Auto Car Project

With ATmega128A

(Telechips) AI 시스템반도체 SW 개발자

박성호 이경주

TABLE OF CONTENTS

01 프로젝트 개요

프로젝트 목적

- AVR의 ATmega128A 마이크로컨트롤러에 대한 이해 및 응용
- Bluetooth 통신 모듈을 이용한 수동 주행 구현
- Ultrasonic (초음파 센서)를 이용한 자율 주행 구현

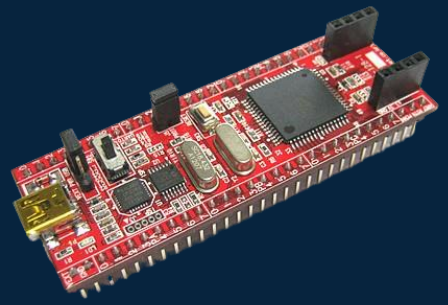
프로젝트 내용

- 자율주행자동차의 동작 구현
- 자율주행 동작의 가시적 효과(LED, BUZZER, FND, I2C LCD)

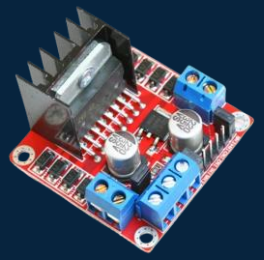
01 프로젝트 개요

주요 부품

ATmega128A



L298N Motor Drive

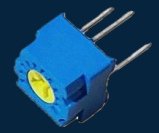


FND

BUTTON

BUZZER

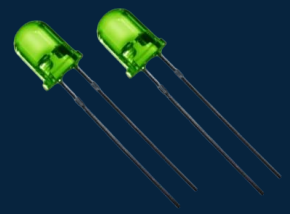
Single Turn Trimmer



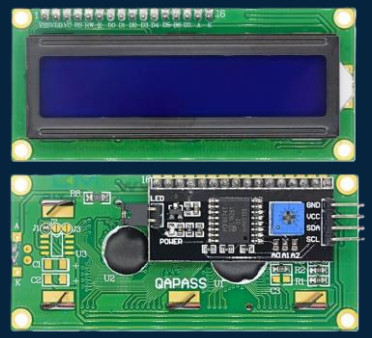
Ultrasonic

DC MOTOR

LED

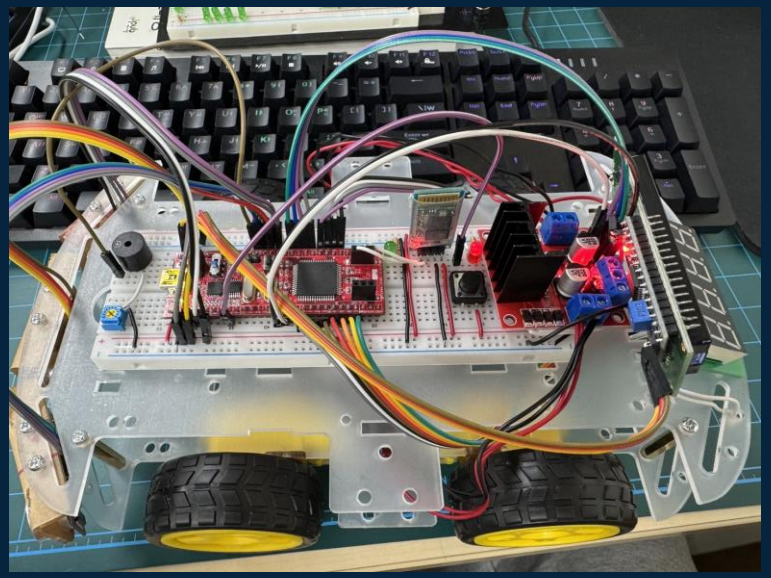


I2C LCD

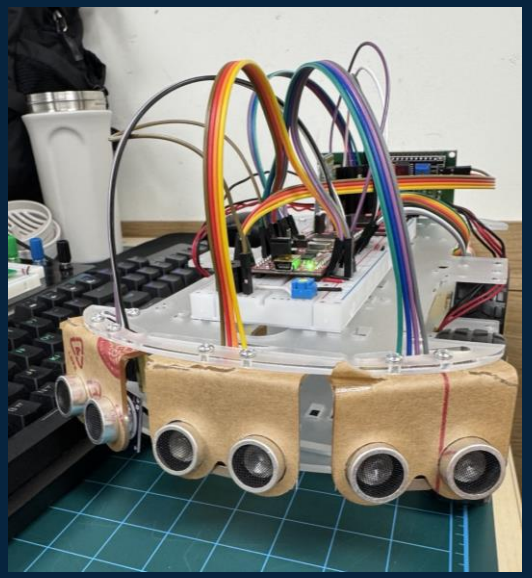


02 Schematic & FSM

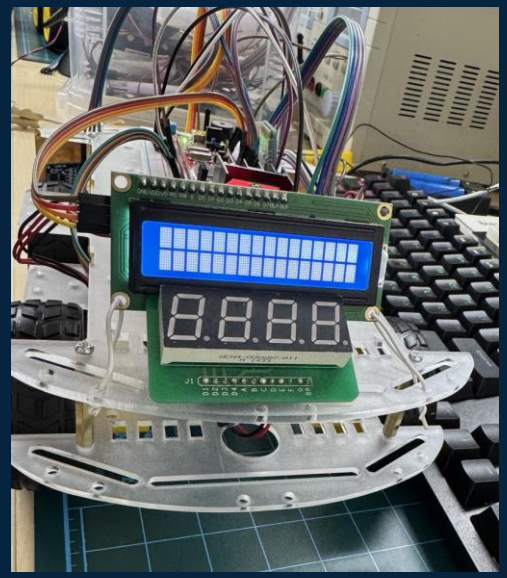
외관 사진



상면



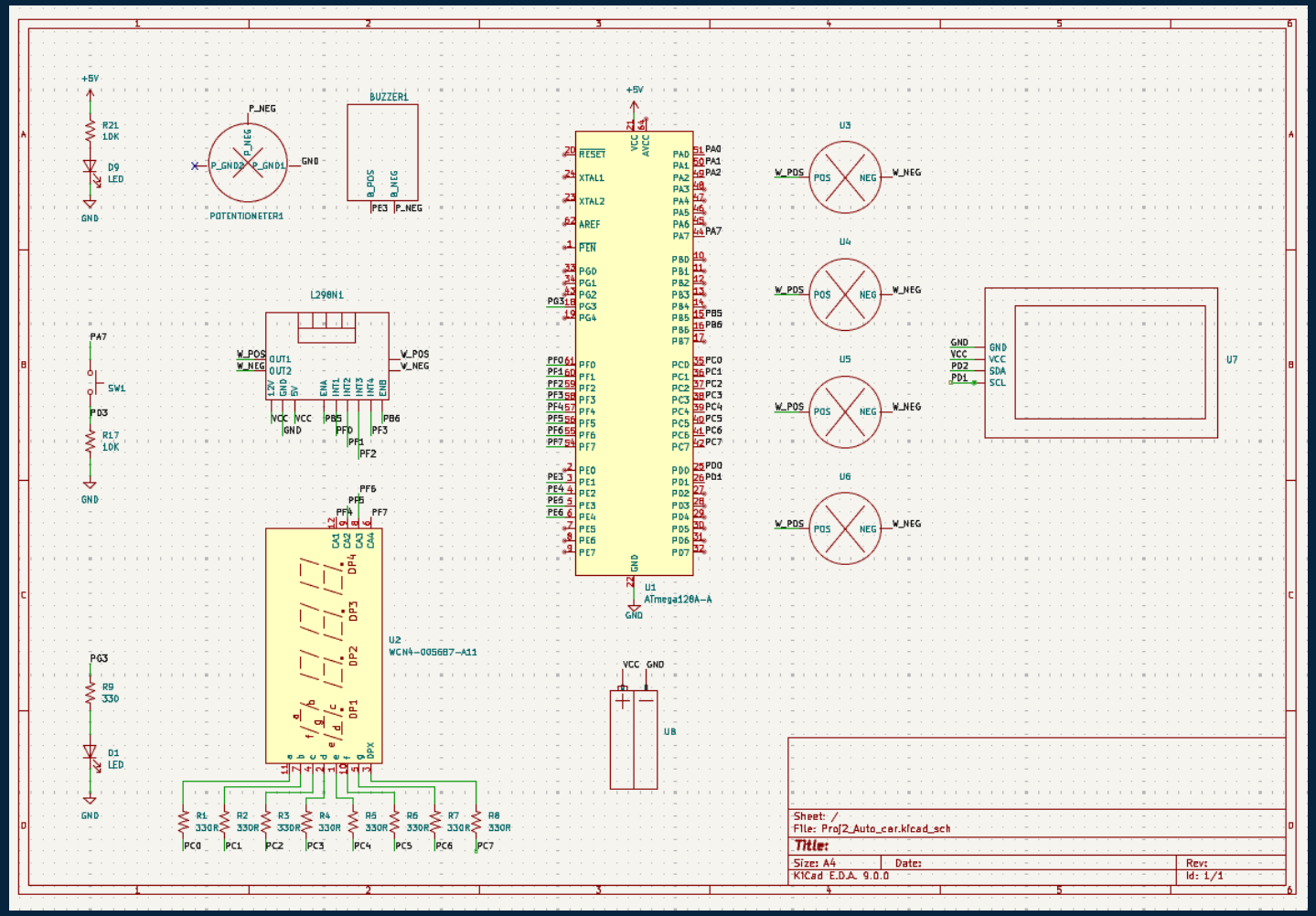
전면



후면

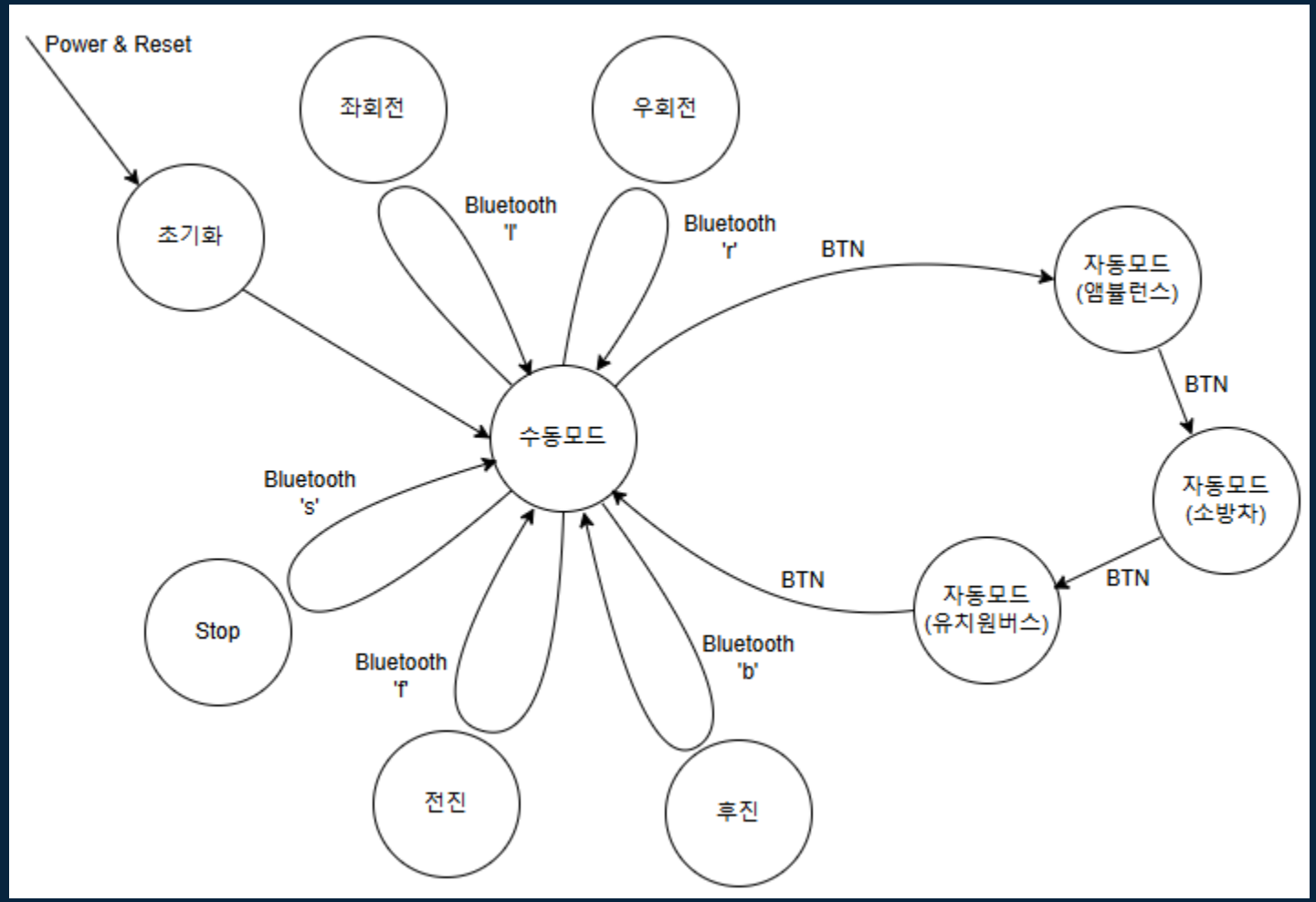
02 Schematic & FSM

Schematic (회로도)



02 Schematic & FSM

FSM (상태천이도)



03 세부 구현 기능

기능 설명

- 기본 전원 - 수동 주행 모드
  - 블루투스 연결 - ‘f’, ‘b’, ‘l’, ‘r’, ‘s’ 키를 통해 주행 가능
- BUTTON - 자율 주행 모드
  - 초음파 센서로 얻은 거리 데이터를 통해 DC 모터(바퀴) 방향 제어
- BUZZER - 버튼을 통해 멜로디 설정 가능 (구급차, 소방차, 어린이차)
- FND - 자율 주행 중 전진, 후진, 좌회전, 우회전에 따른 애니메이션 출력
- I2C LCD - 자율 주행 중 전진, 후진, 좌회전, 우회전에 따른 글자 출력
- LED - 자율 주행 시에는 점등, 수동 모드일 때는 소등

04 코드 설명 (모터제어)

```
void init_timer1(void)
{
    // 분주비 : 64 16000000 / 64 ---> 250000Hz(250Khz)
    // T=1/f = 1/250000Hz ==> 0.000004sec (4us)
    // 250000Hz 에서 256개의 펄스를 count하면 소요시간 1.02ms
    //            127개                      0.5ms
    // P318 표 14-1
    TCCR1B |= 1 << CS11 | 1 << CS10;   // 분주비 : 64

    // timer 1번의 모드 14 : 고속 PWM (P327 표 14-5)
    TCCR1A |= 1 << WGM11; // TOP을 ICR1에 지정을 할 수 있다.
    TCCR1B |= 1 << WGM13 | 1 << WGM12;

    // 비반전 모드 TOP : ICR1 비교일치값(PWM) 지정 : OCR1A, OCR1B 표 15-7
    // 비교일치 발생시 OCR1A, OCR1B의 출력 핀은 LOW로 바뀌고 BOTTOM에서 HIGH로 바뀐다.
    // P350 표 15-7
    TCCR1A |= 1 << COM1A1 | 1 << COM1B1;

    ICR1 = 0x3ff; // 1023 ==> 4ms TOP : PWM값
}
```

모터 포트 할당

- 왼쪽 - PORTF 0 ~ 1 (IN1, IN2)
- 오른쪽 - PORTF 2 ~ 3 (IN3, IN4)

PWM 제어

- 16비트 타이머 / 카운터
- 분주 : 64

04 코드 설명 (수동 주행)

```c
while(1)
{
    // 버튼이 눌리면 모드를 순환시키고, 이전 사운드(모드)를 정지합니다.
    if(get_button(BUTTON0, BUTTON0PIN))
    {
        mode = (mode + 1) % 4;  // 0 -> 1 -> 2 -> 3 -> 0 순환
        stopSpeaker();          // 모드 전환 시 이전 사운드 정지
        _delay_ms(100);         // 디바운싱 지연
    }

    // 모드가 변경된 경우 새 모드에 맞는 사운드를 시작합니다.
    if(mode != prevMode)
    {
        switch(mode)
        {
            case 0:
                // 수동 모드: 사운드 없음
                AUTO_RUN_LED_PORT &= ~(1 << AUTO_RUN_LED_PIN);
                manual_mode();
                break;
            case 1:
                AUTO_RUN_LED_PORT |= 1 << AUTO_RUN_LED_PIN;
                auto_start();
                startAmbulanceSiren(5);  // Ambulance 모드: 5 사이클 기준, 모드가 유지되면 반복 재생
                break;
            case 2:
                AUTO_RUN_LED_PORT |= 1 << AUTO_RUN_LED_PIN;
                auto_start();
                startFireTruckSiren(5);  // FireTruck 모드: 5 사이클 기준, 모드가 유지되면 반복 재생
                break;
            case 3:
                AUTO_RUN_LED_PORT |= 1 << AUTO_RUN_LED_PIN;
                auto_start();
                startTwinkleTwinkle();   // Twinkle 모드: 멜로디 반복 재생
                break;
            default:
                break;
        }
        prevMode = mode;
    }

    // 백그라운드 사운드 업데이트 (speaker.c의 상태 머신 처리)
    speaker_update();

    _delay_ms(50); // 메인 루프 딜레이 (버튼 반복 및 기타 작업)
}

return 0;
}
```

main 함수

```c
void manual_mode(void) // 수동 모드
{
    switch(bt_data)
    {
        case 'F':
        case 'f':
            forward(1000); // 4us x 500 = 0.002sec (2ms)
            break;
        case 'B':
        case 'b':
            backward(1000);
            break;
        case 'L':
        case 'l':
            turn_left(1000);
            break;
        case 'R':
        case 'r':
            turn_right(1000);
            break;
        case 'S':
        case 's':
            stop();
            break;
        default:
            break;
    }

    func_index = DISTANCE_CHECK;
}
```

manual mode 동작 함수

```c
void stop(void)
{
    MOTOR_DRIVER_DIRECTION_PORT &= ~(1 << 0 | 1 << 1 | 1 << 2 | 1 << 3);
    MOTOR_DRIVER_DIRECTION_PORT |= 1 << 0 | 1 << 1 | 1 << 2 | 1 << 3; // 자동차를 stop 모드로 설정

    I2C_LCD_write_string_XY(0, 0, "      STOP      ");
}

void forward(int speed)
{
    MOTOR_DRIVER_DIRECTION_PORT &= ~(1 << 0 | 1 << 1 | 1 << 2 | 1 << 3);
    MOTOR_DRIVER_DIRECTION_PORT |= 1 << 0 | 1 << 2; // 전진 모드

    OCR1A = OCR1B = speed; // PB5(OCR1A) : 왼쪽, PB6(OCR1B) : 오른쪽

    I2C_LCD_write_string_XY(0, 0, "     FORWARD    ");
}

void backward(int speed)
{
    MOTOR_DRIVER_DIRECTION_PORT &= ~(1 << 0 | 1 << 1 | 1 << 2 | 1 << 3);
    MOTOR_DRIVER_DIRECTION_PORT |= 1 << 1 | 1 << 3; // 후진 모드 1010

    OCR1A = OCR1B = speed; // PB5(OCR1A) : 왼쪽, PB6(OCR1B) : 오른쪽

    I2C_LCD_write_string_XY(0, 0, "    BACKWARD    ");
}

void turn_left(int speed)
{
    MOTOR_DRIVER_DIRECTION_PORT &= ~(1 << 0 | 1 << 1 | 1 << 2 | 1 << 3);
    MOTOR_DRIVER_DIRECTION_PORT |= 1 << 0 | 1 << 2; // 전진 모드

    OCR1A = 0; // PB5(OCR1A) : 왼쪽
    OCR1B = speed; // PB6(OCR1B) : 오른쪽

    I2C_LCD_write_string_XY(0, 0, "    TURN LEFT   ");
}
```

각 동작 함수 구현

04 코드 설명 (자율 주행)

```
void auto_start(void)
{
    if(ultrasonic_trigger_timer >= 200)
    {
        ultrasonic_trigger();

        ultrasonic_trigger_timer = 0;
    }
```

초음파 센서 거리 체크

- 0.2sec 마다 초음파 센서 트리거 발생
- 0.2sec 마다 체크 후 트리거 타이머 값 초기화

```
    if(ultrasonic_front_distance > 20)
    {
        if(ultrasonic_left_distance > 20 && ultrasonic_right_distance > 20)
        {
            forward(500);
            fnd_forward_display();
        }
        else if(abs(ultrasonic_left_distance - ultrasonic_right_distance) < 10)
        {
            forward(400);
            fnd_right_display();
        }
        else
        {
            if(ultrasonic_left_distance >= ultrasonic_right_distance)
            {
                turn_left(500);
                fnd_left_display();
            }
            else
            {
                turn_right(500);
                fnd_right_display();
            }
        }
        return;
    }
    if(ultrasonic_front_distance <= 20)
    {
        if(ultrasonic_left_distance <= 20 && ultrasonic_right_distance <= 20)
        {
            backward(400);
            fnd_backward_display();
            _delay_us(100);

            if(ultrasonic_left_distance >= ultrasonic_right_distance)
            {
                turn_left(500);
                fnd_left_display();
            }
            else
            {
                turn_right(500);
                fnd_right_display();
            }
            return;
        }
        if(ultrasonic_left_distance > ultrasonic_right_distance)
        {
            turn_left(500);
            fnd_left_display();
        }
        else
        {
            turn_right(500);
            fnd_right_display();
        }
        return;
    }
}
```

3개의 초음파 센서를 통해 얻은 거리 데이터를 바탕으로 DC 모터의 동작을 제어하는 조건

- 전방 초음파 센서의 데이터가 20cm보다 클 때
  - 왼쪽, 오른쪽 초음파 센서의 데이터가 20cm보다 크면 전진
  - 왼쪽, 오른쪽 초음파 센서의 데이터 차이가 10cm보다 작으면 전진
  - 왼쪽 초음파 센서의 데이터가 오른쪽보다 크거나 같으면 좌회전
  - 오른쪽 초음파 센서의 데이터가 왼쪽보다 크면 우회전

- 전방 초음파 센서의 데이터가 20cm보다 작거나 같을 때
  - 왼쪽, 오른쪽 초음파 센서의 데이터가 20cm보다 작거나 같으면 후진
    - 왼쪽 초음파 센서의 데이터가 오른쪽보다 크거나 같으면 좌회전
    - 오른쪽 초음파 센서의 데이터가 왼쪽보다 크면 우회전
  - 왼쪽 초음파 센서의 데이터가 오른쪽보다 크면 좌회전
  - 오른쪽 초음파 센서의 데이터가 왼쪽보다 크면 우회전

04 코드 설명 (초음파 센서 제어)

```
ISR(INT4_vect) // LEFT
{
    // 1. 상승 edge
    if (ECHO_PIN_LEFT & (1 << ECHO_LEFT))
    {
        TCNT2 = 0;
    }
    // 2. 하강 edge
    else
    {
        // ECHO 핀에 들어온 펄스 길이를 us로 환산
        ultrasonic_left_distance = (1000000.0 * TCNT2 * 128 / F_CPU) / 58;

        sprintf(scm_L,"dis_L= %dcm", ultrasonic_left_distance);
    }
}
```

외부 인터럽트 서비스 루틴(INT4 ~ INT6)

- 에코 펄스가 발생한 시점부터 TCNT2 증가
- TCNT2값과 적절한 식을 이용하여 거리 값 저장
- INT4(왼쪽), INT5(정면), INT6(오른쪽) 루틴 동일

```
void init_ultrasonic(void)
{
    // ---------------- left ----------------
    TRIG_DDR_LEFT |= 1 << TRIG_LEFT;
    ECHO_DDR_LEFT &= ~(1 << ECHO_LEFT);

    EICRB |= 0 << ISC41 | 1 << ISC40;

    TCCR2 |=  1 << CS21 | 1 << CS20;
    EIMSK |= 1 << INT4;

    // ---------------- front ----------------
    TRIG_DDR_FRONT |= 1 << TRIG_FRONT;
    ECHO_DDR_FRONT &= ~(1 << ECHO_FRONT);


    EICRB |= 0 << ISC51 | 1 << ISC50;

    TCCR2 |=  1 << CS21 | 1 << CS20;
    EIMSK |= 1 << INT5;

    // ---------------- right ----------------
    TRIG_DDR_RIGHT |= 1 << TRIG_RIGHT;
    ECHO_DDR_RIGHT &= ~(1 << ECHO_RIGHT);

    EICRB |= 0 << ISC61 | 1 << ISC60;

    TCCR2 |=  1 << CS21 | 1 << CS20;
    EIMSK |= 1 << INT6;
}
```

초음파 레지스터 값 설정

- TRIG DDR 출력모드, ECHO DDR 입력모드 설정
- 초음파 센서 포트에 신호가 들어오면 인터럽트(4, 5, 6) 발생
- 분주비 : 128

04 코드 설명 (부저 멜로디 제어)

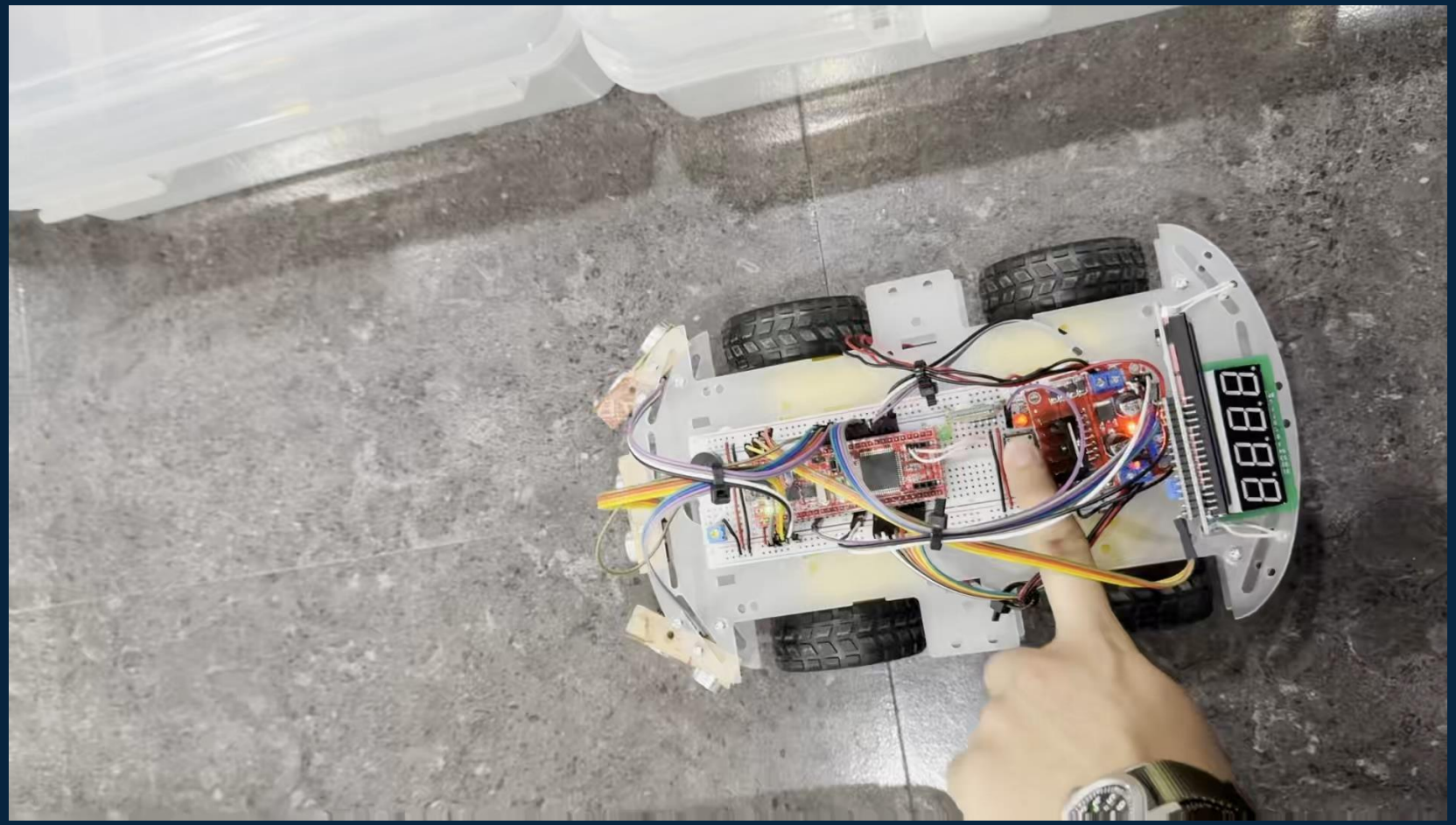
```
ISR(TIMER3_COMPA_vect)
{
    // 현재 모드가 FIRETRUCK일 때만 동작
    if (currentSpeakerMode == SPEAKER_MODE_FIRETRUCK) {
        if (ftState == 0) {
            // 감소 단계: OCR3A 값을 감소시켜 주파수를 높임 (300Hz -> 750Hz)
            if (OCR3A > FT_MIN_OCR3A) {
                OCR3A -= FT_STEP;
            }
            ftInnerCount++;
            if (ftInnerCount >= FT_THRESHOLD) {
                ftInnerCount = 0;
                ftState = 1;
            }
        } else {
            // 상승 단계: OCR3A 값을 증가시켜 주파수를 낮춤 (750Hz -> 300Hz)
            if (OCR3A < FT_MAX_OCR3A) {
                OCR3A += FT_STEP;
            }
            ftInnerCount++;
            if (ftInnerCount >= FT_THRESHOLD) {
                ftInnerCount = 0;
                ftState = 0;
                ftCycleCount++;  // 한 사이클(하강+상승) 완료
                // 기존에는 정해진 사이클 후 정지했으나, 지속 재생을 위해 리셋합니다.
                if (ftCycleCount >= ftTotalCycles) {
                    ftCycleCount = 0;
                }
            }
        }
    }
}
```

부저 제어

- Timer3 사용

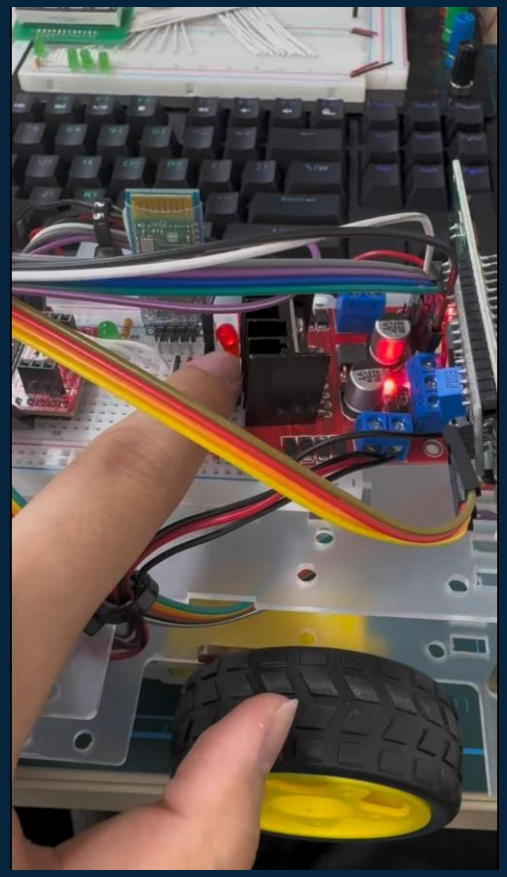
05 결과 및 개선점

자율 주행 영상 (https://youtube.com/shorts/6Wx-I0pbpkk)



05 결과 및 개선점

BUZZER, FND, I2C LCD 영상
(https://youtube.com/shorts/5h3yXWO0DD4)
(https://youtube.com/shorts/7KOlTIEEiLM)



05 결과 및 개선점

문제점

디테일한 자율 주행 구현의 한계

- 초음파 센서의 부정확성, 오버플로우로 인한 정확한 거리 측정에 한계
- 주행 중간 초음파 센서의 데이터를 받아오지 못해 주행을 잠시 멈추는 문제

부족한 배터리 용량

- 배터리 방전 혹은 충전 부족으로 인해 자주 충전을 해주어야 하는 문제

05 결과 및 개선점

개선점

디테일한 자율 주행 구현의 한계

- 초음파 센서의 개수와 위치 추가 (후면, 측면 등)를 통해 주변 장애물에 대한 정확한 데이터 수집
- 초음파 센서의 안정화 - 오버플로우를 감지하고 처리하는 로직 추가

부족한 배터리 용량

- 주행 테스트 전 센서들의 출력 데이터 확인과 같은 과정에서는 Power Supply를 이용하여 배터리 사용 최소화

Thank You

(Telechips) AI 시스템반도체 SW 개발자

박성호 이경주