1주차 보고서:
STM32F413ZHTx 기반 PMIC 시리얼 통신 인터페이스 설계

최용진

2025.07.18

목차

서론

1주차 업무 개요

"STM32F413ZHTx 기반 PMIC 시리얼 통신 인터페이스 개발"

1주차는 **STM32F413ZHTx** 마이크로컨트롤러를 사용하여 5ms 주기로 동작하는 브레이크 시스템의 새로운 **PMIC(Power Management IC)**와 데이터 통신을 구현하는 것을 목표로 한다. 시스템에 남은 통신 포트가 1개이며 고속 통신이 요구되지 않는 제약 조건 하에서, **SPI, I2C, CAN** 등 다양한 시리얼 통신 방식의 특징을 분석하여 최적의 프로토콜을 선정한다.

또한, 데이터 송수신을 위해 **Polling, Interrupt, DMA** 방식을 각각 비교 분석하고, 특히 Interrupt 및 DMA 방식의 핵심 요소인 **IRQ, ISR, Callback**의 동작 원리를 파악한다.

최종적으로 가장 적합할 것으로 예상되는 I2C 통신을 기준으로, 각 데이터 처리 방식에 따른 송신 시퀀스를 Flowchart로 시각화하여 안정적인 통신 시스템의 기반을 마련하고자 한다.

1주차 업무 요약

단계	주요 내용	세부 목표
1. 통신 프로토콜 선정	SPI, I2C, CAN 프로토콜 분석	- 각 통신 방식의 구동 원리, 프로토콜, 회로적 특징 파악 - 장단점 비교 분석 - 프로젝트 요구사항(포트 1개, 저속)에 가장 적합한 통신 방식 선정
2. 데이터 처리 방식 분석	Polling, Interrupt, DMA 방식 파악	- Polling, Interrupt, DMA의 개념 및 CPU 부하 관점에서의 동작 원리 이해 - Interrupt 및 DMA 방식에서 IRQ, ISR, Callback 동작 원리 파악
3. I2C 통신 시퀀스 파악	I2C TX 데이터 통신 시퀀스 시각화	- Polling 방식의 I2C 통신 흐름도 작성 - Interrupt 방식의 I2C 통신 흐름도 작성 - DMA 방식의 I2C 통신 흐름도 작성

1. 시리얼 통신 프로토콜 분석 및 선정

1.1. 시리얼 통신 개요

시리얼 통신은 데이터를 한 번에 한 비트씩 순차적으로 보내는 통신 방식이다. 데이터 비트들이 하나의 통신선을 따라 줄지어 전송된다.

이는 여러 개의 통신선으로 데이터 묶음(일반적으로 8비트)을 한 번에 보내는 **병렬 통신**과 대조된다.

- **시리얼 통신 특징**
    - 단 하나의 데이터 선(또는 송/수신을 위한 두 가닥)만으로 통신이 가능해 **회로 구성이 간단**하고 **비용이 저렴하다.**
    - 병렬 통신에 비해 신호 간 간섭이나 왜곡 문제가 적어 **상대적으로 먼 거리까지 안정적인 데이터 전송**이 가능하다.
- **동기화 방식**: 시리얼 통신에는 송신측과 수신측이 데이터를 주고받는 타이밍을 맞추는 '동기화'가 필수적이다.
    - **비동기(Asynchronous) 통신**: 별도의 클럭(Clock) 신호 없이, 데이터의 시작과 끝을 알리는 **Start/Stop 비트**를 사용해 타이밍을 맞춘다.
    - **동기(Synchronous) 통신**: 송신측과 수신측이 공유하는 **클럭 신호**를 기준으로 정확한 타이밍에 데이터를 주고받는다.

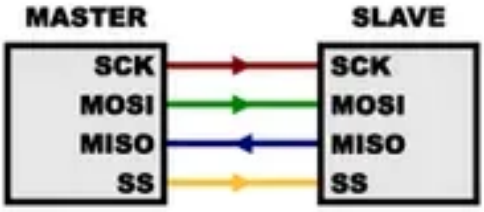
1.2. SPI (Serial Peripheral Interface) 분석

SPI는 **마스터-슬레이브** 구조를 기반으로 하는 **동기식, Full-duplex** 통신 방식이다.

1.2.1. 회로적 특징 및 신호선

SPI는 주로 **4개의 신호선**을 사용하는 4-wire bus 구조가 가장 큰 회로적 특징이다.

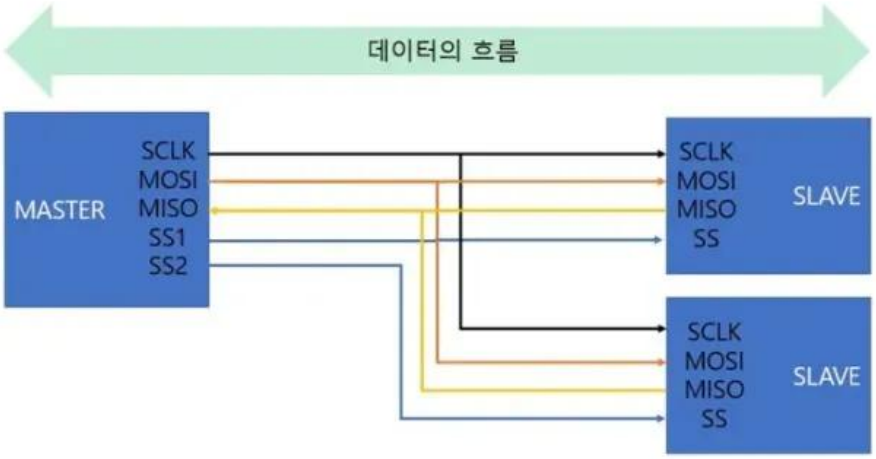
여러 슬레이브를 연결할 때, 클럭(**SCK**)과 데이터(**MOSI, MISO**) 라인은 공유하고 각 슬레이브는 개별적인 **SS** 라인으로 제어된다.



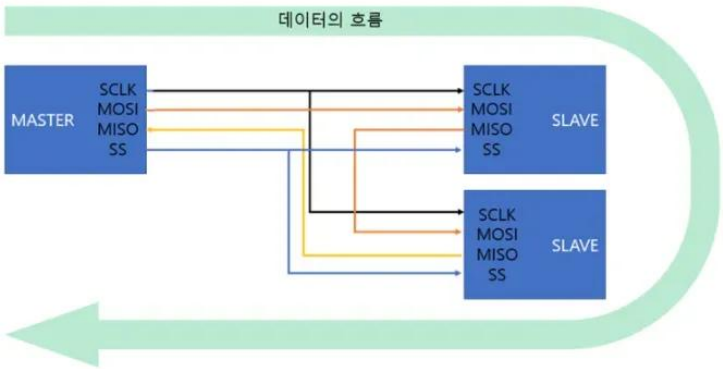
I2C와 달리 신호선이 명확하게 High/Low로 구동되므로 **별도의 풀업 저항이 필요가 없다.**

모든 장치는 마스터가 생성하는 클럭과 데이터 라인을 공유하지만, 각 슬레이브는 개별적인 SS 라인을 통해 제어된다.

따라서 **슬레이브 장치가 많아질수록 마스터의 SS 핀 소모가 늘어나는 단점**이 있다.



하지만 Daisy Chain 방식을 사용해 여러 개의 슬레이브를 직렬로 연결하여, 마스터의 핀 사용을 단 하나만 사용할 수 있다.



Daisy Chain 방식은 장점이 있는만큼 단점도 있기 때문에 상황에 맞게 사용해야 한다.

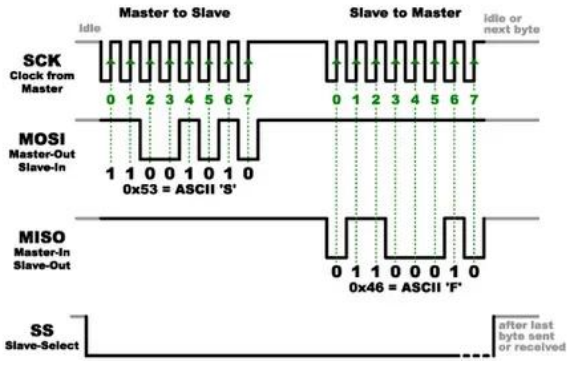
장점	단점
마스터의 핀을 하나만 사용	특정 슬레이브 하나와 통신하기 위해 마스터 사이에 껴있는 모든 슬레이브와 통신해야 함 → 속도 느려짐
배선이 직렬로 이어져 회로 구성 쉬움	모든 슬레이브가 동시에 활성화 되므로 특정 슬레이브만 독립적으로 제어 불가
	중간 슬레이브가 고장 나면 그 뒤에 연결된 모든 슬레이브와의 통신 두절됨

1.2.2. 통신 구동 방식 및 프로토콜

SPI 통신은 마스터가 모든 것을 주도하며, 송수신이 동시에 이루어지는 Full-duplex 방식으로 동작한다.

데이터 전송과정:

1. **슬레이브 선택**: 마스터는 통신하고자 하는 특정 슬레이브의 **SS(CS)** 핀을 **Low** 상태로 만든다.
2. **클럭 생성 및 데이터 교환**:
    - 마스터는 **SCK** 핀을 통해 클럭 신호를 발생시킨다.
    - 클럭의 한 주기마다, 마스터는 **MOSI** 라인을 통해 1비트를 슬레이브로 전송한다.
    - **동시에**, 슬레이브는 **MISO** 라인을 통해 1비트를 마스터로 전송합니다. 이 데이터 교환은 양쪽의 **Shift Register**를 통해 이루어진다.
3. **통신 종료**: 정해진 수의 비트(보통 1바이트) 전송이 완료되면, 마스터는 **SS** 핀을 다시 **High** 상태로 만들어 통신을 종료한다.



1.3. I2C (Inter-Integrated Circuit) 분석

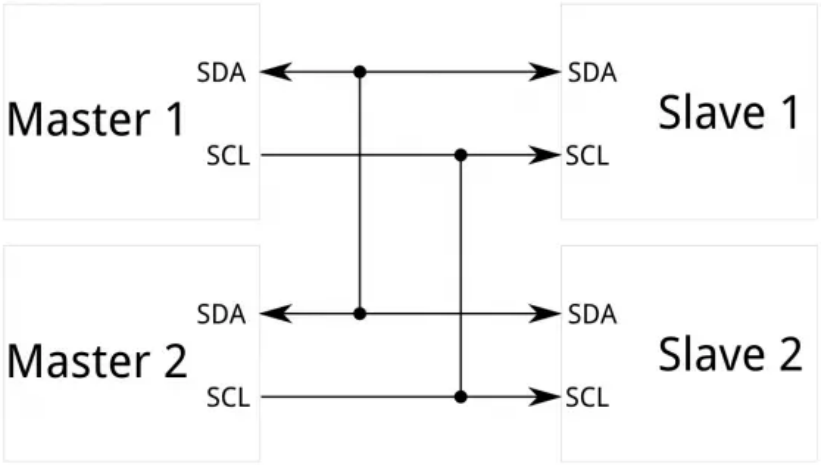
I2C(Inter-Integrated Circuit)는 **단 두 개의 선**이 필요하며 Half-duplex 방식으로 여러 장치가 통신할 수 있도록 만든 **동기식 직렬 통신** 방식이다. 칩과 칩 사이의 근거리 통신(on-board)에 널리 사용된다.

1.3.1. 회로적 특징 및 신호선

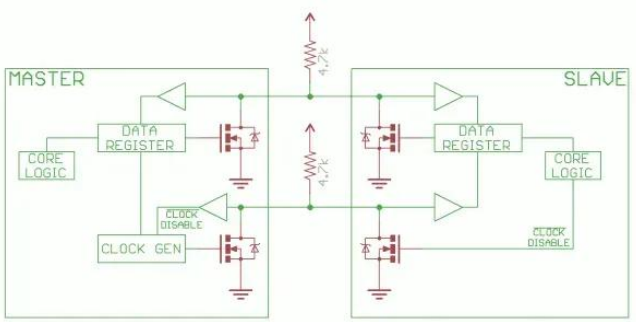
I2C의 가장 큰 회로적 특징은 **오픈-드레인(Open-Drain)** 방식과 **풀업 저항**을 사용한다는 점이다.

이 덕분에 여러 장치가 물리적인 충돌 없이 버스를 공유할 수 있다.

- **신호선 (2-Wire)**
    - **SCL (Serial Clock)**: 통신 타이밍을 동기화하기 위한 **클럭** 라인이다. 마스터 장치가 생성한다.
    - **SDA (Serial Data)**: 실제 **데이터**가 오고 가는 양방향 라인이다.



- **회로 구성**
    - **오픈-드레인**: I2C 버스에 연결된 장치는 SCL과 SDA 라인을 **Low 상태(GND에 연결)로만 만들 수 있습니다.** High 상태로 직접 구동할 수는 없다.
    - **풀업 저항**: 회로의 외부에서 SCL과 SDA 라인을 각각 **VCC(전원)에 저항으로 연결한다**. 이 저항 덕분에, 버스를 사용하는 장치가 없을 때(모든 장치가 라인을 놓아줄 때) 라인의 상태는 자연스럽게 **High**가 된다.



1.3.2. 통신 구동 방식 및 프로토콜

I2C는 **마스터-슬레이브 구조**를 가지며, 모든 통신은 마스터가 시작하고 제어한다.

슬레이브(Slave)는 고유한 Address를 통해 식별된다.

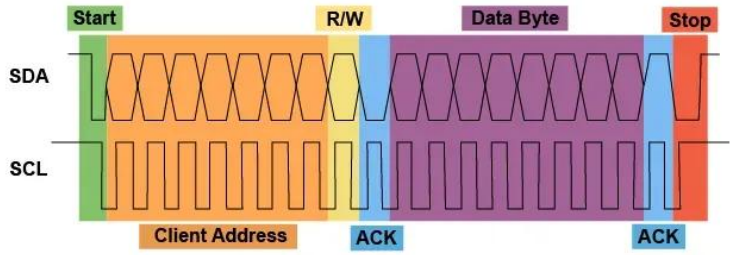
동작순서:

1. START Condition
    - 마스터가 통신의 시작을 알리는 신호
    - **SCL 라인이 High**인 상태에서 **SDA 라인을 High에서 Low로** 내림
2. Address Frame
    - 마스터는 통신하고자 하는 **슬레이브의 7비트 주소**와 데이터 방향을 나타내는 **1비트 Read/Write 신호**를 SDA 라인으로 전송 (0: 쓰기, 1: 읽기)
    - 버스에 연결된 모든 슬레이브 중 자신의 주소와 일치하는 슬레이브만 응답 준비
3. Acknowledge
    - 주소나 데이터를 1바이트 수신한 쪽은 잘 받았다는 의미로 9번째 클럭에 **SDA 라인을 Low로** 당겨 응답
    - 이 신호가 없으면(NACK) 통신에 문제가 있음을 의미
4. Data Frame
    - 슬레이브가 주소에 대해 ACK로 응답하면, R/W 비트에 따라 마스터와 슬레이브

사이에 1바이트 단위의 데이터가 오고 감

- 각 데이터 바이트 전송 후에는 항상 ACK 신호가 뒤따름

5. STOP Condition
    - 마스터가 통신의 종료를 알리는 신호
    - **SCL 라인이 High**인 상태에서 **SDA 라인을 Low에서 High로** 올
    - 이 신호 이후 버스는 다시 다른 장치가 사용할 수 있는 대기 상태가 됨



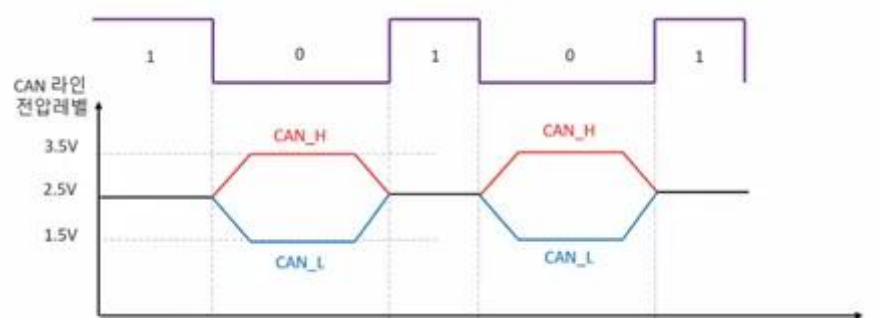
1.4. CAN (Controller Area Network) 분석

CAN은 본래 차량 내 전자제어장치(ECU) 간의 통신을 위해 개발된 방식으로, **매우 높은 신뢰성**과 **노이즈에 강한 특성**을 가진 **비동기 직렬 통신**이다.

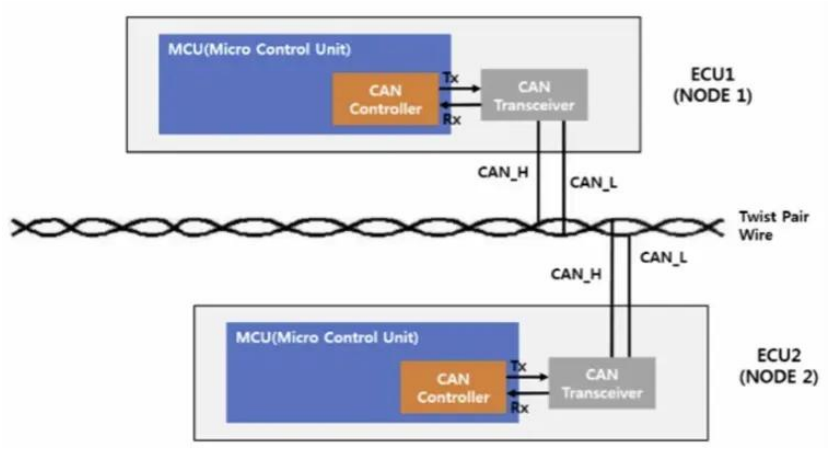
1.4.1. 회로적 특징 및 신호선

CAN의 가장 큰 회로적 특징은 **차동 신호(Differential Signaling)** 방식을 사용하여 외부 노이즈의 영향을 최소화하는 것이다.

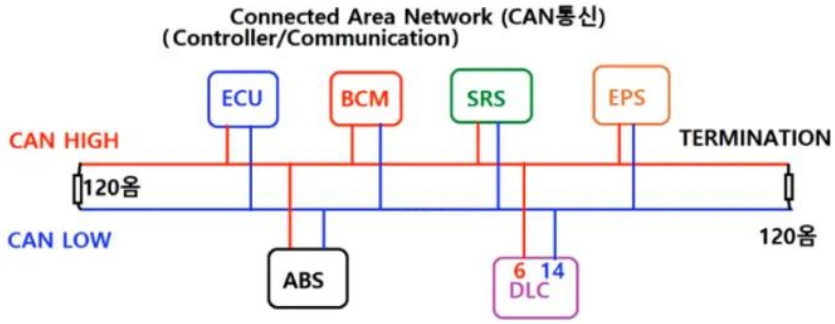
- **신호선 (2-Wire)**
    - 2개의 선 (CAN H/CAN L)만 필요하며 두 신호의 전위차를 통해 데이터를 전송함



- 2개의 선이 Twist Pair로 되어 있어 노이즈에 강함



- **회로 구성**
  - **버스형 토폴로지**: 모든 CAN 제어기(노드)는 두 개의 메인 버스 라인(CAN_H, CAN_L)에 병렬로 연결된다.
  - **종단 저항**: 버스의 양쪽 끝에는 신호 반사를 막고 안정적인 통신을 위해 **120Ω(옴)의 종단 저항**을 반드시 연결해야 한다. (메시지 리셋 역할)



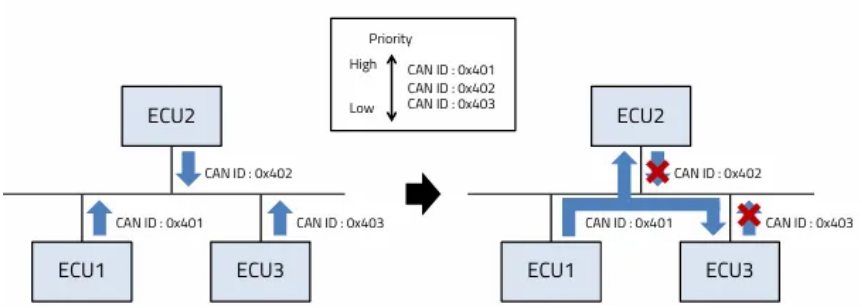
1.4.2. 통신 구동 방식 및 프로토콜

CAN은 정해진 마스터-슬레이브 관계가 없는 **Multi-Master** 방식이며, **메시지 기반**으로 동작한다.

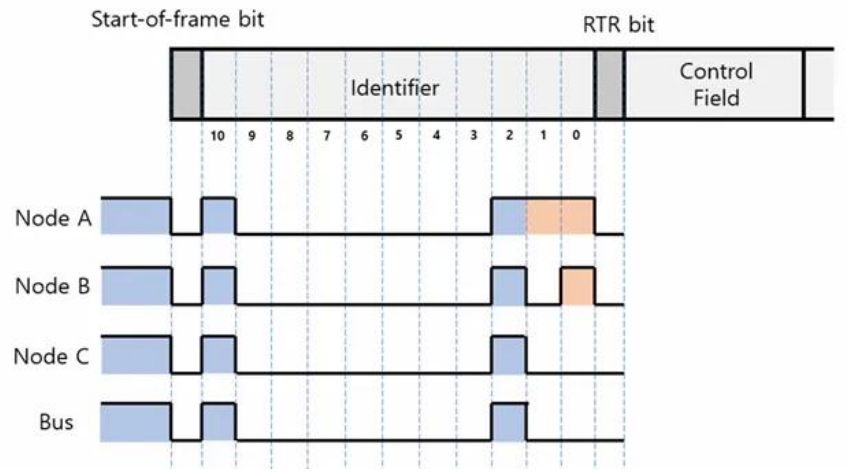
즉, 주소로 대상을 지정하는 것이 아니라, 모든 노드가 버스에 메시지를 Broadcast하면 필요한 노드가 그 메시지를 가져가는 방식이다

CAN의 Arbitration 동작

1. **메시지 기반 통신**: 모든 메시지에는 고유한 ID가 포함되어 있다. 이 ID는 메시지의 종류와 **우선순위**를 결정한다. **ID 값이 낮을수록 우선순위가 높다.**



2. **Dominant(0) & Recessive(1) 비트**:
    - **Dominant 0**: CAN_H와 CAN_L 사이에 명확한 전압 차이가 있는 상태
    - **Recessive 1**: 두 선의 전압 차이가 거의 없는 상태
    - 만약 한 노드가 Dominant(0)를 보내고 다른 노드가 Recessive(1)를 동시에 보내면, 버스는 **Dominant 상태**가 된다.
3. **Arbitration 예시 (11페이지 그림)**
    1. 통신을 원하는 노드(ID: A<B<C)는 먼저 버스가 비어있는지(Recessive 상태) 확인
    2. 여러 노드(A, B, C)가 동시에 통신을 시작하면, 각자 자신의 메시지 ID를 1비트씩 버스에 전송하며 동시에 버스 상태를 읽음
    3. 만약 자신이 Recessive(1)를 보냈는데 버스에서 Dominant(0)가 감지되면, 자신보다 더 높은 우선순위(낮은 ID 값)를 가진 메시지가 있다는 것을 인지하고 즉시 전송을 멈추고 수신 상태로 전환
    4. 이 경쟁에서 끝까지 살아남은, 즉 가장 ID가 낮은(우선순위가 높은) 노드만이 데이터 손상 없이 메시지 전송을 계속 진행

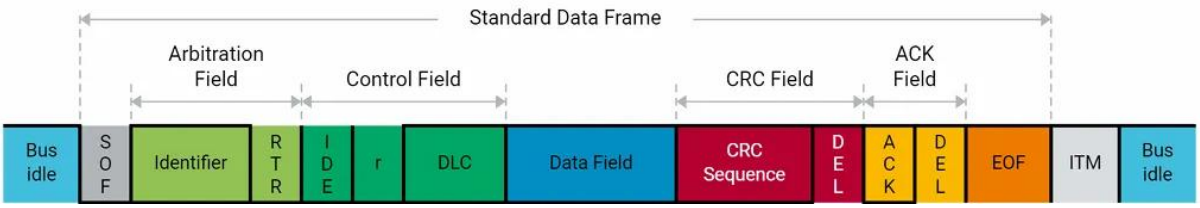


이러한 Arbitration 방식 덕분에 CAN 통신은 여러 노드가 동시에 메시지를 보내려 해도 **충돌이 발생하지 않고** 가장 중요한 메시지가 항상 먼저 전송되는 것을 보장한다.

CAN 메시지 프레임

CAN 통신은 단순히 ID와 데이터만 보내는 것이 아니라, 정해진 규칙에 따라 데이터를 포장해서 프레임 단위로 보낸다. 다음은 가장 일반적으로 사용되는 Standard Data Frame이다.

- SOF (Start Of Frame) : CAN Frame의 시작 비트로 'dominant 0'의 값을 가짐
- Identifier (ID) : CAN Frame의 ID이며 값이 작을수록 높은 우선순위를 가짐
- RTR (Remote Transmission Request) : 다른 노드에게 특정 데이터를 요청할 때 사용함 (Remote Frame)
- IDE (Identifier Extension) : ID의 확장 유무를 나타냄
- DLC (Data Length Code) : 4 Bits 크기로 데이터의 크기를 나타냄
- Data : 전송하고자 하는 데이터이며 CAN Signal을 포함함 (CAN은 최대 8 Bytes, CAN FD는 최대 64 Bytes)
- CRC (Cyclic Redundancy Check) : 데이터의 무결성 검사에 사용됨
- ACK (Acknowledge) : 전송된 CAN Frame이 수신되었는지를 나타냄
- EOF (End Of Frame) : CAN Frame의 끝을 나타냄



1.5. 통신 방식 비교 및 선정

프로토콜	통신 방식	구조	배선/라인	특징
SPI	동기식 통신	마스터-슬레이브	4-wire (MOSI, MISO, SCLK, SS)	단순하고 빠른 직렬 링크
I²C	동기식 통신	멀티-마스터 / 멀티-슬레이브	2-wire (SDA, SCL)	주소 기반 버스, 다중 장치 연결 용이
CAN	시리얼 버스(메시지 프레임)	버스형(메시지 기반 전송)	2-wire (CAN_H, CAN_L)	**노이즈에 매우 강하고 신뢰성 높음**

- 주어진 상황(남은 포트 1개, 저속, 브레이크 시스템)을 고려할 때, 핀 효율성 측면에서는 **I2C**
- 시스템의 신뢰성/안정성 측면에서는 **CAN**이 유리함
- I2C, CAN 중 최종 선택은 PMIC가 어떤 인터페이스를 지원하는지에 따라 결정

2. 데이터 처리 방식 분석 (Polling, Interrupt, DMA)

CPU가 Peripheral와 데이터를 주고받는 방식은 시스템의 효율성을 결정하는 중요한 요소이다.

어떤 방식을 사용하느냐에 따라 CPU의 부담이 크게 달라지며, 주로 **Polling**, **Interrupt**, **DMA** 세 가지 방식으로 나뉜다.

2.1. Polling 방식

폴링은 **CPU가 모든 작업을 주관**하는 가장 간단한 데이터 처리 방식이다.

CPU는 주변장치에 명령을 내린 후, 해당 작업이 완료되었는지 나타내는 **상태 레지스터의 특정 플래그(Flag)를 무한 루프(Loop)를 돌며 계속 확인한다**.

플래그가 원하는 상태로 바뀔 때까지 CPU는 다른 일을 하지 못하고 대기하며, 이를 Busy-waiting라고 한다.

2.2. Interrupt 방식

주변장치가 일이 끝나면 CPU에게 알려주는 이벤트 기반 방식이다. CPU는 주변장치에 작업을 지시한 후 즉시 다른 작업을 수행한다.

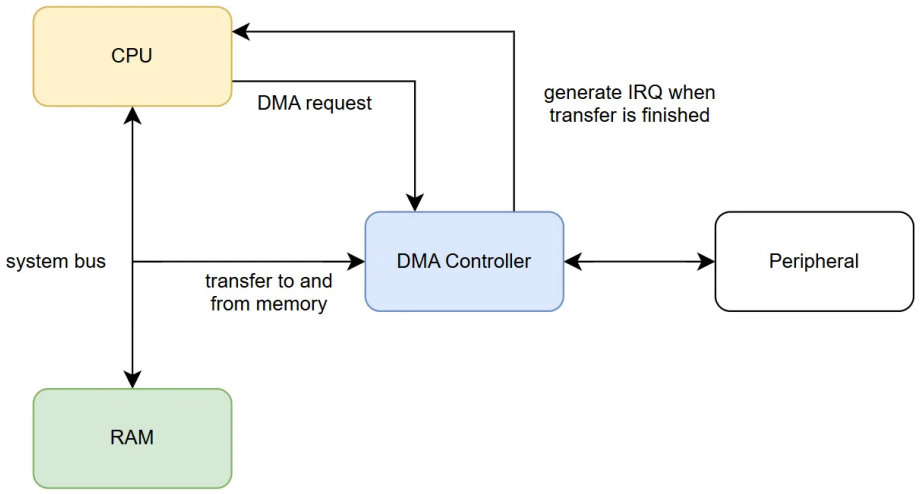
주변장치는 작업을 완료하면 CPU에 **인터럽트 요청(IRQ)** 신호를 보낸다. CPU는 현재 하던 일을 잠시 멈추고, 미리 정의된 **인터럽트 서비스 루틴(ISR)**을 실행하여 요청된 데이터를 처리한 후, 원래 하던 작업으로 복귀한다.

2.3. DMA 방식

DMA (Direct Memory Access)는 CPU의 개입 없이 하드웨어인 **DMA 컨트롤러**가 메모리와 주변 장치 간의 데이터 전송을 직접 처리하는 방식이다.

CPU는 DMA 컨트롤러에게 전송할 데이터의 시작 주소, 목적지 주소, 데이터 크기만 알려주고 전송 시작을 request 한다.

그 후 CPU는 다른 작업에 완전히 집중할 수 있고, 데이터 전송이 모두 완료되면 DMA 컨트롤러가 CPU에게 **인터럽트를 한 번만 발생**시켜 작업 완료를 알려준다.



2.4. IRQ, ISR, Callback

IRQ (Interrupt Request):

> **물리적인 전기 신호**로, 주변장치가 CPU에게 이벤트가 발생했음을 알리는 하드웨어적인 요청이다.

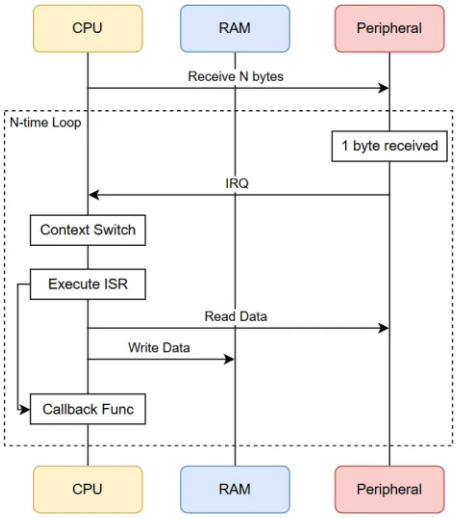
ISR (Interrupt Service Routine):

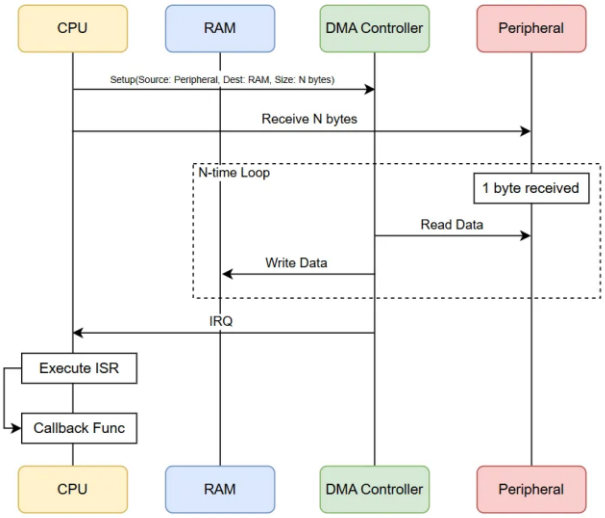
> 특정 IRQ에 대응하여 실행되도록 약속된 함수이다. 이 함수의 메모리 주소는 시스템 시작 시 Vector Table이라는 주소록에 미리 등록되어 있다.

Callback Function:

> ISR에 의해 호출되는, 개발자가 실제 로직을 구현하는 공간이다.

아래 그림은 각각 일반 인터럽트 방식과 DMA 방식에서 주변장치가 N바이트의 데이터를 수신하는 경우를 나타낸다.





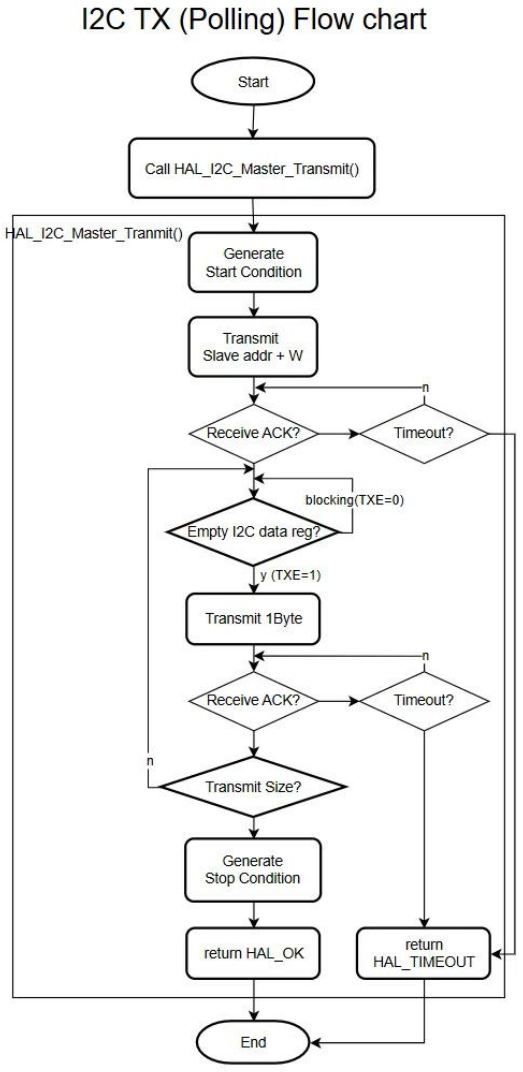
가장 중요한 차이점은 **누가, 그리고 언제 IRQ를 발생시키는가**에 있다.

- **일반 인터럽트**는 **주변장치**가 **매 작은 단위의 일이 끝날 때마다** CPU를 호출하는 방식이다.
- **DMA**는 **DMA 컨트롤러**가 **요청된 큰 덩어리의 일이 모두 끝났을 때 딱 한 번만** CPU를 호출하는 방식이다.

결과적으로 개발자가 최종적으로 사용하는 **콜백 함수는 동일할 수 있지만**, 그 콜백 함수가 호출되기까지의 중간 과정(IRQ의 출처와 ISR의 종류)이 다르며, 이 차이가 두 방식의 CPU 효율성에서 결정적인 차이를 만들어낸다.

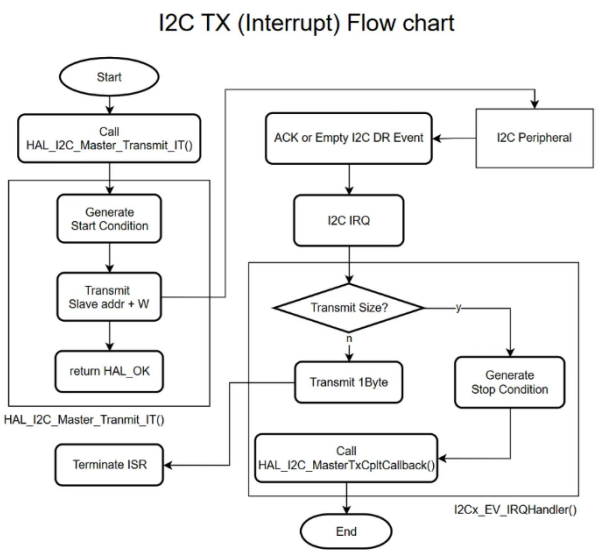
3. STM32 HAL 라이브러리 기반 I2C 통신 시퀀스

3.1. Polling 방식 데이터 송신 Flow Chart



1. HAL_I2C_Master_Transmit() 함수 호출
2. **통신 시작**
   - **START 조건**을 버스에 생성
   - **슬레이브 주소 + Write 비트**를 전송
3. **주소 ACK 확인 (Polling)**
   - ACK를 받을때까지 (ADDR 플래그) 대기
   - Timeout 시간 내에 플래그가 설정되지 않으면 오류(HAL_TIMEOUT)를 반환하고 종료
4. **데이터 전송 루프 (Size 만큼 반복)**
   1. I2C 데이터 레지스터(DR)가 비어있는지 (**TXE 플래그**) 대기 (Polling)
   2. **1바이트 전송**: 데이터 레지스터가 비면, 사용자의 txBuffer에서 1바이트를 읽어와 I2C 데이터 레지스터(DR)에 write
   3. 방금 보낸 1바이트의 전송이 완료되고 ACK를 받았는지(**BTF 플래그**) 대기
5. 루프가 끝나 모든 데이터 전송이 완료되면, HAL 라이브러리가 **STOP 조건**을 버스에 생성
6. 모든 과정이 성공적으로 끝나면 함수는 HAL_OK를 반환하고 종료

3.2. Interrupt 방식 데이터 송신 Flow Chart



1. **(main)** HAL_I2C_Master_Transmit_IT() 함수를 호출

2. **(main)** HAL 드라이버는 I2C 통신에 필요한 인터럽트를 활성화하고, **START 조건**과 **슬레이브 주소** 전송을 시작시킨 뒤, **즉시 HAL_OK를 반환**

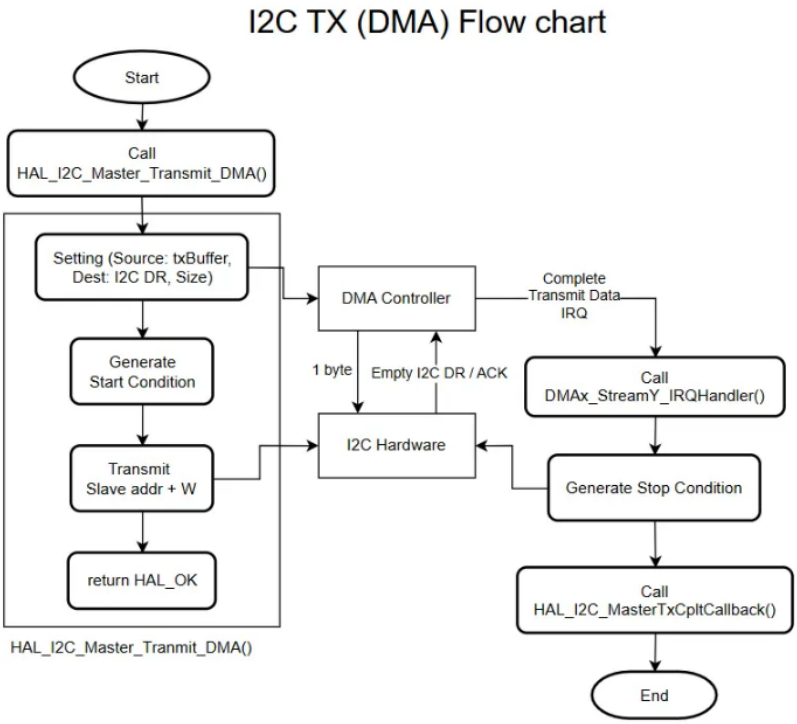
3. 슬레이브로부터 ACK를 받거나, 데이터 레지스터가 비는 등 하드웨어적인 **I2C 이벤트가 발생**하면, **IRQ** 신호가 발생

4. CPU가 **ISR** 진입

5. ISR 내부에서는 아직 전송할 데이터가 남아있는지 확인
    - **남아있다면**: 버퍼에서 다음 1바이트를 읽어와 I2C 데이터 레지스터에 써준 뒤, ISR을 종료하고 원래 하던 작업으로 복귀 (데이터가 모두 전송될 때까지 반복)
    - **모두 전송했다면**: **STOP 조건**을 생성하여 통신을 종료하고, 관련 인터럽트를 비활성화

6. **HAL_I2C_MasterTxCpltCallback() 콜백 함수를 호출**하고 ISR을 종료

3.3. DMA 방식 데이터 송신 Flow Chart



1. **함수 호출 및 설정 (main)**
    - HAL_I2C_Master_Transmit_DMA() 함수 호출
    - **DMA 컨트롤러**에 전송 정보를 설정 (소스: RAM의 txBuffer, 목적지: I2C의 데이터 레지스터(DR), 전송 크기: Size)
2. **I2C 통신 시작 및 CPU 해제 (main)**
    - I2C 하드웨어에 명령하여 버스에 **START 조건**을 보낸 뒤, **슬레이브 주소 + Write 비트**를 전송
    - 함수는 즉시 HAL_OK를 반환
3. **첫 데이터 전송 (하드웨어 자동 처리)**
    - 슬레이브가 주소에 응답하여 **ACK** 신호를 전송
    - I2C 하드웨어는 주소 전송이 완료되어 데이터 레지스터가 비었음을 인지하고, **DMA 컨트롤러**에 **DMA 요청**
    - **DMA 컨트롤러**는 이 요청에 응답하여, RAM에 있던 txBuffer의 첫 번째 바이트를 I2C 데이터 레지스터로 **CPU 개입 없이** 직접 전송
4. **데이터 전송 자동 반복 (하드웨어 자동 처리)**

- I2C 하드웨어는 데이터 레지스터에 채워진 1바이트를 버스로 전송하고, 슬레이브로부터 **ACK** 받음
- 데이터 전송이 끝나 데이터 레지스터가 다시 비면, I2C 하드웨어는 또다시 **DMA 요청**
- 이 과정이 설정된 Size만큼 자동으로 반복

5. **전송 완료 및 IRQ 발생**
   - CPU의 인터럽트 컨트롤러(NVIC)에 **IRQ (인터럽트 요청) 신호를 단 한 번** 보냄
6. **ISR 실행 및 통신 종료**
   - DMA용 ISR을 실행
   - ISR 내부에서 전송 완료 확인
   - **STOP 조건**을 보내 통신을 완전히 종료
7. **콜백 함수 호출**
   - 통신 종료 후, ISR은 최종적으로 HAL_I2C_MasterTxCpltCallback() **콜백 함수** 호출