

A/D 컨버터

- ATmega128A의 A/D 컨버터 기능
- Text LCD
- Text LCD에 글자쓰기
- UART로 PC에서 데이터 받아 Text LCD에 글자 쓰기
- A/D 컨버터로 광센서 읽기 1
- A/D 컨버터로 광센서 읽기 2
- A/D 컨버터로 전압값 읽기
- A/D 컨버터로 가변저항 값에 따라 LED 밝기 제어하기



ATMega128A의 A/D 컨버터 기능

- ATMega128의 A/D 컨버터 특징
 - 10비트 분해능
 - 0.5 LBS Integral Non-linearity(적분 비선형성)
 - ± 2 LBS 정확도
 - 13~260 μ sec 변환시간(50KHz~200KHz), 15kSPS의 최대 분해능
 - 8채널의 멀티플렉스된 단일 입력(A/D 컨버터는 한 개이며 채널을 바꿔가며 아날로그 신호를 입력 받음)
 - 7채널의 차동입력, 10배 또는 200배의 증폭률을 가진 2채널의 차동입력, ADC 결과 값의 좌 정렬
 - 0~Vcc ADC 입력 전압 범위, 선택 가능한 2.56V의 ADC 레퍼런스 전압. 안정된 동작을 위한 MCU의 디지털 전원과 별도의 아날로그 회로 전원단자 AVCC를 가지며, A/D 변환에 필요한 기준전압 AREF 단자 지원
 - Free running 또는 Single Conversion Mode, ADC변환 완료 인터럽트, Sleep Mode Noise Canceler

ATMega128A의 A/D 컨버터 기능

- ATMega128A A/D 컨버터 레지스터
 - ADMUX(ADC Multiplexer Selection Register)
 - A/D 컨버터 멀티플렉서 선택 레지스터
 - ADCSRA(ADC Control and Status Register A)
 - A/D 컨버터 제어 및 상태 레지스터 A
 - ADCH, ADCL
 - A/D 컨버터 데이터 레지스터

ATMega128A의 A/D 컨버터 기능

- ADMUX (ADC Multiplexer Selection Register)
 - A/D 컨버터 멀티플렉서 선택 레지스터
 - ADC모듈의 아날로그 입력 채널 선택
 - ADC모듈의 기준 전압 선택
 - 변환 결과 레지스터의 데이터 저장형식 지정

7	6	5	4	3	2	1	0
REFS1	REFS0	ADLAR	MUX4	MUX3	MUX2	MUX1	MUX0

ATMega128A의 A/D 컨버터 기능

- ADMUX (ADC Multiplexer Selection Register)
 - 비트 7, 6 : REFS1, 0(Reference Selection Bit)
 - ADC모듈에서 사용하는 기준전압을 선택하는 비트
 - ADC에 대한 Voltage Reference 선택 표

REFS1	REFS0	Voltage Reference
0	0	AREF, Internal Vref turned off
0	1	AVCC with external capacitor at AREF pin
1	0	예약
1	1	Internal 2.56V Voltage Reference with external capacitor at AREF pin

ATMega128A의 A/D 컨버터 기능

- ADMUX (ADC Multiplexer Selection Register)
 - 비트 5 : ADLAR(ADC Left Adjust Result)
 - 이 비트를 1로 설정하면 변환결과가 ADC 데이터 레지스터에 저장될 때 ADC Data Register의 좌측으로 끝을 맞추어 저장
 - 비트 4~0 : MUX4~0(Analog Channel and Gain Selection Bit)
 - ADC 모듈의 아날로그 입력 채널을 선택하는 비트
 - MUX 비트에 의한 아날로그 입력 채널 선택표

MUX4~0	Single Ended Input	Positive Differential Input	Negative Differential Input	Gain
00000	ADC0	N/A		
00001	ADC1			
00010	ADC2			
00011	ADC3			
00100	ADC4			
00101	ADC5			
00110	ADC6			
00111	ADC7			

ATMega128A의 A/D 컨버터 기능

– MUX 비트에 의한 아날로그 입력 채널 선택표

MUX4~0	Single Ended Input	Positive Differential Input	Negative Differential Input	Gain
01000	N/A	ADC0	ADC0	10x
01001		ADC1	ADC0	10x
01010		ADC0	ADC0	200x
01011		ADC1	ADC0	200x
01100		ADC2	ADC2	10x
01101		ADC3	ADC2	10x
01110		ADC2	ADC2	200x
01111		ADC3	ADC2	200x
10000		ADC0	ADC1	1x
10001		ADC1	ADC1	1x
10010		ADC2	ADC1	1x
10011		ADC3	ADC1	1x
10100		ADC4	ADC1	1x
10101		ADC5	ADC1	1x
10110		ADC6	ADC1	1x
10111		ADC7	ADC1	1x
11000		ADC0	ADC2	1x
11001		ADC1	ADC2	1x
11010		ADC2	ADC2	1x
11011		ADC3	ADC2	1x
11100		ADC4	ADC2	1x
11101		ADC5	ADC2	1x
11110	1.22V(VBG)	N/A		
11111	0V(GND)			

ATMega128A의 A/D 컨버터 기능

- ADCSRA(ADC Control and Status Register A)
 - A/D 컨버터 제어 및 상태 레지스터 A
 - ADC 모듈의 동작 설정
 - ADC 모듈의 동작 상태 표시
 - 비트 7 : ADEN (ADC Enable)
 - A/D 컨버터 작동유무 지정
 - 1로 설정하면 ADC 모듈 enable
 - 0으로 설정하면 ADC 모듈 disable

7	6	5	4	3	2	1	0
ADEN	ADSC	ADFR	ADIF	ADIE	ADPS2	ADPS1	ADPS0

ATMega128A의 A/D 컨버터 기능

- ADCSRA(ADC Control and Status Register A)
 - 비트 6 : ADSC(ADC Start Conversion)
 - A/D 컨버터 변환 시작
 - 이 비트에 "1"을 설정하면 ADC 변환이 시작
 - ADEN이 1로 설정되고 난 후 첫 번째 변환에 25개의 ADC 클럭 주기가 필요
 - 다음 변환부터는 13 클럭만 필요
 - AD 변환이 종료되고 난 후 자동적으로 0으로 변환
 - 비트 5 : ADFR(ADC Free Running Select)
 - 프리런닝 모드 설정
 - 1 : Free running 모드로 설정
 - 자동으로 계속해서 AD 변환 실행
 - 0 : 단일 변환 모드 (Single conversion mode)로 설정
 - 사용자가 시작하면 한번만 AD 변환을 실행

ATMega128A의 A/D 컨버터 기능

- ADCSRA(ADC Control and Status Register A)
 - 비트 4 : ADIF(ADC Interrupt Flag)
 - A/D 컨버터 인터럽트 플래그
 - A/D 변환의 완료를 알리는 플래그
 - AD변환이 완료되어 ADC Data Register 값이 업데이트 되고 나면 이 비트가 "1"로 세트되면서 AD 변환 완료 인터럽트를 요청
 - 이때 ADIE=1로 설정되고, SREG 레지스터의 I비트가 1로 설정되어 있으면 이 인터럽트가 발생되어 처리
 - 비트 3 : ADIE(ADC Interrupt Enable)
 - A/D 변환완료 인터럽트 허용
 - AD변환 완료 인터럽트를 개별적으로 설정
 - SREG 레지스터의 I비트가 1로 설정되어 있어야 함

ATMega128A의 A/D 컨버터 기능

- ADCSRA(ADC Control and Status Register A)
 - 비트 2~0 : ADPS2~0(ADC Prescaler Select Bit)
 - A/D 컨버터 프리스케일러 선택
 - ADC 모듈에 인가되는 클록의 분주비를 선택
 - ADPS에 의한 ADC Prescaler 설정표

ADPS2	ADPS1	ADPS0	분주비
0	0	0	2
0	0	1	2
0	1	0	4
0	1	1	8
1	0	0	16
1	0	1	32
1	1	0	64
1	1	1	128

ATMega128A의 A/D 컨버터 기능

- ADCH, ADCL
 - A/D 컨버터 데이터 레지스터
 - A/D 컨버터의 결과를 저장하는 레지스터
 - 단극성 입력 사용시 (Single Ended Input)
 - ADMUX레지스터의 MUX(4:0)가 "00000" ~ "00111" 혹은 "11110" ~ "11111"
 - 변환결과가 10비트 양의 정수로 표시된다(0~1023)
 - 차동입력을 사용시(Differencial Input)
 - ADMUX레지스터의 MUX(4:0)가 "01000" ~ "11101"
 - 변환 결과가 10비트 2의 보수로 표현(-512~+511)
 - 반드시 ADCL(하위 데이터)를 먼저 읽어서 저장한 다음에 ADCH(상위 데이터)를 저장

ATMega128A의 A/D 컨버터 기능

- ADCH, ADCL
 - ADMUX 레지스터의 ADLAR = 0 인 경우 : 우정렬

ADCH

15	14	13	12	11	10	9	8
-	-	-	-	-	-	ADC9	ADC8

ADCL

7	6	5	4	3	2	1	0
ADC7	ADC6	ADC5	ADC4	ADC3	ADC2	ADC1	ADC0

ATMega128A의 A/D 컨버터 기능

- ADCH, ADCL
 - ADMUX 레지스터의 ADLAR = 1 인 경우 : 좌정렬

ADCH

15	14	13	12	11	10	9	8
ADC9	ADC8	ADC7	ADC6	ADC5	ADC4	ADC3	ADC2

ADCL

7	6	5	4	3	2	1	0
ADC1	ADC0	-	-	-	-	-	-

ATMega128A의 A/D 컨버터 기능

- ATMega128A A/D 컨버터의 동작 설정
 - ADMUX로 A/D 신호를 입력받을 채널 선택
 - ADCSR로 컨버전 프리스케일러 설정
 - 인터럽트 사용시, 전역 인터럽트 플래그를 'SET'하여 인터럽트 활성화하고, A/D 변환 완료 인터럽트 처리 루틴을 구성
 - ADCSR의 6번 비트를 세트하여 A/D 변환을 시작
 - 변환 완료 플래그를 주기적으로 점검하거나 아날로그 디지털 컨버전 완료 인터럽트를 이용하여 A/D 컨버전 이후의 데이터 처리 루틴을 구성
 - 반드시 ADCL(하위 데이터)를 먼저 읽어서 저장한 다음에 ADCH(상위 데이터)를 저장

ATMega128A의 A/D 컨버터 기능

- ATMega128A의 A/D 컨버터 설정시 클럭의 선택
 - 10비트 분해능으로 정상적인 동작을 위해서는 50kHz~200kHz 범위의 클럭 사용
 - ADCSRA 레지스터의 ADPS2~0 비트에 의하여 2,4,8,16,32,64,128 중의 1가지로 선택
 - 프리스케일러는 ADCSRA 레지스터에서 ADEN=1로 설정한 경우에만 동작

Text LCD

- Text LCD(Character LCD)
 - LCD화면에 정해진 형태의 문자를 정해진 개수만큼 표시할 수 있도록 만들어진 LCD 디스플레이 장치
 - 각종 임베디드 장치들에서 널리 사용
 - 대부분의 마이크로컨트롤러들과 TEXT LCD는 그대로 연결할 수 있도록 신호를 제공
 - 마이크로 컨트롤러 개발 환경에서도 TEXT LCD 관련 함수를 제공
- 실습에 사용되는 Text LCD
 - 16문자*2라인의 표시부
 - Backlight 기능 포함

Text LCD

- 인터페이스 커넥터 핀 기능

핀	Signal Name	기 능
1	VSS	전원 GND
2	VDD	전원 +5VDC
3	VEE	Contrast 제어 전압레벨 ($VDD - VEE = 13.5 \sim 0V$)
4	RS	Register Select (0 = instruction, 1 = data)
5	R/W	Read/Write (0 = FPGA -> LCD, 1 : FPGA <- LCD)
6	E	Enable Signal for read/write LCD
7	DB0 (LSB)	DATA
8	DB1	
9	DB2	
10	DB3	
11	DB4	
12	DB5	
13	DB6	
14	DB7 (MSB)	
15	A	+LED (backlight LED용 전원 +4.4 ~ 4.7V)
16	K	-LED (backlight LED용 전원 GND)

Text LCD

- Text LCD(Character LCD) 구조
 - TEXT LCD는 보통 LCD 표시부와 LCD 제어부를 하나로 하여 LCD 모듈로 시판
 - LCD제어기 구성
 - 명령(Instruction)과 데이터(Data)를 위한 2개의 레지스터
 - BF (Busy Flag)
 - AC(Address Counter)
 - 문자발생램(CGRAM)
 - 문자발생롬(CGROM)
 - 데이터표시램(DDRAM)

Text LCD

- Text LCD(Character LCD) 구조
 - 내장 레지스터
 - 명령레지스터(IR)
 - DDRAM과 CGRAM에 대한 어드레스와 클리어, 커서시프트 등 제어명령을 보유
 - 데이터레지스터(DR)
 - DDRAM과 CGRAM에 쓴 데이터나 읽은 데이터를 일시적으로 저장
 - 레지스터들은 RS(4번핀)과 R/W(5번핀)을 사용하여 선택
 - RS, R/W 신호에 따른 TEXT LCD 제어기 레지스터 선택 방법

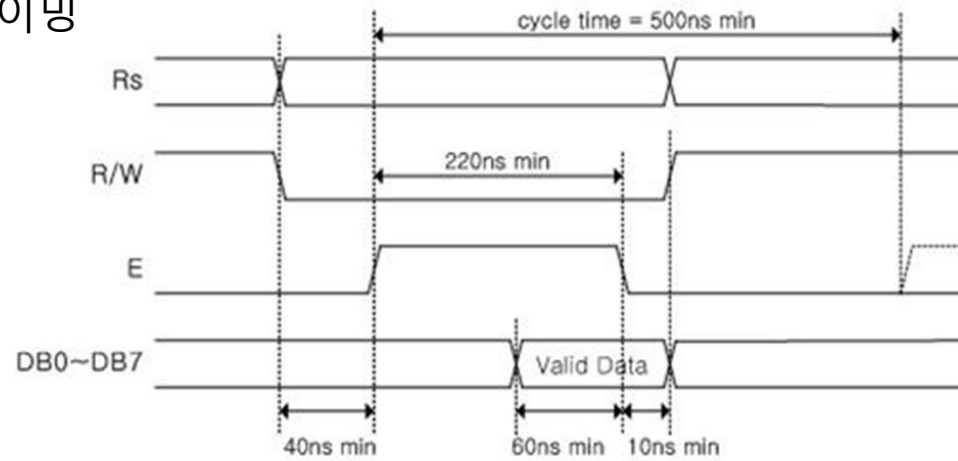
RS	R/W	레지스터 접근
0	0	IR쓰기(각종 제어 명령 쓰기)
0	1	BF읽기, AC읽기
1	0	DR쓰기
1	1	DR읽기

Text LCD

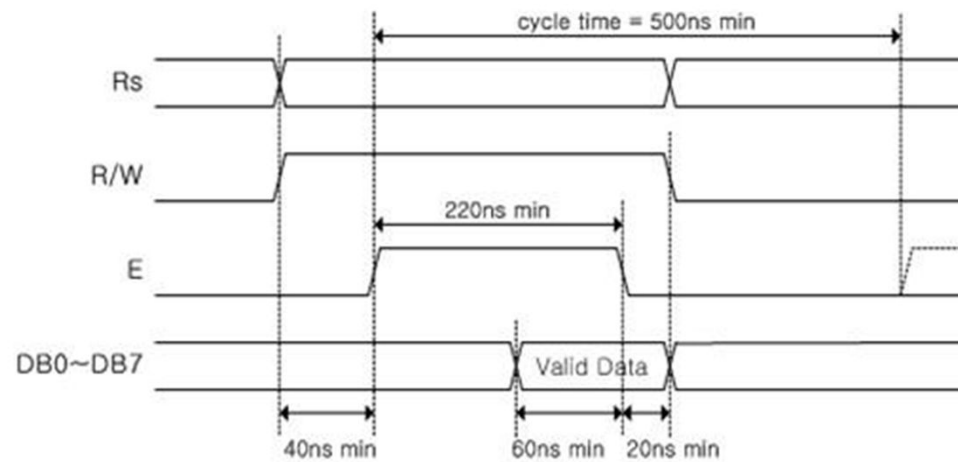
- Text LCD(Character LCD) 구조
 - BF (Busy Flag)
 - 1이면 LCD의 컨트롤러가 동작 중으로 명령 수행 불능
 - 0이면 다음 명령 수행 가능 표시
 - AC(Address Counter)
 - DDRAM과 CGRAM의 주소를 지정할 때 사용
 - IR에 주소 정보를 쓰면 주소 정보가 AC로 전송
 - DDRAM/DDROM에 데이터를 쓰면 AC는 자동으로 +1 혹은 -1이 됨
 - 문자발생램(CGRAM)
 - 사용자가 자유로이 문자를 만들 때 사용하는 램
 - 5x7은 8개, 5x10은 4개 만들어 저장 가능
 - 문자발생롬(CGROM)
 - 5x7, 5x10 도트의 문자를 내장
 - 데이터표시램(DDRAM)
 - 80x8비트 용량으로 80개의 8비트 아스키(ASCII)코드를 저장
 - 0x00-0F 주소가 LCD 1행의 1-16번째, 0x40-4F 주소가 LCD 2행의 1-16번째 문자로 표시

Text LCD

- Text LCD 제어 신호 동작 타이밍



(a) Write from to LCD



(b) Read from LCD

Text LCD

- Text LCD 제어 명령

- LCD 모듈 표시 제어 명령

기능	제어신호		제 어 명 령							
	RS	R/W	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0
Clear Display	0	0	0	0	0	0	0	0	0	1
Return Home	0	0	0	0	0	0	0	0	1	0
Entry Mode Set	0	0	0	0	0	0	0	1	I/D	S
Display ON/OFF Control	0	0	0	0	0	0	1	D	C	B
Cursor or Display Shift	0	0	0	0	0	1	S/C	R/L	0	0
Function Set	0	0	0	0	1	DL	N	F	0	0
Set CG RAM address	0	0	0	1	CG RAM address					
Set DD RAM address	0	0	1	DD RAM address						
Read busy flag and address	0	1	BF	Address Counter						
Data write to CG RAM or DD RAM	1	0	Write address							
Data read from CG RAM or DD RAM	1	1	Read address							

Text LCD

- Text LCD 제어 명령
 - CLEAR DISPLAY
 - 디스플레이 상태를 소거하고 커서를 Home 위치로
 - Return Home
 - DD RAM의 내용은 변경하지 않고 커서만을 Home 위치로
 - Entry Mode SET
 - 데이터를 Read 하거나 Write 할 경우에 커서의 위치를 증가시킬 것인가 (I/D=1) 감소시킬 것인가 (I/D=0)를 결정
 - 이때 화면을 이동 할 것인지 (S=1) 아닌지 (S=0)를 결정
 - Display ON/OFF Control
 - 화면 표시를 ON/OFF 하거나(D) 커서를 ON/OFF 하거나(C) 커서를 깜박이게 할 것인지(B)의 여부를 지정
 - Cursor or Display Shift
 - 화면(S/C=1) 또는 커서(S/C=0)를 오른쪽(R/L=1) 또는 왼쪽(R/L=0)으로 이동

Text LCD

- Text LCD 제어 명령
 - Function SET
 - 인터페이스에서 데이터의 길이를 8비트(DL=1) 또는 4비트(DL=0)로 지정
 - 화면 표시 행수를 2행(N=1) 또는 1행(N=0)으로 지정
 - 문자의 폰트를 5x10 도트(F=1) 또는 5x7 도트(F=0)로 지정
 - Set CG RAM Address
 - Character Generator RAM의 어드레스를 지정
 - 지정 이후에 송수신 하는 데이터는 CG RAM의 데이터
 - Set DD RAM Address
 - Display Data RAM의 어드레스를 지정.
 - 지정 이후에 송수신하는 데이터는 DD RAM의 데이터
 - Read Busy Flag & Address
 - LCD 모듈이 내부 동작중임을 나타내는 Busy Flag(BF) 및 어드레스 카운터의 내용을 read

Text LCD

- Text LCD 제어 명령
 - Text LCD DDRAM Address
 - DDRAM은 표시될 각 문자의 ASCII 코드 데이터가 저장되어 있는 메모리
 - 모두 80개의 번지가 있으며, 화면의 각 행과 열의 위치에는 고유한 어드레스 값이 부여되어 있음
 - 각 행과 행 사이의 어드레스가 연속하여 있지 않으므로 주의해야 함
 - 표시 문자의 위치에 대한 DD RAM의 어드레스

구분	1	2	3	4	13	14	15	16
Line1	00	01	02	03	0D	0E	0F	10
Line2	40	41	42	43	4D	4E	4F	50

Text LCD

— ASCII 도형문자 종류 및 코드 값

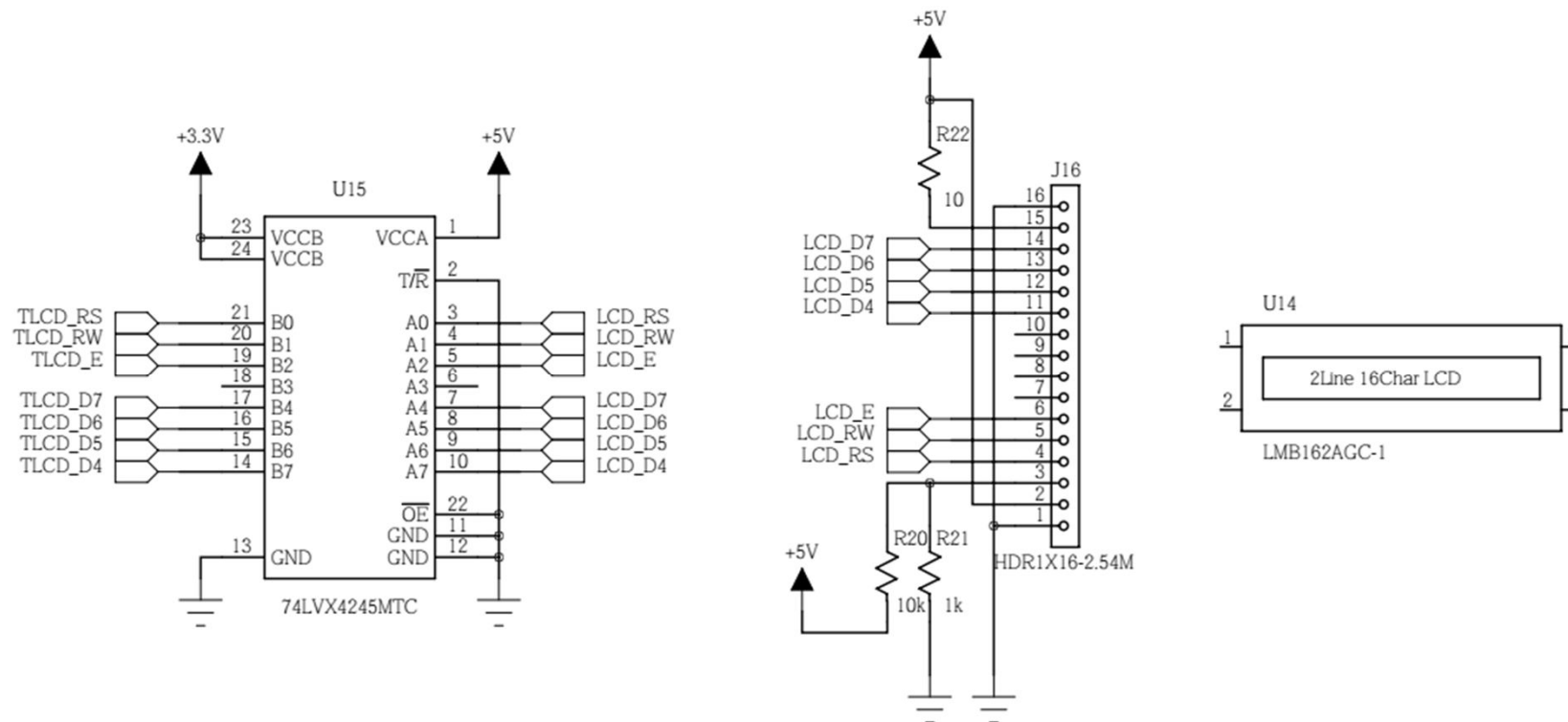
구분	00H	10H	20H	30H	40H	50H	60H	70H	80H	90H
0	사용자 정의 영역	미사용 영역		0	@	P	`	p	미사용 영역	
1			!	1	A	Q	a	q		
2			"	2	B	R	b	r		
3			#	3	C	S	c	s		
4			\$	4	D	T	d	t		
5			%	5	E	U	e	u		
6			&	6	F	V	f	v		
7			'	7	G	W	g	w		
8			(8	H	X	h	x		
9)	9	I	Y	i	y		
A			*	:	J	Z	j	z		
B			+	;	K	[k	{		
C			,	<	L	¥	l			
D			-	=	M]	m	}		
E			.	>	N	^	n	→		
F			/	?	O	_	o	←		

실습 1: Text LCD에 글자 쓰기

- 실습 개요
 - ATmega128A의 GPIO에 TEXT LCD를 연결하고, LCD 화면에 미리 작성된 문장 ("Hello! MCU World !!")을 표시
 - AVR 개발 환경에서 제공하는 TEXT LCD 관련 함수를 이해하면 쉽게 프로그램을 작성할 수 있음
- 실습 목표
 - TEXT LCD의 동작 원리 이해
 - AVR 개발환경에서 제공하는 TEXT LCD 관련 함수 이해
 - TEXT LCD 제어 프로그램 방법 습득

실습 1: Text LCD에 글자 쓰기

- 사용 모듈
 - TEXT LCD 모듈 회로

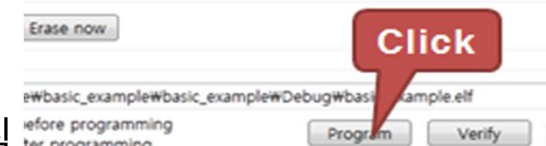
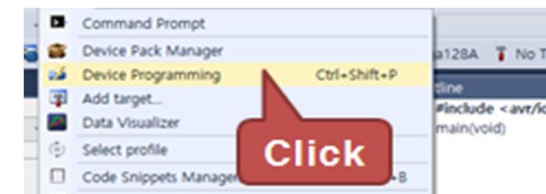
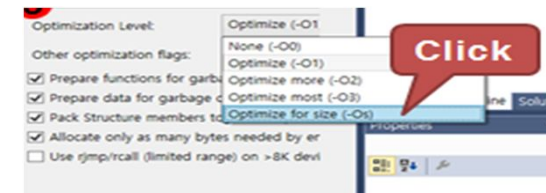
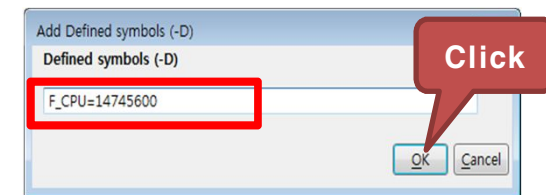


실습 1: Text LCD에 글자 쓰기

- 구동 프로그램 : 사전지식
 - AVR 개발 환경에서 TEXT LCD 관련 라이브러리 함수를 제공
 - lcd.c 라는 파일에 포함
 - Atmel Studio에서 프로젝트를 Build할 때 이 lcd.c 파일, lcd.h 파일, 그리고 lcdconf.h 파일을 함께 포함시켜야 함
 - TEXT LCD용 라이브러리 함수
 - lcdInit : TEXT LCD 초기화
 - lcdGotoXY(unsigned char x, unsigned char y) : TEXT LCD의 커서를 원하는 위치로 이동
 - lcdDataWrite(unsigned char data): TEXT LCD에 하나의 문자 출력
 - lcdPrintData(char* data, unsigned char nBytes) : TEXT LCD에 nBytes길이의 문자열을 출력
 - lcdClear() : TEXT LCD의 화면을 지움

실습 1: Text LCD에 글자 쓰기

- 예제 프로그램 작성 및 구동
 - Atmel Studio 실행
 - New Project 생성
 - Name : 09_TextLcd_Example, Location : D:\WAVR_Example
 - Device Selection : ATmega128A
 - 프로젝트 설정
 - Project 탭에서 "... Properties..." 선택
 - Toolchain -> AVR/GNU C Compiler에서
 - Symbols -> F_CPU=14745600 추가
 - Optimization -> Optimize for size (-OS) 선택
 - 저장 (Ctrl+S)
 - 소스코드 작성
 - 프로젝트 빌드
 - Build 탭에서 "Build Solution" 클릭
 - 프로그래밍
 - Tool 탭에서 "Device Programming" 클릭
 - AVRISP mkII, ATmega128A 선택 후 "Apply" 클릭
 - 인식 완료되면, Memories 탭 선택, "Program" 클릭



실습 1: Text LCD에 글자 쓰기

- 구동 프로그램
 - main.c 코드 작성

```
#include <avr/io.h>      // AVR 입출력에 대한 헤더 파일
#include "lcd.h"          // Text LCD를 사용하기 위한 헤더 파일

int main(void) {
    lcdInit();            // Text LCD를 초기화
    lcdGotoXY(0,0);       // 현재 커서위치를 첫번째줄 첫번째칸으로 이동
    // 첫번째 매개변수는 행을, 두번째 매개변수는 열을 의미
    lcdDataWrite('H');    // 'H'를 출력
    lcdDataWrite('e');    // 'e'를 출력
    lcdDataWrite('l');    // 'l'를 출력
    lcdDataWrite('l');    // 'l'를 출력
    lcdDataWrite('o');    // 'o'를 출력
    lcdGotoXY(3,1);       // 커서위치를 두번째 줄, 네번째 칸으로 이동
    lcdPrintData("MCU World !!",12); // "MCU World !!" 문자열 출력
    while (1) {}
}
```

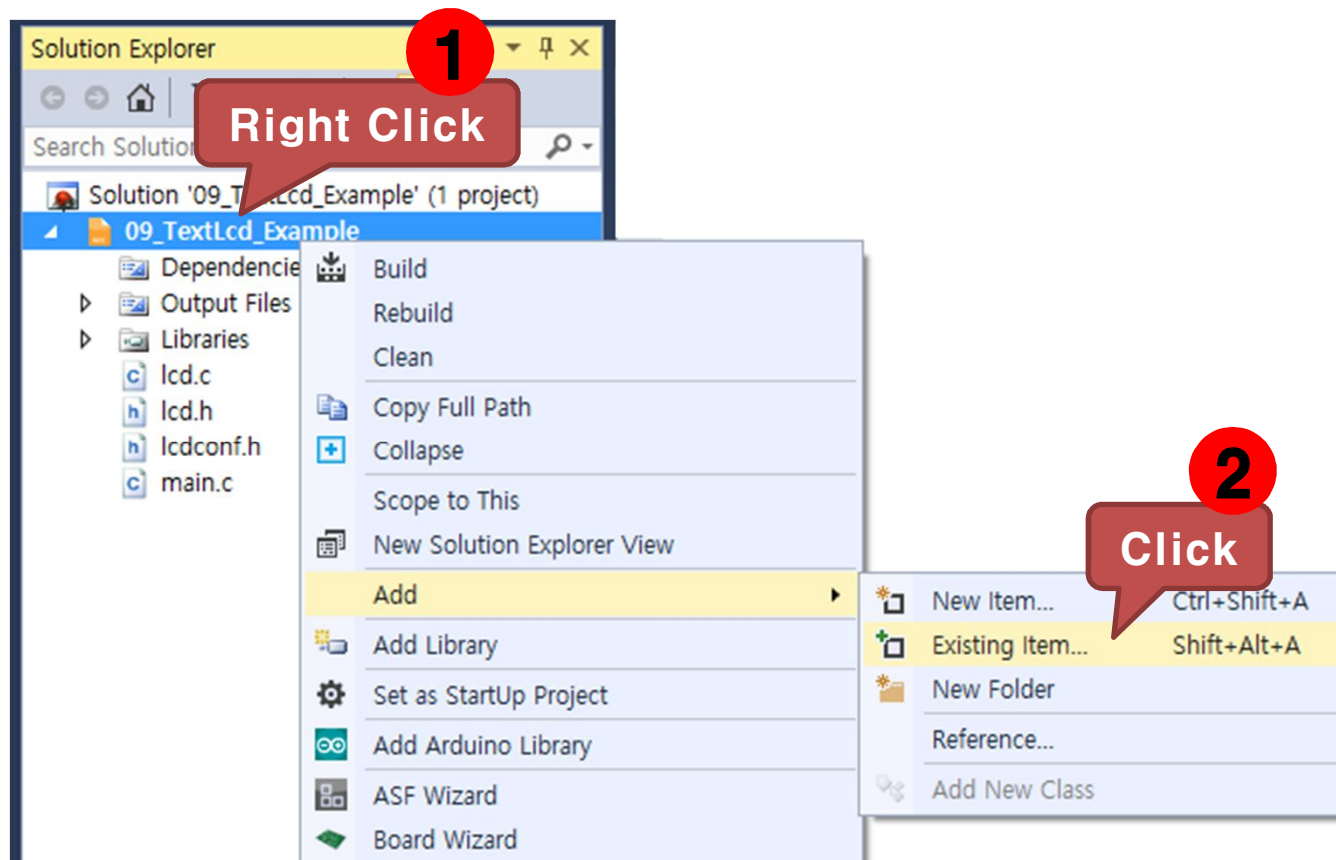

실습 1: Text LCD에 글자 쓰기

- 구동 프로그램 : Atmel Studio 에서 라이브러리 함수 추가하는 방법
 - 라이브러리 함수파일을 프로젝트내로 복사
 - lcd.c, lcd.h, lcdconf.h 파일은 "AVR_Example\library\lcd" 폴더에 존재
 - 라이브러리 파일은 Atmel Studio에서 새 프로젝트를 생성한 후 프로젝트가 생성된 폴더안에 있는 프로젝트 폴더에 복사
 - 예를 들어 TextLcd 라는 프로젝트를 생성하면 "TextLcd" 라는 프로젝트용 폴더가 생성이 되고 이 폴더안에 프로젝트 이름과 같은 "TextLcd" 폴더가 하나 더 생성이 되며 소스 코드 및 빌드 결과물이 저장됨
 - 즉, lcd.c, lcd.h, lcdconf.h 파일을 "TextLcd" 폴더에 복사해 넣으면 됨 (라이브러리 파일은 main.c 파일과 같은 경로에 있어야 함)



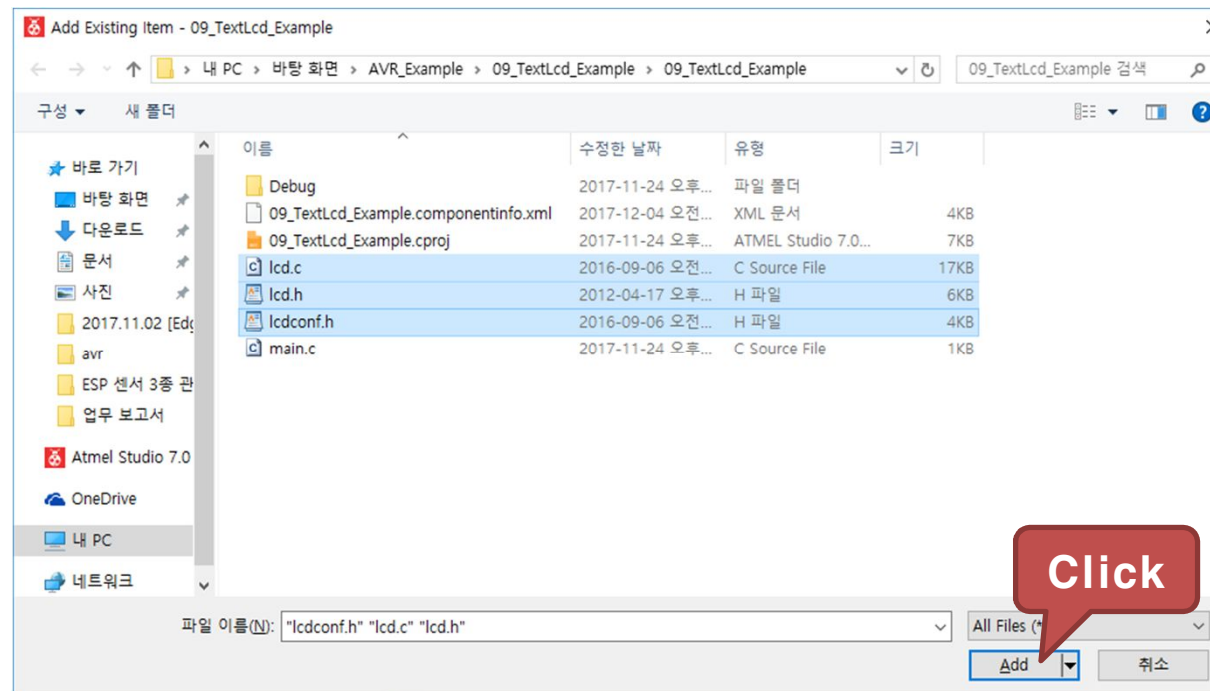
실습 1: Text LCD에 글자 쓰기

- 구동 프로그램 : Atmel Studio 에서 라이브러리 함수 추가하는 방법
 - Atmel Studio 상에서 프로젝트의 솔루션 탐색기에서 라이브러리 파일들을 추가



실습 1: Text LCD에 글자 쓰기

- 구동 프로그램 : Atmel Studio 에서 라이브러리 함수 추가하는 방법
 - lcd.c, lcd.h, lcdconf.h 파일을 각각 추가
 - 또는 한번에 세 파일을 동시에 추가
 - Ctrl 키를 누르고 파일 세개를 순서대로 클릭하여 세파일을 동시에 선택하고 한번에 추가



실습 1: Text LCD에 글자 쓰기

- 구동 프로그램 : 사용 포트와 핀 선언
 - 사용할 포트를 선언하는 헤더화일 : lcdconf.h
 - 이 예제에서는 MCU 모듈의 C 포트와 G 포트를 사용
 - 다음은 lcdconf.h 파일의 일부분

```
#ifdef LCD_PORT_INTERFACE
    #ifndef LCD_CTRL_PORT
        #define LCD_CTRL_PORT    PORTG
        #define LCD_CTRL_DDR    DDRG
        #define LCD_CTRL_RW    0
        #define LCD_CTRL_RS    1
        #define LCD_CTRL_E    2
    #endif
    #ifndef LCD_DATA_POUT
        #define LCD_DATA_POUT    PORTC
        #define LCD_DATA_PIN    PINC
        #define LCD_DATA_DDR    DDRC
        #define LCD_DATA_4BIT
    #endif
#endif
```

Text LCD
제어포트 선언

Text LCD
데이터포트 선언

실습 1: Text LCD에 글자 쓰기

- 구동 프로그램 : 사용 포트와 핀 선언
 - 사용할 포트를 선언하는 헤더화일 : lcdconf.h
 - Text LCD 제어 포트 선언

매크로 선언	매크로 상수 의미	예
#define LCD_CTRL_PORT	포트 데이터 레지스터	PORTA, PORTC
#define LCD_CTRL_DDR	데이터 방향 레지스터	DDRA, DDRC
#define LCD_CTRL_RS	TextLCD RS핀	0 ~ 7 의 값
#define LCD_CTRL_RW	TextLCD RW핀	0 ~ 7 의 값
#define LCD_CTRL_E	TextLCD E핀	0 ~ 7 의 값

- Text LCD 데이터 포트 선언

매크로 선언	매크로 상수 의미	사용 예
#define LCD_DATA_POUT	포트 데이터 레지스터	PORTA, PORTC
#define LCD_DATA_PIN	입력 핀 어드레스	PINA, PINC
#define LCD_DATA_DDR	데이터 방향 레지스터	DDRA, DDRC

- Text LCD 데이터 포트 4비트 사용시 선언

매크로 선언	매크로 상수 의미	사용 예
#define LCD_DATA_4BIT	데이터 제어 비트 설정	4비트 사용시에만 선언

실습 1: Text LCD에 글자 쓰기

- 실행 결과
 - Text LCD 모듈에 "Hello MCU World!!" 문자열이 출력

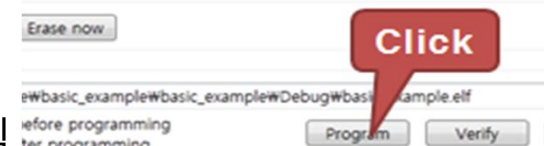
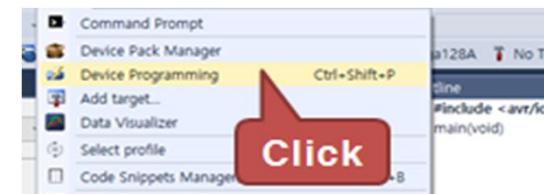
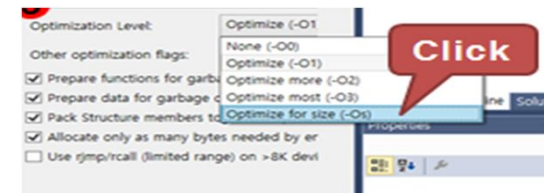
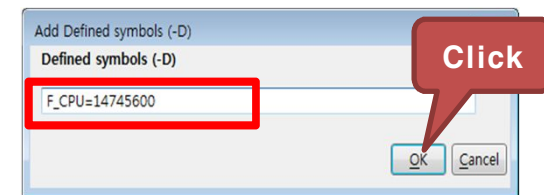


응용 1: UART로 PC에서 데이터 받아 Text LCD에 글자 쓰기

- 실습 개요
 - ATmega128A의 GPIO에 TEXT LCD를 연결하고, LCD 화면에 UART로부터 받은 문장을 표시
 - AVR 개발 환경에서 제공하는 TEXT LCD 관련 함수를 이해하면 쉽게 프로그램을 작성할 수 있음
 - UART통신과 TEXT LCD를 결합하여 응용할 수 있음
- 실습 목표
 - TEXT LCD의 동작 원리 이해
 - AVR 개발환경에서 제공하는 TEXT LCD 관련 함수 이해
 - TEXT LCD 제어 프로그램 방법 습득

응용 1: UART로 PC에서 데이터 받아 Text LCD에 글자 쓰기

- 예제 프로그램 작성 및 구동
 - Atmel Studio 실행
 - New Project 생성
 - Name : 09_TextLcd_Application, Location : D:\WAVR_Example
 - Device Selection : ATmega128A
 - 프로젝트 설정
 - Project 탭에서 "... Properties..." 선택
 - Toolchain -> AVR/GNU C Compiler에서
 - Symbols -> F_CPU=14745600 추가
 - Optimization -> Optimize for size (-OS) 선택
 - 저장 (Ctrl+S)
 - 소스코드 작성
 - 프로젝트 빌드
 - Build 탭에서 "Build Solution" 클릭
 - 프로그래밍
 - Tool 탭에서 "Device Programming" 클릭
 - AVRISP mkII, ATmega128A 선택 후 "Apply" 클릭
 - 인식 완료되면, Memories 탭 선택, "Program" 클릭



응용 1: UART로 PC에서 데이터 받아 Text LCD에 글자 쓰기

- 구동 프로그램
 - main.c 코드 작성

```
#include <avr/io.h>           //AVR 입출력에 대한 헤더 파일
#include "lcd.h"               //Text LCD를 사용하기 위한 헤더 파일

unsigned char cursor = 0;

void putch(unsigned char data)
{
    while((UCSR0A & 0x20) == 0); // 전송준비가 될때까지 대기
    UDR0 = data;                 // 데이터를 UDR0에 쓰면 전송된다
    UCSR0A |= 0x20;
}
```

응용 1: UART로 PC에서 데이터 받아 Text LCD에 글자 쓰기

- 구동 프로그램
 - main.c 코드 작성

```
unsigned char getch(void)
{
    unsigned char data;
    while((UCSR0A & 0x80)==0); // 데이터를 받을때까지 대기
    data=UDR0;                // 수신된 데이터는 UDR0에 저장되어 있다.
    UCSR0A |= 0x80;
    return data;              // 읽어온 문자를 반환한다.
}

int main(void)
{
    unsigned char RX_data = 0;
    DDRE = 0x0e;              // Rx(입력 0), Tx(출력, 1), SW0 ~ 3 입력
```

응용 1: UART로 PC에서 데이터 받아 Text LCD에 글자 쓰기

- 구동 프로그램
 - main.c 코드 작성

```
UCSR0A = 0x00;
UCSR0B = 0x18; // Rx, Tx enable
UCSR0C = 0x06; // 비동기 방식, No Parity bit, 1 Stop bit

UBRR0H = 0x00;
UBRR0L = 0x07; // 115200 bps

lcdInit(); //Text LCD를 초기화

while (1)
{
    RX_data = getch(); // PC로 입력받은 데이터를 변수 RX_data로 저장
```

응용 1: UART로 PC에서 데이터 받아 Text LCD에 글자 쓰기

- 구동 프로그램
 - main.c 코드 작성

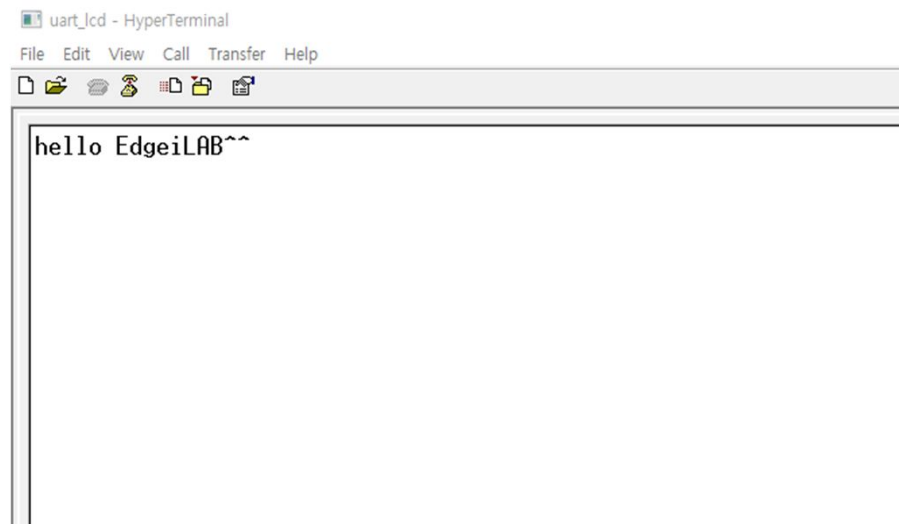
```
        if(cursor == 0)
        {
            lcdClear();          // LCD에 출력된 문자를 클리어한다.
        }

출력    putchar(RX_data);      // 변수 RX_data로 저장된 데이터를 PC로
출력    lcdDataWrite(RX_data); // 변수 RX_data로 저장된 데이터를 LCD로

        cursor++;              // LCD로 출력한 데이터를 카운트한다.
        if(cursor == 16)       // LCD로 카운트한값이 16이면
            lcdGotoXY(0,1);     // 커서를 2행을 이동한다.
        else if(cursor == 32)  // LCD로 카운트한값이 32이면
            cursor = 0;         // 변수 cursor 값을 0으로 초기화 한다.
    }
}
```

응용 1: UART로 PC에서 데이터 받아 Text LCD에 글자 쓰기

- 결과

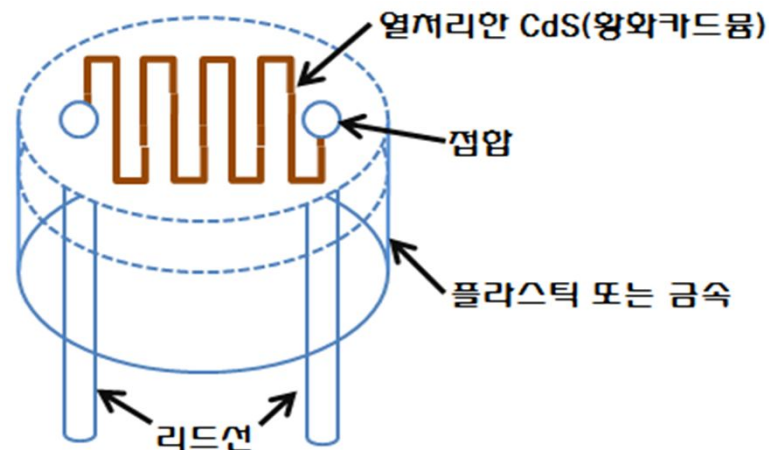


실습 2: A/D 컨버터로 광센서 읽기 1

- 실습 개요
 - ATmega128A 의 A/D 컨버터 기능을 이용하여 광 센서(CdS)로부터 밝기 정보를 읽어내어 Text LCD에 출력
- 실습 목표
 - ATmega128A A/D 컨버터의 동작 원리 이해
 - A/D 컨버터 제어 방법 습득(레지스터 설정)
 - CdS 동작 원리 이해

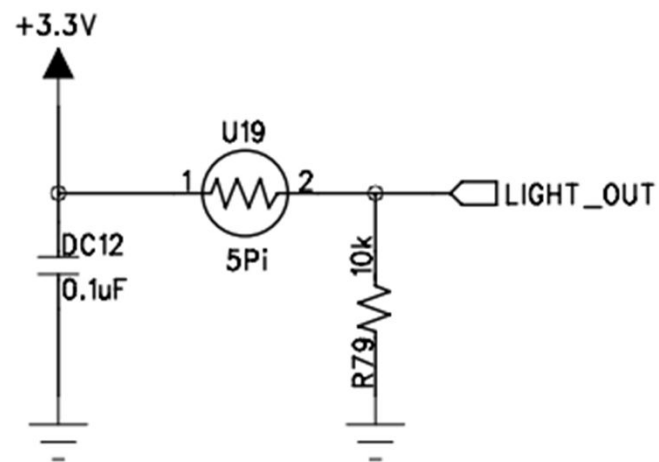
실습 2: A/D 컨버터로 광센서 읽기 1

- CdS
 - 빛의 세기에 따라 내부 저항이 변하는데 밝은 곳에서는 내부 저항이 작아지는 광 가변 저항
 - 저항은 옴의 법칙에 따라 전압에 비례하고 전류에 반비례($R = V/I$)
 - 밝은 곳에서는 저항이 작아져 전류가 증가하고 어두운 곳에서는 전류가 감소



실습 2: A/D 컨버터로 광센서 읽기 1

- 사용 모듈
 - 센서 모듈의 광센서 회로
 - LIGHT_OUT 신호가 Sensor 신호선 포트의 CdS에 매핑
 - 빛이 들어오면 세기에 따라 저항 값이 변화

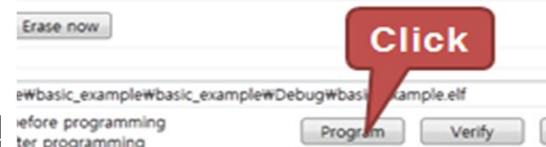
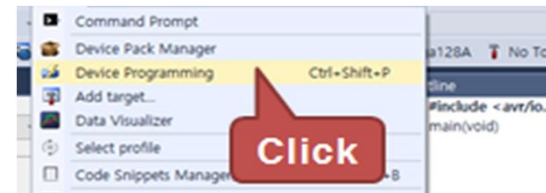
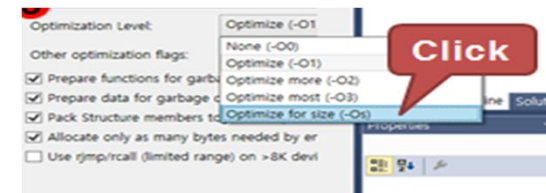
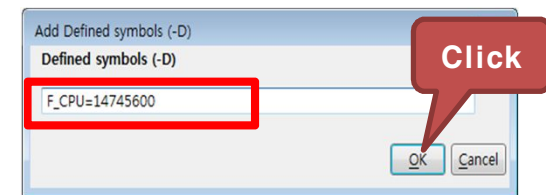


실습 2: A/D 컨버터로 광센서 읽기 1

- 구동 프로그램 : 사전 지식
 - A/D 컨버터 기능을 이용하여 CdS에서 출력되는 빛의 세기에 대한 아날로그 신호를 받아 디지털로 변환한 뒤, 이를 Text LCD에 표시
 - Delay 함수를 이용하여 0.5초 간격으로 데이터를 읽도록 함
 - A/D 컨버터의 설정 방법
 - A/D 컨버터의 입력채널 결정 : 0번 채널 사용
 - 입력 데이터의 정렬 방법과 기준전압 선택 : 데이터 정렬은 디폴트인 우정렬로 하고, 기준 전압을 외부 기준전압 AREF(3.3V) 사용
 - 프리스케일러와 인터럽트 결정 : 프리스케일러는 128분주를 사용하고, 인터럽트는 사용하지 않음
 - 인터럽트를 사용하지 않으므로 설정 끝
 - A/D 컨버팅 제어
 - 설정을 마친 후, ADCSR의 6번 비트를 세트하여 A/D 변환을 시작
 - A/D 변환 도중에는 변환 완료 플래그를 주기적으로 점검하면서 A/D 데이터 레지스터로부터 데이터를 읽어 들이면 된다.
 - 이때, 반드시 ADCL(하위 데이터)를 먼저 읽어서 저장한 다음에 ADCH(상위 데이터)를 읽어 내야 함

실습 2: A/D 컨버터로 광센서 읽기 1

- 예제 프로그램 작성 및 구동
 - Atmel Studio 실행
 - New Project 생성
 - Name : 09_ADC_Example, Location : D:\WAVR_Example
 - Device Selection : ATmega128A
 - 프로젝트 설정
 - Project 탭에서 "... Properties..." 선택
 - Toolchain -> AVR/GNU C Compiler에서
 - Symbols -> F_CPU=14745600 추가
 - Optimization -> Optimize for size (-OS) 선택
 - 저장 (Ctrl+S)
 - 소스코드 작성
 - 프로젝트 빌드
 - Build 탭에서 "Build Solution" 클릭
 - 프로그래밍
 - Tool 탭에서 "Device Programming" 클릭
 - AVRISP mkII, ATmega128A 선택 후 "Apply" 클릭
 - 인식 완료되면, Memories 탭 선택, "Program" 클릭



실습 2: A/D 컨버터로 광센서 읽기 1

- 구동 프로그램
 - main.c 코드 작성

```
#include <avr/io.h>      // AVR 입출력에 대한 헤더 파일
#include <util/delay.h>    // delay 함수사용을 위한 헤더파일
#include "lcd.h"          // Text LCD를 사용하기 위한 헤더 파일

int main(void)
{
    unsigned int AdData = 0;    // 10bit ADC값 저장 변수

    lcdInit();                 // Text LCD를 초기화

    ADMUX = 0x00;              // ADC0선택
    // single mode, 0번 채널, 3.3V 외부 기준전압(AREF)
    ADCSRA = 0x87;             // 10000111
                                // ADC 허가, 128분주
```

실습 2: A/D 컨버터로 광센서 읽기 1

- 구동 프로그램
 - main.c 코드 작성

```
while(1) {  
    // ADC0을 통한 ADC값 읽어오기  
    ADCSRA |= 0x40;      // ADSC AD 개시(Start)  
    while((ADCSRA & 0x10) == 0x00); // ADIF AD 다 될 때까지 기다림  
    AdData = ADC;         // 전압이 디지털로 변환된 값 읽어오기  
    lcdGotoXY(0,0);       // 커서위치를 첫번째 줄, 첫번째 칸으로 이동    lcdPrintData(" Cds: ",6); // " Cds: " 출력  
    lcdDataWrite((AdData/1000)%10 + '0'); // 1000의 자리 출력  
    lcdDataWrite((AdData/100)%10 + '0'); // 100의 자리 출력  
    lcdDataWrite((AdData/10)%10 + '0'); // 10의 자리 출력  
    lcdDataWrite((AdData)%10 + '0'); // 1의 자리 출력  
    _delay_ms(500);  
}  
}
```

실습 2: A/D 컨버터로 광센서 읽기 1

- 구동 프로그램 : Atmel Studio 에서 라이브러리 함수 추가하는 방법
 - 라이브러리 함수파일을 프로젝트내로 복사
 - lcd.c, lcd.h, lcdconf.h 파일은 "AVR_Example\library\lcd" 폴더에 존재
 - 라이브러리 파일은 Atmel Studio에서 새 프로젝트를 생성한 후 프로젝트가 생성된 폴더안에 있는 프로젝트 폴더에 복사

lcd Library 폴더

이름
lcd
lcd
lcdconf

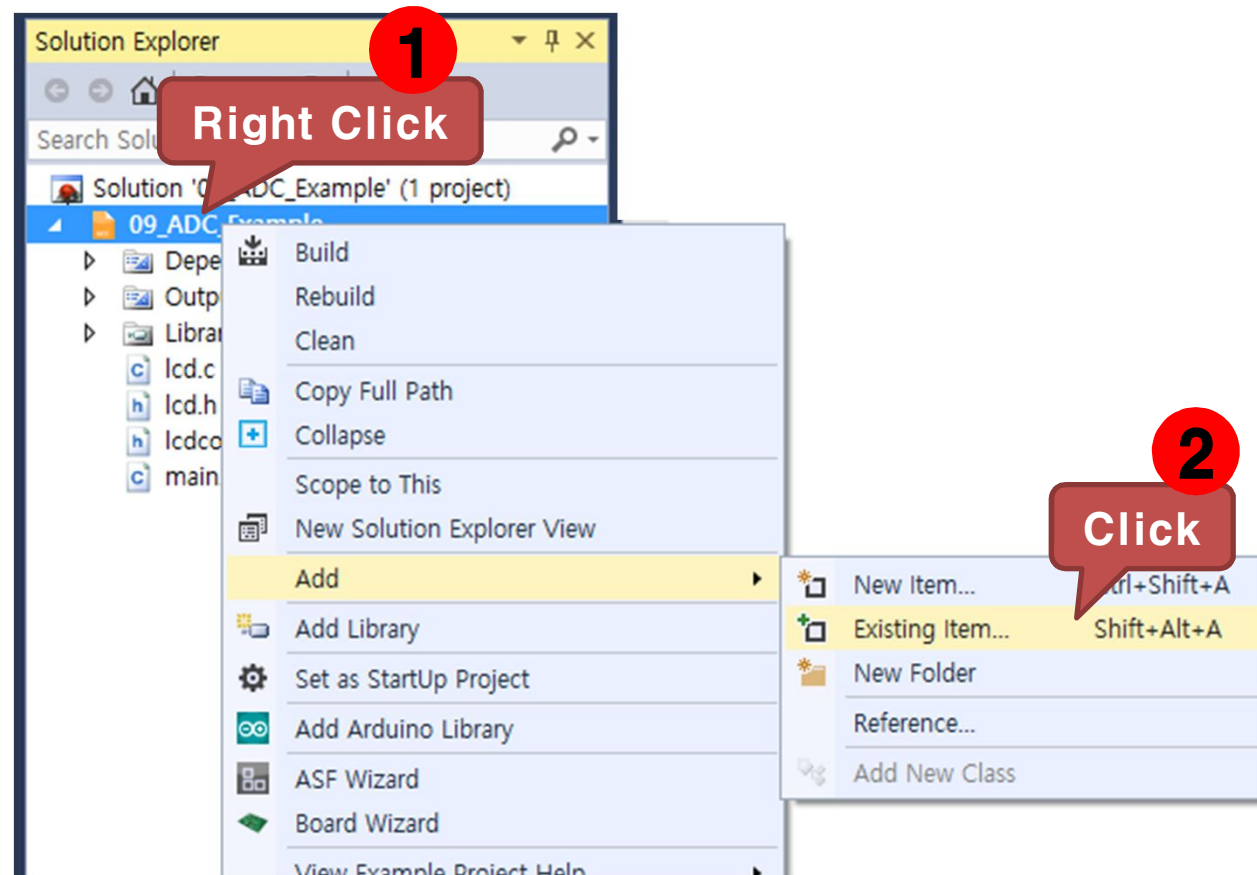
09_ADC_Example 프로젝트 폴더

이름	수정한 날짜	유형
Debug	2016-09-06 오전...	파일
main	2016-09-06 오전...	C S
adccds.componentinfo	2016-09-06 오후...	XMI
adccds	2016-09-06 오전...	ATN

파일 복사

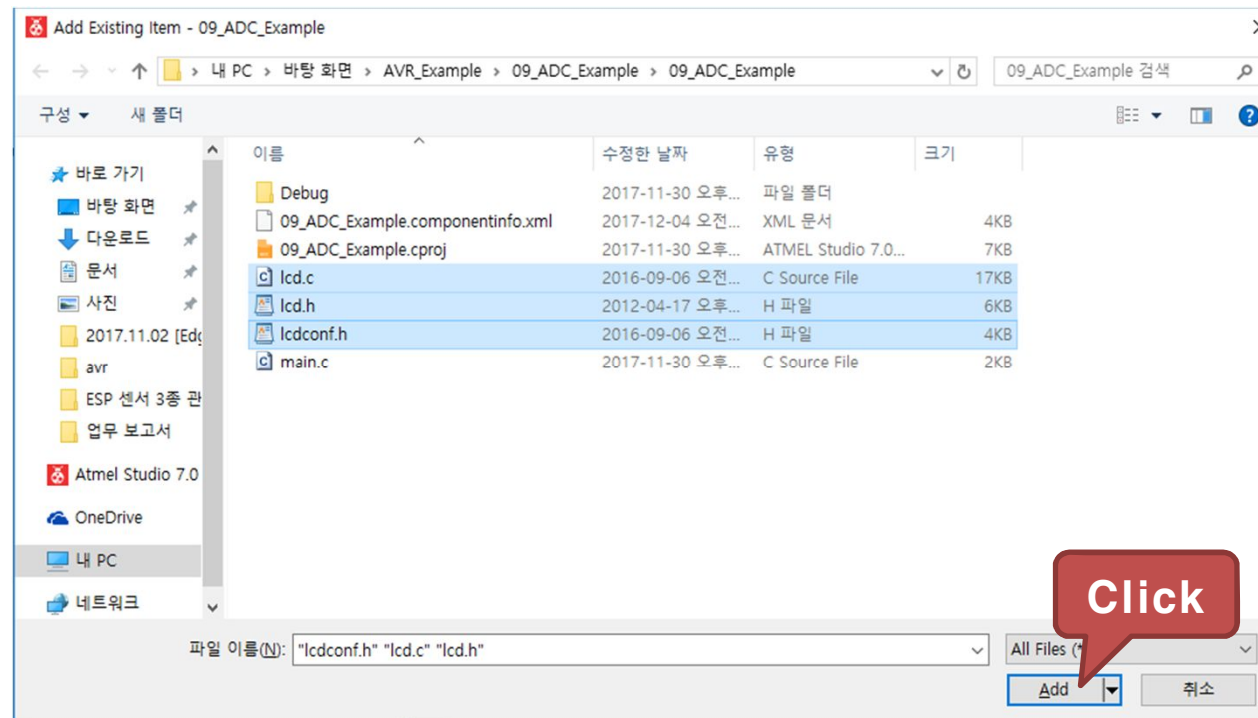
실습 2: A/D 컨버터로 광센서 읽기 1

- 구동 프로그램 : Atmel Studio 에서 라이브러리 함수 추가하는 방법
 - Atmel Studio 상에서 프로젝트의 솔루션 탐색기에서 라이브러리 파일들을 추가



실습 2: A/D 컨버터로 광센서 읽기 1

- 구동 프로그램 : Atmel Studio 에서 라이브러리 함수 추가하는 방법
 - lcd.c, lcd.h, lcdconf.h 파일을 각각 추가하
 - 또는 한번에 세 파일을 동시에 추가
 - Ctrl 키를 누르고 파일 세개를 순서대로 클릭하여 세파일을 동시에 선택하고 한번에 추가



실습 2: A/D 컨버터로 광센서 읽기 1

- 실행 결과
 - Text LCD에 광센서값을 출력
 - 광센서를 손으로 가리면 센서 값이 달라지는 것을 확인

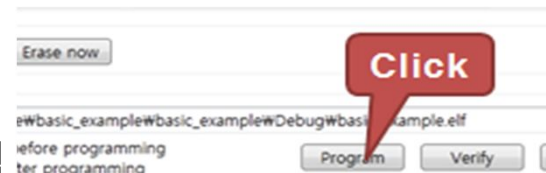
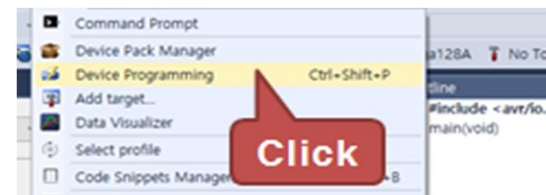
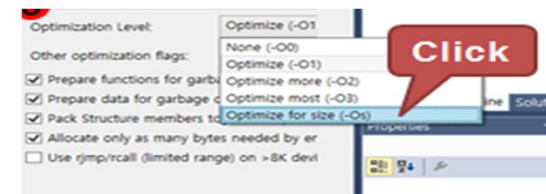
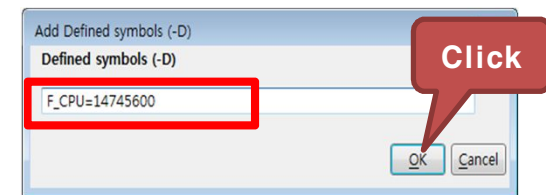


응용 2: A/D 컨버터로 광센서 읽기 2

- 실습 개요
 - ATmega128A 의 A/D 컨버터 기능을 이용하여 광 센서(CdS)로부터 밝기 정보를 읽어내어 PC로 전송하여 출력
 - A/D free running mode를 이용하여 광 센서(CdS)로 부터 밝기 정보를 읽어 냄
- 실습 목표
 - ATmega128A A/D 컨버터의 동작 원리 이해
 - A/D 컨버터 제어 방법 습득(레지스터 설정)
 - CdS 동작 원리 이해

응용 2: A/D 컨버터로 광센서 읽기 2

- 예제 프로그램 작성 및 구동
 - Atmel Studio 실행
 - New Project 생성
 - Name : 09_ADC_Application, Location : D:\WAVR_Example
 - Device Selection : ATmega128A
 - 프로젝트 설정
 - Project 탭에서 "... Properties..." 선택
 - Toolchain -> AVR/GNU C Compiler에서
 - Symbols -> F_CPU=14745600 추가
 - Optimization -> Optimize for size (-OS) 선택
 - 저장 (Ctrl+S)
 - 소스코드 작성
 - 프로젝트 빌드
 - Build 탭에서 "Build Solution" 클릭
 - 프로그래밍
 - Tool 탭에서 "Device Programming" 클릭
 - AVRISP mkII, ATmega128A 선택 후 "Apply" 클릭
 - 인식 완료되면, Memories 탭 선택, "Program" 클릭



응용 2: A/D 컨버터로 광센서 읽기 2

- 구동 프로그램
 - main.c 코드 작성

```
#include <avr/io.h>           //AVR 입출력에 대한 헤더 파일
#include <avr/interrupt.h>     // AVR 인터럽트에 대한 헤더파일
#include <util/delay.h>        // delay 함수사용을 위한 헤더파일

volatile unsigned int ADC_result = 0;

void putch(unsigned char data)
{
    while((UCSR0A & 0x20) == 0); // 전송준비가 될때까지 대기
    UDR0 = data;                 // 데이터를 UDR0에 쓰면 전송된다
    UCSR0A |= 0x20;
}
```

응용 2: A/D 컨버터로 광센서 읽기 2

- 구동 프로그램
 - main.c 코드 작성

```
// 문자열을 출력하는 함수
void putch_Str(char *str)
{
    unsigned char i=0;
    while(str[i]!='\0')
        putch(str[i++]);           //문자열을 출력
}

int main(void)
{
    unsigned int AdData = 0;       //10bit ADC값 저장 변수

    DDRF = 0x00;                  // PF0 입력으로 설정
    DDRE = 0x02;                  // Rx(입력 0), Tx(출력, 1), SW0입력
```

응용 2: A/D 컨버터로 광센서 읽기 2

- 구동 프로그램
 - main.c 코드 작성

```
UCSR0A = 0x00;
UCSR0B = 0x18;          // Rx, Tx enable
UCSR0C = 0x06;          // 비동기 방식, No Parity bit, 1 Stop bit

UBRR0H = 0x00;
UBRR0L = 0x07;          // 115200 bps

ADMUX = 0x40;           //ADC0선택
//single mode, 0번 채널, 3.3V 외부 기준전압(AREF)
ADCSRA = 0xAF;          // 10101111
//ADC 허가, free running Mode, Interrupt 허가, 128분주
ADCSRA |= 0x40;         //ADSC AD 개시(Start)

sei();
while (1)
{
```

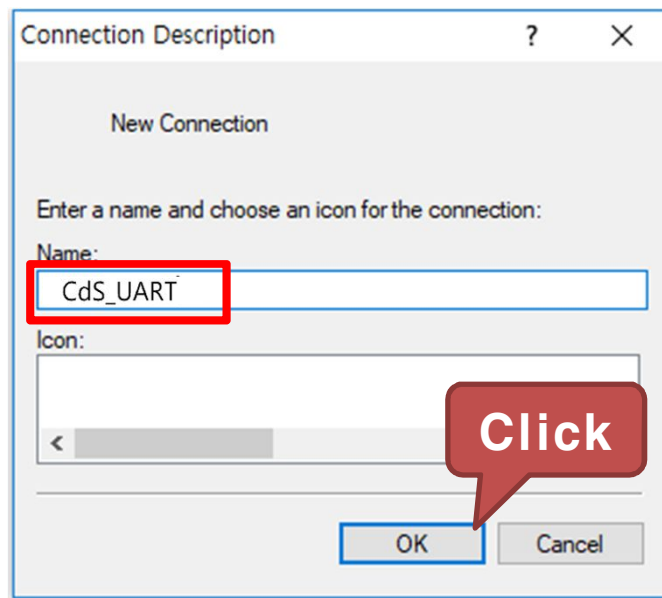
응용 2: A/D 컨버터로 광센서 읽기 2

- 구동 프로그램
 - main.c 코드 작성

```
AdData = ADC_result;           // ADC 값을 변수 AdData에 저장
// ADC 값 출력
putch_Str("\n\r CDS ADC_data : ");
putch((AdData/1000)%10 + '0'); //1000의 자리 출력
putch((AdData/100)%10 + '0'); //100의 자리 출력
putch((AdData/10)%10 + '0');  //10의 자리 출력
putch((AdData)%10 + '0');     //1의 자리 출력
_delay_ms(500);
}
}
SIGNAL(ADC_vect)
{
    cli();
    ADCresult = ADC;           //전압이 디지털로 변환된 값 읽어오기.
    sei();
}
```

응용 2: A/D 컨버터로 광센서 읽기 2

- 하이퍼터미널 실행
 - 연결 설명 창의 이름 란에 "CdS_UART" 라고 확인을 클릭
 - 연결 대상 창의 연결에 사용할 모뎀 번호를 설정하고 확인을 클릭



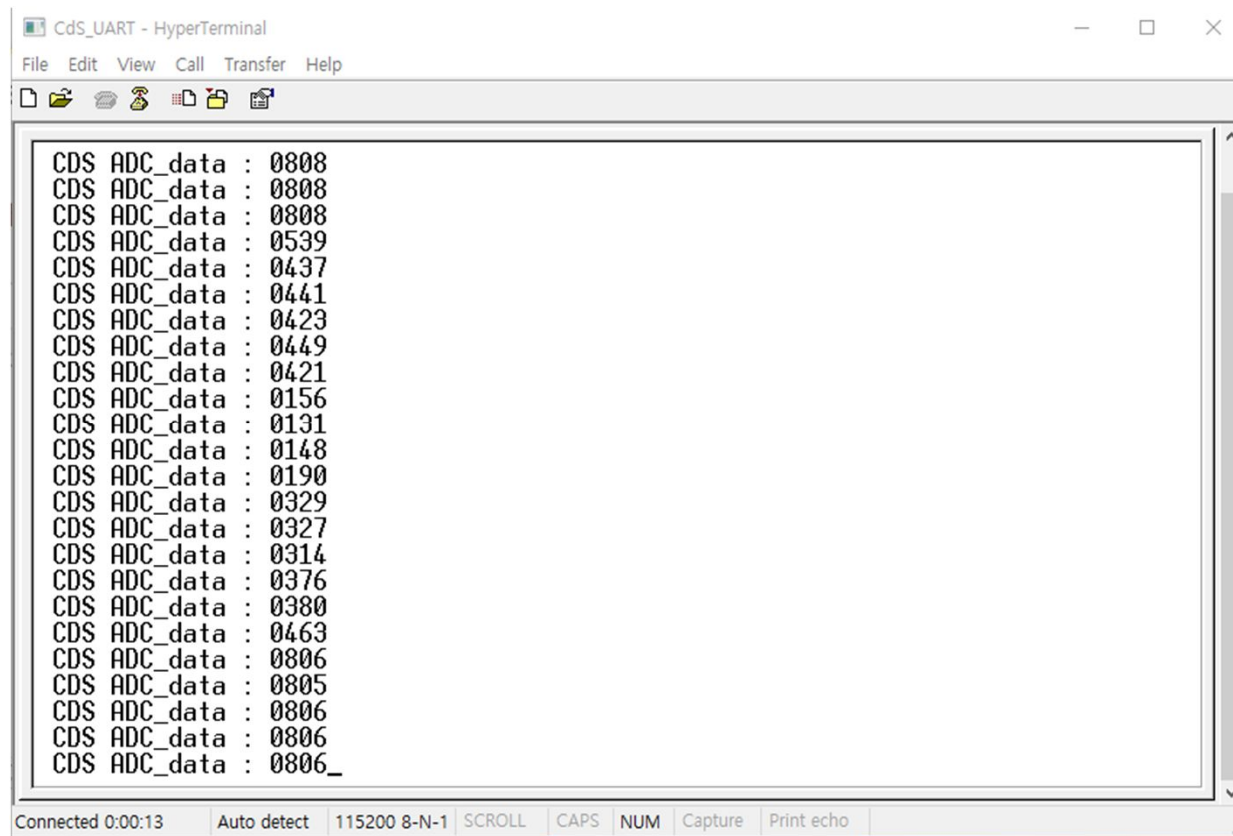
응용 2: A/D 컨버터로 광센서 읽기 2

- 하이퍼터미널 실행
 - 통신 설정을 그림과 같이하고, 확인을 클릭



응용 2: A/D 컨버터로 광센서 읽기 2

- 실행 결과
 - CdS의 조도율에 따라 하이퍼 터미널창에 나오는 결과를 확인



```
CdS_UART - HyperTerminal
File Edit View Call Transfer Help

CDS ADC_data : 0808
CDS ADC_data : 0808
CDS ADC_data : 0808
CDS ADC_data : 0539
CDS ADC_data : 0437
CDS ADC_data : 0441
CDS ADC_data : 0423
CDS ADC_data : 0449
CDS ADC_data : 0421
CDS ADC_data : 0156
CDS ADC_data : 0131
CDS ADC_data : 0148
CDS ADC_data : 0190
CDS ADC_data : 0329
CDS ADC_data : 0327
CDS ADC_data : 0314
CDS ADC_data : 0376
CDS ADC_data : 0380
CDS ADC_data : 0463
CDS ADC_data : 0806
CDS ADC_data : 0805
CDS ADC_data : 0806
CDS ADC_data : 0806
CDS ADC_data : 0806_

Connected 0:00:13  Auto detect  115200 8-N-1  SCROLL  CAPS  NUM  Capture  Print echo
```

실습 3: A/D 컨버터로 전압값 읽기

- 실습 개요
 - ATmega128A 의 A/D 컨버터 기능을 이용하여 가변 저항으로부터 분배되는 전압값을 읽어내어 Text LCD에 출력
 - 가변 저항 : 저항값을 변화시킬 수 있는 저항
- 실습 목표
 - ATmega128A A/D 컨버터의 동작 원리 이해
 - A/D 컨버터 제어 방법 습득(레지스터 설정)
 - 가변 저항의 동작 원리 이해

실습 3: A/D 컨버터로 전압값 읽기

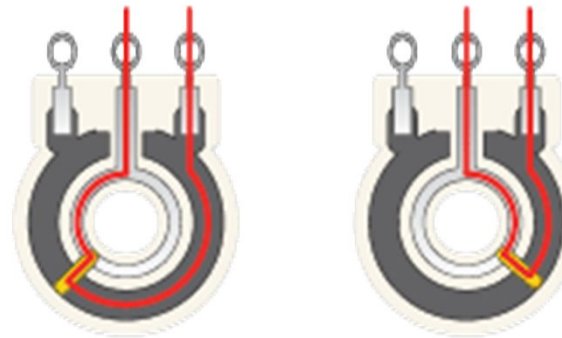
- 사용 모듈

- 가변 저항

- 가변저항은 전자회로에서 저항값을 임의로 바꿀 수 있는 저항기
 - 가변저항을 사용하여 저항을 바꾸면 전류의 크기도 바뀜
 - 장비 운용 중 사용자에게 의해 조작되는 형태와 장비 내부에서 생산 또는 교정 중 조정되는 반고정저항 등의 형태가 있음
 - 소자의 특성상 3단자인 경우가 많다. 일반적으로 회전형의 것이 많이 사용되나 오디오 믹서에서처럼 직동방식도 있음

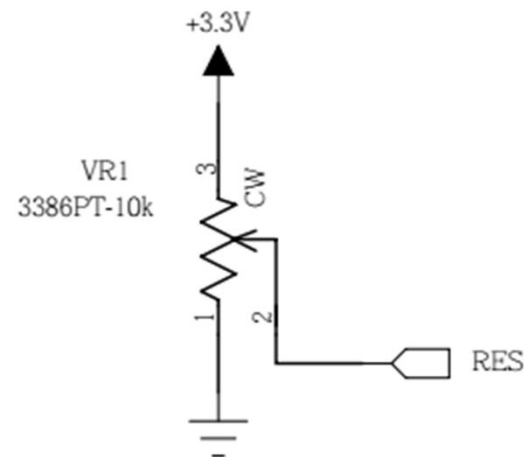
- 가변저항 작동 원리

- 보통 가장자리가 카본띠로 이루어져 있어 다이얼을 돌릴 시 아래 그림처럼 +단자와 -단자간 브릿지 역할을 하는 카본의 길이가 변화하며 그에 따른 저항값도 변함



실습 3: A/D 컨버터로 전압값 읽기

- 사용 모듈
 - 센서 모듈의 가변저항 회로
 - 가변 저항의 회전에 따라 저항 값이 변화하여 VRES 핀으로 출력되는 전압이 변함

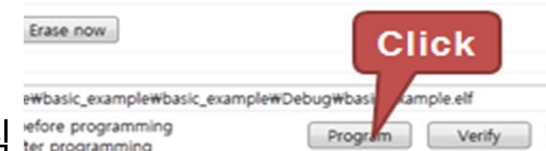
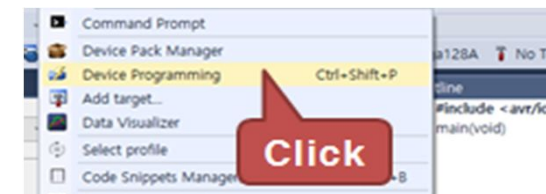
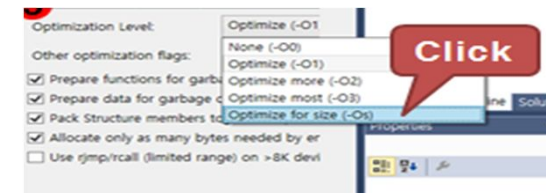
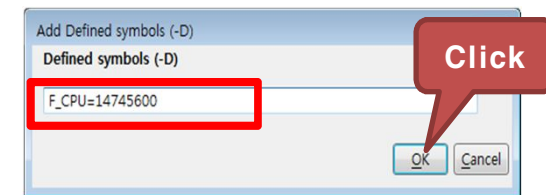


실습 3: A/D 컨버터로 전압값 읽기

- 구동 프로그램 : 사전 지식
 - A/D 컨버터 기능을 이용하여 가변 저항으로 분배되는 전압값의 아날로그 신호를 받아 디지털로 변환한 뒤, 이를 Text LCD에 표시
 - Delay 함수를 이용하여 0.5초 간격으로 데이터를 읽도록 함
 - A/D 컨버터의 설정 방법
 - A/D 컨버터의 입력채널 결정 : 1번 채널 사용
 - 입력 데이터의 정렬 방법과 기준전압 선택 :
 - 데이터 정렬은 디폴트인 우정렬
 - 기준 전압은 외부 기준전압 AREF(3.3V) 사용
 - 프리스케일러와 인터럽트 결정
 - 프리스케일러는 128분주를 사용하고, 인터럽트는 사용하지 않음.
 - 인터럽트를 사용하지 않으므로 설정 끝
 - A/D 컨버팅 제어
 - 설정을 마친 후, ADCSR의 6번 비트를 세트하여 A/D 변환을 시작
 - A/D 변환 도중에는 변환 완료 플래그를 주기적으로 점검하면서 A/D 데이터 레지스터로부터 데이터를 읽으면 됨
 - 이때, 반드시 ADCL(하위 데이터)를 먼저 읽어서 저장한 다음에 ADCH(상위 데이터)를 읽어 내야 함

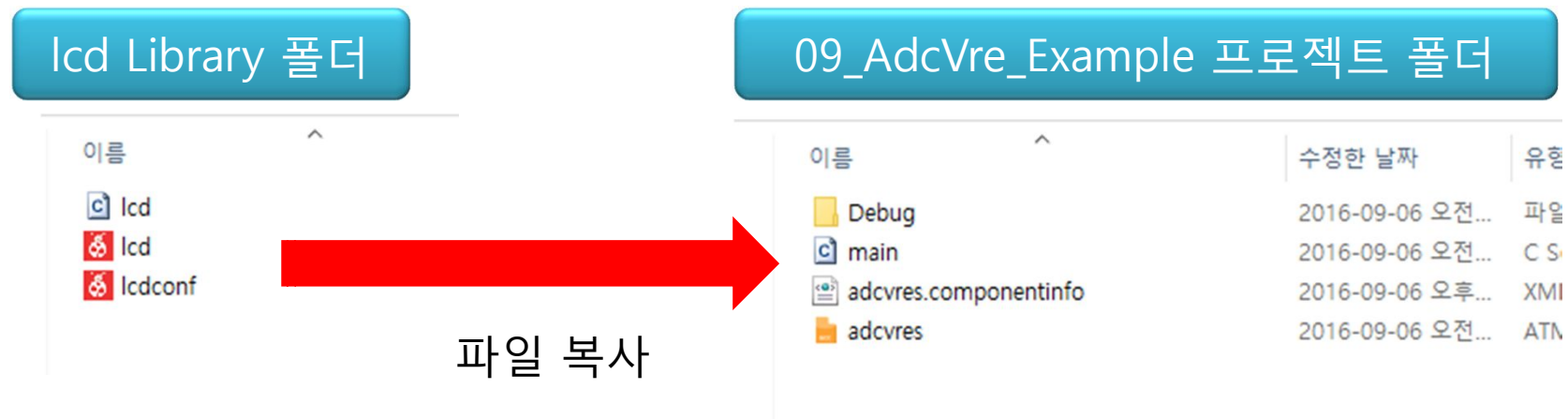
실습 3: A/D 컨버터로 전압값 읽기

- 예제 프로그램 작성 및 구동
 - Atmel Studio 실행
 - New Project 생성
 - Name : 09_AdcVre_Example, Location : D:\WAVR_Example
 - Device Selection : ATmega128A
 - 프로젝트 설정
 - Project 탭에서 "... Properties..." 선택
 - Toolchain -> AVR/GNU C Compiler에서
 - Symbols -> F_CPU=14745600 추가
 - Optimization -> Optimize for size (-OS) 선택
 - 저장 (Ctrl+S)
 - 소스코드 작성
 - 프로젝트 빌드
 - Build 탭에서 "Build Solution" 클릭
 - 프로그래밍
 - Tool 탭에서 "Device Programming" 클릭
 - AVRISP mkII, ATmega128A 선택 후 "Apply" 클릭
 - 인식 완료되면, Memories 탭 선택, "Program" 클릭



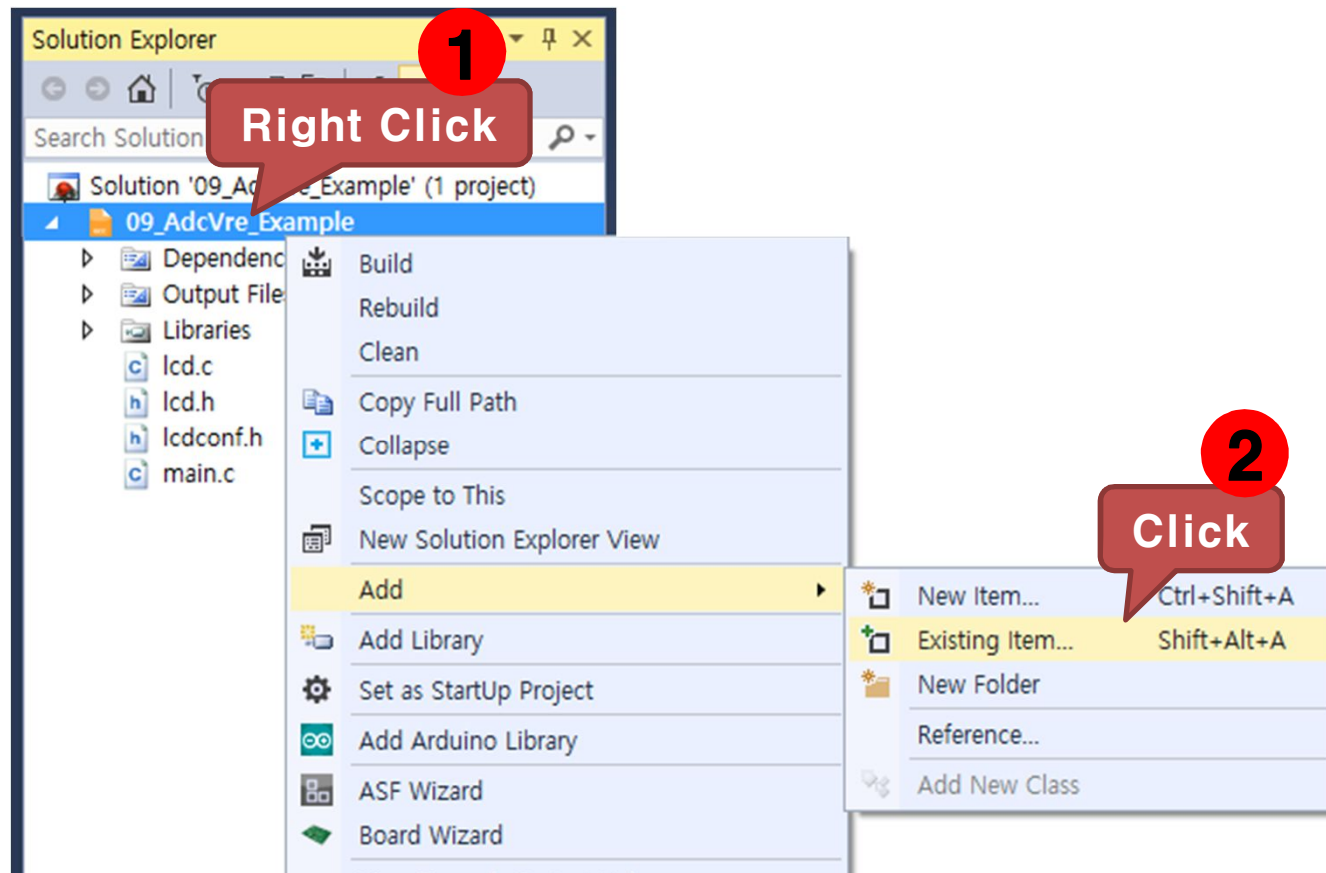
실습 3: A/D 컨버터로 전압값 읽기

- 구동 프로그램 : Atmel Studio 에서 라이브러리 함수 추가하는 방법
 - 라이브러리 함수파일을 프로젝트내로 복사
 - lcd.c, lcd.h, lcdconf.h 파일은 "AVR_Example\library\lcd" 폴더에 존재
 - 라이브러리 파일은 Atmel Studio에서 새 프로젝트를 생성한 후, 새 프로젝트가 생성된 폴더안에 있는 프로젝트 폴더에 복사



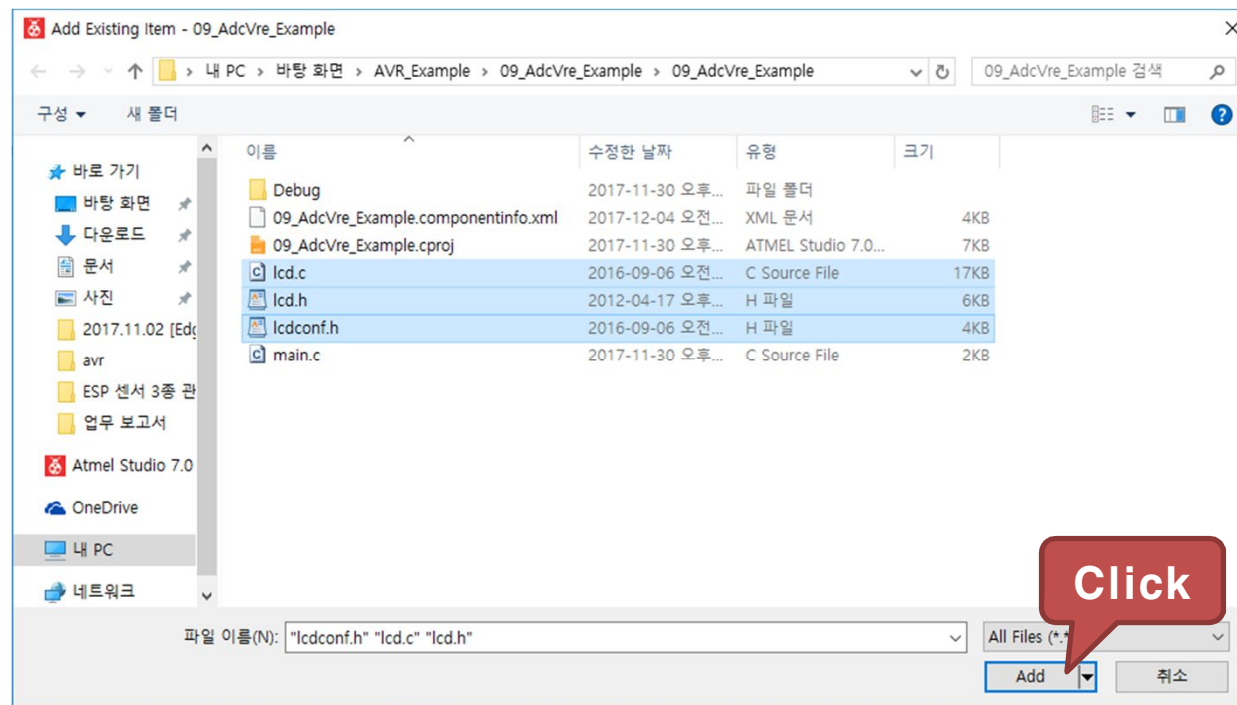
실습 3: A/D 컨버터로 전압값 읽기

- 구동 프로그램 : Atmel Studio 에서 라이브러리 함수 추가하는 방법
 - Atmel Studio 상에서 프로젝트의 솔루션 탐색기에서 라이브러리 파일들을 추가



실습 3: A/D 컨버터로 전압값 읽기

- 구동 프로그램 : Atmel Studio 에서 라이브러리 함수 추가하는 방법
 - lcd.c, lcd.h, lcdconf.h 파일을 각각 추가
 - 또는 한번에 세 파일을 동시에 추가
 - Ctrl 키를 누르고 파일 세개를 순서대로 클릭하여 세파일을 동시에 선택하고 한번에 추가



실습 3: A/D 컨버터로 전압값 읽기

- 구동 프로그램
 - main.c 코드 작성

```
#include <avr/io.h>      // AVR 입출력에 대한 헤더 파일
#include <util/delay.h>    // delay 함수사용을 위한 헤더파일
#include "lcd.h"          // Text LCD를 사용하기 위한 헤더 파일

void printf_2dot1(uint16_t _temp); // 전압을 LCD에 출력하는 함수

int main(void) {
    unsigned int AdData = 0;      // 10bit ADC값 저장 변수
    float v_temp;                 // 전압값을 계산할 변수
    lcdInit();                    // Text LCD를 초기화

    ADMUX = 0x01;                 // ADC1선택
    // single mode, 1번 채널, 3.3V 외부 기준전압(AREF)
    ADCSRA = 0x87;                // 10000111
    // ADC 허가, 128분주
```

실습 3: A/D 컨버터로 전압값 읽기

- 구동 프로그램
 - main.c 코드 작성

```
int main(void) {  
    //...  
    while (1) {  
        // ADC1을 통한 ADC값 읽어오기  
        ADCSRA |= 0x40;      // ADSC AD 개시(Start)  
        while((ADCSRA & 0x10) == 0x00); // ADIF AD 다 될 때까지 기다림  
        AdData = ADC;        // 전압이 디지털로 변환된 값 읽어오기  
        // [(adc/1023*3.3) /1023 ] => 실제 전압값  
        // 실제값에 10을 곱하여 소수점 첫째자리까지 표현  
  
        v_temp = (float) AdData * 33 / 1023;  
        printf_2dot1(v_temp);      // 전압값을 text LCD에 출력  
        _delay_ms(500);  
    }  
}
```

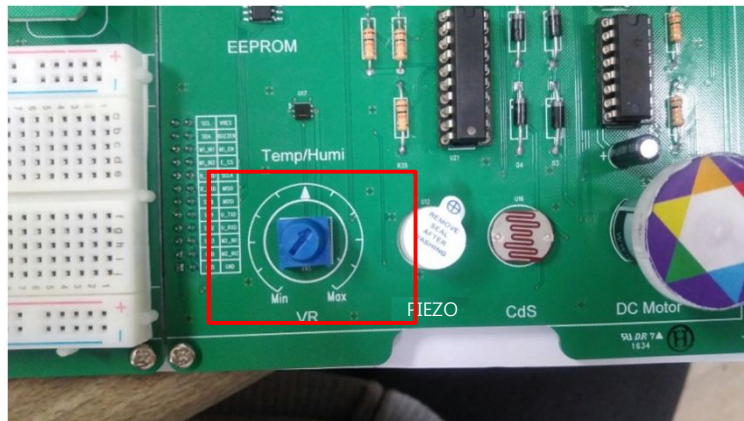
실습 3: A/D 컨버터로 전압값 읽기

- 구동 프로그램
 - main.c 코드 작성

```
void printf_2dot1(uint16_t _temp) {
    uint8_t s100,s10;
    lcdGotoXY(0,0);          // 커서위치를 첫번째 줄, 첫번째 칸으로 이동
    lcdPrintData(" Vres: ",7); // " Vres: " 출력
    s100 = _temp/100;          // 100의 자리 추출
    if(s100> 0) lcdDataWrite(s100+'0'); // 100의 자리 값이 있으면 출력
    else lcdPrintData(" ",1);   // 100의 자리 값이 없으면 빈칸 출력
    s10 = _temp%100;           // 100의 자리를 제외한 나머지 추출
    lcdDataWrite((s10/10)+'0'); // 10의 자리 추출하여 출력
    lcdPrintData(".",1);        // 소숫점 출력
    lcdDataWrite((s10%10)+'0'); // 1의 자리 추출하여 출력
    lcdDataWrite('V');          // 전압 단위 출력
}
```

실습 3: A/D 컨버터로 전압값 읽기

- 실행 결과
 - 가변 저항을 회전시키면 출력되는 전압값이 변함

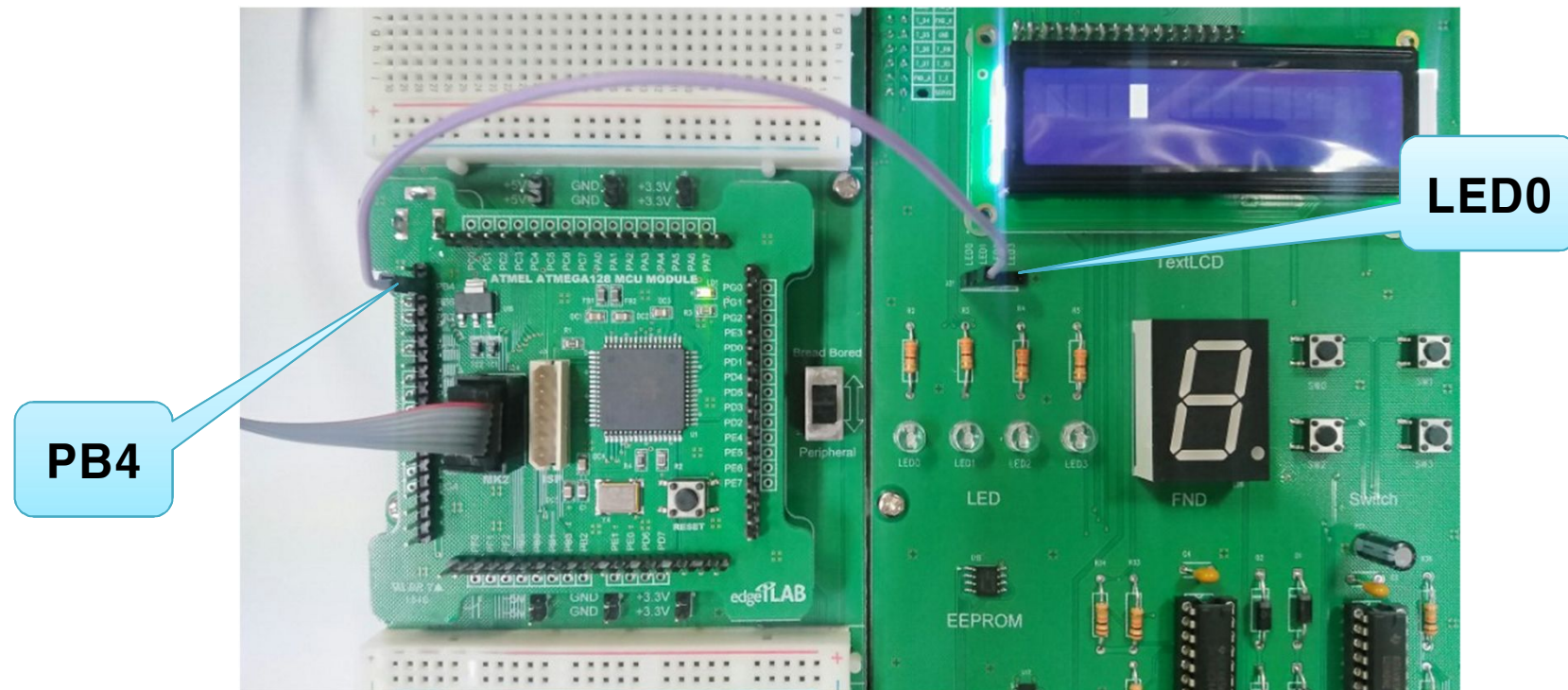


응용 3: A/D 컨버터로 가변저항 값에 따라 LED 밝기 제어하기

- 실습 개요
 - ATmega128A 의 A/D 컨버터 기능을 이용하여 가변 저항으로부터 분배되는 전압 값을 읽어내어 LED 밝기를 제어
 - 가변 저항 : 저항값을 변화시킬 수 있는 저항
- 실습 목표
 - ATmega128A A/D 컨버터의 동작 원리 이해
 - A/D 컨버터 제어 방법 습득(레지스터 설정)
 - 가변 저항의 동작 원리 이해

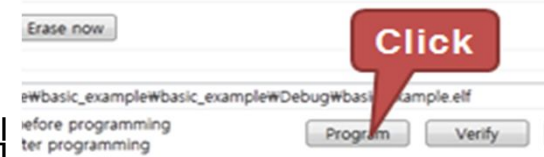
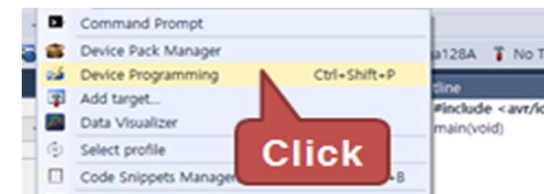
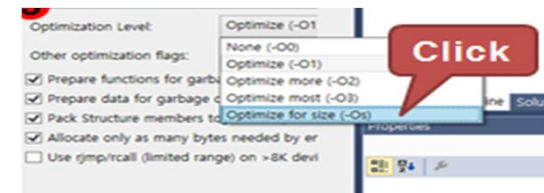
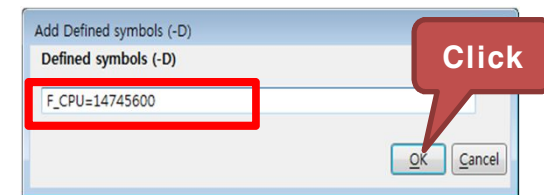
응용 3: A/D 컨버터로 가변저항 값에 따라 LED 밝기 제어하기

- 실습 준비
 - 다음 그림과 같이 연결
 - Edge-MCU 보드의 PB4 → Edge-Peri 보드의 LED0(LED 상단)



응용 3: A/D 컨버터로 가변저항 값에 따라 LED 밝기 제어하기

- 예제 프로그램 작성 및 구동
 - Atmel Studio 실행
 - New Project 생성
 - Name : 09_AdcVre_Application, Location : D:\AVR_Example
 - Device Selection : ATmega128A
 - 프로젝트 설정
 - Project 탭에서 "... Properties..." 선택
 - Toolchain -> AVR/GNU C Compiler에서
 - Symbols -> F_CPU=14745600 추가
 - Optimization -> Optimize for size (-OS) 선택
 - 저장 (Ctrl+S)
 - 소스코드 작성
 - 프로젝트 빌드
 - Build 탭에서 "Build Solution" 클릭
 - 프로그래밍
 - Tool 탭에서 "Device Programming" 클릭
 - AVRISP mkII, ATmega128A 선택 후 "Apply" 클릭
 - 인식 완료되면, Memories 탭 선택, "Program" 클릭



응용 3: A/D 컨버터로 가변저항 값에 따라 LED 밝기 제어하기

- 구동 프로그램
 - main.c 코드 작성

```
#include <avr/io.h>           //AVR 입출력에 대한 헤더 파일
#include <util/delay.h>       // delay 함수사용을 위한 헤더파일

int main(void)
{
    unsigned long AdData = 0;    //10bit ADC값 저장 변수

    DDRB = 0x10;                // 포트B 를 출력포트로 설정한다.
                                // PB4와 LED0을 연결한다.

    TCCR0 = 0x62;                // PC PWM 모드, 8 분주
    // PWM 주기 : F_CPU/256/2/8 = 3.6 KHz
    // 업카운트시 Clear, 다운카운트시 Set 으로 설정
    TCNT0 = 0x00;                // 타이머0 카운터를 초기화 한다.

    ADMUX = 0x41;                // ADC1선택, 3.3V 외부 기준전압(AREF)
    ADCSRA = 0x87;               // 10000111 ADC 허가, single mode, 128분주
```

응용 3: A/D 컨버터로 가변저항 값에 따라 LED 밝기 제어하기

- 구동 프로그램
 - main.c 코드 작성

```
while (1)
{
    // ADC0을 통한 ADC값 읽어오기
    ADCSRA |= 0x40; //ADSC AD 개시(Start)
    while((ADCSRA & 0x10) == 0x00); //ADIF AD 다 될 때까지 기다림.
    AdData = ADC;      //전압이 디지털로 변환된 값 읽어오기.

    if(AdData > 800) // 최대값을 800으로 설정
        AdData = 800;
    AdData = AdData * 255 / 800;
    OCR0 = AdData;    // ADC 값의 따라 LED를 제어한다.

    _delay_ms(200);
}
```

응용 3: A/D 컨버터로 가변저항 값에 따라 LED 밝기 제어하기

- 실행 결과
 - 가변 저항 값에 따라 LED의 밝기가 변함

