

블콩카

자율주행차 그 첫번째 이야기

조장: 정유경(블콩)

팀원: 안상재, 황수정, 이유성, 김시윤, 문지희

강사: 이상훈

목차

■
01 / 블콩카
개요

■
02 / 블콩카
팀원소개

■
03 / 블콩카
진행상황 및
문제점

■
04 / 블콩카
진행계획

01

개요



차량에 부착된 센서, 카메라를 활용해 주변 환경을
인식, 수집된 정보로 모터의 속도와 방향을 제어해
자율주행 시스템을 구현할 계획.

02 팀원 소개

안상재

- ESP8266 펌웨어 업그레이드
- SPI 통신 학습 및 자료정리

정유경

- 모터 관련 자료 조사 및 부품 선정
- 영상처리 학습

황수정

- I2C 통신 학습 및 자료정리

이유성

- PID 제어 학습
- UART 통신 학습 및 자료 정리

김시윤

- ESP8266 소스코드 구현

문지희

- DMA 학습 및 자료정리

03

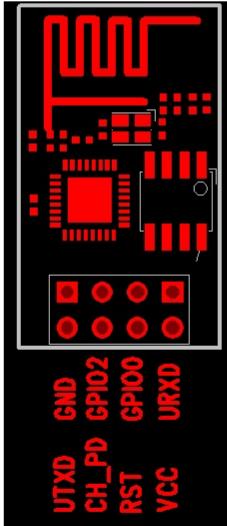
진행상황 및 문제점

안상재

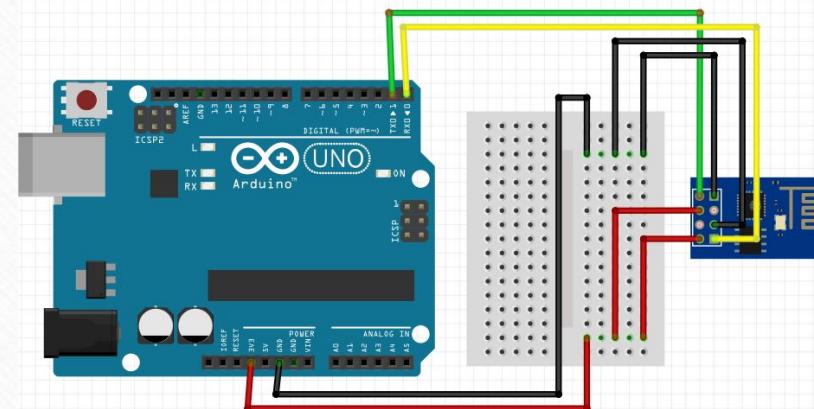
- ESP8266 펌웨어 업그레이드
- SPI 통신 학습 및 자료정리

WIFI 모듈 펌웨어 업그레이드

1. esp8266 모듈 핀 배치도 및 회로 결선



아두이노 UNO ----- esp8266
Tx ----- Tx
Rx ----- Rx
GND ----- GND
3.3V ----- CH_PD, VCC



2. 툴 설치 및 펌웨어 다운로드

- 연결을 다 마쳤으면 아래의 링크로 가서 펌웨어를 업데이트 할 수 있는 프로그램을 다운 받는다.

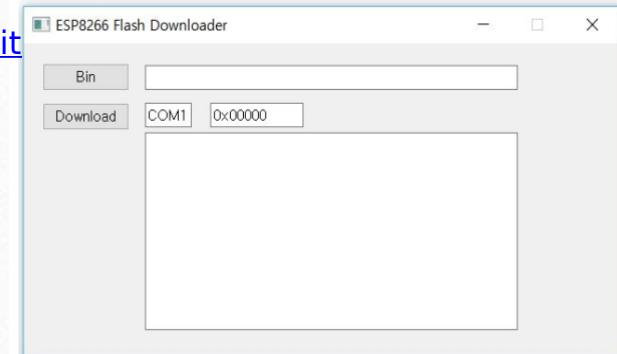
<https://docs.google.com/file/d/0B3dUKfqzZnlwVGc1YnFyUjgxelE/edit>

- 그 후에 펌웨어를 아래의 링크에서 다운받는다.

<https://docs.google.com/file/d/0B3dUKfqzZnlwdUJUc2hkZDUyVjA/edit>

- 첫 번째 링크를 통해서 받은 파일이 있는 위치에 두 번째 링크에서 받은 펌웨어 bin 파일을 넣는다. 그런 후에 esp8266_flasher.exe 파일을 실행시키면 다음과 같은 창이 뜨는 것을 확인할 수 있다.

이름	수정한 날짜	유형
cloud update	2014-09-26 오후 5...	텍스트 문서
ESP_8266_BIN0.92.bin	2014-09-23 오후 3...	BIN 파일
ESP8266_AT_V00180902_02_baudrate w...	2017-01-12 오후 5...	압축(ZIP) 파일
esp8266_flasher	2014-09-23 오후 1...	응용 프로그램
ESP8266_flasher_V00170901_00_Cloud U...	2017-01-12 오후 5...	압축(ZIP) 파일
v0.9.2.2 AT Firmware.bin	2014-09-30 오후 3...	BIN 파일

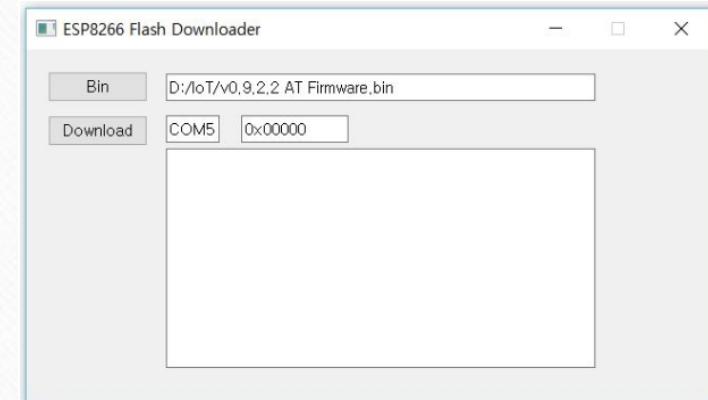


3. 펌웨어 다운로드

- 주의할 점은 파일이 있는 링크안에 한국말로 된 주소가 포함되면 안된다는 것이다.

한국말로 포함된 주소가 있으면 위의 Bin 박스를 클릭해도 아무것도 들어가지 않는다. 그러므로 한국말로 된 링크가 없는 곳으로 디렉토리를 옮긴 후에 진행하면 된다.

포트 넘버를 장치관리자에서 확인한 뒤에 그 포트 넘버를 수정해주고, Bin 박스를 눌러서 펌웨어를 가져온다.



- 다운로드 버튼을 눌러준다. 99%에서 leaving이라는 문자가 나왔다면 성공적으로 펌웨어를 업데이트한 것이므로 종료해도 된다.

그러나 fail to connect라는 문자가 떴다면

1. 전원을 연결한 선을 뺏다가 다시 꽂는다.

(connectin... 중에 뺏다가 다시 꽂음)

2. TX와 RX 선이 제대로 연결되었는지 확인한다.

3. ESP8266에 빨간 불이 들어왔는지 확인한다.

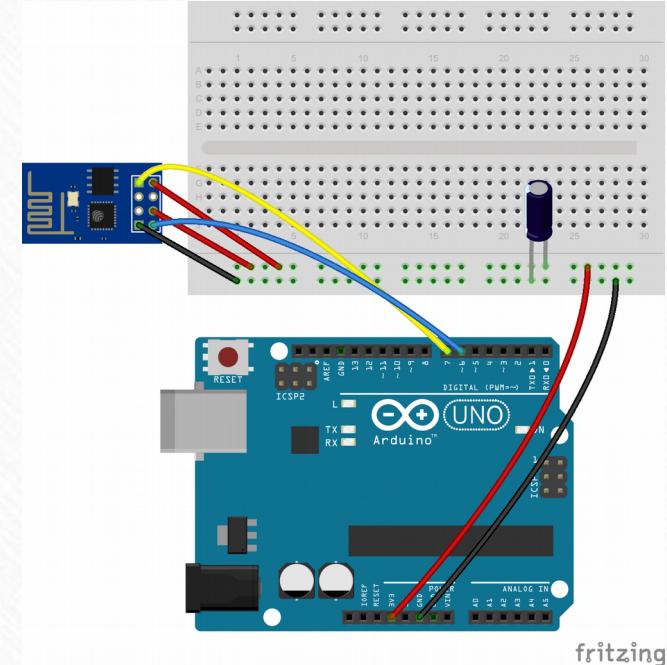
4. 아두이노 USB 포트를 다시 뺏다가 연결한다.

5. GPIO 0에 GND가 제대로 연결되었는지 확인한다.

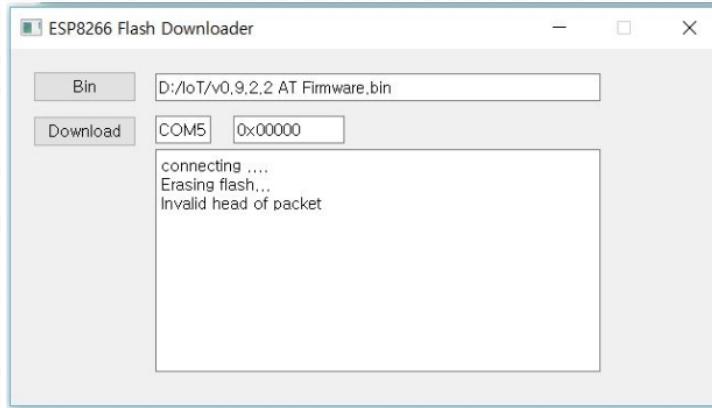
6. 전원을 5.0V로 바꾸어 껴준다. : ESP8266 모듈은 순간적으로 300mA 이상 전류를 잡아먹기 때문에 아두이노의 3.3V 핀을 사용하면 안될 경우가 있다. 다운받는 도중에 계속 Failed to write to target Flash 에러 메시지가 뜬다면 한번 시도해보는 게 좋음.

7. 1~6번을 시도했음에도 다운로드가 잘 되지 않을 경우, 아래 회로와 같이 100UF의 완충용 콘덴서를 VCC-GND에 병렬로 연결해서 재시도 한다.

esp8266 모듈의 동작은 고주파 신호로 동작해서 전원이 불안정할 가능성이 있다. 그러므로 완충용 콘덴서를 달아주면 전원을 안정화하는 데 도움을 줄 수 있다.



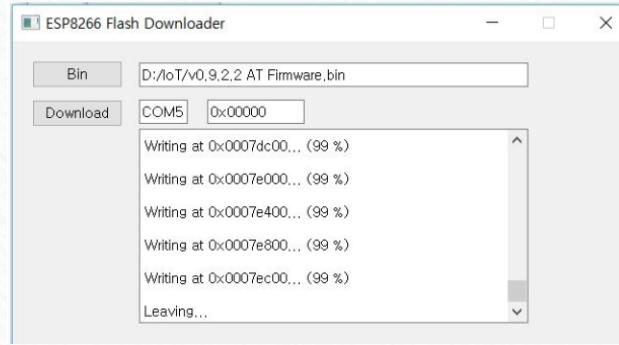
(UNO 아두이노 모드의 ~6번 핀은 TxD, 7번핀은 RxD 핀)



만약 이 오류말고 invalid head of packet이라는 오류가 발생한다면 아두이노 초기화가 필요한 상태임을 의미한다. 그렇기 때문에 그런 경우에는 우측과 같은 코드를 아두이노에 업로드함으로써 초기화해주면 문제가 해결된다.



다음과 같은 화면이 뜬다면 펌웨어 업데이트가 완료된 것이다.



만약 Leaving... 다음에 오류 메시지가 뜨더라도 문제는 없으니 신경쓰지 않아도 된다.

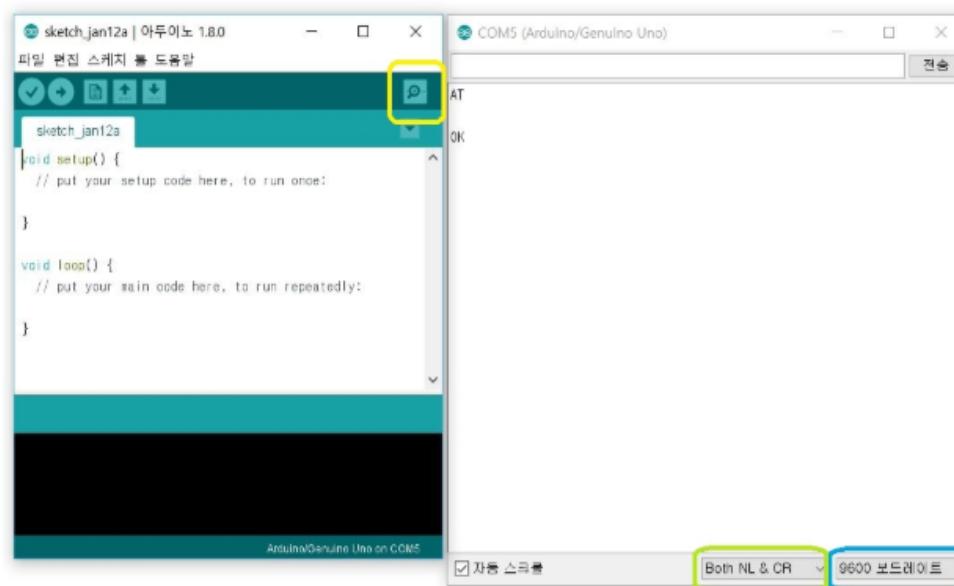
4. 통신 체크

이제 펌웨어 업데이트가 완료되었으니 제대로 펌웨어가 업데이트 되었는지 기본적인 통신 체크를 통해서 확인해본다.

먼저 아두이노 전원을 제거한 후에, 펌웨어 업데이트를 위해서 연결해 놓았던 GPIO 0 핀을 빼준다.

이 핀이 연결되어 있는 상태에서는 통신이 되지 않는다.

그런 후에 다시 아두이노를 PC와 연결해 준다.



그런 후에 아두이노 IDE를 통해서 시리얼 모니터를 켜시고 라인 엔딩을 Both NL & CR로 설정해 준다. 업데이트를 9600 보레이트로 했으니 보레이트도 9600으로 둔다.

시리얼 모니터에 AT를 입력했을 때 OK라는 문자가 전송된다면 정상적으로 동작함을 의미함으로 제대로 업데이트 되었음을 확인할 수 있다.

만약에 ERROR 메시지가 전송되었다면 다시 한 번 AT를 입력해 본다. 그러면 OK를 전송해주는 것을 확인할 수 있다.

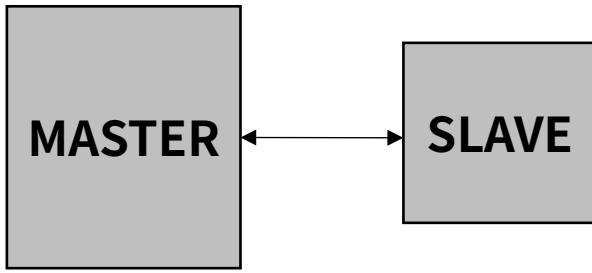
SPI 통신 (*Serial Peripheral Interface*)

1. SPI 특징

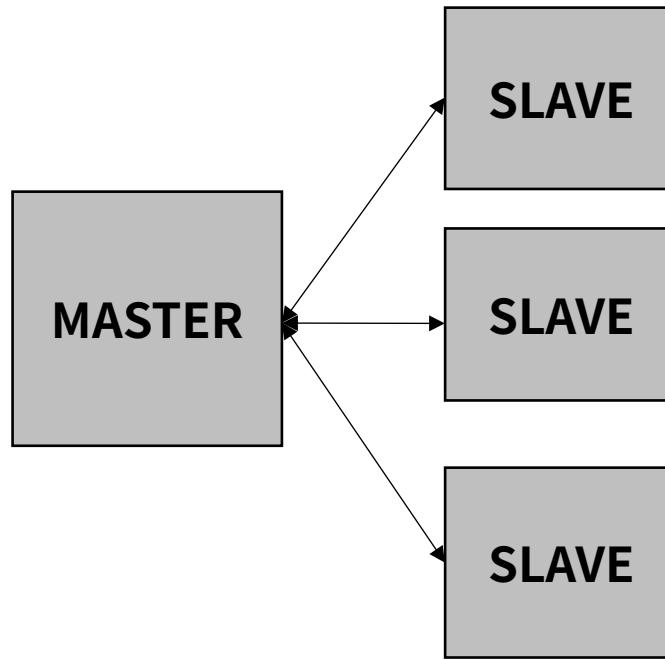
- SPI는 Motorola/Freescale사에서 처음 개발하였음.
- 임베디드 시스템 내부의 모듈, IC간 연결 및 통신 시 주로 사용됨. (온도센서, EEPROM 메모리)
- 단거리 전이중 전송방식 (전화 통화)

2. SPI 버스

- 1:N 통신 (1개의 마스터가 다수의 슬레이브를 제어함)
- 클럭 라인을 별도로 사용하는 동기식(synchronous) 직렬 데이터 버스 방식
 - ⇒ 동기식 통신이란 클럭선(CLK) 을 사용해서 클럭의 엣지마다 데이터를 1bit 전송하는 방식을 말함.
 - ⇒ 비동기식 통신의 대표적인 예로는 UART 통신이 있음.
- 물리적으로 분리된 송신 및 수신 데이터 선로를 사용하므로 전이중 통신이 가능한 장점이 있음.
- 주소 영역을 사용하지 않으므로 여러 개의 슬레이브 장치에 대한 어드레싱을 위해 각 장치별 Chip Select(CS) 신호선이 추가되어야 하는 문제점이 있음.
- SPI 버스 상의 마스터 노드는 클럭을 제공하고 슬레이브 장치를 선택하는 기능을 가진다.
나머지 노드는 슬레이브 노드로써 개별적인 Slave Select(chip select) 선로에 의해 선택됨.



a) 1:1 통신

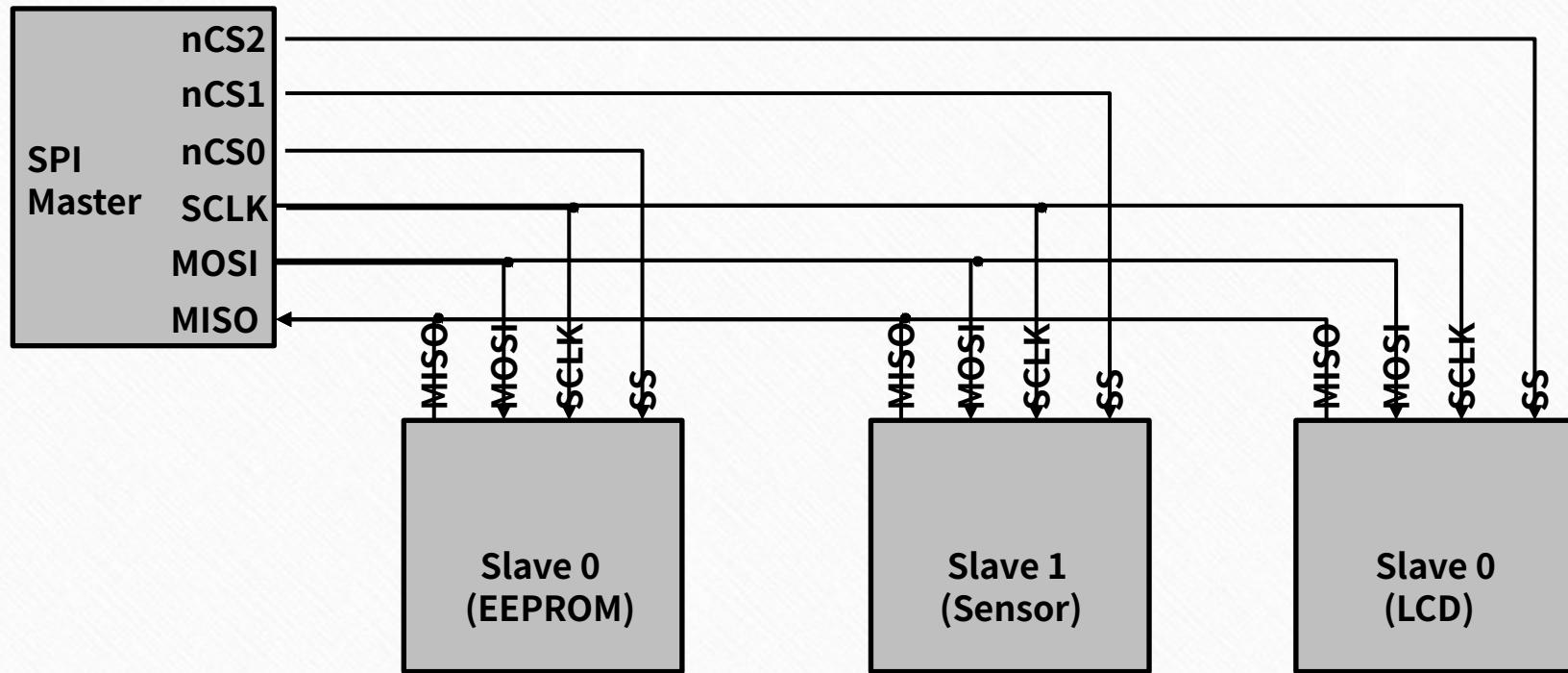


b) 1:N 통신

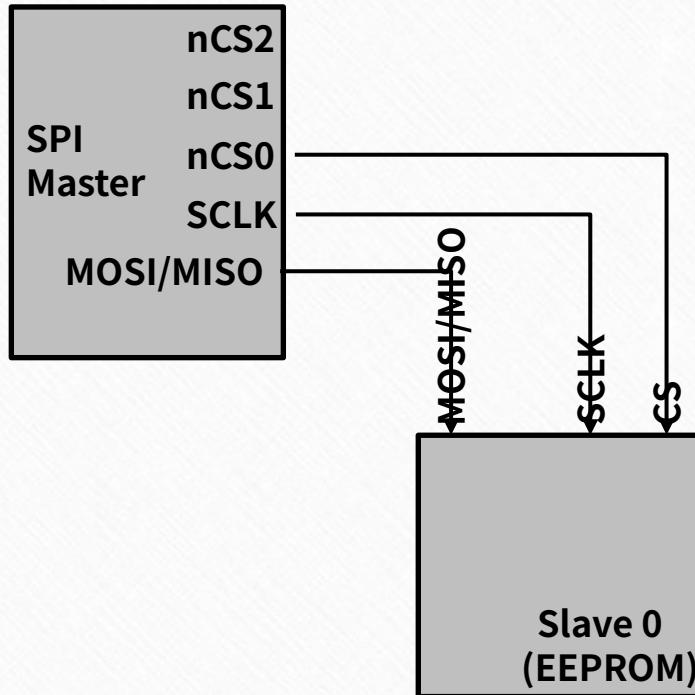
- SCLK(Serial Clock) : 마스터가 생성하는 클럭 신호이다.
- MOSI(Master Output, Slave Input) : 마스터가 출력하는 직렬 데이터 신호이다. MSB부터 송신된다.
- MISO(Master Input, Slave Output) : 마스터가 수신하는 직렬 데이터 신호이다.
- CS(Chip Select) : 마스터에 의한 슬레이브 선택용 제어 신호이다. SS은 보통 active low 방식이지만 그 반대의 경우도 있다. 선택되지 않은 장치의 MISO 신호값은 하이 임피던스값을 유지한다.

3. SPI 버스의 종류

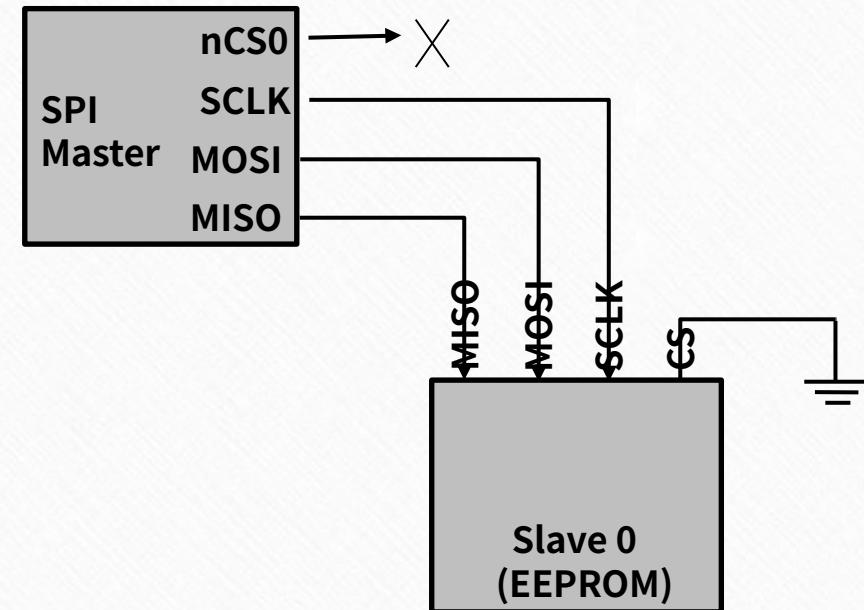
(a) 4-wire SPI



(b) 3-wire SPI/Half-duplex (MicroWire)

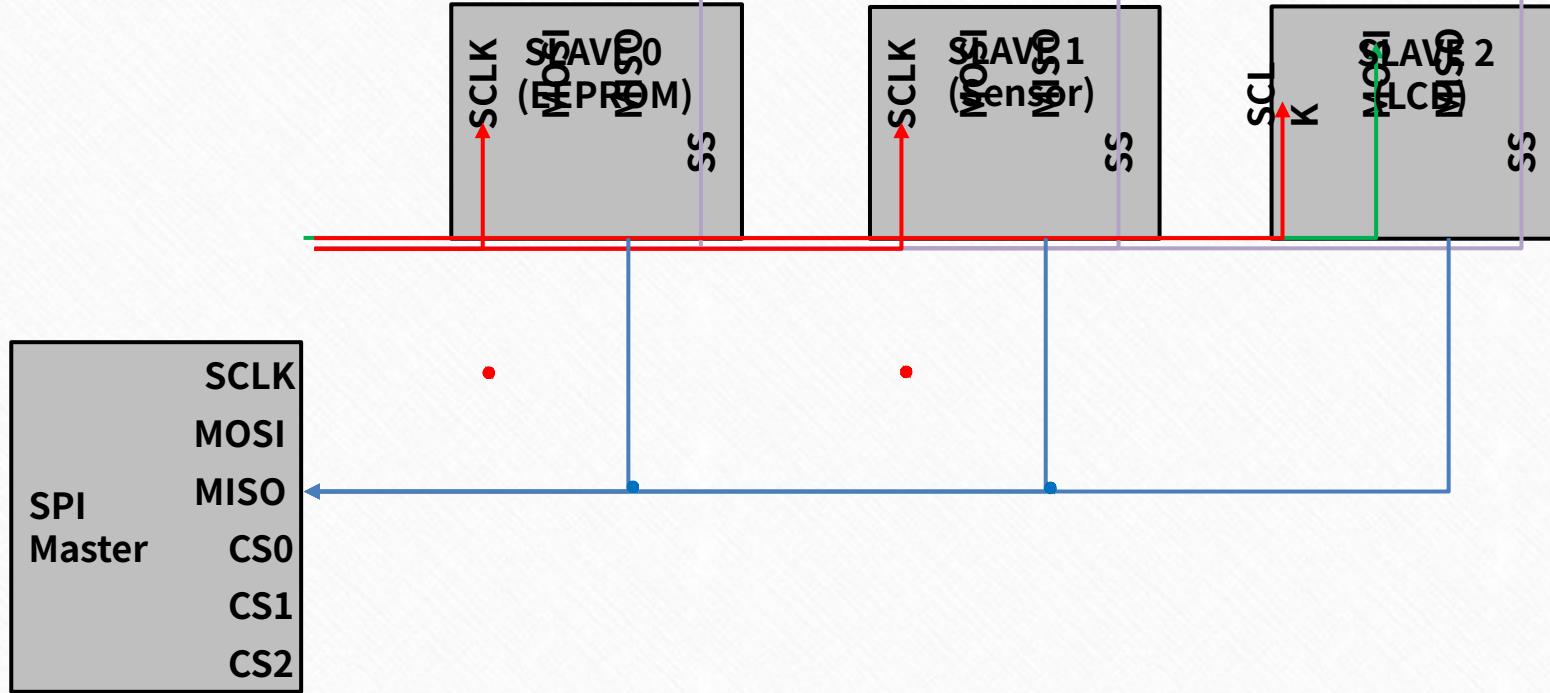


(c) 3-wire SPI (1:1 연결용)

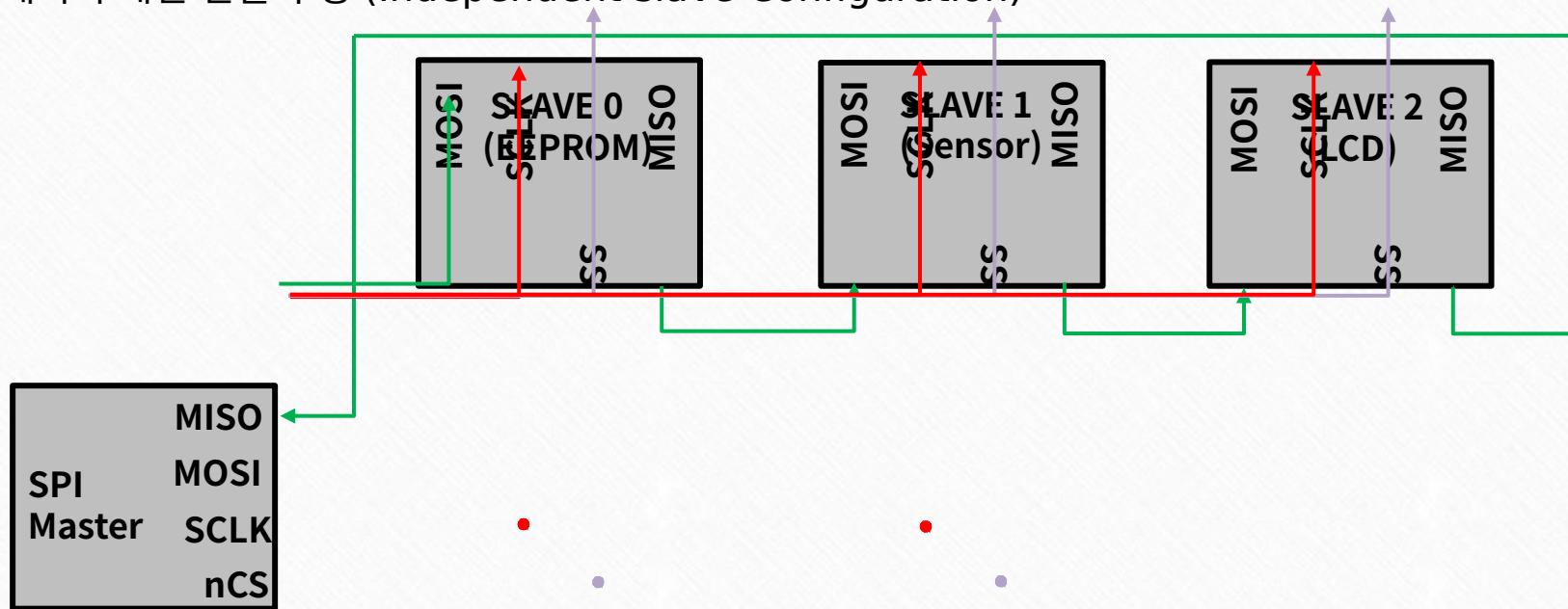


4. SPI 버스의 구성 (여러 개의 SPI 장치 (슬레이브)를 연결 시)

1) 개별적인 슬레이브 연결 구성 (Independent slave Configuration)



2) 데이지 체인 연결 구성 (Independent slave Configuration)



=> 1개의 Chip Select 선로만 사용하여도 다수의 슬레이브에게 데이터를 전달할 수 있음.

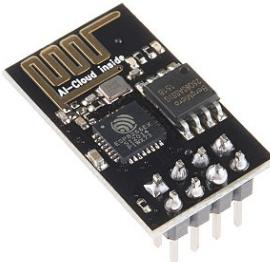
03

진행상황 및 문제점

김시윤 : ESP8266 소스코드 구현



MODULES



Alternately, serving as a Wi-Fi adapter, wireless internet access can be added to any microcontroller-based design with simple connectivity through UART interface or the CPU AHB bridge interface.

Wi-Fi 어댑터로 제공되는 UART 인터페이스 또는 CPU AHB브리지 인터페이스를 통해 간단한 연결로 모든 마이크로 컨트롤러 기반 설계에 무선 인터넷 액세스를 추가 할 수 있다.

ESP8266 ESP-01

Table 2 Pin Descriptions

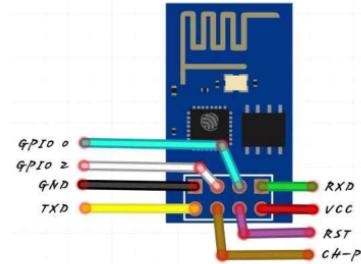
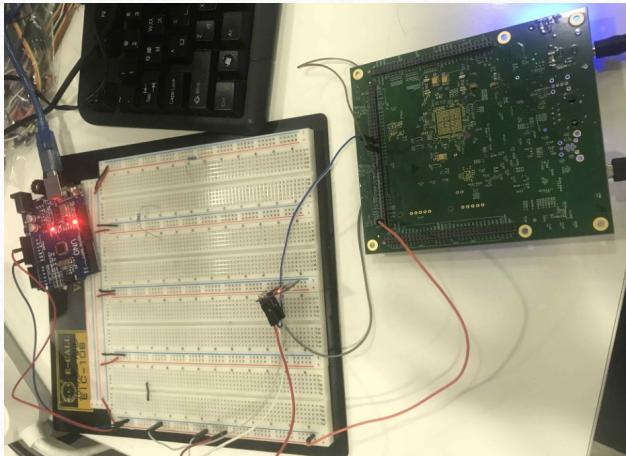
NO.	Pin Name	Function
1	GND	GND
2	GPIO2	GPIO,Internal Pull-up
3	GPIO0	GPIO,Internal Pull-up
4	RXD	UART0,data received pin RXD
5	VCC	3.3V power supply (VDD)
6	RST	1) External reset pin, active low 2) Can loft or external MCU
7	CH_PD	Chip enable pin. Active high
8	TXD	UART0,data send pin RXD

Table 3 Pin Mode

Mode	GPIO15	GPIO0	GPIO2
UART	Low	Low	High
Flash Boot	Low	High	High

우리가 사용하는 모듈은 ESP8266 ESP-01이다.
데이터시트를 통해 각 핀의 동작특성과 핀 설정에 따라
모드를 확인한 후 회로 구성을 실시한다.

회로 구성



ESP8266

RXD

TXD

GND

VCC

GPIO 0

GPIO 2

CH-PD

RST

MCU

SCI3 TX(N2)

SCI3 RX(W3)

GND(아두이노 & MCU)

3.3V(아두이노 or MCU GIO 출력 *아두이노 추천)

GND

X

3.3V

X

AT Command

AT+RST – 리셋 하는 명령어 (초기화) [응답 :OK]

AT – 연결테스트 [응답 :OK]

AT+GMR – ESP8266 의 버전 정보 확인 [응답 :버전정보 OK]

AT+CIPMODE=0 – 전송방식 설정 0 -> 노말 [응답 :OK]

AT+CWJAP="josephahn", "00000000" – AP에 연결 “SSID”, “PW”

AT+CIPCLOSE – 혹시 서버가 열려있으면 닫아라

AT+CIPMUX=1 – 단일연결 다중연결 설정 ,1은 다중연결

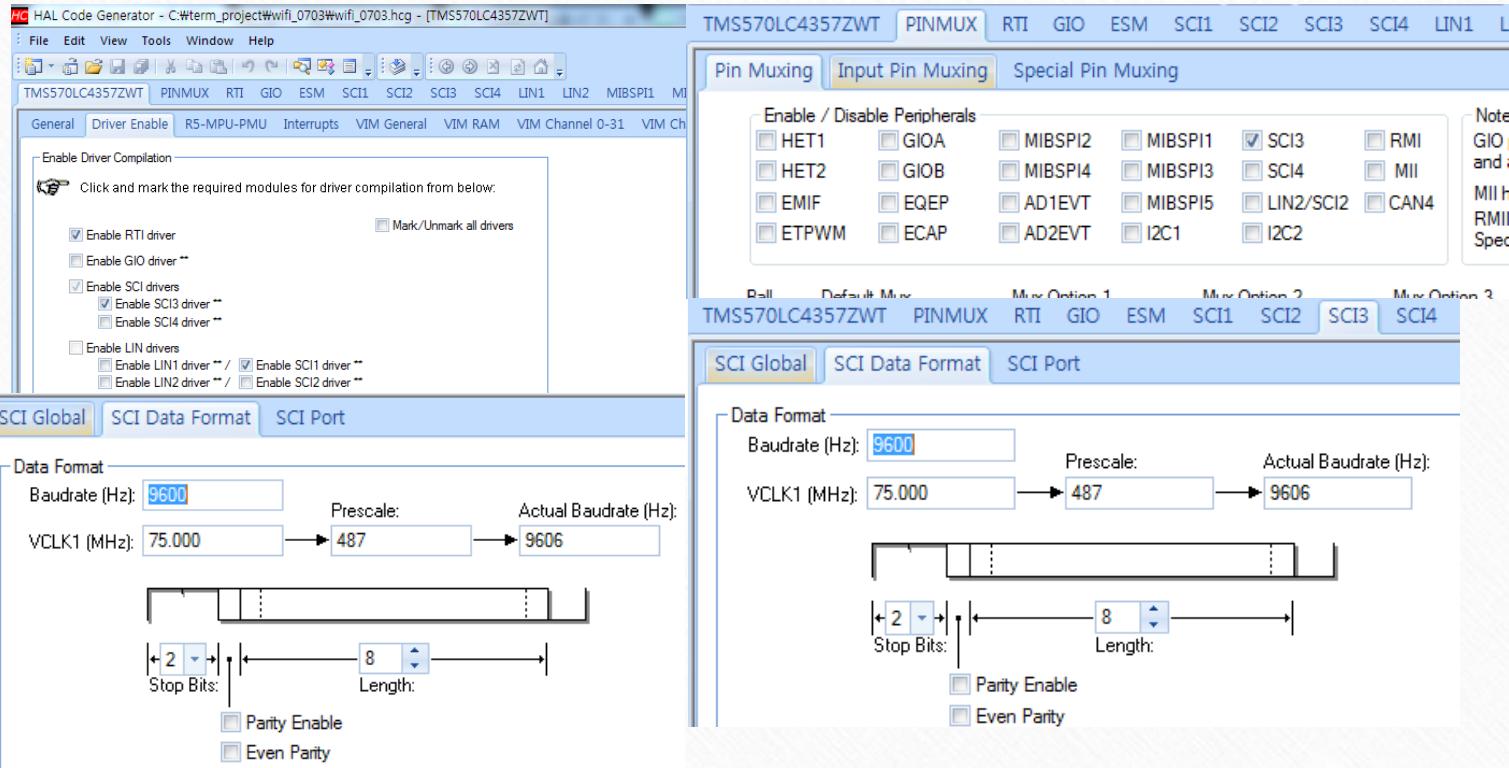
AT+CIPSERVER=1,777 – TCP/IP 서버 오픈, 1은 TCP 0는 UDP, 포트 777

AT+CIFSR – ESP8266 Local IP 와 AP에서 할당한 IP 출력

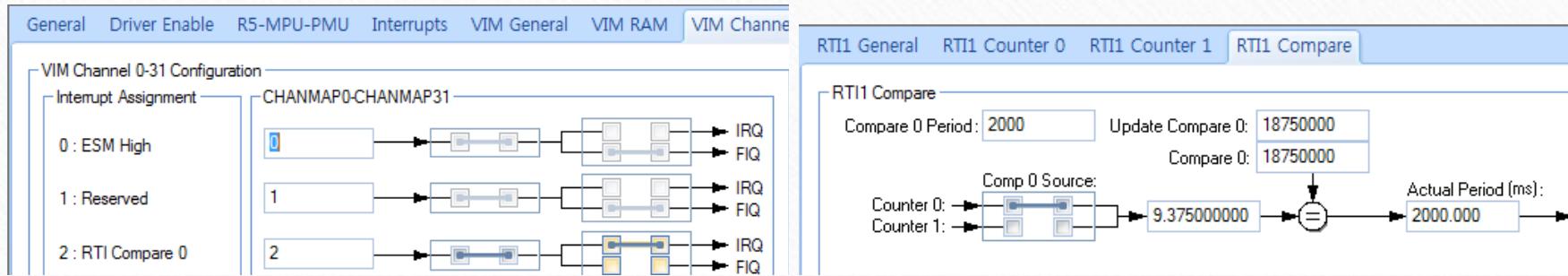
AT+CIPSTO=1000 – 서버 타임아웃 1000초

03

진행상황 문제점



ESP8266 모듈과 TMS570LC4357 보드와의 통신을 위한 SCI3 활성화 Baud rate 9600 설정.
모니터와 보드의 통신을 위한 SCI1 활성화 Baud rate 9600 설정 (ESP8266 펌웨어의 Baud rate =9600)



실험 결과 ESP8266 모듈의 읽는 속도가 보드의 보내는 속도보다 느려 인식을 못하는 현상이 발생, 이를 해결하기 위해 `delay` 함수를 작성하여 제어하였지만, 정확한 시간을 통제하지 못해, rti 인터럽트를 활성화시켜 2초마다 동작시키도록 설정하였다.

```
#include "HL_reg_sci.h"
#include "HL_rti.h"
#include "HL_sci.h"
#include "HL_sys_core.h"
#include "HL_gio.h"
#include <stdlib.h>
#include <stdio.h>
#include <string.h>

uint8 flag = 1; //진행절차 상태 변수

void rtiNotification(rtiBASE_t *rtiREG, uint32 notification)
{
    switch(flag){
        case 1:
            sciSend(sciREG3, 4, "AT\r\n");
            flag++;
            break;
        case 2:
            sciSend(sciREG3,9 , "AT+GMR\r\n");
            flag++;
            break;
        case 3:
            sciSend(sciREG3,14,"AT+CIPMODE=0\r\n");
            flag++;
            break;
        case 4:
            sciSend(sciREG3,
34,"AT+CWJAP=\"josephahn\",\"00000000\"\r\n");
            flag++;
            break;
        case 5:
            sciSend(sciREG3, 13, "AT+CIPCLOSE\r\n");
            flag++;
            break;
        case 6:
            sciSend(sciREG3,13,"AT+CIPMUX=1\r\n");
            flag++;
            break;
        case 7:
            sciSend(sciREG3,25 , "AT+CIPSERVER=1,777\r\n");
            flag++;
            break;
        case 8:
            sciSend(sciREG3,10 , "AT+CIFSR\r\n");
            flag++;
            break;
        case 9:
            sciSend(sciREG3, 16, "AT+CIPSTO=1000\r\n");
            flag++;
            break;
    }
}
```

```
default:  
    rtiDisableNotification(rtiREG1, 1);  
    break;  
}  
}  
  
void main(void)  
{  
    scilInit();  
    rtilInit();  
    rtiEnableNotification(rtiREG1, 1);  
    rtiStartCounter(rtiREG1,  
rtiCOUNTER_BLOCK0);  
    _enable_IRQ_interrupt_();  
    while(1)  
    {  
        sciSendByte(sciREG1,  
        sciReceiveByte(sciREG3));  
    }  
}
```

03

진행상황 문제점

앞에 코드를 입력하여 MCU에 업로드하게 되면 다음과 같은 결과물이 나온다.

오른쪽의 사진은 TCP Client라는 어플과 ESP8266 과의 통신이며, ESP8266에 핸드폰에서 입력한 데이터가 UART를 통해 출력되는 결과를 보여준다.

```
SR  
192.168.4.1  
172.20.10.11  
OK  
00  
  
OK  
ND  
  
ERROR  
  
+IPD,0,2:9  
OK  
  
+IPD,0,2:9  
OK  
  
+IPD,0,21:ffffggsshdhdjdjfjfjf  
OK  
  
+IPD,0,9:i love u  
OK  
Unlink  
[REDACTED]
```



```
void main(void)
{
    gioInit();
    sciInit();
    rtiInit();
    rtiEnableNotification(rtiREG1, 1);
    rtiStartCounter(rtiREG1, rtiCOUNTER_BLOCK0);
    _enable_IRQ_interrupt_();

    while(1)
    {
        if(setting_flag == 0)
            sciSendByte(sciREG1, sciReceiveByte(sciREG3));
        else if(setting_flag == 1)
        {
            if(sciReceiveByte(sciREG3) == 'a')
                gioSetBit(gioPORTB, 0, 1);

            sciREG3->RD = 0;

            if(sciReceiveByte(sciREG3) == 'b')
                gioSetBit(gioPORTB, 0, 0);
        }
    }
}
```

핸드폰에서 받은 데이터로 MCU의 GIO를 통제해 LED를 제어해보았다.

Setting flag = ESP8266의 서버가 열렸는지 확인하는 변수

03

진행상황 문제점



문제점

1. ESP8266 과 MCU의 속도가 달라 ESP8266이 데이터를 인식하지 못하는 현상!

Sol) delay를 사용하여 ESP8266이 데이터를 받을 수 있게 구현 -> 정확한 시간을 할당하기 위해 rti 사용

2. ESP8266 은 자료가 Arduino에 치중 되어있음 자료 참고의 어려움.

Sol) 데이터시트와 AT-Command 매뉴얼 참고, 외국 사이트 참고로 해결.

3. ESP8266 과 핸드폰 통신으로 GIO 제어 시 데이터를 두번 보내야 인식되는 현상.

Sol) SCI 의 sciREG->RD (sci Receive 함수의 데이터 저장 버퍼) 를 0으로 초기화 하여 해결.

- ESP8266 을 AT-Command 를 이용하여 서버 오픈 할 때, 중간에 실패하면 다시 처음부터 진행해야 한다는 단점이 있음,, 하지만 연결 이후에 동작은 문제없이 잘 진행됨을 확인함.
- 만들어져 있는 어플로 통신함, 현재 어플 미구현.

03

진행상황 및 문제점

정유경 : 모터선정 및 opencv 복습

1. 모터선정 - 크기 측정



Motor Specs

전체 무게 : 35 ~ 40kg

최고 속도 : 6km/h (1.67m/sec)

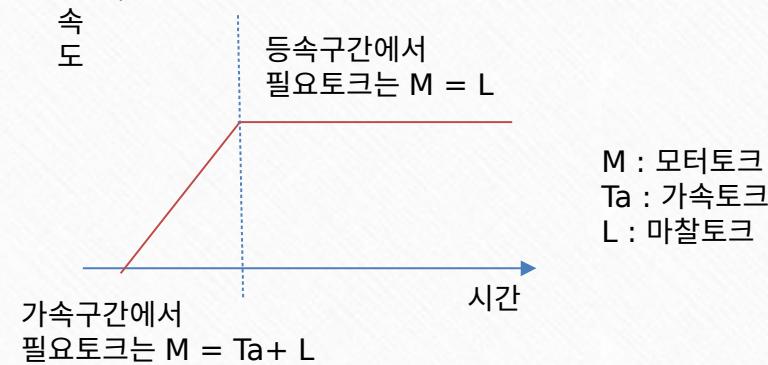
가속시에 필요한 최소 토크 : 가속+ 등속토크

등속운동시에 필요한 최소토크 : 등속토크

(정지, 운동)마찰 계수 : 일반적으로 0. 1

최저 속력 : 1km/h (28cm/sec)

가속시간: 1초



$$\text{가속토크} : T_a = 2260.125/980 \times 2 \times \pi \times f_x(1/t) = 43.44975 \text{ [kgfcm]}$$

$$\text{등속토크} : L = \text{운동마찰계수} \times 40 \times 21/4 = 21 \text{ [kgfcm]}$$

$$\text{부하관성모멘트} : WD^{2/8} = 40 \times 21^{2/8} = 2260.125 \text{ kgcm}^2$$

Motor Specs

가속시간 : 1 [sec]

등속회전속도 : 3회전/sec

바퀴 반지름 : 21cm

Shaft 직경 : 5mm

Pinior gear : 10T (Module Ratio : 1)

모터와 바퀴사이의 기어비: 125.7: 1

($5.8 \times 5.6 \times 43/9 = 125.7$)



따라서 모터 선정시 필요 최소 토크는: $64.44975 / \text{기어비} = 0.5127 \text{ [kgfcm]}$

Motor Specs

BLDC Motor:

Max Power: 2600W

속도상수 Kv: 2150 (전압 1V 당 회전 수 \otimes rpm 분당 회전수)

Max Current: 120A

Max Voltage: 23.3V

Outside Diameter: 40mm

Length: 74mm

Shaft Length: 18mm (5mm)

Weight: 380g

BLDC는 DC와 달리 세 개의 극이 주파수 컨트롤에 의해 위상이 변하면서 회전하므로 반드시 ESC를 연결해서 사용
브러시와 정류자의 접촉x, 잡음x, 장기간 사용에 적합
ESC(electronic speed controller) : 직류를 브러시모터가 쓸 수 있게끔 3극으로 바꾸고 모터의 속도를 제어, 변속기

ex. 7.4V의 2셀 LiPo 배터리를 연결한다면
모터의 분당 회전수는 $2,150 \times 7.4V = 15910$ 회

모터의 성능을 결정하는 2가지 요인: KV, Torque (서로 반비례)

모터의 최대허용전류 < 변속기의 최대 전류 < 배터리 방전 Rate

ESC Specs

모터에 충분한 파워를 공급해줄 ESC를 선정하려면

모터의 overall Wattage와 Burst amps를 고려하여야 함

- 모터의 출력[Watt] = 사용할 배터리의 전압 x 최대허용 전류 x 효율계수
- 허용전류 (일반적으로 연속허용전류) : 연속적으로 돌려도 모터에 무리가 없는 전력
- 최대허용전류 : 유지가능시간(30초), 허용전류를 초과할 경우 모터나 ESC에 손상

ex. 모터의 최대허용전류가 10A라면 30~50%의 여유를 두고 ESC의 출력은 13~15A인 것을 선택

ex. 2200mAh / 20C 배터리의 최대 방전 전류는 44A이므로
44A의 50~60% 즉, 20~25A의 변속기(ESC)를 선택

주의! 4s Max ESC에 6S LiPo를 사용하면 안된다.

LiPo Battery Specs

장점

- 폭발하지 않고 부풀어오르므로 안전,
- 내부가 고체전해질로 채워져 있어서 배터리 형상이 자유로움.

고려사항

- 배터리 크기, 무게
- 적절한 용량(전압, 전류, Power) [mAh or Ah]
 - ex. 2000mAh : 2000mA의 전류세기로
균일하게(Constant) 1시간동안 방전가능
 - but, 과방전시 Lipo배터리 재충전 불가 주의! (ESC이용 3.2V정도에서 방전차단)
모든 리튬폴리머 전지의 정격전압은 3.7V (완충시 4.2V)
 - ex. 4S= $4 \times 3.7V = 14.8V$ (정격전압) / 완전 충전시 16.8V 이 이상으로 충전금지
 - 2P : 병렬구성, 용량 2배

LiPo Battery Specs

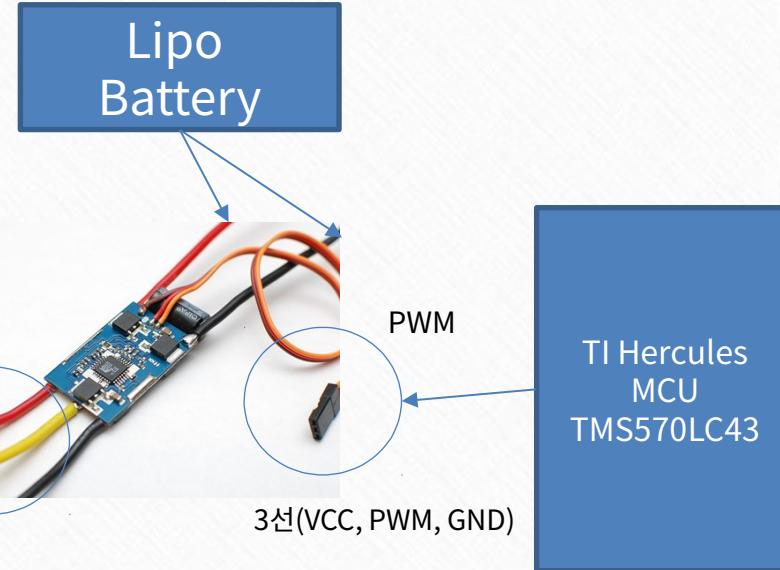
- 방전율
ex. 50A를 필요로 하는 모터에 2000mAh, 20C 배터리(40A)는 부족
- Burst Rate
배터리가 짧은 순간(10~20sec)에 최대한 뽑아낼 수 있는 에너지의 양

Motor Configuration

Pinion Gear:
10T 5mm

BLDC Motor:
Max Power: **2600W**
KV: **2150**
Max Current: **120A**
Max Voltage: **23.3V**
Outside
Diameter: **40mm**
Length: **74mm**
Shaft Length: **18mm**
Weight: **380g**

BLDC
Motor



변속기 ESC는 MCU에서 제공되는 PWM신호를 가지고 BLDC를 제어
변속기에 붙은 배터리의 전원이 변속기를 지나 MCU에 공급

문제점

1. BLDC 모터 선정 시 고려하지 않은 물리량 재계산 필요
: 계산을 통하여 모터스펙을 도출해내려면, 더 많은 자료조사가 필요하나 빠른 모터 구매가 시급함
2. BEC 출력 3A 충분한지 검증 필요
3. 모터 고정 방법에 대한 고찰 필요

03

진행상황 및 문제점

문지희 : DMA 학습 및 자료정리

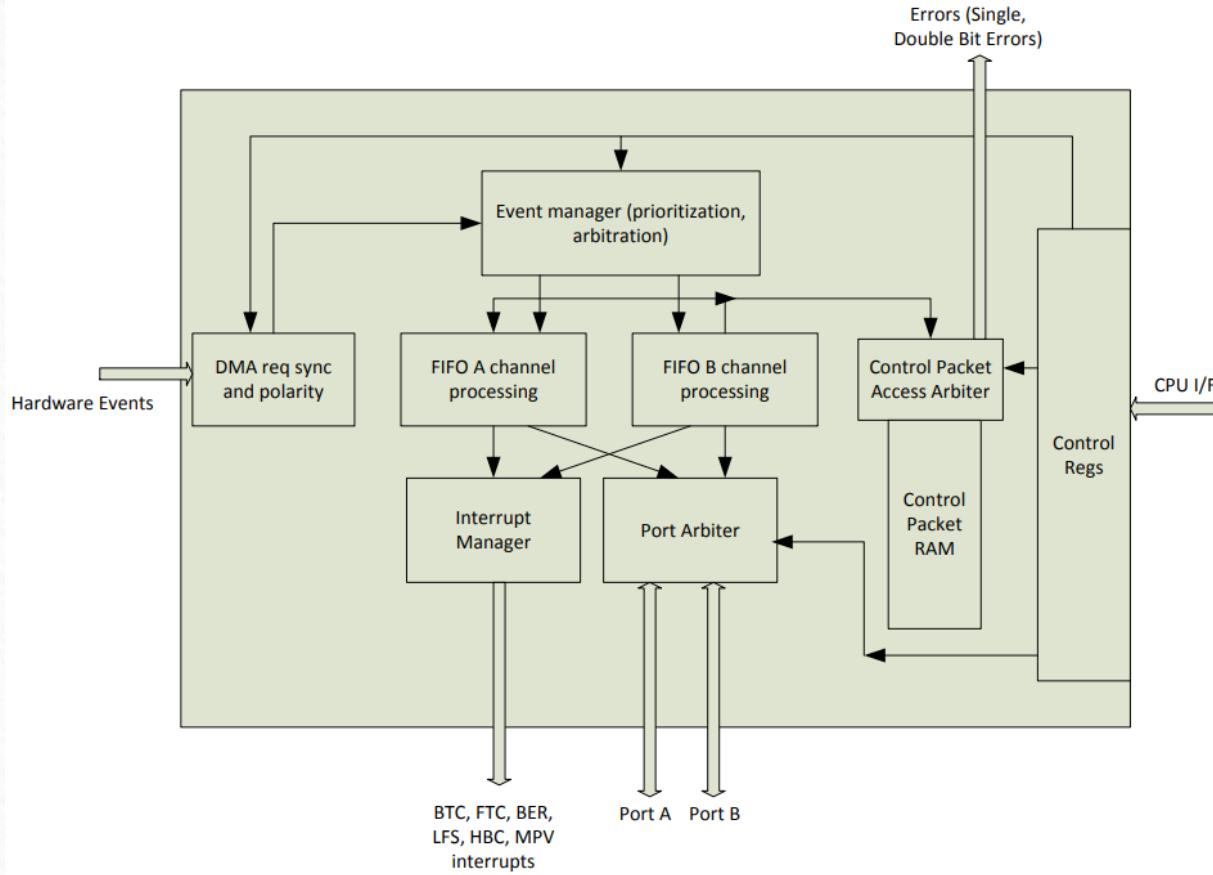
DMA

Direct Memory Access

주변장치들이 메모리에 직접 접근하여 읽거나 쓸 수 있도록 하는 기능으로,
장치들 사이의 데이터가 CPU를 거치지 않는다.

장치 컨트롤러가 데이터의 한 블록을 이동시키는데 CPU의 개입이
필요 없어 많은 양의 데이터를 이동시킬 때 부담이 없다.
CPU에서는 데이터 이동이 완료되면 한번의 인터럽트가 발생하게 되어
효율성이 높아진다.

Figure 20-1. DMA Block Diagram



Port Arbiter

Table 20-1. DMA Ports to System Resources Mapping

DMA Ports	System Resources
Port A	<ul style="list-style-type: none">L2 FlashL2 SRAMEMIF
Port B	<ul style="list-style-type: none">All peripherals, that is, MibSPI registers, DCAN registersAll peripheral memories, that is, MibSPI RAM, DCAN RAM

: L2 Flash • L2 SRAM • EMIF → 주변 장치 레지스터 • 주변 장치 메모리
PARx 레지스터의 각 채널에 0 x1(Port A read, Port B write)을 쓴다.

: 주변 장치 레지스터 • 주변 장치 메모리 → L2 Flash • L2 SRAM • EMIF
PARx 레지스터의 각 채널에 0 x0(Port A write, Port B read)을 쓴다.

: L2 Flash → L2 SRAM
PARx 레지스터의 각 채널에 0 x2(Port A)을 쓴다.

: 주변 장치에서 → 다른 주변 장치
PARx 레지스터의 각 채널에 0 x3(Port B)을 쓴다.

Interrupt Manager

Figure 20-2. Example of a DMA Transfer Using Frame Trigger Source

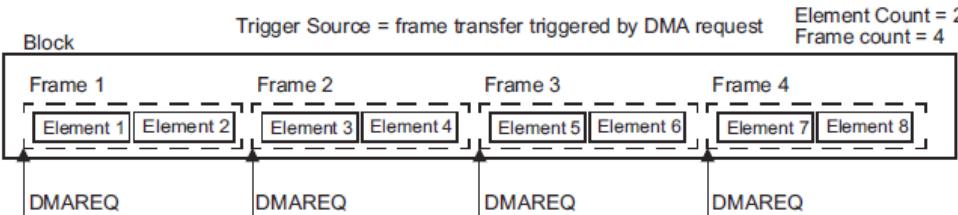
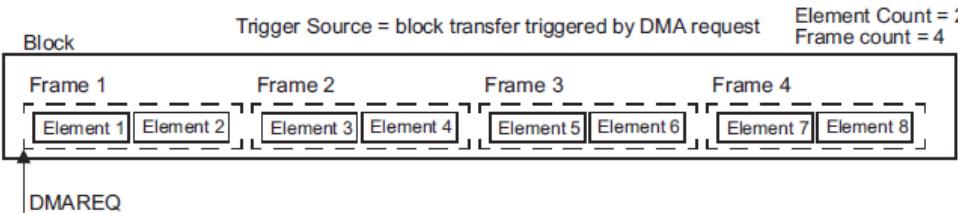


Figure 20-3. Example of a DMA Transfer Using Block Trigger Source



FTC : Frame transfer complete

LFS : Last frame transfer started Interrupt

HBC : First half of block complete Interrupt

BTC : Block transfer complete Interrupt

Control Packet Access Arbiter & Control Packet RAM

Figure 20-4. DMA Request Mapping and Control Packet Organization

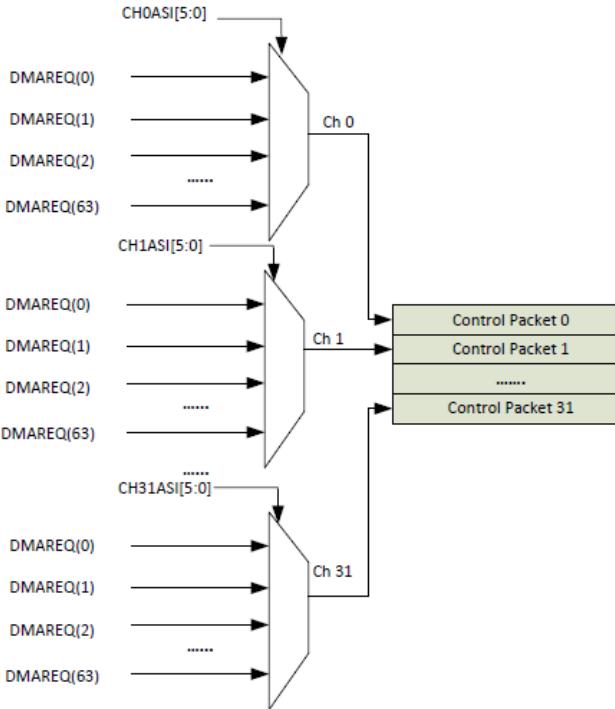


Figure 20-5. Control Packet Organization and Memory Map

The memory map for control packets is organized into four columns:

- Column 1:** Initial Source Address, Channel Configuration, Initial Source Address, Channel Configuration, ...
- Column 2:** Initial Destination Address, Element Offset Value, Initial Destination Address, Element Offset Value, ...
- Column 3:** Initial Transfer Count, Frame Offset Value, Initial Transfer Count, Frame Offset Value, ...
- Column 4:** Reserved, Reserved, Reserved, ...

Specific memory addresses are mapped as follows:

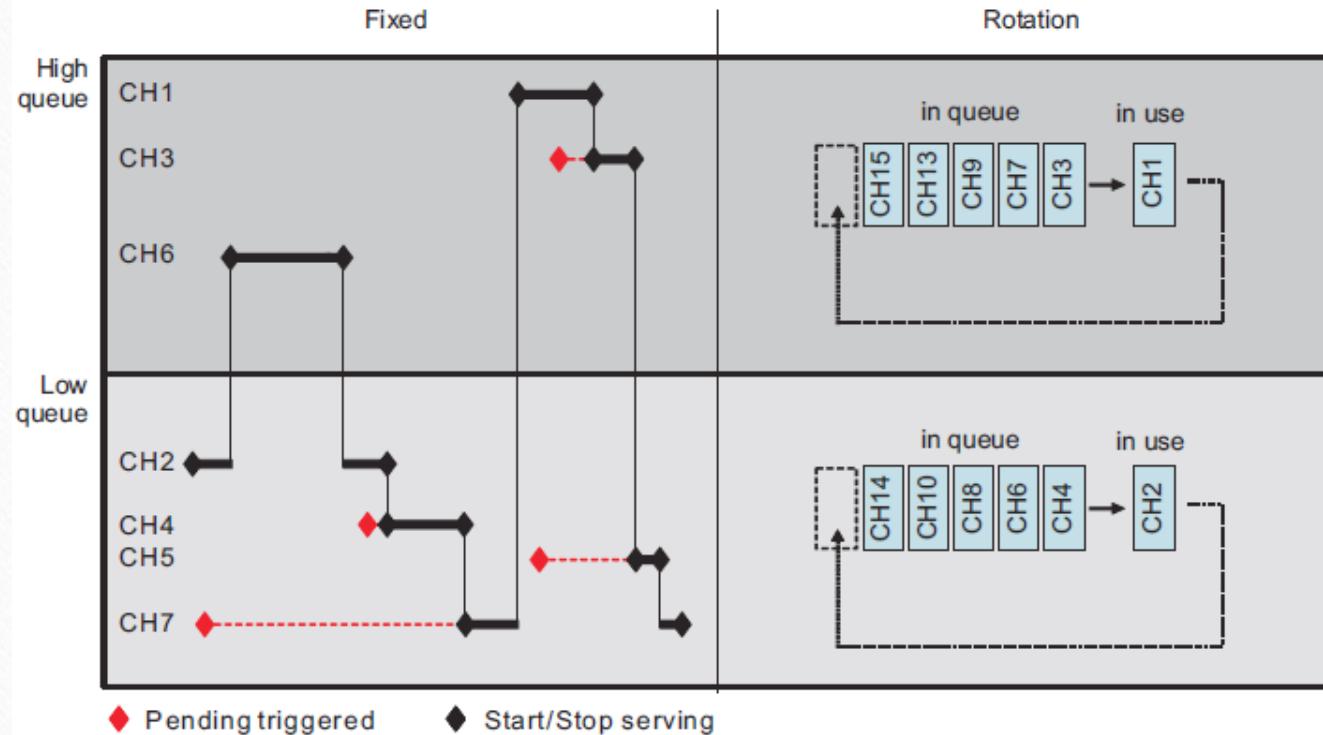
Address	Initial Source Address	Initial Destination Address	Initial Transfer Count	Reserved
Base + 0XXX0				
0x10	Initial Source Address	Initial Destination Address	Initial Transfer Count	
0x20	Channel Configuration	Element Offset Value	Frame Offset Value	
0x30	Initial Source Address	Initial Destination Address	Initial Transfer Count	
0x40	Channel Configuration	Element Offset Value	Frame Offset Value	
0x1E0				
0x1F0	Initial Source Address	Initial Destination Address	Initial Transfer Count	
0x200	Channel Configuration	Element Offset Value	Frame Offset Value	
0x800				
0x810	Reserved	Reserved	Reserved	
0x820				
0x830				
0x840				
0x850				
0x860				
0x870				
0x880				
0x890				
0x8A0				
0x8B0				
0x8C0				
0x8D0				
0x8E0				
0x8F0	Current Source Address	Current Destination Address	Current Transfer Count	

Annotations on the right side of the memory map indicate the organization of control packets:

- Primary CP0:** Base + 0XXX0 to Base + 0X1F0
- Primary CP1:** Base + 0X200 to Base + 0X810
- Primary CPnn:** Base + 0x820 to Base + 0x870
- Working CP0:** Base + 0x880 to Base + 0x8B0
- Working CP1:** Base + 0x8C0 to Base + 0x8D0
- Working CPnn:** Base + 0x8E0 to Base + 0x8F0
- Reserved:** Base + 0x8A0 to Base + 0x8F0

Event manager (prioritization, arbitration)

Figure 20-10. Example of Priority Queues



문제점

1. 통신 및 레지스터 구조 등의 사전지식이 부족하여 데이터 시트를 이해하는데 오랜 시간이 걸림.
2. 소스코드 작성에 어려움을 겪고 있음.

03

진행상황 및 문제점

황수정 : I2C 통신 학습 및 자료정리

황수정

I2C

= Inter-Integrated Circuit Module

Why?

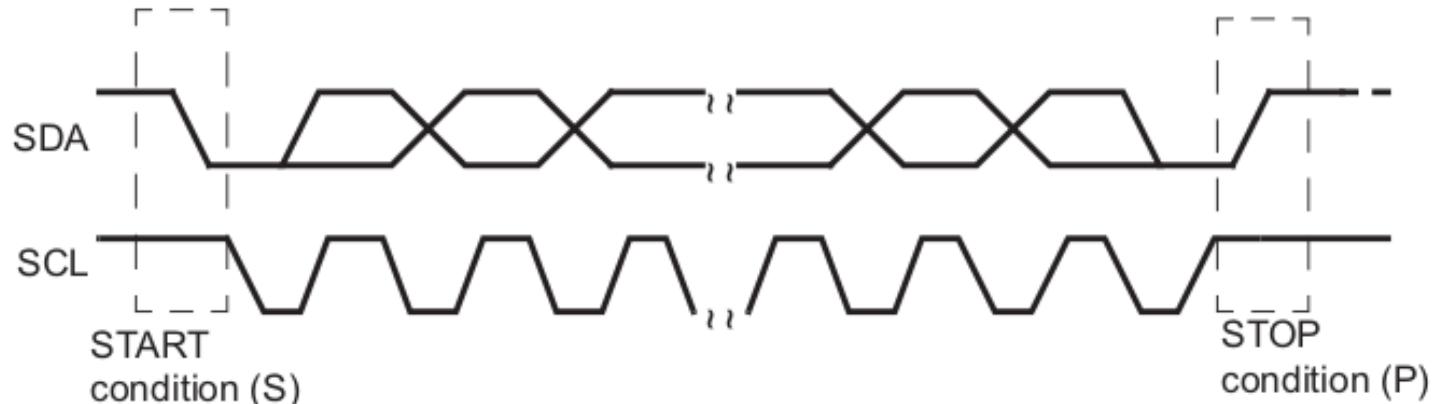
→ UART : 비동기 통신. 두 칩 간에 일종의 약속(보레이트)한 상태에서 정보를 전달하기 때문에 그 약속이 어긋나면 수신 데이터 에러가 발생할 수 있고 **전송 속도 또한 느리다.**

→ SPI : **전송 속도가 빠르다.** 동기 통신으로 1:N 통신이 가능하지만 slave device가 선택되면 그 때부터 1:1통신이다. 따라서 **slave device 수 만큼 SS핀이 필요하다.**

→ 위의 단점을 보완 한 것이다 I2C 통신
SDA, SCL 총 2개의 선으로 SS 핀이 많이 필요하지 않다.

I2C 통신 방법

Figure 31-5. I2C Module START and STOP Conditions



SDA(Serial Data) 데이터를 양방향으로 주고 받는 신호선

SCL(Serial Clock) 마스터에서 나오는 클럭 정보가 흐르는 신호선(동기 통신에서 필요)

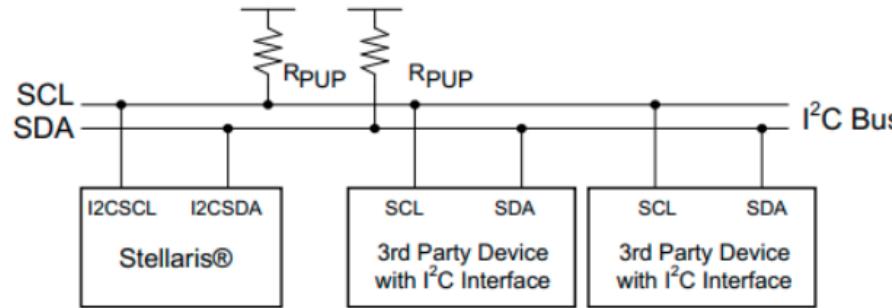
→ SDA에서 여러 신호가 high, low로 생성되면 SCL이 이를 구분하여 통신하게 된다.

SCL이 high인 상태에서 SDA가 high → low로 변할 때, 시작되며 끝은 그 반대의 경우이다.

SDA가 write 경우, 마스터 → 슬레이브로 동작하고 read 경우, 슬레이브 → 마스터로 동작한다.

→ SCL이 low인 상태에서 데이터 변경이 가능하다.

I²C 통신 방법



SDA과 SCL은 풀업 저항으로 연결해 주어야 한다.

통신 라인을 사용하지 않을 때는 라인이 high 상태로 유지하도록 하는 역할을 하기 때문이다. 양방향으로 전기 신호를 주기 (플로팅 상태는 잡음에 매우 취약해 시스템이 불안정해진다.) 때문에 신호를 확실히 잡아 줄 역할이 필요하기 때문이다.

→ 고유 주소(슬레이브가 가지고 있는 주소)를 소프트웨어적으로 찾을 수 있어 한 마스터가 127개의 장치까지 제어가 가능하다. 슬레이브 선택을 위해선 주소 데이터가 불가능하기 때문에 긴 데이터 통신에 부적절하다.

I2C 통신방법

Figure 31-6. I2C Module Data Transfer

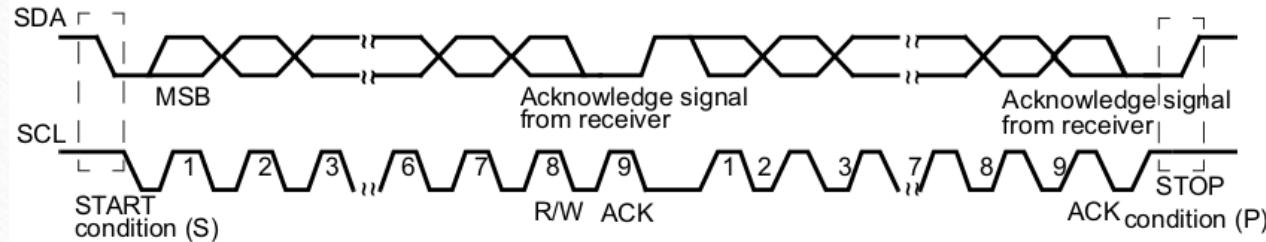
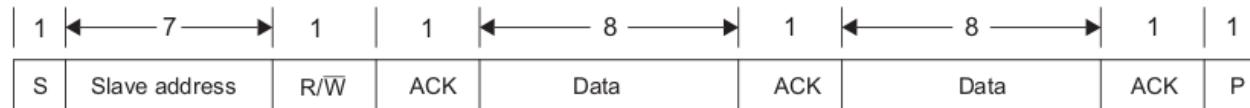


Figure 31-7. I2C Module 7-Bit Addressing Format



1. 마스터력가 자신이 원하는 슬레이브의 7비트 주소를 출력
2. 버스에 연결된 슬레이브 장치들은 SDA선을 감시하면서 자신의 주소와 일치하는지 여부를 검사
3. 주소가 같은 경우, ACK비트에 ‘0’을 출력하고 없을 경우 ‘1’의 상태를 유지한다. 1의 경우, 마스터는 정지조건을 출력하고 통신을 다시 시작한다.
4. Slave 주소 7비트 다음에 오는 8번째 비트는 다음 동작이 R/W(0=R / 1=W)인지 알려 준다.

I2C - CCS 함수

i2cInit	→ i2c 모듈 초기화 설정 함수
i2cSetOwnAdd	→ own address, 마스터 주소를 모듈 셋팅 함수
i2cSetSlaveAdd	→ 슬레이브 주소를 모듈 셋팅 함수
i2cSetBaudrate	→ 런타임 baudrate 변경 함수
i2cIsTxReady	→ Tx buffer ready flag가 설정되었는지 확인하는 함수 플래그가 설정되지 않았으면 0을 반환하지 않고 자기 자신을 반환
i2cSendByte	→ 폴링 모드에서 byte 하나 씩 보내는 함수 보내기 전 전송 버퍼가 비어있을 때까지 루틴에서 대기한다. → i2cIsTxReady를 사용해서 버퍼를 확인하면 대기를 피할 수 있다.
....	

문제점

- ◎ 통신, 회로에 대한 기초 지식 부족으로 각 기능에 대한 이해도가 떨어져 코드 활용 등에 대한 속도가 느림.
- ◎ 프로젝트를 위한 MCU 코드 작성이 원활하게 이루어지지 않음.
 - MCU에 대한 전반적인 이해와 각 함수에 대한 이해가 필요하다고 판단.
빠른 복습과 함께 코드 구현 및 실험
 - 주어진 코드 해석 후, 프로젝트 관련 코드 구현.

03 진행상황 및 문제점

이유성 – PID제어 및 Periphral 분석

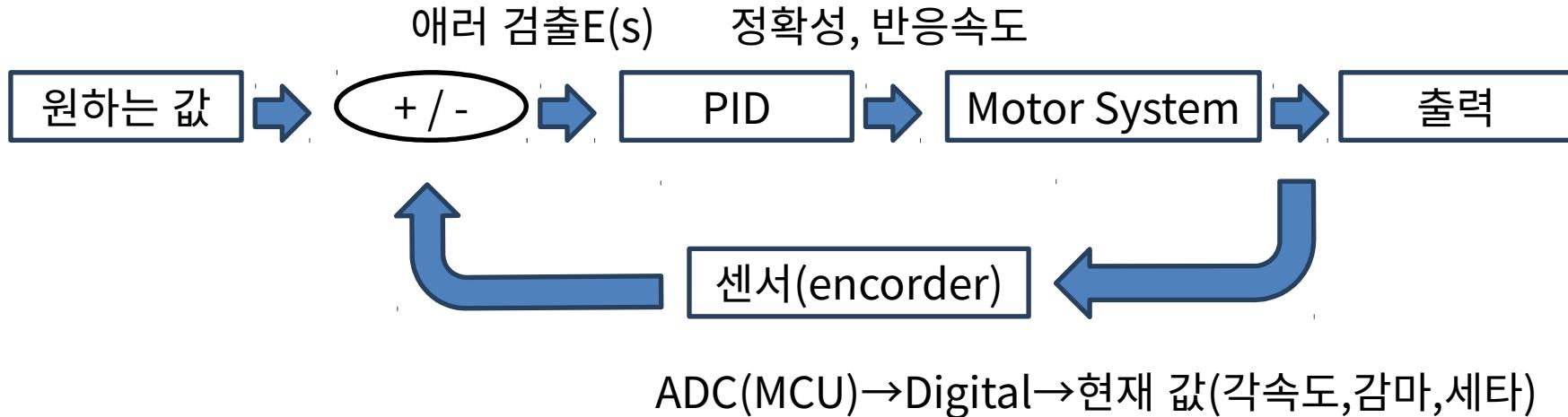
이유성

우선 팀에서 PID제어를 맡았고, PID제어는 블럭선도를 축약하는 것까지 정리 했습니다. 이후 스터디에 참여하면서 먼저 MCU Peripheral을 분석/정리할 계획이고 부족한 C언어도 공부할 생각입니다.

*블럭선도 정리

*우선 UART 통신을 먼저 분석(미완성)

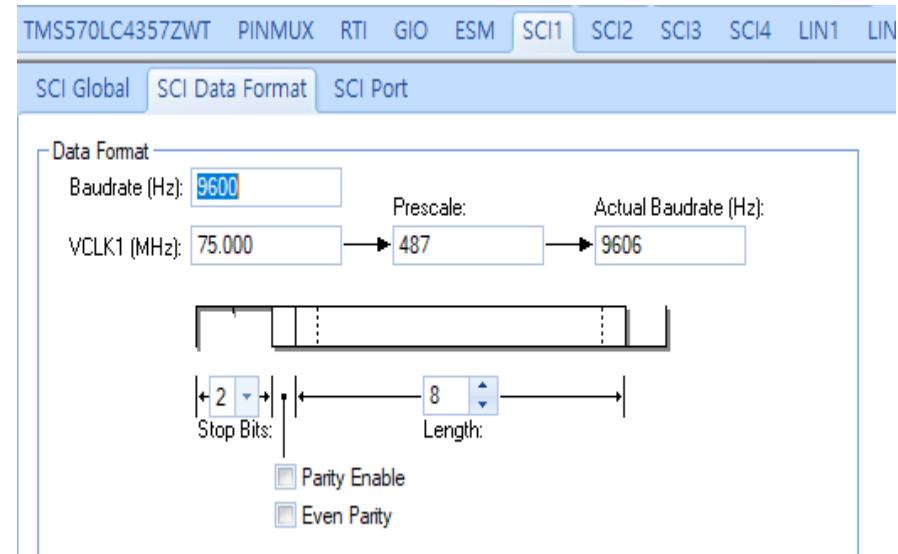
* 블럭선도 정리



* UART 통신

 Click and mark the required modules for driver compilation from below:

- Enable RTI driver
- Mark/Unmark all drivers
- Enable GIO driver **
- Enable SCI drivers
 - Enable SCI3 driver **
 - Enable SCI4 driver **
- Enable LIN drivers
 - Enable LIN1 driver ** / Enable SCI1 driver **
 - Enable LIN2 driver ** / Enable SCI2 driver **



The screenshot shows the configuration interface for the TMS570LC4357ZWT microcontroller. The top navigation bar includes tabs for TMS570LC4357ZWT, PINMUX, RTI, GIO, ESM, SCI1, SCI2, SCI3, SCI4, LIN1, and LIN. The SCI1 tab is selected. Below the tabs, there are three sub-tabs: SCI Global, SCI Data Format (which is currently active), and SCI Port. The main panel is titled "Data Format" and contains the following information:

- Baudrate (Hz): 9600
- Prescale: 487
- Actual Baudrate (Hz): 9606
- VCLK1 (MHz): 75.000
- Bit timing diagram showing a 1-bit period divided into 8 segments, with a 2-bit stop time indicated.
- Stop Bits: 2
- Length: 8
- Parity settings: Parity Enable, Even Parity

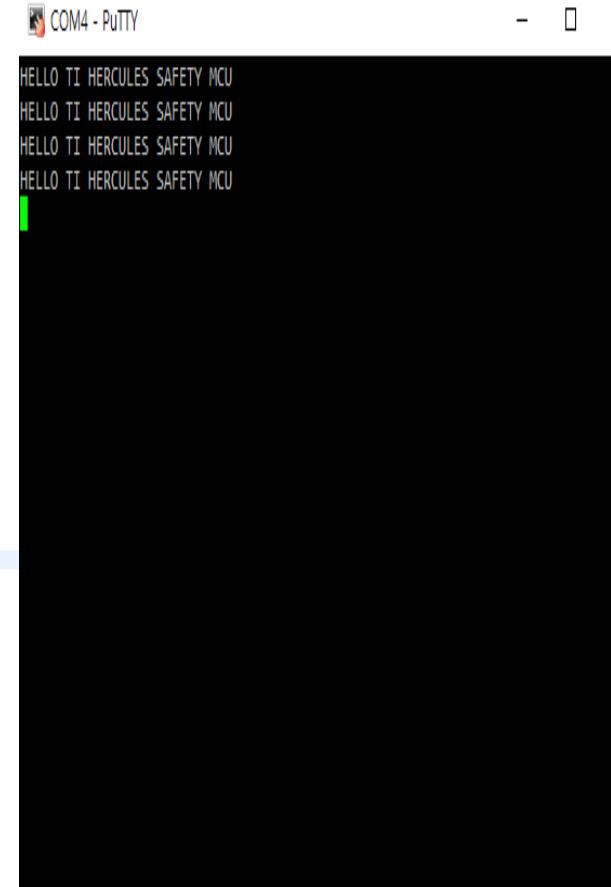
USB UART 를 사용하기 위해 SCI1을 활성화 시킨다.
Baud rate 9600

03

진행상황 문제점

```
1 #include <HL_hal_stdtypes.h>
2 #include <HL_reg_sci.h>
3 #include <HL_sci.h>
4
5 uint8 TEXT1[7] = {'H','E','L','L','O','\n'};
6 uint8 TEXT2[13] = {'T','I',' ','H','E','R','C','U','L','E','S','\n'};
7 uint8 TEXT3[13] = {'S','A','F','E','T','Y','\n','M','C','U','\n','\n'};
8
9 void delay(int counter);
10 void sciDisplaytext(sciBASE_t *sci, uint8 * data, uint32 length);
11
12 int main(void)
13 {
14     sciInit();
15
16     while(1){
17         sciDisplaytext(sciREG1,&TEXT1[0],strlen(TEXT1));
18         sciDisplaytext(sciREG1,TEXT2,strlen(TEXT2));
19         sciDisplaytext(sciREG1,TEXT3,strlen(TEXT3));
20         delay(80000000);
21     }
22
23     return 0;
24 }
25
26 void sciDisplaytext(sciBASE_t *sci, uint8 * data, uint32 length)
27 {
28     int i;
29     for(i = 0 ; i < length;i++)
30         sciSendByte(sciREG1, data[i]);
31 */
32 * while(length--)
33 *
34 *   while((sciREG1->FLR & 0x4) == 4
35 *   ;
36 *   sciSendByte(sciREG1 , *data++);
37 *   }
38 *
39 */
40
41 }
42
43 void delay(int counter)
44 {
45     int i;
46     for(i = 0 ; i <counter ; i++)
47         ;
48 }
49
```

COM4 - PUTTY



```
HELLO TI HERCULES SAFETY MCU
```

문제점

1. PID제어에서 블럭선도 축약하는 것을 많이 해보지 않았다.
 2. UART통신을 많이 구현해 보지 못했다.
 3. UART통신 자료조사는 했지만 정리를 못했다.
 - 4.
 - 5.
- ...
- 공부를 많이 안했다..

04

진행계획

안상재

- SPI 통신자료 정리 및 구현
- DSP 보드 기반의 네트워크
프로그래밍

정유경

- eQEP 자료 정리 및 코드구현
- 영상처리학습
(캐니에지, 가우시안 블러링,
번호판검출하기)

황수정

- I2C 통신자료정리 및 구현

이유성

- 나머지 Peripheral 분석 / 정리 (UART,I2C,
CAN,RTI,PWM.)
- 제어공학 5장 공부 (+블럭선도, 실제제어
해보기)

김시윤

- eCAP 자료 정리 및 구현
- PID 제어 학습

문지희

- DMA 자료 정리 및 구현

감사합니다