AVR – 초음파 센서

임베디드스쿨 2기

Lv1과정

2021. 07. 02

박태인

1. 초음파 센서(정의)

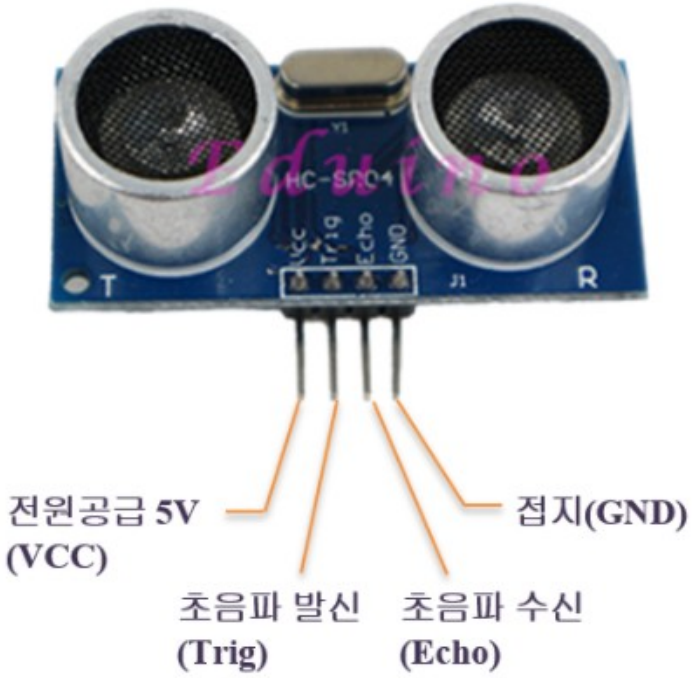
초음파 센서 란?

초음파란 사람의 귀에 들리지 않을 정도로 높은 주파수 (약 20 kHz 이상)의 소리를 말한다. 이와 같은 특성을 이용한 것이 초음파센서로 음파를 쏘아 올리고 반사되어 오는 음파까지의 **시간차를 거리로 계산**하여 측정하는 방식으로 동작된다.

ㄴ 거리 = 속력 x 시간 (여기서 속력에 음파의 속력을 대입)

이번 실습에서 사용 됐던 초음파 센서는 'HC-SR04'라는 센서이며, 기본 스펙은 아래와 같다.

동작 전압	5V
동작 전류	15mA
동작 주파수	40Hz
발생 주파수	40kHz
측정 거리	최대 4m
측정 각도	15도



1. 초음파 센서(동작원리)

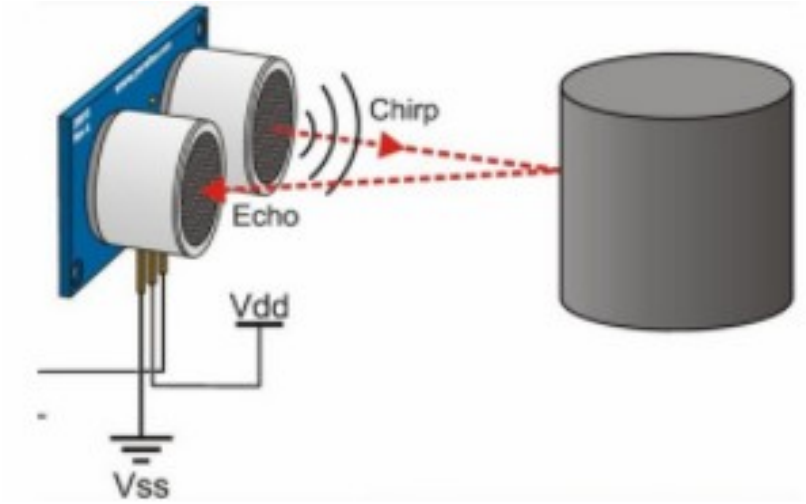
초음파 센서 동작 원리?

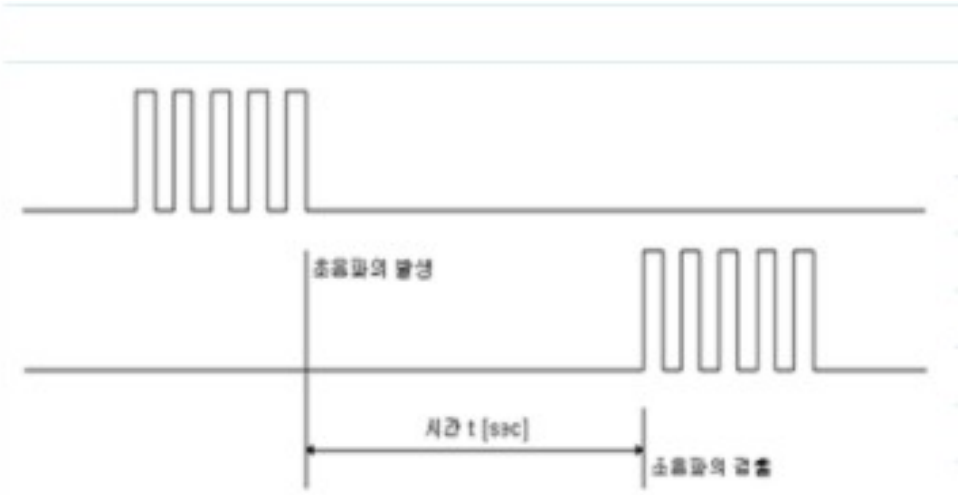
초음파 센서는 두 개의 눈으로 구성되어 있으며, 하나는 **초음파를 발생시켜 송신(Trig)**하는 기능을 수행,
나머지 하나는 **송신된 초음파의 신호를 수신(Echo)**하는 기능을 수행하게 된다.

따라서 두 개의 눈이 서로 송신하고 수신되는 시간의 차이에 따라 물체와의 거리를 계산하는 방식으로 동작하며

주파수가 높은 만큼 파장이 매우 짧기 때문에 거리 방향의 분해력이 우수하고 정밀도가 높은 계측을 할 수 있다.

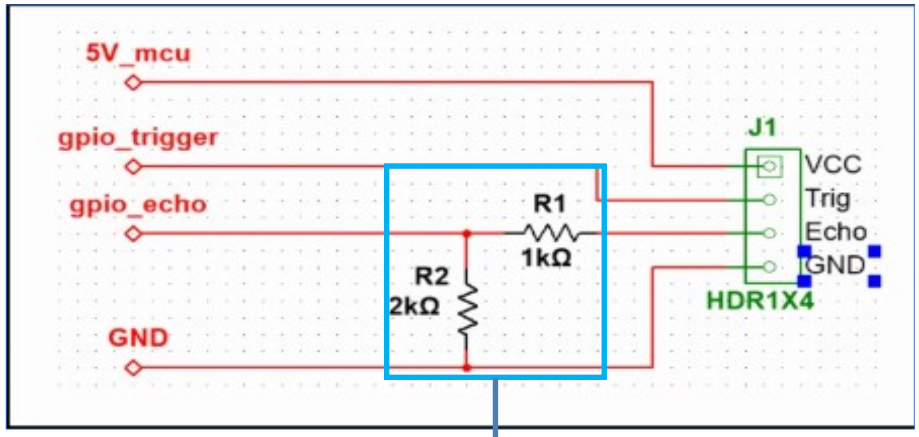
이러한 센서의 원리 때문에 거리감지센서나 레인지 센서 라고도 불린다.

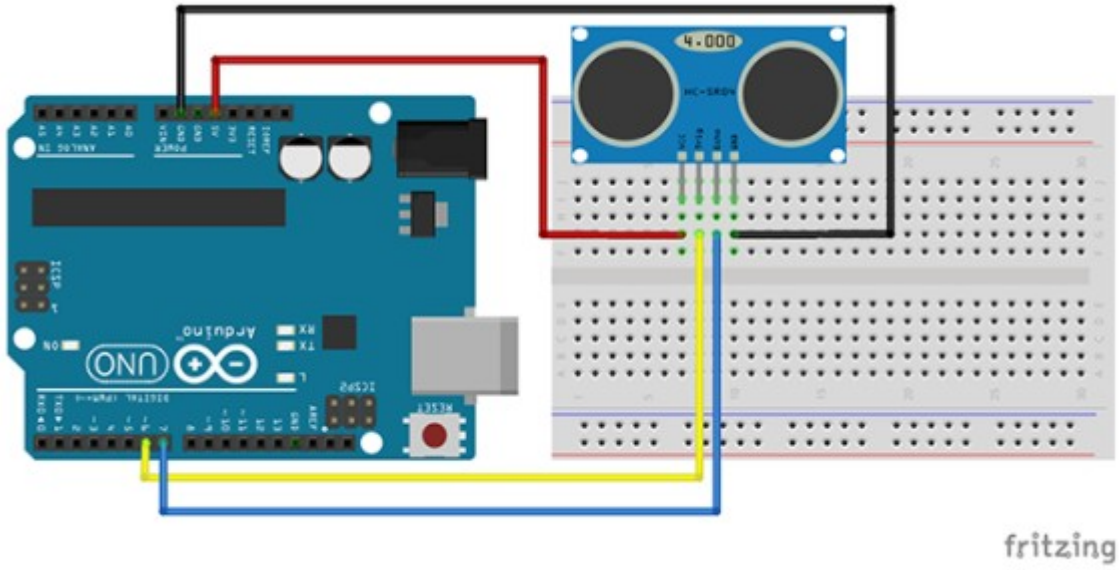




1. 초음파 센서(H/W 연결)

초음파 센서 회로 연결





실제 회로 구성은 위의 Schematic 회로도 대로
회로 결선을 하였으며,

아래 fritzing 은 참고 그림 입니다.

질문 사항)

초음파 센서 회로 구성 할 때 **echo(수신) 쪽에 저항을 사용한 이유**가 있을 까요??

답변)

신호가 입력 될 때 전력밀도가 강한 신호가 입력 될 수 있어 방어용 입니다.

2. 초음파 센서(헤더/변수 선언)

```
#include <avr/io.h>

#define F_CPU   16000000UL
#include    <avr/interrupt.h>
#include    <util/delay.h>

#include    <avr/io.h>

static volatile uint32_t    first_reading = 0;
static volatile uint32_t    second_reading = 0;
static volatile uint32_t    duty_cycle = 0;

void uart_init(void)
{
    UCSR0A |= _BV(U2X0);

    UBRR0H = 0x00;
    UBRR0L = 207;

    UCSR0B |= 0x06;

    UCSR0B |= _BV(RXEN0);
    UCSR0B |= _BV(TXEN0);
}
```

헤더 선언
- ㄴ Clock 160만
- ㄴ interrupt 사용
- ㄴ delay 사용
- ㄴ avr/io 사용

첫 수신, 두번째 수신, 시간 차(duty_cycle) 관련
Static 변수 선언

Static 변수 선언은 낯익지 않으므로 간단히 알아 보면?

ㄴ **Static 변수의 특성?**

- Static 변수는 선언된 함수 내에서만 사용이 가능하며(**지역변수 특성**)
- 단 한번만 초기화를 하며 프로그램이 종료 될 때 까지 메모리 공간에 존재 한다.(**전역변수의 특성**)

즉, 지역변수와 다른 점은 해당 함수가 종료되어도 소멸되지 않는다는 점.

이처럼 **다른 함수에서는 쓸 수 없지만,**
해당 함수내에서는 static 변수는 프로그램이 종료되기 전까지 소멸되지 않습니다.

- 뒤 페이지 계속 -

2. 초음파 센서(Static 변수)

Static 변수

<예제>

```
#include <stdio.h>

void simpleFunc(void);

int main(void) {
	int i;
	for(i = 0; i < 3; i++) {
		simpleFunc();
	}
	return 0;
}
void simpleFunc(void) {
	static int num1 = 0;
	int num2 = 0;
	num1++, num2++;
	printf("static: %d, local: %d₩n", num1, num2);
}
```

다시 한번 static 변수 특징을 정리하면

- **선언 위치는 지역변수**와 같다.
  (순수 C언어 경우 함수의 맨 처음, C++은 어디서든 가능,
  자바 파이선도 마찬가지)
- **특정 선언 지역에서만 접근 할 수 있다.**
- **메모리 공간에서 변수의 저장공간은 전역변수**와 같다.
- 초기값을 주지 않을 경우 항상 0 으로 초기화 되며,
  프로그램을 실행 시킬 때 **단 한번만 초기화** 된다.

선생님 TIP) 현재 코딩중인 Code Blocks 프로그램은 ide 차원에서 C와 C++이 모두 서포트가 되어 static 변수의 위치가 지역변수의 위치가 아니어도 Error가 발생하지 않습니다.

Code::Blocks

Code::Blocks

The free C/C++ and Fortran IDE.

하지만, domain 관점에서 바라보면 좋은 방식이 아닙니다.
결국 유지보수성을 떨어 뜨리고 생산성을 떨구는 역할을 하거든요.

그런 측면에서 C도 구조체로 객체를 만들어 제어하는 방식을
추구하는 것이 좋습니다!

2. 초음파 센서(UART 초기화 함수)

```
#include <avr/io.h>

#define F_CPU   16000000UL
#include    <avr/interrupt.h>
#include    <util/delay.h>

#include    <avr/io.h>

static volatile uint32_t    first_reading = 0;
static volatile uint32_t    second_reading = 0;
static volatile uint32_t    duty_cycle = 0;

void uart_init(void)
{
    UCSR0A |= _BV(U2X0);

    UBRR0H = 0x00;
    UBRR0L = 207;

    UCSR0B |= 0x06;

    UCSR0B |= _BV(RXEN0);
    UCSR0B |= _BV(TXEN0);
}
```

자 다시.. 코딩 해석으로 돌아와서..

이번에는 uart 초기화 함수이다.

그런에 여기서 보면 _BV라는 것을 볼 수 있다.

```
/* BV()는 무엇인가 ?

#define TW_STATUS_MASK  0b11110000

위의 코드는 아래와 동의어

#define TW_STATUS_MASK  _BV(TW7) | _BV(TW6) | _BV(TW5) | _BV(TW4)

BV에 들어간 숫자의 비트만 활성화 시킴

*/
```

즉, BV에 들어가는 해당 비트만 활성화 시킨다는 의미가 된다!

Uart는 저번 시간에 했었지만

복습 차원에서 다음페이지 부터 다시 한번 차근히 살펴보자.

2. 초음파 센서(UART 초기화 함수)

```
void uart_init(void)
{
    UCSR0A |= _BV(U2X0);

    UBRR0H = 0x00;
    UBRR0L = 207;

    UCSR0B |= 0x06;

    UCSR0B |= _BV(RXEN0);
    UCSR0B |= _BV(TXEN0);
}
```

비동기
Baud rate 9600

UCSRnA – USART Control and Status Register n A

Bit	7	6	5	4	3	2	1	0	
	RXCn	TXCn	UDREn	FEn	DORn	UPEn	U2Xn	MPCMn	UCSRnA
Read/Write	R	R/W	R	R	R	R	R/W	R/W	
Initial Value	0	0	1	0	0	0	0	0	

- **Bit 1 – U2Xn: Double the USART Transmission Speed**

This bit only has effect for the asynchronous operation. Write this bit to zero when using synchronous operation.

Writing this bit to one will reduce the divisor of the baud rate divider from 16 to 8 effectively doubling the transfer rate for asynchronous communication.

- UCSR0A 레지스터의 U2X0을 활성화 =>
  비동기 UART 방식에서 활용 되며, Double Speed 방식이다.

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	

Table 19-12. Examples of UBRRn Settings for Commonly Used Oscillator Frequencies

	f_{osc} = 16.0000MHz			
	U2Xn = 0		U2Xn = 1	
Baud Rate (bps)	UBRRn	Error	UBRRn	Error
2400	416	–0.1%	832	0.0%
4800	207	0.2%	416	–0.1%
9600	103	0.2%	207	0.2%
14.4k	68	0.6%	138	–0.1%
19.2k	51	0.2%	103	0.2%
28.8k	34	–0.8%	68	0.6%
38.4k	25	0.2%	51	0.2%
57.6k	16	2.1%	34	–0.8%
76.8k	12	0.2%	25	0.2%
115.2k	8	–3.5%	16	2.1%
230.4k	3	8.5%	8	–3.5%
250k	3	0.0%	7	0.0%
0.5M	1	0.0%	3	0.0%
1M	0	0.0%	1	0.0%
Max.(1)	1Mbps		2Mbps	

Note: 1. UBRRn = 0, error = 0.0%

자, 이제 본격적으로 UBRR0H,L을 통해 Baud Rate를 설정하게 되는데 옆의 표에서 보는바와 같이

F = 16000000, UBRRn = 207, U2Xn = 1 일 경우,

9600 Baud rate 가 된다.

(아직, UCSR0B 설정이 남아 있다.
다음 페이지에 계속)

2. 초음파 센서(UART 초기화 함수)

```
void uart_init(void)
{
    UCSR0A |= _BV(U2X0);

    UBRR0H = 0x00;
    UBRR0L = 207;

    UCSR0B |= 0x06;

    UCSR0B |= _BV(RXEN0);
    UCSR0B |= _BV(TXEN0);
}
```

의미상 Character 사이즈를 설정하고,

RX와 TX 기능을 ON 시키는 코드이다.

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	

- **Bit 2 – UCSZn2: Character Size n**

The UCSZn2 bits combined with the UCSZn1:0 bit in UCSRnC sets the number of data bits (character size) in a frame the receiver and transmitter use.

- **Bit 1 – RXB8n: Receive Data Bit 8 n**

RXB8n is the ninth data bit of the received character when operating with serial frames with nine data bits. Must be read before reading the low bits from UDRn.

RXB8n : RXB8n은 9개의 데이터 비트가 있는 직렬 프레임으로 작동 할 때 수신 된 문자의 9번째 데이터 비트 입니다. UDRn에서 하위 비트를 읽기 전에 읽어야 합니다.

- **Bit 4 – RXENn: Receiver Enable n**

Writing this bit to one enables the USART receiver. The receiver will override normal port operation for the RxDn pin when enabled. Disabling the receiver will flush the receive buffer invalidating the FEn, DORn, and UPEn flags.

- **Bit 3 – TXENn: Transmitter Enable n**

Writing this bit to one enables the USART transmitter. The transmitter will override normal port operation for the TxDn pin when enabled. The disabling of the transmitter (writing TXENn to zero) will not become effective until ongoing and pending transmissions are completed, i.e., when the transmit shift register and transmit buffer register do not contain data to be transmitted. When disabled, the transmitter will no longer override the TxDn port.

2. 초음파 센서(UART RX/TX/문자열 함수)

```c
unsigned char uart_receive (void)
{
    while(!(UCSR0A & (1 << RXC0)))
    {
        ;
    }

    return UDR0;
}
```

```c
void uart_transmit (unsigned char data)
{
    while(!(UCSR0A & (1 << UDRE0)))
    {
        ;
    }

    UDR0 = data;
}
```

```c
void uart_print_string(char *str)
{
    int i;

    for(i=0; str[i]; i++)
    {
        uart_transmit(str[i]);
    }

}
```

- uart_receive 함수

! UCSRnA – USART Control and Status Register n A

Bit	7	6	5	4	3	2	1	0	
	RXCn	TXCn	UDREn	FEn	DORn	UPEn	U2Xn	MPCMn	UCSRnA
Read/Write	R	R/W	R	R	R	R	R/W	R/W	
Initial Value	0	0	1	0	0	0	0	0	

- **Bit 7 – RXCn: USART Receive Complete**

This flag bit is set when there are unread data in the receive buffer and cleared when the receive buffer is empty (i.e., does not contain any unread data). If the receiver is disabled, the receive buffer will be flushed and consequently the RXCn bit will become zero. The RXCn flag can be used to generate a receive complete interrupt (see description of the RXCIEn bit).

RX 동작 flag는 버퍼안에 읽어 들이는 값이 있으면 '1' 없으면 '0'이다.

이를 활용하여 while문을 통해 읽을 값이 있을 때만 while문에서의 무한루프를

벗어나게 하여 **return UDR0 을 해서 수신 값을 읽어 들이는 것**이다.

[While 문 동작은 RXC0이 1일 때, 조건 문 안의 값이 반전(!)되어 0 이 되어 빠져 나간다.]

- **Bit 5 – UDREn: USART Data Register Empty**

The UDREn flag indicates if the transmit buffer (UDRn) is ready to receive new data. If UDREn is one, the buffer is empty, and therefore ready to be written. The UDREn flag can generate a data register empty interrupt (see description of the UDRIEn bit). UDREn is set after a reset to indicate that the transmitter is ready.

TX 동작 시 UDR 버퍼에 데이터를 쓰기 위해서 UDR 데이터가 비어 있는 상태인지 확인해야 한다.

UDREn flag가 '1'일 경우 버퍼가 비어 있으며, 사용 할 준비가 되었음을 의미 한다.

준비가 되었을 때 UDR 버퍼에 data를 싣는다.

[While 문 동작은 UDREn이 1일 때, 조건 문 안의 값이 반전(!)되어 0 이 되어 빠져 나간다.]

2. 초음파 센서(UART RX/TX/문자열 함수)

```
unsigned char uart_receive (void)
{
    while(!(UCSR0A & (1 << RXC0)))
    {
        ;
    }

    return UDR0;
}

void uart_transmit (unsigned char data)
{
    while(!(UCSR0A & (1 << UDRE0)))
    {
        ;
    }

    UDR0 = data;
}
```

```
void uart_print_string(char *str)
{
    int i;

    for(i=0; str[i]; i++)
    {
        uart_transmit(str[i]);
    }

}
```

- **Uart_print_string 함수**

ㄴ 문자열 출력 함수

ㄴ transmit 함수의 한 글자씩 출력하는
함수를 활용하여 문자열 특징인 마지막 널 까지를 출력한다.

문자열(배열)을 인자로 받기 위해 char* 형태로 인자를 받고,

Transmit 함수를 통해 한글자 씩 출력한다.

다음 문자열이 들어 왔을 때는

새로이 for문을 돌기 때문에 i는 다시 0 부터 시작 된다.
-> for문 안에서 int i=0; 이런식으로 바로 선언해 줘도 현재 사용하고 있는 CodeBolcks에서는 Error는 없다.(C++ 적용)

함수 사용법

```
while(1)
{
    distance = get_distance();

    uart_print_string("distance(cm) = ");
    uart_print_8bit_num(distance);
    uart_transmit('\n');

    _delay_ms(1000);
}
```

2. 초음파 센서(UART RX/TX/문자열 함수)

```
void uart_print_8bit_num(uint8_t no)
{
    char num_string[4] = "0";   // 상수화 된 0
    int i, index =0;

    if(no >0)
    {
        for(i=0; no !=0; i++)
        {
            num_string[i] = no % 10 + '0';  // 아스키 0
            no = no / 10;
        }

        num_string[i] = '\0'; // NULL 문자
        index = i - 1;
    }

    for(i = index; i >= 0; i--)
    {
        uart_transmit(num_string[i]);
    }
}
```

- **Uart_print_8bit_num (거리 값 출력 함수)**

Char형 num string 배열[4]를 초기화.
변수 I, index 초기화.

인자 값 no가 0 보다 큰 값인 경우

For문을 돌려
Num_string[i] 배열에 no의 한자리씩 입력
(+'0'을 하는 이유는 숫자 값을 아스키코드화 하기 위함)

No 값이 0 이 아닐 때 까지 한다(no 값이 0이 될 때 까지)

No = no / 10을 한다..
[잘 이해 되지 않아 아래와 같이 손으로 직접 쓰면서 해보았습니다]

ㄴ 다음 페이지!

2. 초음파 센서(UART RX/TX/문자열 함수)

if) no = 127의 숫자가 들어왔다.

```
for ( i=0 ; no !=0 ; i++)
    num_string[0] = no%10
    no = no/10 = 12
```

i = 1이 되고

```
num_string[1] = no%10
no = no/10 = 1
```

i = 2 가 되고

```
num_string[2] = no%10
no = no/10 = 0
```

i = 3이 되고,

조건문 비교를 했더니 어? no가 0이네?

for문 빠져 나옴.

```
num_string[3] = '\0'
index = i - 1;
```

↳ 2.

자 그럼 여기 까지의 num_string의 상황은?

[3]	[2]	[1]	[0]
'\0'	1	2	1

```
for문 ( i = index ; i >=0 ; i--)
{
    transmit (num_string[i])
}
```

이니까.

② index = 2, num_string[2] 출력 (1출력)

i--, index = 1

num_string[1] 출력 (2출력)

i--, index = 0

num_string[0] 출력 (1출력)

i--, index = -1

↳ 조건문 탈출.

for문 빠져 나옴 ∴ 127을 차례로 출력하게 됨.

2. 초음파 센서(센서 동작 코드)

```
void HCSR04_init(void)  // 초음파 센서
{
    // 기존 인터럽트들을 클리어 시킴
    cli();
    // PD6을 출력 핀으로 설정함
    DDRD |= (1 << DDD6);
    // prescaler = 8
    // 16000000 / 8 = 2000000 => 2MHz
    // f <=> 1/T ===> 2000000 <=> 1 / 2000000 => 5 / 10000000 => 0.5us = 500ns
    // set OC0A on compare math
    TCCR0A = (1 << COM0A1) | (1 << COM0A0) | (1 << WGM01) | (1 << WGM00); // Fast PWM mode
    TCCR0B = (1 << CS01);
    // 따라서 0.5 * 236 은 118 : 아래 OCR0A가 235가 된 이유.
    // 전체 0.5 * 256 = 128us 이므로 꺼져 있는게 118 이므로 pulse가 10 인 것이다!
    // 10us trigger pulse, 118us off-time (total 128us)
    OCR0A = 235;
    // 결국 PD6이 trigger로 설정됨

    //PB0을 입력 핀으로 설정
    DDRB &= ~(1 << DDB0);
    // 잡음 차단(필터, 노이즈 캔슬링) + 엣지 검출, prescaler = 8, 입력 캡쳐
    // 20 MHz(8분주비) 를 캡쳐 하는데 10us 듀티 샘플링을 한다.
    // 초음파 센서도 20 MHz이다.
    TCCR1B = (1 << ICNC1) | (1 << ICES1) | (1 << CS11);
    // 입력 캡쳐 인터럽트 활성화
    sei();

    // Timer 1 을 입력 캡쳐 인터럽트로 활용
    TIMSK1 |= (1 << ICIE1);
    // PB0이 echo로 설정됨
}
```

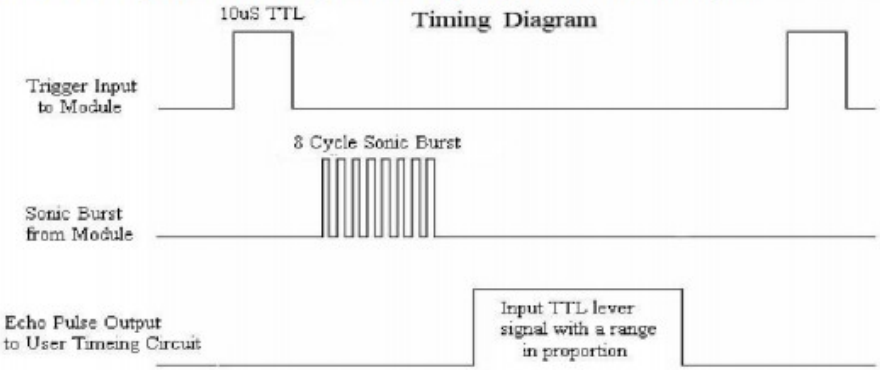
- 초음파 신호 동작을 인터럽트 동작으로 활용 (Counter_interrupt)
  - ㄴ TCCR0A , TCCR0B 설정으로 PWM 파형
  - ㄴ OCR0A 로 인터럽트 시간 설정
  - ㄴ echo의 경우 trigger와는 다른 Timer 설정(TCC1B)

(뒤 페이지 부터 상세히 분석)

10uS 신호를 trigger!
10uS 신호를 만들어야 한다.

Timing diagram

The Timing diagram is shown below. You only need to supply a short 10uS pulse to the trigger input to start the ranging, and then the module will send out an 8 cycle burst of ultrasound at 40 kHz and raise its echo. The Echo is a distance object that is pulse width and the range in proportion .You can calculate the range through the time interval between sending trigger signal and receiving echo signal. Formula: uS / 58 = centimeters or uS / 148 =inch; or: the range = high level time * velocity (340M/S) / 2; we suggest to use over 60ms measurement cycle, in order to prevent trigger signal to the echo signal.



2. 초음파 센서(센서 동작 초기화 코드)

```
void HCSR04_init(void)  // 초음파 센서
{
    // 기존 인터럽트들을 클리어 시킴
    cli();
    // PD6을 출력 핀으로 설정함
    DDRD |= (1 << DDD6);
    // prescaler = 8
    // 16000000 / 8 = 2000000 => 2MHz
    // f <=> 1/T ===> 2000000 <=> 1 / 2000000 => 5 / 10000000 => 0.5us = 500ns
    // set OC0A on compare math
    TCCR0A = (1 << COM0A1) | (1 << COM0A0) | (1 << WGM01) | (1 << WGM00); // Fast PWM mode
    TCCR0B = (1 << CS01);
    // 따라서 0.5 * 236 은 118 : 아래 OCR0A가 235가 된 이유.
    // 전체 0.5 * 256 = 128us 이므로 꺼져 있는게 118 이므로 pulse가 10 인 것이다!
    // 10us trigger pulse, 118us off-time (total 128us)
    OCR0A = 235;
    // 결국 PD6이 trigger로 설정됨
```

트리거 동작
10uS
펄스 만들기

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	

CS02	CS01	CS00	Description
0	0	0	No clock source (Timer/Counter stopped)
0	0	1	$clk_{I/O}$/(no prescaling)
0	1	0	$clk_{I/O}$/8 (from prescaler)
0	1	1	$clk_{I/O}$/64 (from prescaler)

ㄴ 8 분주비(prescaler)

Mode	WGM02	WGM01	WGM00	Timer/Counter Mode of Operation	TOP	Update of OCRx at	TOV Flag Set on(1)(2)
0	0	0	0	Normal	0xFF	Immediate	MAX
1	0	0	1	PWM, phase correct	0xFF	TOP	BOTTOM
2	0	1	0	CTC	OCRA	Immediate	MAX
3	0	1	1	Fast PWM	0xFF	BOTTOM	MAX
4	1	0	0	Reserved	–	–	–
5	1	0	1	PWM, phase correct	OCRA	TOP	BOTTOM
6	1	1	0	Reserved	–	–	–
7	1	1	1	Fast PWM	OCRA	BOTTOM	TOP

Fast PWM 방식
ㄴ TOP, 0xFF
ㄴ BOTTOM은 OC0A 값.

TCCR0A – Timer/Counter Control Register A

Bit	7	6	5	4	3	2	1	0	
0x24 (0x44)	COM0A1	COM0A0	COM0B1	COM0B0	–	–	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	

Table 14-2. Compare Output Mode, non-PWM Mode

COM0A1	COM0A0	Description
0	0	Normal port operation, OC0A disconnected.
0	1	Toggle OC0A on compare match
1	0	Clear OC0A on compare match
1	1	Set OC0A on compare match

OC0A 비교 방식(Set)
ㄴ 인터럽트 동작 된 기간 동안만 Set
나머지 off(clear)

총 해석)

16000000 clock에 8 분주비 => 16000000 / 8 = 2000000 = 2MHz
F(주파수) = 1/T(주기)
ㄴ 따라서 2Mhz의 클럭의 주기는 1/2000000
=> 0.0000005(**0.5us**) = 500 ns
따라서,
총 high의 길이가 **0.5(us) * 256(TOP) = 128us**가 될 수 있는데,
여기서 118us를 off 시키면 나머지 10us의 펄스가 on 된다.

그래서 off 시간을 118us로 만들어야 하는데,
0.5 * 236 = 118 의 값이 나온다.
OCR0A는 0부터 카운터 하므로
이 값은 235가 되는 것이다.

2. 초음파 센서(센서 동작 초기화 코드)

```
//PB0을 입력 핀으로 설정
DDRB &= ~(1 << DDB0);
// 잡음 차단(필터, 노이즈 캔슬링) + 엣지 검출, prescaler = 8, 입력 캡쳐
// 20 MHz(8분주비) 를 캡쳐 하는데 10us 듀티 샘플링을 한다.
// 초음파 센서도 20 MHz이다.
TCCR1B = (1 << ICNC1) | (1 << ICES1) | (1 << CS11);
// 입력 캡쳐 인터럽트 활성화
sei();

// Timer 1 을 입력 캡쳐 인터럽트로 활용
TIMSK1 |= (1 << ICIE1);
// PB0이 echo로 설정됨
}
```

수신 Echo 부분 설정

TCCR1B – Timer/Counter1 Control Register B

Bit	7	6	5	4	3	2	1	0	
(0x81)	ICNC1	ICES1	–	WGM13	WGM12	CS12	CS11	CS10	TCCR1B
Read/Write	R/W	R/W	R	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

입력 노이즈 캔슬

- **Bit 7 – ICNC1: Input Capture Noise Canceler**

Setting this bit (to one) activates the input capture noise canceler. When the noise canceler is activated, the input from the input capture pin (ICP1) is filtered. The filter function requires four successive equal valued samples of the ICP1 pin for changing its output. The input capture is therefore delayed by four oscillator cycles when the noise canceler is enabled.

- **Bit 6 – ICES1: Input Capture Edge Select**

This bit selects which edge on the input capture pin (ICP1) that is used to trigger a capture event. When the ICES1 bit is written to zero, a falling (negative) edge is used as trigger, and when the ICES1 bit is written to one, a rising (positive) edge will trigger the capture.

캡쳐 이벤트 트리거 하는데 사용되는 입력 캡쳐,
상승 에지 캡처 트리거

Table 15-6. Clock Select Bit Description

CS12	CS11	CS10	Description
0	0	0	No clock source (Timer/Counter stopped).
0	0	1	$clk_{I/O}$/1 (no prescaling)
0	1	0	$clk_{I/O}$/8 (from prescaler)

8 분주 비

TIMSK1 – Timer/Counter1 Interrupt Mask Register

Bit	7	6	5	4	3	2	1	0	
(0x6F)	–	–	ICIE1	–	–	OCIE1B	OCIE1A	TOIE1	TIMSK1
Read/Write	R	R	R/W	R	R	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

- **Bit 5 – ICIE1: Timer/Counter1, Input Capture Interrupt Enable** Timer1을 입력 캡쳐 인터럽트로 활용

When this bit is written to one, and the I-flag in the status register is set (interrupts globally enabled), the Timer/Counter1 input capture interrupt is enabled. The corresponding interrupt vector (see Section 11. "Interrupts" on page 49) is executed when the ICF1 flag, located in TIFR1, is set.

뒤에 나올 interrupt vect 이벤트
ㄴ capture event
ㄴ 뒤 페이지 부터
수신 인터럽트 시 동작 코드

Interrupt Vectors in ATmega328P

Table 11-1. Reset and Interrupt Vectors in ATmega328P

Vector No.	Program Address	Source	Interrupt Definition
1	0x0000	RESET	External pin, power-on reset, brown-out reset and watchdog system reset
2	0x002	INT0	External interrupt request 0
3	0x0004	INT1	External interrupt request 1
4	0x0006	PCINT0	Pin change interrupt request 0
5	0x0008	PCINT1	Pin change interrupt request 1
6	0x000A	PCINT2	Pin change interrupt request 2
7	0x000C	WDT	Watchdog time-out interrupt
8	0x000E	TIMER2 COMPA	Timer/Counter2 compare match A
9	0x0010	TIMER2 COMPB	Timer/Counter2 compare match B
10	0x0012	TIMER2 OVF	Timer/Counter2 overflow
11	0x0014	TIMER1 CAPT	Timer/Counter1 capture event
12	0x0016	TIMER1 COMPA	Timer/Counter1 compare match A
13	0x0018	TIMER1 COMPB	Timer/Counter1 compare match B

2. 초음파 센서(센서 수신 동작 코드)

```
ISR(TIMER1_CAPT_vect)
{
    /* 신호가 뜰 때 잡고*/
    if((TCCR1B & (1 << ICES1)) == (1 << ICES1))
    {
        first_reading = ICR1;
    }
    /* 신호가 가라 앉을 때 잡는다 */
    else
    {
        second_reading = ICR1;
    }

    if(first_reading != 0 && second_reading != 0)    // 떠 있는 시간 = duty_cycle
    {
        duty_cycle = second_reading - first_reading;
        first_reading = 0;
        second_reading = 0;
    }

    //엣지 검출 비트 토글(XOR 계산)
    TCCR1B ^= (1<< ICES1);
    // 캡쳐 플래그 클리어
    TIFR1 = (1<< ICF1);
}
```

TCCR1B의 ICES1 (입력 캡쳐 flag가) 1 이면 => 상승 엣지 캡쳐

First_reading (static volatile 변수) 에 'ICR1'값을 넣는다. Why??

- **Bit 6 – ICES1: Input Capture Edge Select**

This bit selects which edge on the input capture pin (ICP1) that is used to trigger a capture event. When the ICES1 bit is written to zero, a falling (negative) edge is used as trigger, and when the ICES1 bit is written to one, a rising (positive) edge will trigger the capture.

When a capture is triggered according to the ICES1 setting, the counter value is copied into the input capture register (ICR1). The event will also set the input capture flag (ICF1), and this can be used to cause an input capture interrupt, if this interrupt is enabled.

When the ICR1 is used as TOP value (see description of the WGM13:0 bits located in the TCCR1A and the TCCR1B register), the ICP1 is disconnected and consequently the input capture function is disabled.

ICES1의 setting 에 따라서 입력 엣지를 캡쳐 했을 때, counter 값을 ICR1 에 복사 한다.

즉, echo에 신호가 들어와 상승 엣지 발생 시 Timer1의 카운터 되고 있는 숫자를 카운트 수를 first_reading 변수에 입력한다.

그리고, first_reading 카운트 값이 0 이 아닐 때 And second_readin이 0이 아닐때 if분으로 들어가는데, 일단 아직 second는 0 이므로 아래로 내려 간다.

TCC1B 를 XOR 계산 해주어 이번에는 하강 엣지를 검출한다.
(**계산 된 거리만큼의 펄스가 동작한다**)

TIFR1 – Timer/Counter1 Interrupt Flag Register

Bit	7	6	5	4	3	2	1	0	
0x16 (0x36)	–	–	ICF1	–	–	OCF1B	OCF1A	TOV1	TIFR1
Read/Write	R	R	R/W	R	R	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

하강 엣지 검출 후 second_readin에도 count 값이 입력되고, if문으로 들어가 두 값의 차이가 duty_cycle 이라는 변수로 들어가게 되는데 이 값이 곧 추후에 거리 값 계산에 사용된다.

- **Bit 5 – ICF1: Timer/Counter1, Input Capture Flag**

This flag is set when a capture event occurs on the ICP1 pin. When the input capture register (ICR1) is set by the WGM13:0 to be used as the TOP value, the ICF1 flag is set when the counter reaches the TOP value.

ICF1 is automatically cleared when the input capture interrupt vector is executed. Alternatively, ICF1 can be cleared by writing a logic one to its bit location.

- TOP은 0xFF ??
- Timer Count가 TOP에 도달하면 자동적으로 capture flag가 클리어 되는듯 하나 ICF1에 1 값 넣어서 capture flag를 클리어 시켰다.

그리고, 다시 또 이러한 동작들을 반복한다.

2. 초음파 센서(거리 계산 코드)

```
uint32_t get_distance(void)
{
    static uint32_t echo_pulse_us;
    static uint32_t distance_cm;

    //32768us = 65536 clock 틱
    // 이것은 결국 timer 1에 prescaler = 8
    echo_pulse_us = (float) duty_cycle * 32768 / 65536;  // 사실 1/2 이나 마찬 가지이지만 표현하기 위함
    distance_cm = echo_pulse_us * 0.034 / 2;     // datasheet 참고(0.034/2)
    // 0.034 인 이유는 음속의 디폴트 값이 340 m/s에 해당하기 때문 : 이 속도가 마하 1
    // dx / dt = v 이거의 대한 변화 값은 속도의 개념도 된다.
    /*
      - 멀어지는 알고 리즘

      첫 번째 샘플 : 50 cm
      두번째 샘플 : 70 cm
      샘플링 주파수 = 100 hz
      속도 = (70-50) * 100 = 2000 cm/s = 20m/s // *100의 이유는 1/100의 또 역수이기 때문 주기니까

      - 다가오는 알고리즘
      첫 번째 샘플 : 70 cm
      두 번째 샘플 : 50 cm
      속도 = (50-70) * 100 = -2000 cm/s = 20m/s
      */
    return distance_cm;
}
```

TIMER1은 65535 clock counter 라고 하면,
Timer1도 분주비 8 을 했었기에

16000000 / 8 = 2000000 = 2MHz

1 / 2000000 = 0.5us. 따라서, 65536 x 0.5(us) = 32768(us)

Timing diagram

The Timing diagram is shown below. You only need to supply a short 10uS pulse to the trigger input to start the ranging, and then the module will send out an 8 cycle burst of ultrasound at 40 kHz and raise its echo. The Echo is a distance object that is pulse width and the range in proportion .You can calculate the range through the time interval between sending trigger signal and receiving echo signal. Formula: uS / 58 = centimeters or uS / 148 =inch; or: the range = high level time * velocity (340M/S) / 2; we suggest to use over 60ms measurement cycle, in order to prevent trigger signal to the echo signal.

$$\text{카운트} \times \frac{\text{us}}{\text{카운트}} = \text{카운트 끼리 약분 되고, us 결과 값 도출}$$

거리 / 시간 개념으로 속도의 계산 방법을 할 수도 있다.

2. 초음파 센서(Main)

```c
int main(void)
{
    uint32_t distance;

    // Insert code

    uart_init();
    HCSR04_init();


    DDRC |= 0x01;
    DDRC &= ~0xFD;

    /* 디버깅용 LED ON */
    PORTC = 0x01;

    while(1)
    {
        distance = get_distance();

        uart_print_string("distance(cm) = ");
        uart_print_8bit_num(distance);
        uart_transmit('\n');

        _delay_ms(1000);
    }

    return 0;
}
```

앞 선 uart_init() 초기화.
HCSR04 초음파 센서 기능 초기화

디버깅용 LED 설정

Distance 변수에 거리 값을 획득 후

문자열 출력 함수로 “distance(CM) = “출력

계산된 거리 값을 받아 출력

Uart 터미널에서 다음 줄로 넘어감.

1 초 마다 코딩 동작!

3. 프로젝트

ORCAD 회로 그리기

ㄴ 아직 작업 진행 중 입니다..

질문

```c
void uart_print_string(char *str)
{
    int i;

    for(i=0; str[i]; i++)
    {
        uart_transmit(str[i]);
    }

}
```

질문1)

여기서 str[i] 라는 조건문은

Str[i] 값이 NULL이 아닐 때 까지 라고
이해하면 되는 걸까요??

질문2)

초음파 신호의 주파수가 20Mhz
라는 것은 어떻게 알 수 있는 것인가요?

그냥 일반적으로 초음파 주파수가 20MHz 일까요
아니면 여기서 사용하는 센서의 특징 일까요?

데이터 시트 상으로 40 kHz의 ultrasound가
나오는 것 같아서 이것과는 다른 의미인지
궁금합니다.

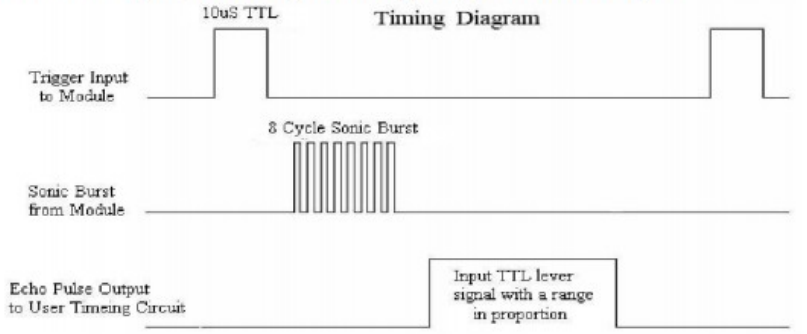
```c
    //PB0을 입력 핀으로 설정
    DDRB &= ~(1 << DDB0);
    // 잡음 차단(필터, 노이즈 캔슬링) + 엣지 검출, prescaler = 8, 입력 캡쳐
    // 20 MHz(8분주비) 를 캡쳐 하는데 10us 듀티 샘플링을 한다.
    // 초음파 센서도 20 MHz이다.
    TCCR1B = (1 << ICNC1) | (1 << ICES1) | (1 << CS11);
    // 입력 캡쳐 인터럽트 활성화
    sei();

    // Timer 1 을 입력 캡쳐 인터럽트로 활용
    TIMSK1 |= (1 << ICIE1);
    // PB0이 echo로 설정됨
}
```

질문3)

말의 의미를 잘 모르겠습니다.ㅠㅠ

Timing diagram

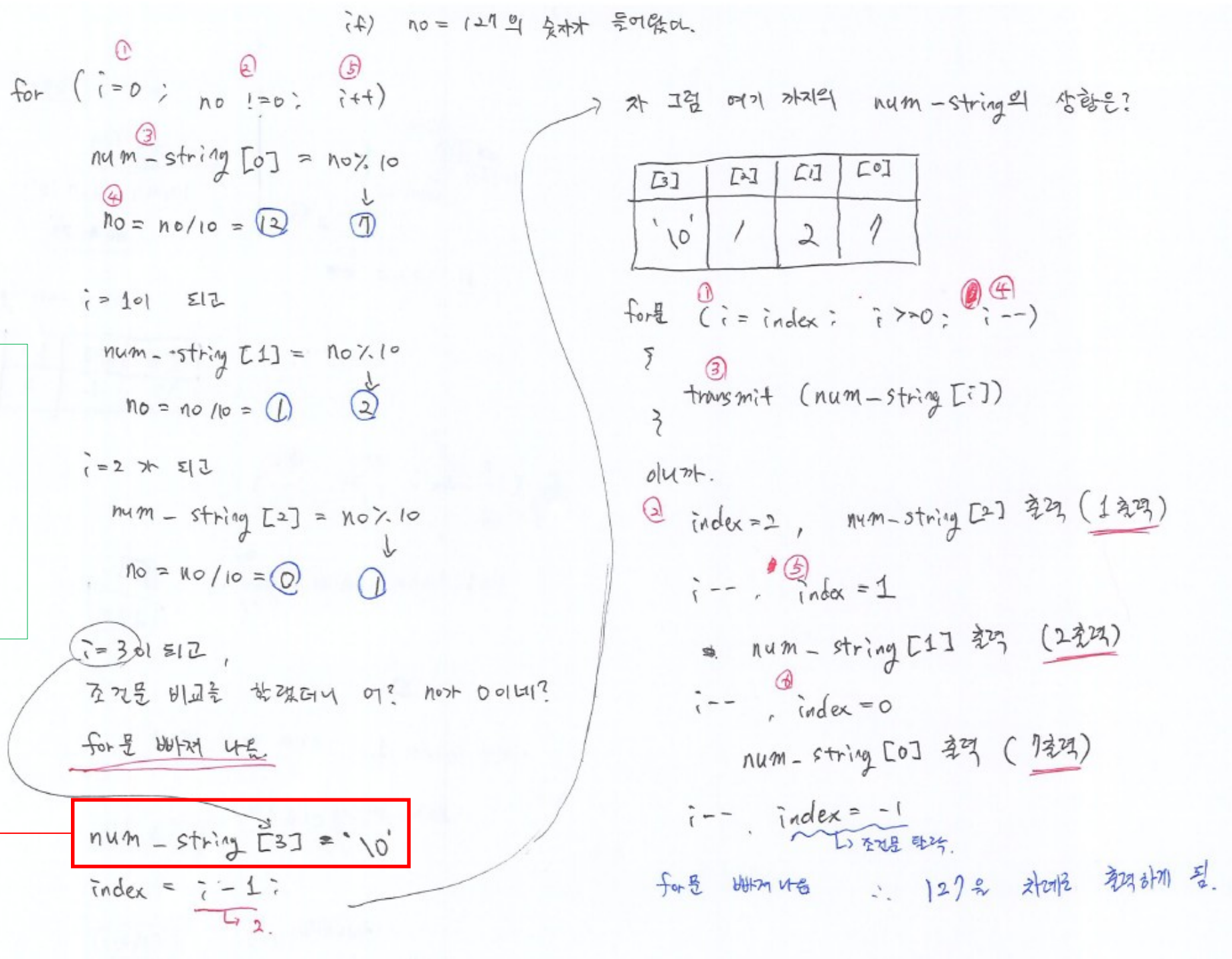
The Timing diagram is shown below. You only need to supply a short 10uS pulse to the trigger input to start the ranging, and then the module will send out an 8 cycle burst of ultrasound at 40 kHz and raise its echo. The Echo is a distance object that is pulse width and the range in proportion .You can calculate the range through the time interval between sending trigger signal and receiving echo signal. Formula: uS / 58 = centimeters or uS / 148 =inch; or: the range = high level time * velocity (340M/S) / 2; we suggest to use over 60ms measurement cycle, in order to prevent trigger signal to the echo signal.



질문

질문4)

NULL 문자를
배열의 제일
마지막에 넣고
출력을 하진 않는
것으로 보이는데..
NULL을 어떻게 활용
코드해석에서 틀린
부분이 있는 걸까
요??



질문

```
void uart_init(void)
{
    UCSR0A |= _BV(U2X0);

    UBRR0H = 0x00;
    UBRR0L = 207;

    UCSR0B |= 0x06;

    UCSR0B |= _BV(RXEN0);
    UCSR0B |= _BV(TXEN0);
}
```

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	

Table 19-7. UCSZn Bits Settings

UCSZn2	UCSZn1	UCSZn0	Character Size
0	0	0	5-bit
0	0	1	6-bit
0	1	0	7-bit
0	1	1	8-bit
1	0	0	Reserved
1	0	1	Reserved
1	1	0	Reserved
1	1	1	9-bit

질문5)

위에서 캐릭터 Size 부분이 Reserved로 해석되는데, 이것은 어떤 의미 일까요??

답변)

0x03;이 되어야 하고
Character size는 8bit 입니다.

```
// 이것은 결국 timer 1에 prescaler = 8
echo_pulse_us = (float) duty_cycle * 32768 / 65536;  // 사실 1/2 이나 마찬 가지이지만 표현하기 위함
distance_cm = echo_pulse_us * 0.034 / 2;     // datasheet 참고(0.034/2)
// 0.034 인 이유는 음속의 디폴트 값이 340 m/s에 해당하기 때문 : 이 속도가 마하 1
// dx / dt = v 이거의 대한 변화 값은 속도의 개념도 된다.
/*
```

질문6)

340M 를 계산 할 때는 mm 단위로 환산해야 하는 이유가 무엇 일까요??

그리고 TIMER1 은 TOP 값이 FFFF 인가요?

```
ISR(TIMER1_CAPT_vect)
{
    /* 신호가 뜰 때 잡고*/
    if((TCCR1B & (1 << ICES1)) == (1 << ICES1))
    {
        first_reading = ICR1;
    }
    /* 신호가 가라 앉을 때 잡는다 */
    else
    {
        second_reading = ICR1;
    }

    if(first_reading != 0 && second_reading != 0)    // 떠 있는 시간 = duty_cycle
    {
        duty_cycle = second_reading - first_reading;
        first_reading = 0;
        second_reading = 0;
    }

    //엣지 검출 비트 토글(XOR 계산)
    TCCR1B ^= (1<< ICES1);
    // 캡쳐 플래그 클리어
    TIFR1 = (1<< ICF1);
}
```

질문 7)

ICR1 값은 TIMER1에 interrupt 동작 시
현재의 count 값이 들어 가는 것으로 이해 했습니다.
그렇다면 만약 우연 하게라도 interrupt 동작 되었을 때, 상승 엣지 검출시 count가'0'값이 검출되어 아래 if문으로 동작하지 않을 수도 있지
않을까요??

질문 8)

캡쳐 플래그를 클리어를 하강 엣지 검출하기도 전에 클리어 하게 되는데..
(이 클리어가 정확히 어떤 의미의 클리어 인지
명확하게 이해가 잘 되질 않는 것 같습니다..)

그 이유가 궁금합니다!