

[펌웨어 프로그래밍]

기 말 프로젝트



담당 교수
팀
팀원

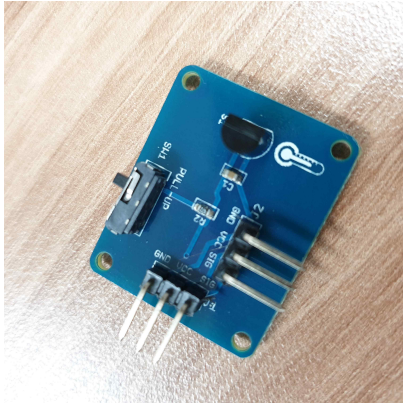
전중남 교수님
1-3
2017038061 김서기
2017038021 박진훈
2015041068 정민규
2015041062 허지용

1단계: 문제 정의

- Arduino 1:
 - ◆ LED 는 0.5 초 간격으로 계속 깜빡인다. (0.5 초 ON, 0.5 초 OFF)
 - ◆ 리모콘에서 숫자를 누르면, LED 의 밝기가 변한다. 0-가장 어둡게, 9-가장 밝게
 - ◆ Serial Monitor 에서 'T' 또는 't'를 전송하면
 - Arduino 2 에게 I2 통신으로 온도를 보내라고 요청하고,
 - Arduino 2 가 보낸 온도를 받아, Serial Monitor 에 출력한다.
- Arduino 2:
 - ◆ 터치 센서를 누르면, 온도를 측정하여 4-digit display 에 출력한다.
 - ◆ I2 통신으로 온도 측정 명령을 받으면, 온도를 측정하고, I2C 로 보낸다.

2단계: 디바이스 기능 및 동작 원리 이해

2-1. 온도 센서의 동작 원리



DS18B20 온도 센서 모듈

- 온도 센서를 센싱하여 한 비트 통신(OneWire) 방식으로 전송하는 모듈이다.
- 한 개의 선(1-Wire)으로 통신하고 여러 개의 온도 센서를 한 개에 선에 멀티드롭 방식으로 연결이 가능하다.
- 64 bit serial number가 내장돼있고, 외부 회로가 불필요하다.
- 섭씨 -55 도 ~ 125 도 측정 가능하고, 섭씨 -10 도부터 85 도까지의 정확도는 +/- 0.5 도이다.
- 온도 해상도를 9 비트 ~ 12 비트까지 프로그램 가능하다. - 12 비트 변환에 최대 750 msec가 소요된다.

2-2. 터치 센서의 동작 원리



정전식 터치 센서(Capacitive Touch Sensor)

- 사람의 손에 의해 터치 센서 입력부를 만졌을 때 손가락의 표면에 센서 입력부의 전하와 반대인 전하가 모이고, 이에 따라 터치센서의 입력부의 전하가 약간 상쇄되어 터치 센서 입력부에서의 전기장이 약화됩니다.
- 이 변화가 일어난 위치를 센서가 감지하는 방식이 '정전식 감응'이라고 합니다.
- 신체 또는 특정 물체가 접촉했을 때 발생하는 정전용량의 변화를 감지하여 동작하며 인체 접촉시 발생하는 미세 정전용량의 차이를 감지하여 High 또는 Low로 신호를 출력해준다.

2-3. LED의 동작 원리



LED의 동작 원리

- LED는 전자 (마이너스 성질)이 많은 N형 (- : negative) 반도체와 정공 (플러스 성질)이 많은 P형 (+ : positive) 반도체를 접합한 것입니다.
- 이 반도체에 순방향 전압을 인가하면, 전자와 정공이 이동하여 접합부에서 재결합하고, 이러한 재결합 에너지가 빛이 되어 방출됩니다.
- 전기 에너지를 일단 열 에너지로 변환하고, 그 후 빛 에너지로 변환하는 기존의 광원에 비해, 전기 에너지를 직접 빛 에너지로 변환하기 때문에 전기 에너지가 낭비되지 않아 고효율의 빛을 얻을 수 있습니다.

2-4. TM1637 세그먼트 디스플레이 동작 원리



TM1637 세그먼트 디스플레이

- TM1637Display.h/TM1637Display.cpp 가 제공하는 라이브러리 함수로 구동
- TM1637Display(uint8_t pinClk, uint8_t pinDIO); u void setBrightness(uint8_t brightness, bool on = true);
- void setSegments(const uint8_t segments[], uint8_t length = 4, uint8_t pos = 0);
- void showNumberDec(int num, bool leading_zero = false, uint8_t length = 4, uint8_t pos = 0);
- void showNumberDecEx(int num, uint8_t dots = 0, bool leading_zero = false, uint8_t length = 4, uint8_t pos = 0);

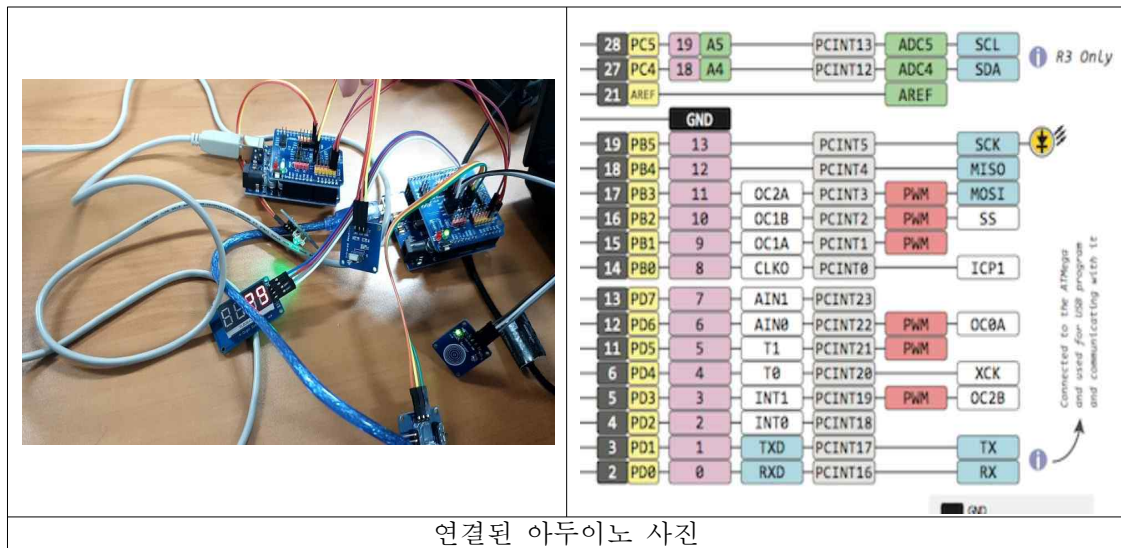
2-5. IR 수신기 모듈 동작 원리



IR 수신기의 동작 원리

- Modulation Carrier란 0과 1이 반복되는 일종의 PWM 출력신호입니다. Duty Ratio(한 Cycle에서 1과 0의 시간 비율)는 크게 중요하지 않고, 단지 수신부와 동기화된 Carrier Frequency가 중요합니다.
- Modulation된 신호를 수신측에서 송신측이 의도한 본래 신호로 되돌리려면 포토트랜지스터로 수신한 신호를 Band Pass Filter(BPF)에 통과시켜서 Demodulation하는 과정을 거쳐야 합니다.
- 적외선 수광 모듈은 포토트랜지스터와 특정 Center Frequency의 BPF를 결합하여 사용하기 편리하도록 하나의 부품으로 패키징해 제작한 부품입니다.

3단계: 회로도 이해



- **Arduino 1**
 - 터치센서 : D8
 - LED : D5 PWM 출력중 하나로 연결
 - I2 : A4 A5
- **Arduino 2**
 - 온도센서 : D2 디지털 출력중 하나로 연결
 - 4-digit display : D12 D13
 - I2 : A4 A5

4단계: 주변장치 구동 방법 이해

Master

1. Serial 통신

1-1. UBRR0 – 9600bps 로 설정. $9600\text{bps} = 103$

UBRRnL and UBRRnH – USART Baud Rate Registers

Bit	15	14	13	12	11	10	9	8	
	—	—	—	—	UBRRn[11:8]				UBRRnH
	UBRRn[7:0]								UBRRnL
	7	6	5	4	3	2	1	0	

1-2. UCSR0B – 송수신 둘다 사용. (TXENn 레지스터 비트 설정) 송신의 경우 Tx Complete가 아니라 효율성을 고려하여 UDRIEn를 사용한다. 인터럽트를 setup에서 설정하면 무한 인터럽트를 걸 가능성이 생기므로 보낼 데이터가 있을 때 해당 인터럽트 비트를 설정, 데이터를 보내고 나면 해당 인터럽트 비트를 해제해 준다.

UCSRnB – USART Control and Status Register n B

Bit	7	6	5	4	3	2	1	0	
	RXCIEn	TXCIEn	UDRIEn	RXENn	TXENn	UCSZn2	RXB8n	TXB8n	UCSRnB
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R	R/W	
Initial Value	0	0	0	0	0	0	0	0	

1-3. UCSR0C – 비동기, 패리티없음, 스톱비트 1, UCSZ:110 으로 설정한다

UCSR0C = 0b00000110

UCSRnC – USART Control and Status Register n C

Bit	7	6	5	4	3	2	1	0	
	UMSELn1	UMSELn0	UPMn1	UPMn0	USBSn	UCSZn1	UCSZn0	UCPOLn	UCSRnC
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	1	1	0	

1-4. 큐 - 큐를 사용하여 송신할 문자열을 넣어주고, 큐에 데이터가 생기면 송신해준다.

4. I2C 통신

4-1. 구동

슬레이브에게 요청 : `Wire.requestFrom()`

읽기 : `Wire.read()`

5. IR 통신

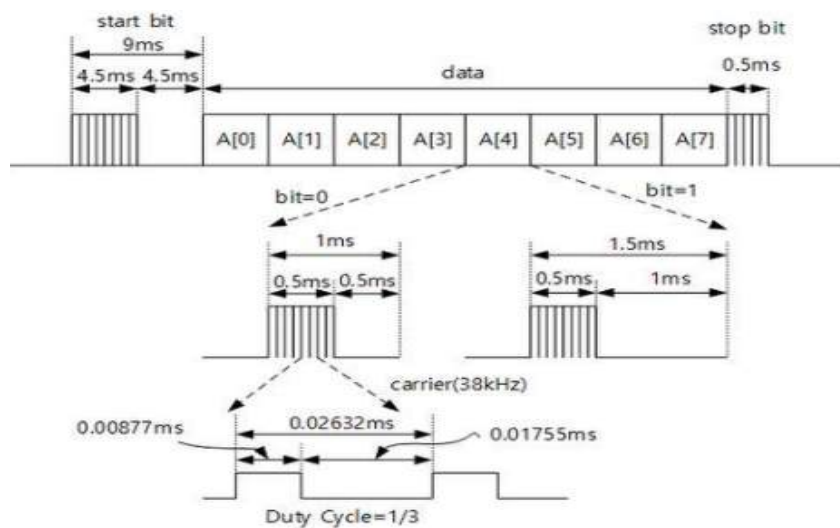
5-1. 구동

초기화 : `IR.Init()`

수신인식 : `IR.IsDta()`

수신된값 저장 : `IR.Recv()`

5-2. 구조



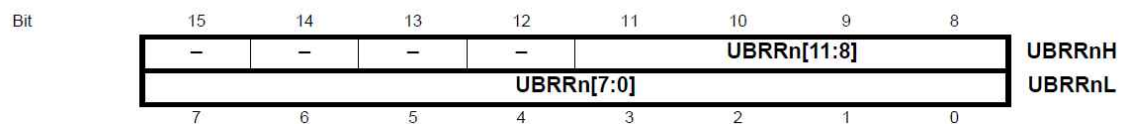
- 라이브러리: `IRSendRev.h`, `IRSendRev.cpp`, `IRSendRevInt.h`

SLAVE

1. Serial 통신

1-1. `UBRR0` – 9600bps 로 설정. $9600\text{bps} = 103$

`UBRRnL` and `UBRRnH` – USART Baud Rate Registers



1-2. `UCSR0B` – 송신만 사용. (`TXENn` 레지스터 비트 설정) 송신의 경우 Tx Complete 가 아니라 효율성을 고려하여 `UDRIEn`를 사용한다. 인터럽트를 setup에서 설정하면 무한 인터럽트를 걸 가능성이 생기므로 보낼 데이터가 있을 때 해당 인터럽트 비트를 설정, 데이터를 보내고 나면 해당 인터럽트 비트를 해제해 준다.

UCSRnB – USART Control and Status Register n B

Bit	7	6	5	4	3	2	1	0	
	RXCIE _n	TXCIE _n	UDRIE _n	RXEN _n	TXEN _n	UCSZ _{n2}	RXB8 _n	TXB8 _n	UCSR _{nB}
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R	R/W	
Initial Value	0	0	0	0	0	0	0	0	

1-3. UCSR0C – 비동기, 패리티없음, 스톱비트 1, UCSZ:110 으로 설정한다

UCSR0C = 0b00000110

UCSRnC – USART Control and Status Register n C

Bit	7	6	5	4	3	2	1	0	
	UMSEL _{n1}	UMSEL _{n0}	UPM _{n1}	UPM _{n0}	USBS _n	UCSZ _{n1}	UCSZ _{n0}	UCPOL _n	UCSR _{nC}
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	1	1	0	

1-4. 큐 - 큐를 사용하여 송신할 문자열을 넣어주고, 큐에 데이터가 생기면 송신해준다.

2. DDRB config

2-1. DDRB

- Touch 는 8번 핀에 연결하므로 DDRB &= ~(1<<0);
- PINB로 터치 인식

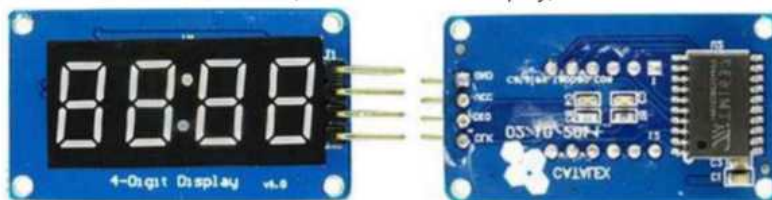
14.4.3 DDRB – The Port B Data Direction Register

Bit	7	6	5	4	3	2	1	0	
0x04 (0x24)	DDB7	DDB6	DDB5	DDB4	DDB3	DDB2	DDB1	DDB0	DDRB
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

3. 4-Digit Display

3-1. 4-Digit Display

- 4 자리의 16 진수를 표시하는 FND(Flexible Numeric Display)



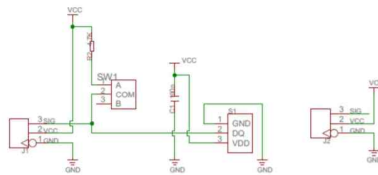
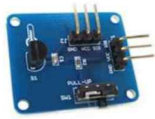
- TM1637 모듈: datasheet_TM1637.pdf
- LED (8 segment * 6 bit) control
- Keyboard (8*2bit) scan interface
- 8 단계 밝기 조절 : setBrightness()

- 2 wire interface (CLK, DIO)
- 숫자 출력 : showNumberDec(), showNumberDecEx()

4. 온도 센싱

4-1. DS18B20 온도 센서 모듈

- 온도 센서를 센싱하여 한 비트로 전송하는 모듈
- 리셋 : reset()
- 주소선택 : select()
- 변환시작, 읽기 : write()
- 값 읽기 : read_bytes



5. Timer Register

5-1. TCCR0A, TCCR0B : OC0A - Not use, OC2B - Not use, CTC, 1/64 설정

- TCCR0A = 0x01
- TCCR0B = 0x03

TCCR0A – Timer/Counter Control Register A

Bit	7	6	5	4	3	2	1	0	
0x24 (0x44)	COM0A1	COM0AU	COM0B1	COM0BU	–	–	WGM01	WGM00	TCCR0A
Read/Write	R/W	R/W	R/W	R/W	R	R	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

TCCR0B – Timer/Counter Control Register B

Bit	7	6	5	4	3	2	1	0	
0x25 (0x45)	FOC0A	FOC0B	–	–	WGM02	CS02	CS01	CS00	TCCR0B
Read/Write	W	W	R	R	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

5-2. TC0 출력을 Compare A match Interrupt Enable (OCIE0A값을 1로 설정)

- TIMSK0 = 0x02

TIMSK0 – Timer/Counter Interrupt Mask Register

Bit	7	6	5	4	3	2	1	0	
(0x8E)	–	–	–	–	–	OCIE0B	OCIE0A	TOIE0	TIMSK0
Read/Write	R	R	R	R	R	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

5-3. OC0RA = 250 으로 설정

6. I2C 통신

OCR0A – Output Compare Register A

Bit	7	6	5	4	3	2	1	0	
0x27 (0x47)	OCR0A[7:0]								OCR0A
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

슬레이브 주소 등록 : Wire.begin()

마스터 요청시 함수호출 : Wire.onRequest("your fuction")

5단계: 알고리즘 설계

[master]

- 전역 변수 설정

```
#define LED 5
#define IR_OUT_PIN 2 // IR 수신기를 D2 에 연결. D3 을 제외한 핀 사용 가능
#define QSIZE 100
```

```
volatile int msec;
uint8_t duty_cycle;
volatile int adValue;
volatile int ledStat;
uint8_t ch;
char reqstr[10];
```

```
unsigned char num_arr[10][5] = {{0,255,104,151},
{0,255,48,207},
{0,255,24,231},
{0,255,122,133},
{0,255,16,239},
{0,255,56,199},
{0,255,90,165},
{0,255,66,189},
{0,255,74,181},
{0,255,82,173}};
```

```
unsigned char dta[20];
int nflag=0;
```

- 문자열 큐 클래스

```
class CQueue {
    uint8_t queue[QSIZE];
    uint8_t head; // 0~99
    uint8_t tail;
public: CQueue() { head = tail = 0; } //큐 reset
    bool enqueue(uint8_t ch) //큐가 full이 아니면 ch를 큐에 저장
    uint8_t dequeue(void) //큐에서 데이터 제거하여 리턴
};
```

- Setup() : 주변장치 설정

```
// LED PD.5(because OCR0B output)
// initialize TC0
//serial initalize
//IR routin initalize
```

- Loop() : t또는T를 받아 slave로 요청 && IR통신후 LED조절.

```
ch = getch(); //시리얼 모니터에서 문자 받기
```

```
if(ch=='t' || ch=='T') //받은게 t|T 인지 검사
Wire.requestFrom(1,4) //1번주소의 슬레이브에 4바이트 요청
while(Wire.available()) //Wire에 데이터가 남아있다면 계속 받는다.
char c = Wire.read(); //받은 데이터 c에 저장
if(c>='0'&&c<='9'){ } //c값이 0~9 사이 값이면 해당 스코프에서 출력한다.
```

```
if(IR.IsDta()) // 데이터를 수신하였다면
for(int j=0;j<10;j++) //0~9 번까지의 버튼을 검사한다.
nflag=1; //눌린버튼을 인식하면1, 인식하지 못하면0
for(int i = 0; i<dta[D_DATALEN]; i++){ } //수신된 데이터들 검사후 알맞은 버튼이
//아니면 해당 스코프에서 nflag=0
```

```
if(nflag){ //LED 밝기 조절 } //nflag가 1이면 LED의 밝기 조절.
```

[slave]

- 전역 변수 설정

```
//Display pin 정의
#define CLK 12
#define DIO 13
#define QSIZE 100

//temperature
OneWire ds(2); // on pin 2 (a 4.7K resistor is necessary)
byte ds_addr[8];
uint8_t found;
uint8_t scratchpad[10];
int16_t raw; // raw data
float celcius; // temperature
char buf[80]; // print buffer
char cel[80];
char tmp[80]; //temperature

char reqint[80]; //온도의 정수부분을 담아 보낼 변수
int tmpup; //store tmp (온도의 정수부분)
int tmpdown; //store tmp (온도의 소수부분)
```

- 문자열 큐 클래스

```
class CQueue {
    uint8_t queue[QSIZE];
    uint8_t head; // 0~99
    uint8_t tail;
public: CQueue() { head = tail = 0; } //큐 reset
    bool enqueue(uint8_t ch) //큐가 full이 아니면 ch를 큐에 저장
    uint8_t dequeue(void) //큐에서 데이터 제거하여 리턴
};
```

- Setup() : 슬레이브 및 주변장치 초기화& 설정

```
Wire.begin(1); //슬레이브 주소
Wire.onRequest(requestEvent); //master에서 요청시 requestEvent함수 호출

UBRR0 = 103; // 9600 bpsf
UCSR0B = 0b00001000; // Data Ready Interrut Disable, Tx Enable
UCSR0C = 0b00000110; // Async, No parity, 1 stop bit, 8 data bits,
```

```
DDRB &= ~(1<<0); // Touch = PD.Bit3 Touch 는 8번 핀  
oldTouch = 0;
```

```
disp.setBrightness(1); //4digit-display 밝기 1  
disp.showNumberDec(0); //초기 디스플레이값 0  
sei();  
found = ds.search(ds_addr);
```

- void getTemp() : 온도 측정 함수

온도센서 reset,
scratchpad 읽기
raw/16 한 값을 온도값celsious로 저장
소숫점의 왼쪽을 tmpup 에 저장
소숫점 자릿수 값을 tmpdown에 저장
tmpup값을 4digit-display에 출력

- void requestEvent() : master의 요청에 응답

```
sprintf(reqint,"%d",tmpup); //char type의 reqint 에 int type의 tmpup(온도값) 저장  
Wire.write(reqint); //온도값 전달
```

- Loop ()

```
//터치 여부 판단  
//온도측정함수 호출
```


6단계: 프로그램 작성 및 디버깅

```
<Master>
#include "IRSendRev.h"
#include <Wire.h>

#define LED 5
#define IR_OUT_PIN 2 // IR 수신기를 D2 에 연결. D3 을 제외한 핀 사용 가능
#define QSIZE 100

volatile int msec;
uint8_t duty_cycle;
volatile int adValue;
volatile int ledStat;
uint8_t ch;
char reqstr[10];

unsigned char num_arr[10][5] = {{0,255,104,151},
{0,255,48,207},
{0,255,24,231},
{0,255,122,133},
{0,255,16,239},
{0,255,56,199},
{0,255,90,165},
{0,255,66,189},
{0,255,74,181},
{0,255,82,173}};

unsigned char dta[20];
int nflag=0;

class CQueue {
    uint8_t queue[QSIZE];
    uint8_t head; // 0~99
    uint8_t tail;
public: CQueue()
    { head = tail = 0; }
    bool enqueue(uint8_t ch)
    {
        uint8_t h;
        bool flag;
```

```

    h = (head + 1) % QSIZE;
    if (h == tail) // QUEUE_FULL
        flag = false; // fail, do nothing
    else {
        queue[head] = ch;
        head = h;
        flag = true; //success
    }
    return flag;
}
uint8_t dequeue(void)
{
    uint8_t ch;
    if (head == tail) //QUEUE_EMPTY
        ch = NULL;
    else {
        ch = queue[tail];
        tail = (tail+1) % QSIZE;
    }
    return ch;
}
};

```

```

struct CQueue TxQueue;
struct CQueue RxQueue;

```

```

ISR(USART_UDRE_vect)
{
    uint8_t ch;
    ch = TxQueue.dequeue();
    if (ch != NULL) {
        UDR0 = ch;
    }
    else{
        UCSRB &= 0b10011000; // Tx Data Ready Interrupt Disable
    }
}

```

```

ISR(USART_RX_vect)
{
    RxQueue.enqueue(UDR0);
}

```

```
}
```

```
void putch(uint8_t ch)
```

```
{
```

```
    cli();
```

```
    TxQueue.enqueue(ch);
```

```
    if ((UCSR0B & (1<<UDRIE0)) == 0) // Tx Data Ready Int. disable
```

```
    UCSR0B |= (1<<UDRIE0); // 0b00100000; Tx Data Ready Interrupt Enable
```

```
    sei();
```

```
}
```

```
void putstr(uint8_t *str)
```

```
{
```

```
    while (*str != NULL)
```

```
    {
```

```
        putch(*str++);
```

```
    }
```

```
}
```

```
uint8_t getch(void)
```

```
{
```

```
    cli();
```

```
    uint8_t ch = RxQueue.dequeue();
```

```
    sei();
```

```
    return ch;
```

```
}
```

```
void setup() {
```

```
    Wire.begin();
```

```
    duty_cycle = 255;
```

```
    // LED PD.5(because OCR0B output)
```

```
    DDRD |= (1 << LED); // output
```

```
    PORTD = 0; // Clear LED
```

```
    // initialize TC0
```

```
    TCCR0A = 0x23; // OC0A not used, OC2B = non-interleaving mode, xx, xx, FAST  
    PWM(CTC[1:0])
```

```
    TCCR0B = 0x0B; // xx, xx, FAST PWM(CTC[2]), 1/64
```

```
    OCR0A = 250; // Compare Match on count 250 = 1 msec
```

```
    OCR0B = duty_cycle;
```

```
    TIMSK0 = 0x02; // TC0 Output Compare A Match Interrupt Enable
```

```

//serial initialize
UBRR0 = 103; // 9600 bps
UCSR0B = 0b10011000; // RXCIE enable, Data Ready Interrut Disable, Tx Enable, RX
Enable
UCSR0C = 0b00000110; // Async, No parity, 1 stop bit, 8 data bits,
//IR routin initialize
IR.Init(IR_OUT_PIN); // IR 루틴 초기화
sei();
}
ISR(TIMERO0_COMPA_vect)
{
    msec = (msec + 1) % 500;
    if (msec == 0) {
        ledStat ^= 1;

    }
}

void loop() {

    ch = getch();

    if(ch=='t' || ch=='T'){
        Wire.requestFrom(1,4);
        while(Wire.available()){
            char c = Wire.read();
            if(c>='0'&&c<='9'){
                sprintf(reqstr,"%c",c);
                putstr(reqstr);
            }
        }
        putstr("\n");
    }

    if(IR.IsDta()) // 데이터를 수신하였다면
    {
        IR.Recv(dta);

        for(int j=0;j<10;j++){
            nflag=1;

```

```

for(int i = 0; i<dta[D_DATALEN]; i++) // payload
{
    if(dta[D_DATA+i] != num_arr[j][i]){
        nflag=0;
    }
}
if(nflag){
    duty_cycle = (j+1)*25; // min:25, max:250
}
}

if(ledStat){
    OCR0B = duty_cycle; //on LED
}
else{
    OCR0B = 0; //off LED
}
}

```

<Slaver>

```
#include <TM1637Display.h>
```

```
#include "OneWire.h"
```

```
#include <Wire.h> //master, slave 의 I2C 통신을위한 헤더
```

```
//Display pin 정의
```

```
#define CLK 12
```

```
#define DIO 13
```

```
#define QSIZE 100
```

```
//temperature
```

```
OneWire ds(2); // on pin 2 (a 4.7K resistor is necessary)
```

```
byte ds_addr[8];
```

```
uint8_t found;
```

```
uint8_t scratchpad[10];
```

```
int16_t raw; // raw data
```

```
float celcius; // temperature
```

```
char buf[80]; // print buffer
```

```
char cel[80];
```

```
char tmp[80];
```

```
char reqint[80];
```

```
int tmpup; //store tmp
```

```
int tmpdown; //store tmp
```

```
volatile int msec;
```

```
volatile int complete=0; //변환 완료
```

```
volatile int trans = 0; //변환 시작
```

```
TM1637Display disp(CLK,DIO);
```

```
uint8_t oldTouch;
```

```
uint8_t touch;
```

```
void setup() {
```

```
    Wire.begin(1); //슬레이브 주소
```

```
    Wire.onRequest(requestEvent); //요청시 requestEvent함수 호출
```

```
// Wire.onReceive(receiveEvent); //데이터 전송 받을 때 receiveEvent함수 호출
```

```
    UBRR0 = 103; // 9600 bpsf
```

```
    UCSRB = 0b00001000; // Data Ready Interrut Disable, Tx Enable
```

```
    UCSRC = 0b00000110; // Async, No parity, 1 stop bit, 8 data bits,
```

```
    DDRB &= ~(1<<0); // Touch = PD.Bit3 Touch 는 8번 핀
```

```

//DDRB |= (1<<1); // LED = PD.Bit4    LED는 9번 핀
//PORTB &= ~(1<<0); // LED OFF
oldTouch = 0;
disp.setBrightness(1);
disp.showNumberDec(0);

// initialize TC0
TCCR0A = 0x01; // OC0A not used, OC2B not used, xx, xx, (CTC[1:0])
TCCR0B = 0x03; // xx, xx,(CTC[2]), 1/64
OCR0A = 250; // Compare Match on count 250 = 1 msec
TIMSK0 = 0x02; // TC0 Output Compare A Match Interrupt Enable

sei();
found = ds.search(ds_addr);
}

ISR(TIMER0_COMPA_vect)
{
    if(trans){
        msec = (msec + 1) % 1000;
        if (msec == 0) {
            complete=1;
            trans=0;
        }
    }else{
        msec=0;
    }
}

void loop() {
    touch = (PINB & (1<<0)) >> 0; //터치 되면 getTemp 호출
    if ((oldTouch == 0) && (touch == 1)) {
        getTemp(); //온도 측정후
    } oldTouch = touch;
}

class CQueue {
    uint8_t queue[QSIZE];
    uint8_t head; // 0~99
    uint8_t tail;

```

```

public: CQueue()
    { head = tail = 0; }
bool enqueue(uint8_t ch)
{
    uint8_t h;
    bool flag;
    h = (head + 1) % QSIZE;
    if (h == tail) // QUEUE_FULL
        flag = false; // fail, do nothing
    else {
        queue[head] = ch;
        head = h;
        flag = true; //success
    }
    return flag;
}
uint8_t dequeue(void)
{
    uint8_t ch;
    if (head == tail) //QUEUE_EMPTY
        ch = NULL;
    else {
        ch = queue[tail];
        tail = (tail+ 1) % QSIZE;
    }
    return ch;
}
};

```

```

struct CQueue TxQueue;

```

```

ISR(USART_UDRE_vect)
{
    uint8_t ch;
    ch = TxQueue.dequeue();
    if (ch != NULL) {
        UDR0 = ch;
    }
    else{
        UCSRB &= 0b00001000; // Tx Data Ready Interrupt Disable
    }
}

```



```

    }
}
void putch(uint8_t ch)
{
    cli();
    TxQueue.enqueue(ch);
    if ((UCSR0B & (1<<UDRIE0)) == 0) // Tx Data Ready Int. disable
        UCSR0B |= (1<<UDRIE0); // 0b00100000; Tx Data Ready Interrupt Enable
    sei();
}
void putstr(uint8_t *str)
{
    while (*str != NULL)
    {
        putch(*str++);
    }
}
void getTemp() {
    // put your main code here, to run repeatedly:
    ds.reset();
    ds.select(ds_addr);
    ds.write(0x44, 1); // start conversion

    trans=1;
    while(!complete);
    complete = 0;

    ds.reset();
    ds.select(ds_addr);
    ds.write(0xBE); //read scratchpad
    ds.read_bytes(scratchpad, 9);

    for (int n=0; n<9; n++)
    {
        //Serial.print(scratchpad[n]);
        //Serial.print(" ");
        sprintf(tmp, "%d", scratchpad[n]);
        putstr(tmp);
        putstr(" ");
    }
}

```

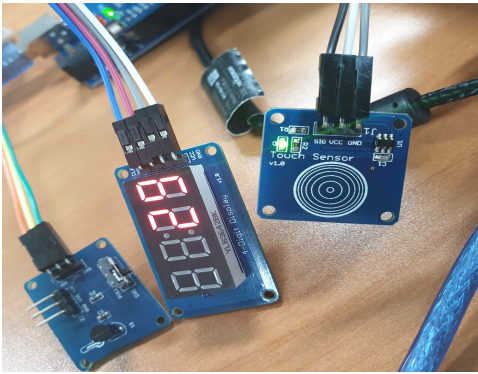

```

//Serial.println();
putstr("Wn");
raw = ((int16_t)scratchpad[1] << 8) + (int16_t)scratchpad[0];
celcius = (float)raw/16.0;
tmpup = (int)celcius;
tmpdown = ((int)(celcius*100))%100;
sprintf(buf, "Temperature: %d ", raw);
putstr(buf);
sprintf(cel, "%d.%d", tmpup, tmpdown);
disp.showNumberDecEx(tmpup); //온도 정수부분.
putstr(cel);
putstr("Wn");
}

void requestEvent() { //master가 값 요청시 호출됨
    sprintf(reqint,"%d",tmpup);
    Wire.write(reqint);
}

```

7단계: 결론

	
각종 센서	시리얼 모니터

김서기

이번 프로젝트를 진행하면서 지금까지 배웠던 것을 다시 생각할 수 있는 프로젝트였다. 인터럽트를 사용하여 터치하였을때 온도 출력, 타이머를 사용하여 0.5초마다 깜빡여 출력하고 또 얼마전에 배웠던 적외선 통신을 이용하여 리모콘으로 LED를 밝기를 조정하였다. 통신과정에서 애로사항이 많았는데 데이터를 보내면 데이터가 가지 않거나 쓰레기값이 많이 나와서 프로젝트를 진행하는데 힘들었다.

박진훈

이때 까지 사용했던 것을 하나씩 되짚어보면서 합칠려고 하니깐 힘들었다. 레지스터로 짜니 가독성이 떨어져서 내가 짰던 코드도 며칠 지나면 이해하기 어려웠다. 함수의 편리성이 간절히 생각나는 시간이었다. 특히 시리얼 송수신 과정에서 데이터형식(정수형, 문자형)의 차이 때문에 입출력에 고생을 하였다. 그래도 프로그램 완성하니 뿌듯하였다.

정민규

이번 펌웨어 프로젝트 및 펌웨어 프로그래밍 수강을 통하여 기기와 사람이 소통할 수 있는 언어 중 레지스터로 소통을 할 수 있다는 자신감을 얻었다. 특히 다 구현을 하고 마지막에 통신을 하였을 때와 delay등 코드를 레지스터로 하나하나 바꿔가는 과정등 세세한 마무리 작업등이 가장 힘들었다. master와 slaver의 통신 코드 예제가 나와 있지 않아 스스로 학습을 하여 했을 때 기분이 좋았다.

허지용

통신은 byte 단위로 하기때문에 int나 float 형식의 변수값을 주게되면 데이터가 소실 될 가능성이 있어 char 타입으로 주었다. 그런데 받는 master 에서 4바이트씩요청했더니 25?? 식으로 쓰레기값이 들어갔다. 센서로 받는 온도는 0~99 도까지 일것이라고 일반화 한다면 2바이트만 요청해도 됐겠지만 예상 못한 예외상황이 있을수 있으니 그냥 4바이트 요청하는데 신 master에서 '0'~'9'까지만 출력하도록 하였다. 이번 프로젝트를 통해 I2C에 대해 많이 알게되었다.