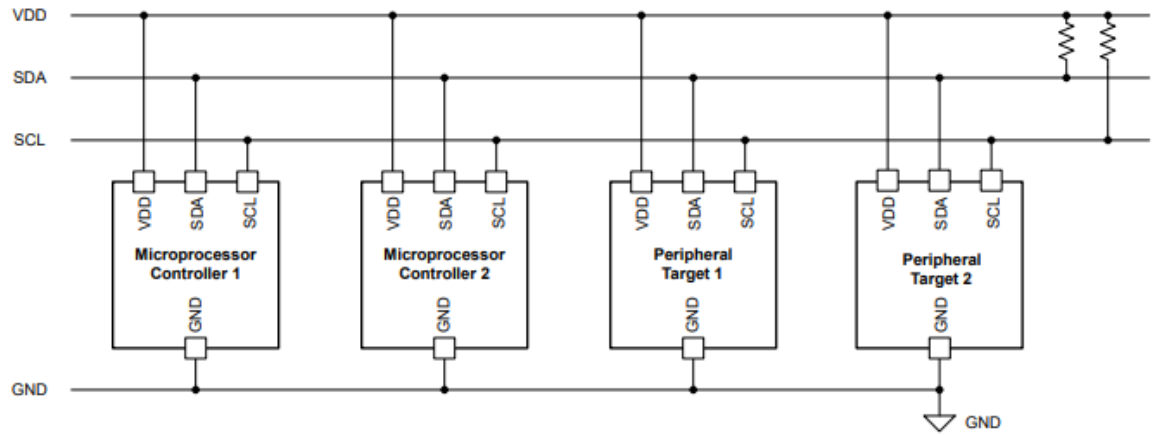
I2C

1. I2C Overview

I2C는 두 개의 선을 사용하는 직렬 통신 프로토콜이다. 데이터 선(SDA)과 클럭 선(SCL)을 사용하여 통신한다. 하나의 통신 버스 위에 여러 개의 슬레이브 장치를 연결할 수 있다. 각 슬레이브는 고유한 주소를 가지고 있으며, 통신은 바이트 단위의 패킷으로 이루어진다. 본 I2C는 Standard-mode를 지원하며, 최대 100kbps의 전송 속도를 제공한다.

2. I2C Physical Layer

2.1Two-Wire Communication



I2C 시스템은 두개의 통신 선을 사용하며, 이 선들은 버스에 연결된 모든 장치들이 공유한다. I2C는 양방향 half-duplex 통신을 지원하여, 한 번에 한 방향으로만 데이터를 주고 받을 수 있는 구조이다. SPI가 통신을 위해 4개의 선(SCK, MOSI, MISO, CS)을 필요로 하는 반면, I2C는 단 두 개의 선(SCL, SDA) 만으로 통신이 가능하다. SCL과 SDA는 모든 장치와 Open-drain 방식으로 연결되며, 이로 인해 공통 전압(Vcc)에 연결된 풀업 저항이 두 라인에 모두 필요하다.

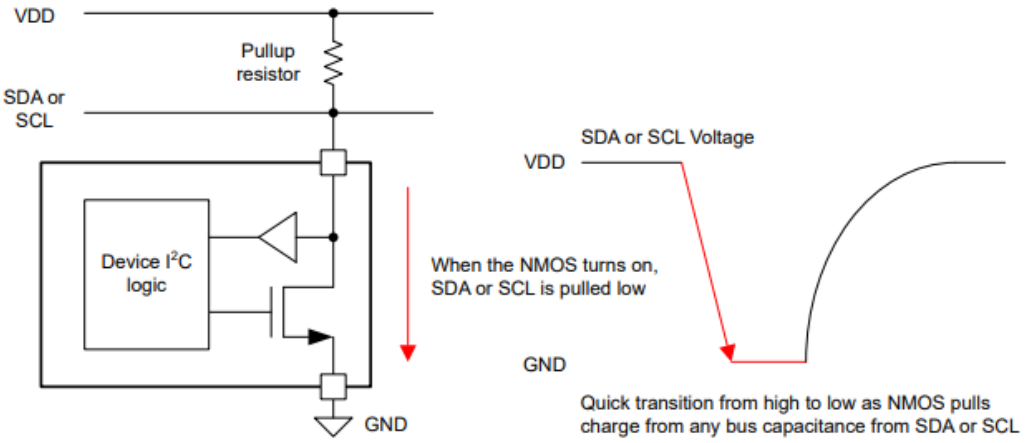
2.1 Open-Drain Connection

SCL과 SDA 라인은 모두 Open-drain 방식으로 연결되며, NMOS 트랜지스터에 의해 제어된다. Open-drain 출력은 내부 회로가 High 상태를 직접 만들 수 없고, Low 상태만 만들 수 있는 구조이다.

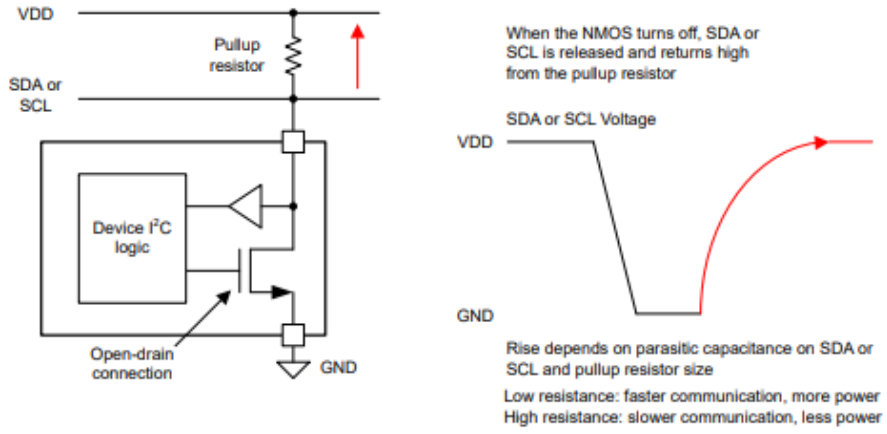
SDA, SCL 라인의 전압 레벨을 설정하기 위해, NMOS를 켜거나 끄는 방식을 사용한다.

- NMOS가 켜지면, 장치는 풀업 저항을 통해 GND로 전류를 당겨, Open-drain 라인을 LOW 상태로 만든다.

  일반적으로 I2C에서 High -> Low 로의 전환은 빠르게 이루어지는데, 이는 NMOS가 라인을 강하게 GND로 당기기 때문이다.



- 반대로 NMOS가 꺼지면 장치는 더 이상 전류를 끌어당기지 않게 되고, 풀업 저항이 라인을 High로 끌어올린다.



I2C 에서 오픈 드레인 방식을 사용하는 주요 장점 중 하나는, 버스 충돌이 발생하더라도 회로에 손상을 주지 않는다는 점이다. 여러 장치가 같은 라인에 연결되어 있어도, 어떤 장치가 라인을 Low로 당기면 전체 라인은 Low 상태가 되며 충돌이 발생하지 않는다. 이와 같은 연결 방식을 Write- AND라고 한다.

반면, 출력 방식이 푸시풀(Push-pull) 구조일 경우, 여러 출력을 한 라인에 연결하면 충돌이 발생

할 수 있다. 예를 들어, 한 장치는 High를 출력하고 다른 장치는 Low를 출력하면 단락이 발생하여 회로가 손상될 수 있다.

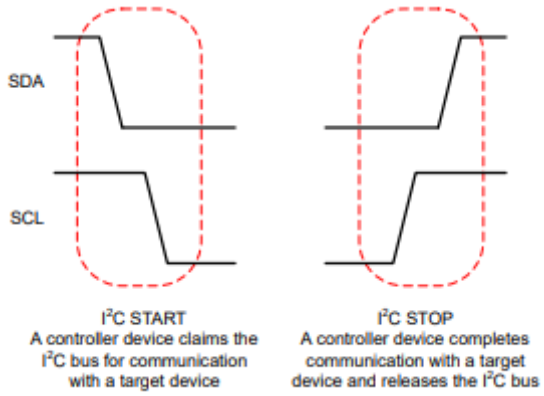
3. I2C Protocol

3.1 START and STOP

I2C 통신은 컨트롤러 장치가 I2C START 조건을 발생시키면서 시작된다.

START 조건을 만들기 위해, 컨트롤러는 먼저 SDA를 LOW로 내리고, 이어서 SCL을 Low로 내린다. 이 시퀀스는 컨트롤러가 I2C 버스를 점유하려는 신호이며, 버스에 연결된 다른 컨트롤러들은 대기해야한다.

통신이 완료되면 컨트롤러는 SCL을 먼저 High로 올리고, 그 다음에 SDA를 High로 올린다. 이는 I2C STOP 조건을 의미한다.



C Code Example

```
void start_I2C(I2C_Typedef *I2Cx){
    I2Cx -> CR = 0x05; //101
}

void stop_I2C(I2C_Typedef *I2Cx){
    I2Cx -> CR = 0x03; //011
}
```

3.2 Logical Ones and Zeros

I2C는 일련의 1과 0으로 이루어진 시퀀스를 통해 직렬 통신을 수행한다. 유효한 데이터 비트로 인식되기 위해서는, 해당 비트의 SCL 상승 에지와 하강 에지 사이 동안 SDA 값이 변하지 않아야 한다.

만약 SCL 상승 에지와 하강 에지 사이에 SDA가 변하는 경우, 이는 START 또는 STOP 조건으로 해석될 수 있다.

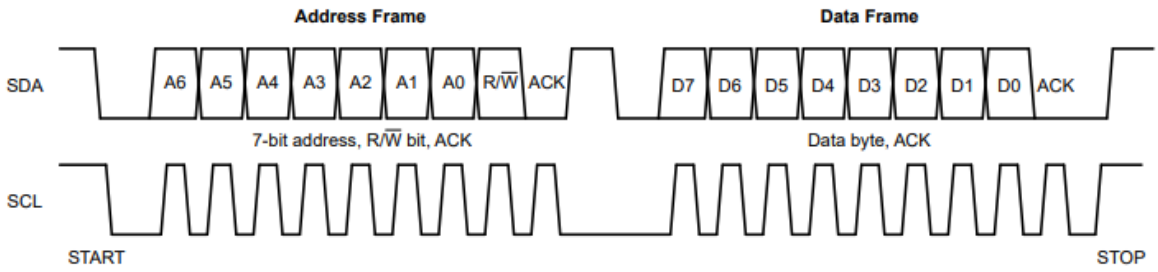
- **Logical one (1)**

  SDA 라인이 해제되어 풀업 저항에 의해 라인이 높은 전압 상태가 될 때 전송

- **Logical zero (0)**

  SDA가 라인을 GND 방향으로 끌어내려 LOW 상태로 만들 때 전송

3.3I2C Communication Frame



I2C 프로토콜은 프레임 단위로 구성되어 있다. 통신은 컨트롤러 장치가 START 조건을 발생시킨 후, 주소 프레임을 전송하면서 시작된다. 주소 프레임 뒤에는 하나 이상의 데이터 프레임이 이어지며, 각 프레임은 1바이트의 데이터로 구성되어 있다.

I2C 버스에 연결된 각 슬레이브 장치는 고유한 7bit I2C 주소를 가지고 있으며, 컨트롤러가 특정 장치와 통신을 시작할 때는 해당 장치의 주소를 포함한 주소 프레임을 먼저 전송한다.

주소 뒤를 잇는 8번째 비트는 읽기/쓰기 비트로,

- 이 비트가 1이면 컨트롤러가 타겟장치로부터 데이터를 읽으려고 한다는 의미이고,
- 0이면 타겟 장치로 데이터를 쓰려는 동작을 의미한다.

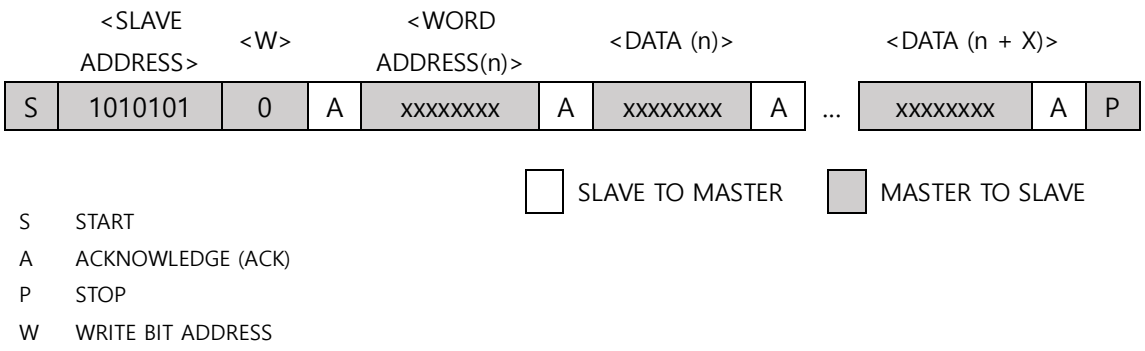
주소 전송이 완료된 후 이 주소를 기반으로 데이터 송수신이 이루어진다.

모든 통신 바이트(주소 또는 데이터) 이후에는 추가적인 9번째 비트가 사용되어 통신 성공 여부를 확인한다. 바이트 전송이 끝난 후 SCL 펄스 동안 SDA를 LOW로 당겨 데이터를 정상적으로 수신했음을 알리는데, 이를 ACK 라고 한다. 반대로 SDA가 High로 유지될 경우, 통신이 실패했음을

나타내며 NACK 라고 한다.

3.3.1 Write Mode (Slave Receiver Mode)

마스터에서 슬레이브로 데이터를 전송할 때, 가장 먼저 전송되는 바이트는 슬레이브 주소이며, 그 뒤를 이어 여러 개의 데이터 바이트가 전송된다. 데이터는 MSB 부터 먼저 전송되며, 슬레이브는 수신한 각 바이트마다 ACK 비트를 반환하여 정상 수신 여부를 마스터에게 알린다.



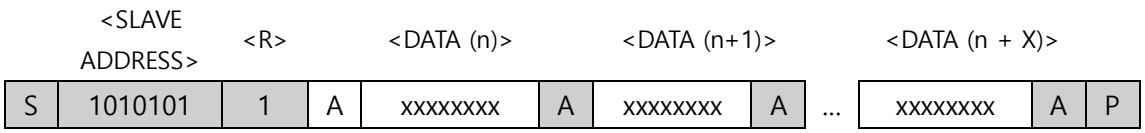
C Code Example

```
void WRITE_I2C(I2C_Typedef *I2Cx, uint32_t data){
    while(is_ready(I2C) ==0);
    I2C -> WDATA =data;
    data_I2C(I2C);
    I2C -> CR =0x00;
}
```

3.3.2 Read Mode (Slave Transmitter Mode)

슬레이브가 마스터에게 데이터를 전송할 때 첫번째 바이트는 슬레이브 주소이며, 이 바이트는 마스터가 전송하고 슬레이브가 ACK 비트로 응답한다.

이후 슬레이브가 여러 개의 데이터 바이트를 마스터에게 전송하며, 마스터는 마지막 바이트를 제외한 모든 바이트에 대해 ACK 비트를 반환한다. 마지막 바이트가 수신되면 마스터는 NACK를 반환하여 전송 종료를 알린다.



SLAVE TO MASTER MASTER TO SLAVE

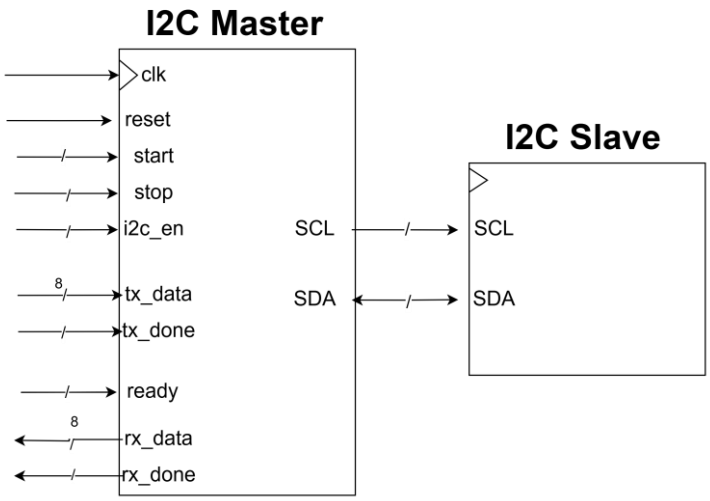
S	START
A	ACKNOWLEDGE (ACK)
P	STOP
W	WRITE BIT ADDRESS
$\bar{A}$	NOT ACKNOWLEDGE

C Code Example

```
void READ_I2C(I2C_Typedef *I2Cx, uint32_t *DATA){
     I2Cx-> CR = 0x07; //111
     while(is_ready(I2C) == 0));
     *DATA = I2C -> DATA1;
}
```

4. I2C Interface Hardware Design

4.1 Signal Descriptions



Signal	Source	Width	Description
clk	System bus clock	1	**Clock**
reset	System bus reset	1	**Reset**.
start	AXI4- Lite I2C Intf CR Register	1	**Control Sig** start, stop, i2c_en 조합에 따라 모드가 결정된다. {start, stop, i2c_en} - 101: 통신 시작 - 011: 통신 종료 - 001: WRITE Mode - 111: READ Mode
stop	AXI4- Lite I2C Intf CR Register	1	
i2c_en	AXI4- Lite I2C Intf CR Register	1	

tx_data	AXI4- Lite I2C Intf WDATA Register	8bit	**Write Data**. I2C 슬레이브로 전송할 8비트 데이터
tx_done	Master	1	**Transmission Done** 데이터 송신 완료를 알리는 flag
ready	Master	1	**Ready** - 통신 중이 아닐 때: I2C가 통신 가능할 때 high - 통신 중: HOLD 상태 일 때 high
rx_data	Master	8	**Read Data** I2C 슬레이브로부터 수신한 8비트 데이터
rx_done	Master	1	**Receive Done** 데이터 수신 완료를 알리는 flag
SCL	Master	1	**SCL (Clock Line)** I2C 통신을 위한 직렬 클럭 신호
SDA	Master/Slave	1	**SDA (Data Line)** I2C 통신을 위한 데이터 라인. 오픈 드레인 방식이며 양방향으로 사용됨

4.2 Register Map

offset	Register	31…8	7	6	5	4	3	2	1	0
0000	CR	Reserved						start	stop	I2c_en
0004	SR	Reserved						rx_done	tx_done	Ready
0008	WDATA	Reserved	tx_data[7:0]							
000C	DATA1	Reserved	rx_data1[7:0]							
0010	DATA2	Reserved	rx_data2[7:0]							
0014	DATA3	Reserved	rx_data3[7:0]							
0018	DATA4	Reserved	rx_data4[7:0]							

4.2.1 Control Register (CR)

Address offset: 0x00

Reset value: 0x0000 0000

31…3	2	1	0
Res	start	stop	i2c_en
	rw	rw	rw

Bit 2 **start**: start bit

I2C 통신 시작 시 설정

Bit 1 **stop**: stop bit

I2C 통신 종료 시 설정

Bit 0 **i2c_en**: enable bit

I2C 인터페이스를 활성화하는 비트

Bus가 점유되지 않았거나, 1바이트 송수신을 완료한 이후 start, stop, i2c_en 신호의 조합에 따라 다음 통신 동작이 결정된다.

4.2.2 Status Register (SR)

Address offset: 0x04

Reset value: 0x0000 0000

31...3	2	1	0
Res	rx_done	tx_done	ready
	r	r	r

Bits 2 **rx_done**: Receive Done

슬레이브로부터의 데이터 수신이 완료되었음을 나타내는 플래그

수신 완료 시 1 SCL 클럭 주기 동안 High(1)로 설정되며,

읽기 동작 수행 시 소프트웨어에서 이를 감지하여 적절한 처리를 수행해야 한다.

Bits 1 **tx_done**: Transmit Done

마스터에서 슬레이브로의 데이터 전송이 완료되었음을 나타내는 플래그

전송이 끝나면 1 SCL 클럭 주기 동안 High(1)로 설정된다.

Bits 0 **ready**:

I2C 모듈이 통신 가능 상태임을 나타냄

4.2.3 WRITE DATA (WDATA)

Address offset: 0x08

Reset value: 0x0000 0000

31…8	7	6	5	4	3	2	1	0
Res	WDATA[7:0]							
	rw	rw	rw	rw	rw	rw	rw	Rw

Bits 7:0 **WDATA[7:0]**: Transmit data value

마스터가 슬레이브에게 전송하고자 하는 Write Data를 저장하는 레지스터

4.2.4 DATA1~4

Address offset: 0x0C, 0x10, 0x14, 0x18

Reset value: 0x0000 0000

31…8	7	6	5	4	3	2	1	0
Res	RDATA[7:0]							
	r	r	r	r	r	r	r	r

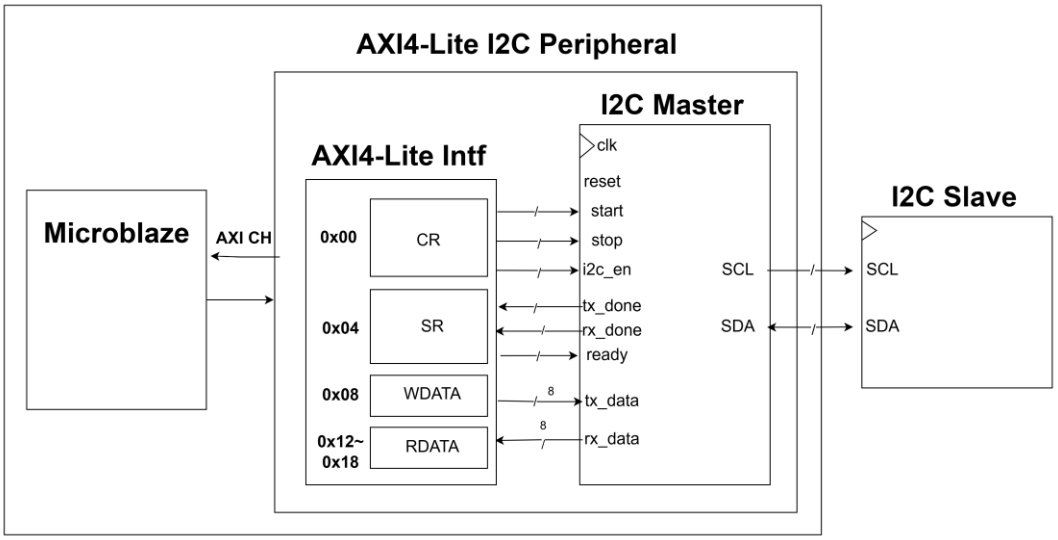
Bits 7:0 **RDATA[7:0]**: Receive data value

슬레이브로부터 수신된 Read Data를 저장하는 레지스터

최대 4바이트까지 연속 수신된 데이터를 저장할 수 있도록 4개의 버퍼 레지스터로 구성되며,

각 레지스터는 읽기 전용이다.

4.3 Top-Level Architecture



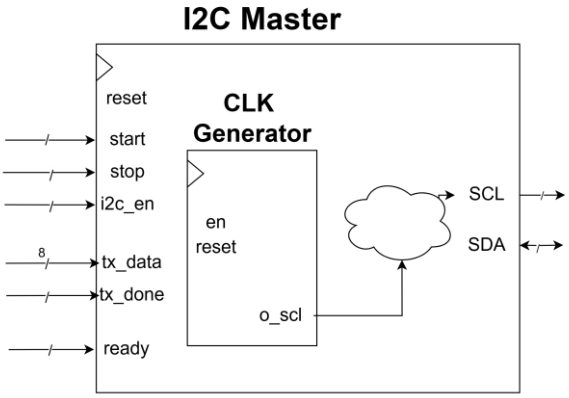
위 그림은 본 설계에서 구현한 AXI4-Lite I2C Peripheral의 시스템 수준 구성도이다. Microblaze와 같은 임베디드 프로세서는 AXI4-Lite 채널을 통해 I2C 모듈과 통신하며, AXI 인터페이스 안에는 제어 레지스터(CR), 상태 레지스터(SR), 데이터 전송(WDATA), 데이터 수신(RDATA) 레지스터가 포함된다.

I2C Master 블록은 내부에 제어 구조를 통해 사용자 명령(start, stop, i2c_en)에 따라 SDA/SCL 신호를 직접 제어한다.

또한 통신 상태를 나타내는 ready, tx_done, rx_done 등의 상태 플래그를 SR을 통해 확인할 수 있으며, 이를 통해 소프트웨어는 I2C 모듈의 동작 상태를 모니터링하고, 데이터 송수신 시점을 제어할 수 있다.

SDA와 SCL는 외부 I2C Slave 장치와 직접 연결 되며, 이를 통해 실제 디바이스와의 읽기/쓰기 통신이 가능하다.

4.4Internal Clock Generator 설계 및 최적화 효과



본 설계에서는 I2C 통신 속도(100kHz) 에 맞춰 SCL 클럭을 생성하기 위한 내부 Clock Generator 를 별도로 구현하였다. 초기 구조에서는 SCL 상승/하강 타이밍을 FSM 내에서 직접 정의하기 위해 각 1 바이트 송/수신, ACK 를 4개의 세부 상태로 분할하여 SCL을 수동 제어하였다.

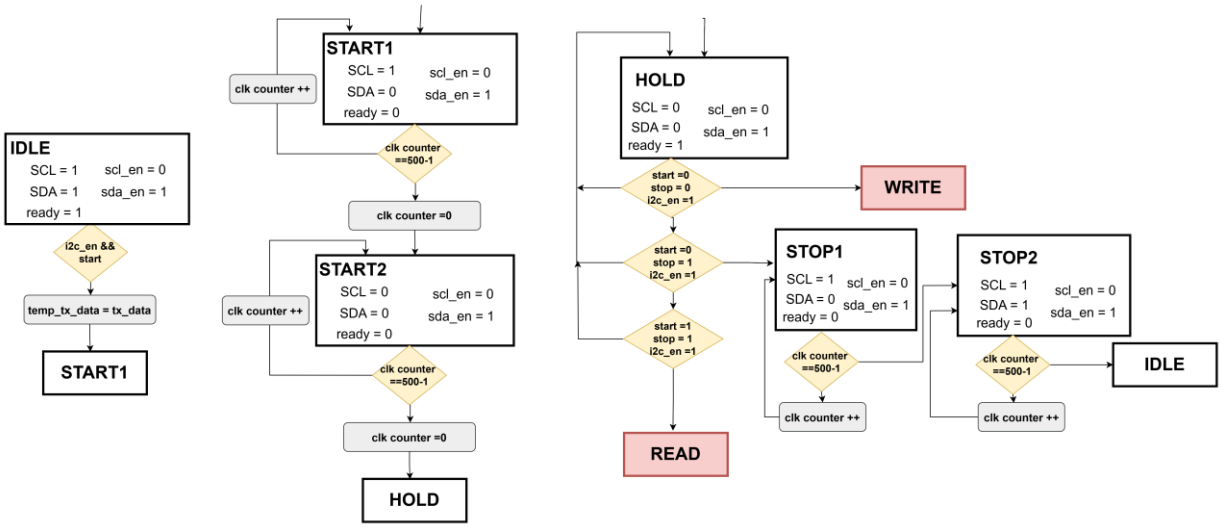
그러나 이러한 방식은 상태 수가 불필요하게 많아져 FSM이 복잡해지고, 상태 전이 조건도 늘어나 가독성과 유지보수성이 떨어졌다.이에 따라 SCL을 제어하는 별도의 Clock Generator 모듈을 분리 설계하고, FSM 에서 해당 Generator 에 enable 신호만 제어하도록 변경하였다. 즉, 상태 전이 흐름은 단순화 되었고, SCL은 enable에 따라 Generator에서 자동 생성되도록 하였다.

이 구조는 다음과 같은 최적화 효과를 가진다.

- FSM의 상태 수 감소 → 코드 및 타이밍 도표 단순화
- SCL 생성 로직의 모듈화 → 재사용성 증가, 독립 테스트 가능
- 타이밍 분리 → 통신 시퀀스 흐름과 물리 신호 타이밍(SCL)을 분리하여 관리

4.5I2C 동작 제어 구조

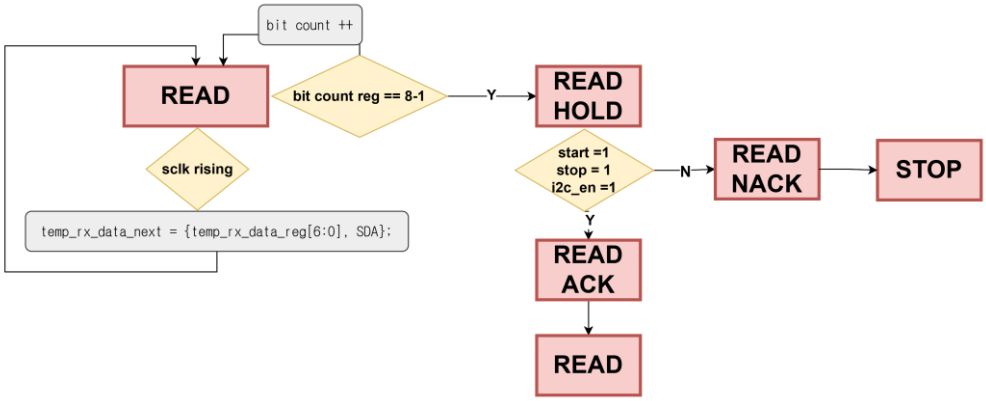
4.5.1 I2C Intf Master



IDLE I2C Intf Master의 default state
모든 신호는 비활성 상태를 유지하며, ready 플래그가 High로 유지되어 새로운 전송 명령을 수용할 수 있다.

START	전송이 필요할 때, interface는 START 상태로 진입하며 이때 I2C 프로토콜에서 정의된 START 조건에 맞게 SDA와 SCL을 제어하며 이후 HOLD state로 천이한다.
HOLD	read, write, stop 중 어떤 동작을 수행할지 명령을 대기하는 상태이다. 이때 i2c_en 신호가 활성화되면, start, stop 비트의 조합에 따라 해당 통신 모드로 즉시 천이된다. WRITE, READ 모드로 진입하는 경우, 내부 Clock Generator의 en 신호가 활성화되어, 1000kHz 주기의 SCL 신호가 자동으로 생성된다.
WRITE	마스터가 슬레이브로 1바이트 데이터를 전송하는 상태이다. 전송이 완료되면 슬레이브로부터 ACK 신호를 수신하고, HOLD state로 천이한다.
READ	슬레이브가 마스터에게 1바이트 데이터를 전송하는 상태이다. 수신이 완료되면 ACK 또는 NACK 신호를 반환하여 수신을 지속할지 결정한다.
STOP	통신을 종료하기 위해 I2C 프로토콜에서 정의된 STOP 조건에 맞게 SDA와 SCL을 제어한다. STOP 조건이 완료되면 FSM은 다시 IDLE 상태로 복귀한다.

4.5.1.1 READ

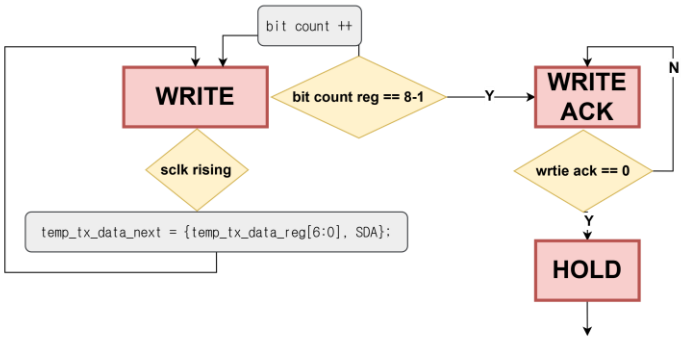


수신이 완료되면 FSM 은 READ HOLD 상태로 진입하여 i2c_en 신호의 활성화를 대기한다.

I2c_en 신호가 활성화 되었을 때, 다음 조건에 따라 FSM 이 동작한다.

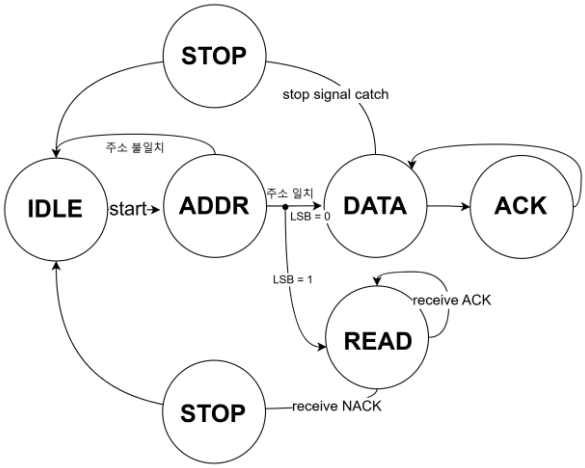
- start 및 stop 신호가 모두 Low 인 경우, 이는 연속적인 데이터 수신을 의미하므로, 마스터는 ACK 신호를 슬레이브로 전송한 후, 다시 READ 상태로 천이하여 다음 바이트를 수신한다.
- start, stop 중 하나라도 High 인 경우, 수신 종료를 의미하므로 마스터는 NACK 신호를 슬레이브로 전송한 뒤, STOP 상태로 천이하여 통신을 종료한다.

4.5.1.2 WRITE



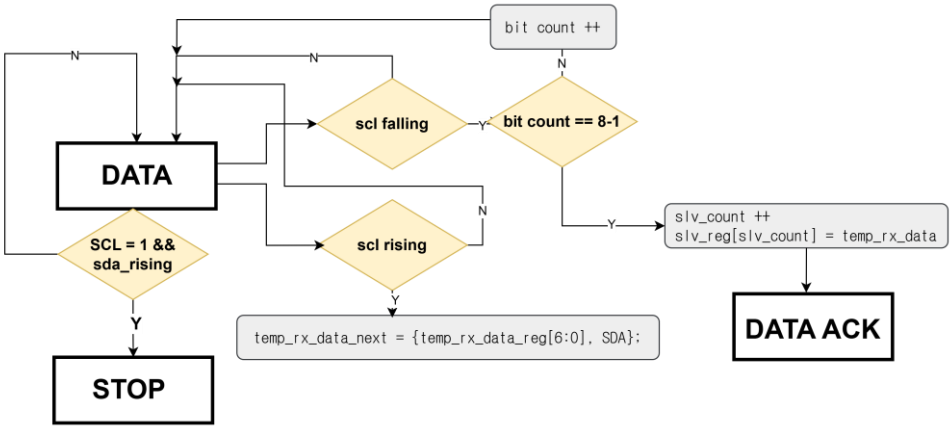
송신의 경우, 1 바이트 송신이 완료되면 슬레이브로부터 ACK 신호를 수신하고 FSM 은 즉시 HOLD 상태로 천이하여 다음 통신 명령의 입력을 대기한다.

4.5.2 Slave



IDLE	I2C Slave의 default state SCL과 SDA 라인을 모니터링하여 START 조건을 대기하며 조건이 충족되면 START 상태로 천이한다.
ADDR	START 상태 이후 수신된 첫 바이트는 슬레이브 주소이다. 수신된 주소가 슬레이브의 고유 주소와 일치하는 경우, 해당 바이트의 LSB를 기준으로 다음 상태를 결정한다.
DATA	ADDR 상태에서 수신한 바이트의 LSB가 0인 경우 진입하는 상태이다. 1바이트 데이터 수신 후 ACK 상태로 천이한다.
ACK	1바이트를 수신 후 정상 수신을 완료했다면 ACK를 전송한다.
READ	ADDR 상태에서 수신한 바이트의 LSB가 1일 때 해당 상태로 진입하며, 1바이트의 데이터를 송신한다. 마스터가 ACK를 반환하는 경우, 이는 연속 송신 요청을 의미하므로 슬레이브는 내부 주소 레지스터를 1 증가시키고 다음 바이트를 송신한다. 반면 NACK를 수신하면 통신 종료를 의미하므로 STOP 상태로 천이한다..
STOP	SCL과 SDA 라인을 모니터링하여 STOP 조건을 대기하며 조건이 충족되면 IDLE 상태로 천이한다.

4.5.2.1 Stuck 상태 방지를 위한 탈출 조건 구현

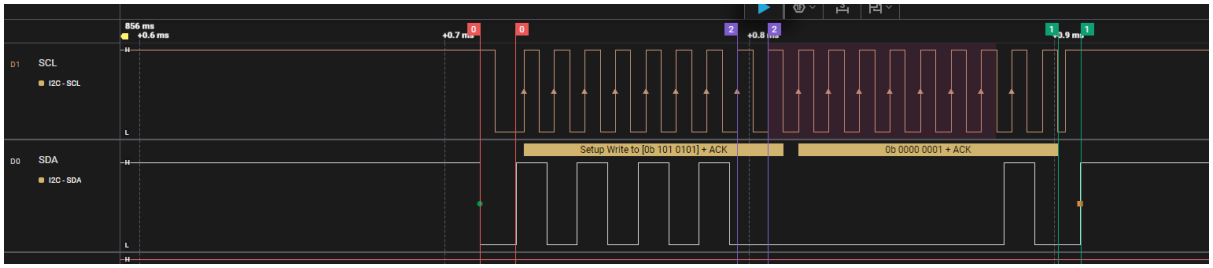


I2C Slave가 데이터 수신 상태에 있을 때, 슬레이브는 마스터의 다음 동작을 수동적으로 기다리는 구조이기 때문에 다음 명령이 주어지지 않으면 FSM이 DATA 상태에 계속 머무르는 문제가 발생할 수 있다.

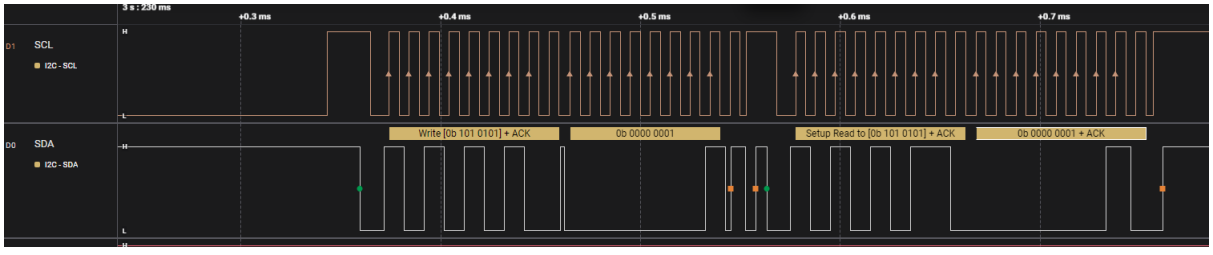
이를 방지하기 위해, DATA 상태에서 STOP 조건을 지속적으로 감시하는 로직을 추가하여 마스터가 데이터를 송신한 후 STOP 조건을 발생시키면, 슬레이브는 이를 감지하여 즉시 STOP 상태로 전이하고 IDLE로 복귀할 수 있다.

4.6 Transaction Examples

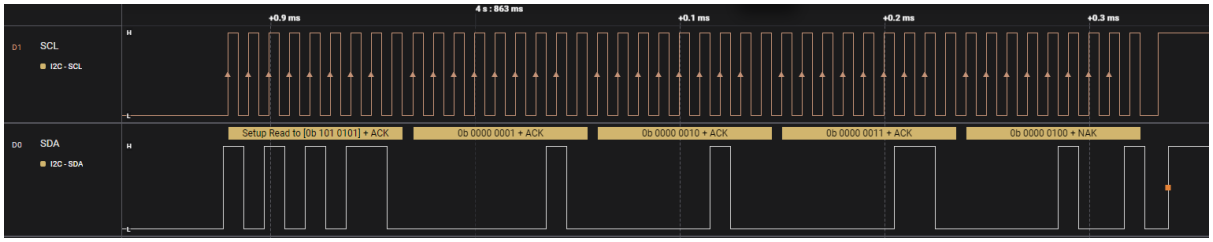
데이터 0x01을 write하는 경우



데이터 0x01을 write하고 read하는 경우



데이터 0x01, 0x02, 0x03, 0x04를 순서대로 write하고 read하는 경우



5. Verification & Testbench

- UVM/단순 TB 둘 중 사용한 방법 요약
- 커버리지·시뮬레이션 결과 스냅샷