임베디드시스템설계

Embedded System Capstone Design

ICE3015-001

과제 2

실습팀 4

정보통신공학과 12181855 황용하

정보통신공학과 12191720 곽재현

정보통신공학과 12191765 박승재

Introduction

analogRead(pin)

```
// cores/arduino/wiring_analog.c
uint32_t analogRead(uint32_t ulPin)
{
  uint32_t ulValue = 0;
  uint32_t ulChannel;

  if (ulPin < A0)
    ulPin += A0;
  ulChannel = g_APinDescription[ulPin].ulADCChannelNumber ;
    // ...

#if defined __SAM3X8E__ || defined __SAM3X8H__
    static uint32_t latestSelectedChannel = -1;
    switch ( g_APinDescription[ulPin].ulAnalogChannel )
    {
        // Handling ADC 12 bits channels
        case ADC0 :
        case ADC1 :
        case ADC2 :
        case ADC3 :
        case ADC4 :
        case ADC5 :
        case ADC6 :
        case ADC7 :
        case ADC8 :
        case ADC9 :
        case ADC10 :
        case ADC11 :
            // Enable the corresponding channel
            if (adc_get_channel_status(ADC, ulChannel) != 1) {
                adc_enable_channel( ADC, ulChannel );
                if ( latestSelectedChannel != (uint32_t)-1 && ulChannel !=
latestSelectedChannel)
                    adc_disable_channel( ADC, latestSelectedChannel );
                latestSelectedChannel = ulChannel;
                g_pinStatus[ulPin] = (g_pinStatus[ulPin] & 0xF0) |
PIN_STATUS_ANALOG;
            }

            // Start the ADC
            adc_start( ADC );

            // Wait for end of conversion
```

```
            while ((adc_get_status(ADC) & ADC_ISR_DRDY) != ADC_ISR_DRDY);

            // Read the value
            ulValue = adc_get_latest_value(ADC);
            ulValue = mapResolution(ulValue, ADC_RESOLUTION, _readResolution);
            break;

        // Compiler could yell because we don't handle DAC pins
        default :
            ulValue=0;
            break;
    }
#endif
    return ulValue;
}
```

지정된 아날로그 핀에서 값을 읽는 함수이다. analogRead 는 내부적으로 adc_start 함수를 호출하여 핀에서 데이터를 읽어온다. adc_start 를 부르고 while 문으로 데이터를 읽어올(Analog to Digital 변환이 끝날) 때까지 기다린다. 변환이 끝나면 adc_get_latest_value 를 통해 값을 가져오고, mapResolution 으로 지정한 해상도로 읽어온 값을 매핑한다. 최대 12 비트 해상도를 지원하기 위해 ADC0 부터 ADC11 까지 정의된 것을 확인할 수 있다.

```
// cores/arduino/wiring_analog.c
static inline uint32_t mapResolution(uint32_t value, uint32_t from, uint32_t
to) {
    if (from == to)
        return value;
    if (from > to)
        return value >> (from-to);
    else
        return value << (to-from);
}
```

mapResolution 은 위와 같이 데이터를 shift 하는 방식으로 구현되어 있다.

해상도는 analogReadResolution 함수를 통해 지정할 수 있다.

```
// cores/arduino/wiring_analog.c
void analogReadResolution(int res) {
    _readResolution = res;
}

void analogWriteResolution(int res) {
    _writeResolution = res;
}
```

Read resolution 의 기본값은 10bits 로 0 부터 1023 까지 범위로 값을 매핑한다. Due 보드는 최대 12bits 까지 resolution 을 설정할 수 있으며, 이는 0 부터 4095 까지의 범위이다.

```
// variants/arduino_due_x/variant.h
/*
 * Analog pins
 */
static const uint8_t A0  = 54;
static const uint8_t A1  = 55;
static const uint8_t A2  = 56;
static const uint8_t A3  = 57;
static const uint8_t A4  = 58;
static const uint8_t A5  = 59;
static const uint8_t A6  = 60;
static const uint8_t A7  = 61;
static const uint8_t A8  = 62;
static const uint8_t A9  = 63;
static const uint8_t A10 = 64;
static const uint8_t A11 = 65;
static const uint8_t DAC0 = 66;
static const uint8_t DAC1 = 67;
static const uint8_t CANRX = 68;
static const uint8_t CANTX = 69;
#define ADC_RESOLUTION    12
```

아두이노에는 각 아날로그 핀에 해당하는 포트 번호가 상수로 선언되어 있다. 따라서 아래와 같이 아날로그 포트 이름을 인자로 전달할 수 있다.

```
analogRead(A0);
```

analogWrite(pin, value)

```
// cores/arduino/wiring_analog.c
// Right now, PWM output only works on the pins with
// hardware support.  These are defined in the appropriate
// pins_*.c file.  For the rest of the pins, we default
// to digital output.
void analogWrite(uint32_t ulPin, uint32_t ulValue) {
    uint32_t attr = g_APinDescription[ulPin].ulPinAttribute;

    if ((attr & PIN_ATTR_ANALOG) == PIN_ATTR_ANALOG) {
        EAnalogChannel channel = g_APinDescription[ulPin].ulADCChannelNumber;
        if (channel == DA0 || channel == DA1) {
            uint32_t chDACC = ((channel == DA0) ? 0 : 1);
            if (dacc_get_channel_status(DACC_INTERFACE) == 0) {
                // ...

            // Write user value
```

```
            ulValue = mapResolution(ulValue, _writeResolution,
DACC_RESOLUTION);
            dacc_write_conversion_data(DACC_INTERFACE, ulValue);
            while ((dacc_get_interrupt_status(DACC_INTERFACE) & DACC_ISR_EOC)
== 0);
            return;
        }
    }

    if ((attr & PIN_ATTR_PWM) == PIN_ATTR_PWM) {
        ulValue = mapResolution(ulValue, _writeResolution, PWM_RESOLUTION);
        // ...
        uint32_t chan = g_APinDescription[ulPin].ulPWMChannel;
        if ((g_pinStatus[ulPin] & 0xF) != PIN_STATUS_PWM) {
            // Setup PWM for this pin
            PIO_Configure(g_APinDescription[ulPin].pPort,
                    g_APinDescription[ulPin].ulPinType,
                    g_APinDescription[ulPin].ulPin,
                    g_APinDescription[ulPin].ulPinConfiguration);
            PWMC_ConfigureChannel(PWM_INTERFACE, chan, PWM_CMR_CPRE_CLKA, 0,
0);
            PWMC_SetPeriod(PWM_INTERFACE, chan, PWM_MAX_DUTY_CYCLE);
            PWMC_SetDutyCycle(PWM_INTERFACE, chan, ulValue);
            PWMC_EnableChannel(PWM_INTERFACE, chan);
            g_pinStatus[ulPin] = (g_pinStatus[ulPin] & 0xF0) | PIN_STATUS_PWM;
        }

        PWMC_SetDutyCycle(PWM_INTERFACE, chan, ulValue);
        return;
    }

    if ((attr & PIN_ATTR_TIMER) == PIN_ATTR_TIMER) {
        // ...
        return;
    }

    // Defaults to digital write
    pinMode(ulPin, OUTPUT);
    ulValue = mapResolution(ulValue, _writeResolution, 8);
    if (ulValue < 128)
        digitalWrite(ulPin, LOW);
    else
        digitalWrite(ulPin, HIGH);
}
```

핀에 아날로그 값 (pwm wave)을 쓰는 명령이다. analogWrite 는 PWM 을 지원하는 핀에서만 PWM 을 출력하고, 지원하지 않는 핀에서는 HIGH 와 LOW 로 구분되는 디지털로 출력한다.

입력받은 값을 mapResolution 를 이용해 해상도를 변경하고, PWMC_SetDutyCycle 를 호출해 출력할 값만큼의 듀티사이클을 설정한다. PWMC_EnableChannel 으로 실제 신호를 출력한다.

출력할 해상도는 analogWriteResolution 를 통해 변경할 수 있다. 기본값은 8bits(0~255)이며 최대 12bits(0~4095)까지 설정할 수 있다.

pulseIn(pin, state, timeout=1000000L)

```
// cores/arduino/wiring_pulse.cpp
/* Measures the length (in microseconds) of a pulse on the pin; state is HIGH
 * or LOW, the type of pulse to measure.  Works on pulses from 2-3
microseconds
 * to 3 minutes in length, but must be called at least a few dozen
microseconds
 * before the start of the pulse.
 *
 * ATTENTION:
 * This function performs better with short pulses in noInterrupt() context
 */
uint32_t pulseIn( uint32_t pin, uint32_t state, uint32_t timeout )
{
    // ...
    uint32_t width = countPulseASM(&(p.pPort->PIO_PDSR), bit, stateMask,
maxloops);

    // convert the reading to microseconds. The loop has been determined
    // to be 18 clock cycles long and have about 16 clocks between the edge
    // and the start of the loop. There will be some error introduced by
    // the interrupt handlers.
    if (width)
        return clockCyclesToMicroseconds(width * 18 + 16);
    else
        return 0;
}
```

핀의 펄스를 읽는 함수이다. state 는 LOW 인지 HIGH 인지 펄스의 종류를 의미한다. 2 마이크로초부터 3 분 길이까지의 펄스를 읽어올 수 있지만, 펄스가 시작되기 전 수십 마이크로 초 이전에 함수를 호출해 둬야 한다. 함수가 반환하는 값의 단위는 마이크로초이다.

attachInterrupt(pin, isr, mode)

지정한 핀의 값이 변할 때 ISR 을 호출하도록 등록하는 함수이다. 아두이노 Due 보드에서는 모든 핀에 ISR 을 등록할 수 있다. mode 로 사용 가능한 값은 LOW, CHANGE, RISING, FALLING, HIGH 가

있다. ISR 내부에서 delay 와 millis 함수는 정상적을 동작하지 않는다. delay 는 동작을 하기 위해 인터럽트가 필요하므로 ISR 내부에서 호출되는 경우에는 작동하지 않는다. millis 도 현재 시각을 계산하기 위해 인터럽트을 이용하므로 ISR 내부에서 값이 증가하지 않는다. 다만, delayMicroseconds 는 카운터를 사용하지 않으므로 정상적으로 작동한다.

Problem

1. Write a review of contents of the practice.
2. Implement a code that functions as follows:
    - Usually, the motor speed is controlled by a potentiometer.
    - If the distance measured by the ultrasonic sensor is closer than 10cm, reduce it to 50% of the current speed
3. Solve the chattering problem using interrupts.

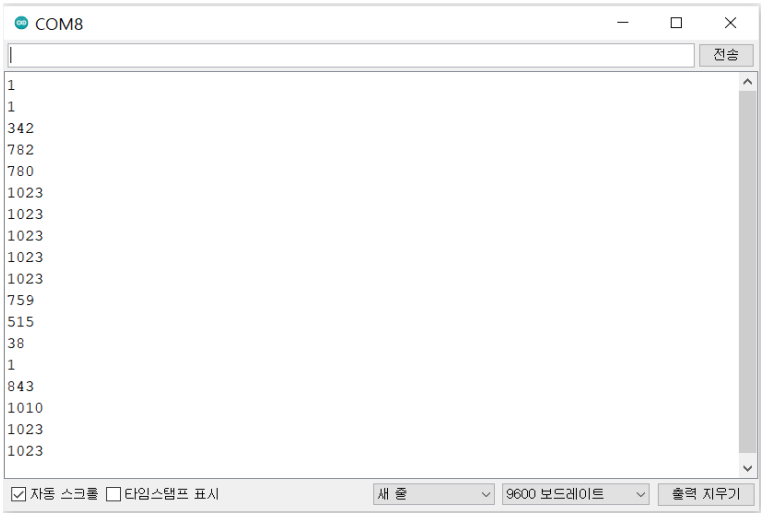
Result

Problem 1

1 번

```
const int AnalogPin = A0; // 사용할 아날로그 핀 정의

void setup() {
    Serial.begin(9600);
}

void loop() {
    int Ain = analogRead(AnalogPin); // 정의한 아날로그 핀에서 아날로그 값 읽음
    Serial.println(Ain); // 아날로그 값을 시리얼모니터에 출력
    delay(1000);
}
```

setup 에서 Serial 통신의 Baud Rate 를 9600bps 로 설정한다. analogRead 을 사용해서 A0 에서의 아날로그 값을 읽고 읽은 아날로그 값을 Serial.println(Ain)을 통해서 시리얼모니터에서 아날로그 값을 출력한다.



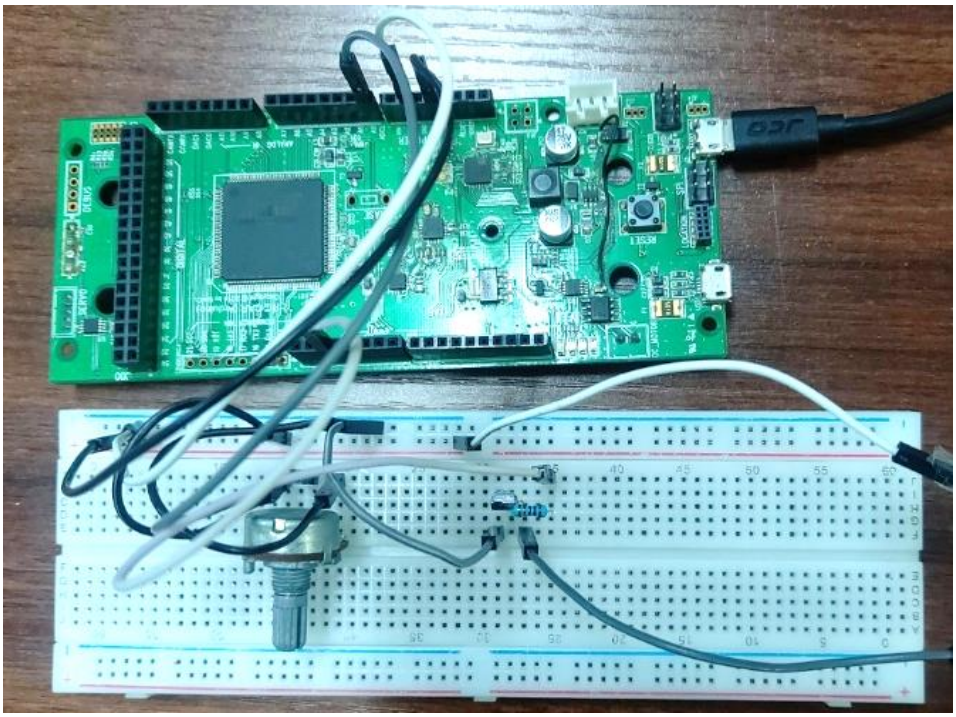
analogRead 는 0~1023 를 값을 출력한다. Read resolution 의 기본값은 10bits 로 0 부터 1023 까지 범위로 값을 매핑하는 것을 확인할 수 있었다.

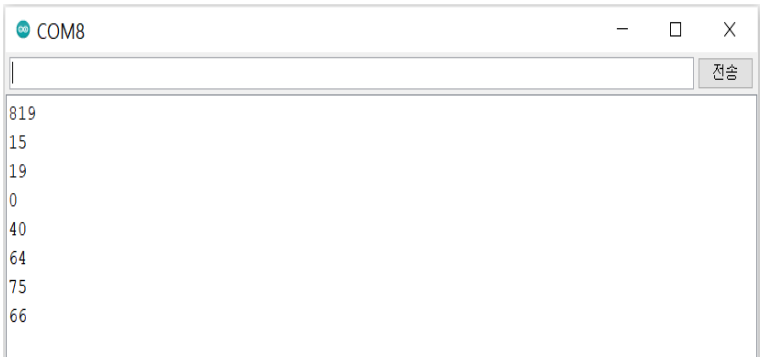
2 번

```
const int Ain = A0; // 아날로그 입력을 받아올 핀: A0
const int IN1 = 2; // 모터를 제어할 핀 2

void setup() {
    pinMode(IN1, OUTPUT);
    analogWriteResolution(10); // PWM resoultion 은 10bit 로 설정
    Serial.begin(9600);
}

void loop() {
    int A = analogRead(Ain); // 아날로그 값을 읽어와 저장
    Serial.println(A); // 아날로그 값을 출력
    delay(100);

    int value = analogRead(Ain); // 아날로그 값을 읽어와 저장
    int pwm = map(value, 0, 4095, 0, 1023); // value 값의 범위를 0 에서
4095 에서 0 에서 1023 으로 변환하여 pwm 변수에 저장
```

```
    analogWrite(IN1, pwm); // PWM 신호를 IN1 핀에 출력하여 모터를 제어
}
```

analogWriteResolution(10);을 통해서 PWM resolution 을 10bit 로 설정했다. 10bit 로 설정을 하게 된다면 0~1023 까지의 단계를 갖게 된다. map(value, 0, 4095, 0, 1023);는 입력하는 값의 범위를 1/4 하게 된다. 따라서 pwm 값의 범위는 0~255 가 되게 된다. 위 코드는 가변저항을 통해 모터의 속도를 제어하는 코드이다.



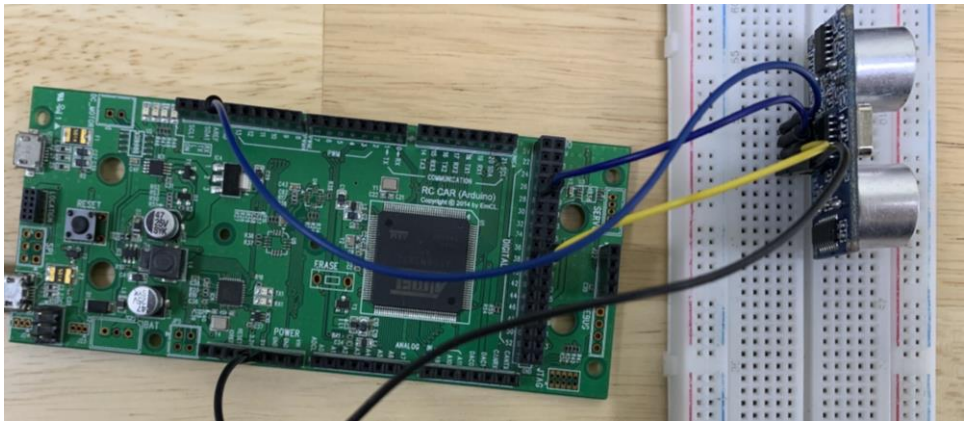


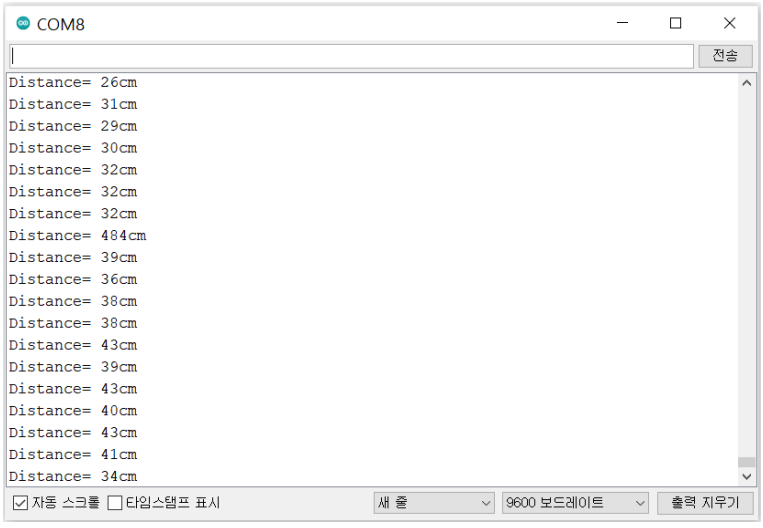
3 번

```
unsigned long distance; // 거리를 저장할 변수를 정의

void setup() {
    pinMode(39, OUTPUT); // trig pin
    pinMode(28, INPUT_PULLUP); // echo pin
    Serial.begin(9600);
}

void loop() {
```

```
    unsigned long Width; // 초음파가 왔다갔다 하는 시간을 저장하는 변수

    digitalWrite(39, HIGH); // trig pin 에 HIGH 신호를 출력시켜 초음파 발사
    delayMicroseconds(10); // 10ms 동안 유지
    digitalWrite(39, LOW);

    Width = pulseIn(28, HIGH); // echo pin 에서 HIGH 신호의 길이를 측정하여 왕복
시간을 계산
    distance = Width / 58; // 왕복시간을 계산하여 거리를 구함

    Serial.print("Distance= ");
    Serial.print(distance);
    Serial.print("cm\n");
}
```

초음파센서는 hc-sr04 라는 제품이다. Trig pin 을 통해서 HIGH 신호를 출력시켜서 초음파를 발사해서 다시 돌아오는 시간을 계산해서 거리를 측정한다. `Width = pulseIn(28, HIGH);`을 통해서 28 번 핀에서 HIGH 신호를 통해 초음파를 발사하고 발사된 초음파가 돌아오는 시간을 `distance = Width / 58;`을 통해 거리를 계산한다. `delayMicroseconds(10);`을 통해서 10ms 동안 HIGH level 을 유지한다.

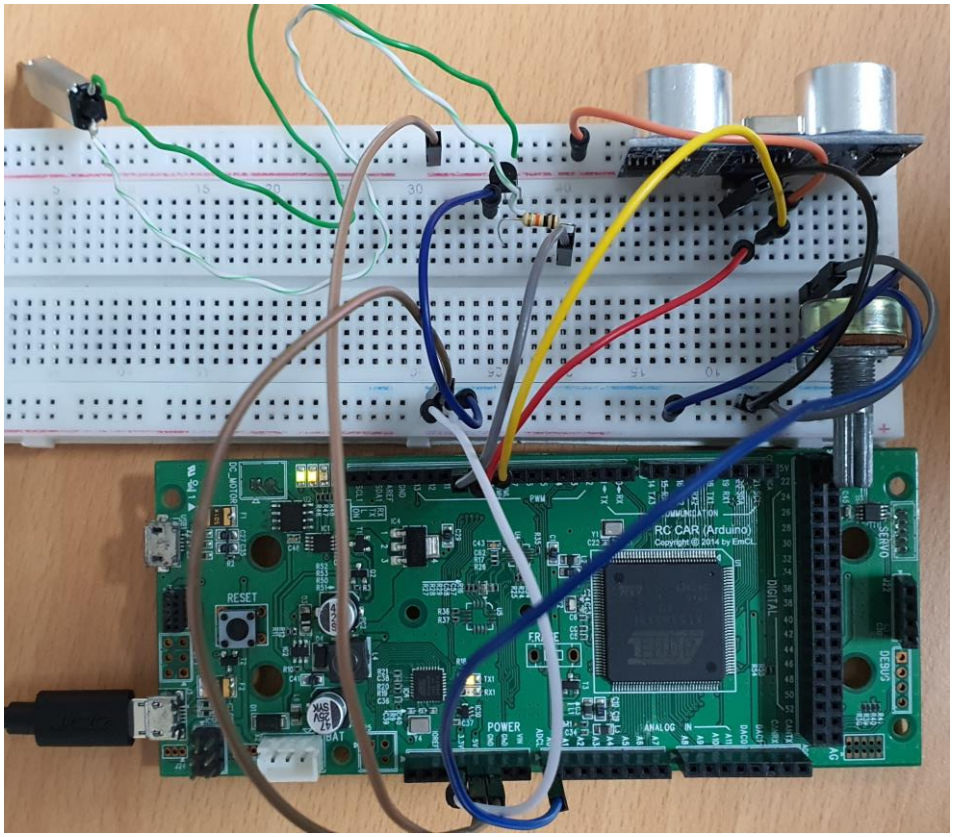


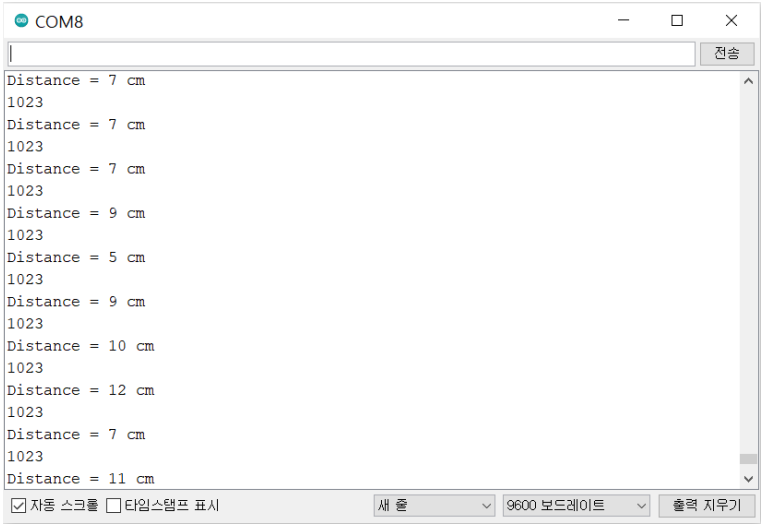


Problem 2

```
unsigned long distance;

const int Ain = A0;

void setup() {
    pinMode(8, OUTPUT); // trig pin
    pinMode(9, INPUT_PULLUP); // echo pin
    pinMode(10, OUTPUT); // motor pin
    analogWriteResolution(10);
    Serial.begin(9600);
}

void loop() {
    unsigned long Width;

    digitalWrite(8, HIGH);
    delayMicroseconds(10);
    digitalWrite(8, LOW);

    Width = pulseIn(9, HIGH);
    distance = Width / 58;

    int val = analogRead(A0);
    Serial.println(val);
    int pwm = map(val, 0, 4095, 0, 1023);

    if (distance <= 10) {
        analogWrite(10, pwm / 2); // 10cm 이하면 모터 속도 50% 감속
    } else {
        analogWrite(10, pwm);
    }

    Serial.print("Distance = ");
    Serial.print(distance);
    Serial.println(" cm");
}
```

위의 코드는 초음파 거리 측정 센서를 이용해 계산한 distance 에 따라 모터의 속도를 제어하는 코드이다. 초음파 센서를 이용해서 거리를 측정하고 이때의 아날로그 값을 `int val = analogRead(A0);` 의 코드를 통해서 val 이라는 변수에 아날로그의 값을 입력한다. val 을 통해서 pwm 신호를 mapping 하고 이때 아날로그 값으로 얻은 거리를 기반으로 모터를 조절한다.
모터는 `analogWrite` 함수를 이용해 PWM 신호를 발생시켜 모터의 속도를 조절하는데, distance 가 10cm 이하일 경우, 모터의 속도를 50% 감속되게 하였다.





Problem 3

1 번

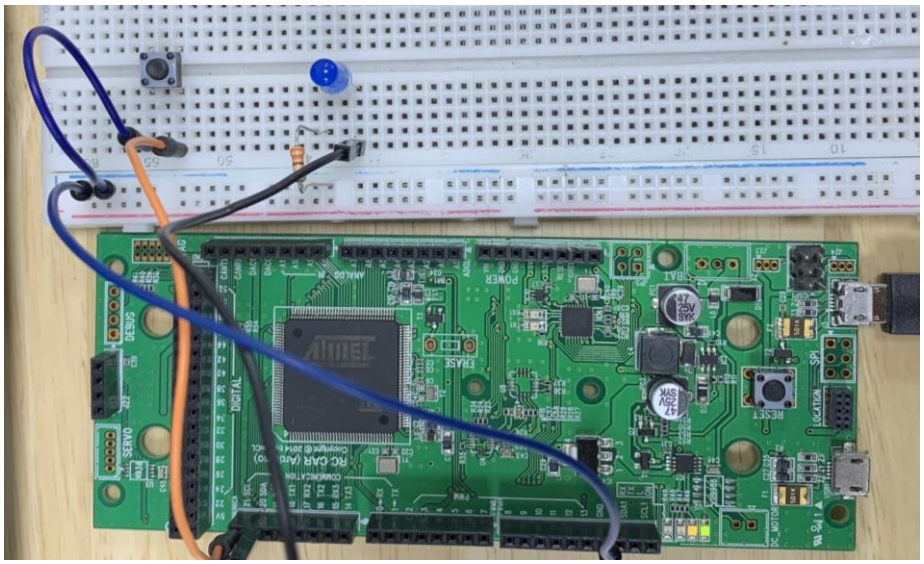
```
#define LD0 13 // 1 초마다 꺼졌다 켜졌다하는 LED
#define LD1 20
#define SW 21

bool LD1_state = false; // LD1 의 상태를 false 로 초기화

void setup() {
```

```
    pinMode(LD0, OUTPUT);
    pinMode(LD1, OUTPUT);
    pinMode(SW, INPUT_PULLUP);
}

void loop() {
    if (digitalRead(SW) == 0) {
        digitalWrite(LD1, LD1_state);
        LD1_state = !LD1_state;
    }

    digitalWrite(LD0, LOW);
    delay(1000);
    digitalWrite(LD0, HIGH);
    delay(1000);
}
```

```
#define LD0 13 // 1 초마다 꺼졌다 켜졌다하는 LED
#define LD1 20
#define SW 21

bool LD1_state = false; // LD1 의 상태를 false 로 초기화

void setup() {
    pinMode(LD0, OUTPUT);
    pinMode(LD1, OUTPUT);
    pinMode(SW, INPUT_PULLUP);

    attachInterrupt(SW, SW_ISR, FALLING); // 스위치 Interrupt 를 FALLING EDGE 에
연결
}

void loop() {
    digitalWrite(LD0, LOW);
    delay(1000);
    digitalWrite(LD0, HIGH);
    delay(1000);
}

// 스위치 Interrupt Service Routine
void SW_ISR() {
    LD1_state = !LD1_state;
    digitalWrite(LD1, LD1_state);
    delayMicroseconds(100000); // 100ms
}
```

위의 두 코드에서 LD0 핀은 1 초마다 켜졌다 꺼졌다가 하는 LED 이고, LD0 는 사용자가 스위치를 누르면 LED 의 상태가 변경되는 코드이다. 두 코드의 차이점은 스위치를 눌렀을 경우, LD1 의 상태를 변경하는 부분이 다르다.

첫 번째 코드의 경우, `loop` 함수에서 스위치의 입력을 통해, LD1 의 상태를 변경하는 코드이다. 그러나 두 번째 코드는, `attachInterrupt` 함수를 사용해서 스위치 핀을 'Falling Edge'에서 Interrupt 를 발생하도록 하는데, 인터럽트가 발생되면 `SW_ISR` 함수가 호출돼 LD1 의 상태를 반전시킨 코드이다.



Conclusion

problem1

1 번

문제 1 번에서 사용한 가변저항은 회전을 시키면 저항의 값이 달라지게 한다. 아두이노 보드를 보게 되면 가변저항은 아날로그 값을 읽을 수 있는 포트에만 연결을 해서 사용할 수 있다. 이때 사용하는 가변저항은 극성은 구별을 하지 않아도 되지만 방향에 따라서 1023 이 가장 높은 저항이 될 수도 0 이 가장 높은 저항이 될 수도 있다.

2 번

가변저항의 아날로그 bit 는 총 1024 단계 (0~1023)의 단계로 나눠진다. 우리가 실험에서 사용하는 Due 의 경우는 12bit 기반에 4096 단계 (0~4095)로 나눠진다. 이때 불러온 analog 의 값을 불러오게 된다면 1023 을 넘게 된다면 자동으로 1023 을 인식을 하게 되므로, 1023 이후의 값에서는 가변저항의 변화에 대한 정확한 값을 읽어올 수 없기 때문에, 이를 활용한 제어는 어려워지게 된다. 이를 해결하기 위해서 문제의 코드에서는 map(value,0,4095,0,1023)의 코드를 사용해서 4096 단계의 아날로그 값을 1024 단계의 아날로그 단계로 변환하는 작업이 필요하다. 해당 단계를 거치게 되면 모든 단계에서 모터의 속도를 조절할 수 있다.

3 번

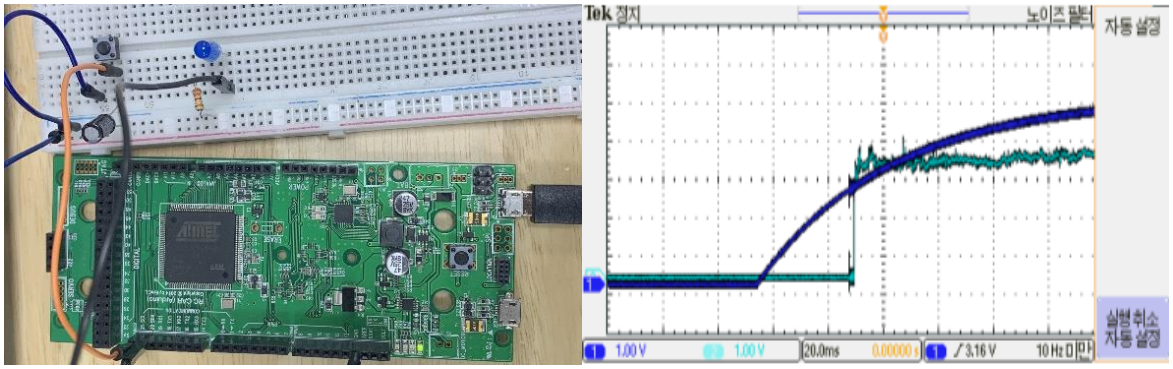
해당 문제에서 사용한 제품은 hc-sr04 라는 제품이다. 실험하면서 초음파센서에 손을 가깝게 가져가도 해당 거리는 2cm 보다 작은 값이 나오지 않았다. 이유는 hs-sr04 의 스펙에서 측정각도 15 도를 기준으로 최소 측정거리가 2cm 최대 측정거리가 5m 가 되기 때문에 너무 작은 값을 얻지 못한다는 결과를 얻을 수 있다.

problem2

초음파 센서를 통해서 10cm 이하일 경우 모터의 속도가 절반으로 감속하는 문제였다. `analogWrite(10, pwm/2)` 의 코드를 사용해서 절반으로 모터의 속도를 줄이는 걸 구현했다. pwm 변수는 map(value, 0, 4095, 0, 1023)의 코드에서 아날로그의 값을 받아와서 저장을 하는 역할을 한다. map(value, 0, 4095, 0, 1023)코드는 0-4095 까지의 범위를 0-1023 범위로 mapping 을 하기 때문에 value/4 를 return 한다고 말할 수 있다. 실험에서 사용하는 due 보드의 경우 4096 레벨이 존재하지만 받아들일 수 있는 레벨은 1024 레벨이기 때문에 위의 map(value, 0, 4095, 0, 1023) 코드를 사용하지 않으면 넘어가는 pwm 값이 1023 이 넘어가도 1023 일때와 똑같이 동작을 하게 된다.

problem3

Problem 3 은 chattering 을 방지하기 위해서 software 적으로 해결을 하는 방법이었다. Delay 를 통해서 chattering 이 일어나는 시간에 대해서는 처리를 하지 않음으로 방지를 했다면 아래의 방법은 capacitor 를 이용해서 chattering 을 방지하는 방법이다.



오른쪽의 사진은 chattering 을 방지하기 위해서 capacitor 를 사용한 회로의 사진이다. 사용한 capacitor 는 극성이 존재하는데 +극은 아두이노에 -극은 ground 에 연결이 되도록 만들었다. chattering 을 극복하기 위해서 사용하는 capacitor 는 스위치가 켜졌을 때 흐르는 전류에 의해

충전이 되면서 왼쪽의 사진과 같이 0 -> 1 로 서서히 올라가게 구현이 되면서 chattering 을 방지할 수 있다.