|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **결 과 보 고 서** | | | | |
| 학 과 | 학 년 | 학 번 | 조 | 성 명 |
| 전자공학과 | 3 | 12191505 | - | 윤수연 |
| 실험 제목 | ADC | | | |
| 1. **실험 과정**    1. 실습 1단계  |  | | --- | | #define F\_CPU 16000000  #include <inttypes.h>  #include <avr/io.h>  #include <util/delay.h>  #include <avr/interrupt.h>  #include <avr/sleep.h>  #include <stdio.h>  void cmd(int);  void data(char);  void display1(char\*);  void display2(char\*);  char str[2][16] = {{"ADC VALUE :"},{" "}};  void ADC\_SingleConversion\_Init(unsigned char port) {  ADMUX = port; //ADMUX값을 설정  }  int ADC\_SingleConversion\_read() {  int output;  ADCSRA |= (1<<ADSC) | (1<<ADEN) ; //Start single conversion  while(ADCSRA & (1<<ADSC)); //wait Conversion to complete  output = ADCL + (ADCH << 8);  ADCSRA &= ~(1<<ADEN); //disable ADC  return output;  }  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;  }  void display1(char \*str){  cmd(0x80);  int i =0;  while(str[i]!='\0'){  data(str[i++]);  }  }  void display2(char \*str){  cmd(0xc0);  int i =0;  while(str[i]!='\0'){  data(str[i++]);  }  }  int main(){  int ADC\_Output;  float volt;  DDRA = 0xff; // set port A as output  DDRB = 0xff; // set port B as output  DDRF = 0xfe; // set port F as intput  EICRA = 0xff;  EIMSK = 0x0f;  SREG = 0x80;  cmd(0x38);  cmd(0x01);  cmd(0x80);  cmd(0x0c);  cmd(0x06);  //위 내용은 LCD강의 참고  while (1){  display1(str[0]); // 1행 출력  display2(str[1]); // 2행 출력  ADC\_SingleConversion\_Init(0x00); // ADC0 설정  ADC\_Output = ADC\_SingleConversion\_read(); // 변환된 digtal 값 읽기  volt = 5.\*(ADC\_Output/1023.); // 가변저항으로 얻은 전압값 계산    sprintf(str[1],"%d.%d%d%dV",((int)volt),((int)(volt\*10))%10,((int)(volt\*100))%10,((int)(volt\*1000))%10); // 전압 값 출력  \_delay\_ms(10);  }  return 0;  } |       그림 . 실습 1단계 회로 구성도  1단계 실험은 ADC port를 이용해 가변 저항에 걸린 전압을 읽어, LCD에 출력하는 것이다. 이전에 자주 사용했던 LCD 함수는 익숙하게 구현할 수 있었고 이와 비슷하게 ADC init 함수로 ADMUX를 조절하여 어느 ADC 핀을 입력으로 사용할 것인지, 기준 전압을 선택했다.  ADC read 함수도 별도로 정의하여, ADCRA를 조절하여 입력을 읽어 정수 출력하여 ADCH,L에 저장된 0~1023 사이의 값을 얻을 수 있다. 이를 위해서 함수 내부에서는 특정하게 이름 지어진 레지스터를 on시키고 off가 될 때까지 기다렸다가 이후에 값이 저장된 레지스터에 접근하는 방식이다. 마치 USART의 단계와 비슷하다.  가변 저항을 통해 읽은 값에, AREF를 5V로 하였고, 가변 저항에도 5V 전원과 GROUND를 연결하여 읽은 값에 5/1023을 곱해주면 가변 저항의 중앙 pin의 전압 값을 알 수 있다. 이것이 ADC의 과정이다. 이 후에는 전압 값을 LCD에 출력하기 위하여 sprintf를 이용하여 문자열을 만들어주고 이를 출력한다.  가변저항은 이번에 PROTEUS에서 처음 사용해본 소자다. POT-HD로 검색하여 소자를 사용할 수 있었는데, 실제로는 저항 값을 돌려가며 조절하는데 PROTEUS에서는 아래 그림과 같이 제어할 수 있다. 마우스로 저항 안쪽을 클릭하면 해당 마우스의 위치에 맞게 화살표가 저항 측면에 찍히고 그 위치에 따라 저항의 크기가 결정된다. 본 실험에서 사용한 저항이 1K옴이었는데, 그 중 저항의 유효 크기가 퍼센트로 표시되는 것을 관찰할 수 있다.    그림 2. 가변 저항 제어 모습   * 1. 실습 2단계  |  | | --- | | #define F\_CPU 16000000  #include <inttypes.h>  #include <avr/io.h>  #include <avr/interrupt.h>  #include <avr/sleep.h>  #include <util/delay.h>  #include <stdio.h>  void ADC\_SingleConversion\_Init(unsigned char port){  ADMUX = port; // ADMUX값을 설정  }  int ADC\_SingleConversion\_read() {  int output;  ADCSRA |= (1<<ADSC) | (1<<ADEN); // start single conversion  while(ADCSRA & (1<<ADSC)); // wait conversion to complete  output = ADCL + (ADCH<<8);  ADCSRA &= ~(1<<ADEN); // disable ADC  return output;  }  void printchar(char ch){  while( !(UCSR0A & (1 << UDRE0)));  UDR0 = ch;;  }    int main()  {  DDRF = 0xf8; //11101010 0xf8  PORTF = 0x07; //10011111;    DDRE = 0x02;  // add  ADMUX = 0x00;  ADCSRA = 0b11000000;    UCSR0A=0x00;  UCSR0B=0b11111000; //10011000  UCSR0C=0b10000110; //10000110    UBRR0H=0;  UBRR0L=103;  SREG=0x80;    int i;  int x, y, z;  int putty[100];    while (1){  ADC\_SingleConversion\_Init(0x00);  x = ADC\_SingleConversion\_read();    ADC\_SingleConversion\_Init(0x01);  y = ADC\_SingleConversion\_read();    ADC\_SingleConversion\_Init(0x02);  z = ADC\_SingleConversion\_read();    sprintf(putty, "X = %4d , Y = %4d, Z = %4d", x, y, z);  \_delay\_ms(10);  for(i=0;i<100;i++){  while((UCSR0A&0b00100000)==0x00){continue;}  UDR0 = putty[i];  }  // printchar('\n\r');  printchar('\n');  printchar('\r');    }  return 0;  } |   2단계 실험은 가속도 센서를 이용하여 x, y, z축에 따라 달라지는 값을 읽어내어 usart 통신을 통해 terminal 상으로 출력하는 실험이었다. 원래는 가속도 센서에 핀을 연결하여 입력을 읽는데 PROTEUS에서 가상 하드웨어 실험하기에는 무리가 있어서 실습 1단계와 같이 가변 저항을 이용했다.  그래서 다소 코드에 유사한 면이 있다. 입력을 읽을　때는 실습 1번에서 사용했던 ADC init 함수와 ADC read 함수를 통해 읽어냈으며, 다른 점은 오직 가변저항의 개수가 3차원의 가속도 센서를 모방하기 위해 1개에서 3개로 늘어났다는 점과, 출력을 우리에게 익숙한 LCD가 아닌 USART TERMINAL을 통해 관찰한다는 점이다. 가변 저항의 개수가 3개로 늘어난 것은 sprintf를 통해 어렵지 않게 3개의 입력을 하나의 문자열로 저장하였다. 이 때 4의 자리 수를 유지하기 위해서 평소와 달리 %d 대신에 %4d를 사용했다. 또한 여러 번 사용한 USART 통신을 위해 사용하던 함수를 이용하여 저장한 문자열을 출력했다.  처음 시뮬레이션을 돌렸을 때, 값은 무언가 쓰이지만 정확히 원한 글자도 아니고 terminal 상에서 줄바꿈도 이루어지지 않는 등 출력이 이상했다. 여러 시행착오 끝에 중간에 delay를 넣어주고 이스케이프 문자열을 두 개로 따로 적어서 해결할 수 있었다.    그림 3. 실험 2 회로 구성도    그림 4. 실험 2 예상 결과도    그림 5. 실험 2 가상 포트 설정    그림 6. 실험 2 ATmega128 설정    그림 7. 실험 2 COMPIM 설정   1. **실험 결과**    1. 실습 1단계     그림 8. 실험 1 실행 결과1    그림 9. 실험 1 실행 결과2   * 1. 실습 2단계     그림 10. 실험 2 실행 직후(puTTy사용)    그림 11. 실험 2 실행 직후(Hypertrm 사용)   1. **고찰**   이번 주차에는 ADC를 사용하여 여러 센서의 값을 읽어보는 실험을 진행했다. 다만 온라인 수업으로 진행되어 여러 센서를 사용해보지는 못했다. 가상 하드웨어 시뮬레이션 공간에서 가변저항을 사용해보긴 했지만 좀 더 자주 쓰이는 센서를 사용해 볼 수 있기를 기대했는데, 아쉬웠다.  다만 전자공학실험2를 배우면서 계속 아픈 손가락이었던 USART를 조금이나마 극복한 것 같은 역사적인 주차였다. 이제까지 USART를 이용한 통신 개념이 나오면 겁부터 먹었는데 이번 주차 2단계 실험에서는 아주 빠르게 USART를 성공시켰다. 가상 프로그램은 여전히 PuTTy가 아닌 HYPERTERM을 사용했다. 다만 가변저항의 값을 읽어오는 과정에서 오류가 있었다. 그래서 결국 이번 주차에서 오랜 시도에도 만족할 만한 결과가 나오지 않았다. 실험을 완료하기 위해 아주 많은 시간을 투자했음에도 실패하여 더욱 아쉬운 것 같다.  코드 작성에 있어서는, ADC자체로도 두 개의 함수가 구현되어 있어 다소 알고리즘이 복잡하다고 생각한다. 그래서 처음 2단계의 코드를 작성할 때 무작정 모든 것을 함수화 했더니 오히려 복잡해서 디버깅이 어려워지고 내가 짠 코드지만 이해하기 어려웠다.  그래서 1단계의 제공코드를 참고하여 다시 간결하게 코드를 작성한 것이 위의 코드다. 하지만 가변저항의 변화된 값이 통신에 제대로 사용되지 않는 것을 확인하며 이번 주차의 실험을 아쉽게 마무리한다. | | | | |
|  | | | | |