|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **결 과 보 고 서** | | | | |
| 학 과 | 학 년 | 학 번 | 조 | 성 명 |
| 전자공학과 | 3 | 12191505 | - | 윤수연 |
| 실험 제목 | 리듬 게임기 | | | |
| 1. **실험 과정**    1. **1단계**     그림 1. 회로 구성도  위의 회로 구성도는 계획을 세울 때는 완성도를 높여 구현하면 꽤 재밌는 리듬게임을 만들 수 있을 것이라고 생각했으나 실제로 도전해보니 전혀 아니었다. 생각한 것과 같이 구현했으나 애초에 설계 상 완성도가 떨어지는 부분이 많았다. 아래가 바로 그 단점 및 문제점을 나름대로 분석한 결과다.   * + 1. **조작 부분**   플레이어가 직접 조작하는 부분으로, 네 개의 button이 해당된다. 사용자의 동선 효율을 위해 버튼 소자끼리 최대한 붙여 놨지만 누르기가 매우 불편하다.  게다가 편하게 구현하기 위해 이제껏 입력을 위해 가장 많이 사용된 button소자를 사용한 것이지만 버튼의 번호를 화면에 나타내고 그 번호에 해당하는 버튼을 누르는 것은 굉장히 직관적이지 않아 재미도 없었다. 이에 버튼의 위치를 반영하여 방향키 등으로 바꿔보려 했으나 아스키 코드에 지원하지 않는 특수 문자였다.   * + 1. **화면**   플레이어가 보는 화면으로, 2line을 사용하는 LCD를 사용했다. 우선 두 개의 라인만 사용하기 때문에 담을 수 있는 정보량이 매우 작다. 게다가 한 line에 한 시간대에 표시할 수 있는 글자 수도 너무나 제한적이다.  이에, 첫번째 줄에는 리듬 게임인 만큼 남은 시간을 표시하고 두 번째 줄에는 플레이어가 누를 버튼 정보들, 즉 꼭 필요한 내용만을 담을 것이라고 계획했다. 하지만 앞서 말한 것처럼 버튼 특수문자가 담기지 않아 다른 정보를 담기에도 턱없이 불충분했다. 따라서 플레이 중 시간 정보와 게임 내용 외에 스코어 등 다른 정보를 전혀 담을 수 없었고 자연스럽게 정적인 느낌이 들었다. 리듬 게임인데 게임이 정적으로 느껴진다.   * + 1. **시간 설정**   Timer/Counter 기능을 이용하여 남은 시간을 나타내는 알고리즘이다. 숫자로 나타내는 것은 직관적이지 못하다고 생각하여 남은 시간을 특수문자로 화면을 채워 넣듯 출력하여 readability를 올리려고 했다. 하지만 단언컨데 이 리듬 게임기가 재미없는 가장 큰 이유라고 생각한다.  우선 매끄러운 시간을 표시하기는 너무 힘들다. 이 시간설정의 구체적인 알고리즘은 남은 시간을 저장하는 전역변수를 사용하는 것이다. 그 다음 매번 반복문을 돌려 남은 시간만큼 길이의 문자열을 새로 만든 다음에 이를 다시 디스플레이 하는 방식이었다. 다소 비효율적이긴 하지만 다른 알고리즘을 사용했을 때 오류가 너무 많이 나서 이렇게 구현했다. 이 알고리즘을 사용하면 비록 가상환경에서 하드웨어를 구현하는 프로테우스 프로그램을 사용했지만 그래도 딜레이가 꽤 있기 때문에 시간 표현이 전혀 매끄럽지 않았다.   * 1. **2단계**   2단계는 리듬 게임이 전혀 리듬감이 느껴지지 않고 button을 이용한 input 및 lcd를 이용한 게임 정보 output 방법에 문제가 아주 많다고 느껴 사용자와의 교류방법을 인터럽트가 아닌 USART를 이용하여 해결하기로 했다. 이렇게 하면 마치 타자 연습을 하듯이 키보드로도 리듬게임이 가능해져서 보다 편하게 플레이어가 게임을 즐길 수 있을 것이라고 생각했다. 이를 위해 USART를 다루었던 3주차 실험에서 사용했던 PuTTy 등 프로그램을 추가적으로 이용하는 것이다. 하지만 이 역시도 문제가 아주 많았다.   1. **프로그램 미숙**   USART는 해당 주차의 실험을 진행할 때도 가장 어려웠고 시간을 많이 투자했던 주차였다. 그런데 PuTTy를 이용하여 리듬게임을 만드려는 이번 7주차가 가장 어려웠던 주차로 갱신되었다. 이렇게 어려울 거면 끝내 성공하기라도 하면 좋으려만, 끝까지 통신이 잘 이루어지지 않았다. 가상 port가 연결된 것 같기는 한데 함수가 제대로 동작하지 않았다. 오랜 시간을 투자했지만 끝내 오류의 원인을 찾을 수 없었다.   1. **여전히 존재하는 문제점**   이번 단계는 아예 성공을 하지 못해서 플레이를 해볼 수는 없었지만 만약 된다고 했더라도 1단계에서 미처 극복하지 못한 점이 있다. 바로 시간 설정 문제다. Input과 Output관련된 문제는 USART를 통해서 거의 모두 극복할 수 있다지만 시간 설정의 경우 1단계와 알고리즘이 동일하여 코드도 같이 가져갔다. 따라서 시간 설정이 전혀 개선되지 않았으며 추가로 LCD관련한 문제도 남아있었다. 여전히 lcd를 사용한 디스플레이로 표시 가능한 양보다 정보량이 한참 많아지며 핀을 너무 많이 점유하게 되긴 하지만 lcd를 두 개 쓰는 방향으로도 생각해보았다.     * 1. **3단계**   깊은 좌절감과 함께 결국 USART를 사용하여 리듬 게임기를 만드는 것을 포기하게 되었다. 하지만 간단한 MCU인 ATmega128지만 충분히 리듬게임을 만들 수 있는데 내가 잘못하는 거라는 생각이 자꾸 발목을 잡았다. 나는 5주차를 제외하고는 Proteus안에서만 다뤄 보기는 했지만 기능이 이렇게 많은데 잘 작동이 안되는 USART를 사용하지 않고서도 충분히 구현할 수 있을 것만 같았다. 그래서 아래와 같은 아이디어를 다시 한번 내 보았다.    그림 2. 새로운 회로도   1. **LED표시 방식**   가장 많이 바뀐 부분은 바로 LED가 추가되어 실제 플레이어가 이 부분으로 게임을 control한다는 점이다. LED는 여기서 총 8개를 한 줄로 세워 배치했는데 모두 켜진 상황이 디폴트 상태이고 하나씩 불이 꺼져 그 꺼진 부분이 아래로 내려오는데 지정한 위치에서 버튼을 눌러 동작을 멈추는 방식이다. 아래로 내려온 꺼진 불은 버튼을 누르지 않아도 다시 위에서 내려오는 동작을 무한 반복한다.  이렇게 할 경우 리듬을 더 잘 느낄 수 있고 시간 설정이 따로 필요없이 직관적으로 플레이할 수 있다. 타이밍에 맞게 눌러야 하는 버튼이 하나기 때문에 보다 간결한 플레이도 마음에 든다.  목표 타겟은 power가 바로 연결된 7번 LED다. 7번 LED에서 버튼을 눌렀을 경우에만 perfect등급을 받는다. 이 전후로 6번과 8번 LED까지는 good등급을 받고 그 외에는 miss판정이다. 이러한 결과는 아래에서 소개할 LCD에 담긴다.  LED의 색은 가독성을 위해 가장 눈에 잘 띄는 색으로 골랐는데, 이제껏 사용해보지 못한 pink 컬러를 사용했다.    그림 3,4. 새로운 게임 방식. LED(좌)와 button(우)   1. **LCD**   기존 아이디어의 큰 문젯거리였던 정보량이 LCD의 표시 영역에 비해 너무 많다는 것은 LED를 새로 도입하며 사라졌다. LED를 사용하면 LCD를 두 개 사용하는 것보다 핀을 적게 사용할 수 있다. 또한 창을 두 개 띄우는 것보다 한 화면에 꼭 필요한 정보만 적는 것이 플레이에 더 효과적일 것이다.    그림 5. LCD   1. **난이도 조절**   실습 5주차에서 우리는 LED의 속도를 두 개의 버튼으로 제어했다. 이를 이용하여 플레이의 속도를 control할 수 있고 이는 게임의 난이도 조정으로 볼 수 있다.    그림 6. 난이도 조절 버튼   1. **게임 재개**   한 번의 플레이가 끝나더라도 다시 플레이 버튼을 눌러주면 게임이 다시 시작된다. LED가 고작 한 열이기도 하고 다소 단순한 방식이다. 여러 번의 피드백을 거치며 오히려 아주 간단한 형태가 된 것이 재미있다.   1. **실험 결과**    1. 코드  |  | | --- | | #include <util/delay.h>  #include <avr/io.h>  #include <avr/interrupt.h>  #include <stdio.h>  #include <stdlib.h>  //constants  unsigned char led\_on;  unsigned int delay;  unsigned int state;  volatile unsigned int hour, min, sec = 0;  volatile unsigned int count=0;  volatile char str1[1][16] = {{"Game Start"}};  volatile char str2[1][16] = {{"Press Button"}};  volatile char str3[1][16] = {{"Perfect!!!!!!!!!"}};  volatile char str4[1][16] = {{"Good!!!!!!!"}};  volatile char str5[1][16] = {{"Miss!!!!!!!!!!!"}};  //fuction declaration about LCD  void cmd(int command);  void data(char str);  void display1(char\*str);  void score(unsigned int x);  void my\_delay(int t){  while(t>0){  \_delay\_ms(10);  t -=10;  }  }  void score(unsigned int x){  if(x == 0b00000010){  display2(str3[0]);  \_delay\_ms(500);  }  else if(x==0b00000100 | x == 0b00000001){  display2(str4[0]);  \_delay\_ms(500);  }  else{  display2(str5[0]);  \_delay\_ms(500);  }  }  //interrupt service routine  ISR(INT0\_vect){  if(led\_on){  unsigned int x = PORTC;  score(x);  //PORTC = 0b10000000;  led\_on = 0;  }  else{  //PORTC = 0b00000000;  led\_on=1;  }  }  ISR(INT1\_vect){  if(delay<2500){ //upper limit 2500  delay += 40;  }  }  ISR(INT2\_vect){  if(delay>0){ //lower limit 0  delay-= 40;  }  }  //main  int main()  {  DDRA = 0xFF; //output LED  DDRB = 0xff; //output LCD  DDRC = 0xFF; //output LCD  DDRD = 0x00; //input button  PORTD = 0xFF; //PUD //0b00000111;    EICRA = 0b10101010;  EIMSK = 0b00000111;    TCCR2 = 0b00000101; //분주비 1024  TCNT2 = 0; //Timer/Counter init zero  SREG |= 0x80; //all interrupt allow    cmd(0b00111100); // set 8bit 2line 4\*10dot  cmd(0x01); // display clear  cmd(0x80); // set DDRAM address or cursor position on display  cmd(0x0c); // display on cursor off  cmd(0x06); // entry mode    delay = 100;  led\_on = 1;  state = 0b00000001;  display1(str1[0]);  display2(str2[0]);  while(1){  if(led\_on){  if(state == 0b00000000){  state = 0b00000001;  }  PORTC = state;  my\_delay(delay);  state = (state<<1);  my\_delay(delay);  }  else if (led\_on==00){  PORTC = 0x00;  }  }  }    void cmd(int command) {  PORTB = command;  PORTA = 0x04;  \_delay\_ms(1);  PORTA = 0x00;  }  void data(char str) {  PORTB = str;  PORTA = 0x05;  \_delay\_ms(1);  PORTA = 0x01;  }  // LCD 첫번째 줄  void display1(char \*str) {  cmd(0x80);  int i = 0;  while(str[i] != '\0') {  data(str[i++]);  }  }  // LCD 두번째 줄  void display2(char \*str) {  cmd(0b11000000);  int i = 0;  while(str[i] != '\0') {  data(str[i++]);  }  }  ///////////////////////////////////////////////// |   LCD와 button입력, 인터럽트 그리고 LED동작에 관한 이야기는 전에 다루었던 정도의 난이도기도 하고 너무 길어질 것 같아 설명을 생략하겠다. 가장 중요한 함수는 score함수다. 이를 이용하여 알고리즘에 따라 플레이어의 플레이를 평가한다.   * 1. 실험 결과     그림 7. 플레이 LED 부분    그림 8. 플레이 시작    그림 9. 플레이 중 화면1    그림 10. 플레이 중 화면 2   1. **고찰**   이번 주차에는 자율 프로젝트로 하고 싶은 주제로 프로젝트를 진행하는 주차였다. 물론 자율인 만큼 부담도 크고 여러가지 제약이 없는 만큼 의욕이 앞섰던 주차였다. 사실 오프라인이었으면 7세그먼트를 동적구동 방식을 이용하여 4개의 7세그먼트를 만들어 시계를 만든 다음 집에서 요리할 때 사용했을 것 같다. 그렇지만 실물 소자도 없고 또 다른 아이디어도 많았기 때문에 한 번쯤 만들어보고 싶었던 미니게임을 만들었다. 하지만 완성도가 기대보다 많이 떨어진다는 점에서 시계를 만들지 못한 것 외에도 아쉬움과 미련이 남는 주차다.  개선점으로는 이번에는 한 LED열 (8개가 한 set)을 사용했지만 이를 여러 줄로 한다면 보다 리듬감이 느껴질 것 같다. 굳이 열이 더 길 필요도 없을 것 같아서 열의 길이를 약간 줄이고 동적 구동으로 여러 개의 열을 구동한다면 핀의 손실도 거의 없을 것이라고 생각된다. 이렇게 여러 줄로 구현할 경우 여러 조합이 생길 수 있기 때문에 random하게 플레이 할 수 있도록 함수를 사용하여 구현해도 재밌을 것 같다.  또한 누적으로 점수를 계산하여 최고기록을 저장하는 등의 기능을 추가하면 더 좋을 것 같다. MCU의 EEPROM등에 저장하면 프로그램의 실행과 관계없이 변수가 저장되니까 이를 이용하면 될 것 같다.  또한 실제로 플레이 해보면서 느낀 점이 처음 게임을 접하는 사람을 위한 튜토리얼이 있었으면 좋겠고, 생각보다 난이도 조절 버튼은 사용할 일이 없다는 것이었다. 비록 완벽하지는 않지만 사실 플레이에 지장이 있는 건 아니었기에 나름 만족하고재미있는 주차였다. | | | | |
|  | | | | |