|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **결 과 보 고 서** | | | | |
| 학 과 | 학 년 | 학 번 | 조 | 성 명 |
| 전자공학과 | 3 | 12191505 | - | 윤수연 |
| 실험 제목 | PWM, Servo Motor | | | |
| 1. **실험 과정**    1. 실습 1단계  |  | | --- | | #define F\_CPU 16000000    #include <inttypes.h>  #include <avr/io.h>  #include <avr/interrupt.h>  #include <avr/sleep.h>  //CTC mode  volatile unsigned char led = 0x0f; // origin led  volatile unsigned int count = 0;  ISR(TIMER2\_COMP\_vect){  count ++;  if (count ==5){  PORTF = (~PORTF);  count = 0;  }  }  int main() {  DDRF = 0XFF;  TCCR2 = 0b00001101;  TIMSK = 0b10000000;  OCR2 = 124;  TCNT2 = 0;  SREG = 0x80;  PORTF = led;    while (1){}    return 0;  } |       그림 . 실습 1단계 회로 구성도  실습 1단계는 PWM을 이용하여 8개의 LED를 절반씩 반복하여 켜지게 하는 단계였다. 오프라인에서 실제 소자로 구현하는 것과 proteus로 가상 하드웨어를 사용하는 것이 차이가 나서, 실제 1초를 proteus에서 구현하기 위해 ISR의 if구분에서 count값의 범위를 5로 대폭 줄였다.  코드는 역시 주어졌는데 덕분에 여러가지 복잡한 계산을 피할 수 있었다.   * 1. 실습 2단계  |  | | --- | | #define F\_CPU 16000000UL  #define BAUD 9600  #define MYUBRR F\_CPU/16/BAUD -1  #include <avr/io.h>  #include <util/delay.h>  #define ENTER '\r' //for Linux  #define CR 0x0D  #define LF 0x0A  volatile unsigned char ch;  char raddc[4]; //save input  int radi; //data type: int  int k = 0; //for index  int new\_ocr; //for update ocr  char prompt[8] = "INPUT : "; //for save prompt  void printprompt(void);  void USART\_Init(unsigned int ubrr){  UBRR0H = (unsigned char)(ubrr >> 8);  UBRR0L = (unsigned char)ubrr;  UCSR0B = (1<<RXEN0) | (1 << TXEN0);  UCSR0C = (3 << UCSZ0);  }  void USART\_Transmit(char data){  while(!((UCSR0A)& (1<<UDRE0)));  UDR0 = data;  }  void USART\_Transmit\_String(char \*str){  while(\*str != '\0')  USART\_Transmit(\*str++);  }  char USART\_Receive(){  while(!(UCSR0A & (1 << RXC0)));  return UDR0;  }    int main()  {  //init OCpin  DDRB = 0x20;    //init USART  UCSR0A = 0x00;  UCSR0B = 0b10011000; //송수신 허용  UCSR0C = 0b10000110; //비동기 통신, 패리티 없고 data bit:8, stop bit: 1  UBRR0H = 0;  UBRR0L = 103; //calculate  //fosc = 14.746MHz, BAUD = 9600bps    //init Timer/Counter  TCCR1A = 0b10100010;  TCCR1B = 0b00011010;  ICR1 = 39999;  TCNT1 = 0;  TIMSK = 0b11000000;  OCR1A = 1999;    SREG = 0x80;    printprompt();    while (1){}  return 0;  }    void printprompt(void){  int i =0;  while(i<8){  while(!((UCSR0A)& (1<<UDRE0)));  UDR0 = prompt[i];  i++;  }  } |   실습 2단계는 숫자를 입력 받고 그 입력 받은 숫자로 motor의 동작을 제어하는 단계다. 숫자를 입력 받기 위해 RX\_vect를 이용해 인터럽트로 putty상에서 숫자를 입력 받는다고 가정하고 ISR을 작성하였다. 위 코드에서는 다소 불필요한 코드가 포함되어 있는데, 실제 코드는 아래에 이어지는 설명에 해당하는 부분만 유효하다.  숫자를 입력 받을 때 수신 완료 인터럽트는 각 들어온 data마다 일어나기 때문에 입력 완료 신호를 주는 부분도 작성했다. 그래서 enter key의 입력을 감자하여 각도를 입력하고 그 입력을 바탕으로 각도를 조절하려고 했다. 원하는 각도를 입력받을 때는 입력 받은 숫자를 저장할 char배열 radc을 선언했는데 각도 입력은 최대 360으로 생각했기 때문에 최대 3자리 숫자를 atoi를 이용해 정수로 바꾸고자 했고 문자열로 인식할 수 있도록 NULL 문자 자리까지 고려하여 radc의 길이는 4로 설정했다.  한편 수신 완료 인터럽트 발생 시, 숫자를 입력받은 경우와 리눅스에서 enter를의미하는 ‘\r’을 입력 받은 경우로 나누어 동작을 제어했다. Putty는 리눅스 기반인 것으로 알고 있기 때문이다. 이 두 경우는 if조건문으로 나누었다.  첫번째 경우, 숫자를 받은 경우에는 ascii code로 문자를 받기 때문에 ch = UDR의 범위가 0을 의미하는 ascii number 48부터 9를 의미하는 ascii number 57까지의 경우를 radc에 넣어주고 10의 자리에 맞게 동작하도록 int k와 ++연산자를 통해 자리 수를 표현했다.  두번째로는, ‘\r’이 입력되는 경우 radc의 마지막에 NULL문자를 추가하고 atoi를 통해 radi에 정수를 저장하여 새로 갱신할 ocr값을 계산하여 OCR안의 값을 바꾸어 줬다.  이후 프롬포트를 출력하는데 이 단계는 char 배열 prompt를 통해 terminal 상에 출력하도록 하는 printprompt함수를 이용하여 해결했다.  이렇게 각도를 입력 받으면 다음은 servo motor의 조작이다. 처음에 overflow 발생 주기를 10ms로 바꾸고 mcu의 clock주파수는 항상 그렇듯 16MHz다. 이후 프리스케일러를 통해 50Hz로 맞춘다. 그리고 timer/counter1의 경우 TCNT를 다 채우지 않고 top 값을 바꾸어 overflow가 일어나는 역치 값을 조절할 수 있었으므로 분주비 8을 가정하여 ICR 값을 구하였다.    위를보면 fast pwm mode에서 분주비 8을 설정하고 ICR1을 39999로 하여, overflow가 일어나는 주기를 20ms으로 설정할 수 있었고, 비반전 모드를 이용했기 때문에 overflow 발생 시 oc pin이 high, 비교일치 발생시 oc pin이 low이므로, 0~180도 사이의 각도 조절을 위해 duty ratio를 5~10%를 맞춰줘야 했다.  ICR1+1에 대한 OCR1+1의 비율이 5~10%이며, OCR 값 1당 변하는 각도를 알아야 입력 받은 각도를 바탕으로 OCR 값을 수정하여 적절한 각도로 변환할 수 있으므로, 이를 구하였다. 180가 변하는 데에 OCR값 2000을 수정하므로, OCR 값 1당 0.09도가 변하기 때문에 입력받은 각도를 0.09로 나누어 1999에 더하여 OCR 값을 수정하였다.      그림 2. 실험 2 회로 구성도  가장 어려웠던 부분은 가상 port를 이용하는 부분이었다. USART를 이용하여 각 소자에 신호를 통신하는데, 인터넷에 무료 puTTy등 여러 가상포트 생성 프로그램이 존재한다. 가상 환경은 아래와 같이 설정해 주었다.  이 외에도 하드웨어를 -90도부터 90도까지가 디폴트인데 0도부터 180도로 바꾸어 코드에 적합하게 만드는 등의 노력을 했다.    그림 3. 실험 2 가상 포트 설정   1. **실험 결과**    1. 실습 1단계     그림 4. 실험 1 실행 결과1    그림 5. 실험 1 실행 결과2   * 1. 실습 2단계     그림 6. 실험 2 실행 직후(puTTy사용)    그림 7. 실험 2 실행 직수(Hypertrm 사용)    그림 8. 실험 2 실행 결과 (입력에 따른 반응)    그림 9. 실험 2 실행 결과 (모터 무반응)   1. **고찰**   이번 주차에는 Timer/Counter의 또 다른 기능을 배웠다. PWM인데, 전반적으로 제대로 기능을 사용해보지 못해 아쉬움이 많이 남는 주차였다. 실험은 1단계과 2단계를 진행했는데 거의 배운 내용으로 이루어진 1단계와 USART를 이용하여 모터 구동을 해야 하는 2단계의 난이도 차이가 너무 컸다. 더하여 2단계 실험이 만족스럽지 못하게 끝나서 기능을 제대로 배웠다고 하기에 아쉬운 부분이 많다.  2단계가 평소보다 어렵게 느껴졌는데, 3단계와 난이도 차이가 거의 나지 않는 것 같았다. USART쪽 문제인 것 같은데 3주차 USART주차에서도 비슷한 문제가 발생했지만 가상 port생성 프로그램을 puTTy가 아닌 다른 프로그램으로 바꾸자 잘 동작 했는데, 이번 주차에서는 여러 번의 시도에도 만족할 만한 결과가 나오지 않았다. 실험을 완료하기 위해 아주 많은 시간을 투자 했는데도 실패하여 더욱 아쉬운 것 같다.  부끄럽지만 이번 주 실습 영상에 2단계 예시가 나왔는데 잠깐 코드가 나왔다. 처음에 코드 작성이 너무 막막하여 그 장면의 코드 일부분을 참고하려고 했었다. 나는 proteus프로그램을 통해 하드웨어를 시뮬레이션 하고 영상에서는 실물 구동이지만 알고리즘은 같을 것이기 때문이다. 하지만 코드의 일부만 보이니 오히려 알고리즘을 파악하기 너무 힘들었고 그냥 처음부터 코드를 다시 작성했다. 그래서 사실 코드에 불필요한 부분이 보일 수 있는데 이는 처음에 코드를 바로 작성한 것이 아니라 영상의 코드 일부분을 베끼려고 했기 때문이다. 하지만 결론적으로 도움이 안되기도 했고 심지어 이해도 못했다. 대신에 직접 작성한 실습 2단계의 코드 설명을 평소보다 자세히 작성하였다. | | | | |
|  | | | | |