

2. I2C

2. I2C

- I2C(Inter-Integrated Circuit; IIC)

- I2C는 풀업 된, SDA 및 SCL의 두 선으로 통신하며, 각 통신선의 역할은 다음과 같다.

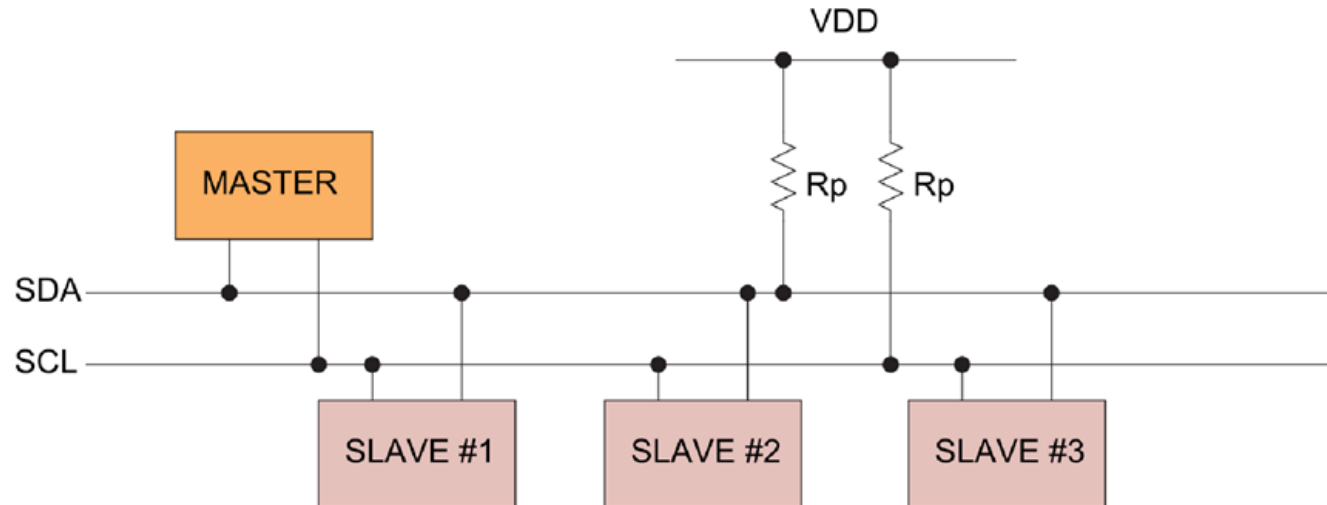
➤ **SCL** : 통신 클럭이다. 모든 통신은 이 클럭에 맞춰서 이루어지며, 해당 클럭을 생성하는 측을 **Master**, 해당 클럭을 받는 측을 **Slave**라 한다.

➤ **SDA** : 데이터 라인, Master에서 Slave로, Slave에서 Master로 데이터 통신이 되는 선이다.

Q. Synchronous / Asynchronous?

Full-Duplex / Half-Duplex?

A. Synchronous / Half-Duplex



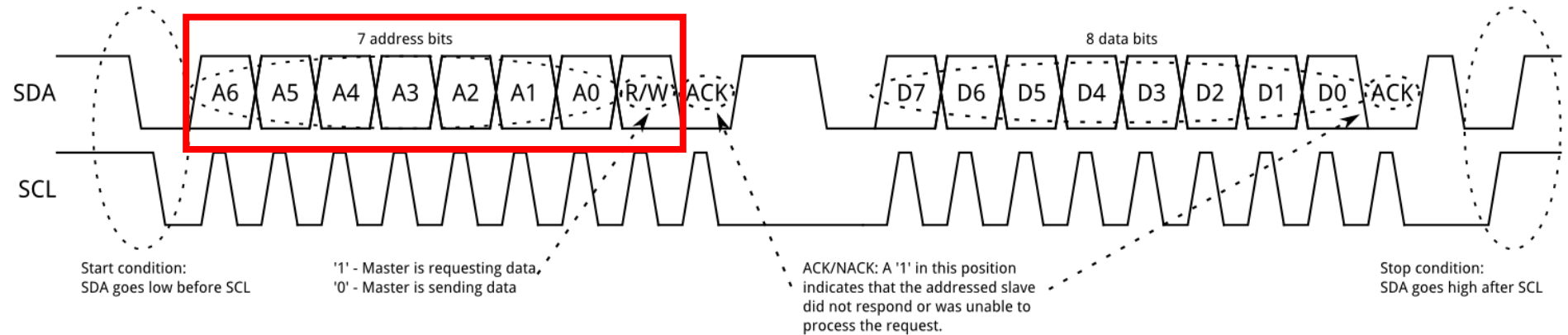
2. I2C

• I2C Frame

- 모든 Slave는 **고유한 7-bit Address**를 가지고 있다.
- Master는 통신의 첫 바이트 부분에 이러한 주소 값을 통하여 특정 Slave를 부른다.
- 첫 바이트는 이러한 Slave 주소와, 데이터 읽기/쓰기 비트가 Master에서 Slave로 보내진다.

Q. 이론상 I2C에는 최대 몇 개의 device가 연결될 수 있는가?

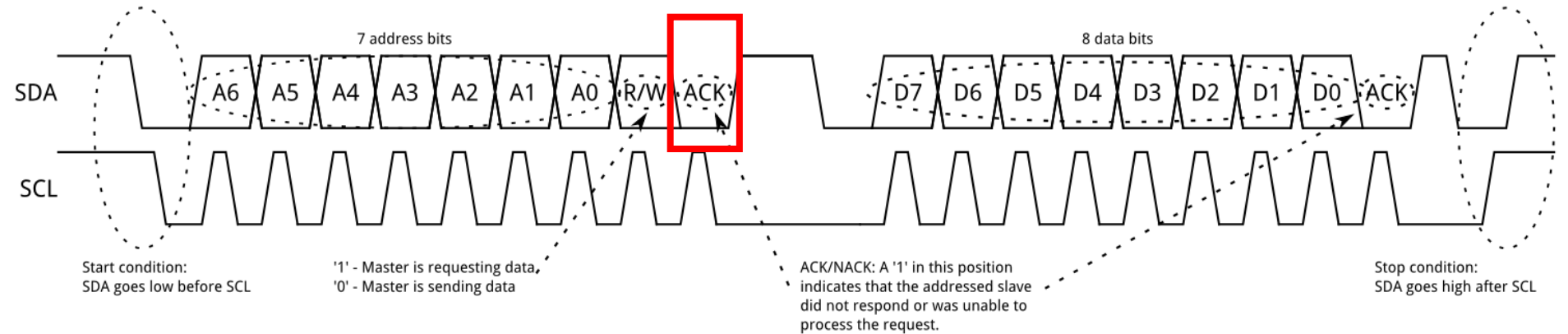
A. 128



2. I2C

• I2C Frame

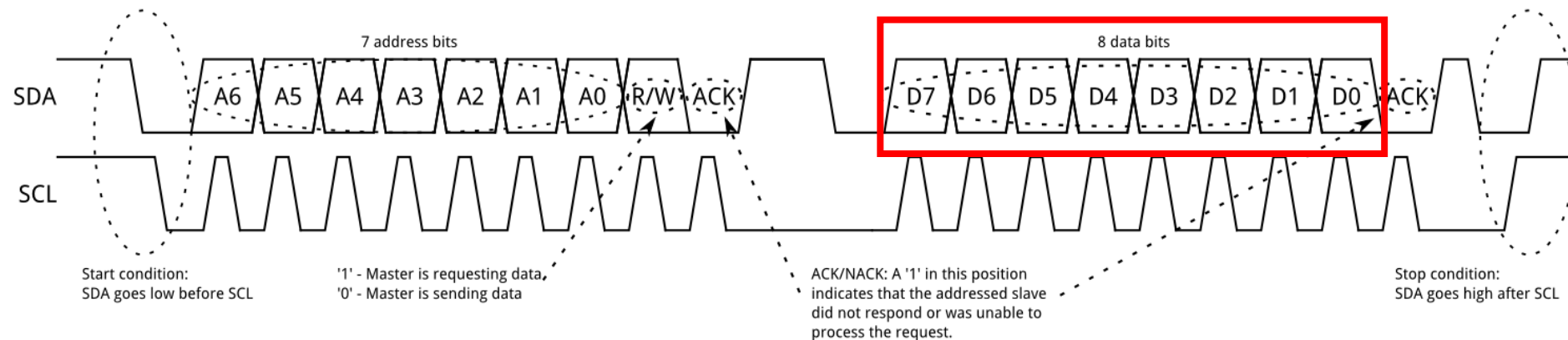
- 만약 해당 Slave가 데이터를 잘 받았다면, 풀업 되어있는 SDA 라인을 Slave가 아래로 잡아당긴다.
- 이를 **ACK**(Acknowledge)라고 한다. 만약 Slave가 응답하지 않아 SDA가 High인 상태를 유지하면 **NACK**(Not acknowledge)이라 한다.



2. I2C

• I2C Frame

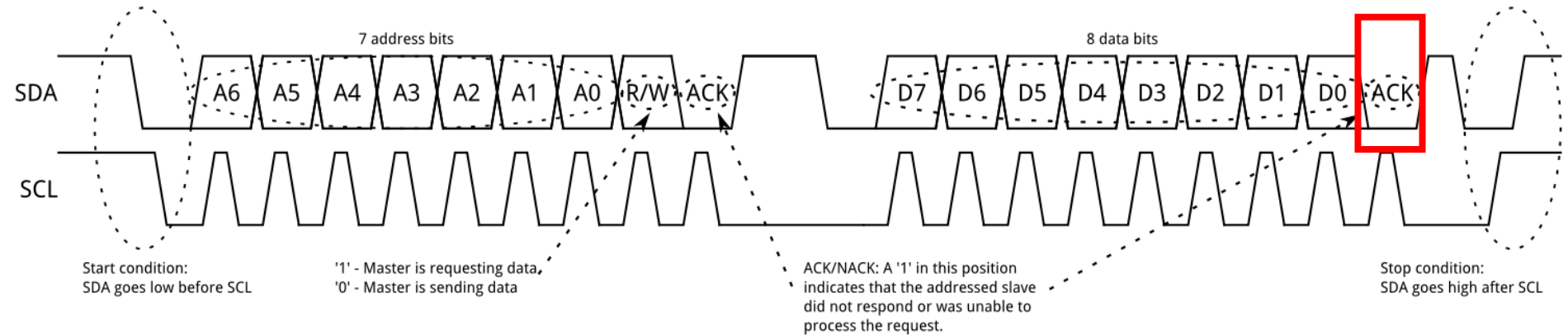
- 데이터 읽기 모드라면 Slave가 SDA 라인을 통해 Master에게 데이터를 보낸다.
- 데이터 쓰기 모드라면 Master가 SDA 라인을 통해 Slave에게 데이터를 보낸다.



2. I2C

• I2C Frame

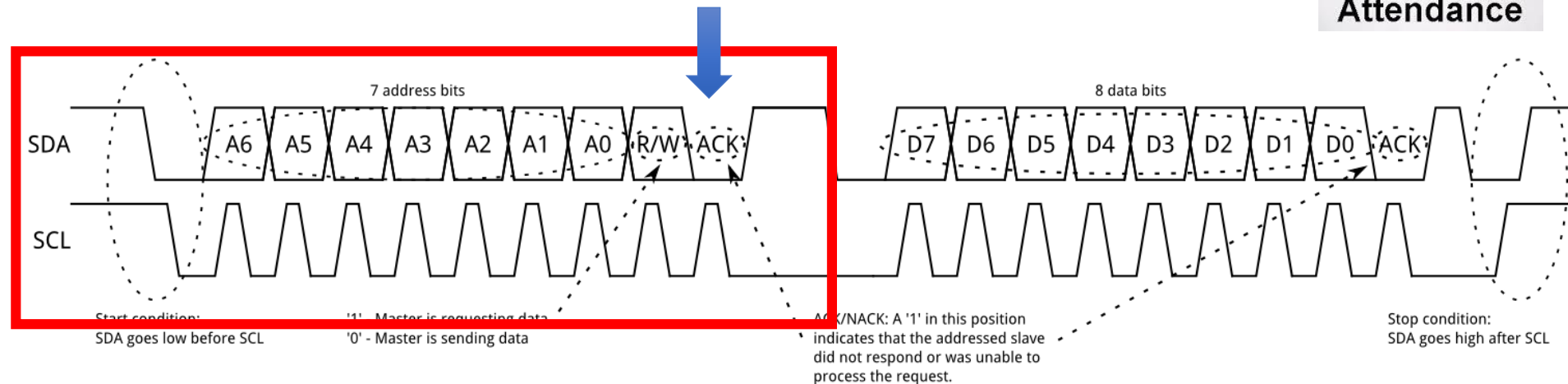
- 데이터 읽기 모드라면 Master가 SDA 라인에 ACK를 발생시킨다.
- 데이터 쓰기 모드라면 Slave가 SDA 라인에 ACK를 발생시킨다.
- 필요에 따라 연속적인 데이터 읽기/쓰기가 가능하다.



2. I2C

• I2C Scanning

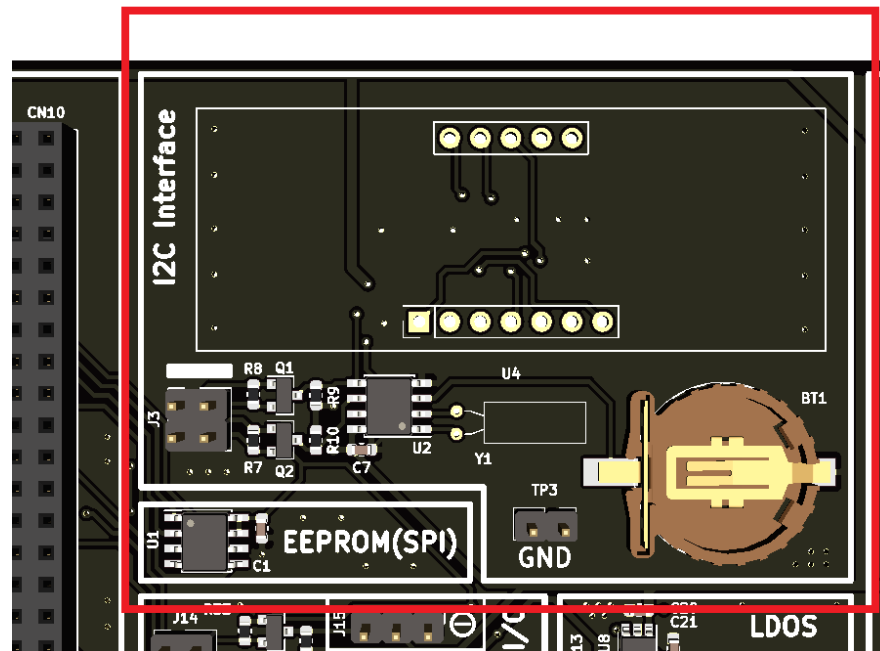
- I2C의 ACK를 이용하면, 현재 I2C Bus에 어떤 Slave들이 연결 되어있는지 알아낼 수 있다.
- 만약 특정 주소에 디바이스가 연결 되어 있으면, **첫번째 통신 Byte에서 ACK**를 받을 수 있을 것이고, **없다면 NACK**이 나타날 것이다.
- I2C 주소는 0~127까지 이므로, 모든 주소에 대해 첫 Byte만 전송하여 응답된 ACK를 관찰해보자.



2. I2C

- I2C Scanning

- 현재 테스트 보드에는 두 개의 I2C Device가 연결되어 있다.
- 해당 블록은 **PB10(SCL)**과 **PB3(SDA)**으로 연결되어 있다.

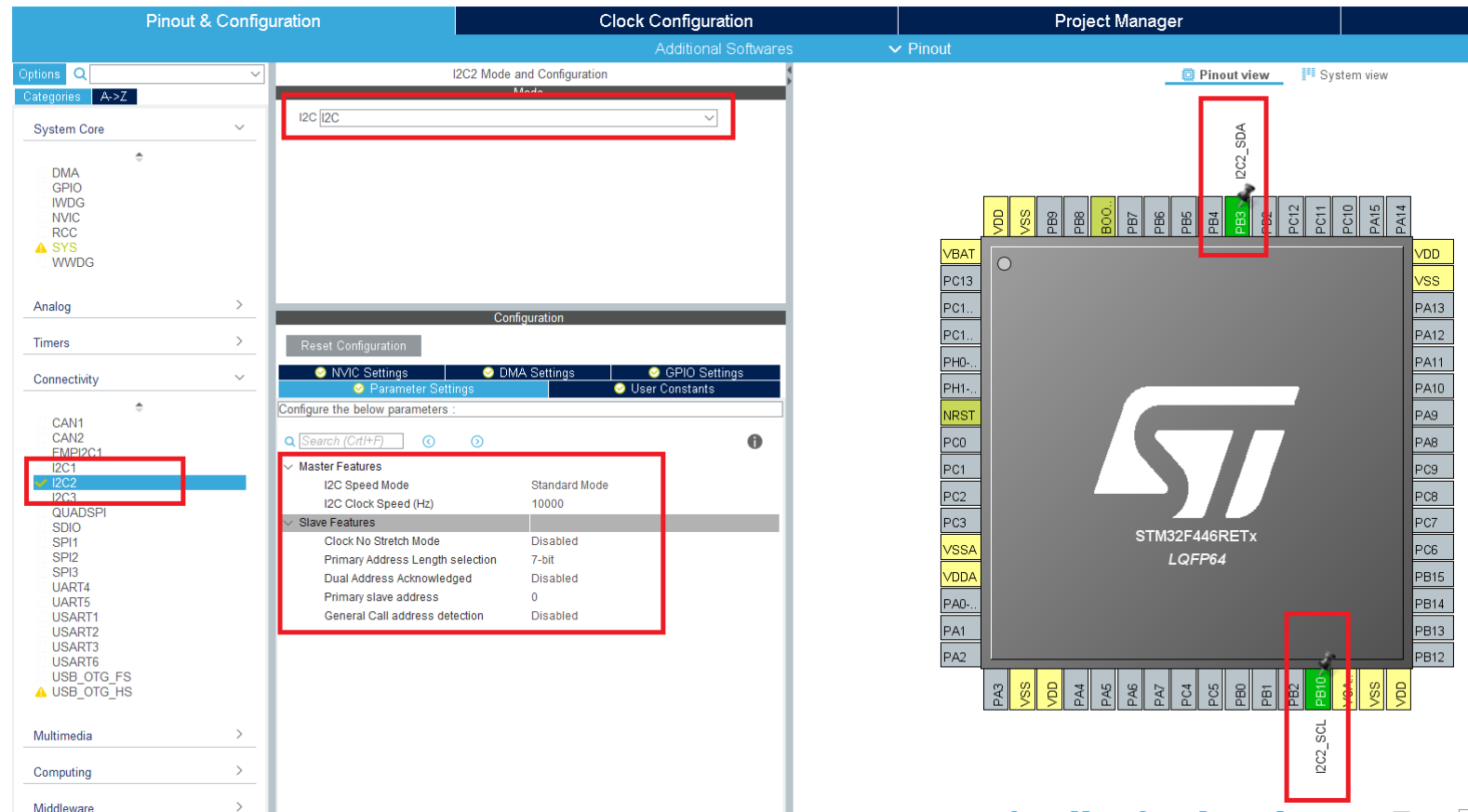


×	PC9	R1	R2	PC8	×
×	PB8	R3	R4	PC6	×
×	PB9	R5	R6	PC5	×
		R7	R8		
		R9	R10	PD8	×
SPI_SCK	PA5	R11	R12	PA12	×
SPI_MISO	PA6	R13	R14	PA11	×
SPI_MOSI	PA7	R15	R16	PB12	SPI_CS
×	PB6	R17	R18	PB11	×
×	PC7	R19	R20		▷
ENC_SW	PA9	R21	R22	PB2	×
PWM_A	PA8	R23	R24	PB1	×
I2C_SCL	PB10	R25	R26	PB15	PWM_B
ENC_CH1	PB4	R27	R28	PB14	PWM_C
ENC_CH2	PB5	R29	R30	PB13	×
I2C_SDA	PB3	R31	R32		×
	PA10	R33	R34	PC4	ADC_5
×	USART_TX	PA2	R35	PF5	×
×	USART_RX	PA3	R37	R38	PF4

2. I2C

- I2C Scanning

- 다음과 같이 PB10과 PB3을 I2C2로 설정하자.
- 현재 테스트 보드의 라우팅이 최적화 되어있지 않은 관계로 통신 속도는 10kHz로 한다.
- 코드를 생성한다.



2. I2C

- I2C Scanning

- Main.c 상단에 크기 128 짜리 배열을 만든다.

```
/* Private user code --  
/* USER CODE BEGIN 0 */  
uint8_t i2cList[128];  
/* USER CODE END 0 */
```

2. I2C

- I2C Scanning

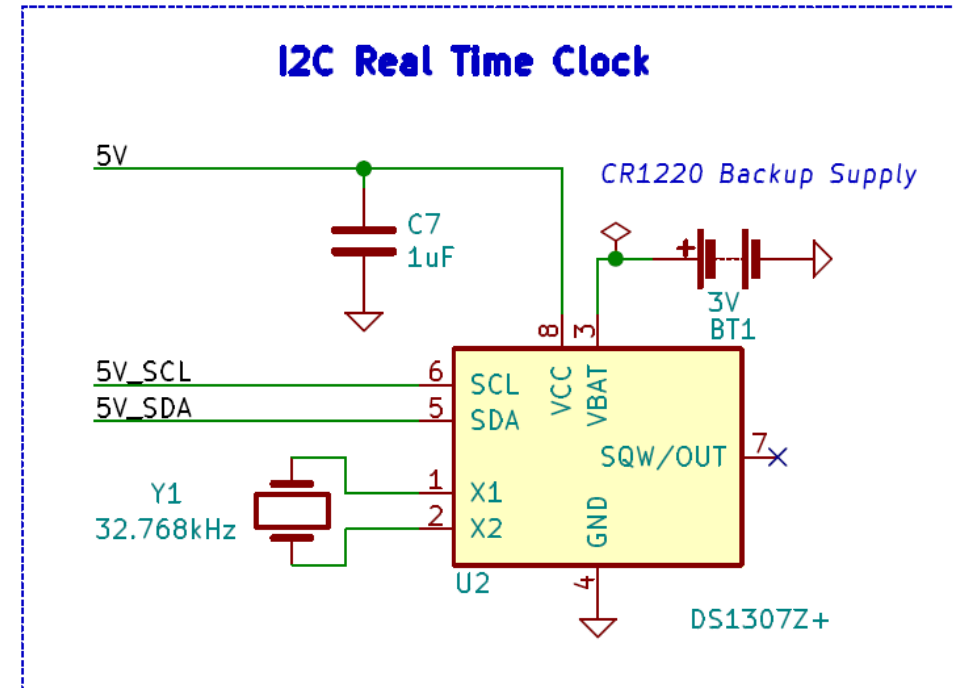
- 다음과 같이 코드를 작성한 후, 업로드를 한다.
- Live Watch를 통해 i2cList를 관찰한다.

```
HAL_I2C2_Init();  
/* USER CODE BEGIN 2 */  
uint8_t address;  
uint8_t txDummy;  
/* USER CODE END 2 */  
  
/* Infinite loop */  
/* USER CODE BEGIN WHILE */  
while (1)  
{  
    for(address = 0 ; address < 128; address ++){  
        if(HAL_I2C_Master_Transmit(&hi2c2, address << 1, &txDummy, 0 ,1000) == HAL_OK){  
            i2cList[address] = 1;  
        }else{  
            i2cList[address] = 0;  
        }  
        HAL_Delay(1);  
    }  
}   
/* USER CODE END WHILE */
```

2. I2C

- RTC(Real Time Clock)

- 현재 I2C Bus에 연결 되어있는 두 디바이스 중 하나는 RTC 모듈이다.
- RTC는 내부에 자체 클럭과 달력을 내장하고 있어서 시간을 읽을 수 있다.
- 현재 사용하고 있는 RTC는 DS1307로, 두 개의 전원 소스로 운용되며, 주 전원이 꺼지면 보조전원 (코인셀)으로 돌아가 전원이 꺼지더라도 시계가 계속 동작하게 할 수 있다.
- DS1307Z의 address 는 **0x68(104)**이다.



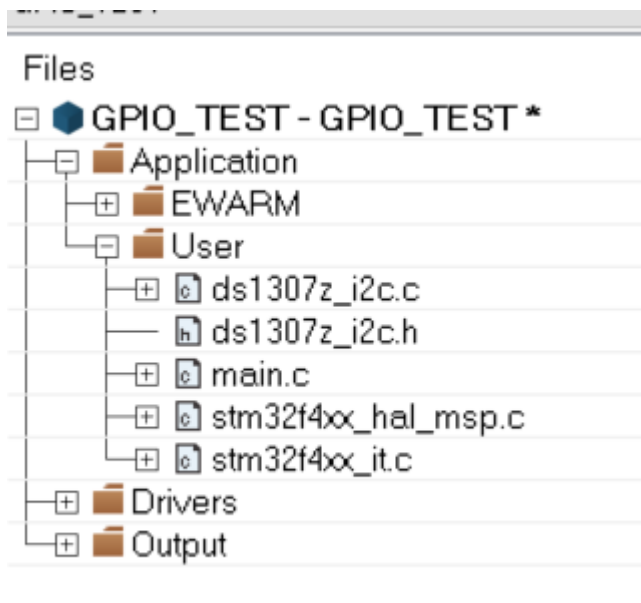
2. I2C

- RTC(Real Time Clock)

- 미리 주어진 "ds1307z_i2c.h" 및 "ds1307z_i2c.c"를 Src 폴더에 복사하고, IAR에서 파일을 추가한다.

이름

ds1307z_i2c.c
ds1307z_i2c.h
main.c
stm32f4xx_hal_msp.c
stm32f4xx_it.c
system_stm32f4xx.c



2. I2C

- RTC(Real Time Clock)

- Main.c에 헤더 파일을 인클루드 한다.

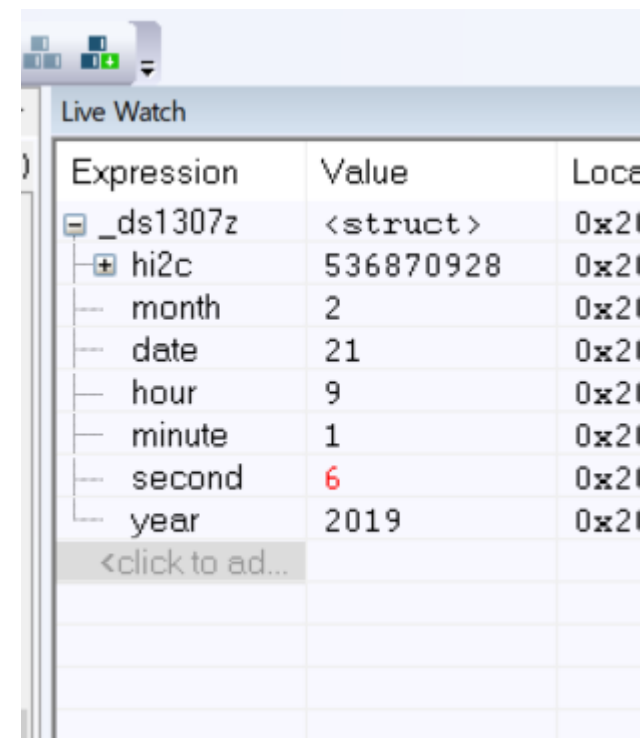
```
/* Private includes -----  
/* USER CODE BEGIN Includes */  
#include "ds1307z_i2c.h"  
/* USER CODE END Includes */
```

2. I2C

- RTC(Real Time Clock)

- 다음과 같이 코드를 작성한 후, Live Watch를 통해 시간의 변화를 관찰하자.

```
MA_I2CZ_Init();  
/* USER CODE BEGIN 2 */  
DS1307Z_Init(&hi2c2);  
  
DS1307Z_writeTime(2019, 2, 21, 9, 0, 0);  
/* USER CODE END 2 */  
  
/* Infinite loop */  
/* USER CODE BEGIN WHILE */  
while (1)  
{  
    DS1307Z_readTime();  
    HAL_Delay(500);  
    /* USER CODE END WHILE */  
}
```



Expression	Value	Location
_ds1307z	<struct>	0x21000000
hi2c	536870928	0x21000004
month	2	0x21000008
date	21	0x2100000C
hour	9	0x21000010
minute	1	0x21000014
second	6	0x21000018
year	2019	0x2100001C
<click to add new expression>		

2. I2C

- RTC(Real Time Clock)

- 다음 데이터 시트의 표를 통해 해당 코드를 이해해 보자.

Table 2. Timekeeper Registers

ADDRESS	BIT 7	BIT 6	BIT 5	BIT 4	BIT 3	BIT 2	BIT 1	BIT 0	FUNCTION	RANGE
00h	CH	10 Seconds			Seconds				Seconds	00–59
01h	0	10 Minutes			Minutes				Minutes	00–59
02h	0	12	10 Hour	10 Hour	Hours				Hours	1–12 +AM/PM 00–23
		24	PM/ AM							
03h	0	0	0	0	0	DAY			Day	01–07
04h	0	0	10 Date		Date				Date	01–31
05h	0	0	0	10 Month	Month				Month	01–12
06h	10 Year				Year				Year	00–99
07h	OUT	0	0	SQWE	0	0	RS1	RS0	Control	—
08h–3Fh									RAM 56 x 8	00h–FFh

0 = Always reads back as 0.

```
void DS1307Z_readTime() {
```

```
    uint8_t addr = 0x00;
    uint8_t readData[7];
```

```
    HAL_I2C_Master_Transmit (_ds1307z.hi2c, DS1307Z_ADDR << 1, &addr, 1, 100);
```

```
    HAL_I2C_Master_Receive(_ds1307z.hi2c, DS1307Z_ADDR << 1, readData, 7, 100);
```

```
    _ds1307z.second = ((readData[0] & 0x70) >> 4)*10 + (readData[0] & 0x0F);
    _ds1307z.minute = (readData[1] >> 4)*10 + (readData[1] & 0x0F);
    _ds1307z.hour = ((readData[2] & 0x30) >> 4)*10 + (readData[2] & 0x0F);
    _ds1307z.date = (readData[4] >> 4)*10 + (readData[4] & 0x0F);
    _ds1307z.month = (readData[5] >> 4)*10 + (readData[5] & 0x0F);
    _ds1307z.year = 2000 + (readData[6] >> 4)*10 + (readData[6] & 0x0F);
```

```
}
```


2. I2C

- RTC(Real Time Clock)

- 다음 그림을 통해 해당 코드를 이해해 보자.

```
void DS1307Z_readTime() {
```

```
    uint8_t addr = 0x00;  
    uint8_t readData[7];
```

```
    HAL_I2C_Master_Transmit (_ds1307z.hi2c, DS1307Z_ADDR << 1, &addr, 1, 100);
```

```
    HAL_I2C_Master_Receive(_ds1307z.hi2c, DS1307Z_ADDR << 1, readData, 7, 100);
```

```
    _ds1307z.second = ((readData[0] & 0x70) >> 4)*10 + (readData[0] & 0x0F);  
    _ds1307z.minute = (readData[1] >> 4)*10 + (readData[1] & 0x0F);  
    _ds1307z.hour = ((readData[2] & 0x30) >> 4)*10 + (readData[2] & 0x0F);  
    _ds1307z.date = (readData[4] >> 4)*10 + (readData[4] & 0x0F);  
    _ds1307z.month = (readData[5] >> 4)*10 + (readData[5] & 0x0F);  
    _ds1307z.year = 2000 + (readData[6] >> 4)*10 + (readData[6] & 0x0F);
```

```
}
```

Figure 4. Data Write—Slave Receiver Mode

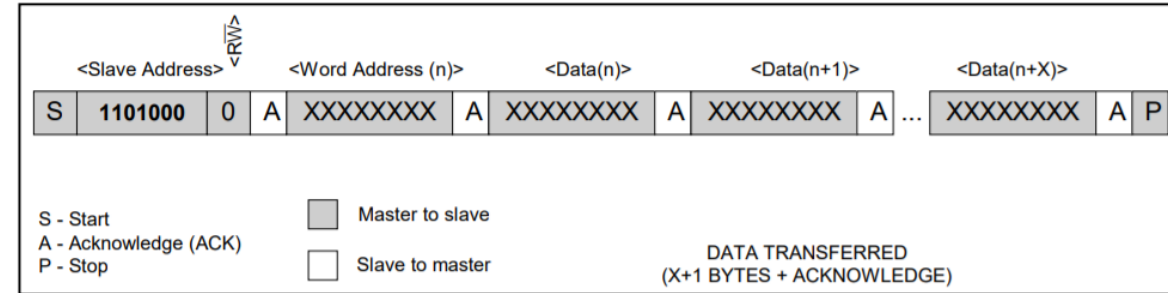


Figure 5. Data Read—Slave Transmitter Mode

