아니라 기준선(baseline)도
시간에 따라 변동시켜야 한다.
예: 수위, 온도, 배경광, 전류 등 천천히 변하는 값.

```
baseline = α * newVal + (1 - α) * baseline;   // α = 0.01 ~ 0.1
```

→ baseline이 점진적으로 이동하면서 환경 변화에 적응.

---

6. FreeRTOS Task 예시

Task	역할
`SensorTask`	주기적으로 ADC/ToF 데이터 수집
`ThresholdTask`	μ, σ 계산 및 adaptive threshold 갱신
`DetectionTask`	임계값 기반 이벤트 감지
`LogTask`	검출 결과 기록 및 MQTT 송신

```
typedef struct {
  float mean;
  float stddev;
  float threshold;
} AdaptiveParams_t;
```

---

7. 파라미터 튜닝

변수	설명	권장값
`WINDOW_SIZE`	이동평균 계산 샘플 수	10~50
`k`	감도 조정 계수	1.0~2.0
`α`	baseline 추적 속도	0.01~0.1
`update_period`	threshold 갱신 주기	100~500 ms

---

8. 응용 분야

- 수위/압력/온도 센서에서의 **비정상 감지 (drift detection)**
- 모터 진동 신호의 **이상 패턴 검출**
- 조도/IR 센서 기반 **사람 감지 안정화**
- ADC 기반 **노이즈 환경 자동 적응**
- ToF/초음파 융합 시 **거짓 반사 제거**

---

9. 시각화 예시

```
측정값 ──────────────╮──────────────
평균 μ ──────────────┼──────────────
Adaptive Threshold ──╯──────────────
        ↑ 이상치 발생 → 자동 감쇠 후 복원
```

임계값이 자동으로 높아졌다가 안정 구간에 들어오면 다시 낮아짐.

---

10. 요약

항목	내용
목적	고정 임계값의 한계를 극복하고 환경 변화에 자동 적응
핵심 원리	μ + kσ 기반의 실시간 통계 계산
장점	노이즈 강건성, 오탐 방지, 실시간 반응성
단점	계산 부하 증가 (윈도우 크기 의존)
적용 대상	거리, 진동, 조도, 압력 등 아날로그/디지털 센서 전반

결론

Adaptive Threshold는 환경 변화나 노이즈에 강인한
자체 학습형 센서 기준선 알고리즘으로,
IoT·임베디드 실시간 계측 시스템의
자동 보정(Auto Calibration) 기반으로 매우 유용하다.

특히 초음파/ToF/수위 복합 시스템이나
진동 이상 감지 알고리즘에서 **신뢰도 기반 필터링**과 결합 시
거짓 이벤트를 대폭 감소시키고 안정적인 측정을 보장한다.

14.2 HMI 및 로깅

- # OLED / LCD 표시

1. 개요

센서 데이터, 시스템 상태, 또는 디버그 정보를 시각적으로 표현하기 위해
OLED(LCD 포함) 디스플레이를 사용한다.
STM32 MCU 환경에서는 **I²C 또는 SPI 인터페이스** 기반의 소형 모듈
(예: SSD1306, SH1106, HD44780, ST7735 등)을 주로 사용한다.

표시 장치는 MCU 내부의 데이터(센서 값, 시간, 알람 상태 등)를
직관적으로 모니터링할 수 있게 하며,
디버깅, 유지보수, 사용자 인터페이스(UI) 구현에 필수적이다.

---

2. 하드웨어 인터페이스

디스플레이	인터페이스	전원	특징
SSD1306 (OLED)	I²C (SDA/SCL)	3.3V	128×32 또는 128×64, 저전력, 고대비
HD44780 (LCD 16×2)	병렬 (4/8bit) 또는 I²C 확장	5V	문자 표시용, 저가형
ST7735 (TFT LCD)	SPI	3.3V	컬러 그래픽, 높은 표현력

---

3. 라이브러리 구조

- **SSD1306 / SH1106** :
  `Adafruit_SSD1306.h`, `Adafruit_GFX.h`
- **HD44780** :
  `LiquidCrystal_I2C.h`
- **ST7735** :
  `Adafruit_ST7735.h`, `Adafruit_GFX.h`

STM32 CubeIDE에서는 HAL 드라이버 기반 I²C/SPI 초기화 후
해당 라이브러리를 포팅하거나 직접 구현할 수 있다.

---

4. 초기화 절차

```
#include "Adafruit_SSD1306.h"

#define SCREEN_WIDTH 128
#define SCREEN_HEIGHT 64
Adafruit_SSD1306 display(SCREEN_WIDTH, SCREEN_HEIGHT, &hi2c1);

void OLED_Init(void)
{
    if(!display.begin(SSD1306_SWITCHCAPVCC, 0x3C)) {
        Error_Handler();  // 연결 실패 시
    }
    display.clearDisplay();
    display.setTextSize(1);
    display.setTextColor(SSD1306_WHITE);
    display.setCursor(0, 0);
    display.println("OLED Ready");
    display.display();
}
```

---

5. 실시간 데이터 표시

센서 데이터, RTC 시간, 시스템 상태를 주기적으로 갱신.

```
void OLED_Update(float distance, float weight, float level)
{
    display.clearDisplay();
    display.setCursor(0, 0);
    display.setTextSize(1);
    display.print("DIST: ");
    display.print(distance);
    display.println(" mm");

    display.print("LEVEL: ");
    display.print(level);
    display.println(" mm");

    display.print("WEIGHT: ");
    display.print(weight);
    display.println(" g");

    display.display();
}
```

→ `osDelay(500)` 주기로 갱신 Task 구성 시, FreeRTOS 환경에서도 안정적 표시 가능.

---

6. FreeRTOS Task 예시

```
void DisplayTask(void *argument)
{
    float distance, level, weight;

    OLED_Init();

    for(;;)
    {
        distance = Get_Distance();
        level = Get_Level();
        weight = HX711_GetValue();

        OLED_Update(distance, weight, level);
        osDelay(500);
    }
}
```

---

7. 텍스트 및 그래픽 출력

기능	설명	예시 코드
텍스트 크기 변경	가독성 향상	`display.setTextSize(2);`
그래픽 라인/사각형	UI 구분	`display.drawRect(0,0,128,16,SSD1306_WHITE);`
진행바	퍼센트 표시	`display.fillRect(0, 56, progress, 8, SSD1306_WHITE);`
아이콘 표시	배터리/와이파이 상태	사용자 정의 bitmap 사용

---

8. LCD 16×2 예시

```
#include "LiquidCrystal_I2C.h"

LiquidCrystal_I2C lcd(0x27, 16, 2);

void LCD_Init(void)
{
    lcd.init();
    lcd.backlight();
    lcd.setCursor(0,0);
    lcd.print("System Ready");
}

void LCD_Update(float distance)
```

```
{
    lcd.setCursor(0,1);
    lcd.print("Dist:");
    lcd.print(distance);
    lcd.print("mm ");
}
```

9. 상태 및 경보 표시

디스플레이는 측정 결과뿐 아니라
시스템 상태(정상, 오류, 알람, 슬립 등)를 시각적으로 표시하는 역할을 수행한다.

상태	표시 내용
정상	`"RUNNING..."`
측정 중	`"MEASURING..."`
오류	`"I2C ERROR"` 또는 `"SENSOR FAIL"`
슬립	`"SLEEP MODE"`
알람 발생	`"ALARM! PUMP ON"`

```
void OLED_ShowStatus(SystemState_t state)
{
    display.clearDisplay();
    display.setCursor(0, 0);
    display.setTextSize(2);

    switch(state) {
        case STATE_RUN: display.println("RUN"); break;
        case STATE_MEASURE: display.println("MEASURE"); break;
        case STATE_SLEEP: display.println("SLEEP"); break;
        case STATE_ERROR: display.println("ERROR"); break;
        case STATE_ALARM: display.println("ALARM"); break;
    }

    display.display();
}
```

10. 전력 최적화

- 디스플레이 OFF 시 전류 절감 (수십 mA → 수 μA 수준)
- `display.ssd1306_command(SSD1306_DISPLAYOFF);`
- 슬립 모드 진입 시 자동 화면 OFF
- RTC 알람 시 재점등

요약

항목	내용
목적	센서/시스템 상태의 시각적 피드백 제공
인터페이스	I²C 또는 SPI
주요 장치	SSD1306, HD44780, ST7735 등
주요 기능	텍스트/그래픽 표시, 상태 로그
FreeRTOS 연계	DisplayTask로 주기적 갱신
전력 관리	Sleep 시 화면 OFF, Wake 시 재활성화

결론

OLED/LCD는 임베디드 시스템의 사용자 피드백 인터페이스로,
센서 데이터, 상태 정보, 경보 등을 실시간으로 표시함으로써
시스템의 디버깅 효율과 사용자 접근성을 크게 향상시킨다.

FreeRTOS 환경에서는 `DisplayTask`를 별도로 두어
센서 Task로부터 Queue나 Message를 수신하여 표시하는 구조가 가장 안정적이다.

- # PC Serial Plotter 시각화

1. 개요

임베디드 시스템에서 측정된 센서 데이터를 **PC Serial Plotter**로 실시간 시각화하면,
디버깅과 튜닝 효율을 크게 향상시킬 수 있다.

STM32, ESP32, Arduino 등 모든 MCU는 UART를 통해
PC의 터미널 프로그램 또는 플로팅 도구로 데이터를 전송할 수 있다.

대표 도구:

- Arduino IDE Serial Plotter
- RealTerm / Tera Term (CSV 로그용)
- CoolTerm / PuTTY
- Python + Matplotlib / PySerial (커스텀 플로터)
- Visual Studio Code + SerialPlot 확장
- STM32CubeMonitor (ST 공식 시각화 툴)

---

2. 하드웨어 연결

항목	내용
MCU TX	USB-UART 모듈 RX (CH340, CP2102 등)

항목	내용
MCU RX	USB-UART 모듈 TX
GND	공통 접지
Baud Rate	일반적으로 **115200bps**
데이터 형식	ASCII 문자열 (`\r\n` 종결)

---

3. 출력 포맷 규칙

Serial Plotter는 여러 변수 값을 동시에 표시할 수 있으나,
데이터 형식이 일관되어야 한다.

(1) 단일 값 출력

```
printf("%d\n", sensorValue);
```

(2) 다중 값 (멀티채널)

```
printf("DIST:%d\tLEVEL:%d\tWEIGHT:%d\n", dist, level, weight);
```

또는

```
printf("%d,%d,%d\n", dist, level, weight);
```

(3) Arduino Plotter 호환 (공백 구분)

```
printf("%d %d %d\n", dist, level, weight);
```

→ 각각 “DIST”, “LEVEL”, “WEIGHT” 그래프가 별도 색으로 표시됨.

---

4. STM32 코드 예시

```
#include "stdio.h"

void UART_SendData(float distance, float level, float weight)
{
    printf("%.2f %.2f %.2f\r\n", distance, level, weight);
}
```

`printf()` 재정의

CubeIDE에서 `syscalls.c` 내 `_write()` 함수를 아래처럼 수정.

```
int _write(int file, char *ptr, int len)
{
    HAL_UART_Transmit(&huart2, (uint8_t*)ptr, len, HAL_MAX_DELAY);
    return len;
}
```

이후 `printf()`로 바로 UART 전송 가능.

---

5. FreeRTOS Task 구성

```
void PlotTask(void *argument)
{
    for(;;)
    {
        float dist = Get_Distance();
        float level = Get_Level();
        float weight = HX711_GetValue();

        printf("%.1f %.1f %.1f\r\n", dist, level, weight);
        osDelay(100);  // 10Hz 시각화 주기
    }
}
```

→ 각 Task에서 공유 데이터를 큐나 전역 변수로 전달받아 출력.

---

6. PC 측 설정

(1) Arduino Serial Plotter

- 보드 선택과 무관하게 UART 장치만 연결해도 동작
- 포트 선택 (COMx)
- Baud: 115200
- 데이터 구분자: 공백/탭
- 그래프는 자동 갱신

(2) STM32CubeMonitor

- ST 공식 툴로, `USART` 또는 `ST-LINK Virtual COM Port` 사용
- 변수 이름, 스케일, 색상 설정 가능
- 실시간 필터링/로깅 가능

(3) Python 커스텀 플로터

```
import serial
import matplotlib.pyplot as plt
from collections import deque

ser = serial.Serial('COM5', 115200)
window = 200
data1, data2, data3 = deque(maxlen=window), deque(maxlen=window),
deque(maxlen=window)

plt.ion()
fig, ax = plt.subplots()

while True:
    line = ser.readline().decode().strip()
    vals = line.split()
    if len(vals) == 3:
        d, l, w = map(float, vals)
        data1.append(d)
        data2.append(l)
        data3.append(w)
        ax.clear()
        ax.plot(data1, label="Distance")
        ax.plot(data2, label="Level")
        ax.plot(data3, label="Weight")
        ax.legend()
        plt.pause(0.01)
```

→ 실시간으로 3개 센서 곡선을 동시에 표시.

---

7. 데이터 스무딩 및 표시 주기

- UART 전송 주기를 너무 짧게 설정하면 (예: <10ms) 버퍼 오버플로 발생 가능
- 일반적으로 50~200ms 간격 권장
- 이동평균 필터나 지수평활(EMA) 적용 시 그래프가 안정적

---

8. CSV 로그 저장

```
printf("%lu,%.2f,%.2f,%.2f\r\n", HAL_GetTick(), dist, level, weight);
```

PC에서 `Tera Term` → **File** → **Log** → **CSV 저장**
이후 Excel, Grafana, Python 등으로 후처리 가능.

---

9. 시각화 예시

```
거리(mm): ──────┐──────────────
수위(mm):  ─────┼──────────────
무게(g):   ─────┘──────┐──────
                       ↑ 이벤트 발생 시 피크
```

센서 이상, 충돌, 펌프 동작 등 시점별 그래프 변화로 즉시 확인 가능.

---

10. 요약

항목	내용
목적	실시간 센서 데이터 시각화 및 디버깅
통신 방식	UART → USB-Serial
출력 형식	ASCII 문자열 (공백/콤마 구분)
주요 도구	Arduino Serial Plotter, STM32CubeMonitor, Python
표시 주기	50~200ms (5~20Hz)
장점	실시간 파형 분석, 오탐/오류 검출, 튜닝 효율 향상

결론

PC Serial Plotter는 MCU 내부의 실시간 데이터를
그래프로 시각화하는 가장 단순하면서 강력한 방법이다.

UART 포트를 통해 센서, PID 제어, 필터링, 스케줄링 등의 동작을
직관적으로 확인할 수 있으며, FreeRTOS 기반 시스템의 성능 검증 및
센서 융합 알고리즘의 튜닝 과정에 필수적인 도구로 활용된다.

- # 데이터로그 기반 분석

1. 개요

데이터로깅(Data Logging)은 **센서나 시스템 상태를 일정 주기로 저장**하여,
시간에 따른 동작 특성을 분석하거나 문제의 원인을 추적하기 위한 핵심 기술이다.

STM32 시스템에서는 주로 **SD 카드(FATFS)**, **EEPROM**, **내부 Flash**, 또는 **UART → PC 로그 전송**을 통해 데이터를 기록한다.
로그된 데이터는 **엑셀, Python, Grafana, MATLAB** 등에서 분석하여
센서 정확도, 시스템 안정성, 제어 알고리즘 성능을 평가하는 데 사용된다.

---

2. 데이터로깅의 목적

목적	설명
성능 분석	PID 제어 응답, FreeRTOS Task 주기성 등 검증
신뢰성 평가	장기 동작 중 센서 드리프트, 오차율 확인
고장 진단	예외(Fault) 발생 전후 시스템 상태 추적
환경 변화 기록	온도, 수위, 전압 등 시계열 데이터 수집
모델링 / 보정	실측 데이터를 이용한 보정곡선 생성

---

3. 데이터 구조 설계

효율적 분석을 위해 로그 데이터는 **CSV 형식**으로 구성한다.
각 항목은 **타임스탬프 + 측정값**으로 구성하며, 필요 시 이벤트 로그를 포함한다.

예시 구조:

```
Time(ms),Distance(mm),WaterLevel(mm),Weight(g),Temp(C),PumpState
0,120.3,80.0,150.2,24.6,OFF
1000,121.0,81.2,150.1,24.7,OFF
2000,119.8,80.7,149.9,24.7,ON
...
```

기본 포맷 설계 규칙

- `,`(콤마)로 구분 (Excel/Grafana 호환)
- `\r\n`으로 행 구분
- 주기: 1~10초(일반 로그), 100~500ms(실시간 계측)
- 단위 표기를 헤더에 명시

---

4. STM32 구현 예시

(1) FATFS 기반 SD카드 로깅

```
FIL file;
char buffer[64];

f_open(&file, "log.csv", FA_OPEN_APPEND | FA_WRITE);
sprintf(buffer, "%lu,%.2f,%.2f,%.2f\r\n",
        HAL_GetTick(), dist, level, weight);
f_write(&file, buffer, strlen(buffer), NULL);
f_close(&file);
```

→ 주기적 Task에서 호출하거나 RTC 알람 이벤트 시 기록.

(2) FreeRTOS Task 구조

```
void LogTask(void *argument)
{
    for(;;)
    {
        LogData_t data = GetCurrentData();
        WriteLogToSD(data);
        osDelay(1000); // 1Hz 기록
    }
}
```

(3) EEPROM / Flash 로그 (소형 시스템)

저용량 MCU에서는 최근 N회 데이터만 저장.
 순환 버퍼 구조(Ring Buffer) 사용.

---

5. 분석 워크플로

1. **데이터 수집** — MCU에서 SD카드 또는 UART로 로그 생성
2. **전송 및 수집** — CSV 파일을 PC로 복사
3. **후처리 및 시각화**
    - Excel: 그래프/통계
    - Python: Pandas + Matplotlib
    - Grafana: 실시간 대시보드
4. **결과 해석 및 모델 개선**

---

6. Python 분석 예시

```
import pandas as pd
import matplotlib.pyplot as plt

df = pd.read_csv("log.csv")
plt.figure(figsize=(10,5))
plt.plot(df["Time(ms)"]/1000, df["WaterLevel(mm)"], label="Water Level")
plt.plot(df["Time(ms)"]/1000, df["Distance(mm)"], label="Ultrasonic")
plt.plot(df["Time(ms)"]/1000, df["Weight(g)"], label="Weight")
plt.xlabel("Time (s)")
plt.ylabel("Sensor Values")
plt.legend()
plt.grid()
plt.show()
```

→ 시간에 따른 센서 추이, 오차, 응답속도를 직관적으로 확인 가능.

---

7. Grafana 연동

- CSV를 InfluxDB 또는 SQLite로 변환
- Grafana Dashboard에서 실시간 차트 구성
- PID 출력, 펌프 상태, 수위 변화 등 **산업 모니터링 시각화**에 적합

---

8. 주요 분석 항목

항목	분석 내용
센서 드리프트	장시간 측정 시 기준값 편차
FreeRTOS Task 지연	로그 타임스탬프 간격 비교
PID 제어 응답	목표 수위 대비 오버슈트/언더슈트
전원 변동	저전압 시 데이터 불안정 구간 검출
오류 발생 패턴	Fault 로그와 이벤트 타임라인 연계

---

9. 성능 및 최적화

- SD카드 접근 시 DMA 사용 (FATFS + SDIO/DMA)
- 버퍼링(`f_sync()` 주기 조정)으로 Flash 마모 최소화
- 실시간 로그는 전용 Task로 분리하여 CPU 부하 분산
- 필요 시 압축(RLE, Delta Encoding) 적용

---

10. 요약

구분	내용
저장 포맷	CSV (Comma-Separated Values)
저장 위치	SD카드, EEPROM, Flash, UART
로깅 주기	100ms~10s
활용 도구	Excel, Python, Grafana
분석 목적	센서 오차 검증, 제어 안정성, Fault 추적

결론

데이터로깅은 단순 기록 기능이 아니라,
시스템 신뢰성을 객관적으로 검증하고 알고리즘을 개선하기 위한 분석 기반이다.

STM32 환경에서 FATFS와 FreeRTOS를 결합하면
장기 운용 중에도 안정적인 데이터 수집과 고해상도 시간 분석이 가능하다.

이 과정을 통해 **센서 보정, 제어 최적화, 고장 예측** 등의 고급 단계로 발전할 수 있다.

14.3 산업 안정성

- # 보호회로 (TVS, ESD, 퓨즈)

1. 개요

전자 회로는 외부 환경(정전기, 서지, 역전압 등)에 의해 쉽게 손상될 수 있으므로,
보호소자(Protection Devices)를 설계 초기에 포함하는 것이 필수적이다.

STM32나 ESP32 같은 MCU 기반 회로는 특히 **I/O 핀, 통신 포트, 전원 라인**이 민감하기 때문에
TVS 다이오드, ESD 보호소자, 퓨즈(Resettable Fuse) 등의 회로적 보호 장치가 필요하다.

---

2. 보호회로의 목적

보호대상	주요 위험요소	보호방법
전원 입력 (5V, 12V 등)	과전압, 역전류, 서지	퓨즈, TVS, 다이오드
통신라인 (UART, USB, RS485 등)	정전기, 낙뢰 서지	ESD Suppressor, TVS 다이오드
아날로그 입력 (센서 신호)	고전압 유입, 노이즈	RC 필터, 클램프 다이오드
MCU GPIO	ESD, 노이즈	시리즈 저항, TVS, 보호 다이오드

---

3. TVS 다이오드 (Transient Voltage Suppressor)

TVS 다이오드는 낙뢰나 전원 스파이크처럼 짧은 순간의 고전압을 빠르게 클램핑(clamping)하여
MCU나 IC를 보호하는 장치다.

(1) 특징

- 동작 속도: 수 나노초(ns)
- 양극/단극형 존재 (Uni/Bidirectional)
- 서지 흡수 후 자동 복귀

(2) 기본 연결 예시

```
5V ----+---- 회로 입력
       |
      [TVS]
       |
      GND
```

- 전원라인: `SMBJ5.0A`, `SMBJ12A` 등 선택
- 데이터라인: `ESD9B5.0ST5G`, `SRV05-4` (USB/UART용)

(3) 주의사항

- **클램핑 전압(Vc)**이 시스템 최대 동작전압보다 충분히 높아야 함
- **패키지**: SMB/SMC (전원용), SOT-23 (신호라인용)

---

4. ESD 보호소자

ESD(정전기 방전)은 인체나 환경에 의해 수 kV~수십 kV의 순간 전압이 발생하는 현상으로,
UART, I²C, USB 등의 노출 핀을 손상시킬 수 있다.

(1) 일반적인 보호 구조

```
UART_TX ----┬---- 회로 입력
            |
         [ESD Suppressor]
            |
           GND
```

(2) 선택 기준

구분	일반값
동작전압	3.3V, 5V
클램핑전압	< 12V
용량(Cj)	< 5pF (고속신호용)

- 저용량 소자 사용 (USB, SPI 등 고속라인)
- 예시: `TPD2E001`, `PESD5V0S1UL`, `SRV05-4`

---

5. 퓨즈 (Fuse)

과전류나 단락(short) 시 회로를 물리적으로 차단하는 보호장치.
보통 전원 입력부(USB VBUS, DC IN)에 배치한다.

(1) 종류

종류	특징
일회용 퓨즈	과전류 시 영구 단선, 교체 필요
리셋형 퓨즈 (PTC)	온도에 따라 저항이 증가, 냉각 시 복구됨
전자식 퓨즈 (eFuse)	IC 기반, 정밀 보호/제한 기능 제공

(2) 선택 예시

입력전압	퓨즈 전류	소자 예시
5V USB	0.5A ~ 1A	`MF-R050`, `MF-R100`
12V DC	1A ~ 3A	`MF-R200`, `MF-R300`

---

6. 보호회로 구성 예시

(1) 전원 입력 보호

```
DC_IN ---- [PTC Fuse] ---- [TVS Diode] ----> 5V Regulator ----> MCU
```

- 퓨즈: 과전류 보호
- TVS: 서지 흡수
- 역전압 방지 다이오드(Shottky): 잘못된 극성 방지

(2) 통신포트 보호

```
UART_RX/TX ---- [22Ω 직렬저항] ---- [TVS or ESD 다이오드] ----> MCU
```

- 시리즈 저항: 전류 제한
- TVS: 서지 차단
- PCB 상 GND 근처 배치 (최소 루프)

(3) 아날로그 입력 보호

```
Sensor Input ---- [10kΩ] ----+---- MCU ADC_IN
                             |
                           [TVS]
                             |
                           GND
```

---

7. PCB 설계 시 주의사항

항목	권장 설계
ESD 루프 최소화	보호소자를 커넥터 근처에 배치
GND Plane 확보	보호소자 접지 경로를 짧고 굵게
전류 용량 고려	TVS/퓨즈의 전류 허용치 확인
노이즈 분리	보호소자와 신호라인 간 커플링 최소화

---

8. 권장 부품 요약

용도	추천 소자	비고
5V/12V 전원 서지 보호	SMBJ5.0A, SMBJ12A	전원라인용 TVS
UART/RS485 라인	SRV05-4, PESD5V0S1UL	다채널 ESD
USB 포트	TPD4EUSB30	초저용량 TVS
전원 입력 과전류 보호	MF-R100, MF-R200	PTC 퓨즈
역전압 보호	SS14, MBR120	쇼트키 다이오드

---

9. 실습 예시

- STM32 보드 DC-IN (12V) 입력부에 `PTC Fuse + SMBJ12A` 추가
- UART 라인에 `SRV05-4` 삽입
- SD카드 3.3V 전원라인에 `SMBJ5.0A` 삽입
- 실험: 정전기 방전(ESD Gun) 및 전원 서지 테스트

---

10. 요약

구분	기능	설치 위치
TVS 다이오드	서지 전압 클램핑	전원, 통신라인
ESD 보호소자	정전기 방전 보호	포트/커넥터 근처
퓨즈 (PTC)	과전류 차단	전원 입력부
역전압 다이오드	극성 반전 보호	전원 입력부
시리즈 저항	전류 제한 및 ESD 완화	GPIO/신호라인

결론

보호회로는 회로 설계의 **보험**과 같다.
평상시에는 동작하지 않지만, 이상 상황에서는 시스템을 지켜주는 **마지막 방어선**이다.

STM32나 ESP32 같은 MCU 보드의 내구성과 신뢰성을 높이려면
TVS + ESD + 퓨즈를 조합한 보호설계가 반드시 포함되어야 한다.

- # 노이즈 차단 (LC 필터)

1. 개요

전자 시스템에서 발생하는 전원 및 신호 노이즈는 MCU 오동작, 센서 데이터 불안정, 통신 오류의 주요 원인이다.
이러한 노이즈를 억제하기 위해 **LC 필터**(Inductor + Capacitor)를 사용하여 전원 및 신호 라인에 **저역통과(Low-Pass) 특성**을 형성한다.

LC 필터는 단순 RC 필터보다 **고주파 감쇠 성능**이 우수하며, 전원 노이즈 제거(Power Line Filtering) 및 EMC 대책에 필수적으로 사용된다.

---

2. LC 필터의 기본 원리

LC 필터는 **인덕터(L)**가 고주파 성분에 대해 높은 임피던스를 제공하고,
커패시터(C)가 고주파 성분을 GND로 바이패스하여 노이즈를 제거한다.

(1) 저역통과 필터(Low-Pass Filter)

```
Vin ──L──┬── Vout
         |
         C
         |
        GND
```

- 저주파(유효신호)는 통과
- 고주파(노이즈)는 차단 (GND로 우회)

(2) 전달함수

$$f_c = \frac{1}{2\pi\sqrt{LC}}$$

- `f_c` : 차단 주파수 (Hz)
- `L` : 인덕턴스 (H)
- `C` : 커패시턴스 (F)

---

3. LC 필터의 동작 특성

주파수 영역	인덕터(L)	커패시터(C)	결과
저주파 (유효신호)	낮은 임피던스	높은 임피던스	신호 통과

주파수 영역	인덕터(L)	커패시터(C)	결과
고주파 (노이즈)	높은 임피던스	낮은 임피던스	노이즈 억제

---

4. LC 필터 구성 방식

(1) π형(파이형) 필터

```
Vin ──C──L──C── Vout
```

- 전원 라인 노이즈 억제에 사용
- 커패시터는 고주파 성분을 GND로 흘려보내고, 인덕터는 서지 억제

(2) T형 필터

```
Vin ──L──C──L── Vout
```

- 강력한 고주파 차단용
- 고전류 라인에서는 권장되지 않음 (전압 강하 발생 가능)

(3) 단일 LC 필터

```
Vin ──L──C── Vout
```

- 소형 회로, MCU 전원핀 근처에 배치
- 가장 일반적 구성

---

5. 설계 지침

(1) 차단 주파수 설정

- MCU 및 센서 회로용 전원 필터: `f_c ≈ 10~50kHz`
- 오디오/ADC 라인: `f_c ≈ 1~5kHz`
- 고속 디지털 전원 (3.3V/1.8V): `f_c ≈ 1~10MHz`

(2) L, C 선정

용도	L	C	비고
MCU 전원	10μH	10μF	일반적인 저노이즈 필터
아날로그 센서	22μH	4.7μF	고감도 입력용
디지털 전원	1μH	100nF	고속 스위칭 전원용

(3) ESR, DCR 고려

- 커패시터 ESR은 0.1Ω 이하로 선택
- 인덕터 DCR은 가능한 한 낮게 (<100mΩ)

---

6. PCB 설계 시 고려사항

항목	권장 설계
배치 위치	전원 입력단 또는 노이즈 민감 부품 근처
GND Plane 확보	커패시터 GND는 저임피던스 경로로 연결
루프 최소화	LC 경로의 루프 면적을 최소화 (EMI 저감)
페라이트 비드(Ferrite Bead)	고주파 노이즈 차단용으로 LC 대체 가능

---

7. 실무 적용 예시

(1) MCU 전원 필터링

```
+5V ---- [Ferrite Bead 100Ω@100MHz] ---- [10μF + 0.1μF] ----> MCU VDD
```

- 고주파 필터링: 페라이트 비드
- 저주파 노이즈 제거: 병렬 커패시터

(2) 센서 전원 분리

```
+3.3V ---- [10μH] ---- [4.7μF] ----> Analog Sensor VCC
```

- 아날로그 센서와 MCU 전원 분리
- 센서측 노이즈 감도 저하

(3) ADC 입력 신호 필터

```
Signal ---- [1kΩ] ----+----> ADC_IN
                      |
                     C(1nF)
                      |
                     GND
```

- 단순 RC 필터로 ADC 샘플링 안정화
- 고주파 외란 억제

---

8. LC vs RC vs Ferrite Bead 비교

구분	구조	장점	단점
LC 필터	인덕터 + 커패시터	넓은 주파수 감쇠, 효율적	부피, 가격 증가
RC 필터	저항 + 커패시터	단순, 저가	전압 강하, 감쇠 약함
Ferrite Bead	고주파 흡수재	초고주파 제거에 효과적	낮은 주파수 노이즈에는 약함

---

9. 실제 설계 팁

- 커패시터는 **멀티 레이어 세라믹(Murata, TDK)** 사용
- 인덕터는 **전류 허용치 ≥ 회로 소비전류 × 1.5배**
- π형 필터 사용 시 GND 패턴은 **짧고 굵게**
- 전원 필터링 후에도 노이즈가 남을 경우, 추가 RC 필터 또는 페라이트 비드 병행

---

10. 요약

목적	권장 구성	비고
MCU 전원 안정화	10μH + 10μF	기본 전원 필터
아날로그 센서 노이즈 억제	22μH + 4.7μF	감도 보장
디지털 라인 고주파 차단	Ferrite Bead + 0.1μF	EMI 억제
ADC 신호 안정화	1kΩ + 1nF	RC 필터 병행 가능

결론

LC 필터는 **전원 안정성과 신호 정확도 확보의 핵심 요소**이다.
단일 커패시터만으로 해결되지 않는 노이즈 문제를 해결하며,
특히 **MCU + 센서 복합 시스템**에서 **EMI/EMC 성능 향상**에 큰 역할을 한다.

올바른 L, C 조합과 최적의 배치 설계는 시스템 신뢰성을 크게 향상시킨다.

전원 안정화 및 EMC 대응

1. 개요

임베디드 시스템의 안정성과 신뢰성은 **전원 품질(Power Integrity)**에 크게 의존한다.
전원 공급이 불안정하거나 외부 전자파(EMI)에 취약할 경우, MCU 리셋, 센서 오동작, 통신 오류가 발생할 수 있다.

따라서 **전원 안정화(Power Stabilization)**와 **EMC(Electromagnetic Compatibility)** 대응 설계는 모든 회로의 기초이며,
특히 STM32, ESP32, 센서, 통신 모듈 등이 혼재된 시스템에서는 필수적인 요소이다.

---

2. 전원 안정화의 핵심 목표

항목	설명
전압 안정성	부하 변동 시에도 일정한 전압 유지
리플 억제	SMPS, DC-DC 변환기 등에서 발생하는 잔류 리플 제거
순간 과도응답 대응	부하 급변 시 전압 강하 방지
잡음 차단	전원선으로 유입되는 고주파 성분 억제
역전류 및 서지 보호	전원 입력 시 외부 서지, 역극성으로부터 보호

---

3. 전원 안정화 구성 블록

(1) 입력단 보호 및 필터링

```
DC_IN (5V/12V)
   |
 [퓨즈] – [TVS 다이오드]
   |
 [LC 필터 / 페라이트 비드]
   |
 LDO or DC-DC Converter
   |
 MCU / Sensor / Module
```

- **퓨즈(PTC)**: 과전류 보호
- **TVS 다이오드**: 전압 서지 흡수
- **LC 필터 or 비드**: 고주파 노이즈 억제

(2) 레귤레이터 안정화

- **LDO (Low Dropout Regulator)**: 저잡음 아날로그 회로에 적합
- **DC-DC (Buck/Boost Converter)**: 고효율 대전류 회로에 적합

출력단 필터링:

```
10μF (탄탈/세라믹) + 0.1μF (MLCC)
```

- 10μF: 저주파 리플 제거
- 0.1μF: 고주파 노이즈 제거

---

4. 디커플링(Decoupling) 설계

MCU, 센서, 통신 칩 주변에는 **디커플링 커패시터**를 반드시 배치해야 한다.

(1) 역할

- 전류 스텝 응답 보상
- 전원 임피던스 저감
- 노이즈 바이패스

(2) 배치 원칙

항목	설계 지침
커패시터 용량	0.1μF ~ 1μF (각 전원핀 근처)
병렬 구성	0.1μF + 4.7μF + 10μF (다중 대역 보상)
배치 위치	MCU 전원핀과 최대한 근접
GND 경로	짧고 넓은 접지면 확보

(3) 예시

```
VDD ──┬─── MCU
      |
    [0.1μF]
      |
     GND
```

- 0.1μF: 고속 스위칭 노이즈 억제
- 4.7μF: 저주파 전원 안정화

---

5. EMC (Electromagnetic Compatibility) 대응

EMC는 **EMI(방출)**와 **EMS(내성)**의 두 측면을 모두 포함한다.
회로는 외부 간섭을 받지 않으면서, 자신 또한 불필요한 전자파를 방출하지 않아야 한다.

(1) EMC 구성 요소

구분	목적	대응 방식
공통모드 노이즈 억제	케이블을 통한 방출 억제	커먼모드 초크, 페라이트 비드
차동모드 노이즈 억제	전원 및 신호선 내 잡음	LC 필터, 커패시터 바이패스
ESD 내성 향상	정전기 방전 대응	TVS, ESD Suppressor
방출 저감	스위칭 전류 루프 최소화	GND Plane, 라우팅 최적화

---

6. PCB 설계 EMC 고려사항

(1) GND Plane 설계

- 전원과 GND는 가능한 **면(Plane)**으로 설계
- 신호 루프는 GND 위로 흐르도록 배치
- 고전류 경로와 민감 신호 분리

(2) 신호 라우팅

항목	설계 원칙
고속신호 (SPI, UART)	짧고 직선으로
아날로그 신호	디지털 라인과 분리
클록라인	GND Plane 위를 통과하도록
커넥터 근처	ESD 보호소자 추가

(3) 전원 분리

- 디지털/아날로그 전원은 별도 LC 필터로 분리
- ADC 기준 전압(VDDA)은 **전용 필터 및 디커플링** 적용

---

7. 실무 예시

(1) STM32 시스템 예시

회로구성	소자	비고
12V DC IN	PTC 퓨즈 + SMBJ12A	과전류/서지 보호
12V → 5V	DC-DC Step Down (LM2596)	전원 변환
5V → 3.3V	LDO (AMS1117-3.3)	저잡음 변환
3.3V 라인	Ferrite Bead + 10μF + 0.1μF	노이즈 차단
MCU/센서	각 VDD 핀별 0.1μF	디커플링
UART 포트	ESD 다이오드	외부 정전기 보호

(2) 전원 노이즈 측정

- 오실로스코프로 VDD 라인 확인
- 리플 전압 50mV 이하 유지
- 부하 급변(모터, 펌프 구동 시)에도 전압 안정성 유지

---

8. EMC 인증 대응 팁

항목	권장 대책
Radiated Emission	LC 필터, Shield Ground, Snubber 회로
Conducted Emission	Line Filter, Ferrite Core
ESD Immunity	ESD Suppressor, 접지 강화
EFT/Burst	TVS, RC Snubber
Surge	MOV, Gas Tube (대형 전원라인)

---

9. 권장 부품

용도	권장 소자	비고
전원 필터링	BLM21PG331SN1 (비드), 10μH 인덕터	고주파 노이즈 억제
입력 보호	SMBJ5.0A, MF-R100	서지/과전류
레귤레이터	AMS1117, MIC5504	저잡음 LDO
디커플링	0.1μF (X7R), 10μF (X5R)	전원라인 병렬 구성
커먼모드 초크	ACM2012D-900-2P	통신라인 노이즈 억제

---

10. 요약

항목	주요 내용
전원 안정화	LC 필터, 디커플링, LDO 조합으로 안정 전압 유지
EMC 방출 억제	GND Plane, 페라이트 비드, 시그널 라우팅 최적화
EMC 내성 강화	TVS, ESD 보호소자, 필터 적용
측정 검증	리플/EMI 측정으로 실효성 확인

결론

전원 안정화와 EMC 대응은 하드웨어 설계의 “보이지 않는 품질”이다.
기능보다 중요한 것은 **노이즈 환경에서도 안정적으로 동작하는 회로**를 만드는 것이다.

- 입력단 보호 → LC 필터링 → 레귤레이터 안정화 → 디커플링
- PCB 설계 시 전원 루프 최소화 및 GND Plane 확보
- ESD/서지/리플 대비 보호소자 조합

이 일련의 과정이 완성될 때, 시스템은 **산업 환경에서도 신뢰할 수 있는 수준**의 내성을 확보하게 된다.