

IMU Sensor MPU 6050

Innova Lee(이상훈)
gcccompil3r@gmail.com

Overview

1. IMU 센서인 MPU-6050 을 활용한 결과물들
2. IMU 센서인 MPU-6050 에 대한 조사
3. TI Cortex-R5F 기반의 펌웨어 코드 작성을 통한 MPU6050 실험 및 결과

What can we do with MPU-6050 ?



우선 이러한 쿼드콥터를 만드는데
IMU 센서인 MPU-6050 이 사용되었습니다.

이 프로젝트에 관련한 전반적인 사항은 아래 유튜브 링크에 있습니다.

<https://www.youtube.com/watch?v=j0UD9Vu8qEg&t=21s>

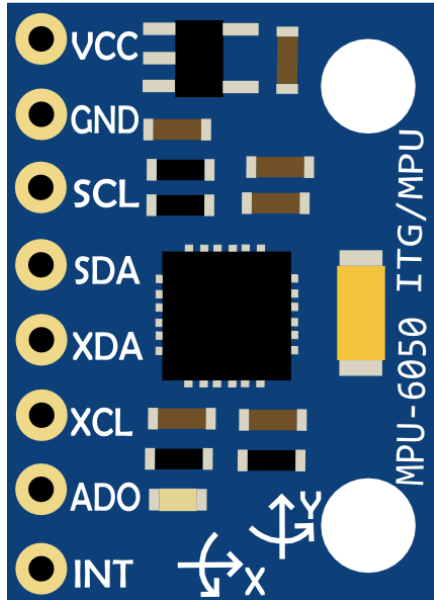
그 외에도 이러한 선박의 자세를 제어하기 위해서도 MPU-6050 을 활용할 수 있었습니다.

사용한 보드는 위의 쿼드콥터와 동일합니다.

참고로 이 프로젝트는 Lidar, Camera 등을 더 추가하여 물로켓으로 표적을 정밀 타격하는 작업까지 추가했습니다.
(이 작업은 다른 기관에서 진행하였습니다)



MPU-6050



아래는 MPU-6050 에 대한
Datasheet 에서 발췌한 스펙 정보입니다.
통신 인터페이스가 MPU-6050 의 경우 I2C 로 제한됩니다.

Primary Differences between MPU-6000 and MPU-6050

Part / Item	MPU-6000	MPU-6050
VDD	2.375V-3.46V	2.375V-3.46V
VLOGIC	n/a	1.71V to VDD
Serial Interfaces Supported	I ² C, SPI	I ² C
Pin 8	/CS	VLOGIC
Pin 9	AD0/SDO	AD0
Pin 23	SCL/SCLK	SCL
Pin 24	SDA/SDI	SDA

Advantage & Disadvantages of MPU-6050

MPU-6050 IMU 센서의 장점과 단점에 대해 알아보도록 하겠습니다.

우선 IMU 의 약자는 Inertial Measurement Unit 의 약자로 관성측정장치에 해당합니다.
종류에 따라서 자이로스코프, 가속도만 있는 6 축 센서와 지자기까지 포함한 9 축 센서가 있습니다.
참고로 6 축의 경우 MPU-6xxx 의 형태가 되며 9 축의 경우 MPU-9xxx 의 형태가 됩니다.

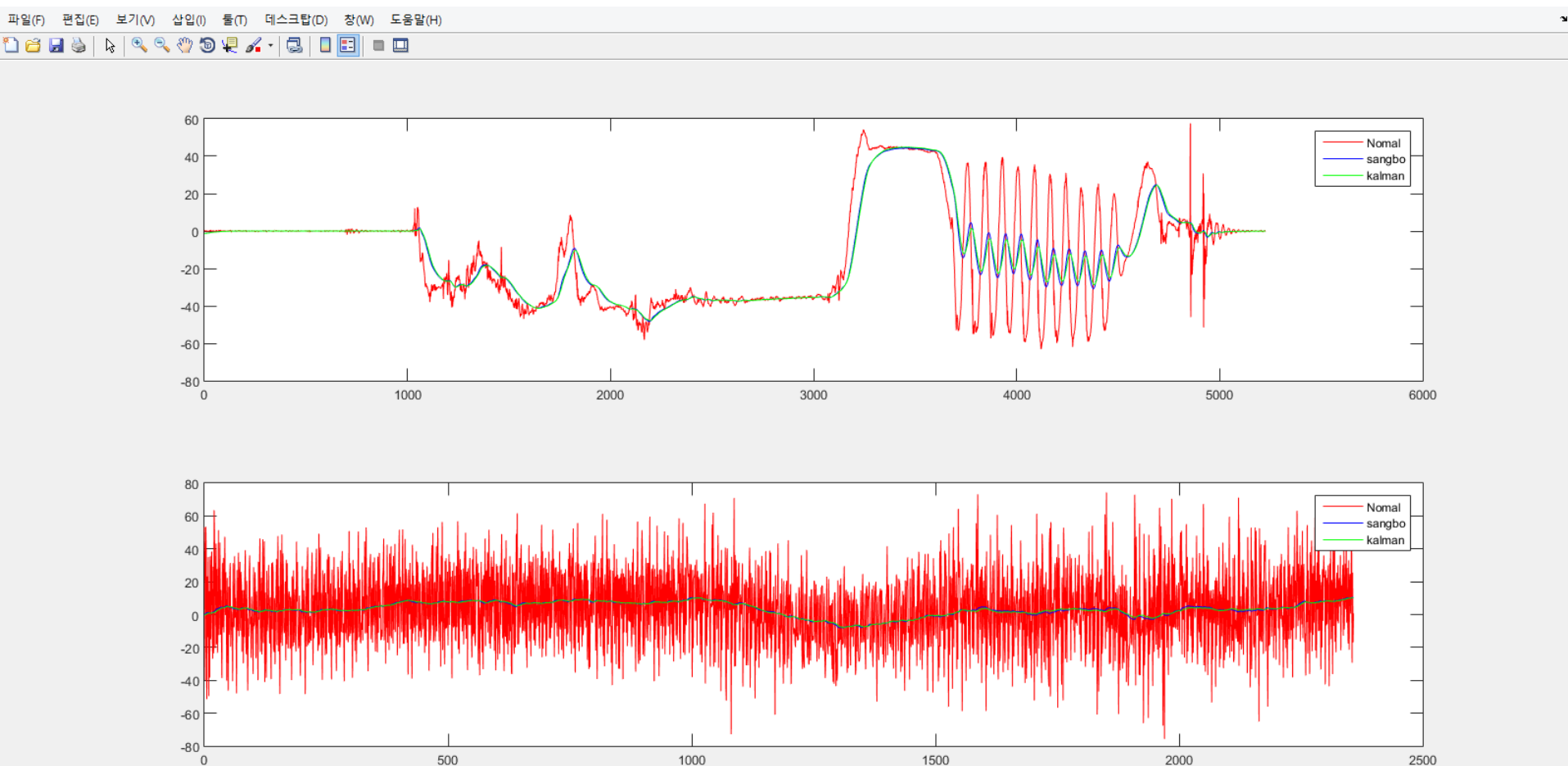
자이로스코프는 각속도(rad/s) 를 측정하며 시간당 몇 degree 를 회전했는지 필요할때 사용합니다.
가속도는 말 그대로 속도에 대한 미분항인 가속도를 측정합니다.
그러므로 적분을 통해 속도와 변위를 계산할 수 있습니다.
(문제는 DSP - 신호처리 관점에서 샘플링 타임 때문에 오차가 누적될 수 있다는 것입니다)
지자기는 자북을 기준으로 자기선속의 세기를 측정하여 자북을 기준으로 얼마나 틀어졌는지를 측정합니다.

모든 전자기기는 Clock 에 동기화되어 동작(FPGA 기본 상식)하므로 이산적으로 데이터를 수집하게 됩니다.
이로 인해서 자이로스코프는 각속도를 측정하지만 적분 오차가 존재하고 지구 자전으로 인한 오차가 존재합니다.
반면 가속도 가속도 센서는 병진운동(등가속도 운동의 법칙 등등)에 대한 측정에 취약합니다.
또한 진동과 같은 외란에도 민감하게 반응한다는 단점이 존재합니다.
그러나 시간이 지나도 오차가 누적되지 않는다는 장점이 존재합니다.

이 둘의 장단점을 상쇄시키는 상보 필터가 존재합니다.
외력에 의한 급격한 변동으로 인한 문제도 발생할 수 있기 때문에
해당 상황을 파형이 잘 따라갈 수 있도록 칼만 필터도 함께 사용하고 있습니다.

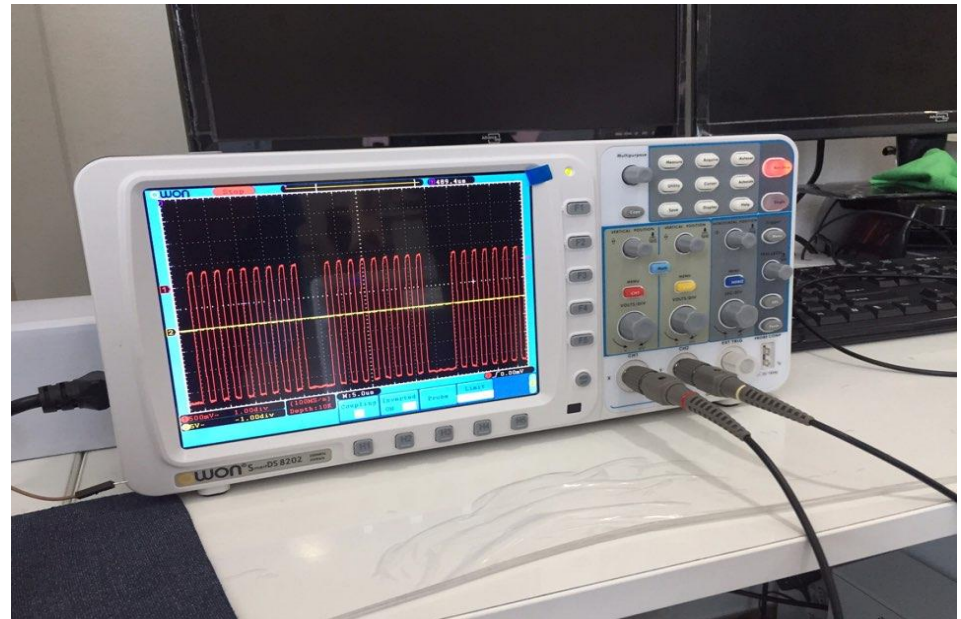
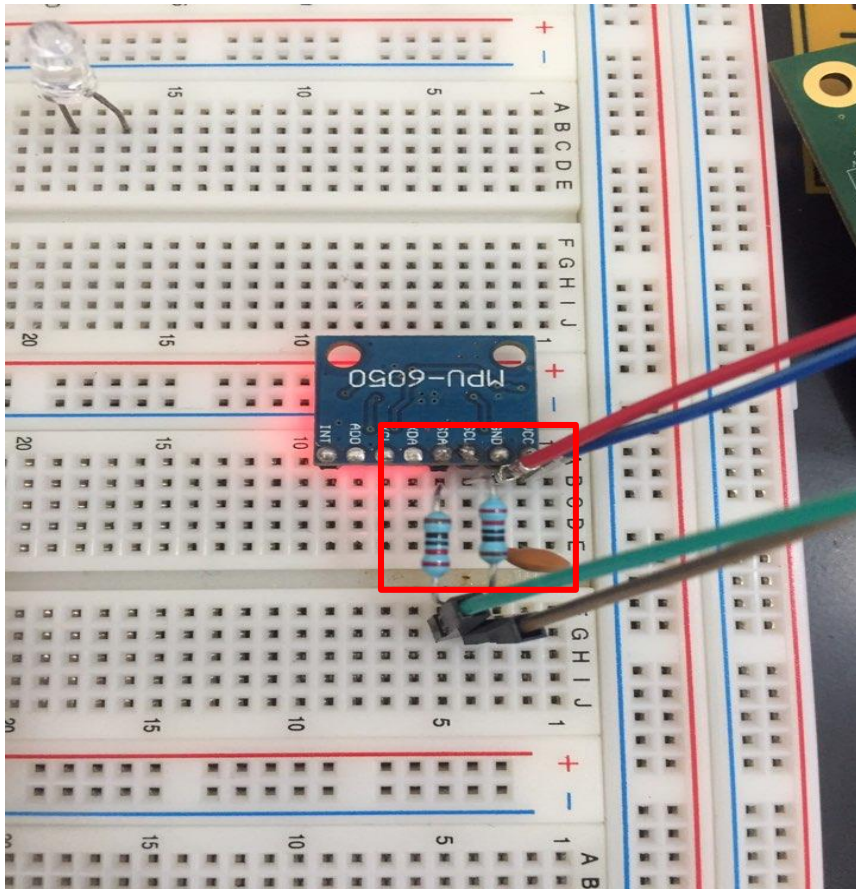
Complementary & Kalman Filter

상보 필터와 칼만필터를 기반으로 MPU-6050 을 Matlab 을 통해 테스트한 결과입니다.



Oscilloscope Based MPU-6050 Test

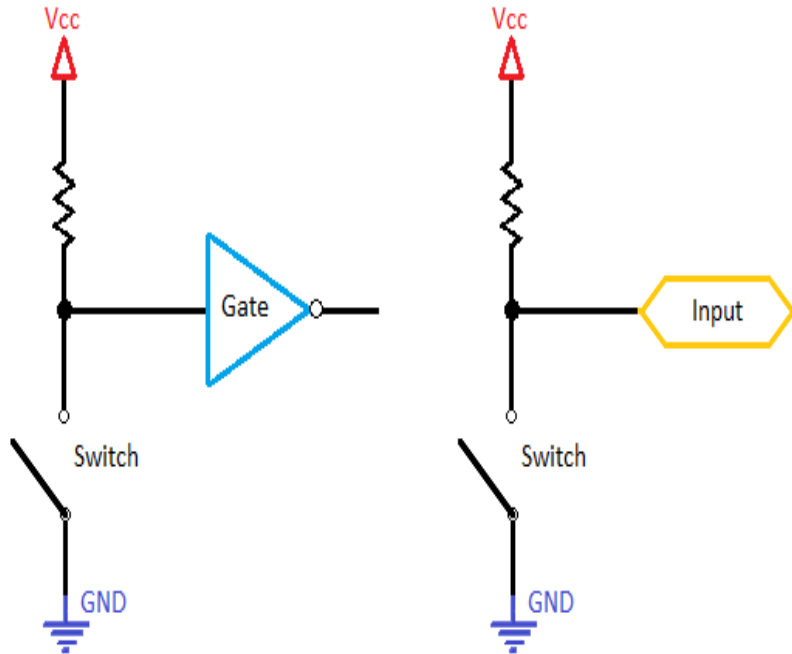
좌측 아래는 노이즈 방지를 위해 풀업 저항을 달아준 모습이며
우측은 MCU 를 통해 통신을 정상적으로 수행하고 있음을 볼 수 있는 파형입니다.



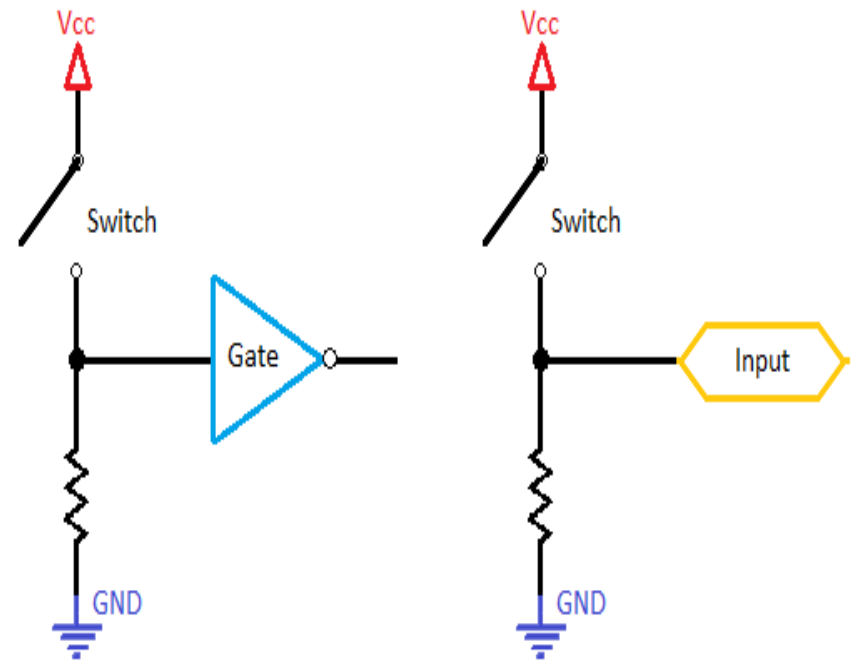
Mpu6050의 경우 I2C 통신을 하는데
이때 SCL, SDA 핀에 풀업 저항을 달아주어
Floating 상태를 방지하였습니다.

Pull-up Pull-down Resistor

Pull-up Resistor

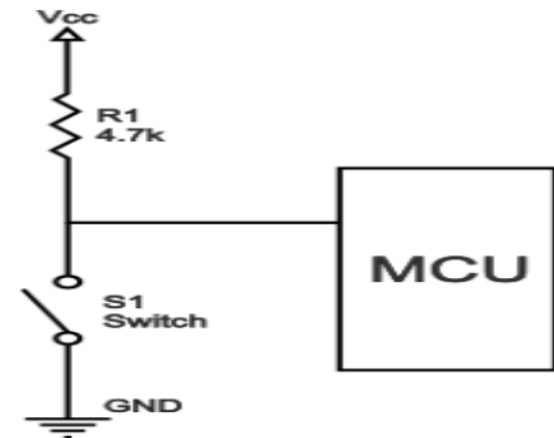
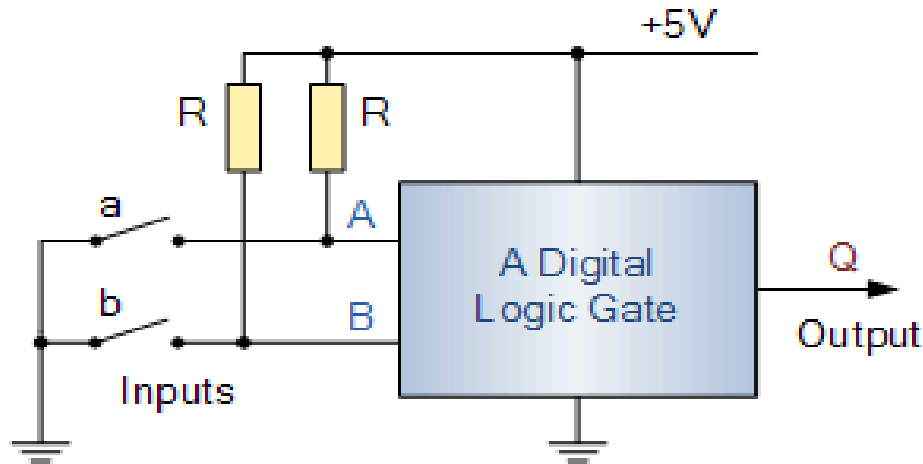


Pull-down Resistor



1. 입력 일 때 floating 상태를 방지하고 high, low를 올바르게 인식하기 위해서 사용합니다.
2. 입력 일 때 입력신호를 사용하지 않으나 나중에 사용하려는 경우에 사용합니다.
3. 출력 일 때 오픈 컬렉터, 오픈 드레인 회로일 경우 기준 전압을 위한 회로로 사용합니다.
4. 출력 일 때 출력 전류를 증대 시키려는 경우에 사용합니다.
5. 초기값을 주기 위해서 사용합니다.

What is Pull-up Resistor ?



풀업 저항은 모든 조건의 핀에서 잘 정의 된 논리 레벨을 보장하기 위해 논리 회로에 사용되는 저항입니다.

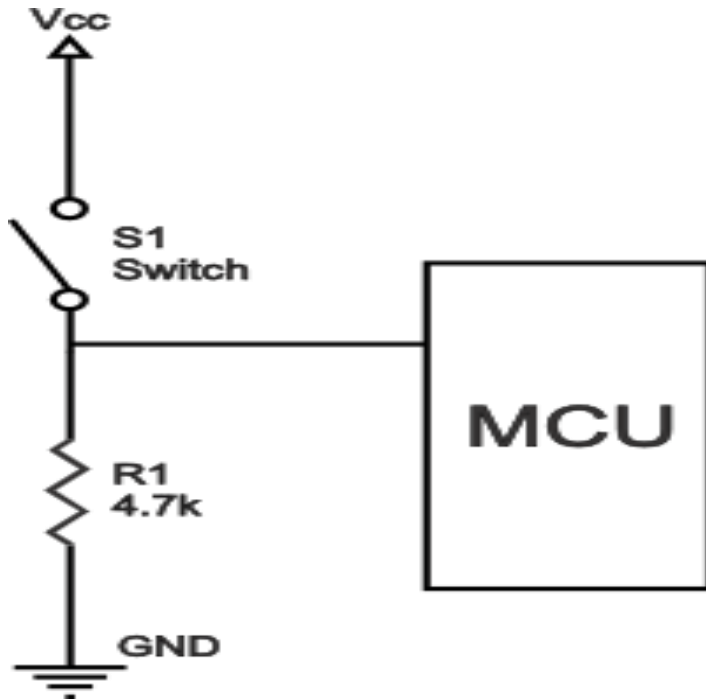
디지털 논리 회로는 high, low, floating의 3가지 로직 상태를 가지고 있습니다.

high 임피던스 상태는 high 또는 low 로직 레벨로 끌어 올려지지 않았을 때 발생하지만 대신 floating 상태가 됩니다.

Floating 상태를 방지하기 위해 풀업 저항을 사용하여 논리적 high 상태로 끌어 올리는데 사용합니다.

What is Pull-down Resistor ?

풀다운 저항은 풀업 저항과 동일한 방식으로 작동하지만 핀을 논리적으로 낮은 값으로 끌어 올리는 점이 다릅니다. 풀다운 저항은 논리 회로의 임피던스 보다 큰 저항을 가져합니다. 그렇지 않으면 전압을 너무 많이 끌어 당길 수 있고 핀에서의 입력 전압은 로직 high의 일정한 값으로 유지됩니다.

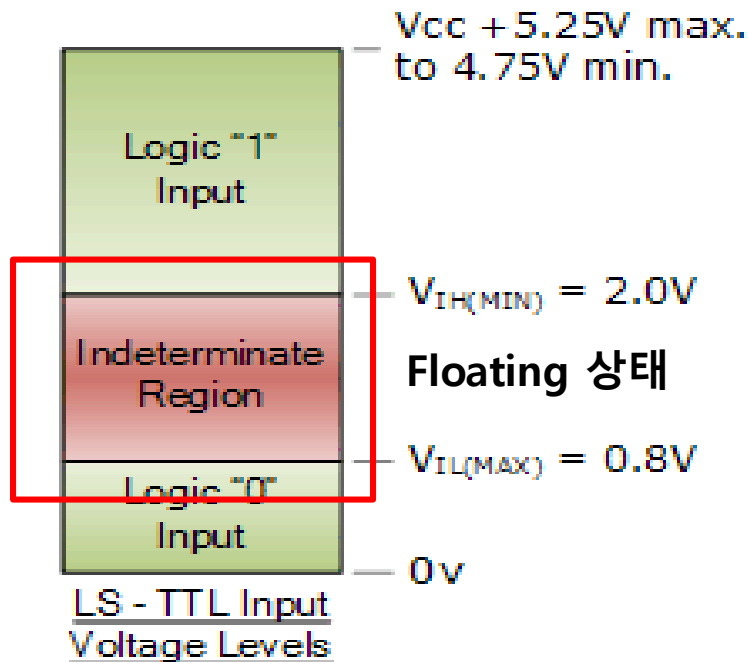


NOTE : 풀다운 저항 보다 풀업 저항을 많이 사용하는 이유는 풀업 저항이 풀다운 저항보다 노이즈에 강하기 때문에 풀업 저항을 많이 사용합니다.

What is Floating State ?

Floating 상태란 high 또는 low로 인식이 안 되는 경우를 말합니다.

이런 상태를 Floating 되었다라고 하며 Floating 상태는 잡음에 매우 취약해지므로 시스템이 불안정해 집니다.



5V 기준으로 0.8V 이하는 low, 2.0V 이상은 high로 인식합니다.
0.8V ~ 2.0V 사이의 전압은 low, high 상태도 아닌
인식을 못하는 상태이며 이 것을 floating 상태라고 합니다.

TI Cortex-R5F Architecture

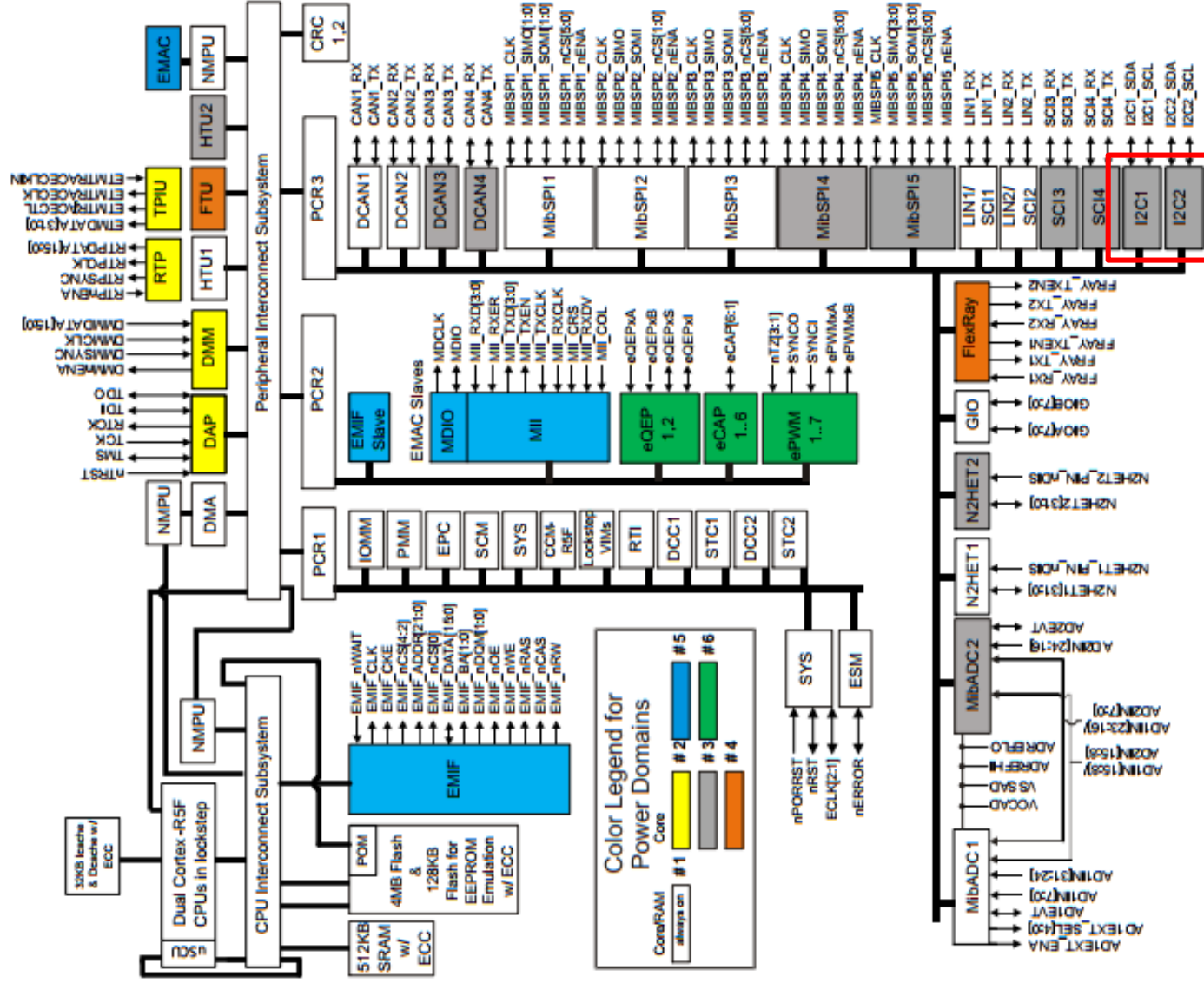
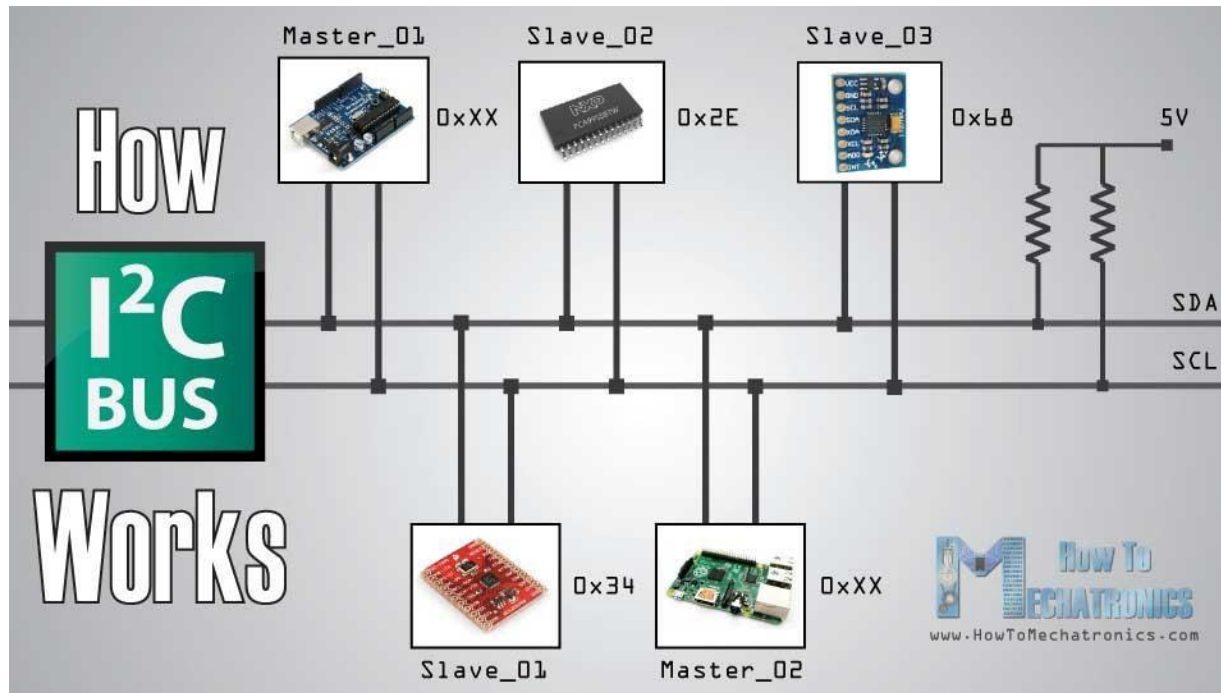


Figure 1-1. Functional Block Diagram

Copyright © 2016, Texas Instruments Incorporated

I2C(Inter-Integrated Circuit)

이제 I2C 라는 저속의 주변장치 사이의 통신을 위해 필립스에서 만든 규격을 살펴보도록 하겠습니다.
앞서서 설명하였듯이 일대일 통신의 단점을 보완하기 위해 SPI 가 나타났는데 I2C 도 마찬가지로 해당합니다.
I2C 는 저속의 여러 주변장치들이 최소한의 연결선만을 사용하여 통신할 수 있도록 만들어졌습니다.
반면 SPI 는 고속의 통신을 목표로 하는 점에서 차이가 있습니다.



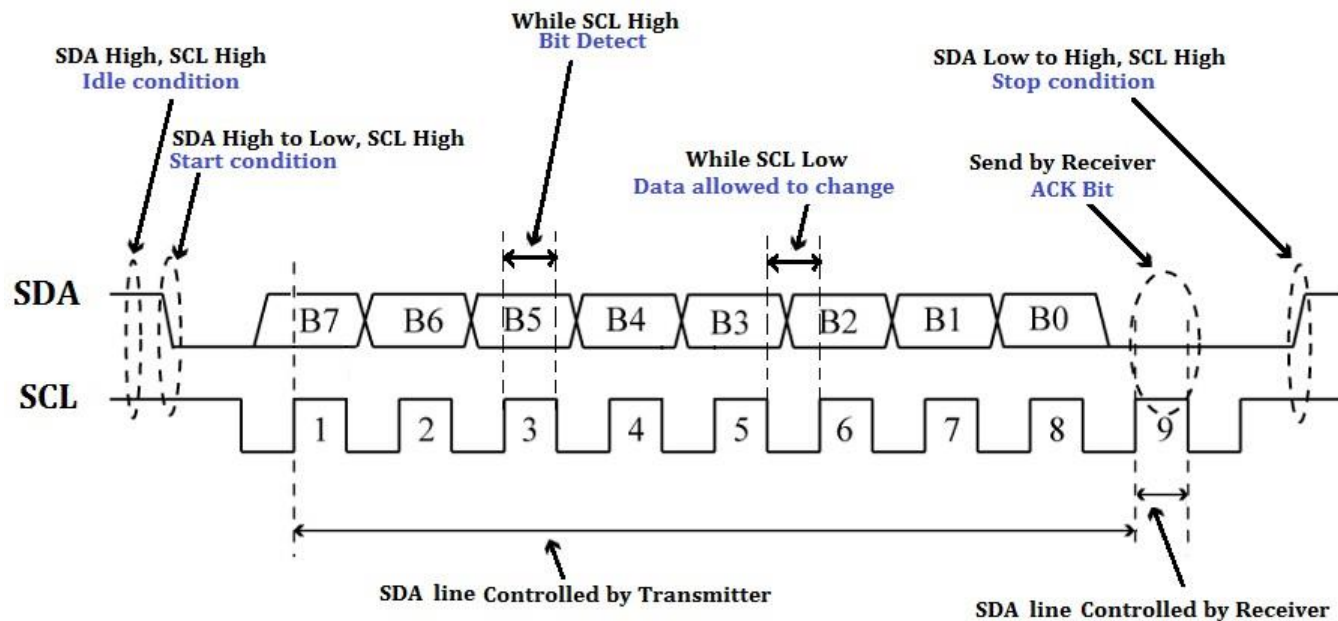
SPI	I2C
Four wires	Two wires
Full duplex	Half duplex
Higher throughput (then I2C)	Lower throughput
Synchronous protocol	Synchronous protocol
No slave acknowledgment	Acknowledgment
Simple	Complicated

I2C 는 SPI 와 마찬가지로 Master Slave 구조를 가지지만 연결된 Slave Device 의 개수와 무관하게 Data 전송을 위한 SDA(Serial Data), Clock 전송을 위한 SCA(Serial Clock) 2 개의 선만을 필요로 합니다.

SDA 는 Data 가 전송되는 통로로 하나만 존재합니다.
송신과 수신은 SDA 를 통해 이루어지므로 송신과 수신은 동시에 이루어질 수 없는 반이중(Half-Duplex) 방식을 사용합니다.

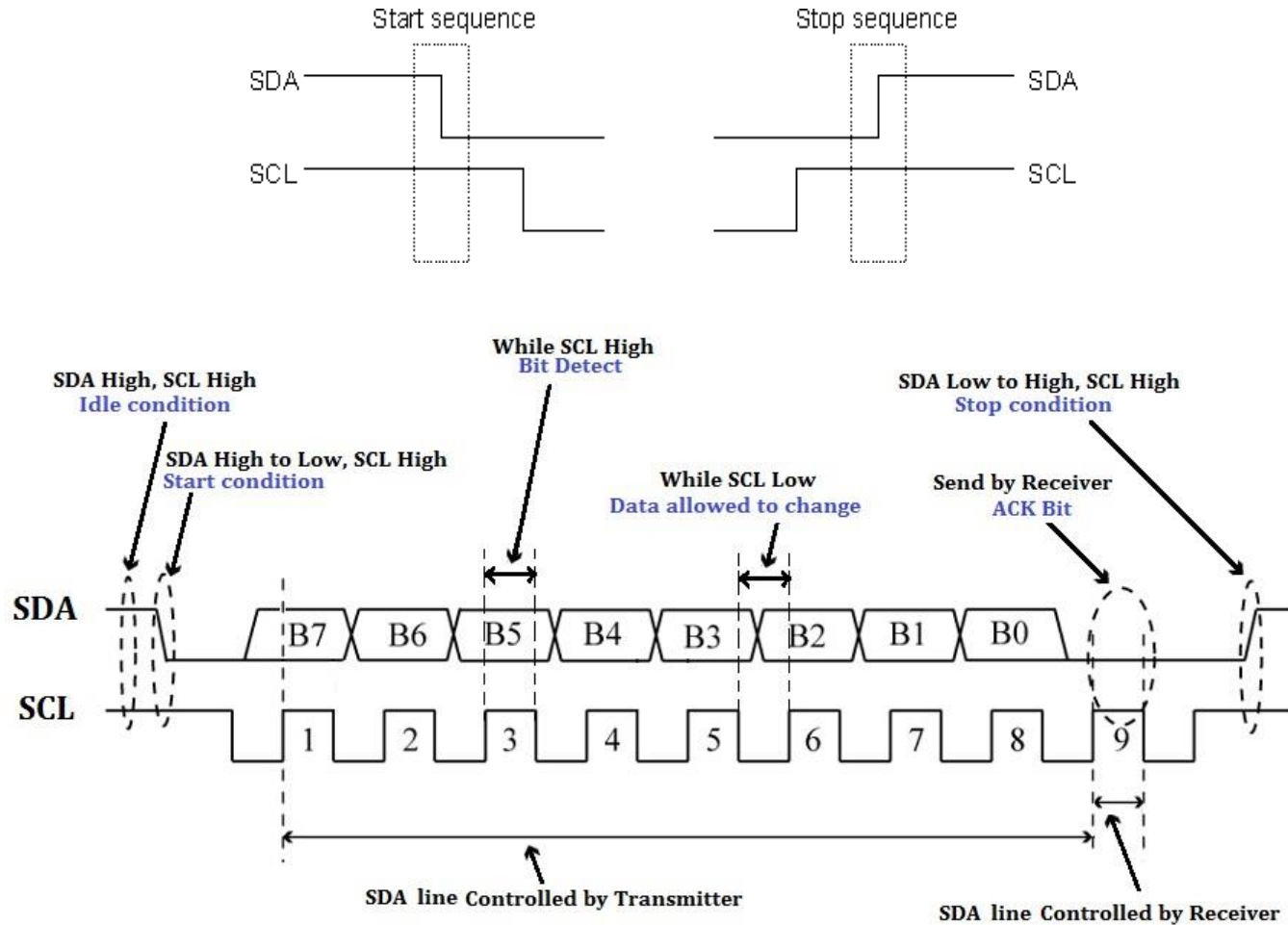
SCL 은 Clock 전송을 위한 통로로 SPI 에서와 마찬가지로 Master 가 Clock 을 생성하고 Data 전송의 책임을 가지고 있습니다.

I2C 역시 SPI 와 마찬가지로 동기 방식으로 동작하지만 SPI 에서와는 달리 위상과 극성에 따른 여러 가지 전송 모드가 존재하지 않습니다.
수신된 Data 는 SCL 이 HIGH 인 경우에만 Sampling 이 가능합니다.
따라서 SCL 이 HIGH 인 경우 SDA 의 Data 는 안정된 상태에 있어야만 합니다.
Data Transition 은 SCL 이 LOW 인 상태에서만 가능합니다.



그러나 SCL 이 HIGH 인 경우에도 전이가 발생하는 두 가지 예외 상황이 있는데 데이터 전송 시작과 종료를 나타내는 경우에 해당합니다.

SCL 이 HIGH 인 경우 SDA 가 HIGH 에서 LOW 로 바뀌는 경우는
데이터가 전송되기 시작함을 나타내고
LOW 에서 HIGH 로 바뀌는 경우는
데이터 전송이 끝났음을 나타냅니다.



I2C 는 SPI 와 마찬가지로 1:N 통신을 지원합니다.

SPI 에서는 각각의 Slave 가 전용의 SS(Slave Select)

혹은 CS(Chip Select) Line 을 가지고 HW 적으로 Data 를 송수신할 Slave 를 선택합니다.

반면 I2C 에서는 Slave 가 고유의 주소를 가지고 SW 적으로 Data 를 송수신할 Slave 를 선택합니다.

I2C 는 7 bit Address 를 사용합니다.

7 bit 가 애매하다고 생각할지 모르지만 나머지 한 bit 는 읽기와 쓰기를 선택하기 위해 사용됩니다.

HIGH 값이 주어진 경우 Master 는 지정한 Slave 로부터 전송되는 Data 를

SDA Line 에서 읽어들이는 것임을 나타내고 LOW 값이 주어진 경우

Master 는 지정한 Slave 로 SDA Line 을 통해 Data 를 전송할 것입니다.

7 bit 의 주소 중 0000 000 은 Master 가 여러 개의 Slave 에

동시에 Message 를 보내는 용도(General Call)로 사용하기 위해 예약되어 있으므로 사용할 수 없습니다.

1111 xxx 주소 역시 이후 사용을 위해 예약되어 있어 사용할 수 없습니다.



예약되어 있는 신호들의 리스트를 살펴보자면 아래와 같습니다.

Address	Purpose
0000000 0	General Call – addresses all devices supporting the general call mode
0000000 1	Start Byte
0000001 X	CBUS addresses
0000010 X	Reserved for different bus formats
0000011 X	Reserved for future purpose
00001XX X	High-speed Master code
11110XX X	10-bits slave addressing
11111XX X	Reserved for future purposes

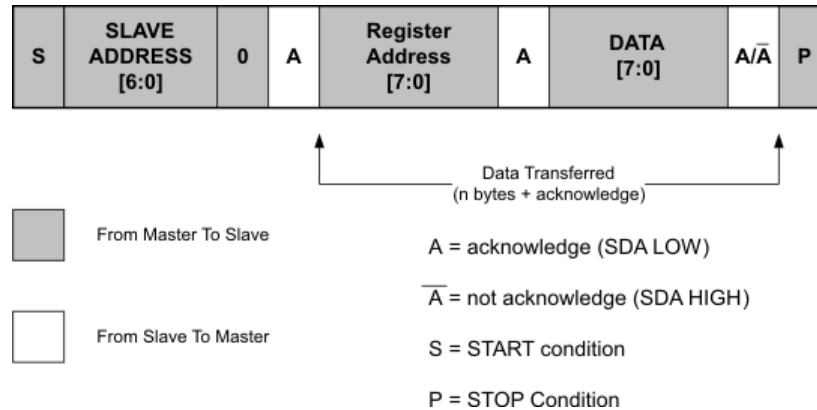
Table 1: I²C addresses reserved for special puposes

Master 가 시작 신호(S) 와 7 bit 주소를 보내고 LOW 값(W)을 보냈다면
지정한 Address 를 가지는 Slave 는 Master 가 1 byte 의 Data 를 전송할 것임을 인식하고 수신을 대기하게 됩니다.
또한 마지막에 HIGH 값®을 보냈다면 지정한 Address 를 가지는 Slave 는 Master 로 1 byte 의 Data 를 전송할 것입니다.

Data 를 수신한 Device 는 Data 를 수신했음을 알려주어야 합니다.
I2C 는 Byte 단위로 Data 를 전송하며 8 bit 의 Data 가 전송된 이후 SDA Line 은 HIGH 상태에 있습니다.
Byte 단위 Data 가 전송된 이후 수신 장치는 정상적인 수신을 알리기 위해 9 번째 bit 를 송신 장치로 전송합니다.
LOW 값은 ACK(Acknowledgement) bit 로 수신 장치가 송신 장치에 정상적으로 Data 를 수신했음을 알려주기 위해 사용되며
HIGH 값은 NACK bit 로 정상적인 Data 수신 이외의 상황이 발생했음을 알려주기 위해 사용됩니다.

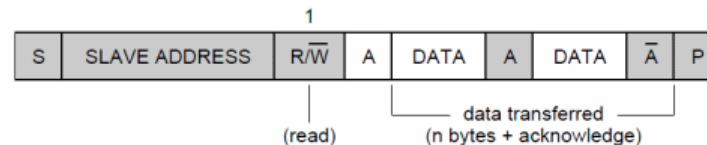
Data 를 수신한 장치가 정상적으로 Data 를 수신하지 못하면 수신 확인인 ACK bit 를 전송하지 않아
SDA 는 HIGH 상태를 유지하고 있을 것이므로 별도로 NACK bit 를 전송하지 않아도 효과는 동일합니다.

Master 에서 Slave 로 n byte 의 Data 를 송신하는 경우를 생각해보겠습니다
 먼저 Data 전송 시작 bit(START) 와 7 bit Address, 그리고 Data 송신 신호를 보냅니다.
 지정된 주소의 Slave 는 Data 를 수신할 준비를 시작하면서 9 번째 bit 인 수신 확인 신호를 보냅니다.
 이후 Master 는 n byte 의 Data 를 송신하게 되며 매 byte 가 수신된 이후 Slave 는 수신 확인 bit 를 Master 로 전송할 것입니다.
 Data 전송이 끝나면 Data 송신 종료 bit(STOP) 을 전송함으로써 통신을 끝냅니다.



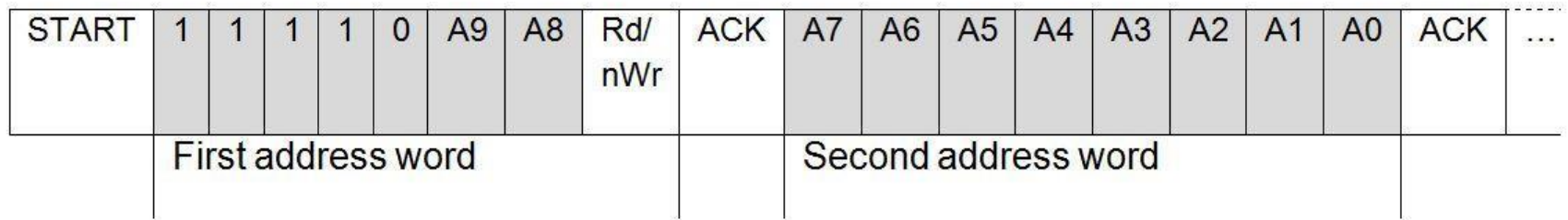
Master 가 Slave 로부터 n byte 의 Data 를 수신하는 경우도 송신과 비슷합니다.
 먼저 Data 전송 시작 bit 와 7 bit Address, 그리고 Data 수신 신호를 보냅니다.
 지정된 Address 의 Slave 는 Data 를 송신할 준비를 시작하면서 9 번째 bit 인 수신 확인 신호를 보냅니다.
 이후 Slave 는 n byte 의 Data 를 송신하며 매 byte 가 수신된 이후에 Master 는 수신 확인 bit 를 Slave 로 전송합니다.
 마지막 n 번째 byte 가 수신된 이후 Master 는 NACK 를 Slave 로 전송하여 수신이 완료되었음을 알립니다.

데이터 전송이 끝나면 Data 송신 종료 bit 를 전송함으로써 통신을 끝냅니다.
 Data 송신의 경우와 마찬가지로 Data 전송이 끝났음을 나타내는 종료 bit(STOP) 는 Master 에 의해 보내집니다.



10 bit 주소는 아래와 같이 전송합니다.

추가 정보는 <http://www.byteparadigm.com/applications/introduction-to-i2c-and-spi-protocols/> 에서 보다 자세하게 확인할 수 있습니다.



10 bits address:

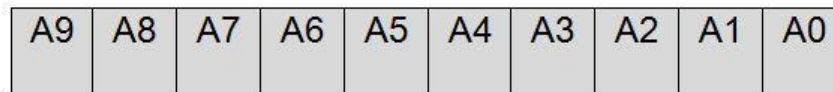
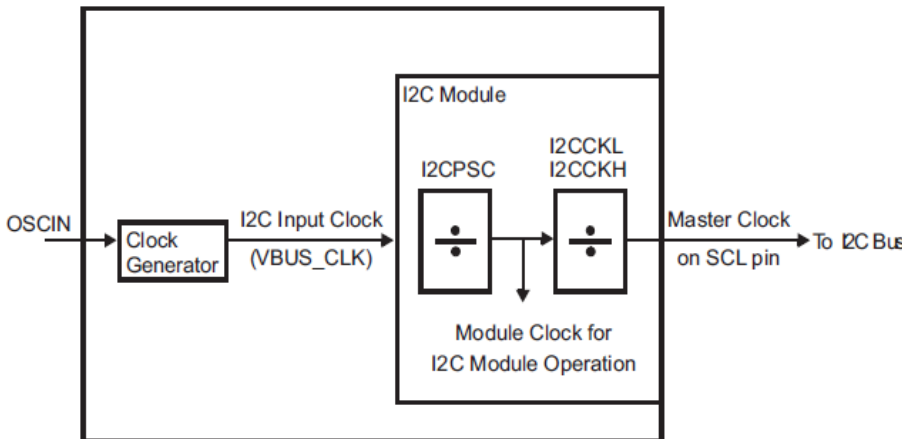


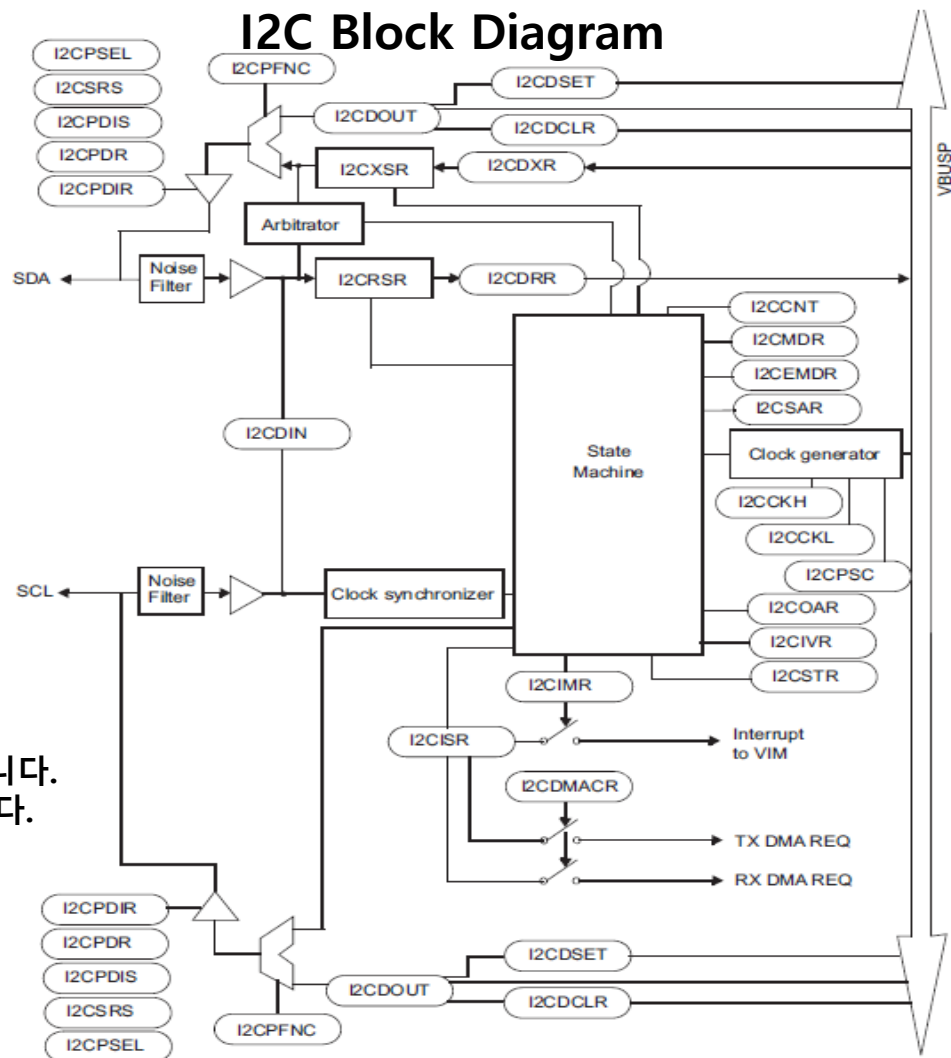
Figure 7: I²C 10-bits addressing. A 10-bits address is split into 2 words. The first word contains a conventional code on its 5 most significant bits to mark a 10-bits address, followed by the 2 MSBs of the 10-bits address and the Rd/nWR bit. The second address word contains the 8 least significant bits of the 10-bits address. This addition ensures backward compatibility with the 7-bits addressing scheme.

TI Cortex-R5F I2C Summary

Figure 30-3. Clocking Diagram for the I2C Module

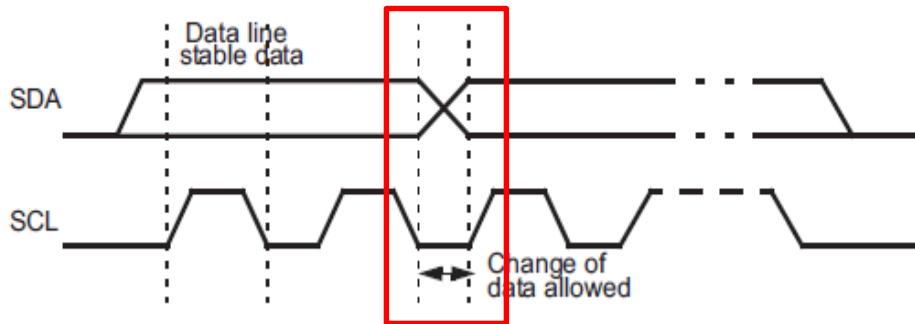


I2C 모듈은 장치 clock 생성기에서 생성된
입력 clock을 사용하여 모듈 clock과 마스터 clock을 생성합니다.
I2C 입력 clock은 디바이스 주변 장치 클록 (VBUS CLK)입니다.



TI Cortex-R5F I2C Conditions

Figure 30-4. Bit Transfer on the I2C Bus

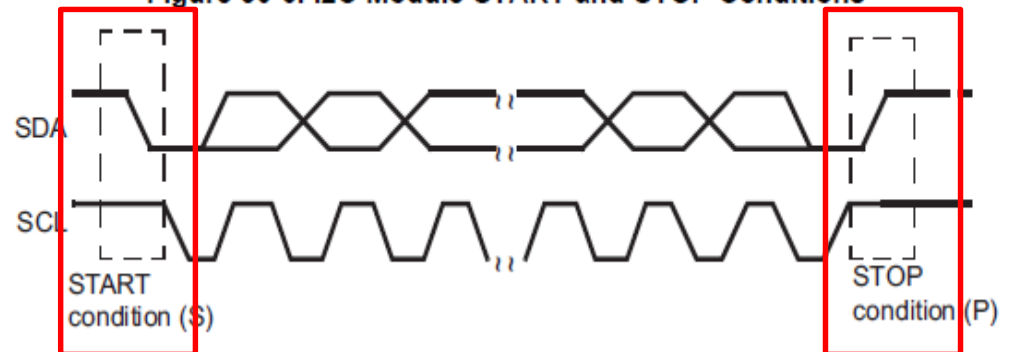


데이터 라인의 high 및 low 상태인 SDA는 clock 신호가 낮은 경우에만 바뀔 수 있습니다.

START 조건은 SCL high 일때
SDA 라인에서 high-low 전환으로 정의 됩니다.

STOP 조건은 SCL high 일때
SDA 라인에서 low- high 전환으로 정의 됩니다.

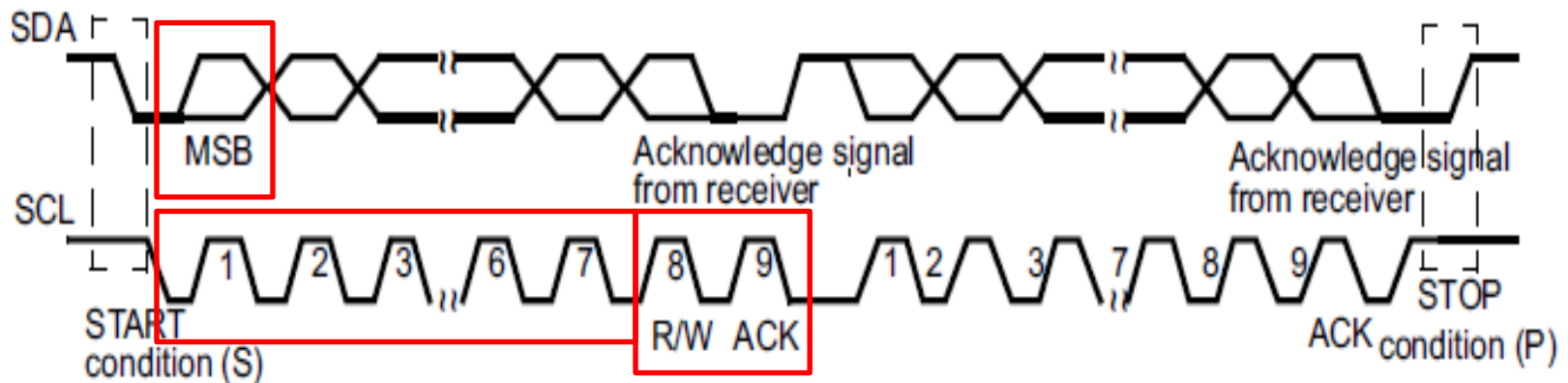
Figure 30-5. I2C Module START and STOP Conditions



TI Cortex-R5F I2C Data Transfer

데이터는 최상위 비트(MSB)로
먼저 전송 됩니다.

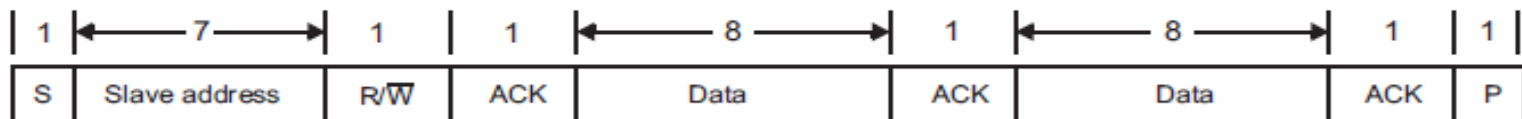
Figure 30-6. I2C Module Data Transfer



START 조건 (S) 이후 첫 번째 byte는 항상 7bit address를 가지며 R/W bit 또는 8 bit의 데이터를 구성됩니다. 첫 번째 byte의 여덟 번째 bit R/W는 데이터의 방향을 결정합니다. R/W bit가 0일 때, 마스터는 선택된 슬레이브 장치에 데이터를 쓰고 (전송), R/W bit가 1일 때, 마스터는 슬레이브로 부터 데이터를 읽습니다(수신). ACK의 경우 메시지 뒤에 확인 bit로 사용됩니다.

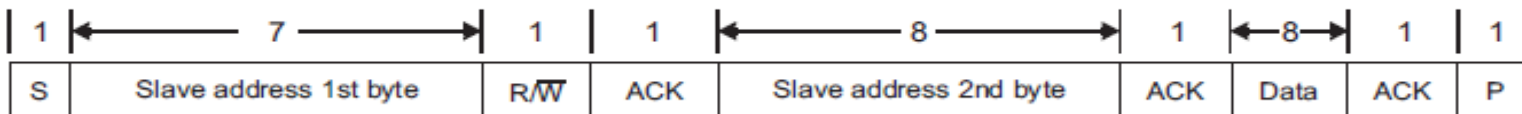
TI Cortex-R5F I2C Addressing Format

Figure 30-7. I2C Module 7-Bit Addressing Format



7 비트 주소 지정 형식에서 START 조건 이후 첫 번째 바이트는 7 비트의 슬레이브 주소와 R/W 비트(LSB)가 옵니다. R/W 비트는 데이터 전송 방향을 결정합니다. 그 후에는 ACK 비트 확인 비트 사용 후에 8 비트의 데이터가 사용됩니다.

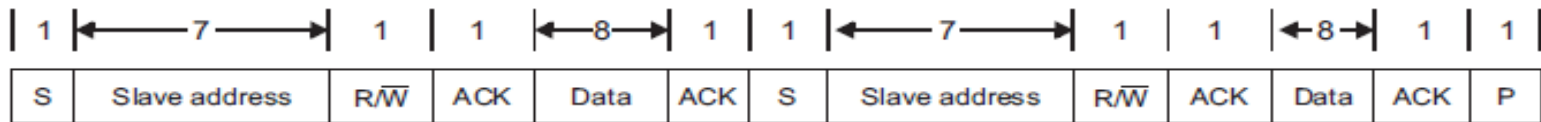
Figure 30-8. I2C Module 10-bit Addressing Format



10 비트 주소 지정 형식은 7 비트 주소 형식과 유사하지만 마스터는 두 개의 개별 바이트 전송으로 슬레이브 주소를 보냅니다. 첫 번째 바이트에서는 7 비트의 슬레이브 주소를 사용 후에는 두 번째 바이트에서는 10 비트 슬레이브 주소의 나머지 8 비트를 사용합니다. 2개의 슬레이브 주소 바이트 전송 후에는 데이터를 사용하게 됩니다.

TI Cortex-R5F I2C Repeat & Free Format

Figure 30-9. I2C Module 7-Bit Addressing Format with Repeated START

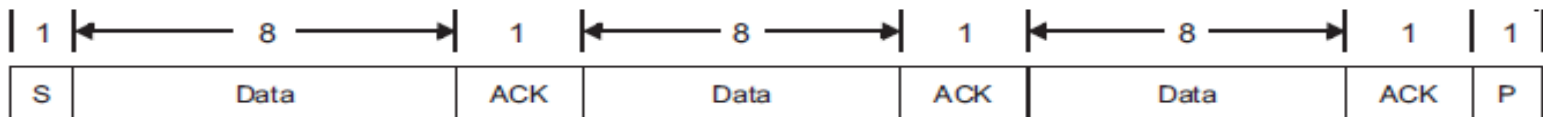


마스터는 또 다른 START 조건을 동작 시킬 수 있습니다.

마스터는 STOP 조건을 생성하기 전에 임의의 수의 데이터를 송수신 할 수 있습니다.

반복되는 START 조건은 7비트 또는 10비트 주소로 자유롭게 사용이 가능합니다.

Figure 30-10. I2C Module in Free Data Format

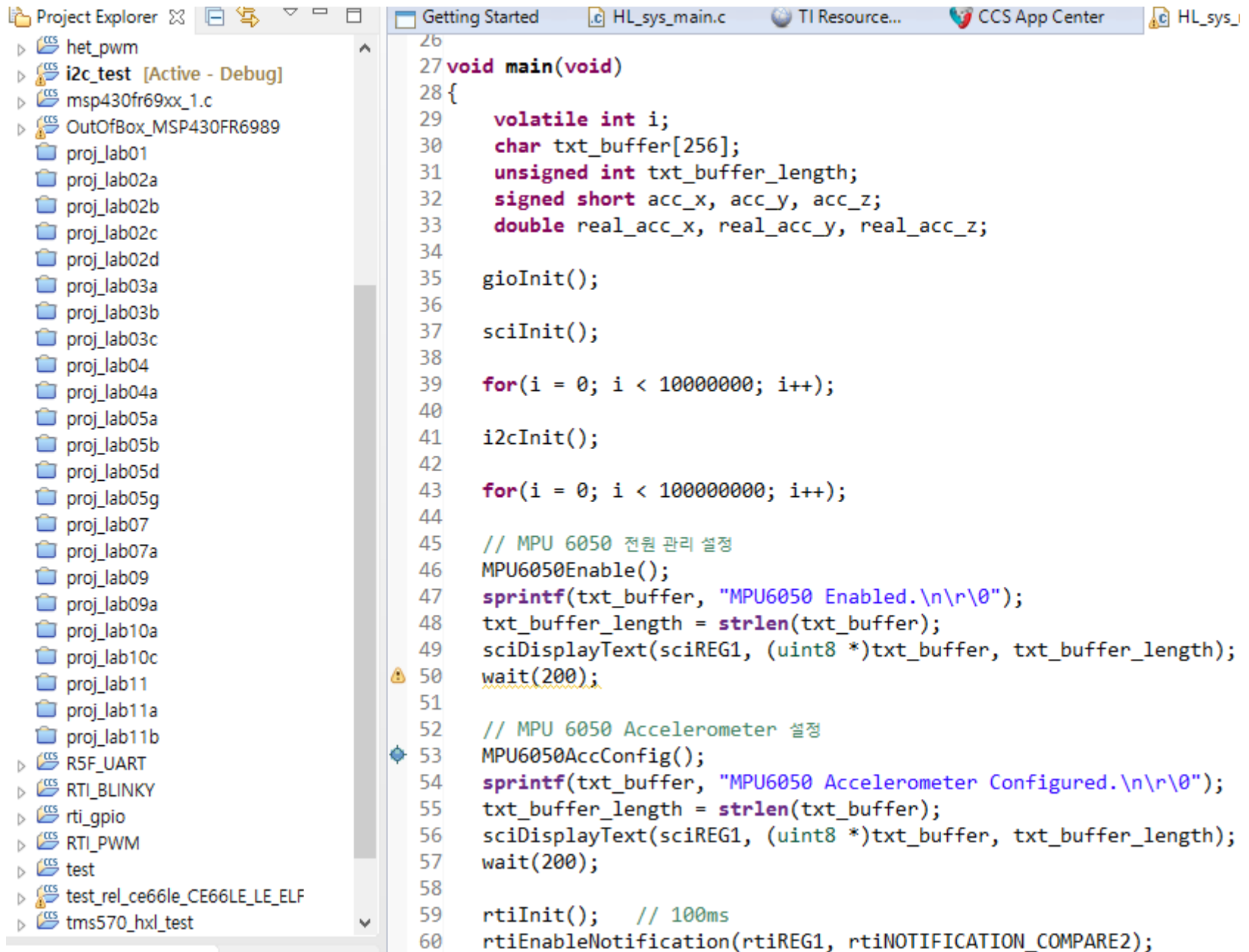


이 방식은 START 조건 이후에 주소 및 데이터 방향 비트가 없이 데이터만을 사용하는 방식입니다.

따라서 송수신기와 수신기는 모두 자유 데이터 형식을 지원해야하며

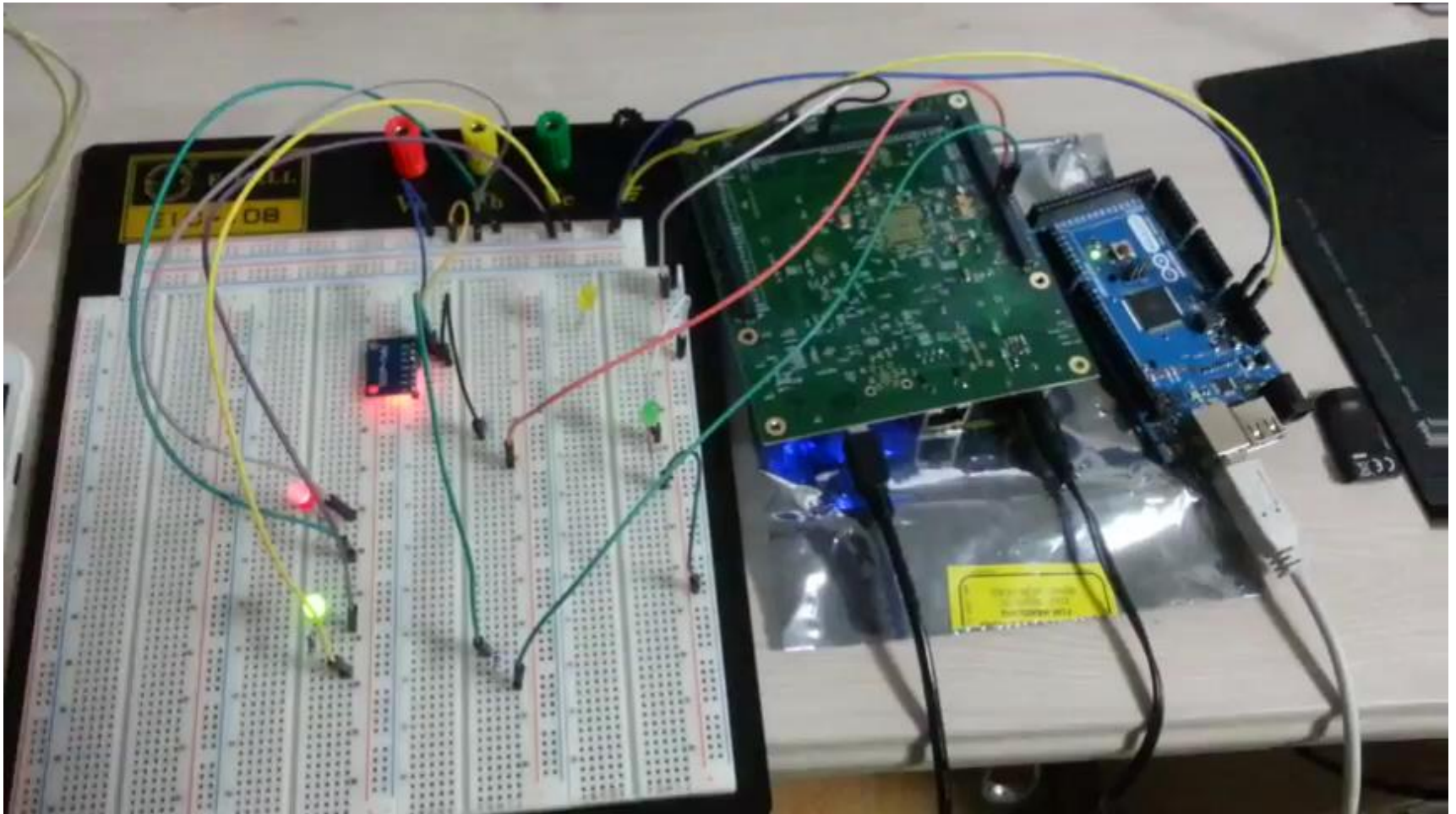
데이터 전송의 방향은 일정하게 진행되어야 합니다.

위 정보들에 기반하여 아래와 같이 C 언어 기반으로 펌웨어 코드를 작성하여 작업을 완료하였습니다.



```
26
27 void main(void)
28 {
29     volatile int i;
30     char txt_buffer[256];
31     unsigned int txt_buffer_length;
32     signed short acc_x, acc_y, acc_z;
33     double real_acc_x, real_acc_y, real_acc_z;
34
35     gpioInit();
36
37     sciInit();
38
39     for(i = 0; i < 10000000; i++);
40
41     i2cInit();
42
43     for(i = 0; i < 100000000; i++);
44
45     // MPU 6050 전원 관리 설정
46     MPU6050Enable();
47     sprintf(txt_buffer, "MPU6050 Enabled.\n\r\0");
48     txt_buffer_length = strlen(txt_buffer);
49     sciDisplayText(sciREG1, (uint8 *)txt_buffer, txt_buffer_length);
50     wait(200);
51
52     // MPU 6050 Accelerometer 설정
53     MPU6050AccConfig();
54     sprintf(txt_buffer, "MPU6050 Accelerometer Configured.\n\r\0");
55     txt_buffer_length = strlen(txt_buffer);
56     sciDisplayText(sciREG1, (uint8 *)txt_buffer, txt_buffer_length);
57     wait(200);
58
59     rtiInit(); // 100ms
60     rtiEnableNotification(rtiREG1, rtiNOTIFICATION_COMPARE2);
```

MPU-6050 을 실험하기 위해
임시 방편으로 5V 전원을 아두이노에서 빌렸음.



References

1. TI Cortex-R5F(TMS570LC4357HDK) Reference Manual
<http://www.ti.com/lit/ds/spns195c/spns195c.pdf>
2. TI Cortex-R5F(TMS570LC4357HDK) Technical Reference Manual
<http://www.ti.com/lit/ug/spnu563/spnu563.pdf>