|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **결 과 보 고 서** | | | | |
| 학 과 | 학 년 | 학 번 | 조 | 성 명 |
| 전자공학과 | 3 | 12191505 | - | 윤수연 |
| 실험 제목 | Buzzer | | | |
| 1. **실험 과정**    1. 실습 1단계  |  | | --- | | #include<avr/io.h>  #include<util/delay.h>  int main(void){    unsigned char key;    DDRA = 0b10000000;  DDRE = 0b00000000;  PORTE = 0b11111111;  while(1) {  key = PINE;    switch (key) {  case 0b11111110: // 도  PORTA = 0b10000000;  \_delay\_us(1908\*4);  PORTA = 0b00000000;  \_delay\_us(1908\*4);  break;    case 0b11111101: // 레  PORTA = 0b10000000;  \_delay\_us(1700\*4);  PORTA = 0b00000000;  \_delay\_us(1700\*4);  break;  case 0b11111011: // 미  PORTA = 0b10000000;  \_delay\_us(1515\*4);  PORTA = 0b00000000;  \_delay\_us(1515\*4);  break;    case 0b11110111: // 파  PORTA = 0b10000000;  \_delay\_us(1432\*4);  PORTA = 0b00000000;  \_delay\_us(1432\*4);  break;    case 0b11101111: // 솔  PORTA = 0b10000000;  \_delay\_us(1275\*4);  PORTA = 0b00000000;  \_delay\_us(1275\*4);  break;    case 0b11011111: // 라  PORTA = 0b10000000;  \_delay\_us(1136\*4);  PORTA = 0b00000000;  \_delay\_us(1136\*4);  break;  case 0b10111111: // 시  PORTA = 0b10000000;  \_delay\_us(1012\*4);  PORTA = 0b00000000;  \_delay\_us(1012\*4);  break;    case 0b01111111: // 도  PORTA = 0b10000000;  \_delay\_us(954\*4);  PORTA = 0b00000000;  \_delay\_us(954\*4);  break;    default:  break;  }  }  } |       그림 1. 실습 1단계 회로 구성도  실습 1단계는 8개의 버튼을 이용해 도레미파솔라시도(높은 도)를 구현하는 것이다. 각 음계는 해당하는 주파수를 갖고 있기 때문에 각 음에 따라 다른 펄스를 sounder와 연결된 핀으로 출력해주면 해당하는 음을 들을 수 있을 것이다.  그래서 펄스를 만든 후 sounder에 넣을 줄 알았는데 Delay를 이용하여 물리적으로 펄스파를 구현한 것이 인상적이었다. delay기간 동안 low나 high를 번갈아 인가하여 조정하는 것이다. Delay를 인가하는 방법은 음률의 기준이 되는 라 음을 기준으로 아래와 같다.    그림 2. 실습 1단계 delay 계산 과정  각 음에 해당하는 주파수의 역수가 바로 주기가 되기 때문에 그렇게 delay를 구하면 되는데, 이 때 펄스가 두 번 toggle 되어야 하나의 주기가 완성되기 때문에 2를 한 번 더 나누어 준다. 조금 의아한 부분은 delay를 줄 때 \*4를 했다는 것이다. 아마도 주기를 길게 하여 주파수를 낮게 하려고 하는 의도인 것 같다. 주파수에 2를 곱하면 한 옥타브가 올라가기 때문이다.  그런 식으로 모든 음에 대해 delay를 계산하여 인가해준다. 이 때 사용한 음계는 피타고라스 음계다.   * 1. 실습 2단계  |  | | --- | | #define F\_CPU 16000000UL  #include<avr/interrupt.h>  #include<util/delay.h>  #include<avr/io.h>  #include<util/delay.h>  unsigned char key; //button input  int main(void){    //init TC  TCCR2 = 0x00; //off  TIMSK = 0b10000000; //OVF x OCF o  TCNT2 = 0;  OCR2 = 0;    SREG = 0x80;    //init pin  DDRA = 0b10000000;  DDRE = 0x00;  PORTE = 0xFF;  DDRB = 0xFF;  while(1){  key = PINE;  switch(key){  case 0b11111110: // 도  TCCR2 = 0b00011101; //CTC, toggle,분주비 1024  OCR2 = 119;  \_delay\_ms(1000);  break;    case 0b11111101: // re  TCCR2 = 0b00011101; //CTC, toggle,분주비 1024  OCR2 = 106;  \_delay\_ms(1000);  break;    case 0b11111011: // mi  TCCR2 = 0b00011101; //CTC, toggle,분주비 1024  OCR2 = 95;  \_delay\_ms(1000);  break;    case 0b11110111: // pa  TCCR2 = 0b00011101; //CTC, toggle,분주비 1024  OCR2 = 89;  \_delay\_ms(1000);  break;    case 0b11101111: // sol  TCCR2 = 0b00011101; //CTC, toggle,분주비 1024  OCR2 = 80;  \_delay\_ms(1000);  break;    case 0b11011111: // ra  TCCR2 = 0b00011101; //CTC, toggle,분주비 1024  OCR2 = 70;  \_delay\_ms(1000);  break;    case 0b10111111: // si  TCCR2 = 0b00011101; //CTC, toggle,분주비 1024  OCR2 = 63;  \_delay\_ms(1000);  break;    case 0b01111111: // do  TCCR2 = 0b00011101; //CTC, toggle,분주비 1024  OCR2 = 60;  \_delay\_ms(1000);  break;    default :  break;  }  TCCR2 = 0x00; //off  OCR2 = 0; //init  }  } |   실습 2단계는 1단계와 같은 동작을 Timer/Counter의 CTC동작 모드로 구현하는 것이다. CTC모드에서는 펄스를 만드는 동작에서 비교일치 인터럽트만을 사용하기 때문에 OCR의 값을 잘 조절해주는 것이 관건이었다. 원하는 주파수의 펄스 파형을 만들기 위해 먼저 아래와 같이 계산을 진행했다.    그림 3,4. 실습 2단계 OCR2값 계산 과정  위에서 구한 값을 기반으로 코드를 살펴보면 Timer/Counter2를 사용하여 TCCR2등 레지스터를 의도와 맞게 설정했다. 큰 알고리즘은 입력 버튼을 연결한 port E의 pin을 읽어 들이고 이에 맞추어 펄스의 주파수를 변경시키는 것이다. 입력을 받을 때 main함수 안의 while 반복문에서 switch문을 사용했다는 점에서 실험 1단계와 유사하다.  또한 원하지 않을 때 소리가 나는 것을 방지하기 위해서 사용하지 않을 때는 Timer/couner2의 동작을 off해야 했는데 TCCR2의 cs bit를 모두 0으로 만들고, 필요할 때 레지스터에 원하는 동작에 해당하는 값 0b00011101을 넣는 것으로 해결했다.  한편 하드웨어 적으로는 실험 2단계에서 Timer/Counter2를 사용했기 때문에 OC2 출력 핀을 사용해야했다. 이는 PORT B의 7번 핀을 말하며, 해당 핀에도 여러 이름 중 하나로 표시되어 있다. 회로 구성도를 보면 그래서 sounder의 위치를 해당 핀으로 조정한 것을 볼 수 있다.    그림 5. 실험 2 회로 구성도   1. **실험 결과**    1. 실습 1단계   [13주차 1단계 실험 결과](https://www.youtube.com/embed/PtrWkD3xfDk?feature=oembed)  그림 6. 실험 1 실행 결과   * 1. 실습 2단계   [13주차 2단계 결과](https://www.youtube.com/embed/j9A1F8SPJHE?feature=oembed)  그림 7. 실험 2 실행 결과   1. **고찰**   이번 주차에는 처음 사용하는 소자인 sounder를 통해 소리를 출력하는 주차였다. 이번 주차 2단계는 강의 노트에 그저 CTC모드를 사용하라고만 적혀 있어서 처음으로 조교님의 실험 동작을 봐도 어떻게 구현할 지 바로 머리에 떠오르지 않은 문제다. 그래서 처음에 CTC모드로 delay를 만들고 1단계와 알고리즘은 동일하게 코드를 구현했으나 이후 CTC모드와 PWM모드를 다시 공부하고 나서 CTC모드는 펄스 출력에 특화된 모드라는 것을 깨닫았다. 이를 바탕으로 펄스를 직접 만들어 sounder에 인가하여 문제를 제대로 해결할 수 있었다.  실험 1번 결과 영상에서 다시 눌러보는 음은 다른 음들과 길이가 많이 차이가 있다고 생각될 때 다시 한번 음을 들어보았다. Buzzer는 1단계와 2단계 모두 button으로 조작되는데, button이 눌리는 것이 click의 미세한 차이도 감지하는 것인지, 똑 같은 길이로 click했다고 생각했는데도 불구하고 음의 길이는 제각각이다. 아마 button을 눌러 회로가 short되는 순간에만 clock이 buzzer에 인가되기 때문에 실제 소자로 실험을 진행한다고 하더라도 이 정도의 차이는 있을 수 있다고 생각된다. 인터럽트를 사용하여 구현했더라면 더욱 정확한 동작을 볼 수 있었을 것이라고 생각이 든다.  2단계 실험도 비슷한 이유로 처음에 동작이 원활하지 않았다. 그 이유는 바로 OCR2의 초기화를 동작이 끝난 이후에 해주지 않아서 그런 것이었는데, 이후 고치고 나서 원하는 동작을 확인할 수 있었다. 하지만 인터럽트를 사용하지 않아 button동작이 원활하지 않다는 것은 느낄 수 있었다. 이는 button동작에 따른 delay를 넣어줬기 때문에 어쩔 수 없는 동작이다.  그리고 2단계 코드 작성에서 구한 OCR2값을 각 음계 순서에 맞게 array로 정리하면 코드의 readability가 올라갈 것이라고 생각했지만 array의 index를 표시하는 변수를 추가해야 해서 오히려 복잡해 보였다. 따라서 현재 코드로 실험을 진행했다.  이번 주차의 실험 결과는 소리를 듣는 것이기 때문에 워드 결과 보고서로는 결과를 확인하기가 쉽지 않다. 따라서 아래 링크에 실험 결과를 게시했다.  1단계 실험 결과: <https://youtu.be/PtrWkD3xfDk>  2단계 실험 결과: https://youtu.be/j9A1F8SPJHE | | | | |
|  | | | | |