임베디드 시스템 설계 중간과제 보고서

WAV File player

12181407 이종범

목차

1. 인터페이스 구현과정

0) 인터럽트 설정

```
void ext_int3_init() {
    EICRA |= _BV(ISC31);
    EICRA &= ~_BV(ISC30);

    EIMSK |= _BV(INT3);
    EIFR |= _BV(INTF3);
}

void ext_int1_init() {
    EICRA &= ~_BV(ISC10);
    EICRA |= _BV(ISC11);

    EIMSK |= _BV(INT1);
    EIFR |= _BV(INTF1);
}
```

Ext1_init은 INT1 인터럽트를 활성화시키는 함수이며, falling edge를 검출한다
Ext2_iniit은 rotary encoder의 INT2 인터럽트를 활성화시키는 함수이며 encoder에서 발생하는 falling edge를 검출한다.

```
ISR(INT1_vect) {
    if (play_state == 0) { //정지 상태면
        play_state = 1; //재생상태로 바꾸고
        change = 1;
    }
    else {
        play_state = 0; //재생 중이면 정직로 바꿔준다
        choice = 1; //인터페이스 이용가능한 설정으로 진입하도록 만들어준다
    }
}

ISR(INT3_vect) {
    if (digitalRead(PhaseB)) cnt++;

    else cnt--;

    if (cnt >= 15) cnt = 0;

    else if (cnt <= 0) cnt = 0;
}
```

스위치 Falling edge 검출 시, 재생상태를 바꾸는 일을 수행하고, 엔코더 falling edge 검출 시, cnt값을 0~15까지 증가시키는 일은 수행한다.

1) 클래스 방식

```
File root;
File entry[16]; //모든 파일들을 담는 파일 배열 생성
File stage;
```

```
void readfile(File dir) {
    dir.rewindDirectory();
    dir.openNextFile();
    for (int i = 0; i < 4; i++) {
        entry[i] = dir.openNextFile();
    }
}
```

```
display.setCursor(0, 0); //0번줄 0번째 칸
display.print("                "); //0번
display.setCursor(0, 0); //0번줄 0번째 칸
display.print(entry[cnt].name()); //각 파

display.setCursor(0, 1); //1번줄 0번째 칸
display.print("                "); //1번
display.setCursor(0, 1); //0번줄 0번째 칸
display.print(state[play_state]); //재생
```

처음에는 파일명을 배열에 저장해서 이를 출력하려고 했다. 하지만, 배열에 파일명을 저장하게 되면 배열의 모든 내용들의 마지막 파일명으로 모두 변경되어버리는 문제가 발생했다. 그래서 대안으로 클래스를 이용해 파일을 여러 개 다루는 방식을 선택했다. 이 방법은 인터페이스 제작에 확실한 방법이었지만, 파일명을 띄우기 위해서는 이름을 출력하는 함수를 호출하는 것과 여러 개의 클래스가 만들어진다는 점에서 분명한 단점으로 작용하였다. 실제 인터페이스와 결합 시, 그냥 정지가 아닌 일지정지의 형태로 파일을 읽게 되어, 파일 헤더를 읽어올 수 없었고 모드 변경을 바로 하는 것이 불가능해지는 문제가 발생했다.

2) 문자열 방식

```
void readfile(File dir) {
    root.openNextFile();
    for (int i = 0; i < 16; i++) { //넉넉하게 16개 정도의 파일의 이름을 읽어와 title_list에 파일명을 저장한다
        stage = root.openNextFile(); //파일을 하나씩 연다
        transfer_name2title = stage.name(); //파일명을 일시적으로 갖고 있는다
        strcpy(title_list[i], transfer_name2title); //파일명을 주어진 배열에 넣어준다
    }
```

앞서 말한 클래스의 단점을 극복하기 위해, 문자열을 하나씩 차근차근 저장하는 방식을 선택했다.
이 방법은, 파일명을 일시적으로 저장하는 포인터를 만들고 이를 주어진 배열에 저장하는 방식이다.
이 방법을 사용하면 파일 재생 시, 문자열을 기준으로 파일을 열기 때문에 재생모드 설정이 쉬워진다.

3) 핵심 아이디어

인터페이스를 구성하기 위한 핵심 아이디어는 다음과 같다.

1. 파일명은 배열에 저장된 문자열을 띄운다.
2. 클래스를 사용한 방식은 실행속도 저하와 파일 헤더 읽기의 어려움을 불러온다
3. 문자열을 훼손없이 저장하기 위해서는 포인터에 문자열을 한 번 저장하고 이를 배열에 옮겨주는 작업을 해야 한다.

2. 음원 재생 구현과정

1) 포인터 미사용

포인터를 사용하지 않은 코드는 다음과 같다.

```
ISR(TIMER1_COMPA_vect) {
          if (count > 2999) {
                    a++;
                    count &= 0x00;
          }
          if (a == 2) a = 0;
          OCR2A = (buffer[a][count]);
          OCR2A = ((buffer[a][count + 1] + 0x80) & 0xFF);
          OCR4B = buffer[a][count + 2];
          OCR4A = (buffer[a][count + 3] + 0x80) & 0xFF;

          count +=1;
}
```

이 방식의 문제점은 배열의 인덱스를 2 이상씩 증가시키게 되면 재생이 안 되는 문제가 발생했다.

즉, 음원을 재생하기 위한 시간이 아예 존재하지 않았다는 것을 알 수 있었다. 이를 개선하기 위해 처음에는 전역변수를 직접 다루지 않고 지역변수에 전역변수 값을 담아 이를 ISR에서 처리하고 ISR이 끝날 때, 이를 전역변수에 반환하는 방식을 선택했더니 정상적인 재생이 이루어졌다.

그래도 이 방식은 배열의 성분들을 호출하기 위한 인덱스가 2개가 필요하다는 점에서 비효율적인 코드라고 볼 수 있다. 예를 들어, 0번 버퍼를 읽어오는 중에는 0번은 고정이 되어 있음에도 불필요하게 인덱스 하나를 더 사용해야하는 문제가 발생한다는 것이다. 따라서 교수님께서 설명해주신 포인터를 이용한 방식을 적용하기로 했다.

2) 포인터 사용

포인터를 사용한 방식의 장점 및 효과에 대해 서술하고 이후 핵심 아이디어 내용에 포인터를 사용하기 위한 방법과 아이디어를 적겠다.

포인터를 사용한 결과, 실행속도가 월등히 빨라짐을 알 수 있었다. 하지만 버퍼를 쓰는 타이밍이 안 맞아서 음원이 2배속으로 들리는 문제가 발생했다. 이는 버퍼의 재생상태를 확인하는 변수를 통해 개선이 가능했다.

3) 핵심 아이디어

포인터를 사용한 음원 재생방식의 핵심 정보와 아이디어는 다음과 같다.

1. 2중 배열의 첫 번째 요소는 해당 배열의 첫 번째 주소를 담고 있다
2. 포인터에 버퍼의 주소를 가리키게 만들면 0~2999를 다루는 인덱스만 사용해도 음원 재생이 가능해진다.
3. 결과적으로 변수를 한 개만 사용하고도 모든 버퍼요소를 다룰 수 있게 되는 것이다.

버퍼의 잘못된 재생으로 인한 2배속 문제 해결 아이디어는 다음과 같다.

1. 처음에는 일부러 사용하지 않는 행 인덱스를 계속 남겨 행 인덱스로 버퍼의 재생상태를 구분했다.
2. 행 인덱스 대신 각 버퍼의 상태를 체크하는 플래그와 같은 변수를 2개 만들어준다
3. 플래그와 열 인덱스의 값을 더해 그 값이 인덱스의 끝 값을 넘어가게 되면, 해당 버퍼가 끝난 것으로 인식한다.
4. loop에서는 플래그 변수만을 확인해서 버퍼 사용 여부를 확인한다

3. 인터페이스 + 음원 재생기능 결합

인터페이스와 음원을 결합하기 위해서는 파일 헤더 읽고 모드 설정, 문자열로 표현된 파일명으로 파일 오픈하기, 스위치에서 검출한 인터럽트를 통해 파일 재생/정지 변경하기와 같은 기능들이 추가되어야 한다.

핵심 아이디어는 다음과 같다

1. 파일 헤더는 0~44Byte까지 존재하므로 한 번에 44바이트를 fileheader 배열에 저장한다
2. 배열에 저장되어 있는 정보 중 필요한 부분의 정보만 뽑아 재생 모드를 설정할 수 있도록 한다.
3. 문자열로 표현된 파일명을 SD.open()의 인자로 넣어준다.

➔ 이 부분을 처음에는 클래스를 배열로 만들어 개별적인 파일 클래스를 직접 열어서 모드설정, 음원 재생을 했다. 하지만, 이미 열리고 한 번 읽힌 파일에서 다시 파일 헤더를 읽을 순 없었으므로 문자열을 이용해 파일을 열고, 매번 파일을 새롭게 열어 stage라는 클래스에 저장해서 음원을 재생하는 방식을 선택했다. 이렇게 되면, 항상 파일이 새로 열리기 때문에 파일 헤더를 항상 읽어올 수 있는 장점이 있다.

4. 스위치에서 falling edge를 검출해서 정지모드로 들어갈 때, encoder enable, timer interrupt를 disable 시키고 모드 변경 여부를 확인하는 변수 choice를 0으로 만들어줘서 일시정지 시, 한 번 설정한 내용을 여러 번 할 반복할 필요 없게 만들었다.
5. 음원 재생도 마찬가지로 파일 재생 시, 파일 헤더를 읽고 모드 변경하는 코드의 수행을 재생 중에는 한 번만 실행되도록 만들기 위해 설정 변경 여부를 확인하는 변수 change를 0으로 바꾸고

재생모드로 들어간다.

6. 음악이 다 끝났을 때 정지모드 진입은 stage,available()이 0일 때 일시정지 확인 변수 play_stae를 0으로 만들어 주면 된다.

4. 결과

WAV File player의 완성본은 16-bit stereo 22.05kHz 까지 정상적으로 재생하는 모습을 확인할 수 있었다. 16-bit stereo 44.1kHz 음원의 재생은 프로그램 실행속도의 문제로 음원이 늘어지는 모습을 확인할 수 있었다. 이 문제점은 불필요한 작업을 제거하거나, SD library의 버전을 좀 구버전으로 맞추게 되면 정상 작동할 것으로 보인다. 왜냐하면, SD Library의 경우 고버전의 경우 파일 읽어오는 과정이 좀 더 느리다고 알려져 있기 때문이다.

마지막으로 이번 프로젝트를 하면서 일상생활에서 흔히 보는 player를 만들기 위해서 많은 기능들이 동원되는 것과 Arduino MEGA 2560의 성능이 생각했던 것 보다 그렇게 좋지 않다는 것을 알 수 있었다. 처음으로 제대로 된 작품을 만들어보는 프로젝트를 진행했는데, 앞으로도 이런 특별한 작품을 만들어 보는 프로젝트를 하는 방법, 개발에서 사용해야 할 Divide & Conquer를 어떻게 하는지를 배울 수 있었다.