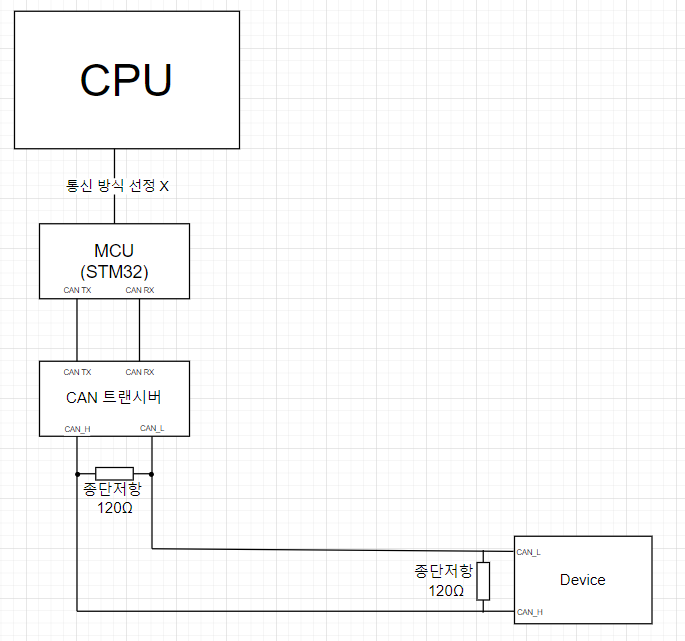
**CAN 통신**

**H/W 및 구성**

CAN 통신의 경우 장치 수에 관계없이 CAN\_H, CAN\_L 의 선 2개만 필요하게 된다.

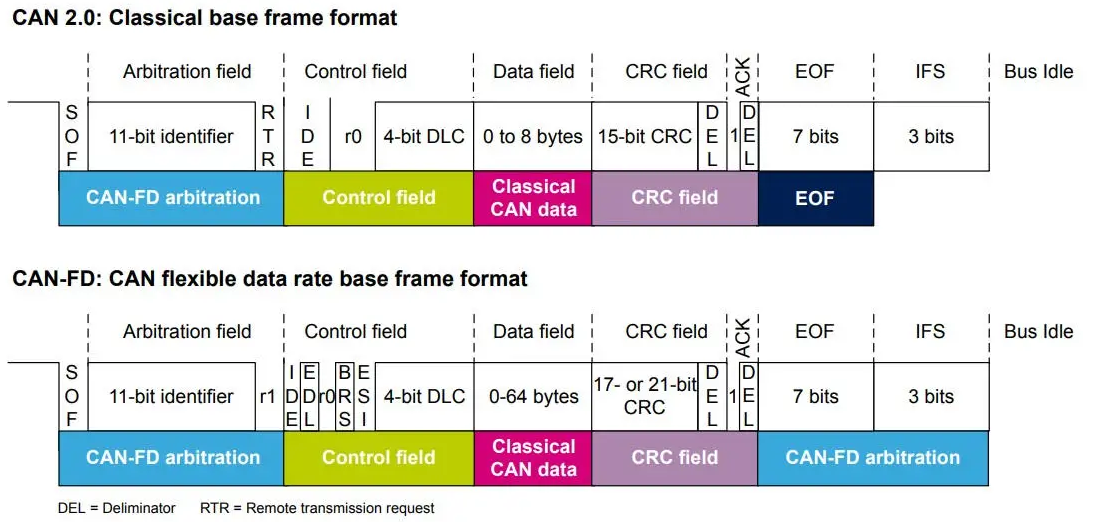
(이는 차동 신호로 통신하기 때문에 GND 연결이 필요 없다. )



can 통신의 종단저항을 설치하는 이유

전송 선로의 임피던스를 조정하여, 신호 반사를 최소화하고 전송 신호의 왜곡을 방지하는 역할을 한다.

CAN통신의 경우 종단 저항은 120Ω을 사용하고 있다. 최소 85Ω에서 최대 130Ω까지의 오차를 허용하고 있다



CAN 2.0의 경우 데이터 데이터 프레임 또는 원격 프레임과 같은 프레임 유형을 정확하게 하기 위해 RTR비트를 전송한다. CAN-FD의 경우 데이터 프레임만 지원하고 RRS을 전송한다.

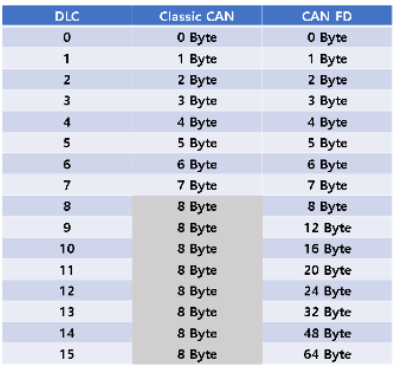
CAN-FD는 CAN 2.0 과 다르게 제어필드에 3가지 새로운 비트가 추가되었다.

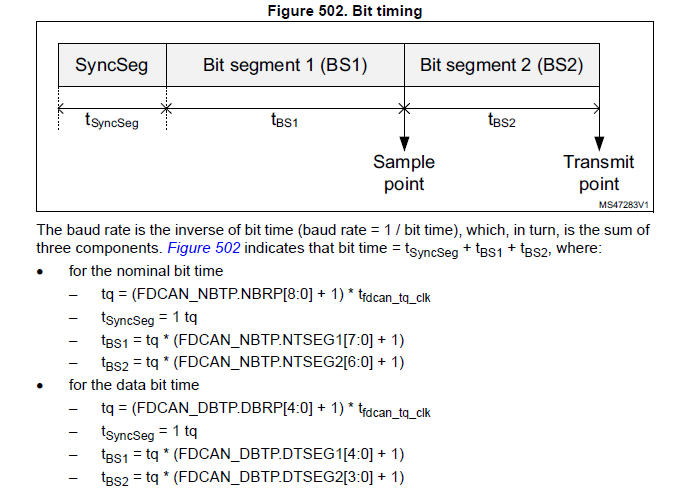
EDL(Extend Data Length)비트: 프레임이 CAN-FD임을 나타내기 위한 비트라고 판단된다.

BRS(Bit Rate Switching)비트: 두개의 비트전송률이 활성화 되었는지 여부를 나타낸다.

ESI(오류 상태 표시기): 노드가 오류 활성 또는 오류 수동 모드인지 여부를 나타낸다.

|  |  |  |
| --- | --- | --- |
| 특징 | CAN 2.0 | CAN-FD |
| 호환성 | CAN-FD 불가능 | CAN 2.0 A/B 지원 |
| 최대 비트 율(Mbit/s) | Frame bitrate : up to 1 | 중재 bitrate : up to 1s  Data bitrate : up to 8 |
| DLC field(4bits) code | Coded in 0 to 8 | Coded in 0 to 64 |
| ID의 길이 | Standard의 경우 11bit  Extended의 경우 29bit | |

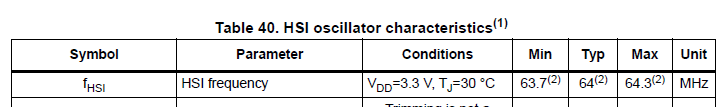


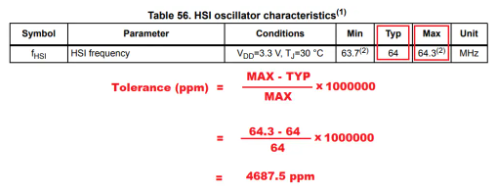


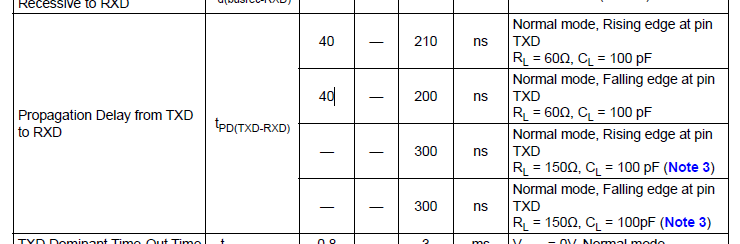
[CAN Bit Time Calculation (can-wiki.info)](http://www.bittiming.can-wiki.info/)

[Bit Timing Calculator for CAN FD (kvaser.com)](https://www.kvaser.com/support/calculators/can-fd-bit-timing-calculator/)

STM32H503rb의 클럭허용오차 계산 = 4687.5ppm



ATA6563-GAQW1의 NODE delay = 200으로 계산

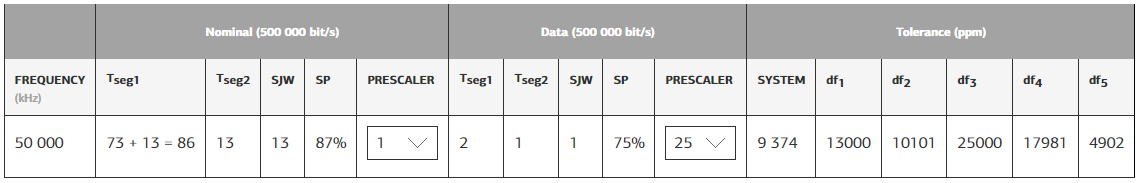


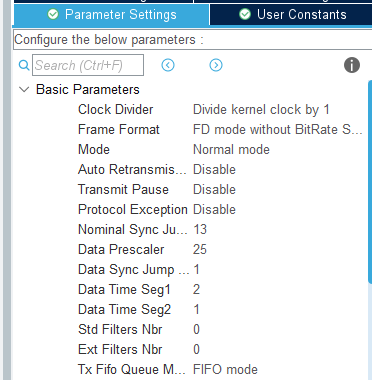
500kbps의 경우

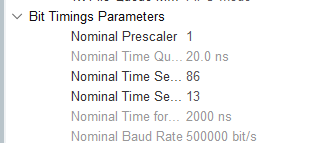
CAN통신



FD-CAN통신

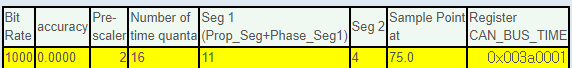




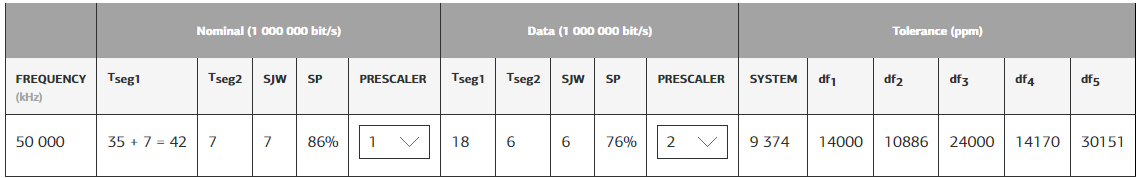


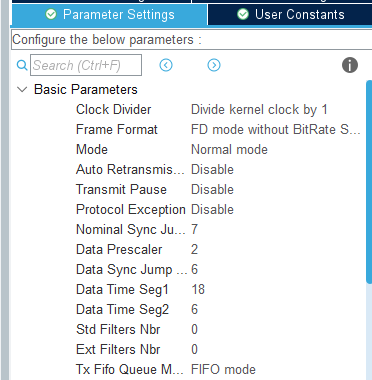
1000kbps의 경우

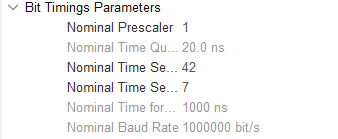
CAN통신



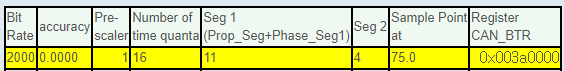
FD-CAN통신





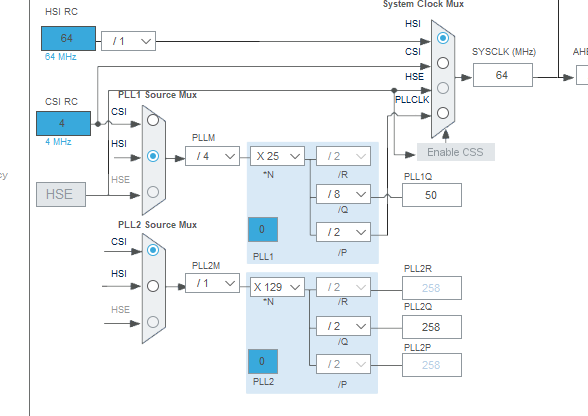


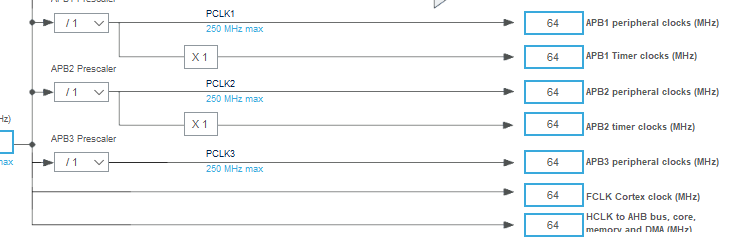
2000kbps의 경우

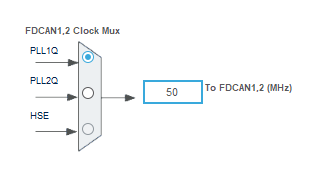


**프로그래밍**

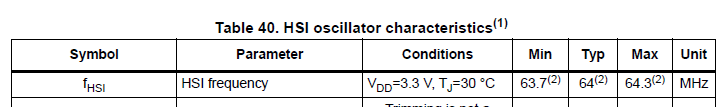
FDCAN 관련 clock 설정

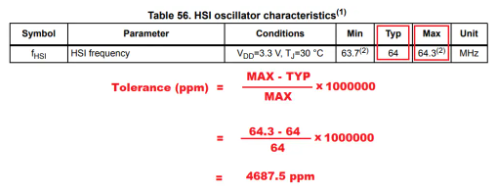




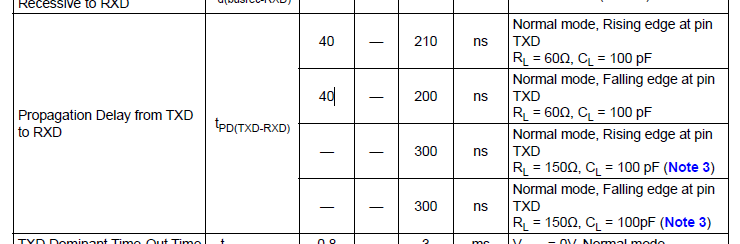


STM32H503rb의 클럭허용오차 계산 = 4687.5ppm

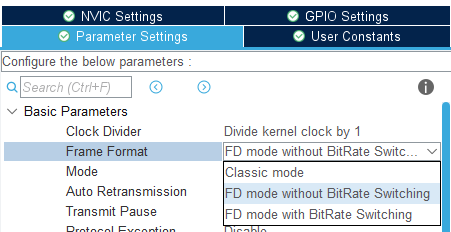




ATA6563-GAQW1의 NODE delay = 200



CAN 2.0/FD-CAN 설정 방법



Frame Format 로 Classic mode 일경우 Classic can방식으로 통신 가능하다

FDmode BitRate는 없는모드와 있는모드로 나눠진다. 먼저 FD mode without BitRate Switching으로 설정하였다.

FDCAN\_RxHeaderTypeDef RxHeader;

FDCAN\_TxHeaderTypeDef TxHeader;

FDCAN\_FilterTypeDef sFilterConfig;

**static** **void** **MX\_FDCAN1\_Init**(**void**)

{

/\* USER CODE BEGIN FDCAN1\_Init 0 \*/

/\* USER CODE END FDCAN1\_Init 0 \*/

/\* USER CODE BEGIN FDCAN1\_Init 1 \*/

/\* USER CODE END FDCAN1\_Init 1 \*/

hfdcan1.Instance = FDCAN1;

hfdcan1.Init.ClockDivider = FDCAN\_CLOCK\_DIV1;

hfdcan1.Init.FrameFormat = FDCAN\_FRAME\_FD\_NO\_BRS;

hfdcan1.Init.Mode = FDCAN\_MODE\_NORMAL;

hfdcan1.Init.AutoRetransmission = *DISABLE*;

hfdcan1.Init.TransmitPause = *DISABLE*;

hfdcan1.Init.ProtocolException = *DISABLE*;

hfdcan1.Init.NominalPrescaler = 1;

hfdcan1.Init.NominalSyncJumpWidth = 1;

hfdcan1.Init.NominalTimeSeg1 = 16;

hfdcan1.Init.NominalTimeSeg2 = 3;

hfdcan1.Init.DataPrescaler = 1;

hfdcan1.Init.DataSyncJumpWidth = 1;

hfdcan1.Init.DataTimeSeg1 = 11;

hfdcan1.Init.DataTimeSeg2 = 4;

hfdcan1.Init.StdFiltersNbr = 0;

hfdcan1.Init.ExtFiltersNbr = 0;

hfdcan1.Init.TxFifoQueueMode = FDCAN\_TX\_FIFO\_OPERATION;

**if** (HAL\_FDCAN\_Init(&hfdcan1) != *HAL\_OK*)

{

Error\_Handler();

}

/\* USER CODE BEGIN FDCAN1\_Init 2 \*/

sFilterConfig.IdType = FDCAN\_EXTENDED\_ID;

sFilterConfig.FilterIndex = 1;

//sFilterConfig.RxBufferIndex = 1;

sFilterConfig.FilterType = FDCAN\_FILTER\_DUAL;

sFilterConfig.FilterConfig = FDCAN\_FILTER\_TO\_RXFIFO0;

sFilterConfig.FilterID1 = 0x22;

sFilterConfig.FilterID2 = 0x7ff;

**if**(HAL\_FDCAN\_ConfigFilter(&hfdcan1, &sFilterConfig) != *HAL\_OK*)

{

Error\_Handler();

}

TxHeader.Identifier = 0x11;

TxHeader.IdType = FDCAN\_EXTENDED\_ID;

TxHeader.TxFrameType = FDCAN\_DATA\_FRAME;

TxHeader.DataLength = FDCAN\_DLC\_BYTES\_16;

TxHeader.ErrorStateIndicator = FDCAN\_ESI\_ACTIVE;

TxHeader.BitRateSwitch = FDCAN\_BRS\_OFF;

TxHeader.FDFormat = FDCAN\_FD\_CAN;

TxHeader.TxEventFifoControl = FDCAN\_NO\_TX\_EVENTS;

TxHeader.MessageMarker = 0x0;

HAL\_NVIC\_SetPriority(*FDCAN1\_IT0\_IRQn*, 0, 0);

HAL\_NVIC\_EnableIRQ(*FDCAN1\_IT0\_IRQn*);

/\* USER CODE END FDCAN1\_Init 2 \*/

}

FDCAN\_Init 부분이다. 이곳에서 hfdcan1.Init은 먼저 ioc에서 설정한 부분이 나와 수정하지 않았다. 그 후 sFilterConfig를 위에 선언해준 후 IdType은 EXTENDED과 STANDARD로 변경이 가능하다. 이값은 STANDARD일경우 11BIT, EXTENDED는 29BIT로 송수신이 가능하게 된다. 그 후 FilterID1,2는 송수신 쪽이 같아야지 가능한것으로 판단된다. 그 후 TxHeader로 Identifier에서 송신할 때 본인의 ID을 적을 수 있다. IdType은 위에나온 sFilterConfig와 같은 값으로 통일하였다.

HAL\_NVIC\_SetPriority(*FDCAN1\_IT0\_IRQn*, 0, 0);

HAL\_NVIC\_EnableIRQ(*FDCAN1\_IT0\_IRQn*);

이것의 경우 인터럽트를 허용해주기 위해 작성하였다. (사실 작성되어있을수도 있지만, Callback함수가 작동하지않아 추가하였다.)

**if**(HAL\_FDCAN\_Start(&hfdcan1)!=*HAL\_OK*)

{

Error\_Handler();

}

**if**(HAL\_FDCAN\_ActivateNotification(&hfdcan1, FDCAN\_IT\_RX\_FIFO0\_NEW\_MESSAGE, 0)!=*HAL\_OK*)

{

Error\_Handler();

}

여기서는 FDCAN을 시작해준 후 ActivateNotification 으로 인터럽트를 활성화 하는것으로 판단되었다.

STM32에서 CAN은Uart와 I2C와 다르게 Receive 나 Transmit 등 간단한 함수로 구성되어 있지는 않았으며 여러가지의 함수들이 많은 기능으로 구성되어 있는 것으로 판단하였다. 그리하여 먼저 while문 안에서 보내는 쪽에서는 계속 보낼 수 있게 설정하고 받는 쪽에서는 계속 받을 수 있도록 설정하였다.

if (HAL\_FDCAN\_GetRxFifoFillLevel(&hfdcan1, FDCAN\_RX\_FIFO0) > 0)

{

if (HAL\_FDCAN\_GetRxMessage(&hfdcan1, FDCAN\_RX\_FIFO0, &RxHeader, RxData) != HAL\_OK)

{

Error\_Handler();

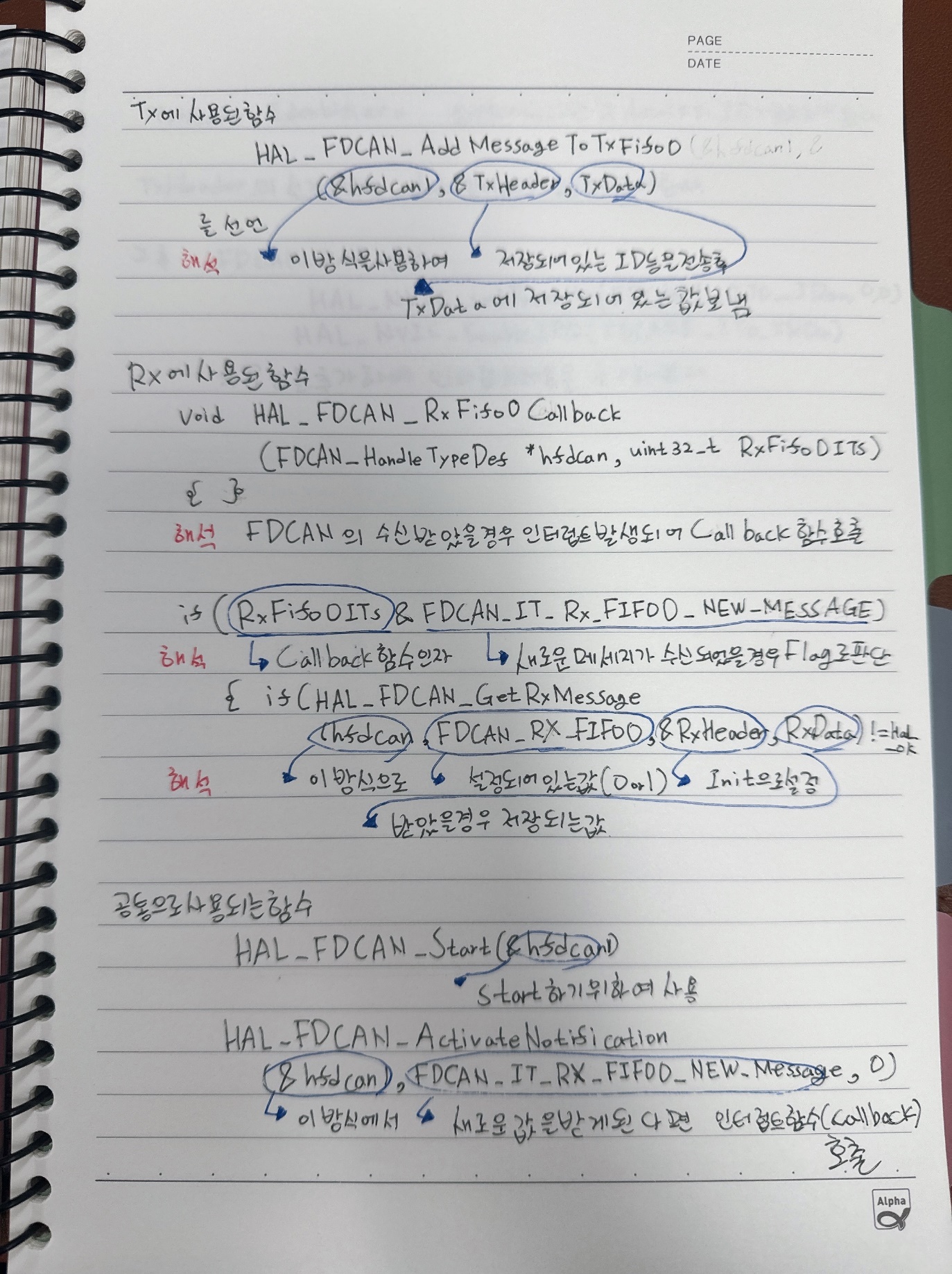
}

}

먼저 while문에 HAL\_FDCAN\_GetRxFifoFillLevel 함수를 사용하여 데이터를 받았을 경우 안에 있는 HAL\_FDCAN\_GetRxMessage 함수를 사용하여 RxData에 받은 데이터를 저장할 수 있게 한다

보내는 쪽에서는 HAL\_FDCAN\_AddMessageToTxFifoQ(&hfdcan1,&TxHeader,TxData)를 사용하여 TxData에 저장되어있는 값을 메시지에 추가하여 보내는 것으로 판단되었다,

수신 쪽에 있는 TxHeader.Identifier = 0x33으로 설정하고 그 후 TxHeader을 설정 한 후 전송을 하였을 경우 수신 쪽에 RxHeader의 값이 송신 쪽의 TxHeader로 변경되게 된다.



**구현 방법**

LED 를 사용하여 DATA를 송신했을 경우 On/Off 표현

보드 내장 LED로 수신 받았을 경우 On/Off 표현

스위치를 눌렀을 경우 인터럽트 호출하여 송신

송신파트에서 스위치를 눌렀을 경우 송신ID를 변경하여 송신 및 수신 확인

ID 타입 (Standard,Extended)로 변경해 송/수신 확인

스위치를 눌러 보드레이트(통신 속도)를 변경하여 송/수신 확인

사용 PORT

PA5 보드 내장 LED (OUTPUT)

PB10 FD-CAN1\_TX

PB12 FD-CAN1\_RX

PC2 외부 인터럽트 스위치(EXTI2)

PC3 외부 인터럽트 스위치(EXTI3)

PC8 외부 LED (OUTPUT)