

03

진행상황 및 문제점

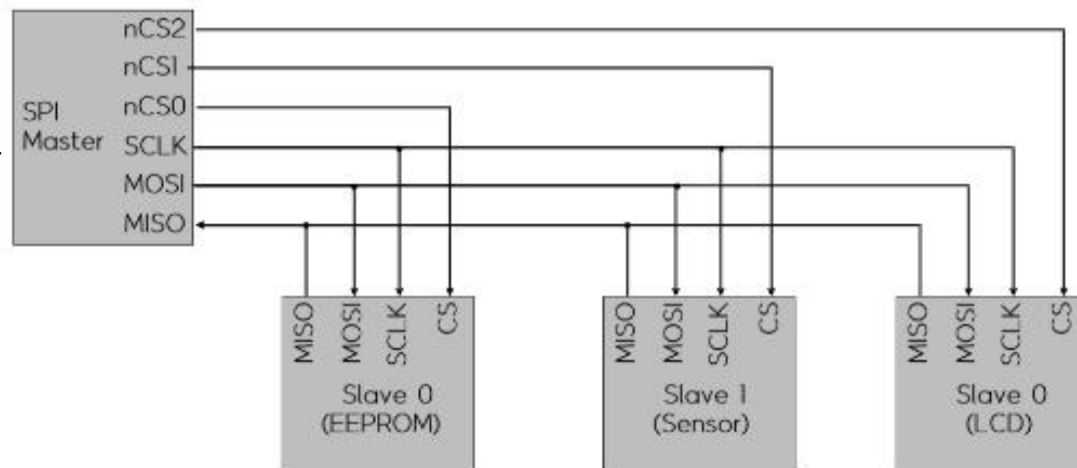
이유성-MCU UART, 회로, C

●MCU UART

통신 UART - SPI - I2C - CAN

- UART
1. 비동기 Serial 통신이고, 1:1 통신이며 Tx-Rx선이 있어서 전이중 전송방식이다.
 2. Start Bit - Stop Bit 가 있어서 느리고, 전송 효율이 낮다 = 동일 클럭을 공유하지 않기 때문에 데이터 라인 상에서 0과 1을 계속 측정하여 변화를 관측해야 하기 때문에 전송 효율이 낮다.
 3. 송-수신간에 Buad rate가 안 맞을 경우 Error (데이터가 누락 될 수 있다 = 불안정)

- SPI
1. 동기 Serial 통신이고, 1:N통신이며(장치가 선택되면 1:1통신)CLK선을 이용해서 빠르며 비동기통신에 비해 안정적이다.
 2. Master 노드는 클럭제공 Slave 장치 선택하는 기능
 3. 주소영역을 사용하지 않으므로 여러 개의 Slave장치에 대한 어드레싱을 위해 각 장치 별 CS(Chip Select) 신호선이 추가 되어야 하는 문제점이 있어서 비용이 많이 들고 소형화가 힘들다.

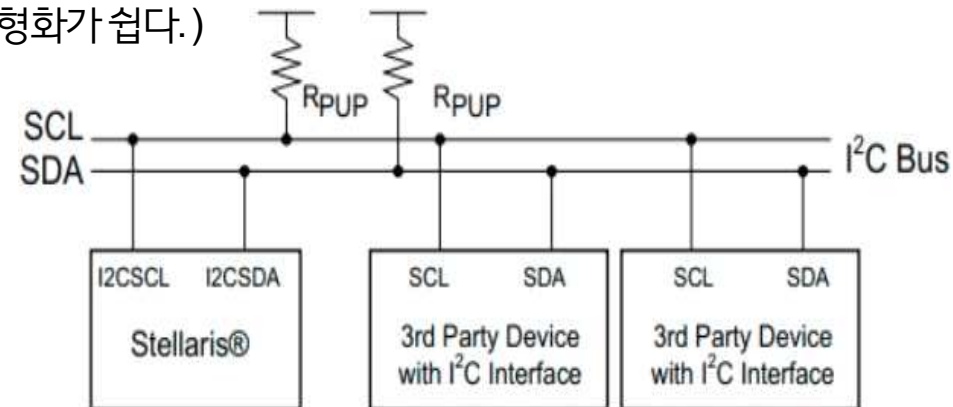


●MCU UART

통신 UART - SPI - I2C - CAN

I2C

1. 동기 Serial 통신이고 1:N통신이며 2개의 클럭선과 데이터선(버스선)이 있다.(SPI보다 소형화가 쉽다.)
2. SDA(데이터선)은 양방향 전송방식이다.
3. 10~400Kbp 저속도이며, 저용량데이터 전송방식이다.
4. CLK가 1일 때 SDA가 1→0이면 Start, SDA가 0→1이면 Stop하는 Protocol방식.



CAN

1. 비동기 Serial 통신이고 1:N통신이며 ID가 있다.
2. 대용량 전송 방식.
3. 잡음에 강하고 Data선이 2개(빠른 Data 전송선과 느린 Data 전송선)가 있다.
4. Data 정확도가 높다.

●MCU UART

UART (Universal Asynchronous Serial Receiver and Transmitter)

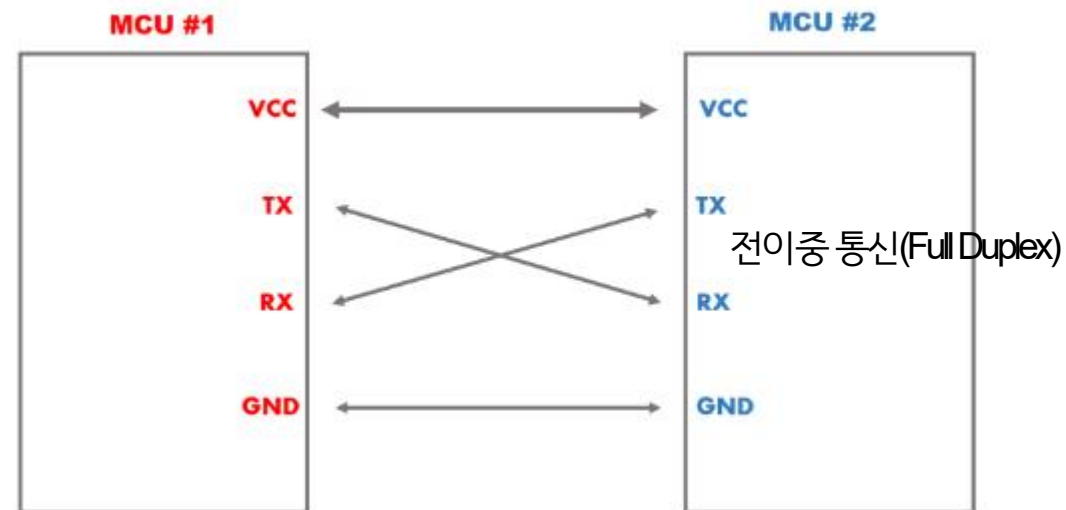
1. 비동기 Serial 통신이고, 1:1 통신이며 Tx-Rx선이 있어서 전이중 전송방식이다.

동기화를 한다는 것은 무엇을 공유하면서 템포를 맞춘다는 의미로 받아들일 수 있다.

↳ 대상: 클럭

↳ 데이터를 송수신한다.

↳ 양쪽의 클럭이 정확하게 동기화 되어야함.



UART는 클럭선이 없기 때문에 다른 것을 맞춰줘야하는데 그것이 Baud rate라는 Data 전송 속도이다.

●MCU UART

UART (Universal Asynchronous Serial Receiver and Transmitter)

UART 통신의 프로토콜

1. Logic은 1로 유지된다. 여기서 데이터가 전송을 시작할 때 Logic은 1에서 0으로 바뀌게 되고, Falling Edge를 검출하여 데이터 전송을 시작하겠다.(Start Bit를 보내게 된다)

물론 Falling Edge를 검출한 이후에도 1비트의 길이보다 더욱더 빠르게 여러번 데이터를 검출하여 Logic0으로 바뀐것이 노이즈에 의한 것이 아니라는것을 판별한다. 데이터를 보낼때에도 마찬가지이다. 일반적으로 8비트를한번에 보내게 되는데 데이터를 전송하고 난뒤에 각비트의 데이터 전송 중에 여러번 데이터를 샘플링하여 데이터가 노이즈에 의한것이 아니라는것을 확인한다.

그리고 마지막으로 PB 즉 Parity Bit를 체크하고 다시 STOP Bit, Logic0 에서 Logic 1로 변경함으로써 데이터 전송이 끝났다는것을 알리게 된다

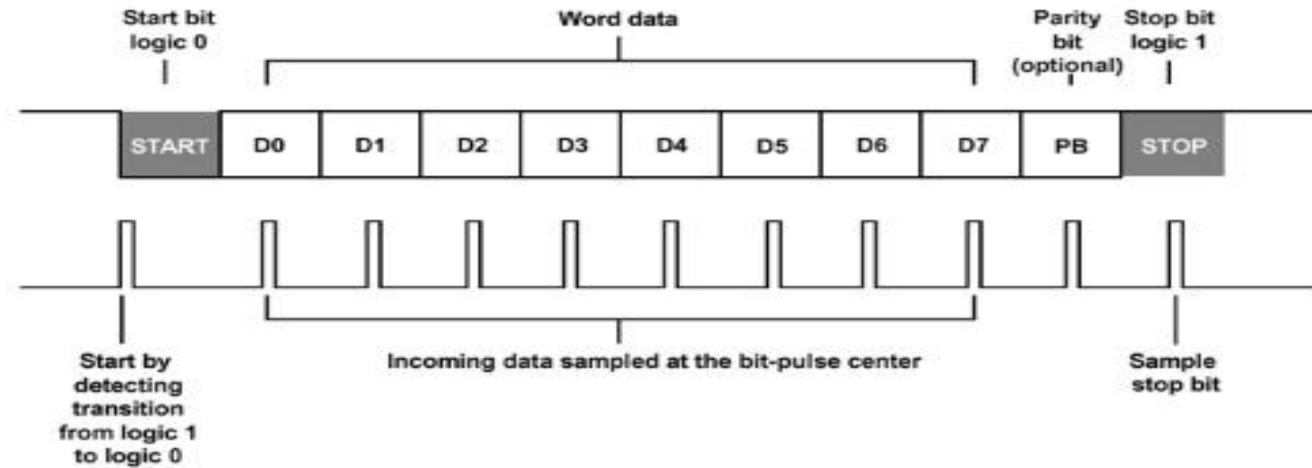
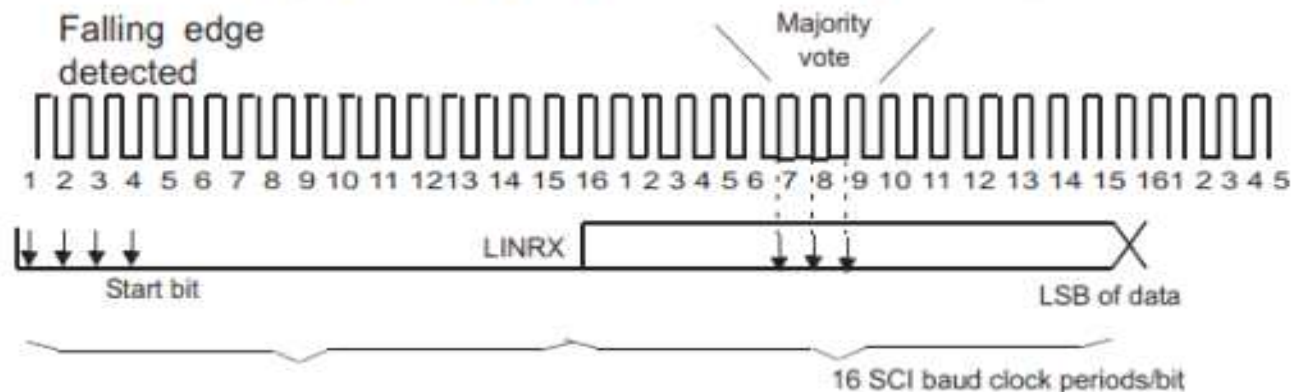


Figure 29-4. Asynchronous Communication Bit Timing



●MCU UART

UART- 입력한 것을 그대로 출력하고 엔터누르면 개행하기.

HCG설정

1.


- ☐ Enable LIN drivers
☐ Enable LIN1 driver ** / ☒ Enable SCI1 driver **
☐ Enable LIN2 driver ** / ☐ Enable SCI2 driver **

2.

Data Format

Baudrate (Hz):

VCLK1 (MHz): → Prescale: → Actual Baudrate (Hz):



Stop Bits: Length:

☐ Parity Enable
☐ Even Parity

●MCU UART

```
#include <HL_hal_stdtypes.h>
#include <HL_reg_sci.h>
#include <HL_sci.h>

int main(void)
{
    int i=0;
    sciInit();
    uint8 msg = 0;

    while(1)
    {
        while(1){
            here :
            while(!sciIsRxReady(sciREG1)) //입력을 받을 때 까지 기다림
                msg = sciReceiveByte(sciREG1);

            if(msg == '\r' || msg == '\n')
                if(msg == '\r' || msg == '\n'){
                    sciSendByte(sciREG1, '\n');
                    sciSendByte(sciREG1, '\r');
                    goto here ;
                }

            break;
        }
        sciSendByte(sciREG1,msg);
    }

    return 0;
}
```

// 입력 한 것을 그대로 출력하다가 엔터를 넣을 때 개행만 해주면 프로그램.

●MCU UART

UART+GIO PUTTY입력 값에 따라 LED를 토글시킴 정해진 값아닌 것을 누르면 error메세지가 뜬.

HCG설정

1.

- ☒ 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 **

2.

Data Format

Baudrate (Hz):

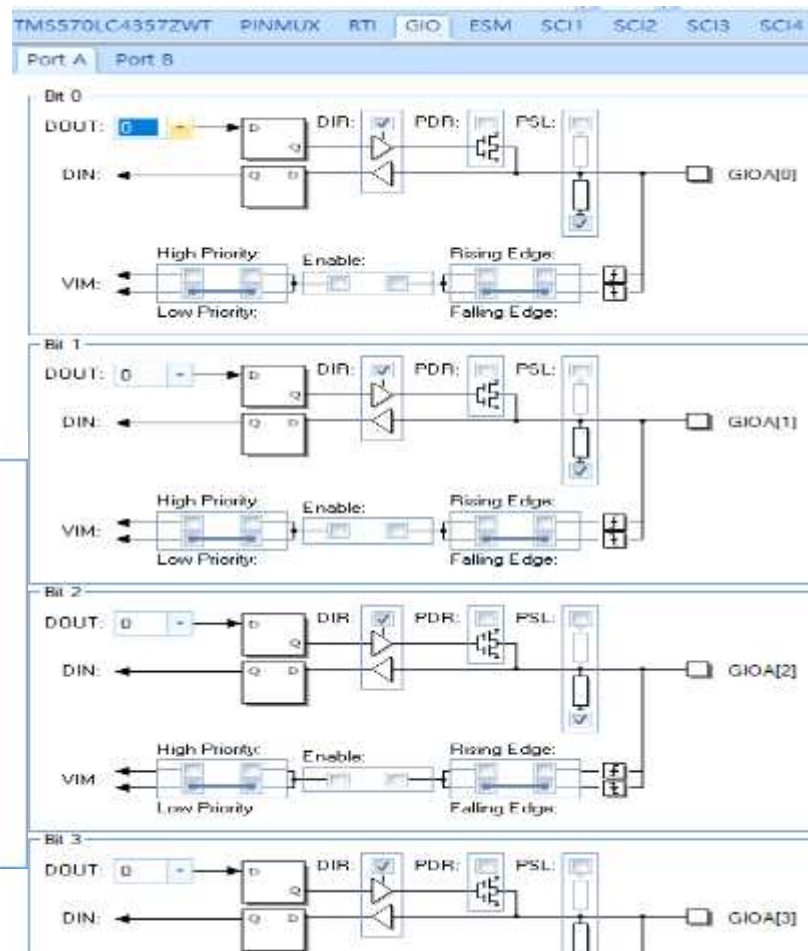
VCLK1 (MHz): Prescale: Actual Baudrate (Hz):

Stop Bits: Length:

☐ Parity Enable

☐ Even Parity

3.



●MCU UART

```
#include <HL_gio.h>
#include <HL_hal_stdtypes.h>
#include <HL_reg_gio.h>
#include <HL_reg_sci.h>
#include <HL_sci.h>
#include "string.h"
#include "stdlib.h"
#include "stdio.h"

void gio_msg(void);
void control_gio(void);

uint8 counter = 0;

void main(void)
{
    uint8 msg[32] = {0};
    gioInit();
    sciInit();

    while (1)
    {
        while (!sciIsRxReady(sciREG1))
        {
            counter = sciReceiveByte(sciREG1);
            control_gio();

            if (counter <= 52 && counter >= 49)
            {
                sprintf(msg, "Current value : %d \n\n", counter-48);
                sciSend(sciREG1, strlen(msg), msg);
                gio_msg();
            }
        }
    }

    void gio_msg(void)
    {
        char msg2[32] = {0}; //책은 초기화할 때 0만 넣어주면 0~31다 0이 들어가 있다.
        msg[31] = 0;
        // if (counter<= 52 || (counter-48) >= 49){
        //     sprintf(msg2, "Toggle %d pin\n\n", counter-48);
        //     sciSend(sciREG1, strlen(msg2), msg2);
        // }
        /*
        sciSend(sciREG1, 7, "Toggle ");
        sciSendByte(sciREG1, counter);
        sciSend(sciREG1, 5, "pin\n\n");
        */
    }
}
```

```
void control_gio(void)
{
    uint8 err_msg[32] = {0};
    switch (counter-48)
    {
        case 1:
            gioToggleBit(gioPORTA, 0);
            break;
        case 2:
            gioToggleBit(gioPORTA, 1);
            break;
        case 3:
            gioToggleBit(gioPORTA, 2);
            break;
        case 4:
            gioToggleBit(gioPORTA, 3);
            break;
        default:
            sprintf(err_msg, "default value : %d\r\n", counter-48);
            sciSend(sciREG1, strlen(err_msg), err_msg);
            break;
    }
}
```

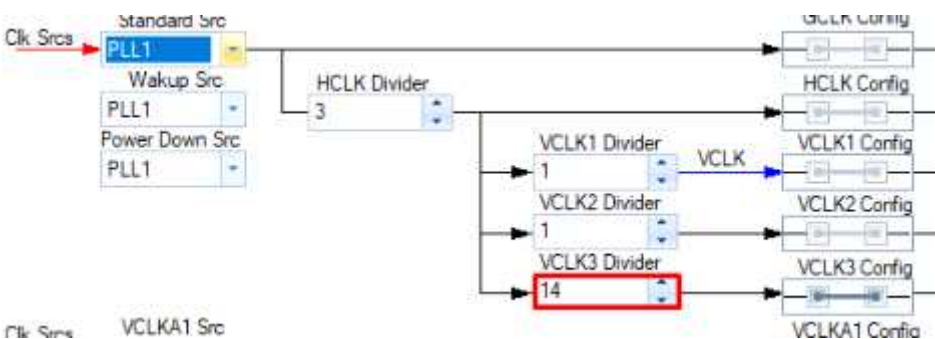
//PUTTY에 1,2,3,4를 누르면 1,2,3,4번에 연결된 LED를 토클시킴 1,2,3,4외에 누르면 error메세지가 뜬다. 패더링도 없다.
//문제점 동시에 안 됨.

●MCU UART

UART+PWM- 입력하는 값에 따라 pwm이 달라짐(servo모터 사용해서 조향각제어)

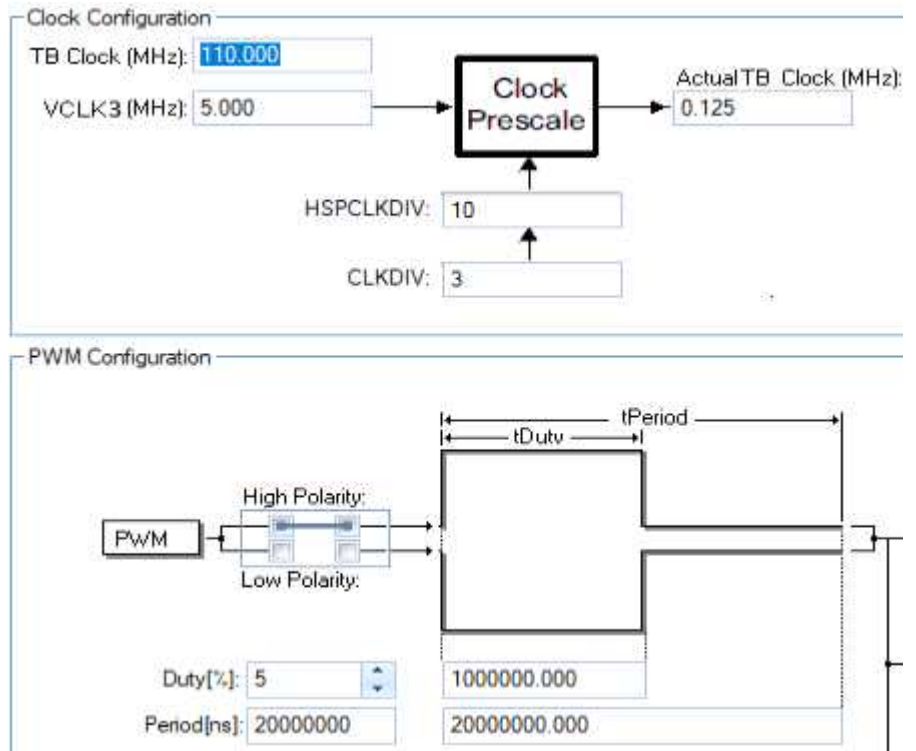
HCG설정

1. ☐ Enable LIN drivers
☐ Enable LIN1 driver ** / ☒ Enable SCI1 driver **
☐ Enable LIN2 driver ** / ☐ Enable SCI2 driver **

2.  Clock Configuration Diagram showing the clock tree. The main clock source is PLL1, which is divided by HCLK Divider (3) to produce HCLK. HCLK is then divided by VCLK1 Divider (1), VCLK2 Divider (1), and VCLK3 Divider (14) to produce VCLK, VCLK2, and VCLK3 respectively. The VCLK3 Divider is highlighted with a red box.

3. Data Format
 Baudrate (Hz): 9600
 VCLK1 (MHz): 37.500
 Prescale: 243
 Actual Baudrate (Hz): 9606
 Stop Bits: 2
 Length: 8
☐ Parity Enable
☐ Even Parity

4. - Enable ETPWM modules
☒ Enable ETPWM1

5.  Clock Configuration and PWM Configuration Diagrams. The Clock Configuration section shows TB Clock (MHz) at 110.000, VCLK3 (MHz) at 5.000, and Actual TB Clock (MHz) at 0.125. The PWM Configuration section shows a PWM signal with High Polarity and Low Polarity options, Duty[%] at 5, and Period[ns] at 20000000. The diagram also shows the relationship between tDuty and tPeriod.

●MCU UART

```
#include "HL_sys_common.h"
#include "HL_sci.h"
#include "HL_etpwm.h"

uint8 counter ;

void message_func(void);
void control_pwm(void);

void main(void)
{
    sciInit();
    etpwmInit();
    etpwmStartTBCLK();

    while(1){
        while(!sciIsRxReady(sciREG1))1
            ;
        counter = sciReceiveByte(sciREG1);
        message_func();
        if(counter>=49 && counter <= 57){
            control_pwm();
        }
    }

    void message_func(void)
    {
        uint8 msg[32] = {0};
        uint8 err_msg[32] = {0};
        int rotation = -90 + ((counter-49)*22.5);
        if(counter>=49 && counter <= 57){
            sprintf(msg,"current value = %d\r\n",counter-48);
            sciSend(sciREG1,strlen(msg), msg);
            sprintf(err_msg,"current rotation value : %d\r\n",rotation);
            sciSend(sciREG1,strlen(err_msg),err_msg);
        }
        else{
            sprintf(err_msg,"%d is default value\r\n",counter -48);
            sciSend(sciREG1,strlen(err_msg),err_msg);
        }
    }
}
```

```
void control_pwm(void)
{
    int x = counter -48;
    switch(x){
        case 1 :
            etpwmSetCmpA(etpwmREG1, 125 + ((14.625)*(x-1)));

        case 2 :
            etpwmSetCmpA(etpwmREG1, 125 + ((14.625)*(x-1)));

        case 3 :
            etpwmSetCmpA(etpwmREG1, 125 + ((14.625)*(x-1)));

        case 4 :
            etpwmSetCmpA(etpwmREG1, 125 + ((14.625)*(x-1)));

        case 5 :
            etpwmSetCmpA(etpwmREG1, 125 + ((14.625)*(x-1)));

        case 6 :
            etpwmSetCmpA(etpwmREG1, 125 + ((14.625)*(x-1)));

        case 7 :
            etpwmSetCmpA(etpwmREG1, 125 + ((14.625)*(x-1)));

        case 8 :
            etpwmSetCmpA(etpwmREG1, 125 + ((14.625)*(x-1)));

        case 9 :
            etpwmSetCmpA(etpwmREG1, 125 + ((14.625)*(x-1)));

        default :
    }
}
```

//<http://akizukidenshi.com/download/ds/towerpro/SG90.pdf> sg90 서보모터 데이터 시트
// 문제점 : -90~90도까지 돌아야하는데 -90에서 0도까지밖에 돌지 않는다.

모터주기 20ms 맞추고 PWM Duty 1ms~2ms로 Servo -90~90도 제어하는 것인데 실제 반밖에 돌아가지 않음. Duty를 1ms보다 적게 했더니 더 돌아감 (오실로스코프로 측정해 볼 것.)

●MCU UART

```

#include "HL_sys_common.h"
#include "HL_etpwm.h"

void func(int x);
void delay(int delay);

void main(void)
{
    int x = 0;
    etpwmInit();
    etpwmStartTBCLK();

    while(1){
        func(x++);
        delay(20000000);
        if(x == 19)
            x = 0 ;
    }
}

void func(int x)
{
    etpwmSetCmpA(etpwmREG1, 51.875+ ((14.625)*x));
}

void delay(int delay)
{
    int i ;
    for(i = 0 ; i < delay ; i++)
        ;
}

// 이렇게하면 서보모터가 거의 180도 돌아간다.

```

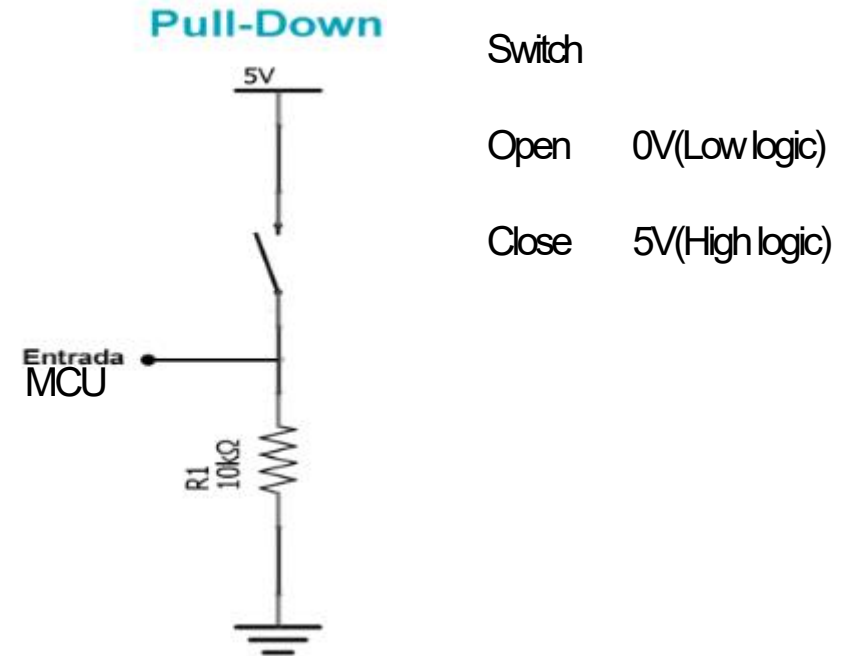
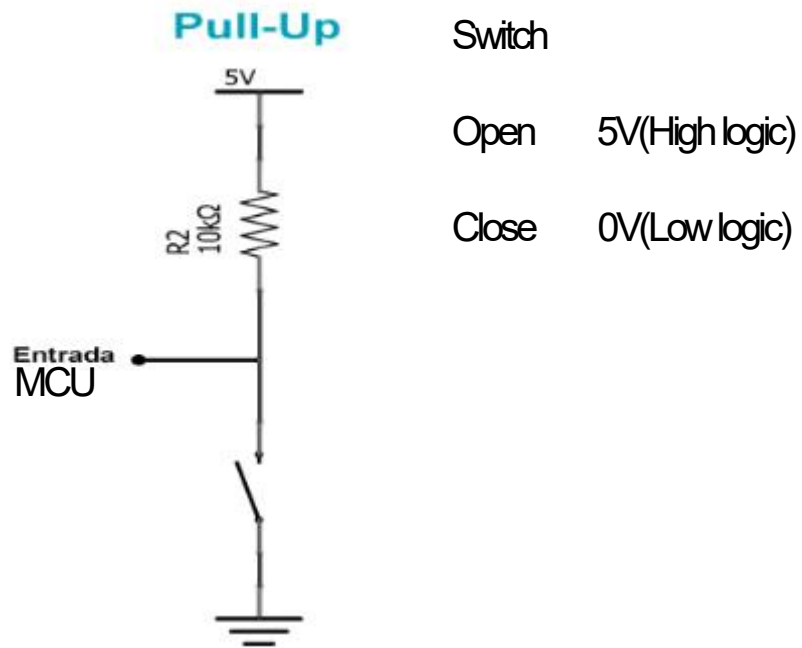
실제 듀티비를 조정 했더니 180도 돌아감.

duty 0.5ms ~ duty 2.5ms까지 작동한다.

● 회로 (Pull up , Pull down, Open Drain(Open Collector) , Push-pull 회로 , 싱크전류 , 소스전류)

* Pull up

* Pull down



왜 이렇게 필요할까?

ex) $3.3V \uparrow = 1$, $2.2V \downarrow = 0$ 으로 가정하면 2.5V는 0?, 1?이라고 하기에 애매한 값(이 경우를 Hi-Z(하이 임피던스) or Floating 상태라고함)
이 나오는데 이 값은 디지털 신호에서 치명적이다(오작동을 일으킬 수 있다.)
결과적으로 오작동이 나지 않게 확실히 0 또는 1로 만들어 주는 역할을 한다.

● 회로 (Pull up , Pull down, Open Drain(Open Collector) , Push-pull 회로 , 싱크전류 , 소스전류)

* **Open Drain(Open Collector)** - MOSFET or TR로 만들었냐에 따라 명칭이 달라짐.

Pull up 저항과 같이 쓰는게 일반적.

Switch가 Master Chip 내부에 들어가 있는 경우를 Open Collector라고 함

Master Chip : 다른 칩을 컨트롤하는 디지털 칩

Slave Chip : 다른 칩에 의해서 컨트롤 하는 디지털 칩

그림;

명령 1	명령2	Output
Low	Low	Low
Low	High	Low
High	Low	Low
High	High	High

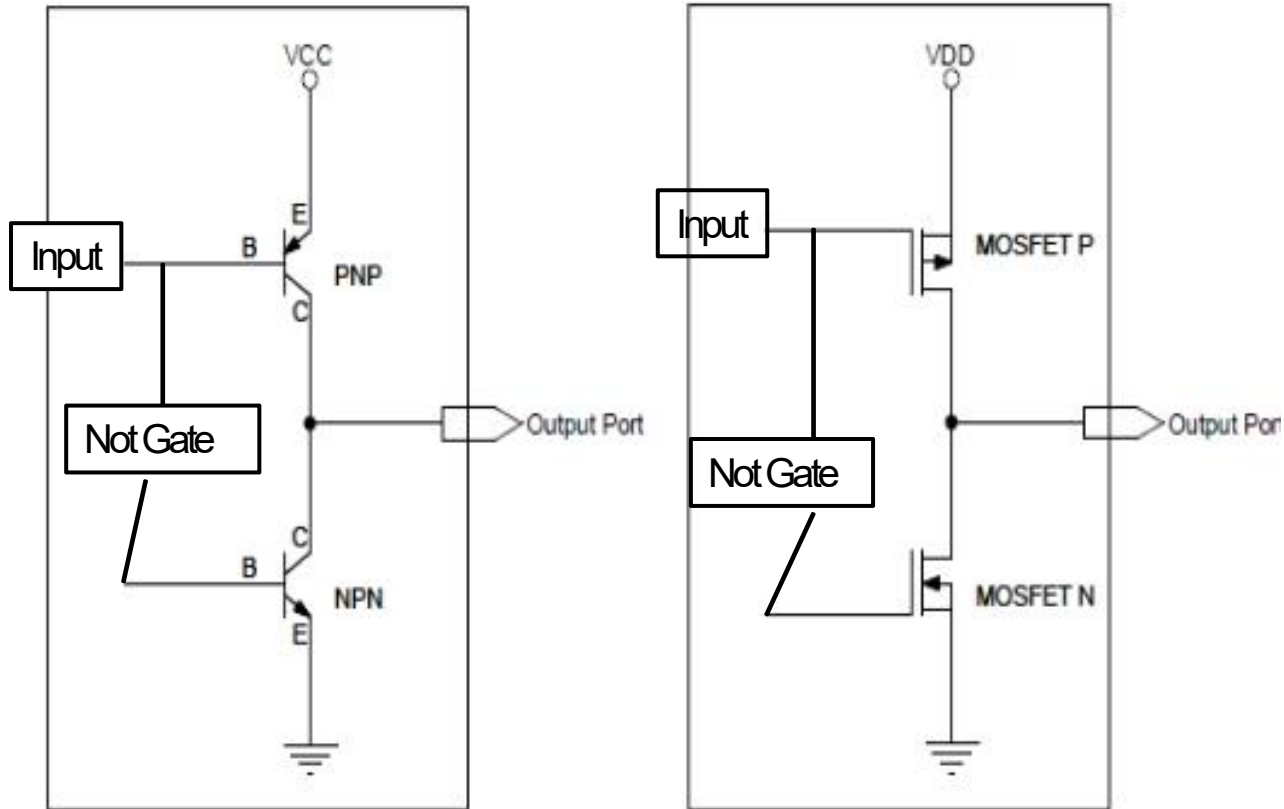
특징 : 개별소자의 출력이 모두 논리 '1'일 때만
전체 출력이 1이다.

Open Collector

- * 장점 1. 확장성이 뛰어난 와이어드 AND 연결이다.
2. 소자의 동작 전원 전압과 다른 전압을 출력할 수 있다.
- * 단점 1. 토템폴 소자에 비해 소자가 느리다.
2. Pull-up 저항으로 인해 전력 소비가 발생한다

● 회로 (Pull up, Pull down, Open Drain(Open Collector), Push-pull 회로, 싱크전류, 소스전류)

* Push-pull 회로



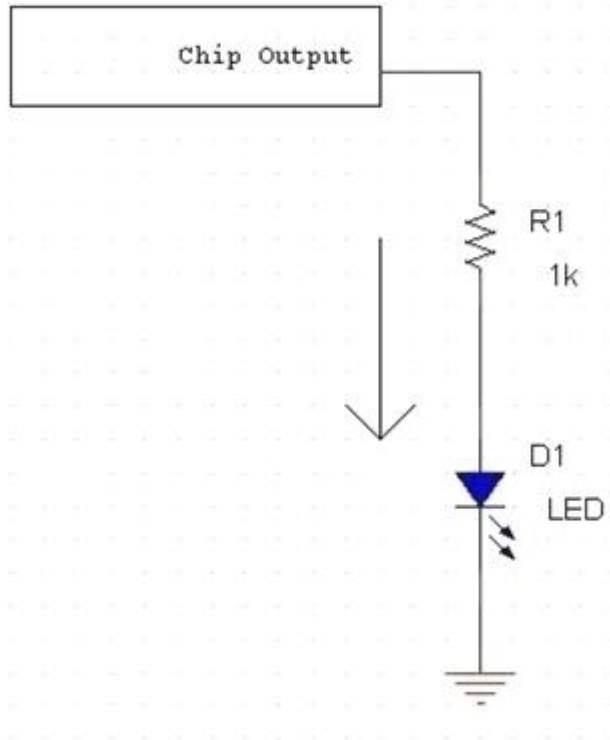
위쪽 Tr이 Vcc쪽으로 밀어올리는 Push 역할이고,
아래 Tr이 GND쪽으로 당기는 Pull의 역할을 한다.

두 개의 Tr은 입력신호가 동일하면 안되므로
일반적으로 Not gate를 통해 서로 다른 신호를
입력하게 한다.

Pull up, Pull down과 비슷한 개념

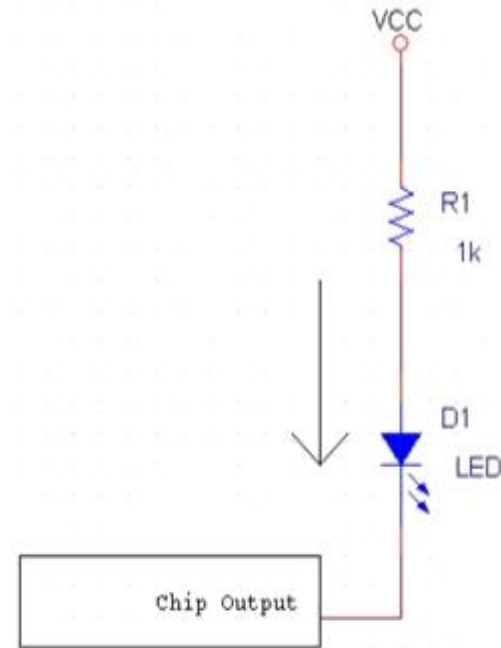
● 회로 (Pull up , Pull down, Open Drain(Open Collector) , Push-pull 회로 , 싱크전류 , 소스전류)

* 싱크 전류(Sink Current), 소스 전류(Source Current) - MCU에 무리가 가서 거의 안쓴다.



- 칩이 싱크 전류를 가진다면 출력 쪽으로 전류가 흘러간다는 뜻이다.

- 칩의 출력과 +전원 사이에 소자를 연결하여 칩의 출력이 Low(0V)일 때 동작한다



- 칩이 소스 전류를 가진다면 출력에서 바깥으로 전류가 흐른다는 뜻

- 칩의 출력과 0V 사이에 소자를 연결하여 출력이 High(1V)일 때 동작한다.

감사합니다.