[PIC 마이컴 실험하기] 하드웨어 및 소프트웨어 필요사항

마이컴 실험실

2011/09/29 11:18 http://blog.naver.com/ubicomputing/150120079097

하드웨어 및 소프트웨어 필요사항

임베디드시스템을 개발하기 위해서는 몇몇 하드웨어와 소프트웨어가 필요합니다. 필요한 하드웨어가 프로젝트의 성격이나 복잡성에 따라 변하긴 하지만, 아래에 언급하는 하드웨어는 여기서 언급하는 실험들을 진행하는데 필요합니다.

1. 실험회로를 구성하기 위한 브레드보드. 브레드보드를 이용하면 회로를 마음대로 구성하고 바꾸어 볼수 있습니다. 회로가 잘 동작하는것이 검증되면 PCB로 회로를 옮기면 됩니다.
2. 마이컴 칩(PIC16F688, PIC16F628A)
3. PIC 프로그래머. 펌웨어를 PIC 마이컴에 로드하기 위한 장비로 ICSP(In-Circuit Serial Programming)기능이 있는 것으로 준비해야합니다.
4. PC. PC는 펌웨어를 개발하여 컴파일하고 PIC 프로그래머를 통해 펌웨어를 마이컴에 심는데 사용됩니다.
5. 레귤레이트된 5VDC 전원. 브레드보드에 전원을 공급하는데 사용됩니다.
6. 테스트장비로 사용될 디지털 멀티미터.
7. 저항, LED, 캐패이시터, 전선 등의 기타 부품들

mikroProg PIC계열 프로그래머

필요한 소프트웨어 툴

위의 하드웨어와 함께 아래의 소프트웨어 제품들이 실험에 필요합니다.

1. 컴파일러. 펌웨어를 개발하고 컴파일하는데 필요합니다. 여기서는 PIC용 mikroC Pro가 사용됩니다.
2. MCU 프로그래밍 소프트웨어. 하드웨어 프로그래머 업체가 공급합니다. mikroC Pro내에 프로그래밍 소프트웨어가 탑재되어 있습니다.

가치창조기술 | www.vctec.co.kr

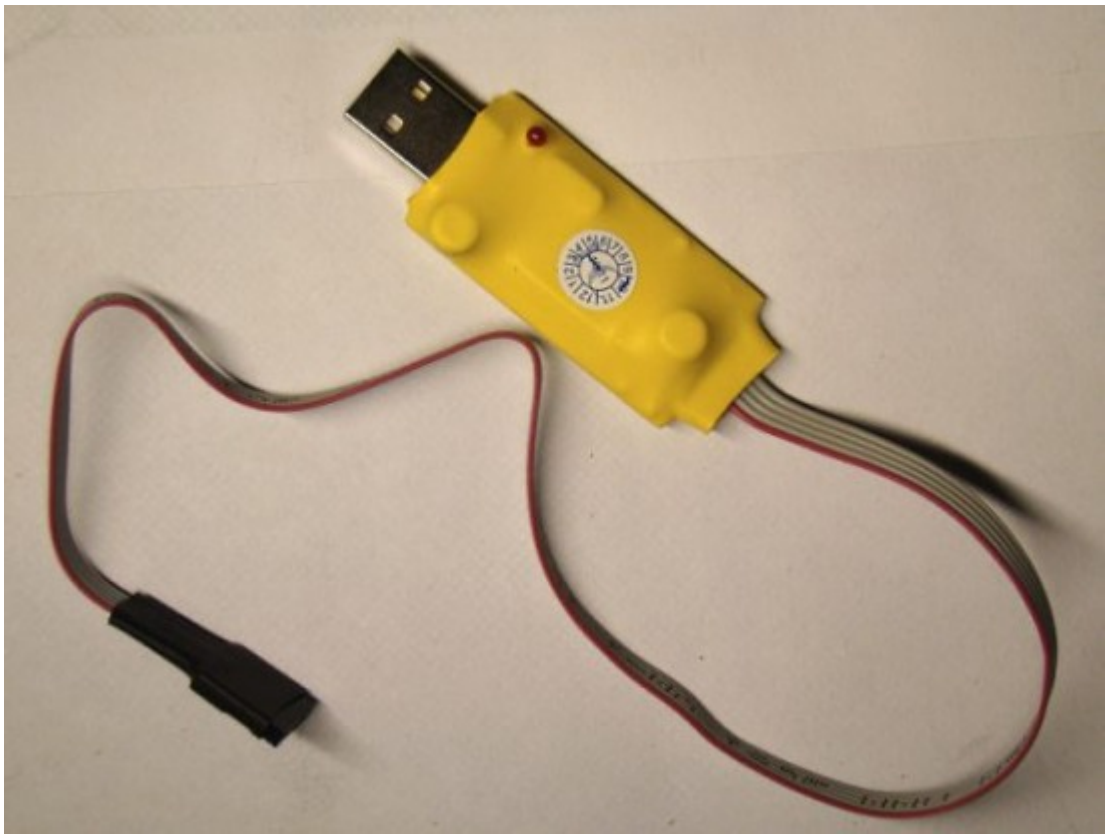
[PIC 마이컴 실험하기] PIC 프로그래머 선택하기

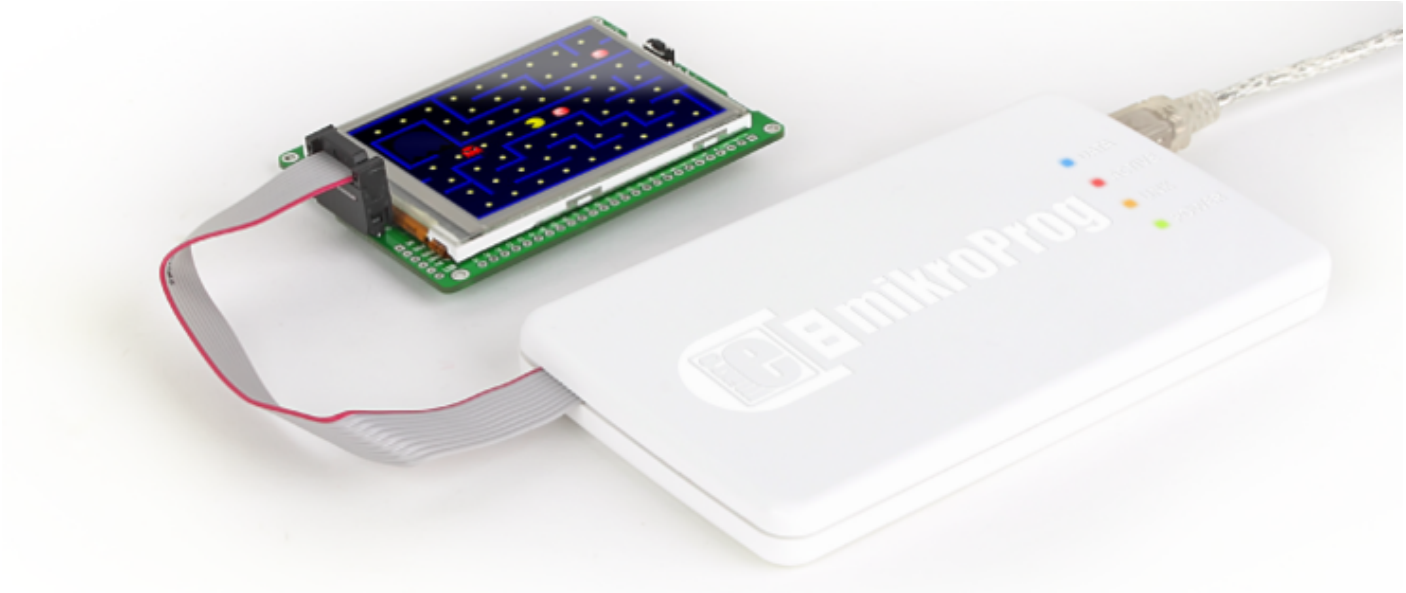
마이컴 실험실

2011/09/29 12:09 http://blog.naver.com/ubicomputing/150120083050

PIC 마이컴의 세계에 처음 발을 들여놓으신 분들은 어떤 프로그래머를 사야할지 궁금하실 껍니다. 아마도 굉장히 많은 PIC프로그래머가 있기때문일텐데요.

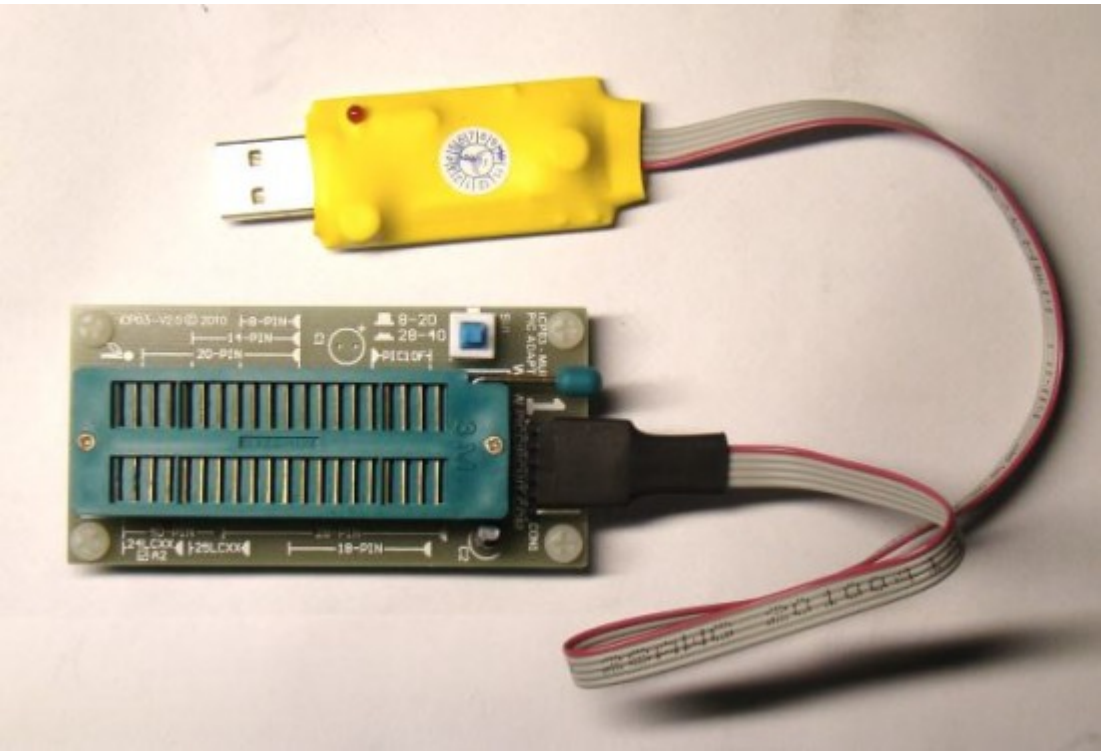
여기서 기준을 제시하겠습니다.
먼저, ICSP(In-Circuit Serial Programming)기능이 있는 USB PIC 프로그래머입니다. ICSP기능이 있으면 MCU가 타켓 보드에 붙어있어도 펌웨어를 재 프로그래밍 할수 있어, 프로그래밍 할때마다 MCU를 탈거해야하는 수고를 덜 수가 있습니다. ICSP의 사용을 위해서는 MCU와 프로그래머간 총 5개의 연결을 필요로 합니다. Vcc, Gnd, programming voltage (Vpp), data (ICSPDAT 혹은 PGD), clock (ICSPCLK 혹은 PGC) 입니다.





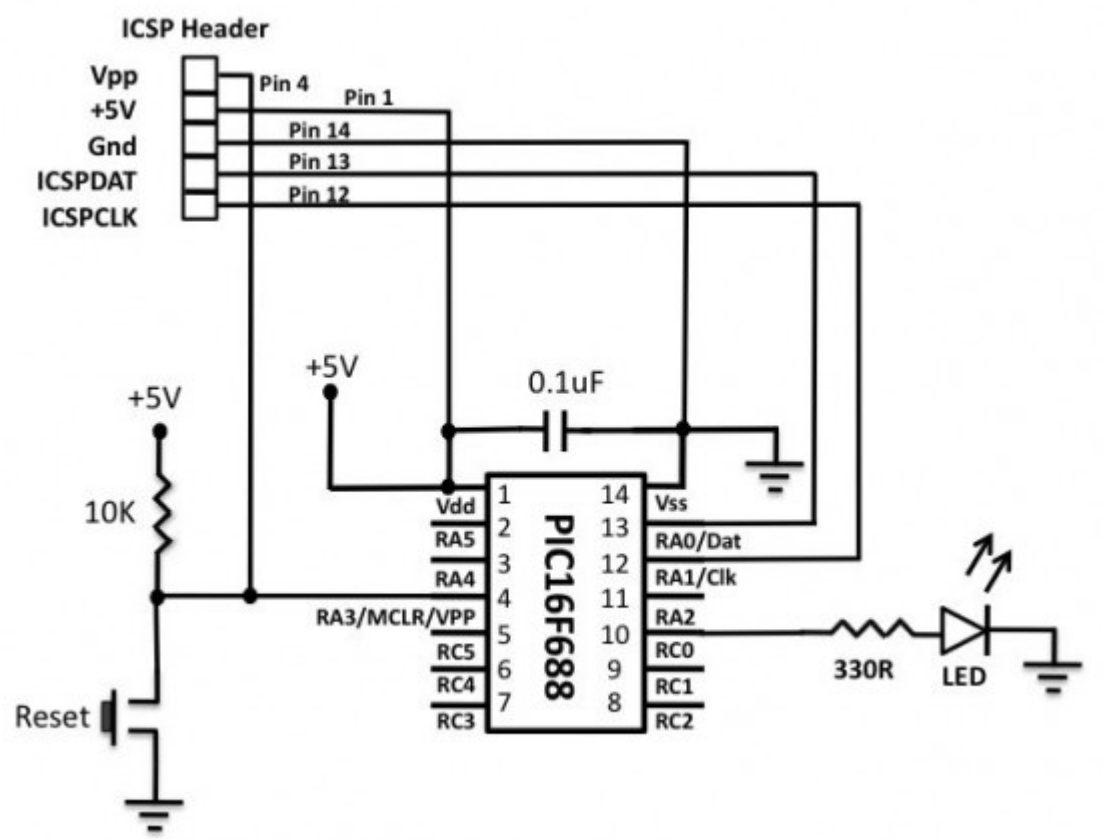
<프로그래머>

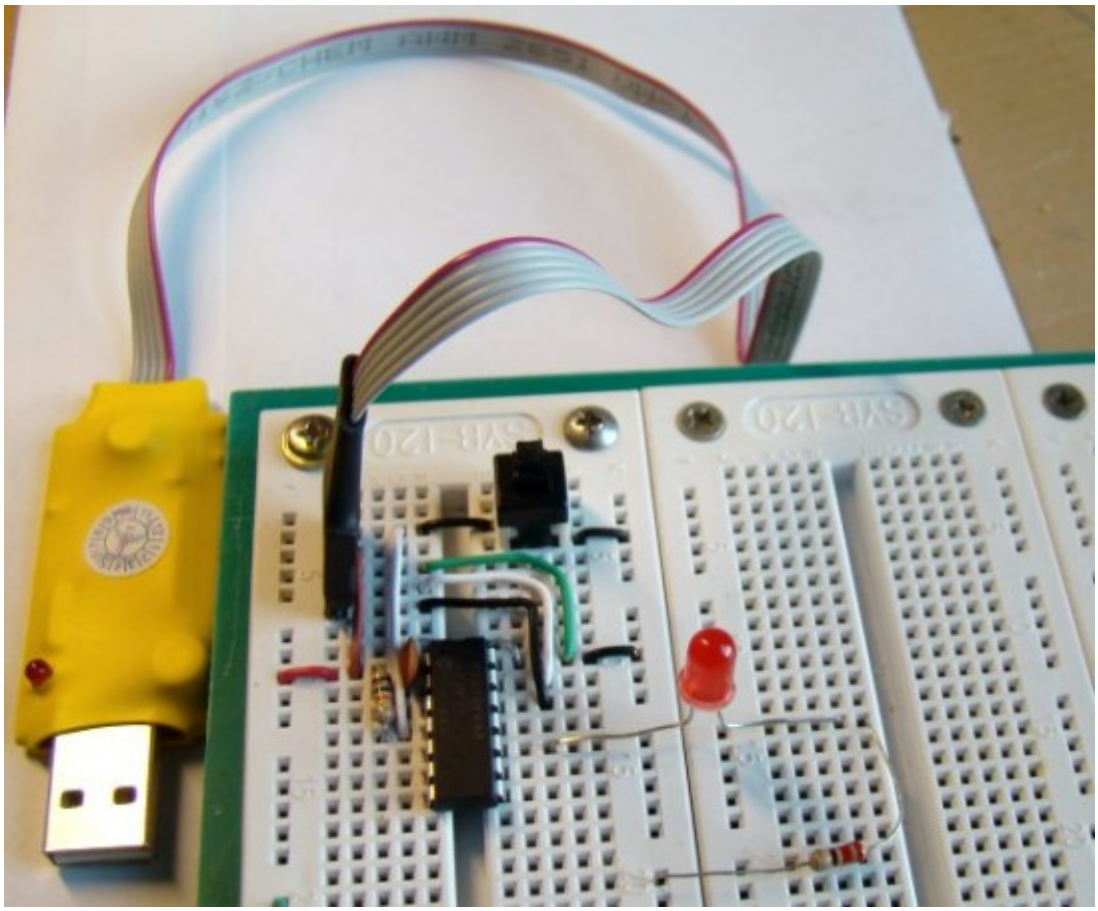
타켓보드에서 MCU를 프로그래밍하기 위해서는 보드에 ICSP헤더(5핀)를 가지고 있어야 합니다. Vpp, +5V, Gnd, ICSPDAT, ICSPCLK 이며 Vpp핀은 와이어상에 빨간색으로 표시가 되어 있습니다. ZIF 소켓을 이용하여 다음과 같이 연결하여 다양한 종류의 PIC MCU를 프로그래밍 할 수도 있습니다.

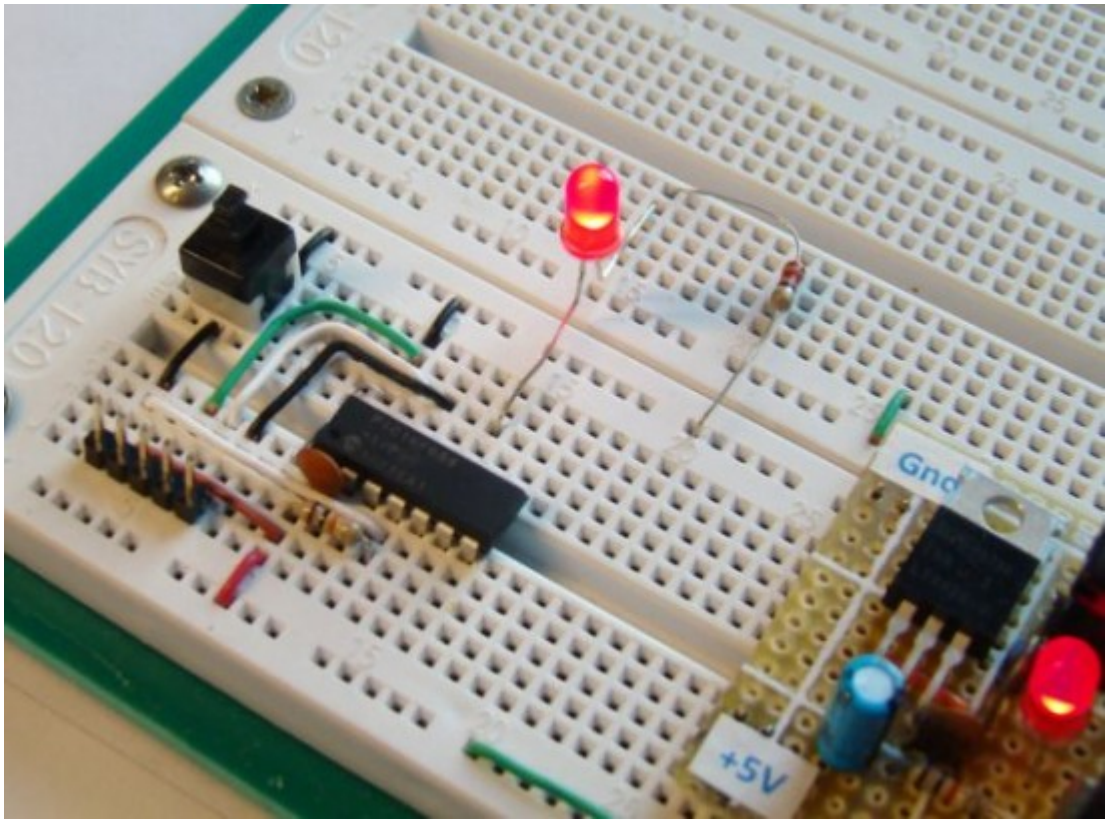



USB
PICFlash2
MicroElectronika
MCU VCC
RB6
RB7
MCLR
GND
MCU VCC
MCU RB6
MCU RB7
MCU MCLR
GND
DIP8
DIP18
DIP14

LED깜박이기 프로그램을 PIC16F688에 올려 프로그래머를 테스트하여 보았습니다.







가치창조기술 | www.vctec.co.kr

[PIC 마이컴 실험하기] 브레드보드에 안정적 전원공급하기

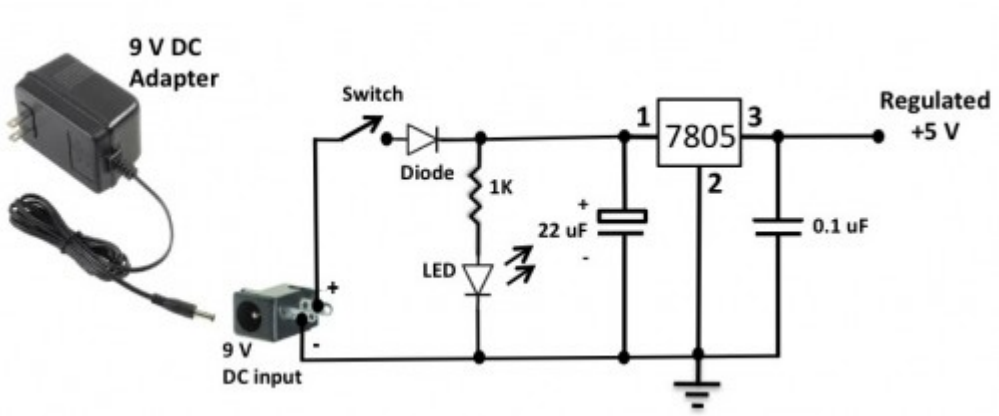
마이컴 실험실

2011/09/29 15:39 http://blog.naver.com/ubicomputing/150120100749

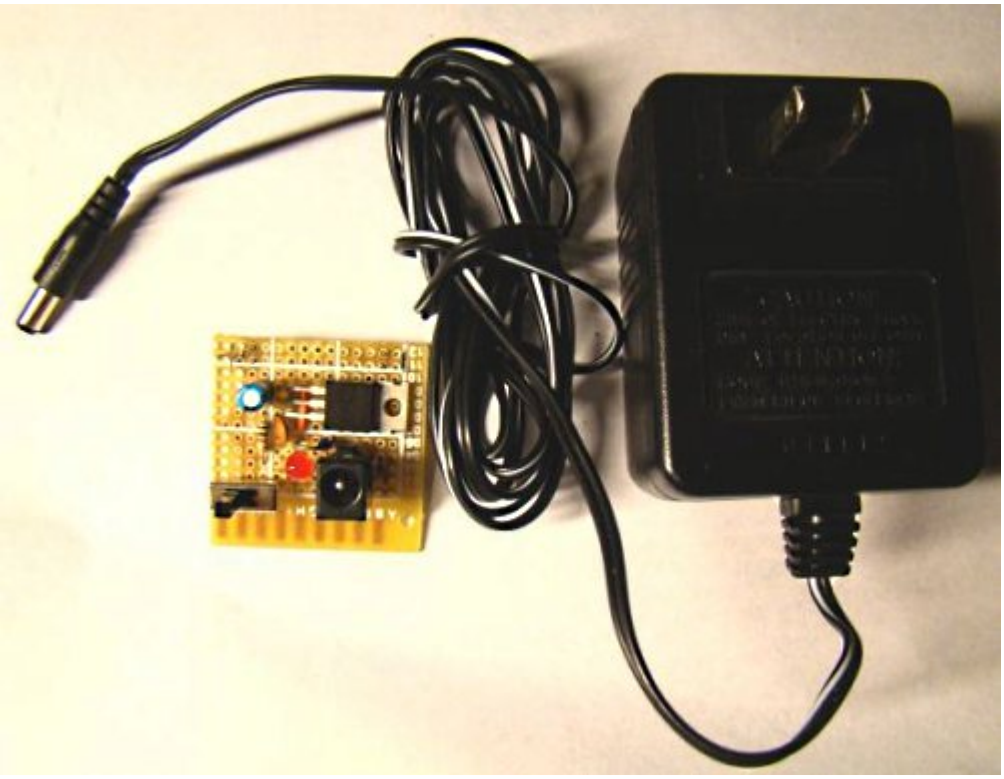
임베디드 시스템은 동작하기 위해서는 전원이 필요합니다. 대부분의 부품들은 다양한 전압에서 동작을 합니다. 예를 들어 PIC16F688은 2V에서 5.5V에서 동작가능합니다. 이말은 AA 사이즈 배터리 세개(4.5V)면 전원을 공급할 수 있고 전압이 2.2V이하로 떨어지지 않는한 잘 동작한다는 의미입니다.

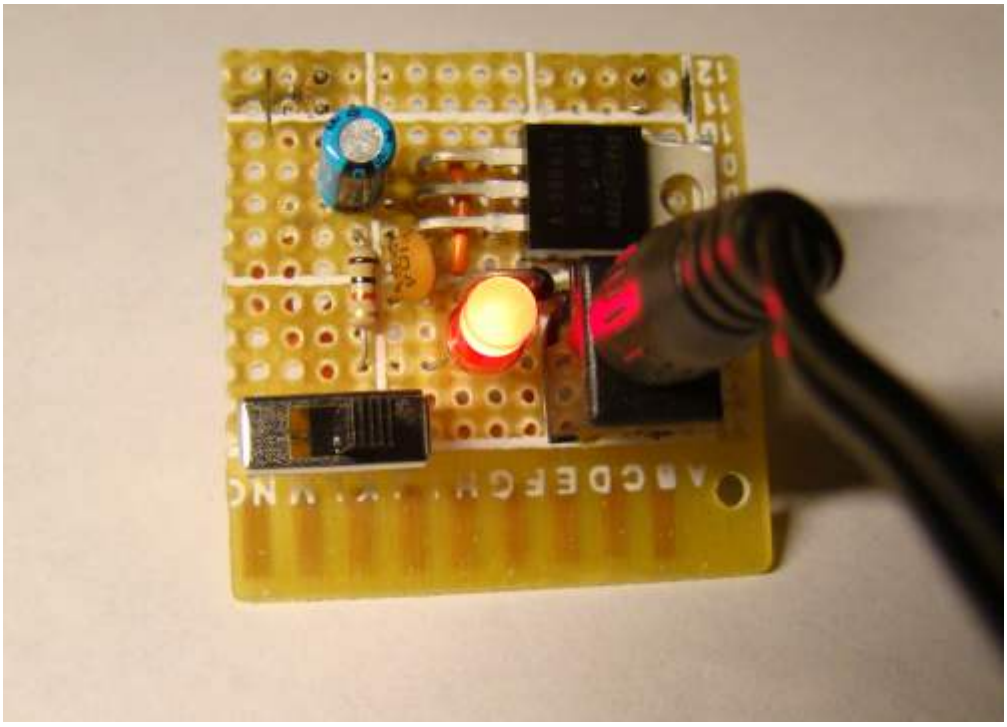
하지만 임베디드 어플리케이션에서는 전압이 꾸준히 유지되어야 하는 어플리케이션도 있습니다. 예를 들어 아날로그-디지털 컨버터를 사용하는 어플리케이션이 그렇습니다. ADC는 아날로그 입력신호에 대해 정확한 디지털 카운트를 제공하기 위해서 고정 리퍼런스전압을 필요로합니다. 그런데 리퍼런스 전압이 안정적이지 않으면 ADC 출력은 의미가 없는 것입니다. 그래서 브레드보드에 레귤레이트된 +5V 전원을 공급하기 위해서 LM7805 리니어 레귤레이터 IC를 사용하였습니다.

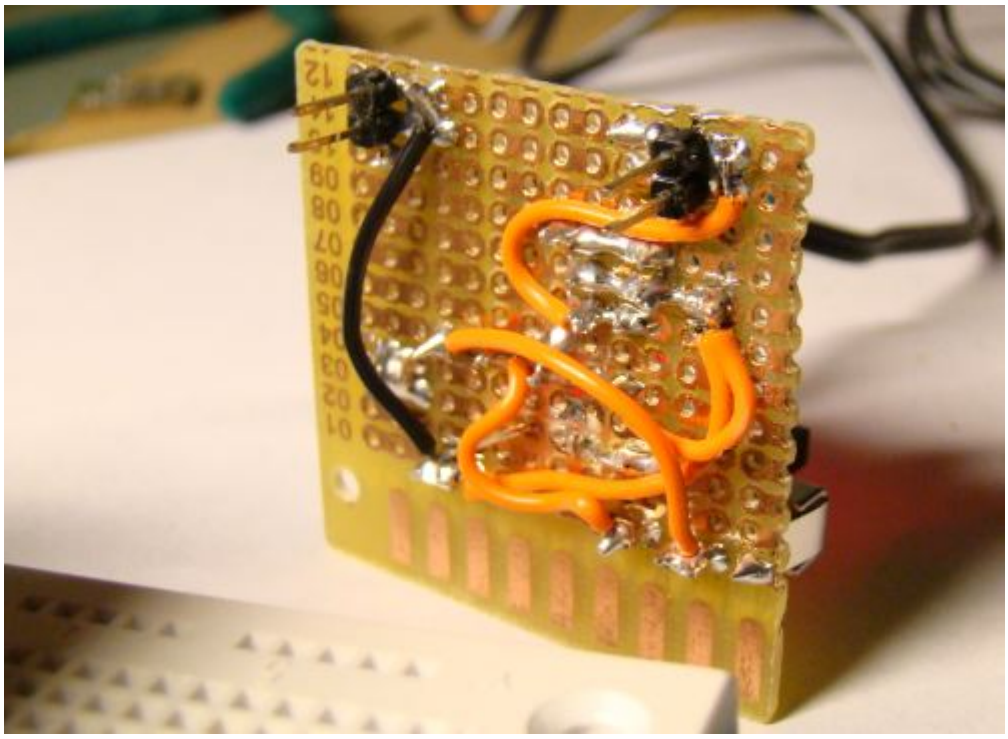
LM7805은 DC 입력 전압 7-25V를 5v로 변환하며 두개의 외부 캐패이시터를 필요로 합니다.



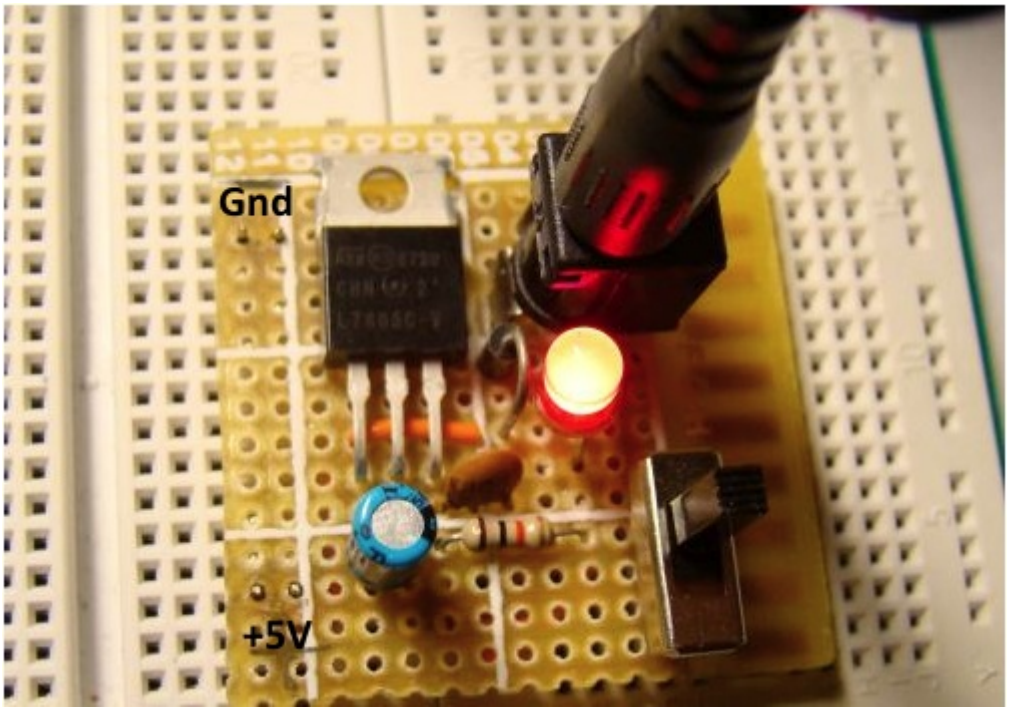
1암페어 9VDC 전원어댑터를 통해 LM7805에 전원을 공급합니다. 12-24V 전원 어댑터도 동작하지만 전원이 높으면 그만큼 열이 많이 납니다. 위의 회로를 브레드보드에 납땜합니다.







수컷 헤더 핀을 5V와 Gnd에 납땜하여 브레드보드에 꼽을 수 있게 만들었습니다.



전원공급유닛이 이제 준비 되었습니다. 디지털 멀티미터기로 출력전압이 5V인지 확인해보세요.

가치창조기술 | www.vctec.co.kr

[PIC 마이컴 실험하기] 브레드보드에 기초적인 PIC16F688회로 셋업하기

마이컴 실험실

2011/09/29 19:07 http://blog.naver.com/ubicomputing/150120119369

이전 게시물에서 언급했지만 모든 회로는 브레드보드상 구성될 것입니다. figure 1은 PIC16F688의 핀 다이어그램으로 내장 오실레이터를 가진 14핀 mcu입니다. 각각 방향 컨트롤이 가능한 12개의 I/O핀을 제공하고 있으며, LED를 직접적으로 구동시킬 수 있습니다. 12개의 핀중 8개의 핀은 10비트 ADC 채널로 사용될 수 있습니다. PIC16F688의 다른 기능은 차후에 살펴보기로 하고 브레드보드에 셋업할 간단한 회로를 살펴보도록 하겠습니다.

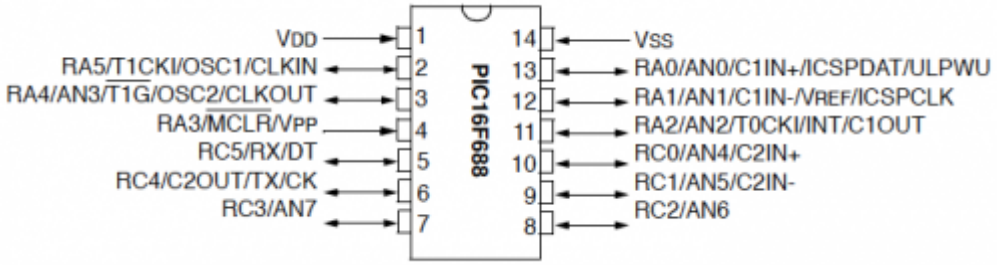


Figure 1: PIC16F688 핀 다이어그램

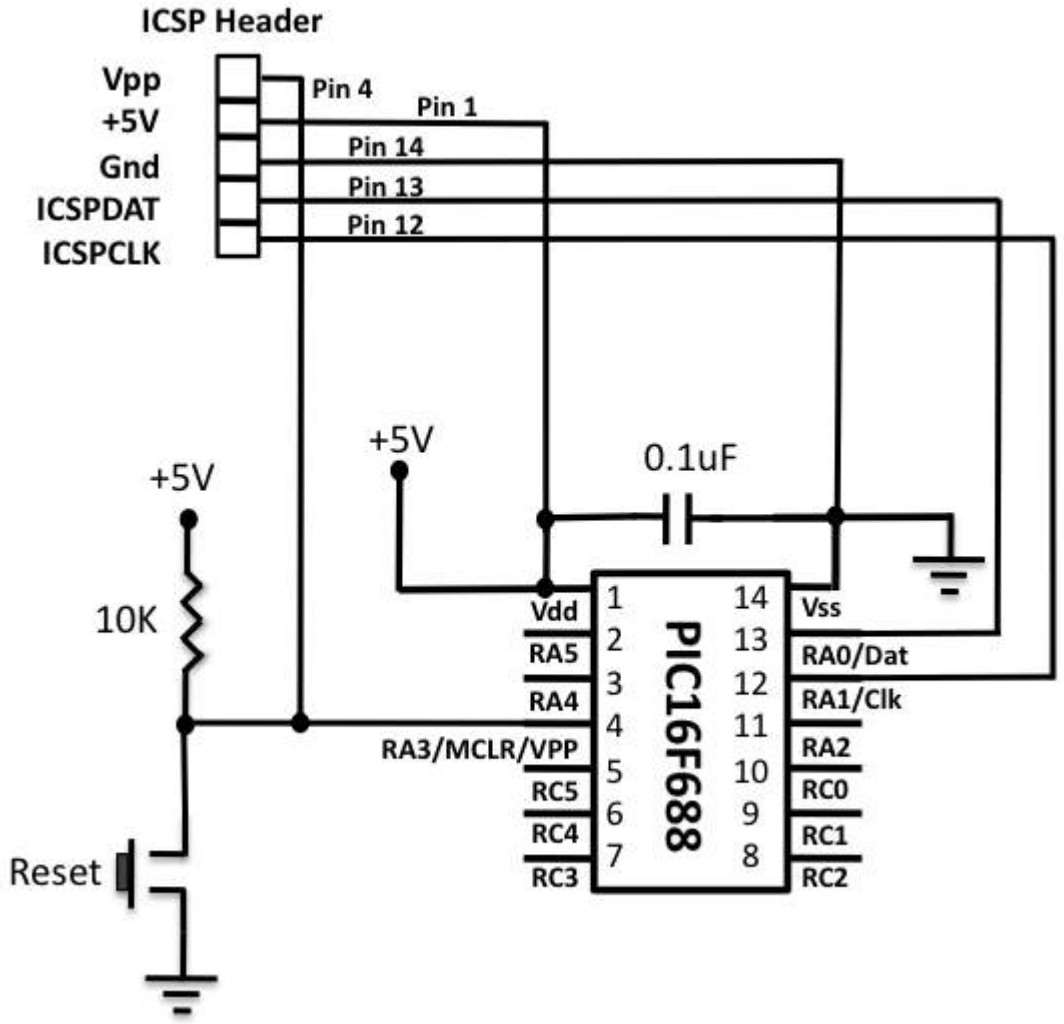


Figure 2: 리셋스위치를 연결한 기초 셋업

Figure 2 는 PIC16F688 마이컴에 아주 최소한의 컴포넌트를 연결하여 본 것입니다. 내장 오실레이터를 사용할 것이기 때문에 외장 오실레이터 회로도 없습니다만 이는 두개의 핀(2, 3번)을 I/O용도로 사용할 수 있게 하여 줍니다.

MCU를 리셋하고 프로그램을 처음부터 다시 실행해야 하는 많은 경우들이 있습니다. 리셋을 위해서는 figure2에 나온 것처럼 리셋버튼을 MCU의 MCLR(Master Clear)핀에 연결하면 됩니다. 보통 MCLR 핀은 로직 1상태에 있다가 리셋버튼이 눌리게 되면 로직 0 상태로 갑니다. MCLR핀이 로직 0상태가 되면 PIC16F688을 리셋하게 됩니다. 버튼이 정상으로 돌아오면 MCU는 프로그램의 처음부터 다시 실행하며 정상적으로 동작하게 됩니다. MCLR핀은 소프트웨어를 통해서 비활성화 시키고 I/O핀으로 사용할 수 도 있습니다.

PIC16F688은 타켓보드상에서 두개의 핀[*ICSPDAT (RA0, 13)*, *ICSPCLK (RA1, 12)]*을 이용하여 시리얼하게 프로그램이 가능합니다. 이런 기술은 *in circuit serial programming*(ICSP)으로 알려져 있습니다. ICSP동안에는, MCLR/Vpp 핀이 프로그래밍장비에 의해 13V까지 전압이 공급되기때문에 실제회로는 이런 높은 전압으로부터 보호되어야 합니다. 그래서 10K 저항이 5V 전원과 MCLR/Vpp 사이에 연결되고, 이는 프로그래밍시에 발생할수 있는 전압문제를 막을수 있습니다. 프로그래밍이 되고 있을때는 절대로 리셋버튼을 누르지 마십시오. 위험합니다.

RA1/ICSPCLK 핀은 클럭 라인으로 ICSP동안에 프로그래밍장치에 의해 동작됩니다. 그리고 RA0/ICSPDAT은 양방향 데이타라인으로 역시 프로그래머에 의해, 또 데이타검증시에는 PIC16F688에 의해 동작됩니다. 이런 핀들은 프로그래밍시에 실제회로에 영향을 주지 않기 위해 반드시 실제회로와 분리되어야 합니다.

그리고 100 nF 캐페이시터는 디커플링 캐페이시터 역활을 하기 때문에 MCU 가까이에 위치 시켜야 합니다. 아래 브레드보드상의 컴포넌트 위치를 살펴보세요.

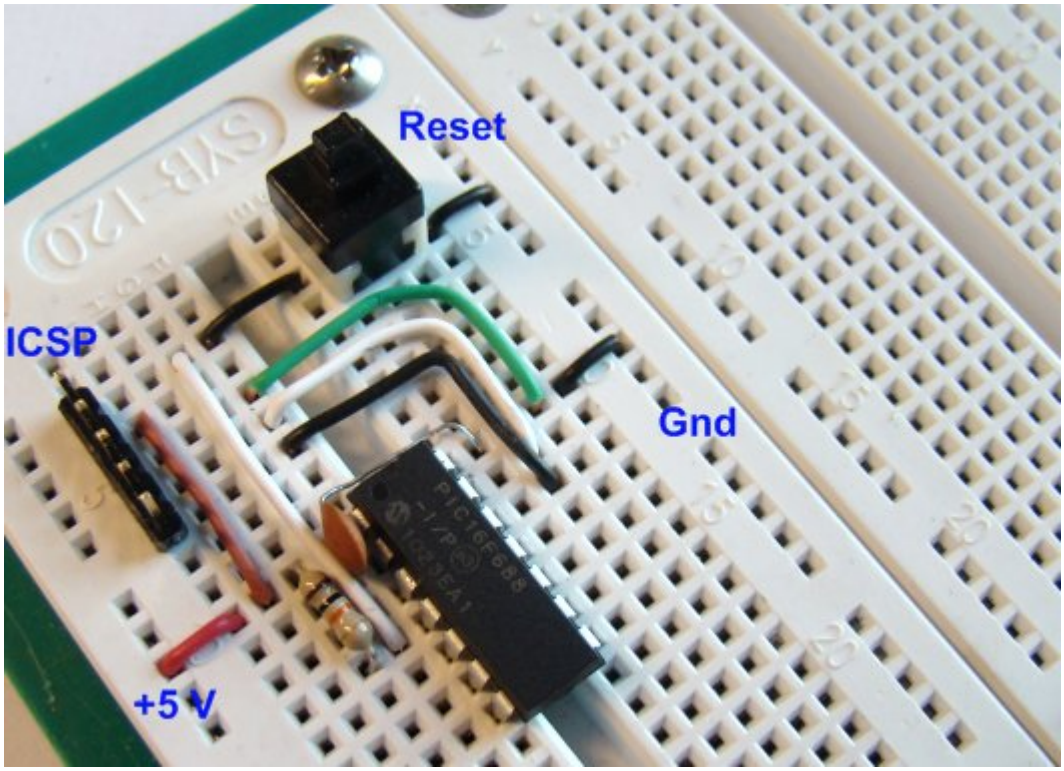


Figure 3: 브레드보드상에 PIC16F688와 아주 간단한 회로를 구축한 모습

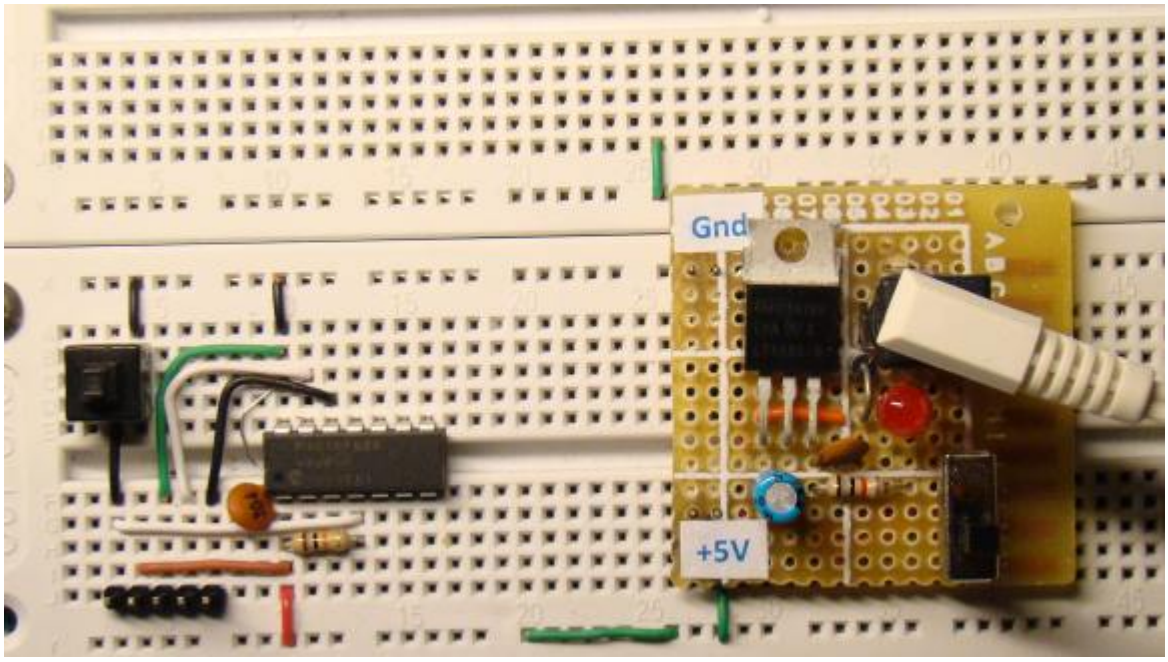


Figure 4: 5V 전원이 브레드보드에 연결된 기초 PIC16F688 회로

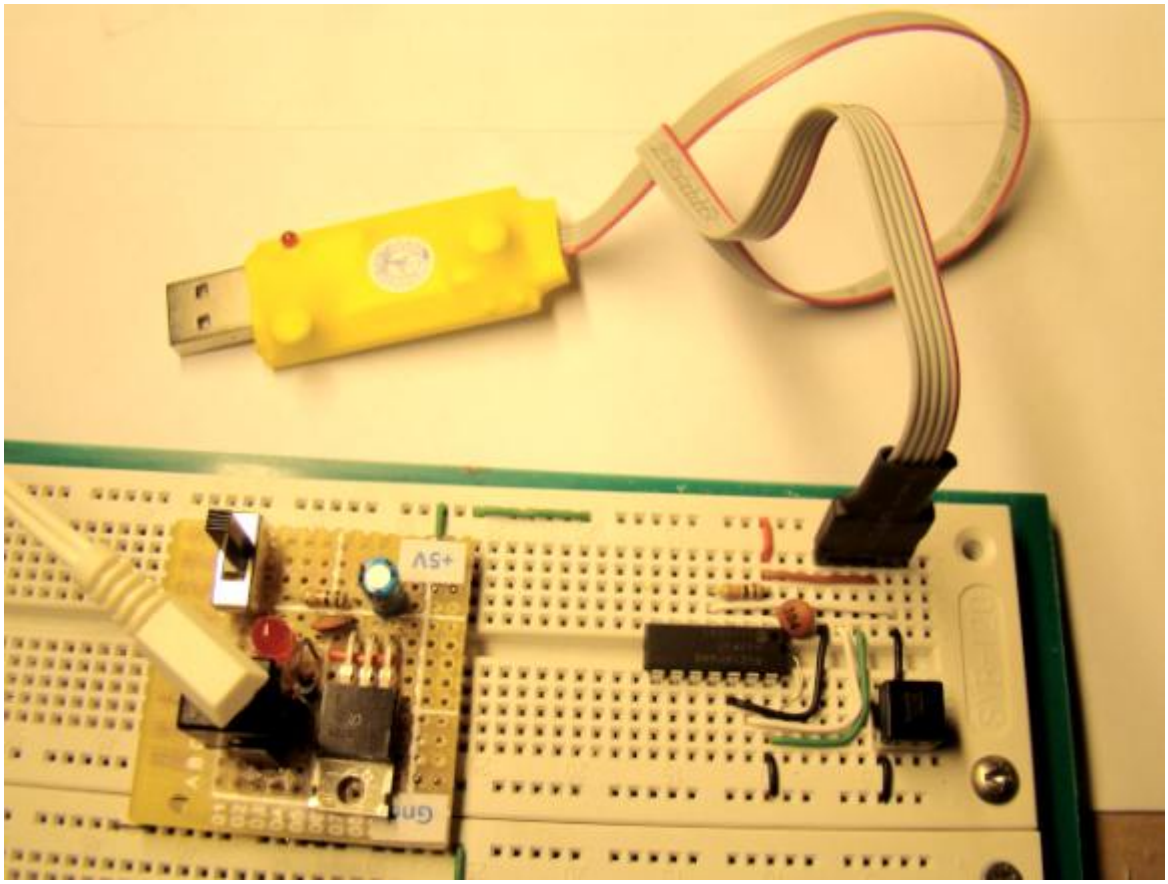


Figure 5 : 기초적인 PIC16F688 셋업회로에 5V 전원과 프로그래머가 브레드보드에 연결된 모습.

자 이제는 MCU를 프로그래밍할 수 있게 된것 같군요.

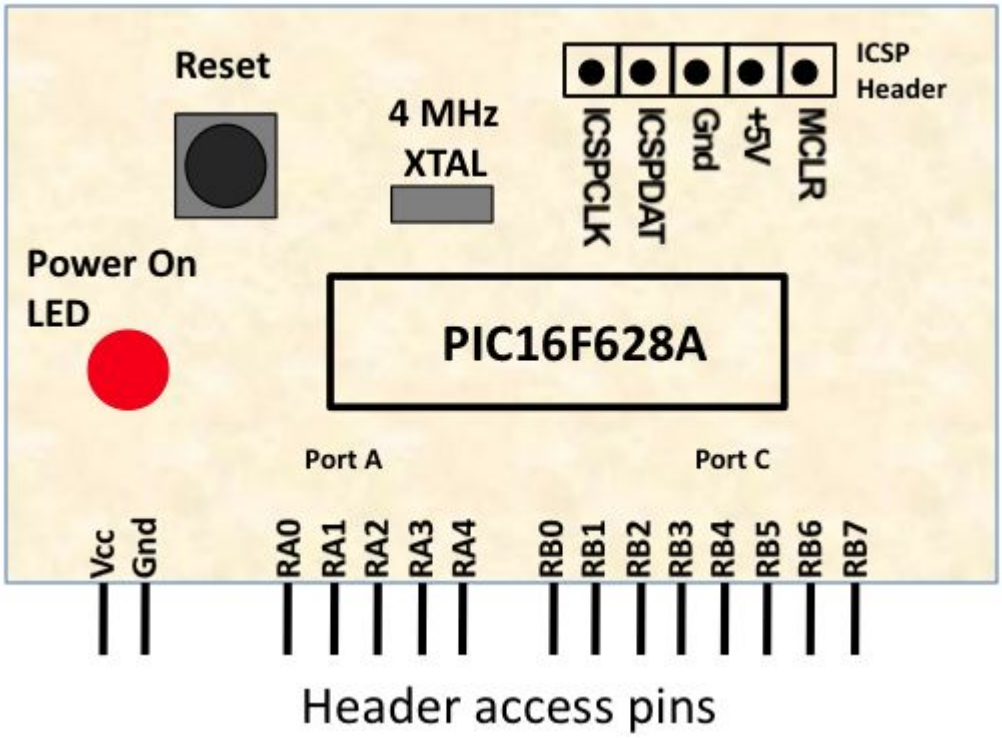
[PIC 마이컴 실험하기] 빠른 프로토타이핑을 위한 PIC16F628A 모듈

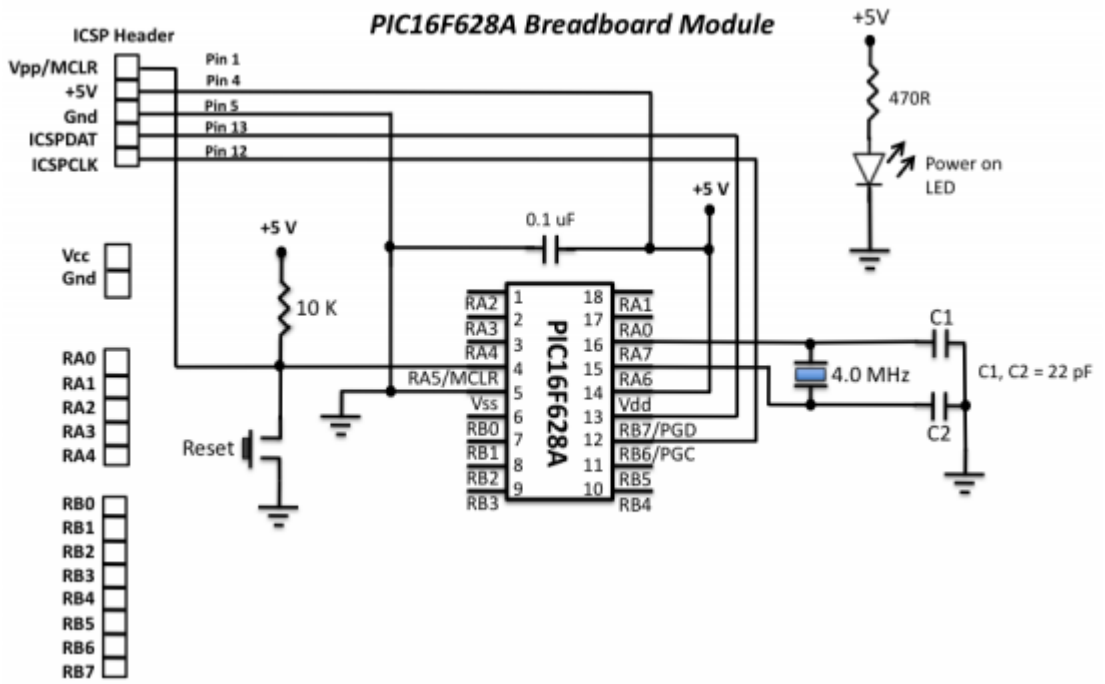
마이컴 실험실

2011/09/30 11:58 http://blog.naver.com/ubicomputing/150120173710

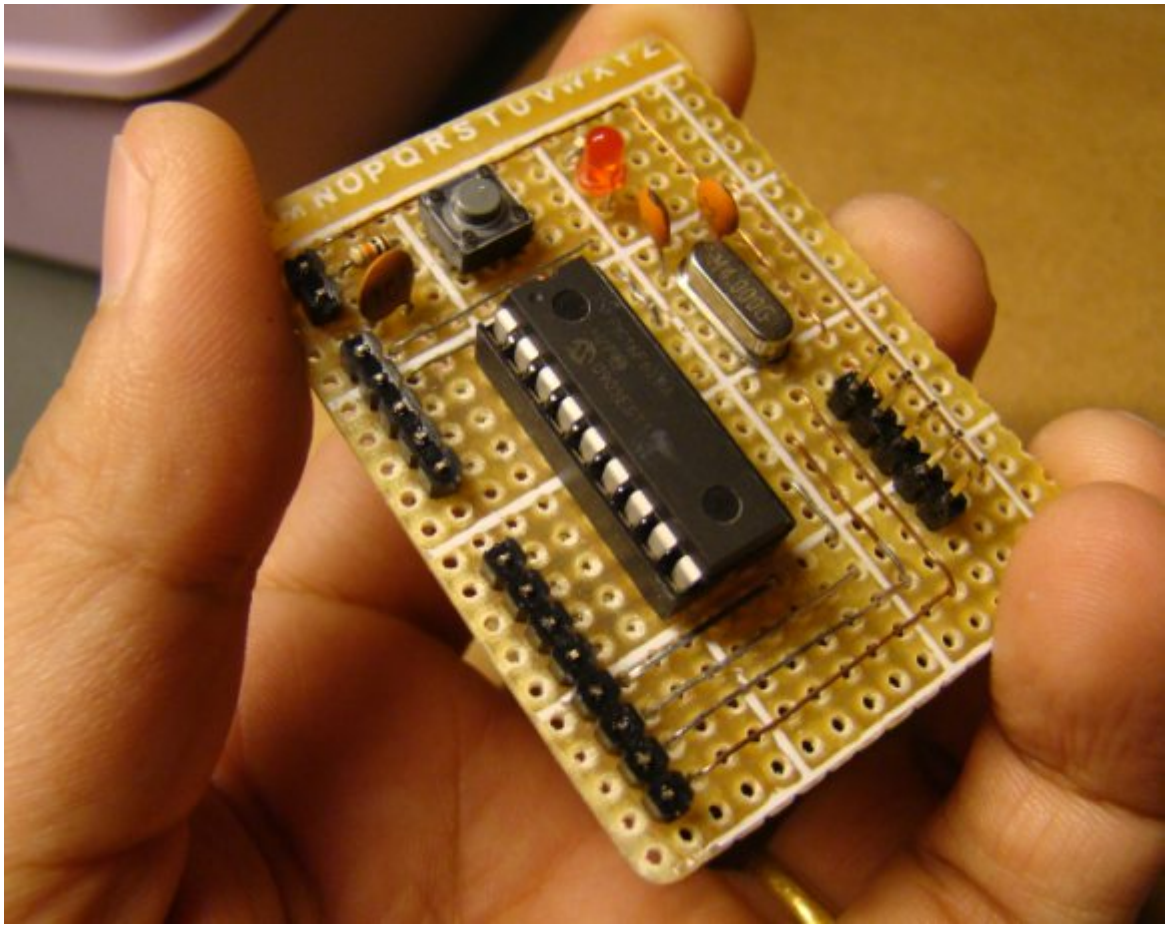
이번 게시물에서는 PIC16F628A MCU용 브레드보드 모듈을 구성해보겠습니다. PIC16F628A의 전원공급 핀과 I/O포트 핀은 숫컷 핀헤더를 통해 접근할수있게하여 다른 브레드보드에 쉽에 연결할 수있게 합니다. 지난 게시물의 PIC16F688 브레드보드 모듈과 다른 점은 MCU가 외장 4.0MHz 크리스탈과 연결된다는 점입니다. 그래서 이 모듈은 좀더 정확한 시간 계산이 필요한 어플리케이션에 적합합니다. 게다가 PIC16F628A는 8비트 데이터를 직접 PORTB에 읽고 쓸 수 있습니다. (PORTB가 8비트이기때문. PIC16F688의 다른 포트는 8비트가 아닙니다.)

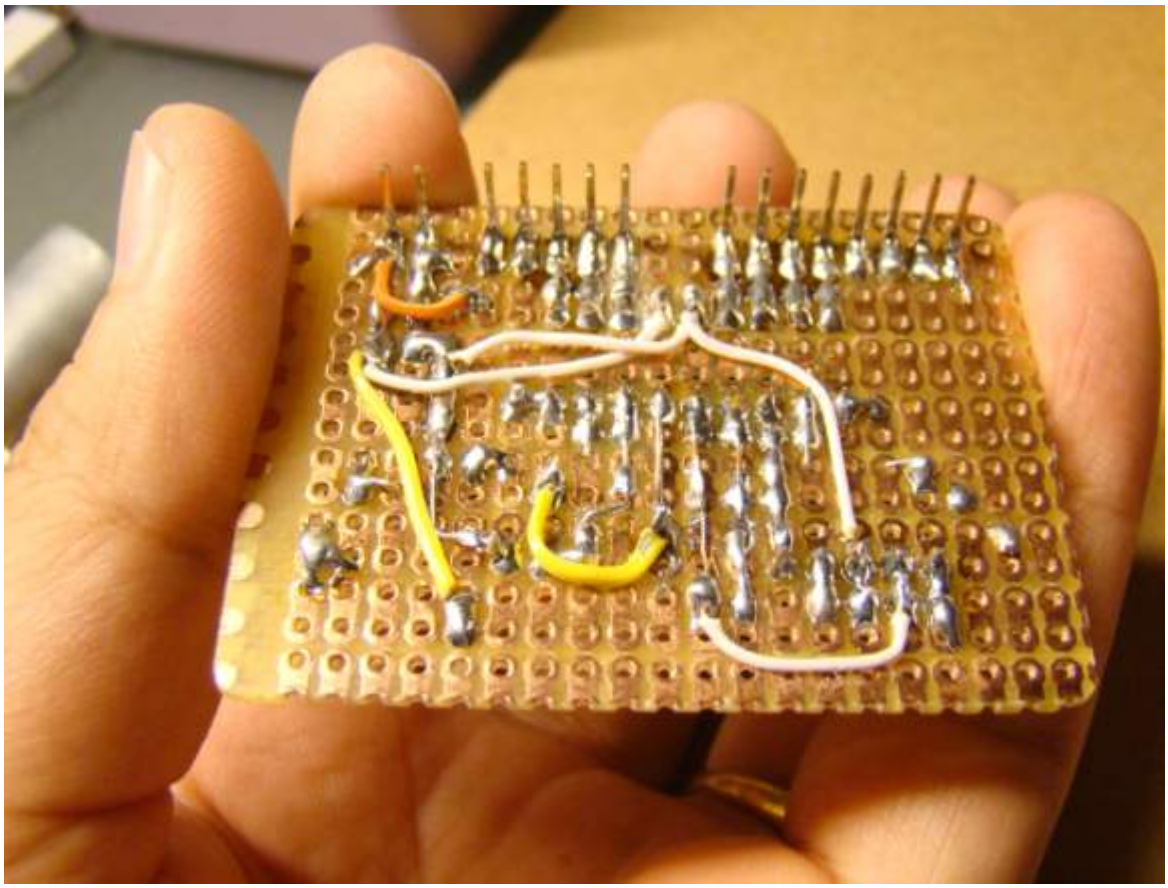
모듈의 레이아웃과 회로도는 다음과 같습니다. 모듈은 In-Circuit Programming을 위한 ICSP 헤더핀과 리셋스위치, power-on led를 가지고 있고, PORTB의 모든 핀과, PORTA의 RA0~RA4핀에 대한 연결 핀을 가지고 있습니다. RA6, RA7핀은 외부 크리스탈을 연결하는데 사용이 되고 반면에 RA5은 입력전용핀이며 리셋회로를 위해 사용이 됩니다.



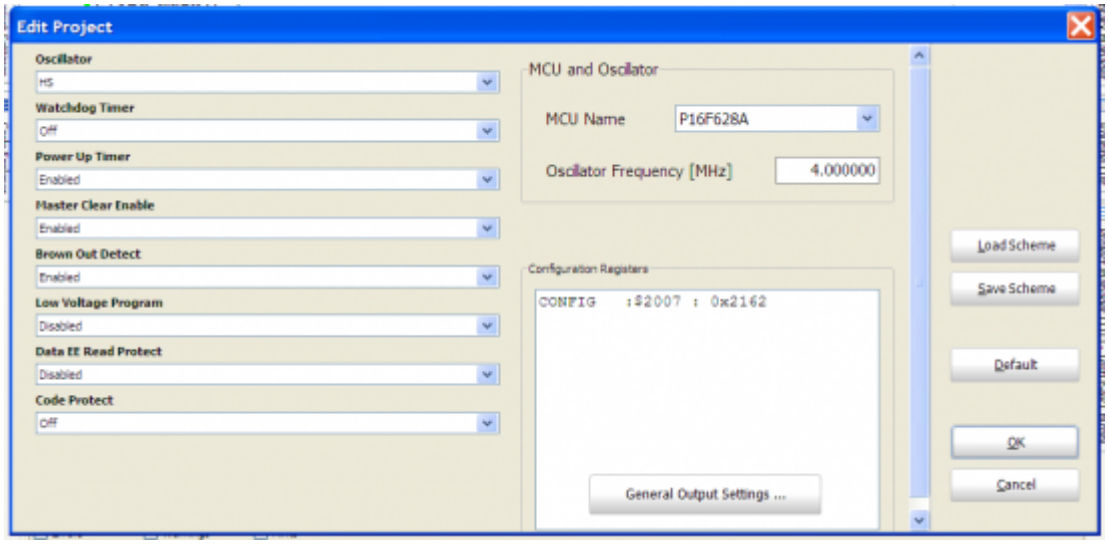


위의 회로를 브레드보드에 납땜하였습니다.





자, 이제는 브레드보드모듈을 테스트할 때입니다. 테스트를 위해 PIC16F628A의 PORTB핀과 연결된 8개의 LED를 추적하는 프로그램을 작성하였습니다. LED의 양극은 PORTB 핀에 연결되고 음극은 330오옴 저항을 통해 그라운드됩니다. 프로그램은 C언어로 작성되어서 PIC용 mikroC 컴파일러로 컴파일 되었습니다. configuration bit 셋팅에서는, MCLR이 반드시 enable되어 있어야하고 clock소스는 XT나 HS로 되어 있어야합니다. 둘다 4.0Mhz에서 동작합니다.

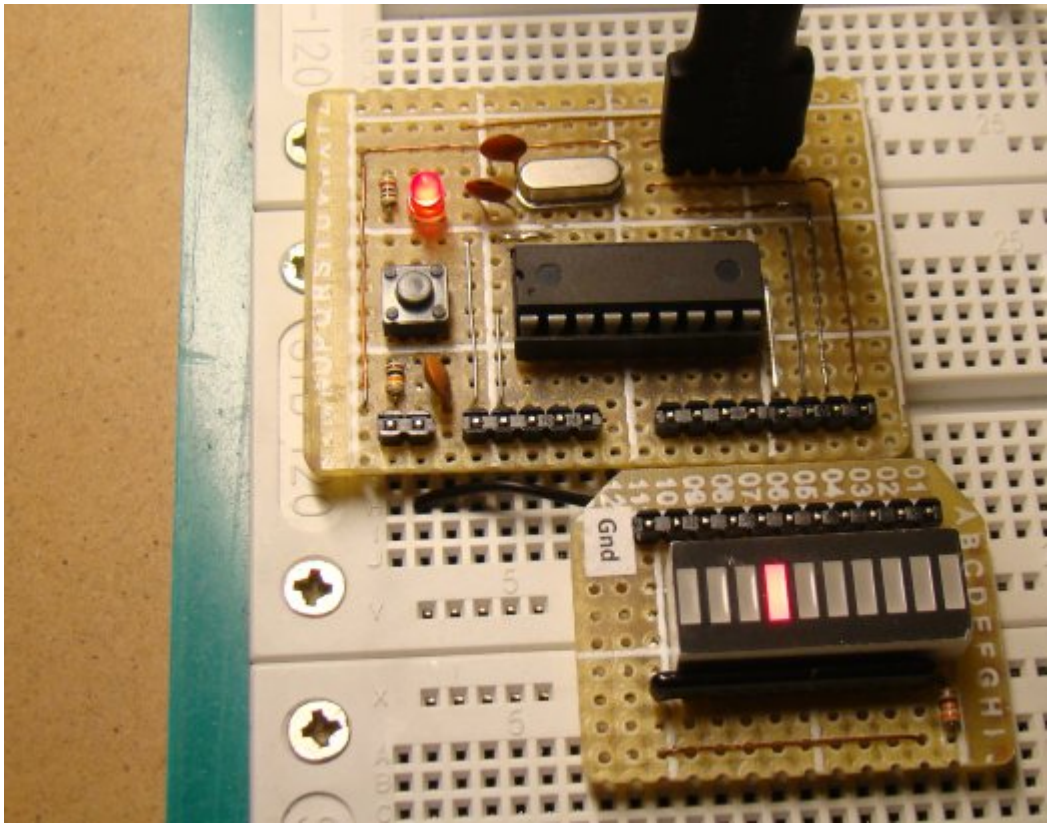


mikroC로 작성한 테스트 프로그램입니다.

```
/*

Project: LED chaser program for testing the PIC16F628A module

Eight LEDs are connected to PORTB pins

Copyright @ Rajendra Bhatt

Dec 2, 2010

MCU: PIC16F628A

Oscillator: XT, 4.0000 MHz

MCLR Enabled

*/

unsigned short i, j;

void main() {

 CMCON = 0x07; // Disable comparators

 PORTB = 0x00; // Start with all zero O/Ps

 TRISB = 0x00; // PORTB pins all O/Ps

 do {

  i = 1;

  for(j=0; j<8; j++) {

   PORTB = i;

   Delay_ms(100);

   i = i<< 1; // Left shift one bit

  }

 }while(1);
```

}

프로그램이 로드된 후에, 전원으 넣고 LED를 살펴보세요.



가치창조기술 | www.vctec.co.kr

[PIC 마이컴 실험하기] 1. LED깜빡이게 하기

마이컴 실험실

2011/09/30 14:49 http://blog.naver.com/ubicomputing/150120187132

이번 게시물에서는 LED를 ON/OFF시키는 실험을 할것입니다. 간단한 프로젝트이지만 코드가 작성되고, 컴파일되어 PIC 마이컴에 탑재되고, 브레드보드상의 회로가 에러없이 셋업되었는지를 확인하여 볼 수 있는 좋은 실험입니다. PIC16F688의 포트핀중 하나에 LED를 연결하여 1초동안 반복적으로 깜빡이게 하여보겠습니다.

선행 이론

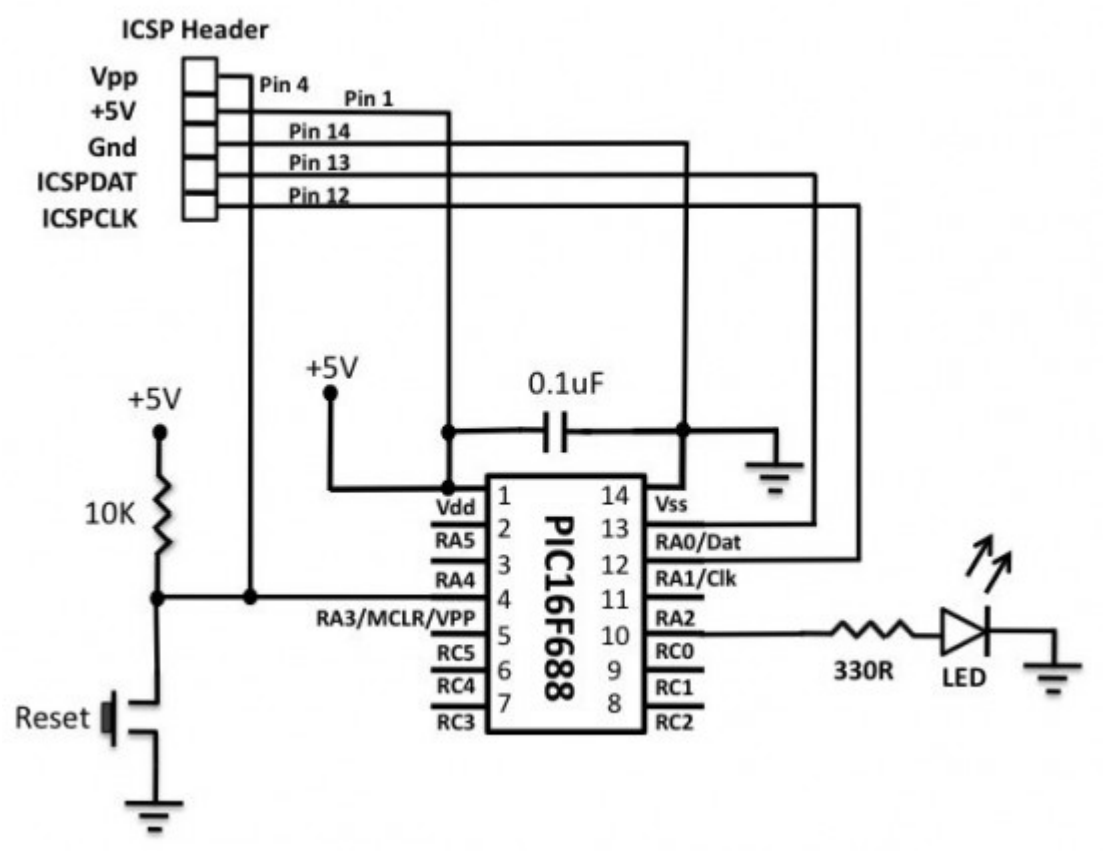
다음의 주제에 대해 먼저 선행 학습이 필요합니다.

- PIC16F688의 디지털 I/O포트(PORTA, PORTC)
- 방향 제어 레지스터 TRISA, TRISC
- SFR 레지스터 CMCON0, ANSEL

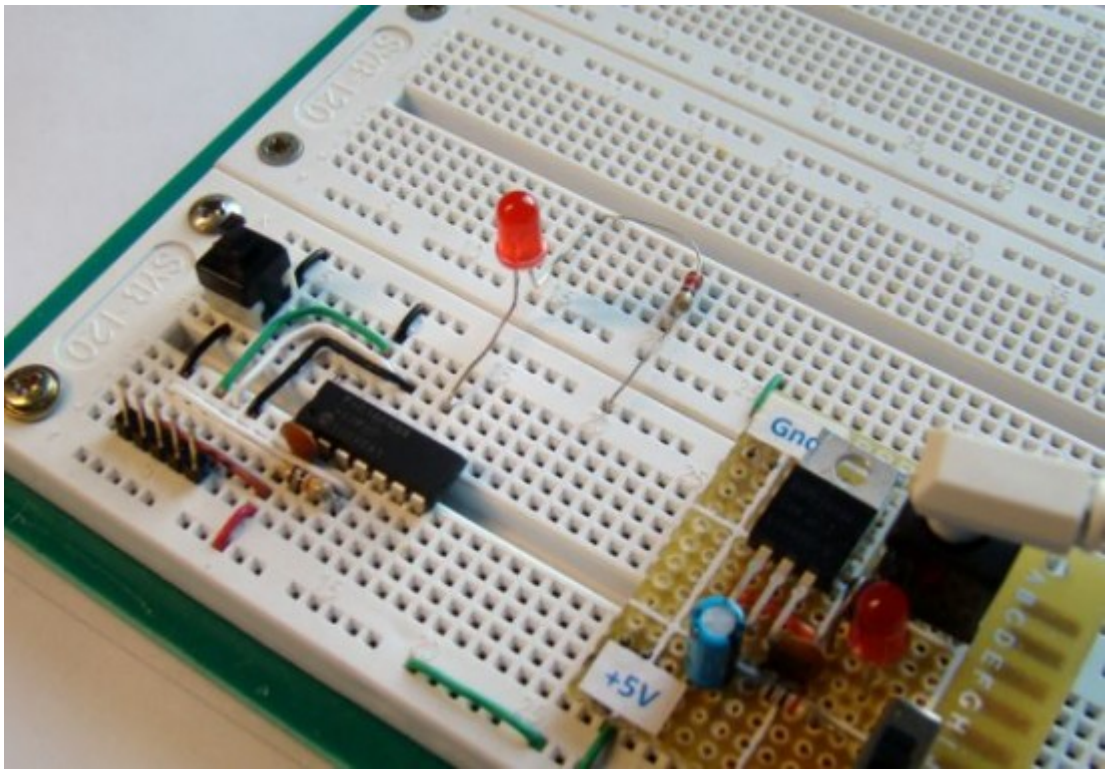
위의 주제에 대해 익숙치 않으시면 다음의 게시물을 참고하세요. PIC16F688의 디지털 I/O 포트

회로도

이전 게시물에서 브레드보드에 셋업하였던 기초회로(브레드보드에 기초적인 PIC16F688회로 셋업하기)에 LED를 RC0와 전류를 제한하기 위한 저항(470Ohm)에 연결합니다. 회로도는 아래와 같습니다.



LED 깜빡이기용 회로

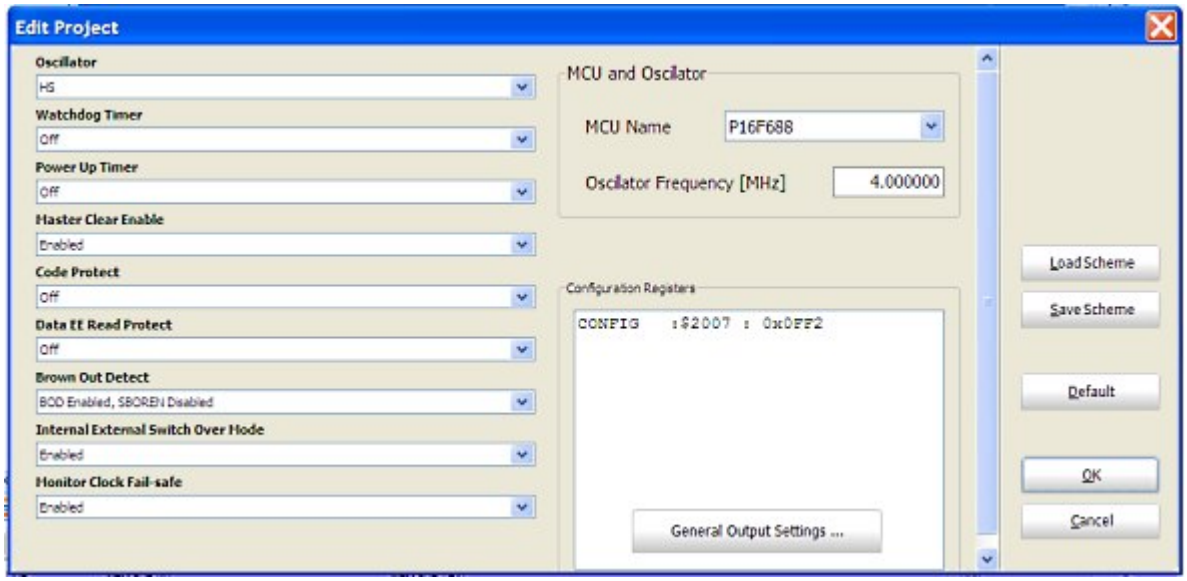


브레드보드에 프로토타이핑

소프트웨어

mikroC에서 새로운 프로젝트를 열고, 디바이스 이름으로 PIC16F688을 선택합니다. 다음으로 4.0MHz를 디바이스 클럭으로 할당하고 프로젝트 이름과 폴더를 지정합니다. 여기서는 Lab1으로 이름을 지정하고 FlashLED라는 폴더에 저장하였습니다. mikroC 프로젝트 파일은 .mccpi 확장자를 자고 있습니다. 다음은 "Add File to Project"인데 여기서는 추가할 파일이 없으므로 다음으로 넘어갑니다. 다음 단계는 라이브러리 선택입니다. "Include All" 옵션을 선택하고 Finish버튼을 눌러 종료합니다.

그러면, void main()함수가 포함된 프로그램 윈도우를 보게 되실 껍니다. 이제 Project -> Edit Project 메뉴로 가시면, 아래와 같은 창을 보게 되실껍니다.



이 창에서 PIC16F688 마이크로컨트롤러 안에 있는 14비트의 CONFIG 레지스터의 configruation bit를 프로그램 할 수 있습니다. configuration bit는 각 개발자가 어플리케이션의 필요에 맞게 커스터마이제이션을 할 수 있게 하여 줍니다. 디바이스에 전원이 들어오면, 이 설정비트의 상태들이 디바이스의 모드를 결정하므로 본 실험에 맞게 configuration bit를 프로그램합니다.

아래를 선택,

Oscillator -> Internal RC No Clock

Watchdog Timer -> Off

Power Up Timer -> On

Master Clear Enable -> Enabled

Code Protect -> Off

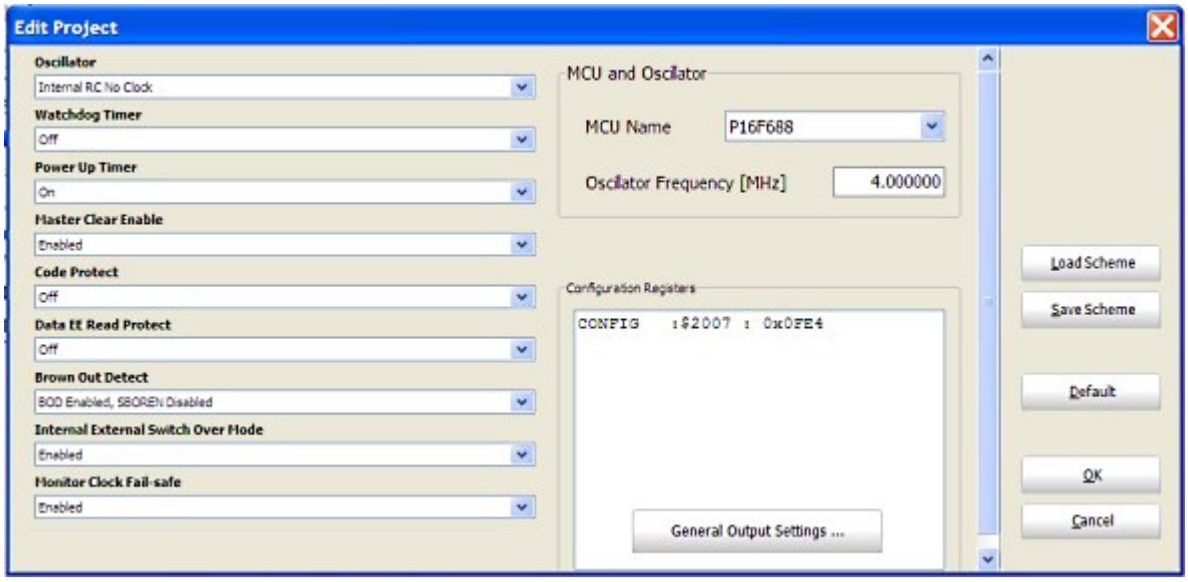
Data EE Read Protect -> Off

Brown Out Detect -> BOD Enabled, SBOREN Disabled

Internal External Switch Over Mode -> Enabled

Monitor Clock Fail-Safe -> Enabled

Power-Up Timer를 ON 시켜놓았기 때문에 72ms 후에 프로그램이 시작하기 시작합니다. 외부 전원이 안정화되기에 충분한 시간이여서 시작시에 mcu를 수동으로 리셋할 필요를 없애줍니다.



적절한 configuration bit 선택

아래는 LED깜빡이기 소스입니다. 카피하여서 프로그램윈도우에 붙여넣으시고 컴파일 및 빌드하시면 에러나가 나지 않으면 HEX파일이 프로젝트 폴더에 생성 될 것입니다. 이 HEX파일을 프로그래머를 이용하여 PIC16F688에 프로그래밍합니다.

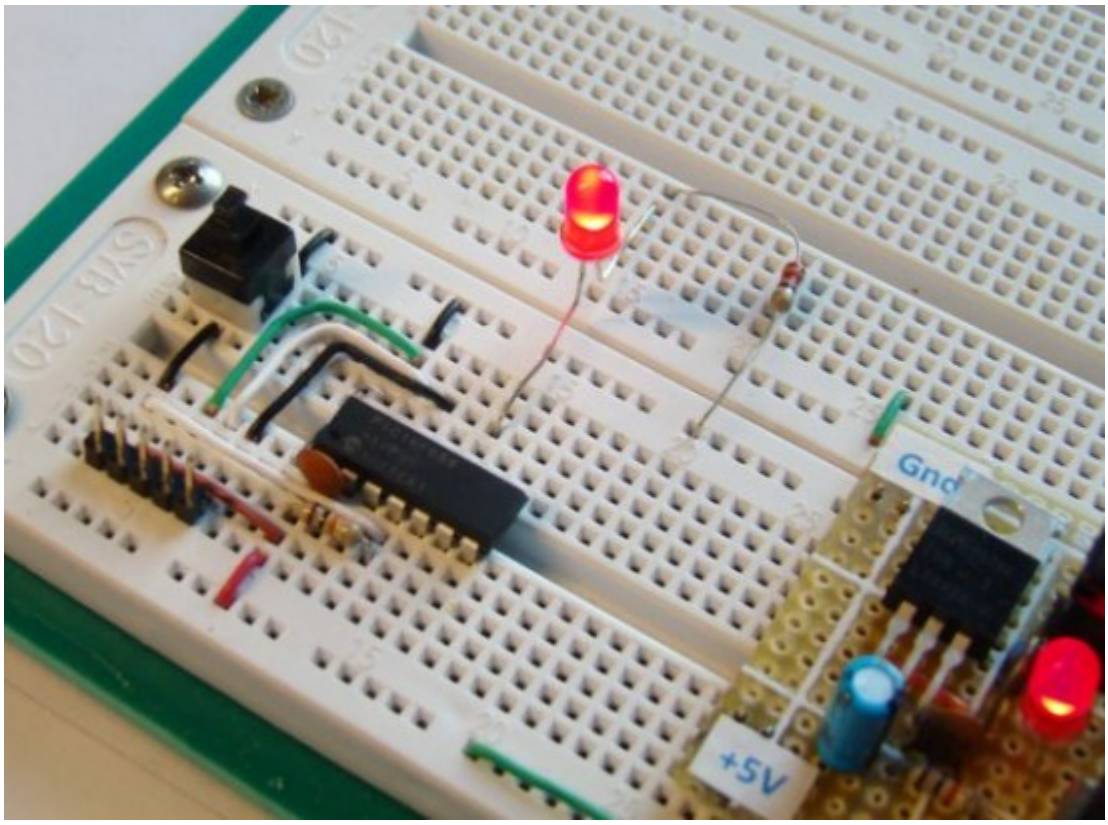
```
/*
Lab 1: Flashing LED with PIC16F688
Internal Oscillator @ 4MHz, MCLR Enabled, PWRT Enabled, WDT OFF
*/
// Define LED @ RC0
sbit LED at RC0_bit;
void main() {
ANSEL = 0b00000000; //All I/O pins are configured as digital
CMCON0 = 0×07 ; // Disbale comparators
```

```
TRISC = 0b00000000; // PORTC All Outputs
TRISA = 0b00001000; // PORTA All Outputs, Except RA3

do {
LED = 1;
Delay_ms(1000);
LED = 0;
Delay_ms(1000);
} while(1);  // Infinite Loop
}
```

내장 라이브러리 함수인 Delay_ms() 를 이용하여 LED ON/OFF시 1초간 딜레이를 생성하였습니다.

결과

LED가 1초마다 ON/OFF되는 것을 볼수 있네요.



[PIC 마이컴 실험하기] 2. 기본적인 디지털 입력과 출력

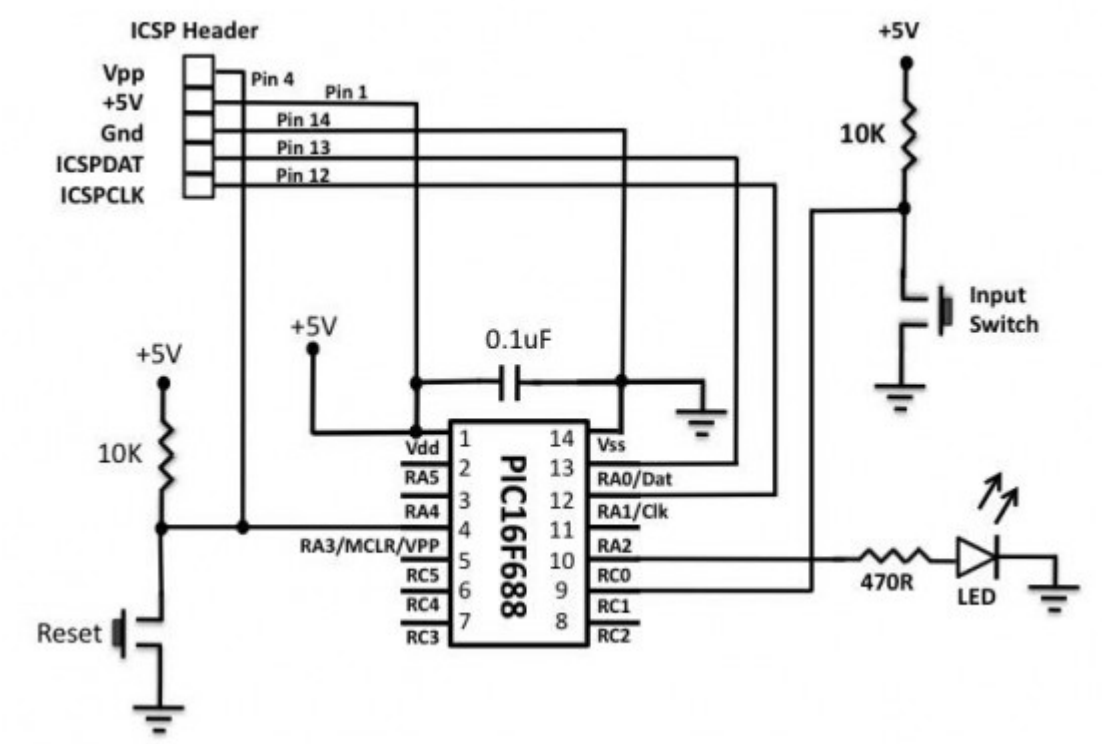
마이컴 실험실

2011/09/30 17:01 http://blog.naver.com/ubicomputing/150120200060

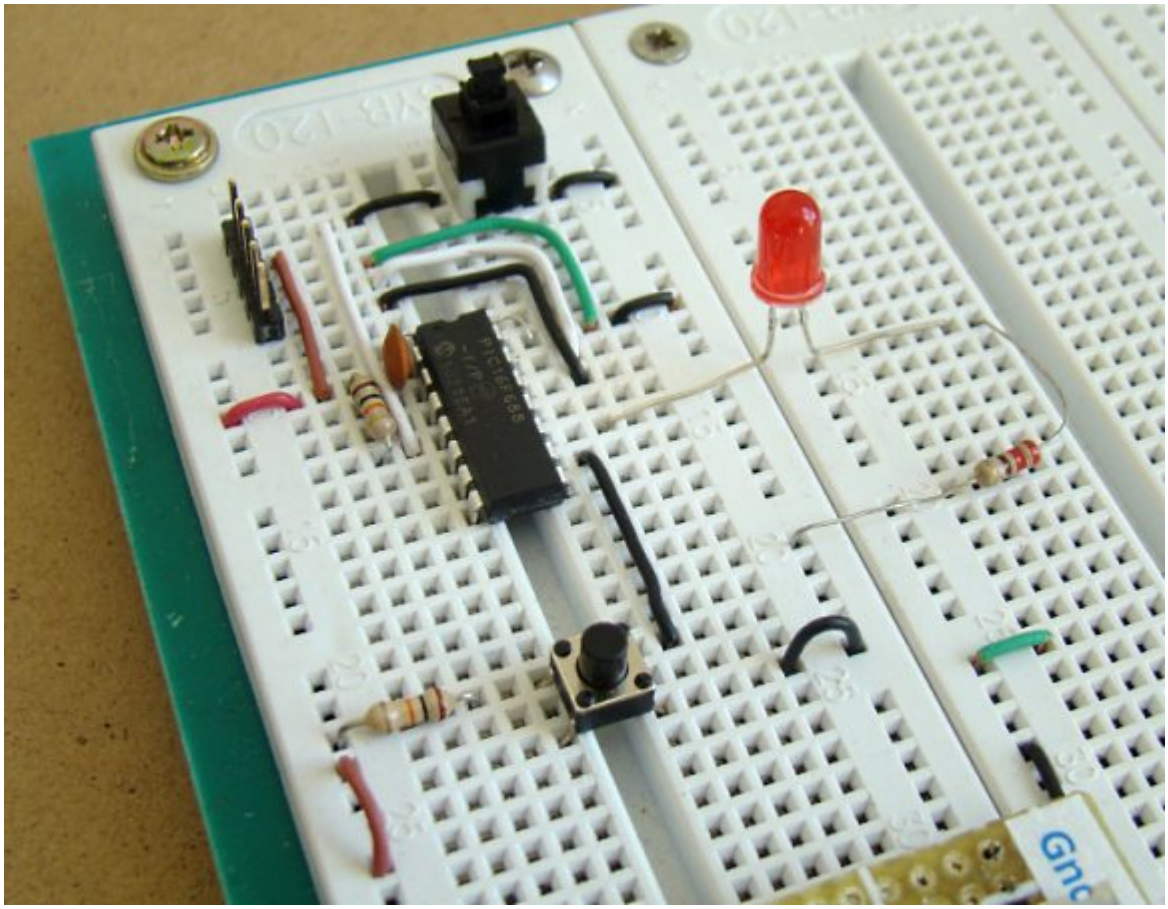
이번 게시물에서는 푸쉬버튼 스위치로부터 어떻게 디지털 입력을 읽어내는지를 알아보도록 하겠습니다. 디지털입력은 1과 0의 두개의 값만을 가집니다. 푸쉬버튼 스위치의 설정은 다른 포트에 연결된다는 것만 빼고는 리셋스위치의 설정과 동일합니다. 스위치의 상태가 RC1을 통해 읽혀지고 버튼이 눌릴때마다 RC0에 연결된 LED가 ON/OFF 될 것입니다. 참고) PIC16F688의 디지털 I/O 포트

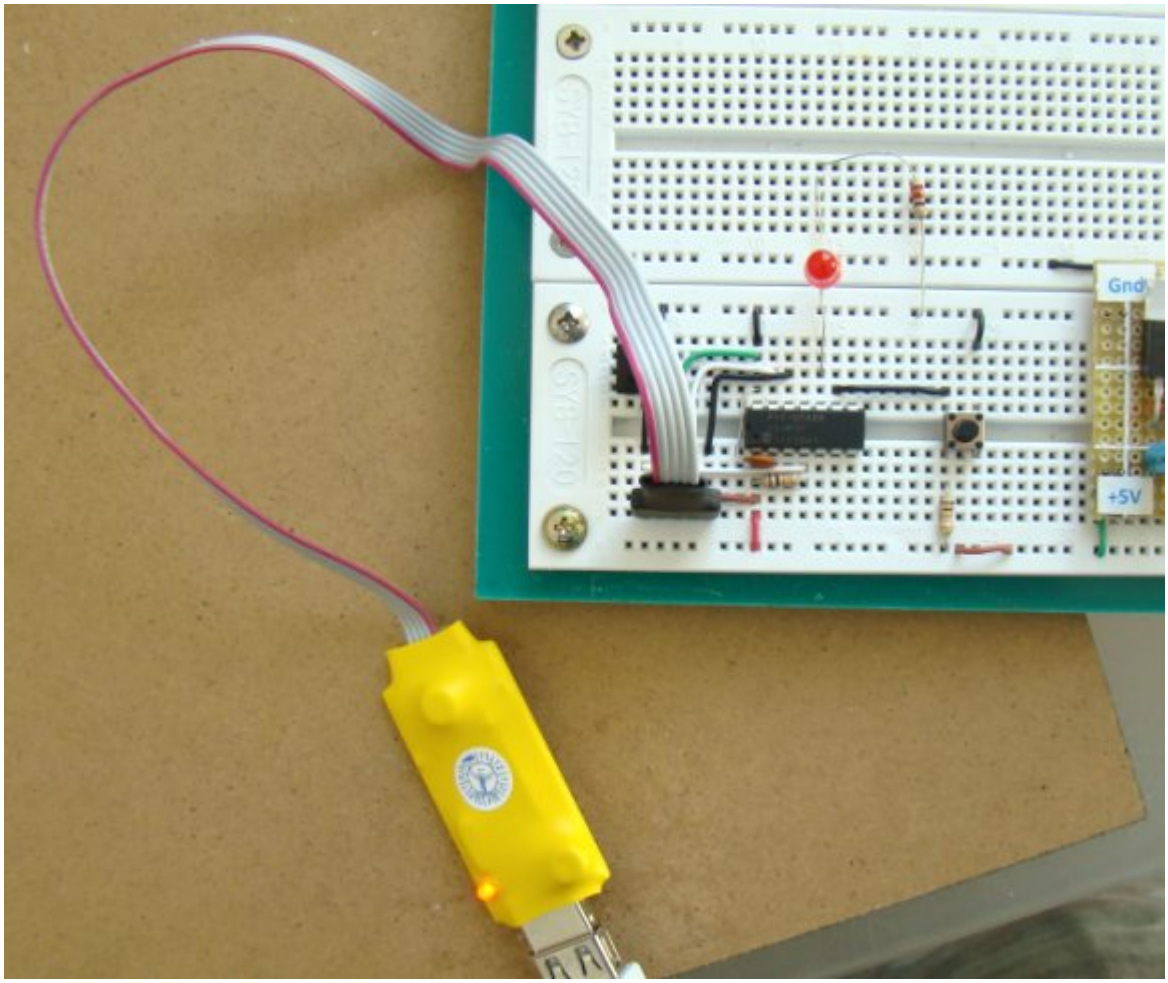
회로도

지난 게시물 LED 깜박이기 에서 구성하였던 회로에, 푸쉬버튼을 RC1에 연결하도록 하겠습니다.(아래 회로도 참조) PIC 마이컴의 입력포트는 매우 높은 입력 임피던스를 가지고 있어서, 스위치가 열린 정상적인 상태 동안에 RC1 핀은 외부의 10K 저항을 통해 High로 올려집니다. 스위치가 눌렸을때 RC1은 그라운드로 내려지고 MCU는 그것을 로직"Low"로 읽습니다. 이러한 방식으로 풀업저항을 가진 푸쉬버튼은 MCU에 디지털 입력 1 혹은 0 을 줄 수 있습니다.



디지털 입력을 읽기 위한 회로


PIC16F628
Gnd



브레드보드상에 프로토타이핑된 회로

소프트웨어 설정

RC1에 스위치가 연결되었기때문에 RC1은 반드시 입력으로 정의가 되어야 합니다. 이것은 TRISC 레지스터의 해당 비트를 설정함으로써 가능합니다. 다시한번 말씀드리지만, 컴패레이터와 ADC채널을 disable시키는 것을 잊지 마십시오.

이 실험을 위한 코드는 아래에 있습니다. 코드를 mikroC 컴파일러로 컴파일하고 HEX파일을 PIC에 심습니다. mikroC 컴파일러는 푸쉬버튼으로부터 디지털 입력을 읽기 위한 내장 라이브러리(Button)을 가지고 있습니다. 그 문법은 다음과 같습니다.

Prototype	`unsigned short Button(unsigned short *port, unsigned short pin, unsigned short time, unsigned short active_state);`
Returns	Returns 0 or 255.
Description	Function eliminates the influence of contact flickering upon pressing a button (debouncing). Parameter `port` specifies the location of the button; parameter `pin` is the pin number on designated `port` and goes from 0..7; parameter `time` is a debounce period in milliseconds; parameter `active_state` can be either 0 or 1, and it determines if the button is active upon logical zero or logical one.
Requires	Button pin must be configured as input.

Example	Example reads RB0, to which the button is connected; on transition from 1 to 0 (release of button), PORTD is inverted:
	
	`do {`
	`  if (Button(&PORTB, 0, 1, 1)) oldstate = 1;`
	`  if (oldstate && Button(&PORTB, 0, 1, 0)) {`
	`  PORTD = ~PORTD;`
	`  oldstate = 0;`
	`  }`
	`} while(1);`

Button 명령에 debounce 기능이 빌트인 되어 있네요.

스위치 디바운싱

푸쉬버튼과 같은 기계적인 스위치가 회로 연결을 끊기 위해 눌르거나 띄거나 할때, 안정적인 상태로 진입하기 전까지 스위치는 여러 개의 open-close 전이를 만들어 만들어 냅니다. 이런 현상을 스위치 바운스라고 부르며 수십 밀리세컨드동안 지속되기도 합니다. 우리에게는 매우 짧은 순간이지만 mcu는 이러한 전이를 감지 할수 있어 버튼이 여러번 눌리거나 띄어졌거나 한 동작으로 인식할 수 있습니다. 그렇기 때문에 mcu가 스위치의 상태를 읽기 전에 스위치는 반드시 디바운싱되어 있어야 합니다.

디바운싱 회로는 추가적인 회로를 필요로 하기때문에 임베디드시스템의 비용을 올리는 역활을 합니다. 가장 쉬운 솔루션은 디바운싱을 소프트웨어로 구현하는 것입니다. 가장 간단한 로직은 키가 눌리거나 띄어졌을시 5-20ms 동안 기다렸다가 스위치의 상태를 읽는 방법이 있습니다.

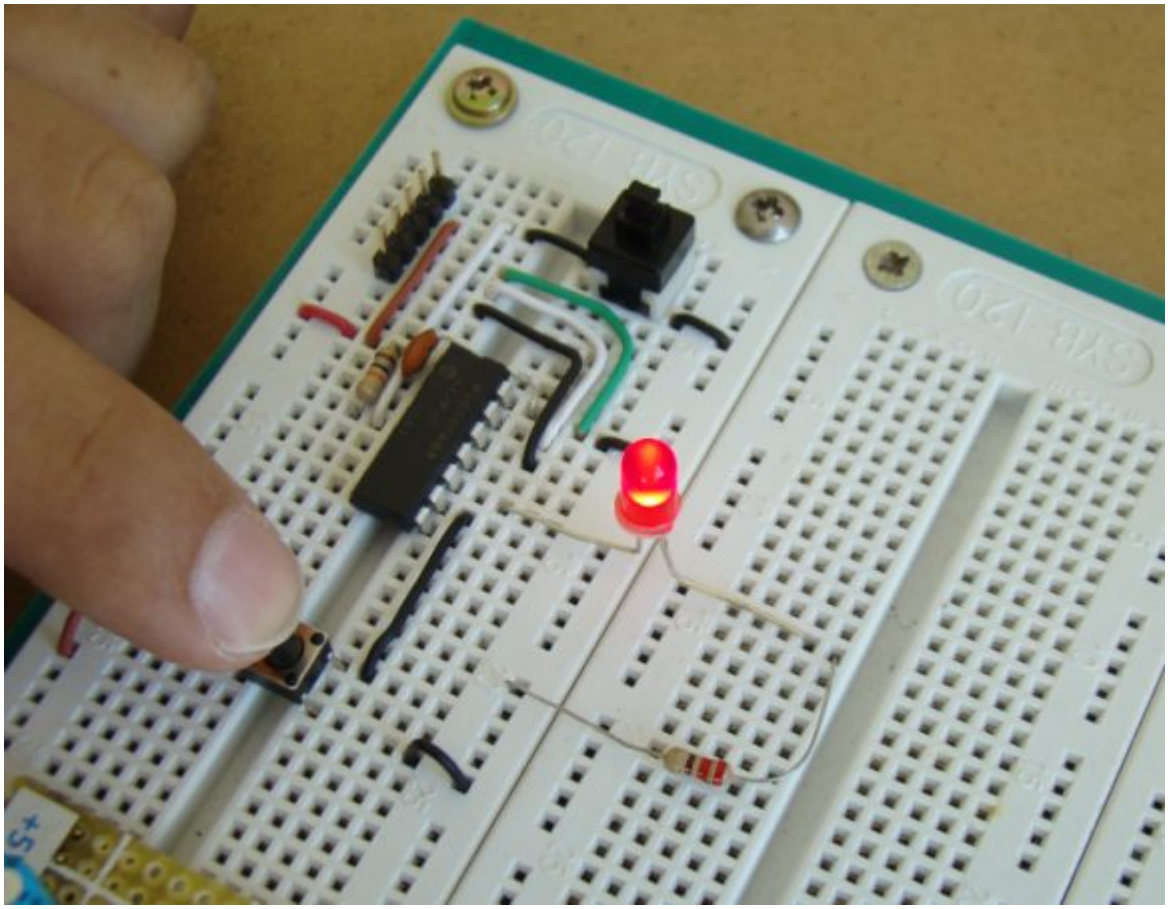
configuration bit는 1번 게시물에서 보았던 것과 동일합니다.

```
/*
 Lab 2: Toggle LED with a Tact Switch Input
 Internal Oscillator @ 4MHz, MCLR Enabled, PWRT Enabled, WDT OFF
*/
```

```
// Define LED @ RC0 and Tact switch @ RC1
sbit LED at RC0_bit;
sbit Switch at RC1_bit;
#define Switch_Pin 1
#define Switch_Port PORTC
#define Debounce_Time 20
void main() {
ANSEL = 0b00000000; //All I/O pins are configured as digital
CMCON0 = 0x07 ; // Disbale comparators
TRISC = 0b00000010; // PORTC All Outputs
TRISA = 0b00001000; // PORTA All Outputs, Except RA3
LED = 0;
do {
 if (Button(&Switch_Port, Switch_Pin, Debounce_Time, 0)) {
   if (!Switch) {
   LED = ~LED;
   }
   while (!Switch); // Wait for release of the button
 }
} while(1);  // Infinite Loop
}
```

결과

스위치가 눌렸을때마다, LED가 ON/OFF 됩니다. 스위치가 눌리면 MCU는 버튼이 띄어질때까지 기다립니다.



가치창조기술 | www.vctec.co.kr

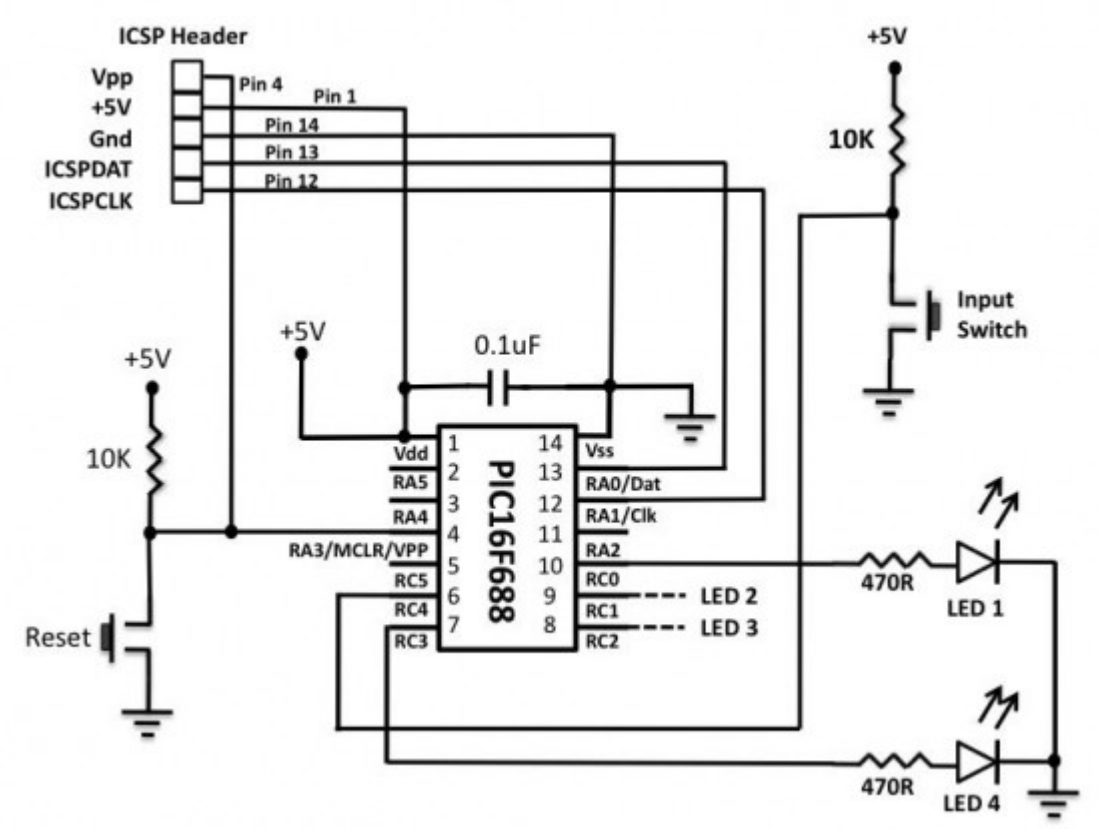
[PIC 마이컴 실험하기] 3. 4-bit 바이너리 카운터 만들기

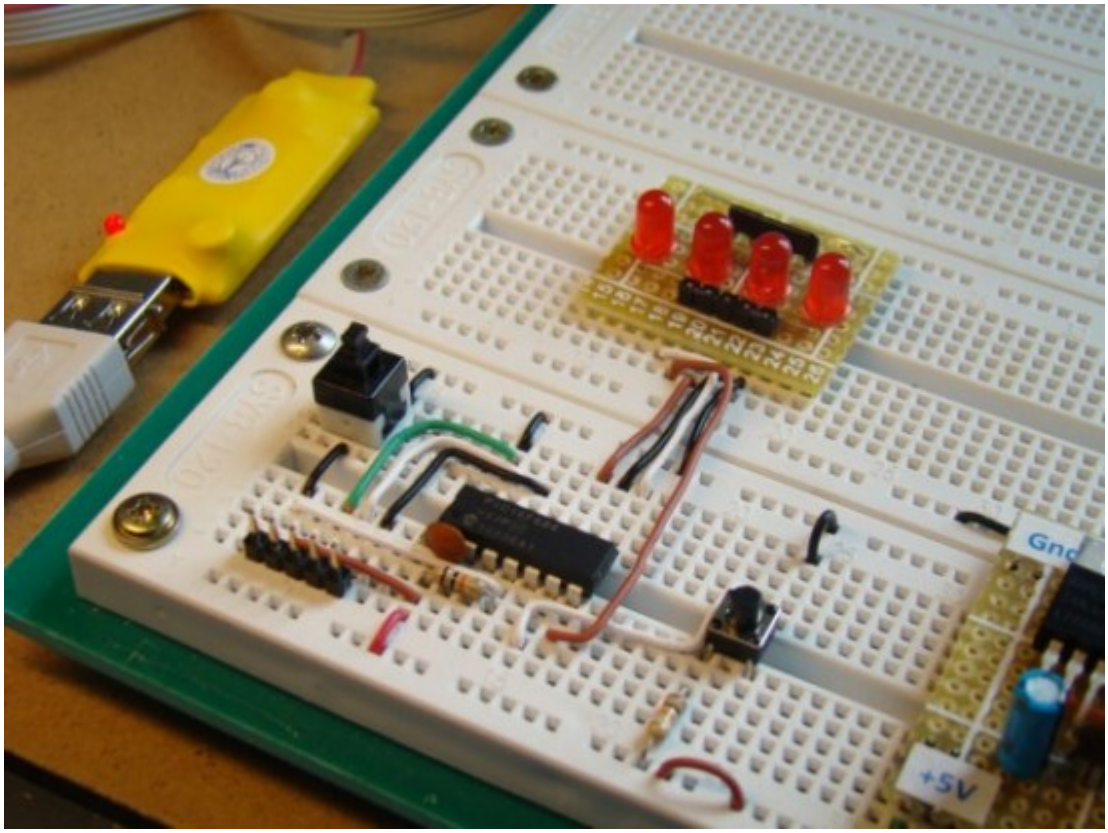
마이컴 실험실

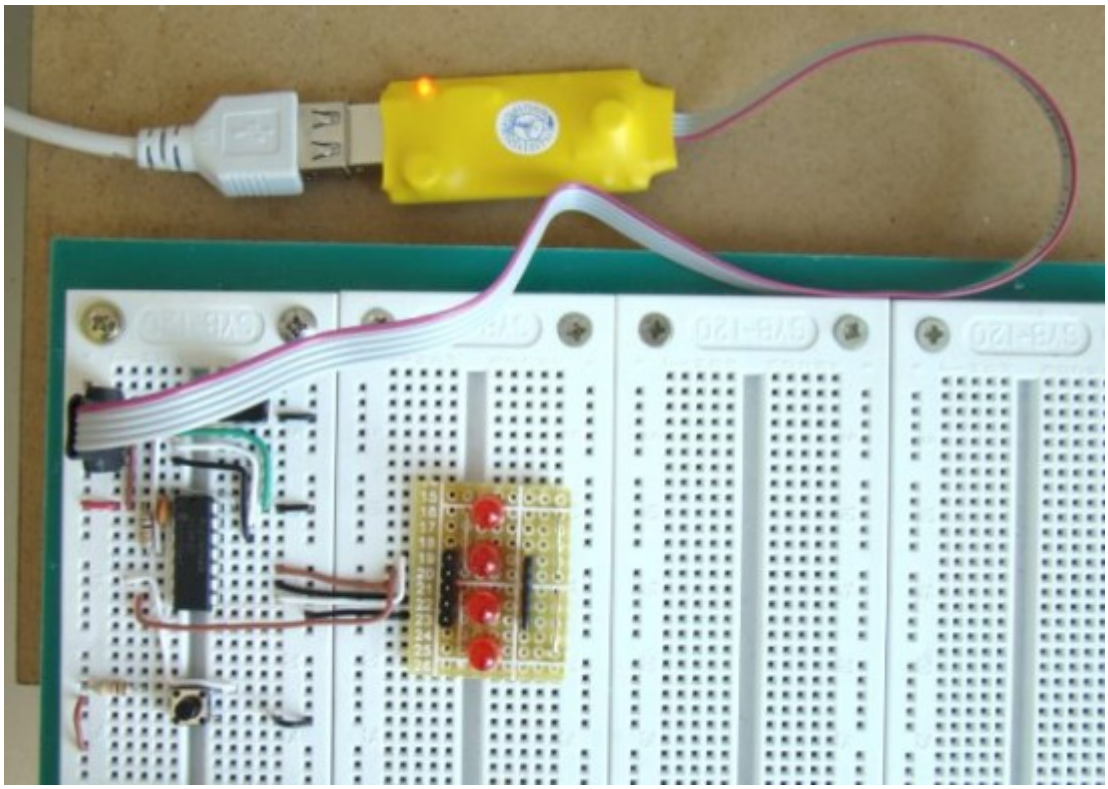
2011/10/04 13:44 http://blog.naver.com/ubicomputing/150120502652

이번 게시물에서는 바이너리를 카운트하는 LED에 대해 다루어보겠습니다. 바이너리 1과 0은 LED를 ON/OFF 시킴으로 표시합니다. 4개의 LED를 이용하여 0에서 15(0000-1111)까지 세는 4비트 바이너리 카운터를 만들어보겠습니다. 4개의 LED는 RC0~RC3 포트핀에 전류제한저항(각각 470Ω)과 함께 연결됩니다. 푸쉬버튼은 RC4에 연결되어 카운터에 입력을 제공합니다. 카운터는 0부터 세기 시작하며 버튼이 눌릴때마다 1씩 증가합니다. 카운터가 15에 도달하면, 즉 모든 LED가 켜지면, 다음 버튼이 눌릴때 0으로 리셋됩니다.

아래는 회로도와 브레드보드에 회로를 구현한 모습니다. PIC16F688마이크로컨트롤러는 내부클럭으로 4.0MHz를 사용합니다.







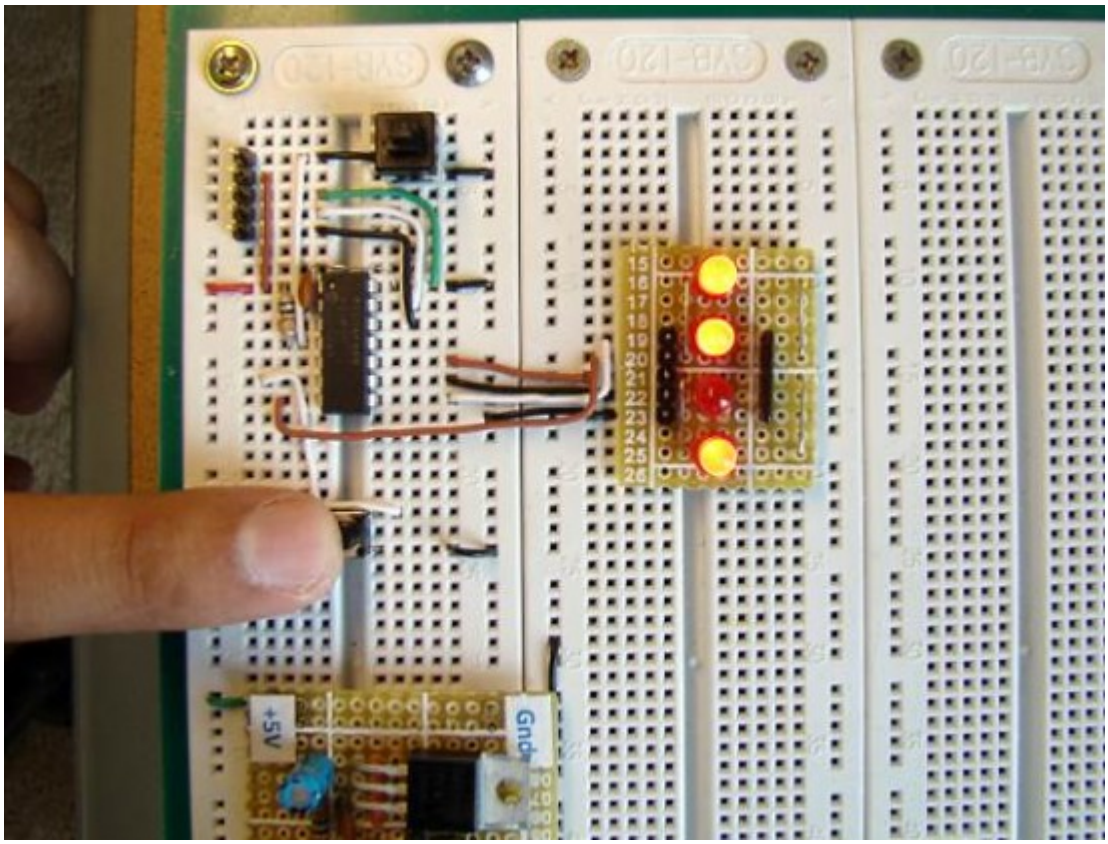
소프트웨어쪽을 살펴보록 하겠습니다. 아래는 mikroC로 작성한 소스입니다. PORTC핀 RC0-RC3을 출력으로 설정하고 RC4핀을 입력으로 설정합니다. 컴페레이터를 비활성화시킵니다.(CMCON0=7) 모든 I/O핀을 디지털로 설정합니다.(ANSEL=0). Button() 함수를 이용하여 푸쉬버튼으로부터 입력을 읽습니다.

```
/*
 Lab 3: 4-bit up counter
 Internal Clock @ 4MHz, MCLR Enabled, PWRT Enabled, WDT OFF
*/
// Define Tact switch @ RC4
 sbit Switch at RC4_bit;
// Define button Switch parameters
 #define Switch_Pin 4
 #define Switch_Port PORTC
 #define Debounce_Time 20  // Switch Debounce time 20ms
 unsigned short count;
 void main() {
 ANSEL = 0b00000000; //All I/O pins are configured as digital
 CMCON0 = 0x07 ; // Disbale comparators
 TRISC = 0b00010000; // PORTC all output except RC4
 TRISA = 0b00001000; // PORTA All Outputs, Except RA3
 count = 0;
 PORTC = count;
 do {
 if (Button(&Switch_Port, Switch_Pin, Debounce_Time, 0)) {
 if (!Switch) {
 count ++;
```

```
 if (count ==16) count =0;

 PORTC = count;

 }

 while (!Switch); // Wait till the button is  released

 }

 } while(1);  // Infinite Loop

}
```

결과

카운터는 모든 LED가 꺼져있는 0에서 부터 시작하여 매 버튼이 눌릴시마다 1씩 증가합니다. 카운터가 15까지 도착하면 리셋되어 0에서부터 다시 시작합니다.



가치창조기술 | www.vctec.co.kr

[PIC 마이컴 실험하기] 4. Character LCD 연결하기

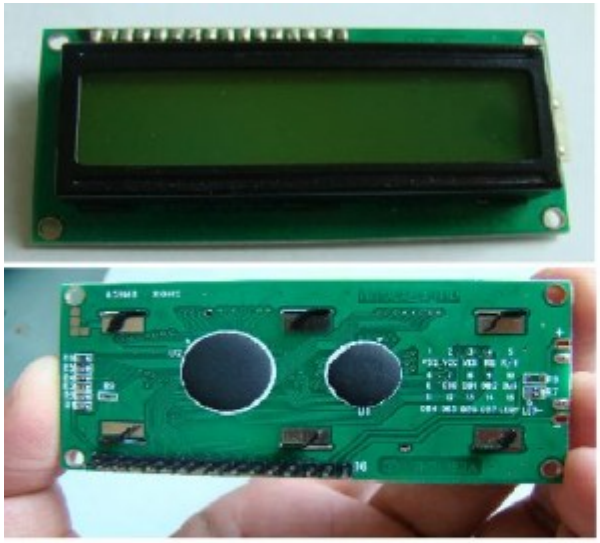
마이컴 실험실

2011/10/04 19:31 http://blog.naver.com/ubicomputing/150120536301

HD44780기반 LCD는 싸고 문자를 디스플레이할수 있기때문에 인기가 좋습니다. 게다가 MCU에 쉽게 연결할수 있고 하이레벨 컴파일러는 이 LCD를 위한 내장 라이브러리를 보유하고 있습니다. 이번 게시물에서는 HD44780기반 Character LCD를 PIC16F688 MCU에 연결하여 보도록 하겠습니다. 연결에는 PIC16F688의 6개 I/O라인이 필요한데 이중 4개는 데이타 라인이며 2개는 컨트롤 라인입니다. "Welcome to Embedded lab.com"을 LCD에 출력하여 보도록 하겠습니다.

선행이론

모든 HD44780기반 캐릭터 LCD는 14개의 핀을 통하여 연결됩니다. 8개의 데이타 핀(D0-D7), 3개의 컨트롤 핀(RS, E, R/W), 그리고 새게의 전원라인(Vdd, Vss, Vee)입니다. 몇몇 LCD는 백라이트 기능을 가지고 있어서 빛이 없는 곳에서 데이터를 읽을수 있게 하여 주며 LED+, LED-의 두개의 추가적인 연결핀을 가지고 있습니다. 아래는 16핀 LCD모듈과 핀다이어그램입니다.



16-pin LCD, Pin 15 Led+ and Pin 16 is LED-

Pin No.	Name	Function
1	V_{ss}	Ground
2	V_{dd}	+ve supply
3	V_{ee}	Contrast
4	RS	Register Select
5	R/W	Read/Write
6	E	Enable
7	D0	Data bit 0
8	D1	Data bit 1
9	D2	Data bit 2
10	D3	Data bit 3
11	D4	Data bit 4
12	D5	Data bit 5
13	D6	Data bit 6
14	D7	Data bit 7

Source: Everyday Practical Electronics, 1997

컨트롤 핀

컨트롤 핀 RS는 LCD와 MCU간의 데이터전송이 실제 디스플레이 데이터인지 아니면 명령/상태정보인지 선택합니다.

MCU가 LCD상태를 읽기 위해 LCD에게 명령을 보낼 필요가 있을때, RS는 반드시 LOW가 되어야 합니다. 비슷하게 LCD에 실제 문자 데이터가 전송되거나 받을때는 HIGH가 되어야 합니다.

데이터 전송의 방향은 R/W핀에 의해 제어됩니다. R/W핀이 LOW일시 명령어나 문자 데이터는 LCD모듈에 쓰여지고 HIGH일시, 문자데이터나 상태정보가 LCD 레지스터에서 읽혀집니다. 여기서는 MCU에서 LCD모듈로의 단방향 통신만을 할것이므로 R/W핀은 LOW로 계속 설정됩니다.

Enable핀(E)는 실제 데이터 전송을 초기화 합니다. LCD디스플레이에 쓸때, 데이터는 오직 E 핀이 HIGH에서 LOW로 변화할때 전송이 됩니다.

전원공급 핀

대부분의 LCD모듈 데이터쉬트를 보면 작동을 위해 +5VDC 을 추천하고 있지만, 몇몇 LCD는 3.0-5.5V에서도 잘 동작합니다. Vdd핀은 전원양극에 연결되어야 하며 Vss는 그라운드되어야 합니다. 핀3은 Vee로 디스플레이의 명암을 조절하는데 사용됩니다. 대부분의 경우에 이 핀은 프리셋 포텐티오미터를 통해 0-2V 전압에 연결됩니다.

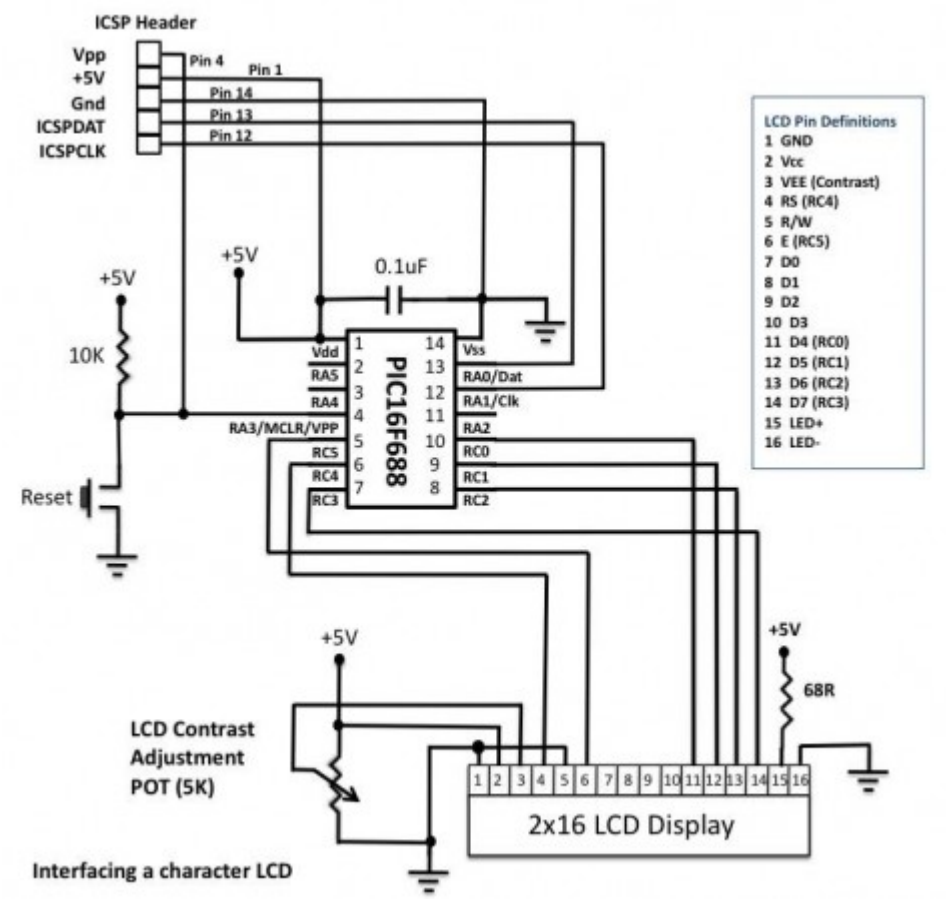
데이타 핀

핀7-핀14는 데이터라인입니다(D0-D7). 디스플레이와의 데이터전송은 8비트나 4비트 모드에서 가능합니다. 8비트 모드는 1바이트 전송을 위해서 8개의 모든 데이터라인을 사용합니다만 4비트 모드에서는 1바이트는 두개의 4비트 니블을 통해 전송됩니다. 그렇기때문에 4비트모드에서는 상위 4개의 데이터라인(D4-D7)만 사용이 됩니다. 이 방법은 4개의 입출력핀을 절약할 수 있기때문에 장점이 있습니다. 여기서는 4비트 모드를 사용하겠습니다.

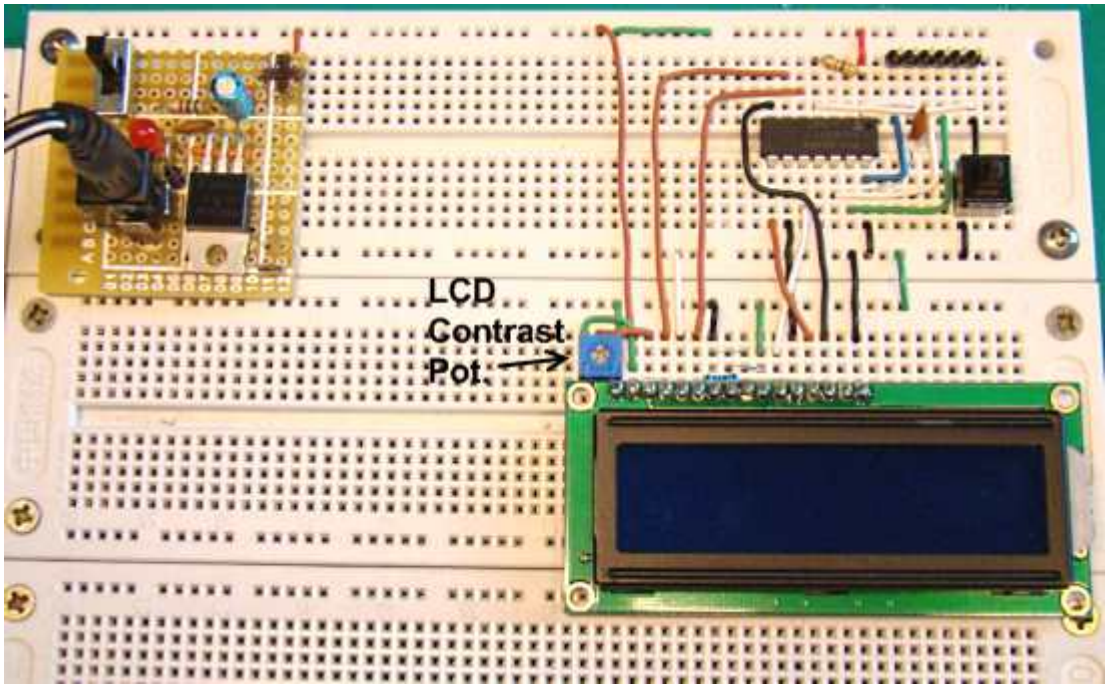
LCD에 대한 좀더 자세한 정보를 원하신다면 다음의 두개의 문서를 참고하십시오. How to use intelligent LCDs *Part 1*, and *Part 2*.

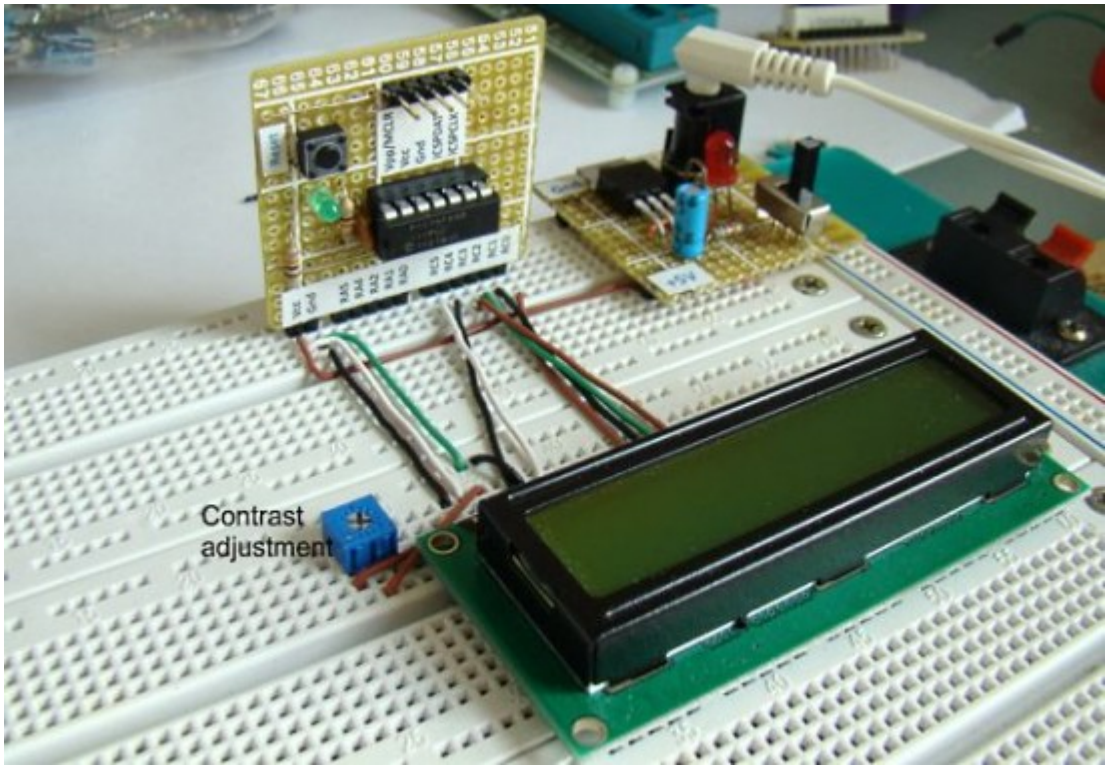
회로도

MCU와 LCD모듈간 데이터전송은 4비트모드에서 될것입니다. LCD로부터 데이터를 읽지 않을 것이므로 LCD의 R/W 핀(5)은 그라운드시킵니다. RC0-RC3은 LCD의 4비트 데이터라인(D4-D7, 11핀-14핀)으로 사용됩니다. 컨트롤라인 RS와 E는 RC4와 RC5에 연결됩니다. 그래서, PIC16F688의 6개의 I/O핀이 LCD모듈에 의해 사용이 됩니다. LCD명암조절은 아래와 같이 5K 포텐티오미터를 사용하여 됩니다. 만약 LCD모듈이 LED백라이트를 가지고 있다면, 68Ω 저항을 핀15나 핀16에 직렬연결하여 LED로 들어가는 전류를 제한합니다. 자세한 회로도는 아래와 같습니다.

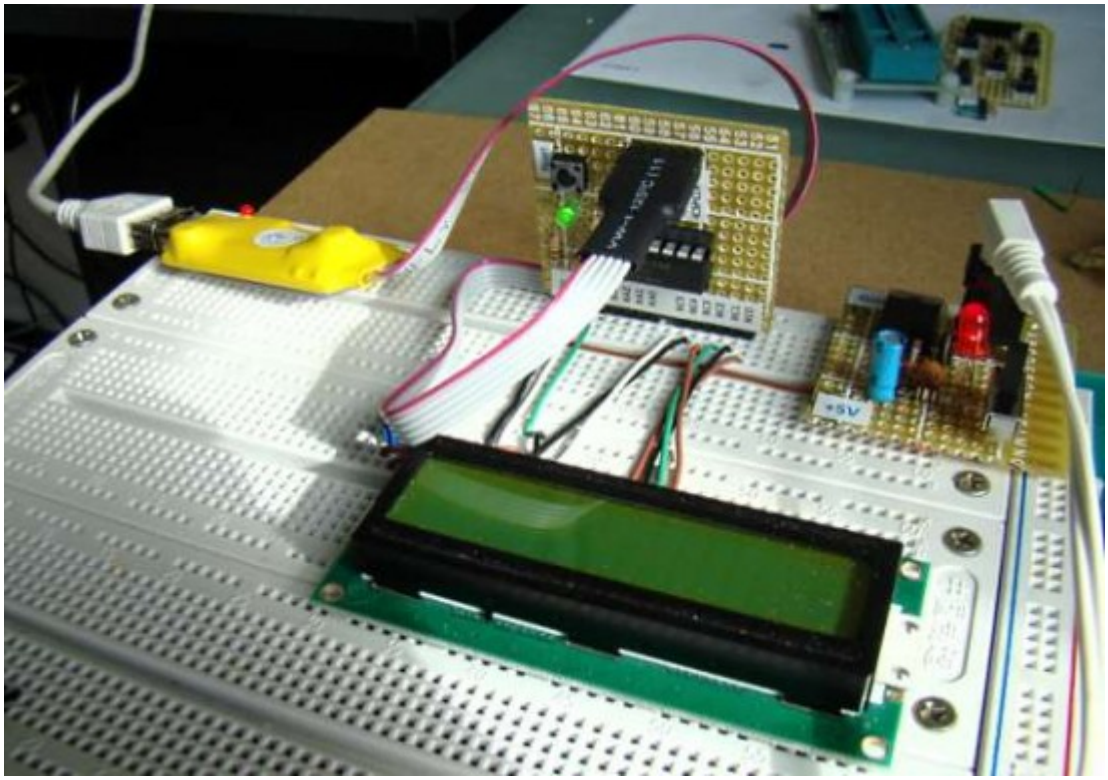


브레드보드에 위의 회로를 구성하여 보았습니다. 브레드보드상의 와이어를 줄여 공간을 활용하고 싶다면 이전 게시물을 참조하여 PIC16F688 모듈을 구성하여 아래의 그림과 같이 브레드보드에 삽입하세요.





PIC16F688 브레드보드 모듈과 LCD연결



소프트웨어

LCD데이터와 컨트롤라인은 PORTC를 통하여 작동합니다. 그래서 PORTC는 출력포트로 정의되어 있어야합니다(TRISC=0). 컴패레이터 기능은 비활성화 시키고(CMCON0=7) 모든 I/O핀은 디지털로 선택합니다.(ANSEL=0).

HD44780 LCD를 프로그래밍하는 것은 약간 복잡한데 왜냐하면 정확한 타이밍과 다양한 신호와 명령을 적절한 순서로 처리해야하기 때문입니다.

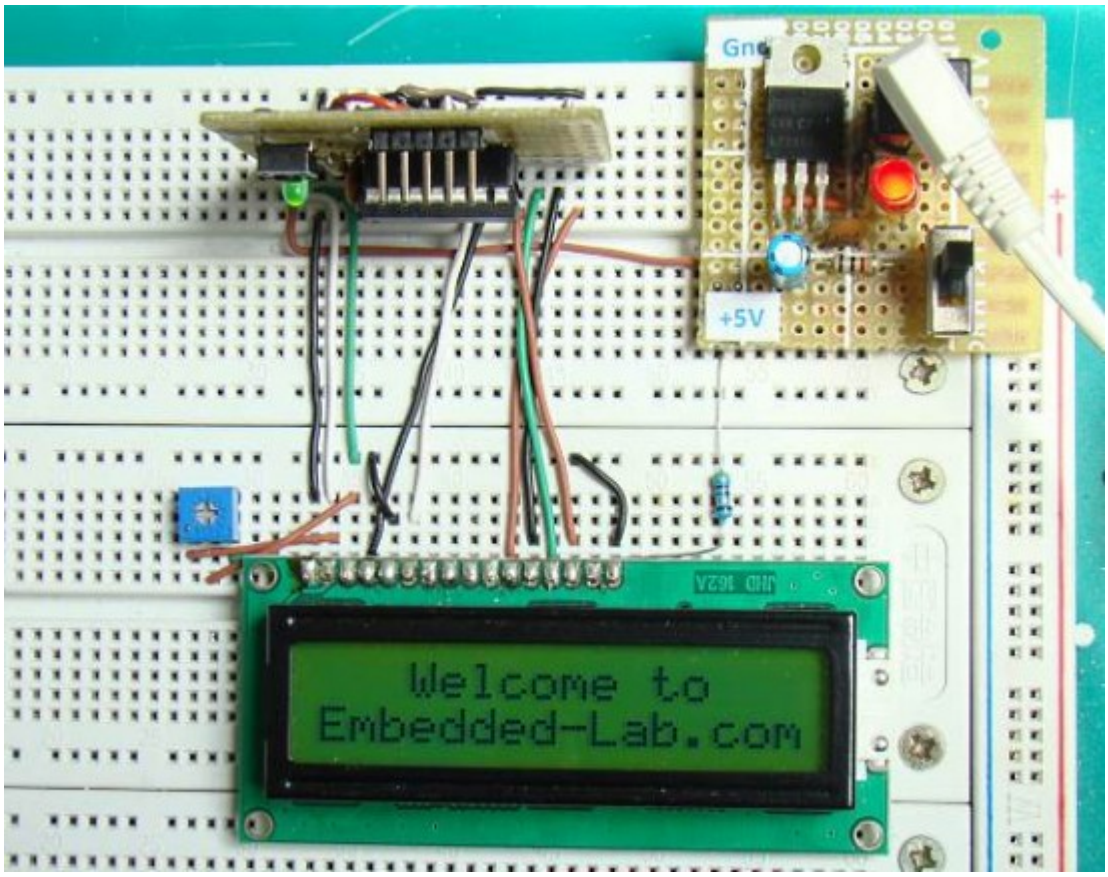
하지만 다행이도 mikroC 컴파일러는 HD44780기반 LCD모듈과 4비트 모드로 통신하기 위한 내장 라이브러리를 가지고 있고 이것은 프로그래밍을 매우 간편하게 만들어 줍니다. LCD용 내장 함수를 사용하기 전에, PIC MCU로 연결된 LCD핀을 반드시 정의하여주어야 합니다. 아래의 소스상의 주석은 mikroC 컴파일러의 LCD라이브러리를 어떻게 사용하는지 설명하고 있습니다.

```
/*

Lab 4: Blink Character Message on LCD

Internal Clock @ 4MHz, MCLR Enabled, PWRT Enabled, WDT OFF

<pre></pre>*/

// Define LCD module connections.

 sbit LCD_RS at RC4_bit;

 sbit LCD_EN at RC5_bit;

 sbit LCD_D4 at RC0_bit;

 sbit LCD_D5 at RC1_bit;

 sbit LCD_D6 at RC2_bit;

 sbit LCD_D7 at RC3_bit;

 sbit LCD_RS_Direction at TRISC4_bit;

 sbit LCD_EN_Direction at TRISC5_bit;

 sbit LCD_D4_Direction at TRISC0_bit;

 sbit LCD_D5_Direction at TRISC1_bit;

 sbit LCD_D6_Direction at TRISC2_bit;
```

```
 sbit LCD_D7_Direction at TRISC3_bit;

// End LCD module connection definition

// Define Messages

 char message1[] = "Welcome to";

 char message2[] = "Embedded-Lab.com";

 void main() {

  ANSEL = 0b00000000; //All I/O pins are configured as digital

  CMCON0 = 0x07 ; // Disbale comparators

  TRISC = 0b00000000; // PORTC All Outputs

  TRISA = 0b00000000; // PORTA All Outputs, Except RA3

  Lcd_Init();                    // Initialize LCD

  do {

   Lcd_Cmd(_LCD_CLEAR);            // CLEAR display

   Lcd_Cmd(_LCD_CURSOR_OFF);        // Cursor off

   Lcd_Out(1,4,message1);           // Write message1 in 1st row

   Lcd_Out(2,1,message2);           // Write message2 in 2nd row

   Delay_ms(1000);                  // Wait for 1 sec

   Lcd_Cmd(_LCD_CLEAR);     // Clear display

   Delay_ms(1000); // Wait for 1 sec

  } while(1);            // Infinite Loop

 }
```

결과

컴파일후 HEX파일을 PIC16F688에 로드한후 전원이 들어오면 Welcome to Embedded Lab.com 이라는 메세지를 LCD에 보실수 있습니다. 매 초마다 깜빡입니다.



가치창조기술 | www.vctec.co.kr

[PIC 마이컴 실험하기] 5. ADC(Analog-to-digital conversion)

마이컴 실험실

2011/10/05 14:02 http://blog.naver.com/ubicomputing/150120596978

아날로그-디지털 변환(ADC)는 임베디드 시스템에서 필수적인 요소입니다. 임베디드 시스템자체는 디지털이지만 시스템이 다루는 주변환경은 온도, 속도, 압력, 마이크로폰 출력 등의 아날로그이기 때문입니다. 이런 아날로그 신호는 mcu에 의해 가공되기 전에 모두 디지털데이터로 변환이 되어야 합니다.

이번 게시물에서는 PIC16F688을 이용하여 아날로그 신호를 읽고 디지털로 변환한 출력값을 어떻게 LCD에 출력하는지 알아보겠습니다. 입력 아날로그 신호는 포텐티오미터를 이용하여 조절한 0-5V의 전압이 될것입니다.

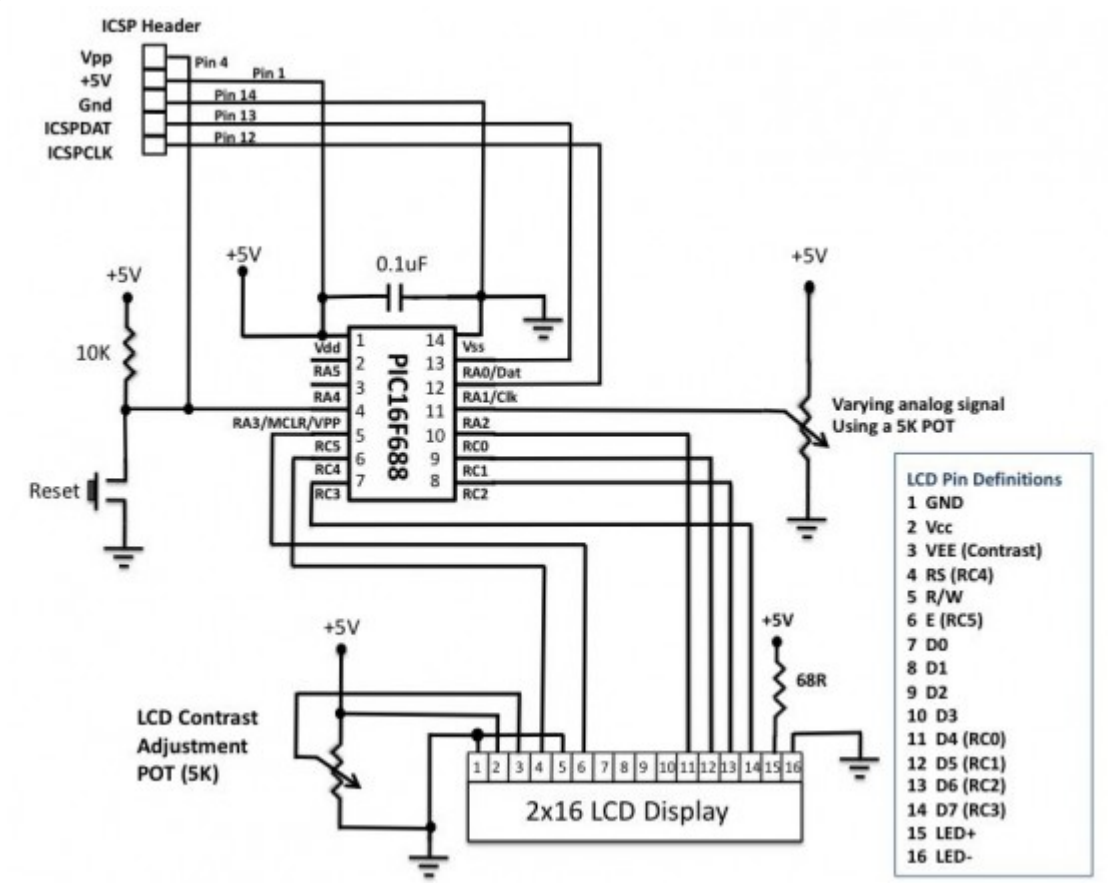
선행이론

PIC16F688 MCU는 8개의 입력채널을 가진 내장형 10비트 ADC를 가지고 있습니다. 8개의 채널은 RA0, RA1, RA2, RA4, RC0, RC1, RC2, RC3을 통해 사용이 가능합니다. 핀아웃을 보면 이 핀들에 대해 AN0-AN7의 라벨이 붙어 있는 것을 확인 할 수 있습니다. ADC의 입력채널로 핀들이 쓰여질때 이것들은 하나의 Sample and Hold 회로로 멀티플렉스되고, 그 Sample and Hold회로의 출력은 A/D converter의 입력에 연결이 됩니다.
A/D 변환은 successive approximation based conversion이며 변환된 10비트 결과는 ADC 결과값 레지스터인 ADRESH(A/D Result higher byte)와 ADRESL(A/D Result lower byte)에 저장이 됩니다. 이 두개의 레지스터는 8비트입니다.
A/D모듈의 기능은 ANSEL, ADCON0, ADCON1의 세개의 레지스터에 의해 컨트롤됩니다. 좀더 자세한 내용은 이전 게시물 "PIC16F688의 ADC 채널"을 참조하십시오.

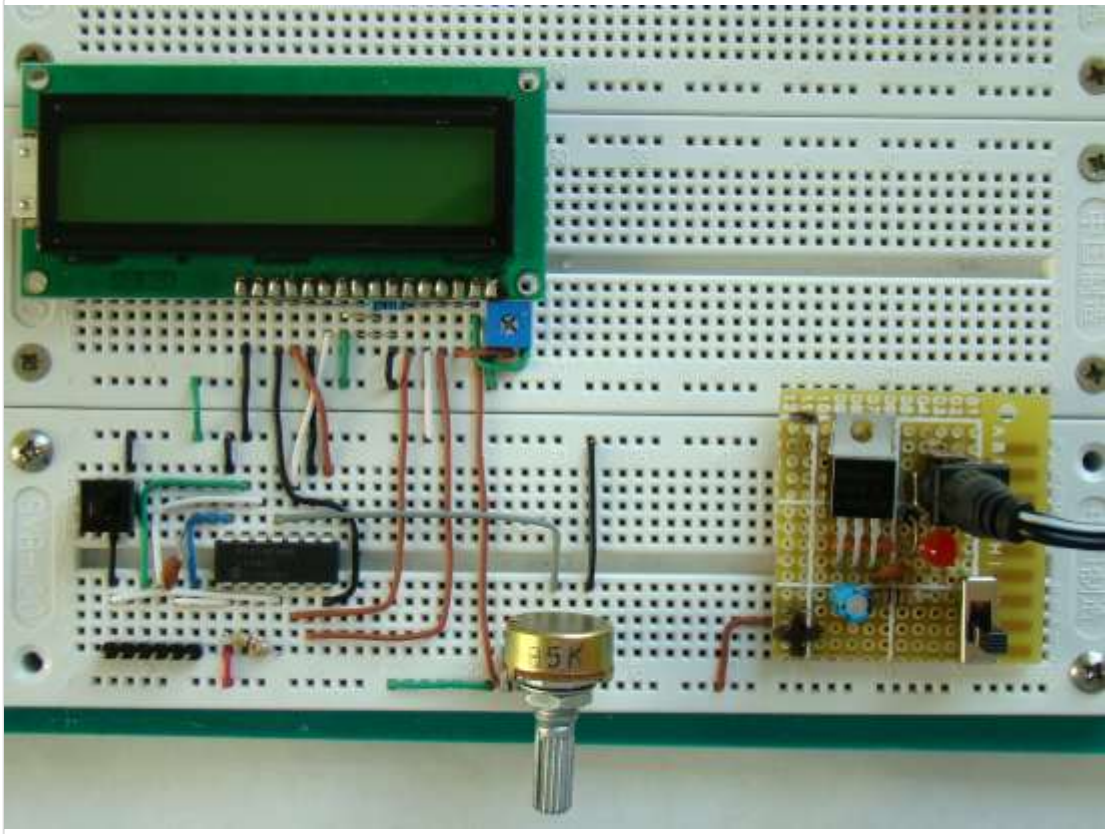
회로도

10비트 A/D변환을 데모하기 위한 테스트 회로는 아래의 그림과 같습니다. ADC용 테스트 입력전압은 5V전원에 연결된 5K 포텐티오미터로부터 얻어지며 PIC16F688의 RA2/AN2핀에 연결이 되어 있습니다. 공급전압(+5V)는 A/D변환의 리퍼런스 전압으로 선택되었습니다. 그렇기때문에 10비트 ADC는 0-5V사이의 아날로그 전압을 변환하여 0-1023사이의 디지털 숫자로 보여줄 것입니다. 이 디지털 숫자는 LCD에 표시가 되게 됩니다.

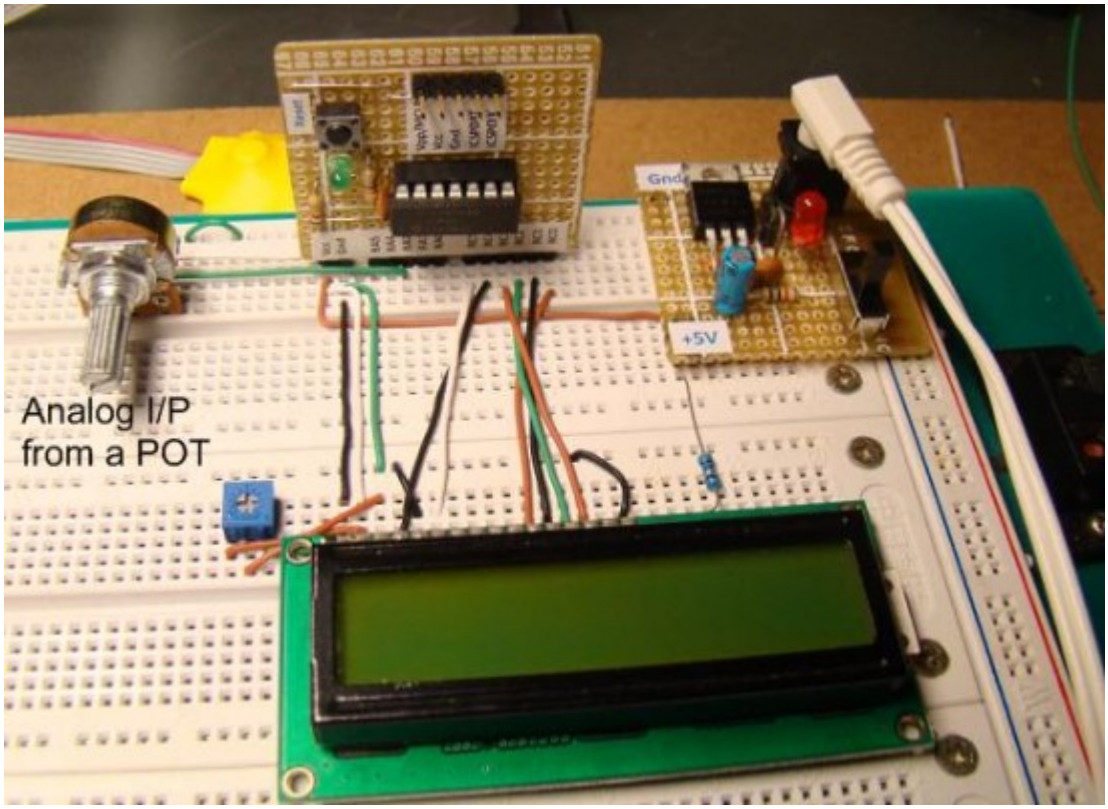
아래의 회로는 이전 게시물 "LCD연결하기"에서 보았던 회로와 5K 포텐티오미터가 연결되어 있는 것을 빼고는 많이 다르지 않습니다.



회로도



브레드보드상에 회로 셋업



PIC16F688 브레드보드 모듈을 이용하여 회로 셋업

소프트웨어

ANSEL, ADCON0, ADCON1 레지스터를 설정하여 프로그래밍합니다. RA2/AN2는 아날로그 입력으로 사용되므로 해당 ANSEL 비트는 반드시 셋팅되어야 합니다. ADCON0레지스터에서는 CHS1을 셋팅하고 CHS0와 CHS2는 클리어하여 AN2 채널을 내장 Sample & Hold 회로에 연결합니다.

ADCON0의 VCFG비트를 클리어해서 전원전압(+5V)을 A/D변환의 리퍼런스 전압으로 선택합니다. ADCON1 레지스터는 A/D변환을 위한 클럭소스를 선택하는데 사용이됩니다. 하지만 mikroC 컴파일러는 ADC를 위한 내장 RC클럭을 사용하는 ADC_Read() 내장 라이브러리 함수를 가지고 있습니다. 그래서 ADCON1레지스터를 초기화할 필요가 없습니다. 프로그램 코드는 아래와 같습니다.

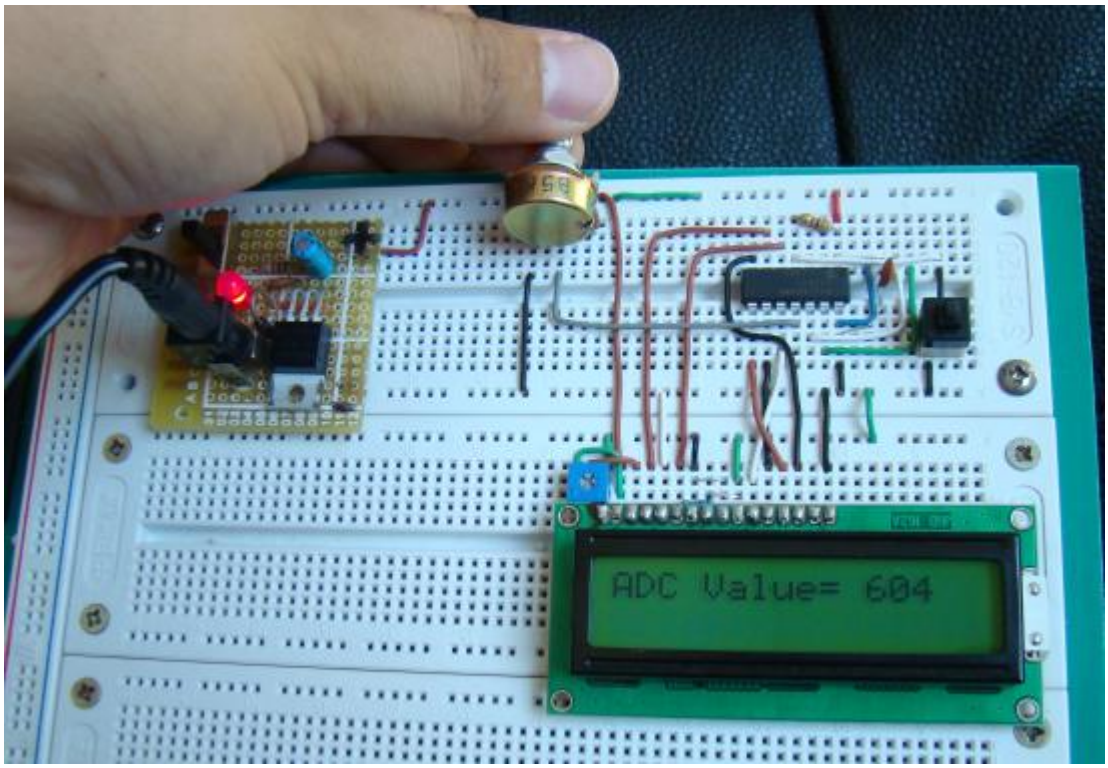
```
/*

 Lab 5: Analog-to-digital converter

 Internal Oscillator @ 4MHz, MCLR Enabled, PWRT Enabled, WDT OFF

 Copyright @ Rajendra Bhatt

 October 10, 2010

*/
```

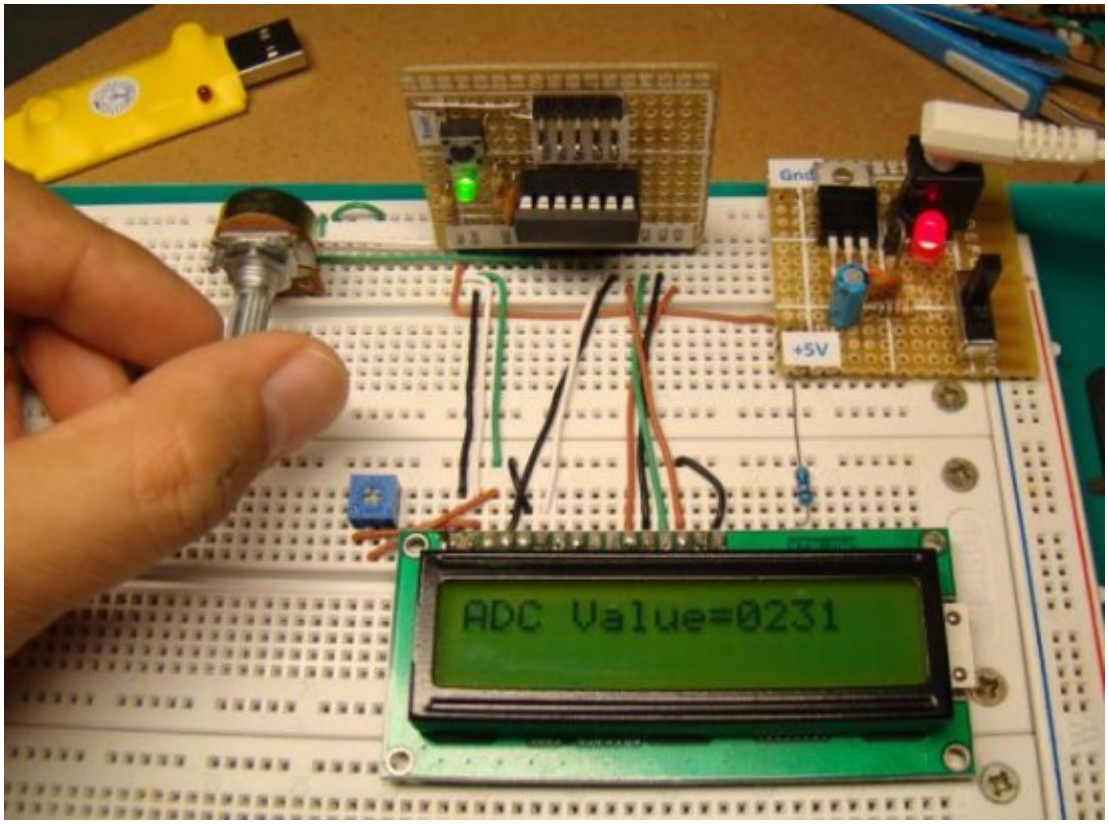
```
// LCD module connections
 sbit LCD_RS at RC4_bit;
 sbit LCD_EN at RC5_bit;
 sbit LCD_D4 at RC0_bit;
 sbit LCD_D5 at RC1_bit;
 sbit LCD_D6 at RC2_bit;
 sbit LCD_D7 at RC3_bit;
 sbit LCD_RS_Direction at TRISC4_bit;
 sbit LCD_EN_Direction at TRISC5_bit;
 sbit LCD_D4_Direction at TRISC0_bit;
 sbit LCD_D5_Direction at TRISC1_bit;
 sbit LCD_D6_Direction at TRISC2_bit;
 sbit LCD_D7_Direction at TRISC3_bit;
// End LCD module connections
// Define Messages
 char message1[] = "ADC Value= ";
 char *temp = "0000";
 unsigned int ADC_Value;
 void main() {
  ANSEL = 0b00000100; // RA2/AN2 is analog input
  ADCON0 = 0b00001000; // Analog channel select @ AN2
  CMCON0 = 0x07 ; // Disbale comparators
```

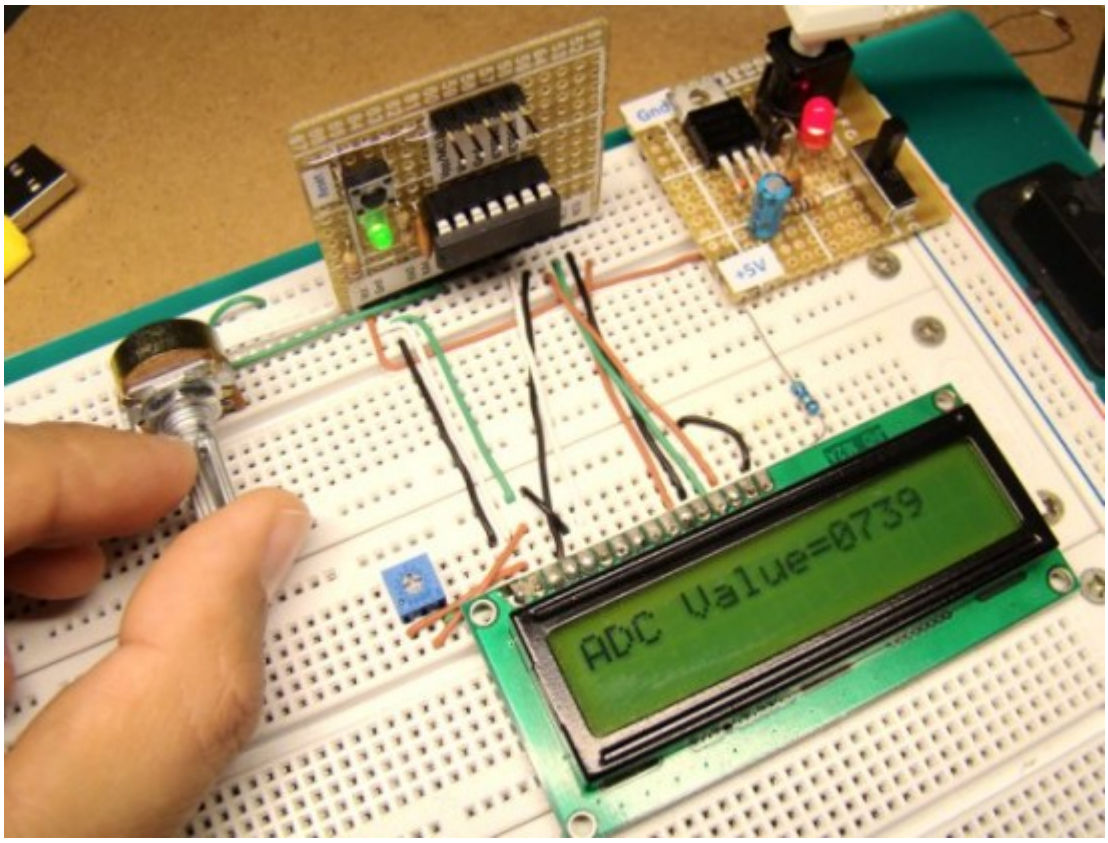
```
 TRISC = 0b00000000; // PORTC All Outputs

 TRISA = 0b00001100; // PORTA All Outputs, Except RA3 and RA2

 Lcd_Init();                    // Initialize LCD

 Lcd_Cmd(_LCD_CLEAR);           // CLEAR display

 Lcd_Cmd(_LCD_CURSOR_OFF);      // Cursor off

 Lcd_Out(1,1,message1);         // Write message1 in 1st row

 do {

  adc_value = ADC_Read(2);

  temp[0] = adc_value/1000 + 48; // Add 48 to get the ASCII character value

  temp[1] = (adc_value/100)%10 + 48;

  temp[2] = (adc_value/10)%10 + 48;

  temp[3] = adc_value%10 + 48;

  Lcd_Out(1,11,temp);

  Delay_ms(2000);

 } while(1);

}
```

결과

포텐티오미터를 움직여가면 0-1023사이에서 움직이는 10비트 변환 결과를 볼 수 있습니다.


ADC Value= 604


Gnd
+5V
ADC Value=0231



가치창조기술 | www.vctec.co.kr

[PIC 마이컴 실험하기] 6. Seven Segment(7-Seg) 디스플레이

마이컴 실험실

2011/10/06 13:52 http://blog.naver.com/ubicomputing/150120682742

Seven segment LED 디스플레이는 VCR, 전자렌지, 장난감 등의 가정용품등에 많이 사용됩니다. 7-Seg 디스플레이는 기번적으로 10진수를 표시하기 위해사용되지만 몇몇 알파벳이나 다른 문자도 표현을 할 수있습니다. 이번 게시물에서는 7-Seg디스플레이를 PIC16F688 MCU에 연결하는 방법에 대해 설명하겠습니다. 이를 위해 00h-0Fh까지를 카운트하는 16진수 카운터를 만들어보도록 하겠습니다.

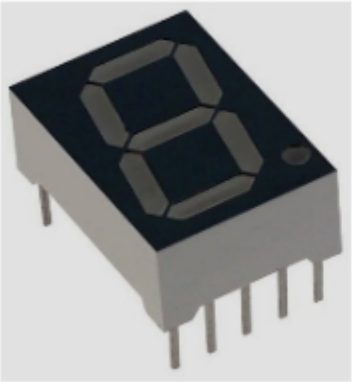
선행이론

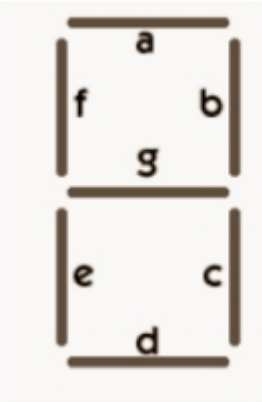
7-Seg LED 디스플레이는 7개의 LED를 日 형태로 배열한 것으로 각각의 LED를 켜거나 끔으로서 0에서 9까지의 숫자를 표시하는 디스플레이 입니다. 그래서 7-Seg 디스플레이를 동작시키기 위해서는 마이크로컨트롤러로부터 7개의 출력이 필요합니다. 만약 소수점을 표시해야된다면 1개의 추가적인 출력이 필요합니다.

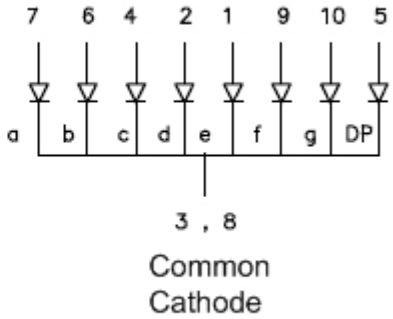
각각의 LED 세그먼트는 소문자로 a,b,c,d,e,f,g로 표기되어 있고 소수점은 dp로 표기됩니다. 디스플레이 안의 8개의 LED는 일반적인 양극이나 음극설정으로 배열될 수 있습니다. 일반적인 음극 디스플레이에서는 모든 LED세그먼트의 음극을 묶어 그라운드로 연결하여야 합니다. 그후에 양극에 로직1을 적용하여 LED세그먼트를 ON시킵니다. 일반적인 양극 디스플레이에서는 모든 양극을 묶어 전원전압인 Vcc에 연결합니다. 그후 음극에 로직 0을 적용하여 LED세그먼트를 ON 시킵니다.

한개 이상의 7-Seg 디스플레이가 사용될때는 필요한 MCU 핀을 줄이기위해 멀티플렉싱 기술이 사용이 됩니다.

본 실험에서는 LT543 음극 7-Seg 디스플레이가 사용되었고 LED가 ON되면 빨간색 빛을 내보냅니다. 이 모듈은 10개의 핀을 가지고 있고 설정은 아래와 같습니다.



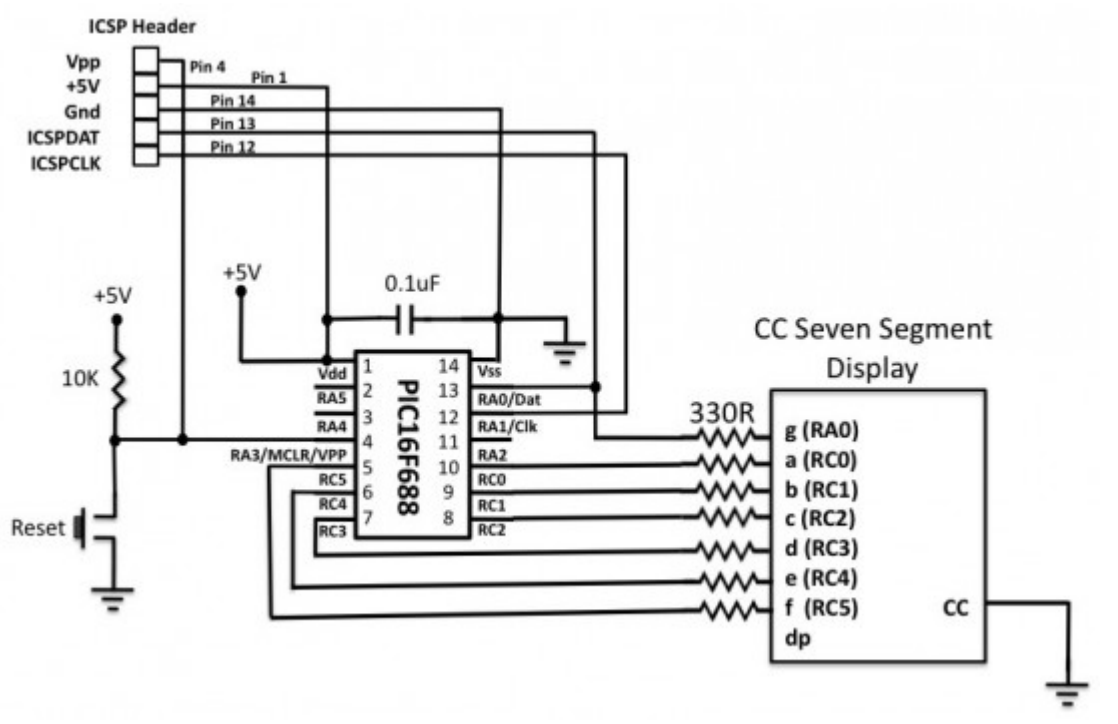




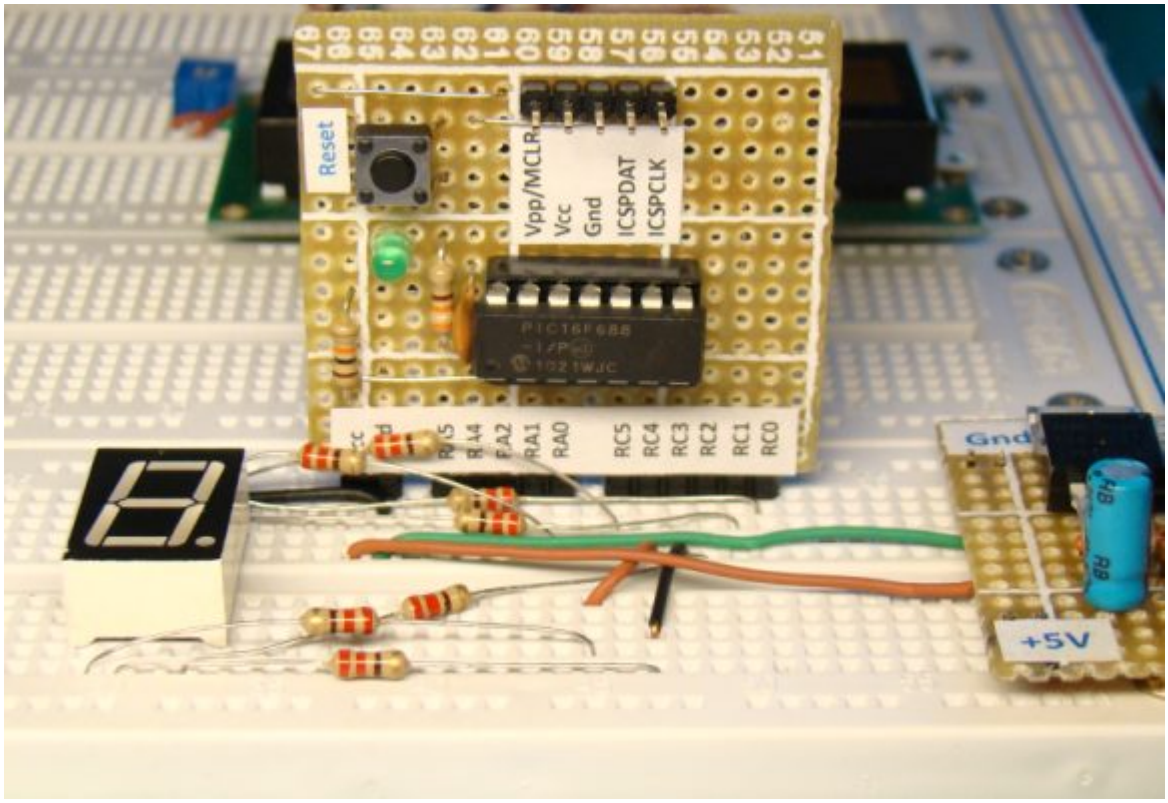
모든 음극은 함께 묶여 있고 이 연결은 3번과 8번핀에 연결되어 있으며, 전류제한 저항이 직렬로 연결된 MCU의 I/O핀을 통해 양극에 신호를 줄수 있습니다. 숫자를 디스플레이하는것은 적절한 세그먼트 LED를 ON/OFF시키는 것이 필요합니다. 예를 들어 숫자 7을 디스플레이하기 위해서는 세그먼트 a, b, c만 ON상태이어야 합니다.

회로도

아래는 음극 7-Seg LED 디스플레이를 PIC16F688에 연결하기 위한 회로도 입니다. MCU의 I/O핀은 세그먼트 LED에 필요한 전류를 공급합니다. 전류를 제한하기 위해 MCU의 핀들과 세그먼트 LED사이에 각각 330 Ω의 저항을 달았습니다. 세그먼트 LED a-f는 RC0-RC5를 통해 동작하고, 세그먼트 g는 RA2에 의해 동작합니다. 소수점은 사용하지 않습니다. 좀 더 자세한 사항은 사용하는 세그먼트 LED의 데이터쉬트를 참고하십시오.



7-Seg 디스플레이 연결 회로도



브레드보드상에 회로셋업하기

소프트웨어

세그먼트 LED는 PORTC와 PORTA를 통해 동작합니다. 그래서 이 두개의 포트는 디지털 출력으로 정의되어 있어야하며 이 핀들에 있는 컴패레이터 역시 비활성화 시켜야 합니다. 소스코드는 세그먼트 LED에 0부터 15까지 1초간격으로 표시를 하고 다시 0으로 되돌아가도록 짜여져 있습니다. 세그먼트LED에는 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F로 표시됩니다. 아래의 프로그램을 mikroC 컴파일러로 컴파일하고 HEX파일을 PIC16F688에 올려 보십시오. 클럭 및 configuration bit셋팅은 첫번째 게시물 "LED깜빡이기"를 참고하세요.

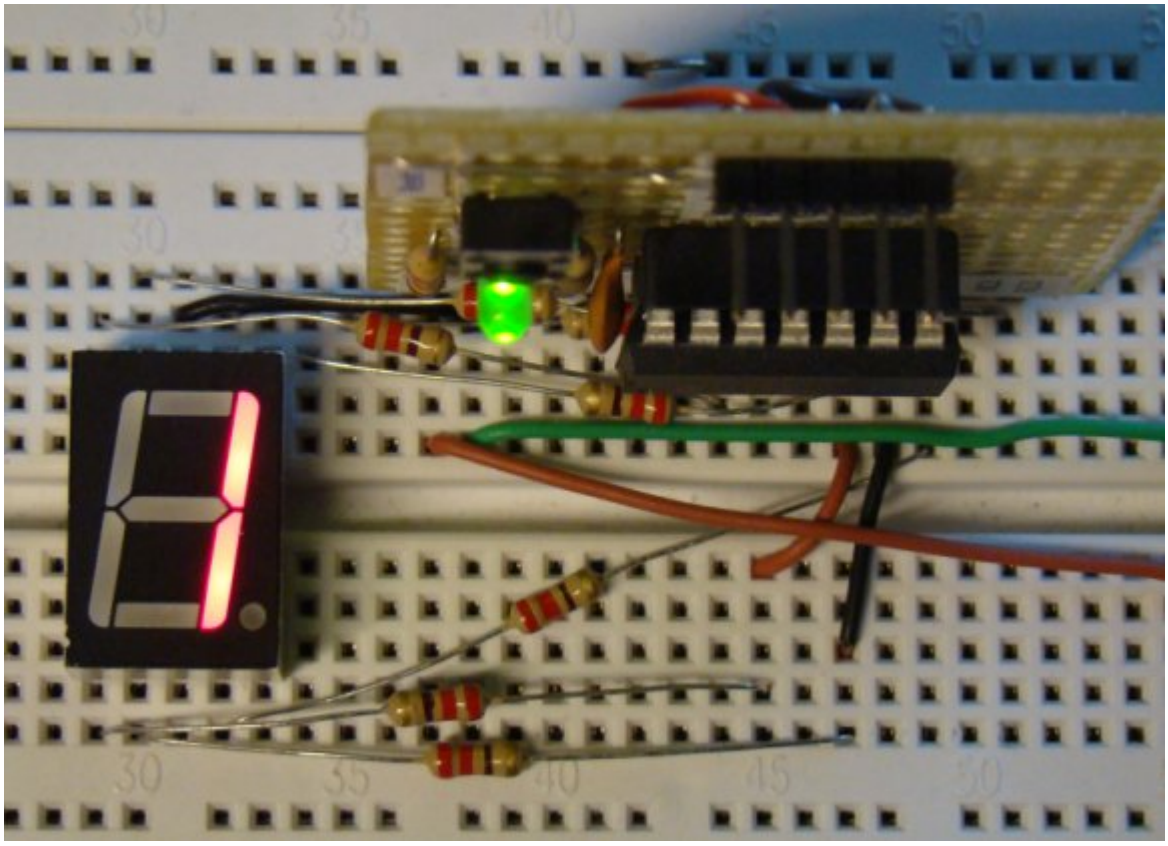
```
/*
 Lab 6: Seven segment display
 Copyright @ Rajendra Bhatt
 November 13, 2010
*/
// Define seven segment connections
 sbit seg_a at RC0_bit;
 sbit seg_b at RC1_bit;
 sbit seg_c at RC2_bit;
 sbit seg_d at RC3_bit;
 sbit seg_e at RC4_bit;
```

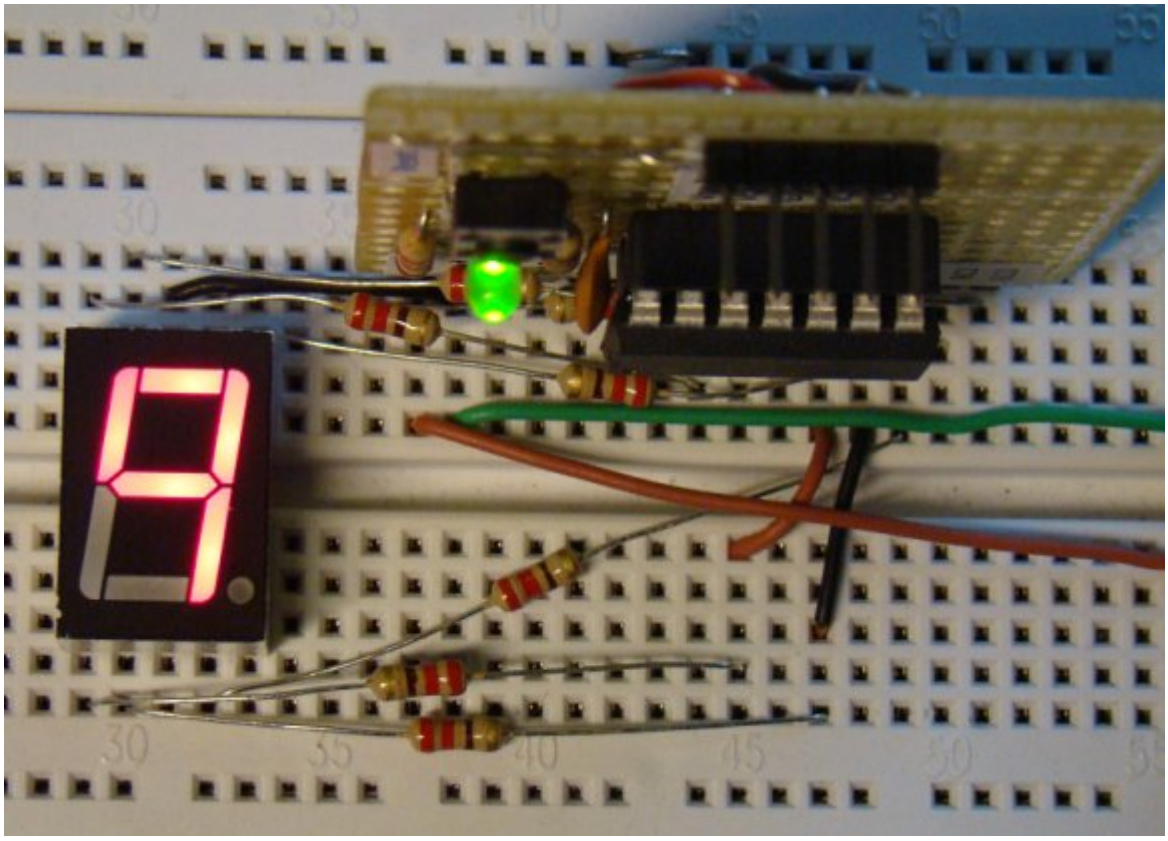
```
sbit seg_f at RC5_bit;
sbit seg_g at RA0_bit;
unsigned short count=0;
void main() {
 ANSEL = 0b00000000; //All I/O pins are configured as digital
 CMCON0 = 0x07 ; // Disbale comparators
 TRISC = 0b00000000; // PORTC All Outputs
 TRISA = 0b00001000; // PORTA All Outputs, Except RA3
 do {
  switch (count) {
  case 0 : seg_a=1; seg_b=1; seg_c=1;
          seg_d=1; seg_e=1; seg_f=1; seg_g=0;
          break;
  case 1 : seg_a=0; seg_b=1; seg_c=1; seg_d=0;
          seg_e=0; seg_f=0; seg_g=0;
          break;
  case 2 : seg_a=1; seg_b=1; seg_c=0; seg_d=1;
          seg_e=1; seg_f=0; seg_g=1;
          break;
  case 3 : seg_a=1; seg_b=1; seg_c=1; seg_d=1;
          seg_e=0; seg_f=0; seg_g=1;
          break;
  case 4 : seg_a=0; seg_b=1; seg_c=1; seg_d=0;
          seg_e=0; seg_f=1; seg_g=1;
          break;
  case 5 : seg_a=1; seg_b=0; seg_c=1; seg_d=1;
          seg_e=0; seg_f=1; seg_g=1;
          break;
  case 6 : seg_a=1; seg_b=0; seg_c=1; seg_d=1;
          seg_e=1; seg_f=1; seg_g=1;
          break;
  case 7 : seg_a=1; seg_b=1; seg_c=1; seg_d=0;
          seg_e=0; seg_f=0; seg_g=0;
          break;
```

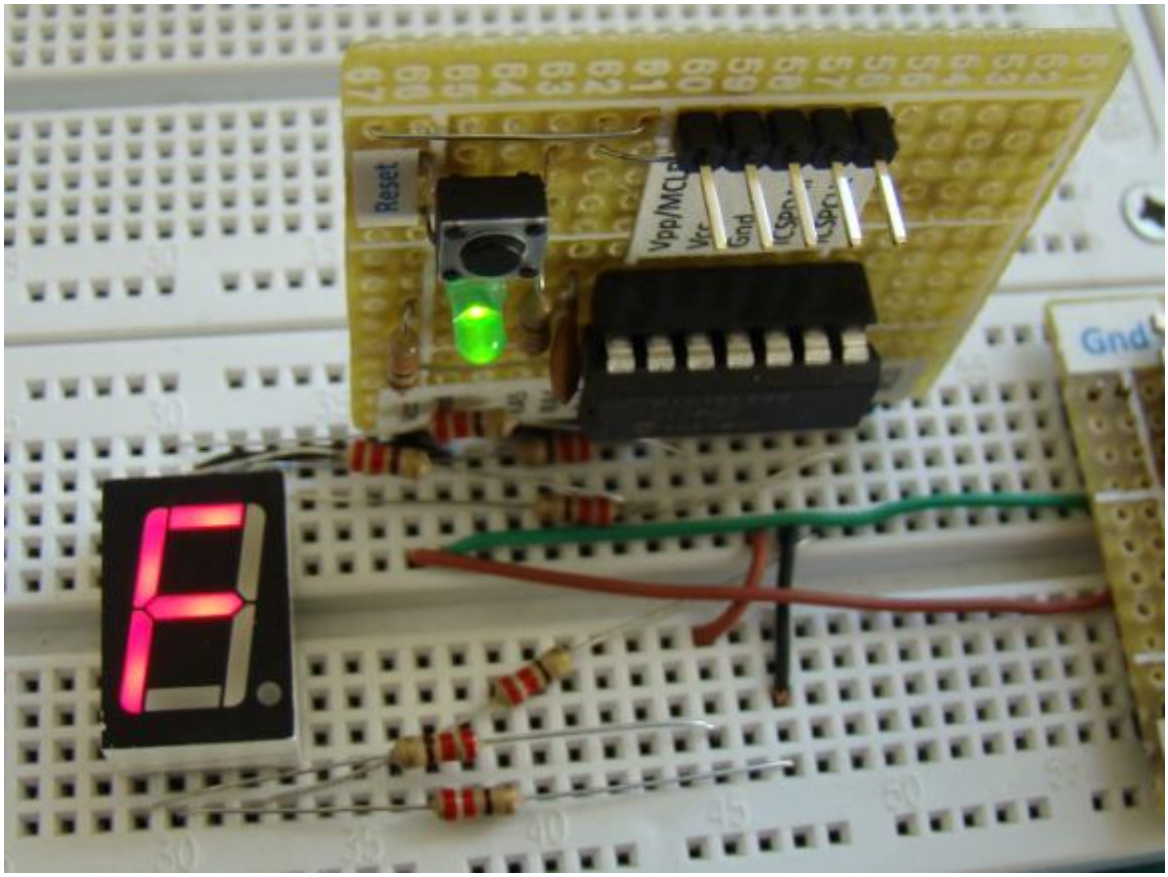
```
  case 8 : seg_a=1; seg_b=1; seg_c=1; seg_d=1;
          seg_e=1; seg_f=1; seg_g=1;
          break;
  case 9 : seg_a=1; seg_b=1; seg_c=1; seg_d=0;
          seg_e=0; seg_f=1; seg_g=1;
          break;
  case 10 : seg_a=1; seg_b=1; seg_c=1; seg_d=0;
          seg_e=1; seg_f=1; seg_g=1;
          break;
  case 11 : seg_a=0; seg_b=0; seg_c=1; seg_d=1;
           seg_e=1; seg_f=1; seg_g=1;
           break;
  case 12 : seg_a=1; seg_b=0; seg_c=0; seg_d=1;
           seg_e=1; seg_f=1; seg_g=0;
           break;
  case 13 : seg_a=0; seg_b=1; seg_c=1; seg_d=1;
           seg_e=1; seg_f=0; seg_g=1;
           break;
  case 14 : seg_a=1; seg_b=0; seg_c=0; seg_d=1;
           seg_e=1; seg_f=1; seg_g=1;
           break;
  case 15 : seg_a=1; seg_b=0; seg_c=0; seg_d=0;
           seg_e=1; seg_f=1; seg_g=1;
           break;
 } //case end
 count ++;
 if(count ==16) count =0;
 Delay_ms(1000); // Wait for 1 sec before updating the display
} while(1);  // Infinite Loop
}
```

결과

프로그램이 mcu에 로드되면 전원을 켜고 HEX 카운터가 0부터 F까지 디스플레이를 하는 것을 보실수 있습니다.







가치창조기술 | www.vctec.co.kr

[PIC 마이컴 실험하기] 7. PIC 타이머와 카운터 (Part 1)

마이컴 실험실

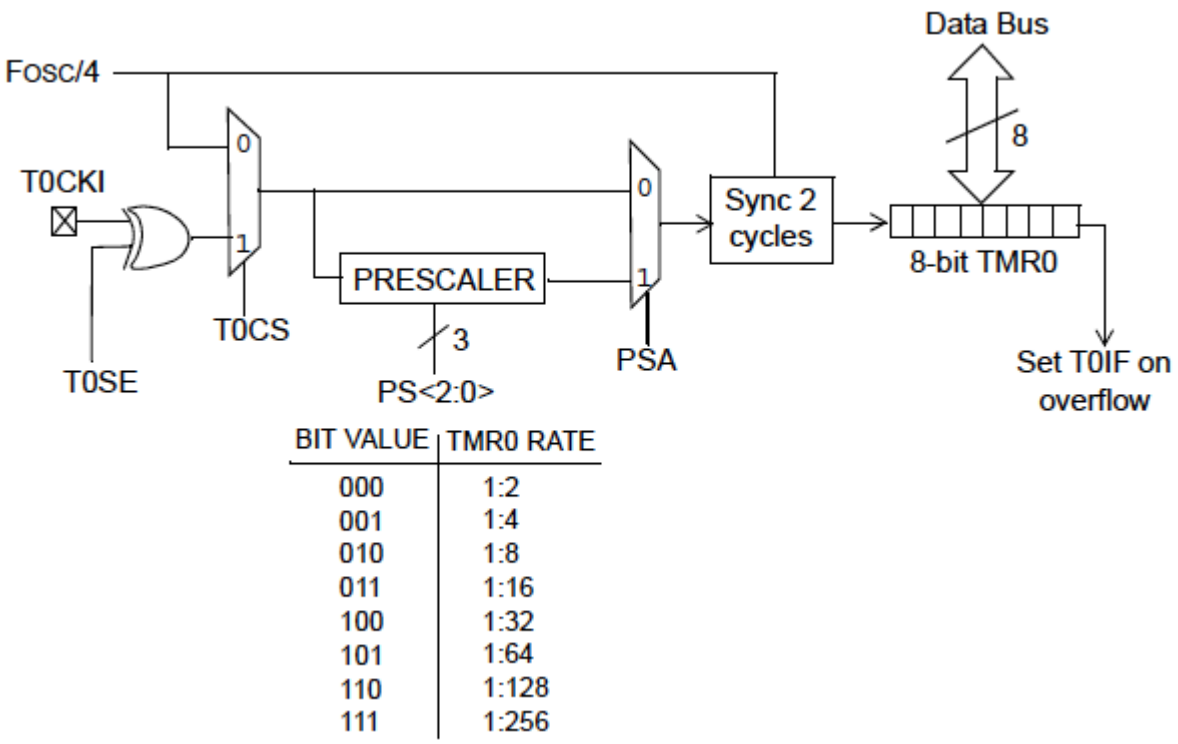
2011/10/07 16:26 http://blog.naver.com/ubicomputing/150120792778

PIC MCU는 타이머라고 불리우는 한개 이상의 정밀한 타이밍 시스템을 가지고 있습니다. 이 타이머들은 정해진시간에 이벤트 생성하기, 이벤트 지속기간 측정하기, 날짜와 시간 저장하기, 이벤트 카운트하기 등의 시간과 관련된 다양한 기능을 수행하는데 사용됩니다. 타이머 모듈의 핵심 요소는 들어오는 매 펄스를 세는 바이너리 카운터입니다. 독립적으로 작동하기때문에, 메인프로그램이 실행되더라고 펄스를 동시에 카운트할수 있습니다. PIC16F688 MCU는 Timer0, Timer1의 두 개의 내장 하드웨어 타이머 모듈이 있습니다. 이번 게시물에서는 Timer0 모듈의 기능에 대해 살펴보도록하겠습니다.

선행이론

타이머/카운터 모듈의 기본적인 컨셉은 "MCU의 타이머와 카운터 원리"에서 설명되었습니다. PIC16F688의 Timer0 모듈은 8비트 동기 카운터로 타운터 값을 TMR0라고 불리는 SFR레지스터에 저장합니다. 이 레지스터는 소프트웨어를 통해 언제든지 읽고 쓸수 있습니다. 레지스터에 어떤 값을 쓴다면, 카운터는 그 값에서부터 카운터 증가를 시작합니다.

Timer0 모듈이 프로세서의 인스트럭션 클럭에 의해 동작이 된다면, 고정된 프로세서 클럭에 의해 일정한 비율로 증가하기때문에 Timer0모듈을 타이머라고 말할 수 있습니다. 왜냐하면 카운트된 펄스의 숫자를 안다면 경과된 시간을 유추하여 낼수 있기때문입니다. Timer0는 T0CKI(Timer Zero Clock Input)라는 별칭이 붙여진 RA2핀(11)에 들어오는 외부 펄스를 카운트 할수 있습니다. 외부 펄스를 카운트할때는 Timer0모듈은 카운터로 동작합니다. 이 두가지 모드에 대해서는 각각 다루어 보도록 하겠습니다.



Timer0 모듈의 블럭 다이어그램 (Source: Microchip)

타이머 모드

타이머모드는 TOCS비트(OPTION레지스터, bit 5)를 클리어함으로써 선택이 가능합니다. 타이머모드에서는 TMR0레지스터는 매 인스트럭션 사이클을 증가시킵니다. 8비트 레지스터이기 때문에 TMR0은 00에서 FF(255)까지 카운트가 가능하며 FF에 도달했을때 00으로 돌아갑니다. 그리고 이 레지스터 오버플로우는 INCON(Interrupt Control) 레지스터의 T0IF(Timer0 Interrupt Flag) 비트를 1로 셋팅함으로써 기록됩니다.

인터럽트가 활성화 되어 있을때 T0IF비트를 셋팅하는것은 Timer0 인터럽트를 트리거 시킵니다. Timer0 인터럽트는 INTCON레지스터의 T0IE 비트(Timer0 Interrupt Enable)와 GIE(Global Interrup Enable) 비트를 셋팅시 활성화 됩니다. 이 인터럽트는 타임아웃을 알리시에나 TMR0레지스터의 오버플로우를 알려주는데 사용될수 있습니다. T0IF비트는 인터럽트서비스루틴에 의해 반드시 클리어되어야만 타이머 인터럽트를 다시 발생시킬수 있습니다.

PIC MCU를 4MHz 클럭으로 동작시킨다고 했을때 인스트럭션 클럭은 1MHz가 됩니다(1 instruction cycle = 4 clock cycles). 그렇다면 카운터는 매 1 μs 마다 정확히 클럭이 공급됩니다. 이것은 Timer0가 0에서 시작하여 다시 0으로 돌아오는데 256 μs 가 걸린다는 것을 의미합니다. TMR0 레지스터에 적당한 값을 미리로딩하여 좀더 작은 타이머 인터벌 선택이 가능합니다. 예를들면, TMR레지스터에 206의 값을 미리로딩하면, Timer0 오버플로우는 50 μs 후에 일어나게 됩니다.

게다가 프리스케일러(eight bit programmable divider)를 이용하여 좀더 긴 시간 기간을 측정할수 있습니다. 프리스케일러는 클럭소스와 타이머사이에서 주파수를 나누는 회로입니다. 입력주파수를 2-256사이의 값으로 나눌수가 있습니다.

1MHz의 인스트럭션클럭이 있다면, 타이머의 최대값은 256 x 256 μs = 65.536 ms (프리스케일러 값을 256으로 사용 시)가 됩니다. 프리스케일러의 값은 OPTION레지스터의 PS0, PS1, PS2비트를 통하여 소프트웨어상 선택이 가능합니다. Timer0모듈을 프리스케일러와 사용하기 위해서는 OPTION레지스터의 PSA(Prescaler Assignment)비트를 반드시 클리어 해야합니다. 만약 PSA비트가 셋팅이 되어 있다면 Timer0모듈에 프리스케일러가 할당되지 않습니다. OPTION레지스터와 INTCON레지스터의 각각의 비트에 대한 설명은 아래와 같습니다.

REGISTER 5-1: OPTION_REG – OPTION REGISTER (ADDRESS: 81h OR 181h)

R/W-1	R/W-1	R/W-1	R/W-1	R/W-1	R/W-1	R/W-1	R/W-1
$\overline{\text{RAPU}}$	INTEDG	T0CS	T0SE	PSA	PS2	PS1	PS0
bit 7							bit 0

bit 7 **$\overline{\text{RAPU}}$:** PORTA Pull-up Enable bit
1 = PORTA pull-ups are disabled
0 = PORTA pull-ups are enabled by individual port latch values in WPUA register

bit 6 **INTEDG:** Interrupt Edge Select bit
1 = Interrupt on rising edge of RA2/INT pin
0 = Interrupt on falling edge of RA2/INT pin

bit 5 **T0CS:** TMR0 Clock Source Select bit
1 = Transition on RA2/T0CKI pin
0 = Internal instruction cycle clock (CLKOUT)

bit 4 **T0SE:** TMR0 Source Edge Select bit
1 = Increment on high-to-low transition on RA2/T0CKI pin
0 = Increment on low-to-high transition on RA2/T0CKI pin

bit 3 **PSA:** Prescaler Assignment bit
1 = Prescaler is assigned to the WDT
0 = Prescaler is assigned to the Timer0 module

bit 2-0 **PS<2:0>:** Prescaler Rate Select bits

Bit Value	TMR0 Rate	WDT Rate(1)
000	1 : 2	1 : 1
001	1 : 4	1 : 2
010	1 : 8	1 : 4
011	1 : 16	1 : 8
100	1 : 32	1 : 16
101	1 : 64	1 : 32
110	1 : 128	1 : 64
111	1 : 256	1 : 128

Note 1: A dedicated 16-bit WDT postscaler is available for the PIC16F688. See **Section 11.7 "Watchdog Timer (WDT)"** for more information.

Legend:			
R = Readable bit	W = Writable bit	U = Unimplemented bit, read as '0'	
- n = Value at POR	'1' = Bit is set	'0' = Bit is cleared	x = Bit is unknown

REGISTER 2-3: INTCON – INTERRUPT CONTROL REGISTER (ADDRESS: 0Bh OR 8Bh)

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
GIE	PEIE	T0IE	INTE	RAIE	T0IF	INTF	RAIF
bit 7							bit 0

bit 7 **GIE:** Global Interrupt Enable bit
1 = Enables all unmasked interrupts
0 = Disables all interrupts

bit 6 **PEIE:** Peripheral Interrupt Enable bit
1 = Enables all unmasked peripheral interrupts
0 = Disables all peripheral interrupts

bit 5 **T0IE:** TMR0 Overflow Interrupt Enable bit
1 = Enables the TMR0 interrupt
0 = Disables the TMR0 interrupt

bit 4 **INTE:** RA2/INT External Interrupt Enable bit
1 = Enables the RA2/INT external interrupt
0 = Disables the RA2/INT external interrupt

bit 3 **RAIE:** PORTA Change Interrupt Enable bit(1)
1 = Enables the PORTA change interrupt
0 = Disables the PORTA change interrupt

bit 2 **T0IF:** TMR0 Overflow Interrupt Flag bit(2)
1 = TMR0 register has overflowed (must be cleared in software)
0 = TMR0 register did not overflow

bit 1 **INTF:** RA2/INT External Interrupt Flag bit
1 = The RA2/INT external interrupt occurred (must be cleared in software)
0 = The RA2/INT external interrupt did not occur

bit 0 **RAIF:** PORTA Change Interrupt Flag bit
1 = When at least one of the PORTA <5:0> pins changed state (must be cleared in software)
0 = None of the PORTA <5:0> pins have changed state

Note 1: IOCA register must also be enabled.
2: T0IF bit is set when Timer0 rolls over. Timer0 is unchanged on Reset and should be initialized before clearing T0IF bit.

Legend:			
R = Readable bit	W = Writable bit	U = Unimplemented bit, read as '0'	
- n = Value at POR	'1' = Bit is set	'0' = Bit is cleared	x = Bit is unknown

Source: PIC16F688 Datasheet from Microchip

카운터 모드

카운터모드는 OPTION레지스터의 TOCS비트를 셋팅함으로써 선택이 가능합니다. 이 모드에서는 Timer0모듈이 RA2/T0CKI핀에 들어오는 외부 클럭펄스를 카운트합니다. 카운터는 OPTION레지스터의 T0SE(Timer0 Source Edge)의 설정에 따라 rising 혹은 falling edge시에 클럭펄스를 카운트합니다. T0SE비트를 셋팅시에는 타이머는 RA2/T0CKI 핀에 도착하는 펄스의 falling edge가 검출시에 증가합니다.

T0CKI입력에 최대 클럭 주파수는 내부 클럭의 동기화 요구사항에 의해 제한됩니다. PIC MCU의 각각 머신 사이클(혹은 인스트럭션 사이클)은 4개의 클럭사이클로 이루어져 있고 Q1, Q2, Q3, Q4로 이름지어져 있습니다. T0CKI와 내부 클럭의 동기화는 각각의 머신사이클의 Q2, Q4상의 프리스케일러 출력을 샘플링함으로써 이루어집니다. 그러므로 T0CKI의 외부클럭은 최소한 머신 사이클의 기간(2Tsoc, Tosc는 메인 오실레이터의 period)의 절반에 리지스터-캐페이시터 딜레

이 20ns를 더한 시간에는 high나 low로 남아 있게됩니다. 이것은 T0CKI핀을 통해 들어온 펄스 넓이의 최소값을 결정합니다. 입력 클럭펄스의 최소 시간 단위는 그러므로 4Tsoc + 40ns입니다.

예를 들어, 메인 오실레이터의 주파수가 4MHz(Tosc 0.25μs)이라고 할때, 머신 사이클은 4 x Tsoc = 1μs 길이가 됩니다. 프리스케일러 없이 카운터로 직접 들어가는 외부 클럭신호는 최소한 2Tosc +20ns = 520ns 동안 high를 유지해야하며, 최소한 동일한 시간 동안 low를 유지하여, 총 520 x 2 = 1040ns의 시간단위를 갖습니다. 그러므로, 최대 입력 주파수 제한은 1/1040ns = 961.5KHz입니다. 만약 프리스케일러가 사용이 된다면, PIC16F688의 스펙은 외부 클럭 입력이 최소한 10ns동안 high혹은 low가 되어야 한다고 말합니다. (10ns시 T0CKI핀의 최대 주파수를 50MHz). 더 자세한 사항은 마이크로칩의 타이머 튜토리얼 *Part 1*, *Part 2* 를 참조하십시오.

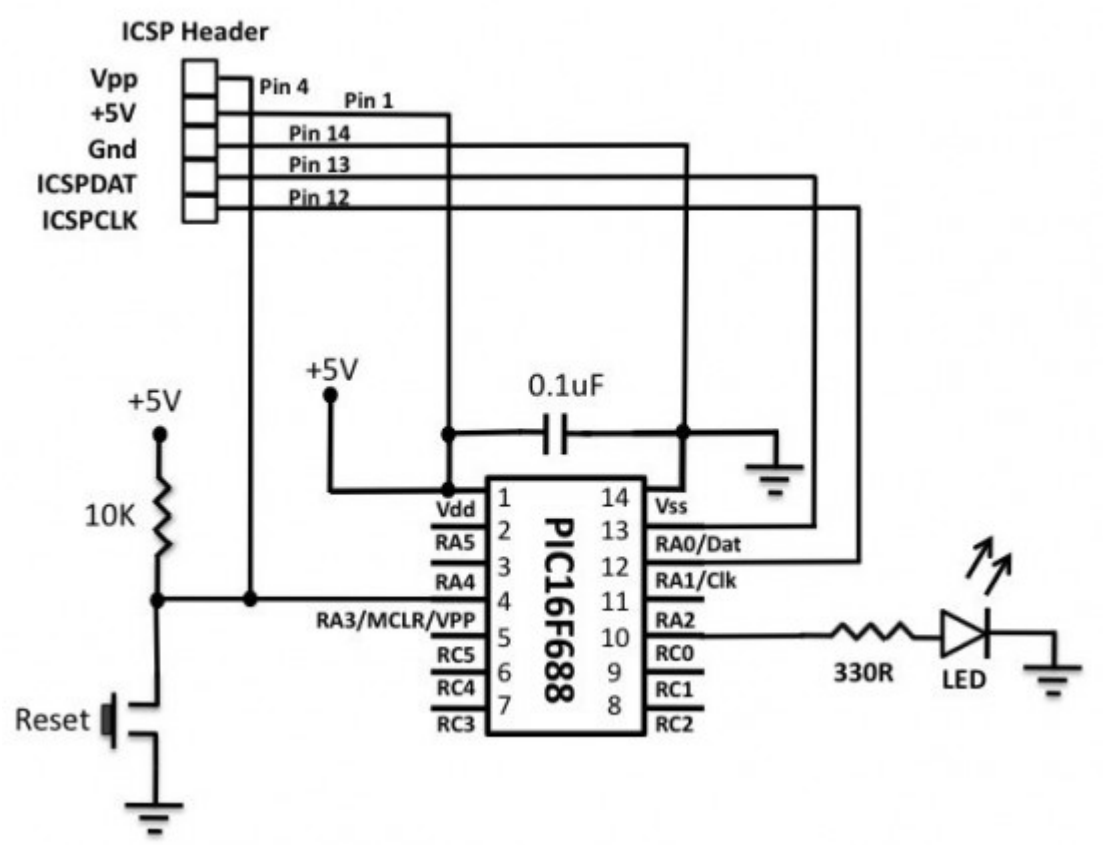
코드 및 셋업

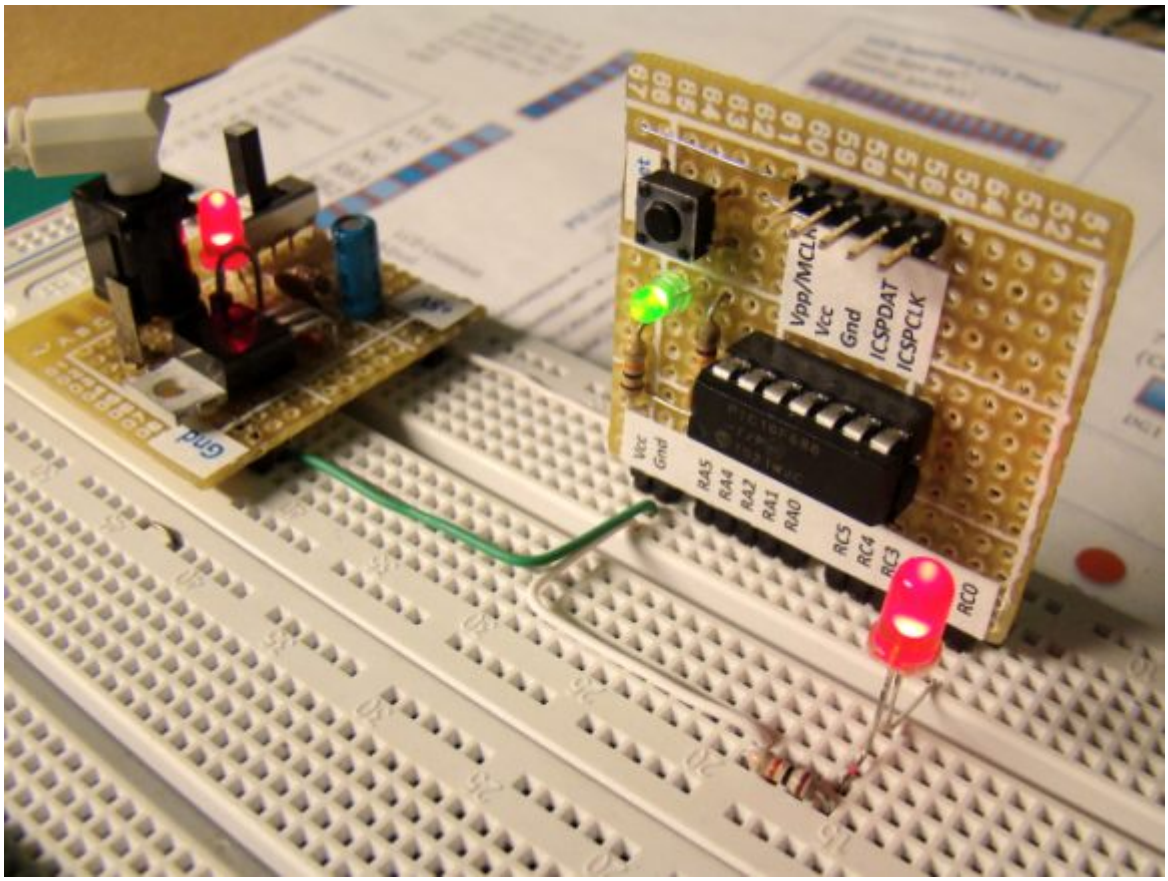
그럼 이제 Timer0모듈 실험을 하여보도록 하겠습니다. 먼저 Timer0모듈을 타이머로 이용하여 대략 1초정도의 딜레이를 만들것입니다. PIC16F688은 4MHz 클럭 주파수로 동작되어 머신사이클은 1 μs가 됩니다. 1초 딜레이를 만들기 위해서는 타이머는 1000000 머신 사이클을 카운트하여야 합니다. 만약 프리스케일러 값을 256을 사용한다면 1초 딜레이를 위해서 3906번만 카운트하면 됩니다. TMR0에 39를 프리로드한다면 256-39 = 217 카운트가 일어나면 오버플로우 되게 됩니다. 1초 딜레이를 만들기 위한 필요한 오버플로우는 3906/217 = 18 번입니다. 즉 TMR0 레지스터(39를 프리로딩)가 18번째 오버플로우 시마다 대략 1초의 시간이 지나게 됩니다. 아래의 코드는 위의 내용을 구현하여 LED를 대략 1초에 한번씩 깜빡이게 만들었습니다.

```
/*

Lab 7: 1 sec timer using TIMER0

Internal Oscillator @ 4MHz, MCLR Enabled, PWRT Enabled, WDT OFF

Copyright @ Rajendra Bhatt

Nov 18, 2010

*/

// Define LED connection

sbit LED at RC0_bit;

unsigned short Num;

// Interrupt service routine
```

```
void interrupt() {
  Num ++;          // Interrupt causes Num to be incremented by 1
  if(Num == 18) {
   LED = ~LED;      // Toggle LED every sec
   Num = 0;
  }
  TMR0 = 39;        // TMR0 returns to its initial value
  INTCON.T0IF = 0;  // Bit T0IF is cleared so that the interrupt could reoccur
}
void main() {
 CMCON0 = 0x07;    // Disable Comparators
 ANSEL = 0x00;     // Disable analog channels
 TRISC = 0x00;     // PORTC O/P
 LED = 0;
 Num = 0;
 OPTION_REG = 0x07; // Prescaler (1:256) is assigned to the timer TMR0
 TMR0 = 39;         // Timer T0 counts from 39 to 255
 INTCON = 0xA0;     // Enable interrupt TMR0 and Global Interrupts
 do {
 // You main programs goes here
 } while(1);  // infinite loop
}
```

아래의 회로는 이전 게시물 "LED깜빡이기"에 나온 회로와 동일한 회로입니다. LED는 RC0핀을 통해 동작합니다.



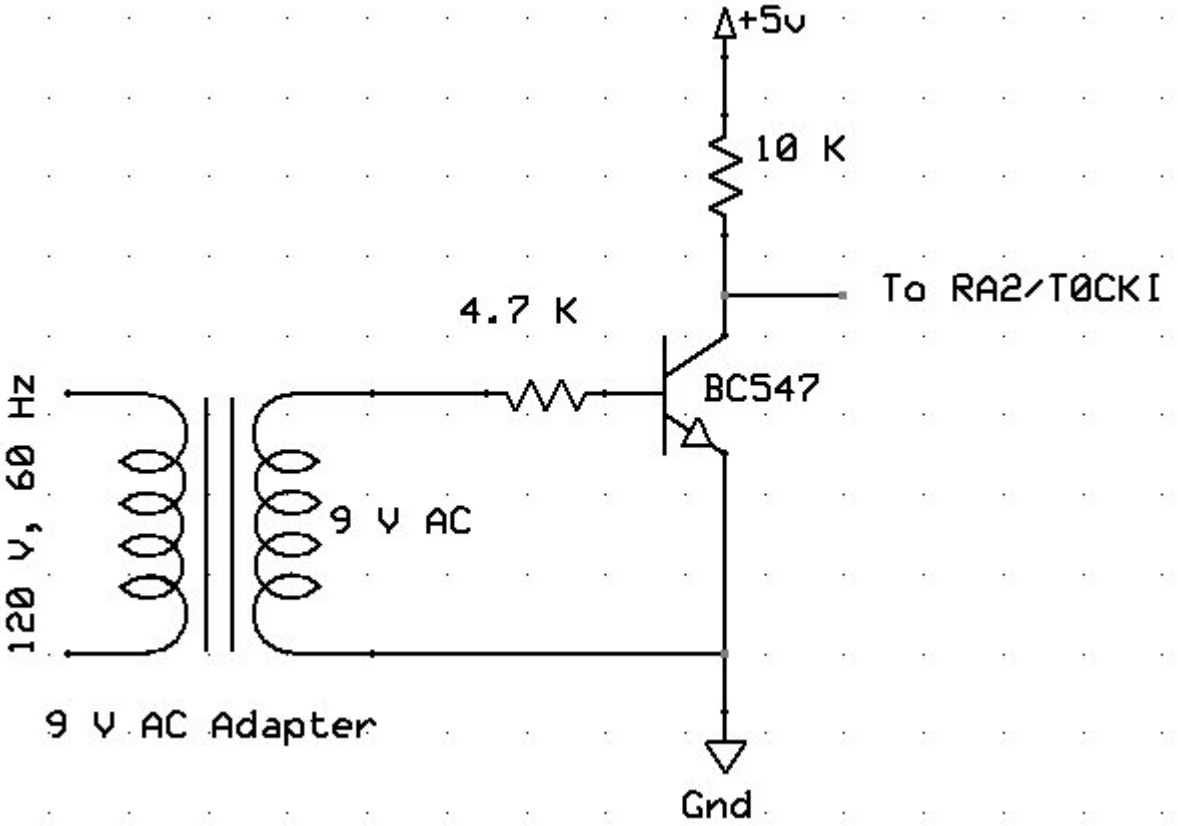


Timer0모듈을 사용하여 LED를 깜빡이게 하는 회로셋업

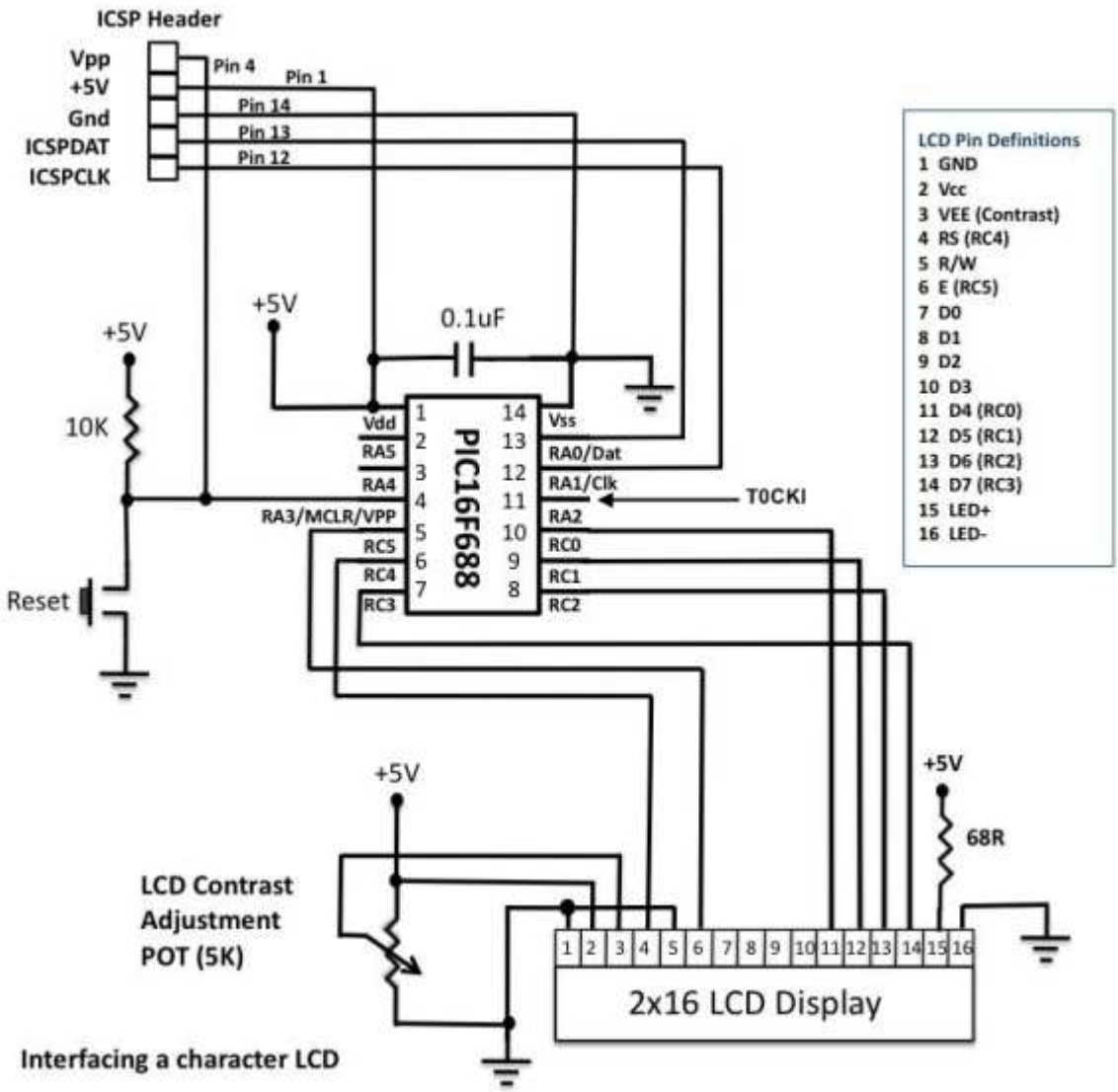
알림: 우리가 만든 1초 딜레이는 아주 정확하지는 않습니다. 그 이유는 다음과 같습니다.

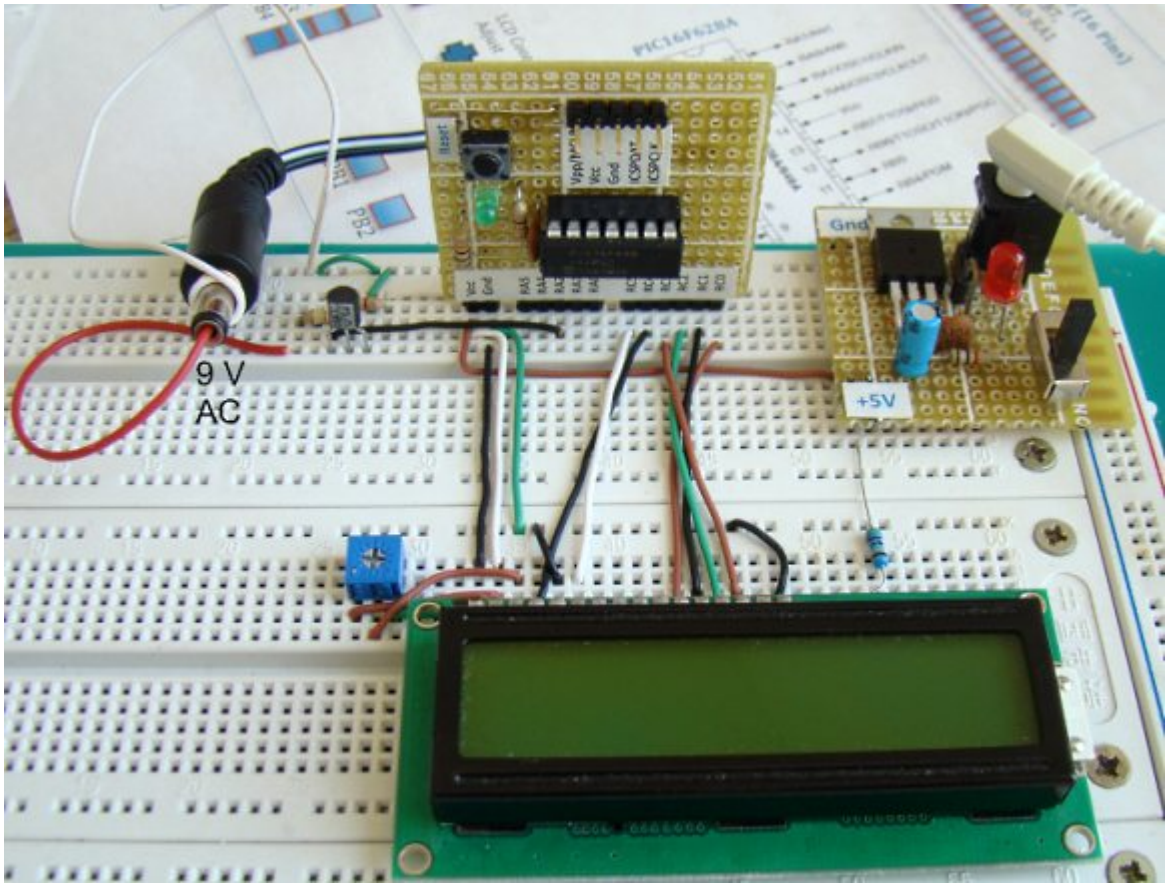
- 이 예제에서, PIC16F688 MCU는 4MHz 내부 RC오실레이터에 의해 동작합니다. 이 오실레이터는 아주 정확하거나 안정적이지는 않습니다. 정확함을 위해서는 반드시 외부 크리스탈 오실레이터를 사용하여야 합니다.
- 위의 프로그램은 하이레벨 프로그래밍 언어로 작성되었는데, 아주 정교한 하드웨어 타이머를 위해서는 어셈블리어로 프로그램을 작성하는 것이 좋습니다. TMR0 read/write연산에 영향을 미치는 어떠한 연산이라도 추가적인 딜레이로 계산되어야 합니다.

다음으로, Timer0 모듈을 RA2/T0CKI핀을 통해 외부 클럭펄스를 카운트 할수 있는 카운터로 사용하여 보겠습니다. 외부 클럭 소스는 AC전원을 이용할 것입니다. AC전원을 120V, 60HZ sinewave 신호라고 할때, 이 신호는 AC어댑터를 이용하여 9V, 60Hz 신호로 먼저 다운시킵니다. 이 신호를 T0CKI핀에 적용시키기 전에, 신호는 반드시 rectified되어야 하고 피크 전압은 5V로 짤라놓아야 합니다. 아래의 회로는 9V, 60Hz AC전압을 T0CKI입력에 적합하게 약 +5V의 스퀘어웨이브 형태로 변환할 것입니다.



출력되는 스퀘어웨이브는 PIC16F688의 RA2/T0CKI핀에 연결됩니다. TMR0는 0부터 시작하여 들어오는 펄스를 1초동안 카운트하고 그 결과를 LCD스크린에 디스플레이합니다. 1초동안 들어오는 펄스의 숫자는 들어오는 신호의 주파수를 의미하며 60Hz가 될 것입니다. 회로도는 아래와 같습니다.


ICSP Header
Vpp
+5V
Gnd
ICSPDAT
ICSPCLK
Pin 4
Pin 1
Pin 14
Pin 13
Pin 12
LCD Pin Definitions
1 GND
2 Vcc
3 VEE (Contrast)
4 RS (RC4)
5 R/W
6 E (RC5)
7 D0
8 D1
9 D2
10 D3
11 D4 (RC0)
12 D5 (RC1)
13 D6 (RC2)
14 D7 (RC3)
15 LED+
16 LED-
+5V
0.1uF
+5V
10K
Vdd
RA5
RA4
RA3/MCLR/VPP
RC5
RC4
RC3
PIC16F688
Vss
RA0/Dat
RA1/Clk
T0CKI
RA2
RC0
RC1
RC2
Reset
+5V
+5V
68R
LCD Contrast
Adjustment
POT (5K)
2x16 LCD Display
Interfacing a character LCD



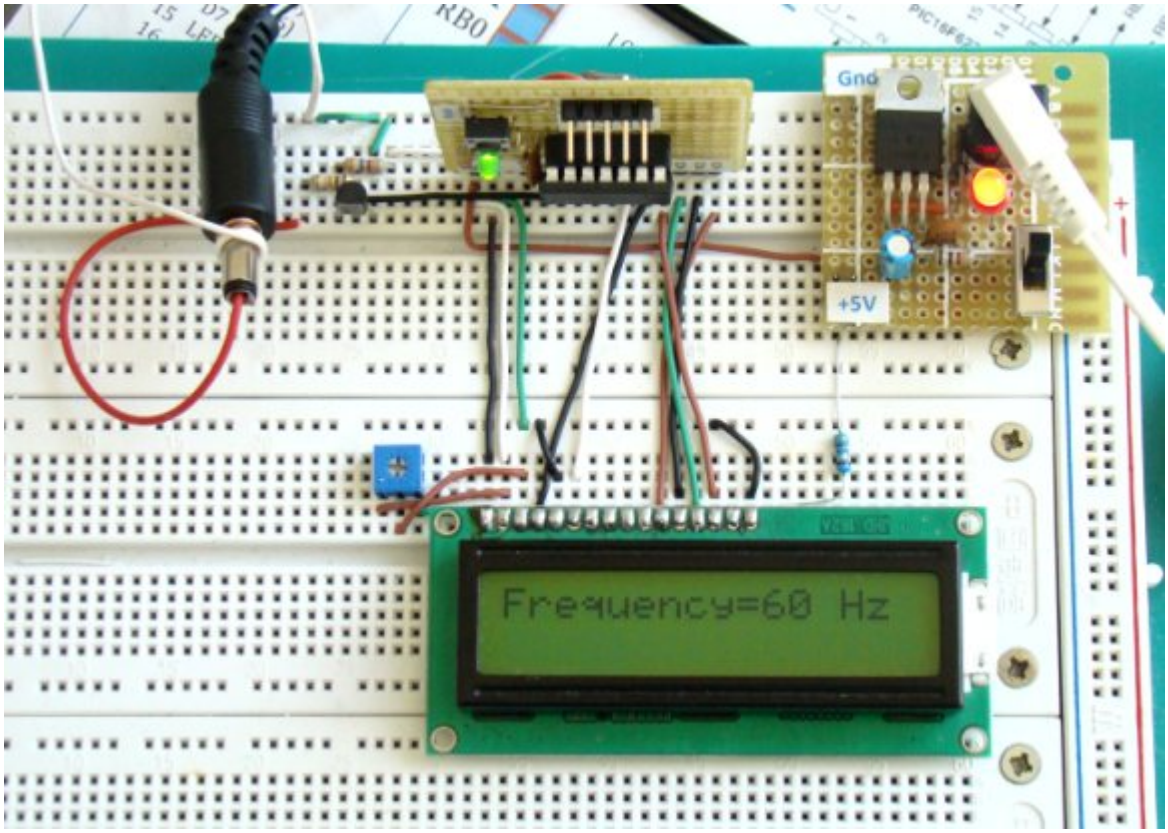
브레드보드상에 카운터 회로 셋업

프로그래밍 시에 ANSEL, CMCON0, OPTION 레지스터를 초기화하여야 합니다. 타이머를 카운터로 동작시키기 위해 OPTION레지스터의 T0CS비트를 셋팅하세요.

```
/*

Lab 7: Timer0 as a counter

Internal Oscillator @ 4MHz, MCLR Enabled, PWRT Enabled, WDT OFF

Copyright @ Rajendra Bhatt

November 18, 2010

*/

// LCD module connections

sbit LCD_RS at RC4_bit;
```

```
sbit LCD_EN at RC5_bit;

sbit LCD_D4 at RC0_bit;

sbit LCD_D5 at RC1_bit;

sbit LCD_D6 at RC2_bit;

sbit LCD_D7 at RC3_bit;

sbit LCD_RS_Direction at TRISC4_bit;

sbit LCD_EN_Direction at TRISC5_bit;

sbit LCD_D4_Direction at TRISC0_bit;

sbit LCD_D5_Direction at TRISC1_bit;

sbit LCD_D6_Direction at TRISC2_bit;

sbit LCD_D7_Direction at TRISC3_bit;

// End LCD module connections

// Define Messages

char message1[] = "Frequency=   Hz";

char *freq = "00";

void Display_Freq(unsigned int freq2write) {

 freq[0] = (freq2write/10)%10 + 48;    // Extract tens digit

 freq[1] =  freq2write%10     + 48;    // Extract ones digit

 // Display Frequency on LCD

 Lcd_Out(1, 11, freq);

}

void main() {
```

```
 CMCON0 = 0x07;    // Disable Comparators

 ANSEL = 0x00;     // Disable analog channels

 TRISC = 0x00;     // PORTC O/P

 TRISA = 0b00001100; // RA2/T0CKI input, RA3 is I/P only

 OPTION_REG = 0b00101000; // Prescaler (1:1), TOCS =1 for counter mode

 Lcd_Init();                // Initialize LCD

 Lcd_Cmd(_LCD_CLEAR);       // CLEAR display

 Lcd_Cmd(_LCD_CURSOR_OFF);  // Cursor off

 Lcd_Out(1,1,message1);     // Write message1 in 1st row

 do {

  TMR0=0;

  Delay_ms(1000);  // Delay 1 Sec

  Display_Freq(TMR0);

 } while(1);  // Infinite loop

}
```



가치창조기술 | www.vctec.co.kr

[PIC 마이컴 실험하기] 8. 비동기 시리얼 통신

마이컴 실험실

2011/10/10 13:37 http://blog.naver.com/ubicomputing/150121031176

PIC16F628A는 내장 USART(Universal Synchronous Asynchronous Receiver Transmitter)하드웨어를 가지고 있어 메모리칩, LCD, PC 등과 같은 다양한 시리얼 장치와 통신을 할수 있게 합니다. USART는 동기(전송부와 수신부간 클럭동기화) 및 비동기(클럭동기화 필요없음) 두가지 작동모드가 있습니다. 비동기모드가 더 많이 사용이 되므로, 이번 게시물에서는 PIC MCU와 PC간 양방향 시리얼 데이터 링크를 셋업하여 보겠습니다.

선행이론

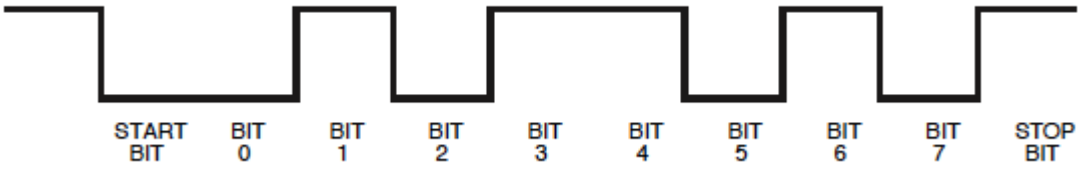
시리얼 통신은 사용가능한 I/O핀이 많지 않은 관계로 mcu기반의 시스템에서 많이 사용됩니다. 장거리 통신에 비해 시리얼 데이터 전송은 좀더 단순하고 비용효과적인데 데이터링크를 위한 필요한 하드웨어 연결을 Tx, Rx, Gnd의 세개로 줄일 수 있기 때문입니다.

시리얼 통신은 동기식 및 비동기식 두가지 타입이 있습니다. 시리얼 데이터링크에 있어 핵심은 송신부와 수신부간의 동기화를 유지하는 것입니다. 비동기식은 start bit, stop bit를 이용하여 데이타를 동기화 합니다. 각각의 바이트는 start bit + 바이트 + (패러티 비트) + stop bit 형식으로 프레임이 구성되어 전송됩니다. 송신부와 수신부는 같은 data rate, data bits 수, stop bit수를 사용하도록 초기화 되어야 합니다.

대기상태에서, 전송출력은 로직하이입니다. 트랜스미터가 문자 바이트를 송실할 준비가 되었을때, 수신부에 1클럭 기간동안 전송라인을 low로 만듬으로 신호를 보냅니다. 이것이 start bit이며 이 비트는 리시버에게 프레이임이 바로 전송될 것임을 알려줍니다. 리시버는 적용된 프로토콜에 따라 예상되는 캐릭터 비트를 읽는데, 트랜스미터에 의해 전송라인이 로직하이(한개나 한개이상의 스탑비트)로 변경될때가지 읽습니다. 전체 프로세스는 트랜스미터가 문자바이트를 보내야 할 때마다 반복됩니다. 이런 형태의 시리얼 데이터 전송은 비동기라 불립니다. 왜냐하면 데이터를 보낼때마다 스타트 비트를 이용하여 리시버와 트랜시버를 재동기화하기 때문입니다. 하지만 각각의 프레임 내부에서는 송수신부는 동기화된 상태입니다.

반면에 동기식 시리얼 통신은 문자들을 블럭단위로 전송하는데 위와 같은 framing bits를 사용하지 않습니다. 트랜스미터와 리시버는 별도의 클럭라인으로 동기화 됩니다. 몇몇 경우에서는 클럭신호에 전송되는 문자들이 들어 있기도 합니다.
이 두가지 시리얼 통신에 있어서 데이터를 주고 받는 rate를 baud rate라고 합니다.

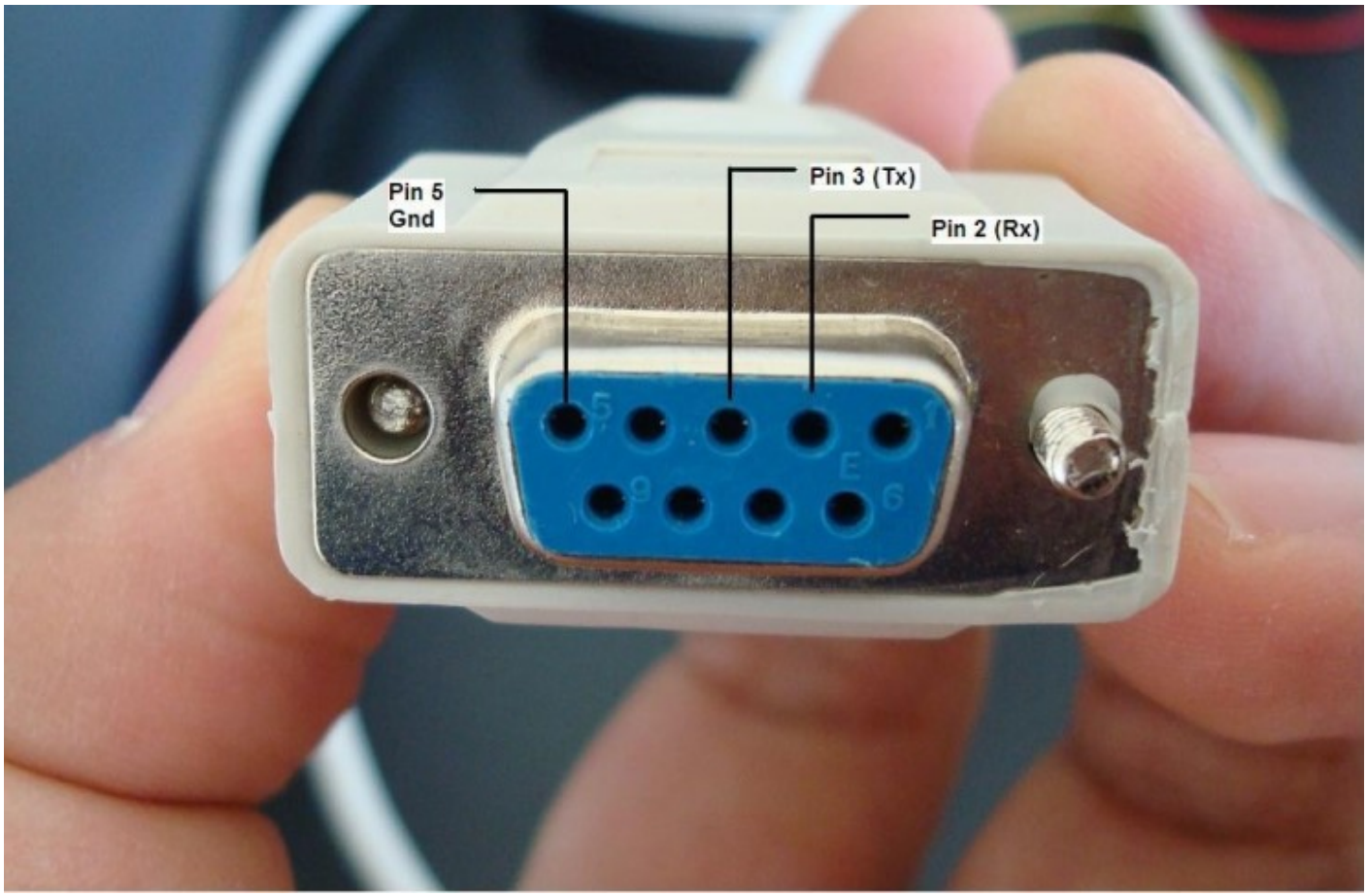
PIC16F628A MCU에 있는 USART모듈은 동기/비동기 시리얼통신을 모두 지원합니다만 비동기 시리얼 통신에 최적화되어 있습니다. 비동기 모드에서 RB2는 TX 출력처럼 동작하며 RB1은 RX입력처럼 동작합니다. 시리얼 데이타 한 바이트는 10비트 스트링으로써 전송이 됩니다. (시작비트 + 8개의 데이터 비트 + 스탑 비트) 아래의 그림을 참조하세요.



PC의 시리얼 포트는 시리얼 통신을 위해 RS232-C 표준을 사용합니다. 이 표준은 시리얼 통신의 전기적, 기계적, 기능적 신호와 절차에 대해 정의하고 있습니다. RS232-C의 로직하이는 -3V~-15V(보통 -12V) 사이의 전압신호이며 로직로우는 +3V~+15V(보통 +12V)사이의 전압 신호입니다. PIC마이크로컨트롤러와 같지 않게 RS232-C HIGH는 음의 전압이고 LOW는 양의 전압입니다. 아래의 테이블은 RS232-C의 25핀, 9핀, RJ-45 커넥터용 표준 연결을 보여줍니다. 다양한 핀 인터페이스가 있지만 여기서는 TX, RX, GND신호를 이용하여 최소한의 시리얼 인터페이스를 구현하여 보도록 하겠습니다.

Definition of Common RS-232-C Lines

	CONNECTOR			CODE	
DB-25	DB-9	RJ-45	FUNCTION	NAME	DIRECTION
1		4	Ground	G	
2	3	6	Transmit data	TXD	Output
3	2	5	Receive data	RXD	Input
4	7	8	Request to send	RTS	Output
5	8	7	Clear to send	CTS	Input
6	6		Data set ready	DSR	Input
7	5		Chassis ground	G	
8	1	2	Carrier detect	CD	
20	4	3	Data terminal ready	DTR	Output
22	9	1	Ring indicator	RI	Input



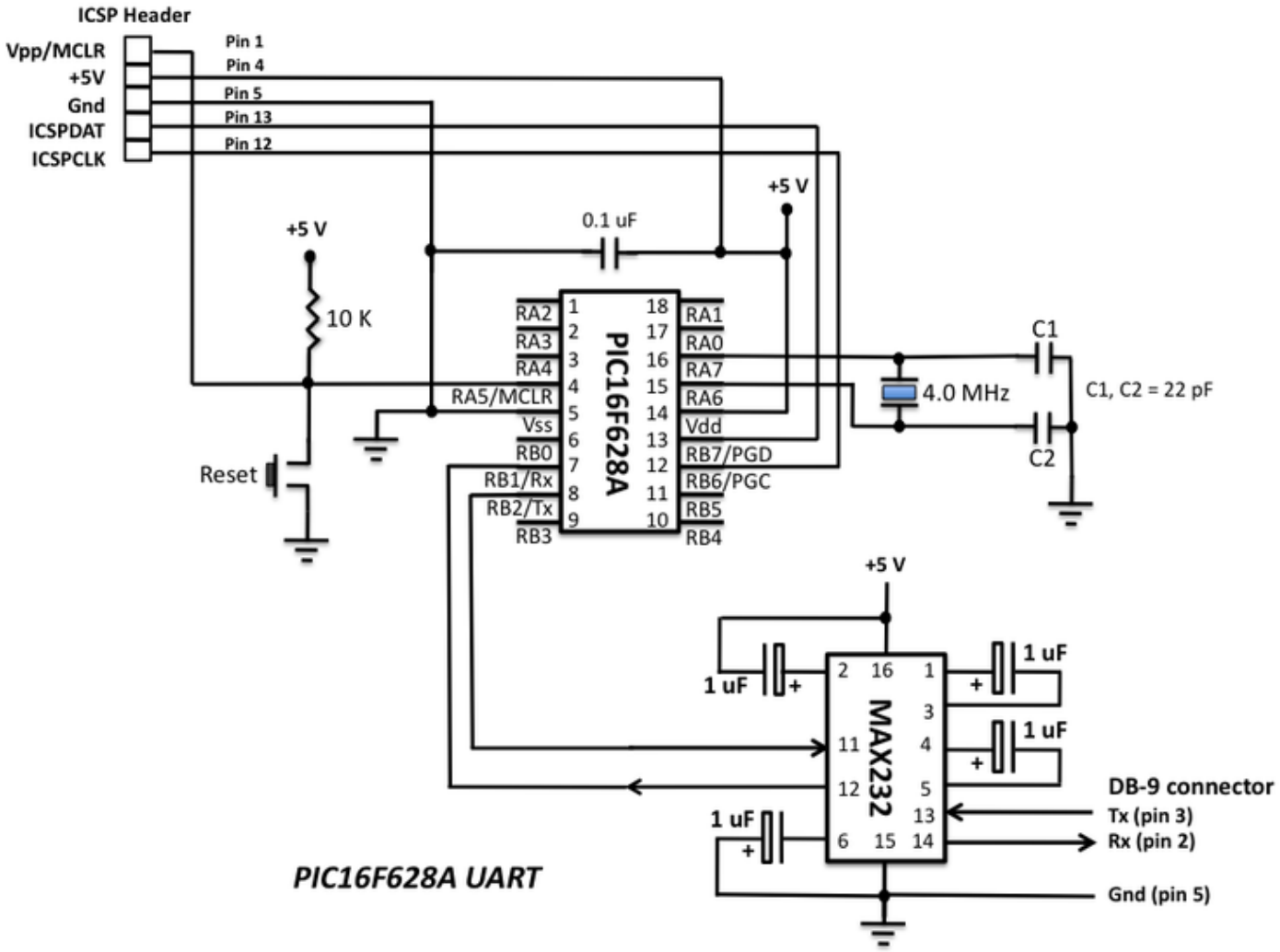
DB9 암커넥터 RS232신호 핀 모습

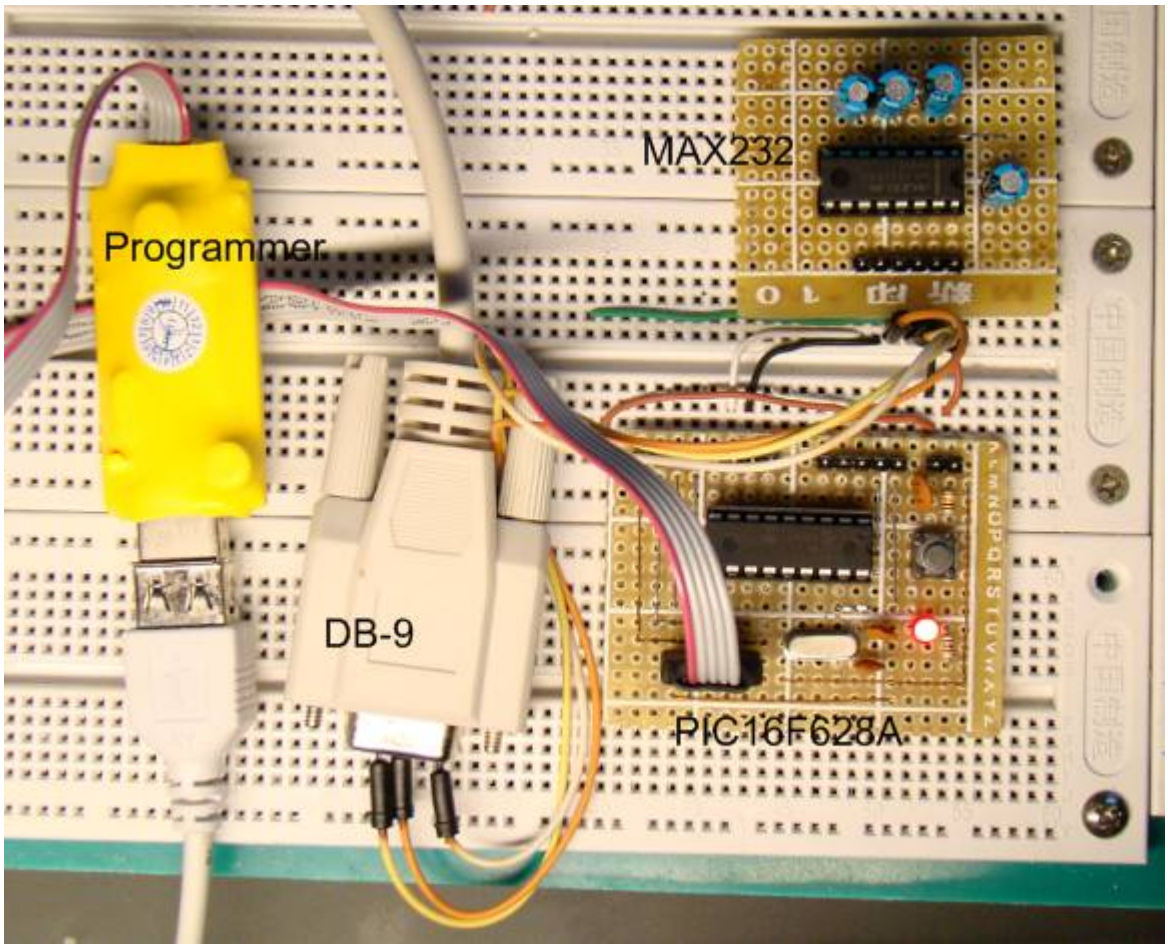
PC의 RS232-C 시리얼 포트와 통신하기 위해 비동기식 모드를 사용할 것입니다. PIC16F628A는 이미 비동기식 시리얼 통신을 지원하는 하드웨어(USART)를 가지고 있기때문에 필요한 모든것은 PIC의 TTL신호를 RS232-C 레벨로 변환하여 주는 외부 레벨 쉬프터입니다. 이것은 Maxim사의 MAX232 칩으로 구현할수 있습니다. 이 칩은 RS232-C통신에 필요한 +-12V를 생성하기 위해서 내장 충전펌프를 가지고 있고 이를 위해 몇개의 외장 캐패이시터를 필요로합니다.

PC의 시리얼 포트와 바이트 데이터를 주고 받기 위한 간단한 방법은 윈도우의 하이퍼터미널을 사용하는 것입니다. 하이퍼터미널을 실행하고 시리얼포트를 선택하고, baud rate, bit수, parity bit,등을 셋팅합니다. 하이퍼터미널이 시리얼 포트에 연결되면 시리얼포트로 보내진 데이터는 하이퍼터미널 화면에 나타나게 됩니다.

회로셋업

본 실험을 위한 회로는 아래와 같습니다. PIC16F628A브레드모듈과 MAX232칩을 이용한 레벨쉬프터를 가지고 있습니다. MAX232칩은 내부 차징펌프를 위해 네개의 외장 캐패이시터(각 1 uF)를 필요로 합니다. 좀더 자세한 사항은 MAX232칩의 datasheet를 참고하세요. PC쪽에는 9핀 커넥터를 통해 TX, RX, GND의 3개의 라인이 COM포트에 연결되었습니다.

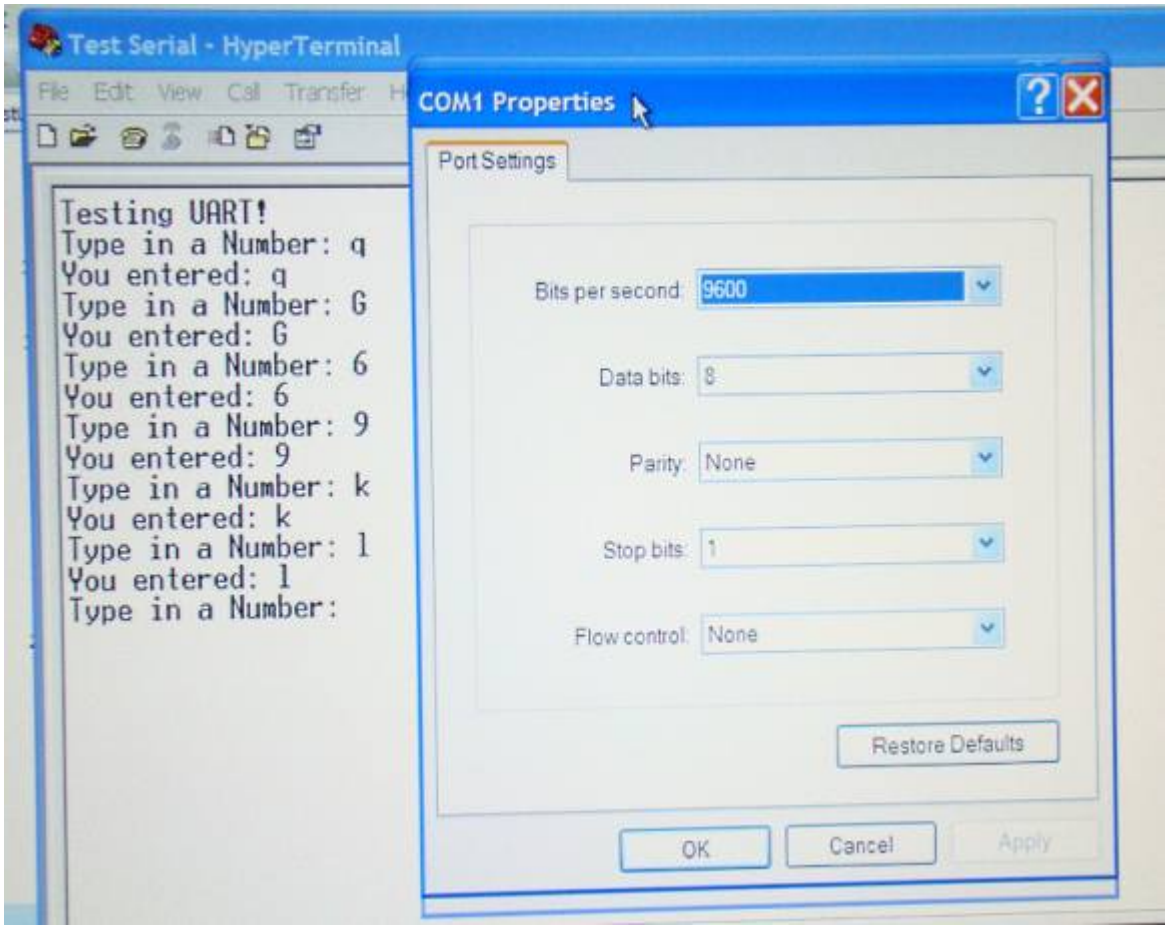

ICSP Header
Vpp/MCLR
+5V
Gnd
ICSPDAT
ICSPCLK
Pin 1
Pin 4
Pin 5
Pin 13
Pin 12
+5 V
0.1 uF
+5 V
10 K
Reset
RA2
RA3
RA4
RA5/MCLR
Vss
RB0
RB1/Rx
RB2/Tx
RB3
PIC16F628A
RA1
RA0
RA7
RA6
Vdd
RB7/PGD
RB6/PGC
RB5
RB4
C1
4.0 MHz
C1, C2 = 22 pF
C2
+5 V
1 uF
1 uF
1 uF
1 uF
MAX232
DB-9 connector
Tx (pin 3)
Rx (pin 2)
Gnd (pin 5)
PIC16F628A UART



소프트웨어

소스코드는 mikroC 컴파일러로 작성되었습니다. mikroC 컴파일러는 풀듀플렉스모드의 비동기시리얼통신용 UART라이브러리를 제공합니다. 이것은 프로그래밍을 매우 쉽게 합니다. 예를들어, PIC16F628A의 하드웨어 UART모듈을 9600baud rate로 초기화하고 싶을때, 단지 UART1_Init(9600)으로 코딩하기만 하면 됩니다. 여기에 나와있는 예제코드는 PIC16F628A와 PC간에 두개의 비동기 시리얼 링크를 설정하고 마이크로컨트롤러가 'Type in a Number'라는 메세지를 전송합니다. 이 메세지는 하이퍼터미널 윈도우에 디스플레이되고 키보드를 이용하여 아무 문자나 입력합니다. 이문자는 com포트를 통해 mcu로 전송될 것입니다. PIC MCU는 다시 이것을 읽어 PC로 보낼 것입니다. 하이퍼터미널의 셋팅은 다음과 같아야 합니다.

Bits per second: 9600, **Data Bits**: 8, **Parity**: None, **Stop bits**: 1, **Flow control**: None.

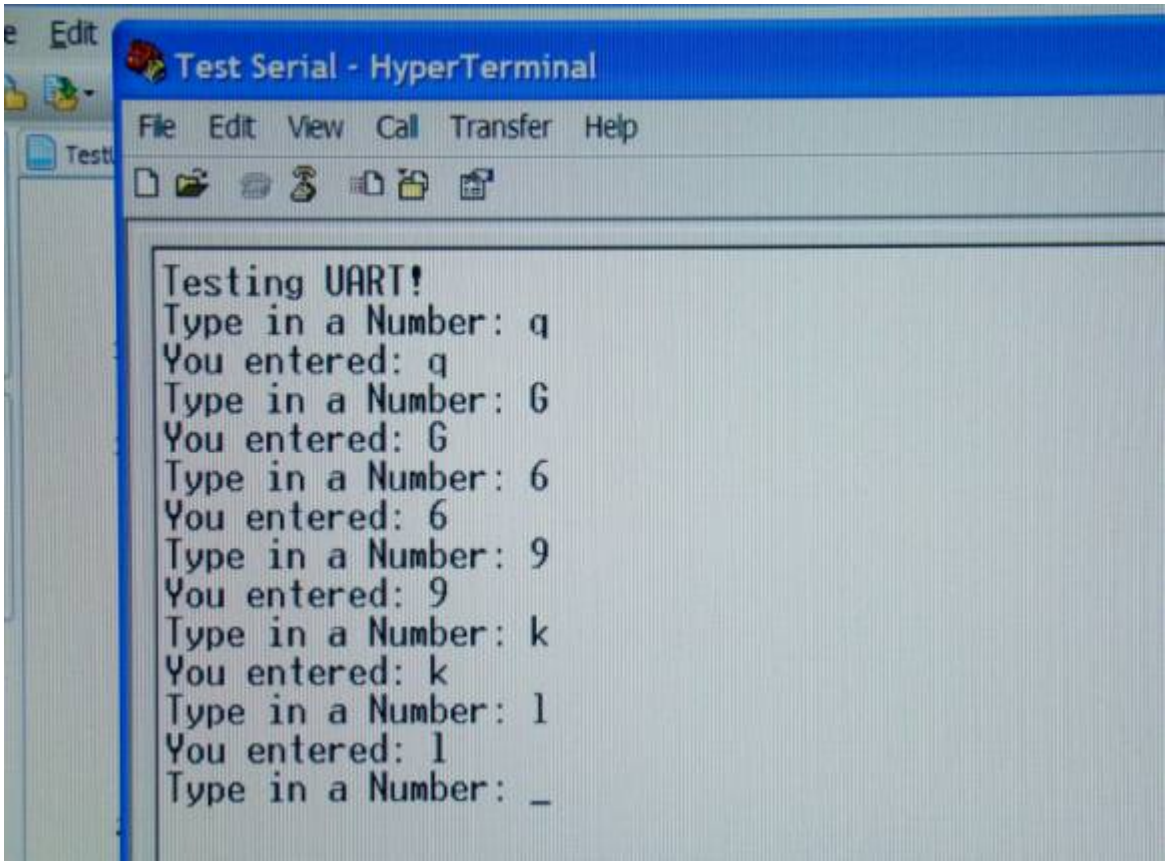



```
/*
Lab 8: Hardware UART
MCU: PIC16F628A
External 4MHz Crystal, MCLR Enabled, PWRT Enabled, WDT OFF
Copyright @ Rajendra Bhatt
Dec 12, 2010
*/

void newline(){
 UART1_Write(13); // Carriage Return
 UART1_Write(10); // Line Feed
}

void main() {
 unsigned char MyError, Temp;
 CMCON = 7;       // Disable Comparators
 TRISB = 0b00000010;
```

```
 UART1_Init(9600);
 Delay_ms(100);
 UART1_Write_Text("Testing UART! ");
 newline();

 do {
  UART1_Write_Text("Type in a Number: ");
  while(!UART1_Data_Ready());
  Temp = UART1_Read();
  newline();
  UART1_Write_Text("You entered: ");
  UART1_Write(Temp);
  newline();
 } while(1);
}  // End main()
```

결과

하이퍼터미널에 나온 결과화면입니다.



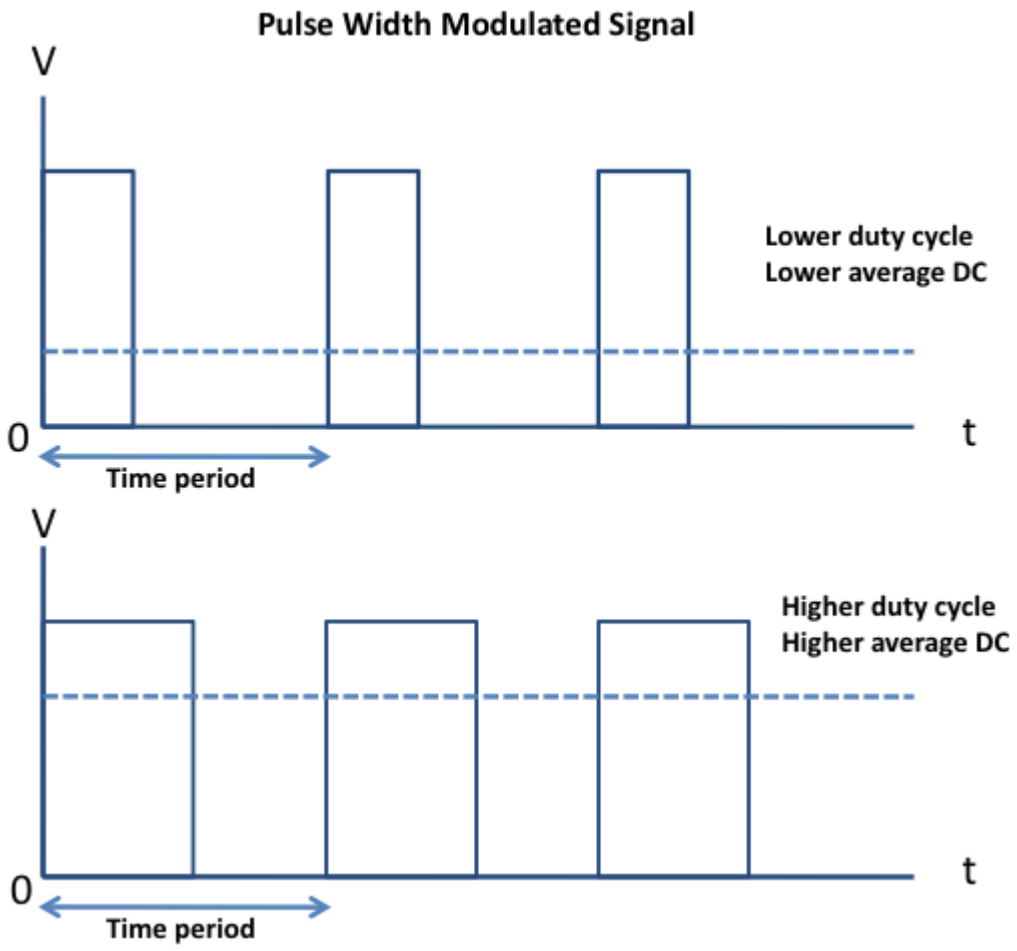
가치창조기술 | www.vctec.co.kr

[PIC 마이컴 실험하기] 9. PIC CCP모듈을 이용한 PWM(Pulse Width Modulation)

마이컴 실험실

2011/10/11 18:26 http://blog.naver.com/ubicomputing/150121143394

Pulse width modulation (PWM)은 ON/OFF 디지털 신호를 이용하여 전달되는 전력의 양을 조절하는 기술입니다. 신호가 ON되어 있는 기간의 조각을 duty cycle이라고 합니다. 신호의 평균 DC값은 이 듀티사이클에 따라 변화됩니다. 듀티사이클은 신호가 항상 꺼져있는 0인부분과 신호가 항상 on되어 있는 사이 어느 값이나 가질수 있습니다. 만약 on상태의 신호가 +5v를 가지고 off일시 0v을 가진다면 신호의 듀티 사이클을 조절하여 0에서 5v사이의 어떤 전압도 시뮬레이션이 가능합니다. 이 방법은 흔히 DC모터의 속도나 램프의 밝기를 조절하는데 사용이 됩니다. 이번 게시물에서는 PIC16F688A를 이용하여 어떻게 PWM신호를 생성하고 이를 이용하여 LED의 밝기를 조정하는지 살펴보도록 하겠습니다.
PIC16F688A는 PWM신호를 생성하기 위해 CCP(Capture/Compare/PWM)라는 내장 하드웨어 모듈을 가지고 있습니다.



선행이론

PIC16F628A의 Capture/Compare/PWM (CCP) 모듈은 매우 다재다능합니다. Capture, Compare 기능은 16비트 TMR1과 잘 통합되었으며, PWM기능은 3번째 타이머 8비트 TMR2를 사용합니다. CCP모듈은 CCPR1L, CCPR1H라고 불리는 두개의 8비트 레지스터를 가지고 있으며 두개를 합쳐 16비트 레지스터를 만드는데 이는 capture, compare, 혹은 PWM스트림의 듀티 사이클을 만드는데 사용합니다. CCP모듈은 CCP1CON레지스터에 의해 제어됩니다.

CCP1CON – CCP OPERATION REGISTER (ADDRESS: 17h)

U-0	U-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
—	—	CCP1X	CCP1Y	CCP1M3	CCP1M2	CCP1M1	CCP1M0
bit 7							bit 0

bit 7-6 **Unimplemented**: Read as '0'

bit 5-4 **CCP1X:CCP1Y**: PWM Least Significant bits

Capture Mode
Unused

Compare Mode
Unused

PWM Mode
These bits are the two LSbs of the PWM duty cycle. The eight MSbs are found in CCPRxL.

bit 3-0 **CCP1M<3:0>**: CCPx Mode Select bits

0000 = Capture/Compare/PWM off (resets CCP1 module)
0100 = Capture mode, every falling edge
0101 = Capture mode, every rising edge
0110 = Capture mode, every 4th rising edge
0111 = Capture mode, every 16th rising edge
1000 = Compare mode, set output on match (CCP1IF bit is set)
1001 = Compare mode, clear output on match (CCP1IF bit is set)
1010 = Compare mode, generate software interrupt on match (CCP1IF bit is set, CCP1 pin is unaffected)
1011 = Compare mode, trigger special event (CCP1IF bit is set; CCP1 resets TMR1
11xx = PWM mode

Legend:			
R = Readable bit	W = Writable bit	U = Unimplemented bit, read as '0'	
-n = Value at POR	'1' = Bit is set	'0' = Bit is cleared	x = Bit is unknown

Capture 모드에서 RB3/CCP1핀에 이벤트가 발생했을때 CCPR1L과 CCPR1H 레지스터는 TMR1레지스터의 16비트 값을 기록합니다. 이벤트는 다음과 같을때 발생가능합니다.

- falling edge시 마다
- rising edge시 마다
- 매 4번째 rising edge시 마다
- 매 16번째 rising edge시 마다

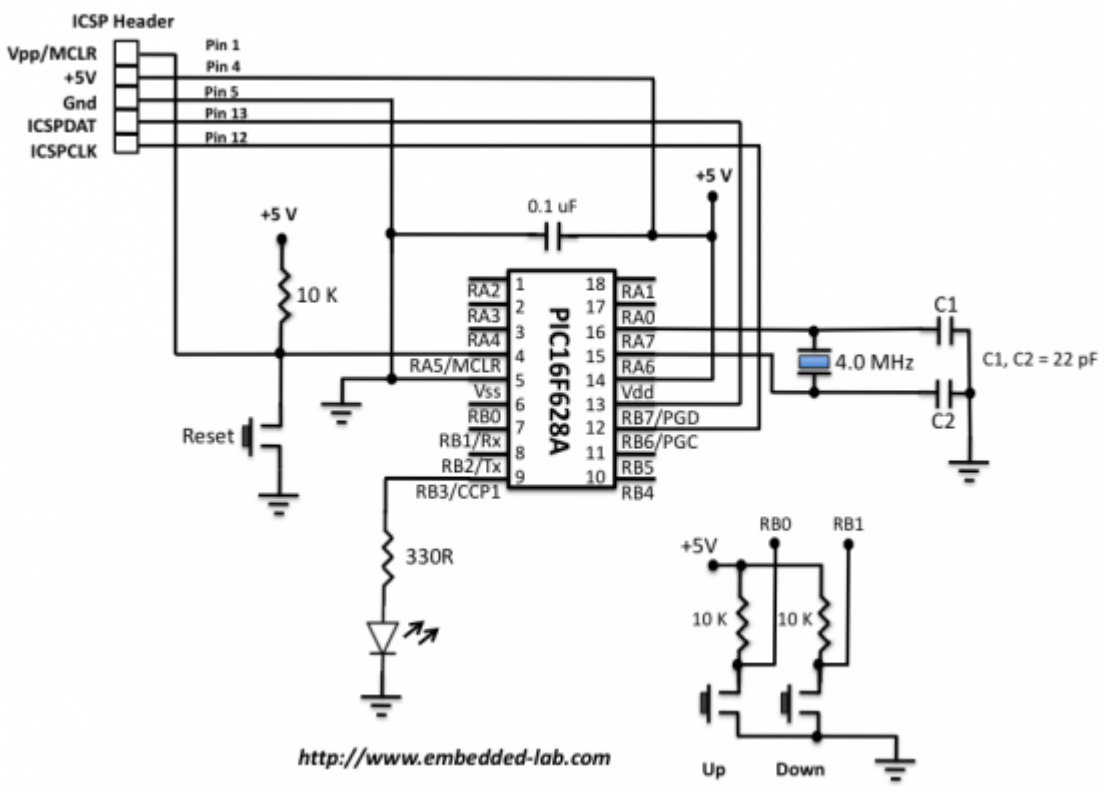
RB3/CCP1핀은 반드시 입력으로 설정되어야 하고 Timer1은 반드시 타이머 모드이거나 동기화된 카운터 모드이어야 합니다. Capture 이벤트도 인터럽트를 트리거 할수 있습니다.

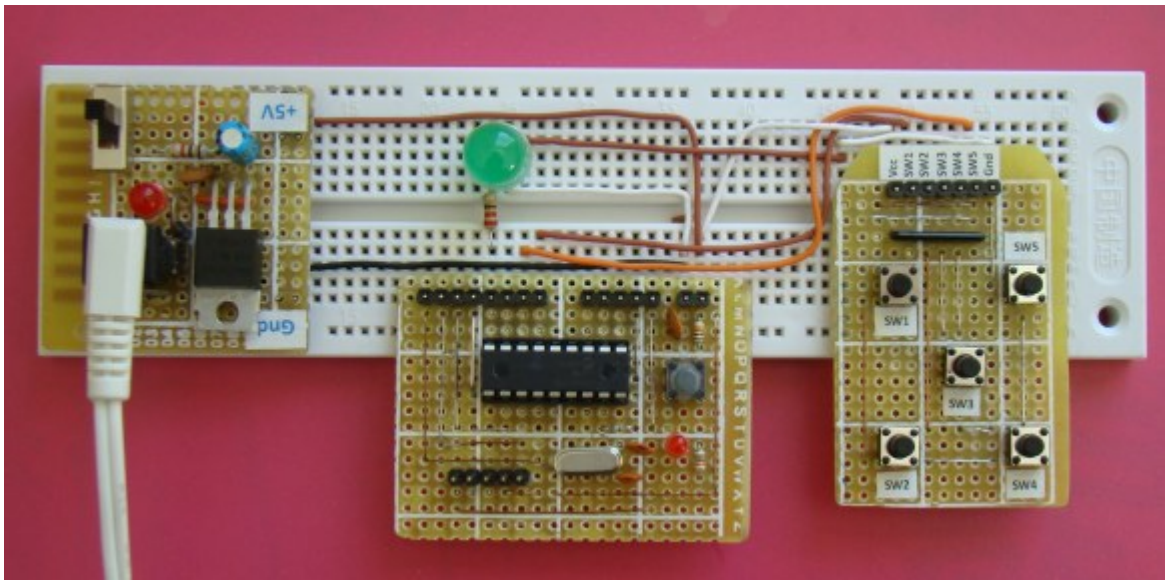
Compare모드에서 TMR1레지스터의 값은 CCPR1H와 CCPR1L로 만들어진 16비트 값과 계속 비교가 됩니다. 일치하는 것이 발견시 RB3/CCP1핀이 CCP1CON레지스터의 셋팅에 따라 HIGH나 LOW 혹은 변화없음 상태가 됩니다. RB3/CCP1핀은 해당 TRISB비트를 클리어하여 출력으로 설정하여야 합니다.

PWM모드에서 RB3/CCP1핀은 10비트 분해능의 주기적인 디지털 웨이브폼 출력을 낼수 있으며 출력넓이와 듀티 사이클을 프로그래밍이 가능합니다. PWM모드에서 동작시키기 위해서는 CCP1 핀은 반드시 출력으로 설정되어야 합니다. 생성되는 웨이브폼의 듀티사이클은 10비트 값으로 상위 8비트는 CCPR1L레지스터에 하위 두비트는 CCP1CON레지스터의 비트5번과 4번에 저장이 됩니다.

회로

이 실험을 위한 셋업은 매우 간단합니다. 입력을 제공하기 위해서 두개의 택트 스위치를 RB0와 RB1핀에 연결합니다. LED는 330R 전류제한 저항을 통해서 RB3/CCP1핀으로 부터 나오는 PWM출력에 의해 동작됩니다. 출력PWM신호의 듀티 사이클은 두개의 택트 스위치 입력에 따라 증가되거나 감소됩니다. LED의 밝기 역시 증가되거나 감소됩니다.





브레드보드에 회로셋업하기

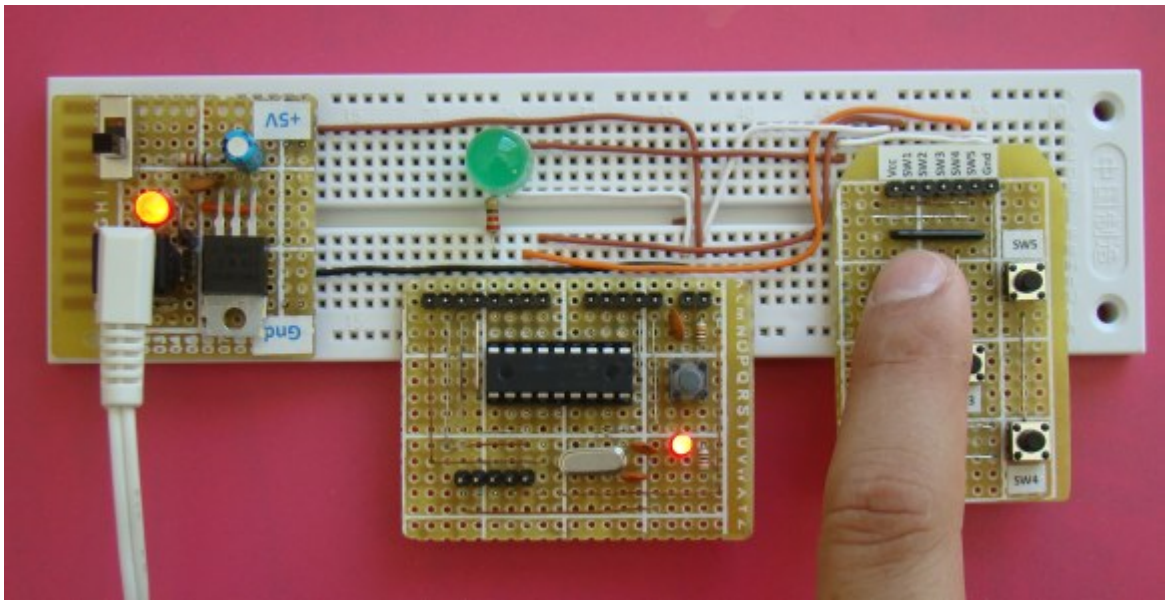
mikroC 컴파일러는 CCP모듈을 이용하여 PWM 동작을 제어하는 4개의 라이브러리 루틴을 제공합니다. PWM1_Init(**const long** frequency), PWM1_Set_Duty(**unsigned short** duty_ratio), PWM1_Start(**void**), PWM_Stop(**void**)인데 컴파일러 매뉴얼에서 디테일한 내용을 찾아볼수 있습니다. 아래의 프로그램은 0에서 250까지 범위에서 듀티사이클을 25씩 변화시켜 10개의 서로다른 LED밝기를 만드는 프로그램입니다. 듀티사이클은 UP, DOWN버튼을 누르면 변화됩니다.

```
/*
Lab 9: Pulse Width Modulation
Description: CCP module generating a PWM signal
MCU: PIC16F628A
Oscillator: XT, 4.0 MHz, MCLR Enabled
*/
sbit UP at RB0_bit;
sbit DOWN at RB1_bit;
unsigned short new_DC, current_DC;

void debounce(){
  Delay_ms(300);
}

void main() {
CMCON = 0x07; // Disable comparators
```

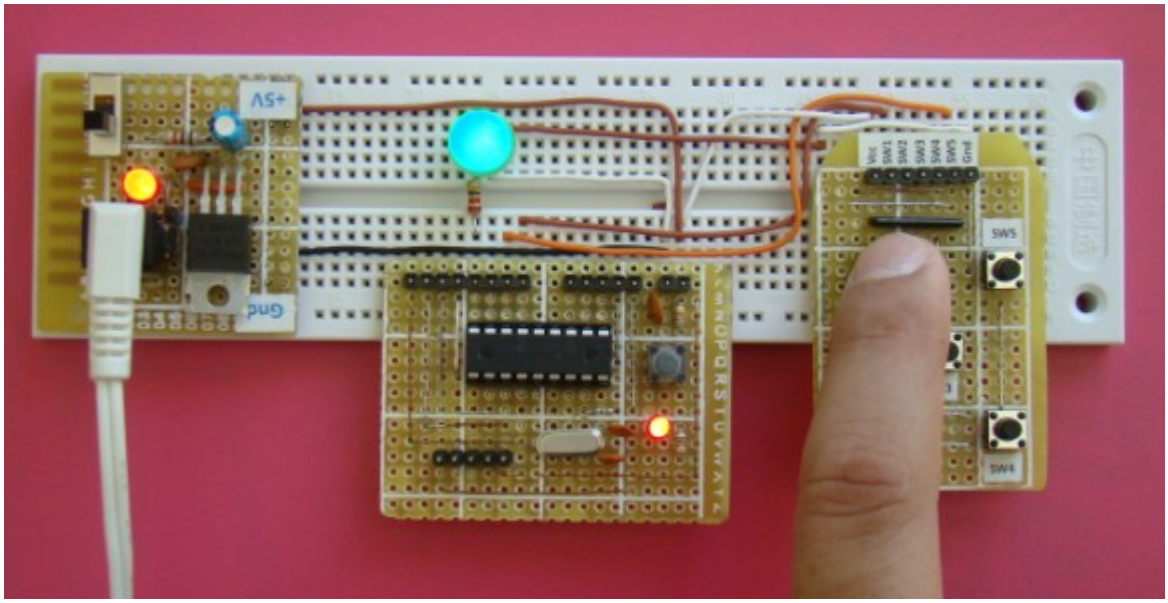
```
PORTB = 0x00;
TRISB = 0b00000011; // RB0, RB1 input, RB3 (PWM1) output
PWM1_Init(5000);    // PWM module initialization (5KHz)
new_DC = 0;         // Initial value of variable Duty Cycle
current_DC = 0;
PWM1_Start();       // Start PWM1 module with Zero DC
PWM1_Set_Duty(current_DC);
do {
  if (!UP){      // If the button connected to RB0 is pressed
   debounce();
   if (new_DC < 250)      // Don't go above 250
   new_DC = new_DC + 25 ; // increment Duty Cycle by 25
  }
  if (!DOWN) {   // If the button connected to RB1 is pressed
   debounce();
   if (new_DC !=0)        // Don't go below 0
   new_DC= new_DC - 25 ; // Decrement Duty Cycle by 25

  if (current_DC != new_DC) {
   current_DC = new_DC ;
   PWM1_Set_Duty(current_DC); // Change the current DC to new value
  }
} while(1);
}  // END main()
```

결과

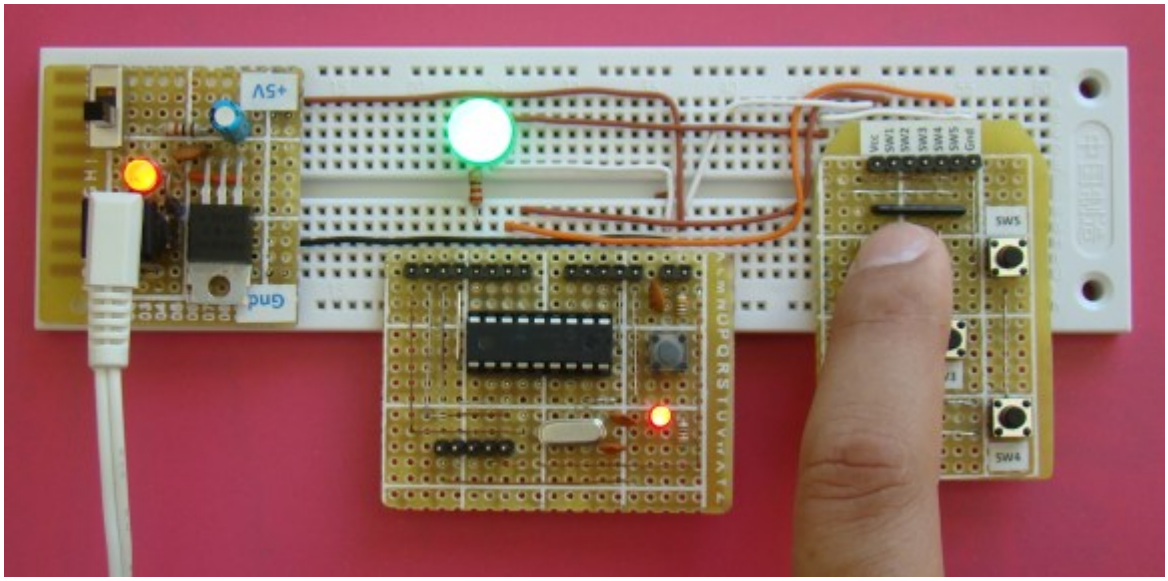
회로에 전원이 들어오면, LED는 밝기 0(zero duty cycle) 부터 시작합니다. UP버튼을 누르게 되면 듀티사이클을 증가되고 LED가 빛나기시작합니다. UP버튼을 계속 누르면 듀티사이클이 1에 가까워 질때까지 LED는 계속 밝아 집니다.



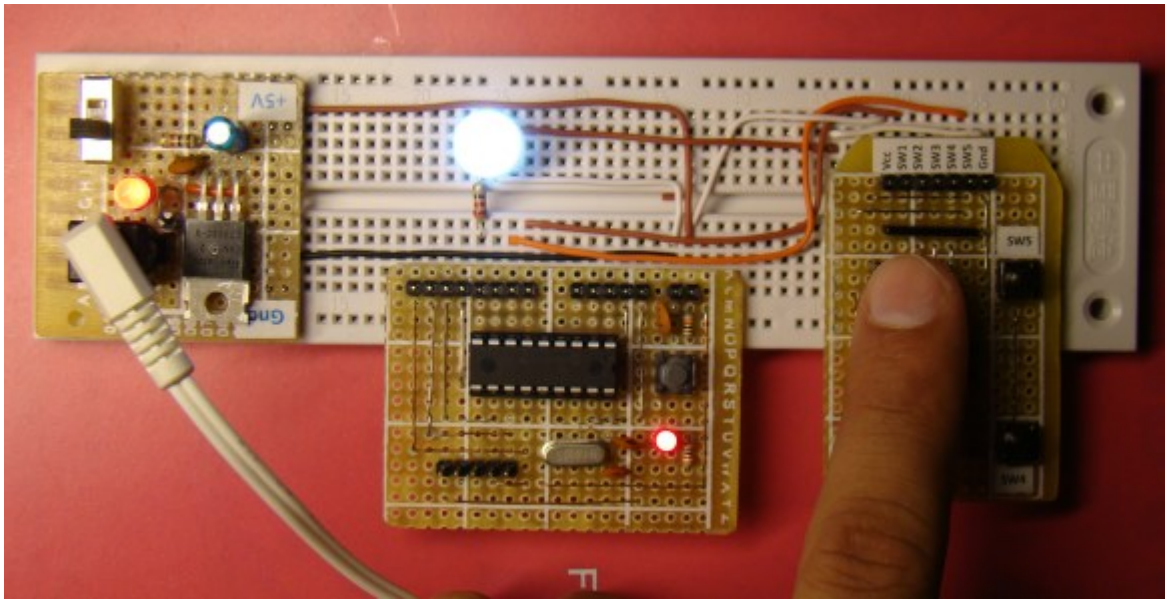
0% Duty Cycle



25 % Duty Cycle



50% Duty Cycle



~ 100% Duty Cycle

가치창조기술 | www.vctec.co.kr

[PIC 마이컴 실험하기] 10. PIC마이컴에 DC모터 연결하기

마이컴 실험실

2011/10/12 12:12 http://blog.naver.com/ubicomputing/150121201320

움직이는 무언가를 만드는 것이 임베디드 시스템을 만드는 큰 재미중에 하나가 아닐까 싶은데요. 임베디드시스템을 움직이게 만드는 장치는 DC모터, RC 서보, 스텝퍼 모터가 대표적입니다. 이번 게시물에서는 DC모터와 PIC 마이컴을 어떻게 연결하는지 살펴보겠습니다.

선행이론

DC모터는 두개의 자기장의 상호작용을 통하여 전기적인 에너지를 기계적인 에너지로 바꾸어주는 전기적으로 제어되는 장치입니다. 자기장중 하나는 고정자에 붙어 있는 영구자석에서 나오며, 다른 하나의 자기장은 회전자에 감겨있는 모터코일에 흐르는 전류에 의해 생성됩니다. 이 두개의 자기장은 회전자가 회전하는 성질을 가지게 만들어 주게 되죠. 본 게시물에서는 아래와 같은 브러쉬드 DC모터를 사용합니다.

브러쉬드 DC모터의 모드

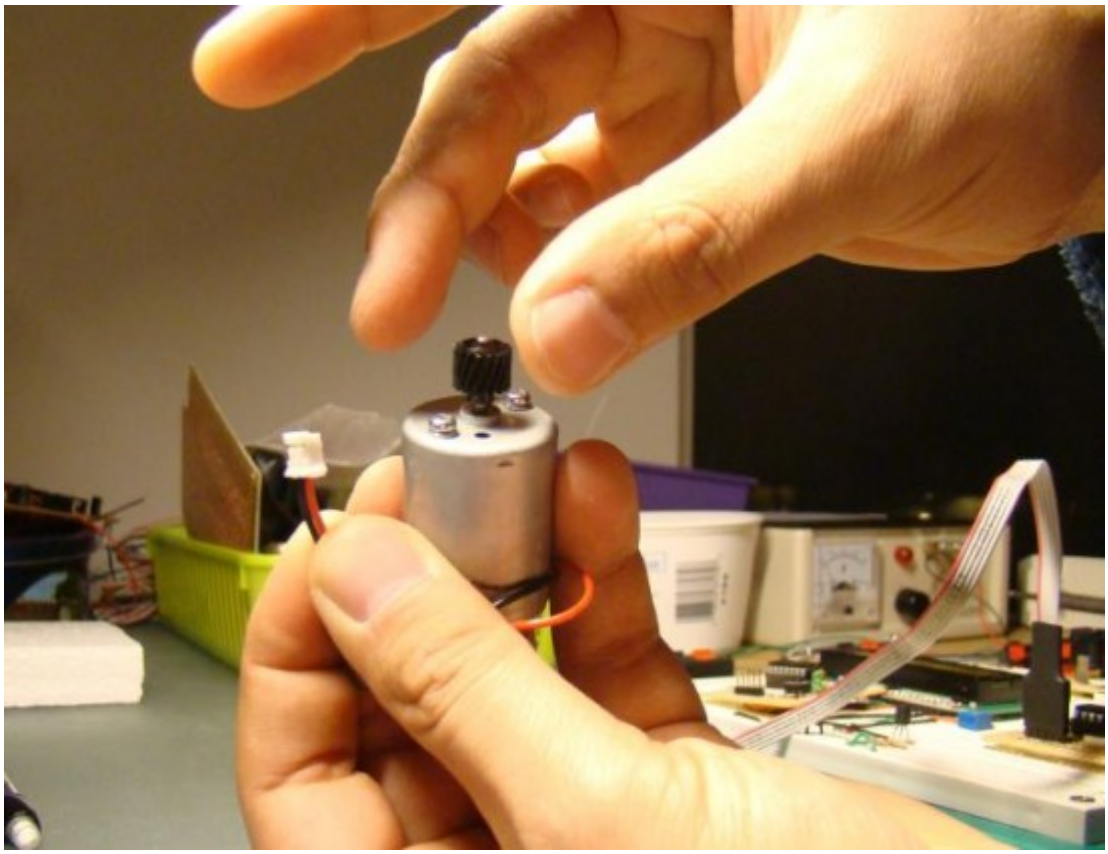
브러쉬드 DC모터는 4개의 기본적인 모드를 가지고 있습니다. *clockwise mode*, *counter-clockwise mode*,*coast mode*, *brake mode* 입니다.

Clockwise모드에서는 모터는 시계방향으로 회전하고 counter-clockwise모드에서는 반시계방향으로 회전합니다. 이 두 가지 모드를 변환하는 것은 매우 쉬운데 전원의 +/-를 변환하면 됩니다.

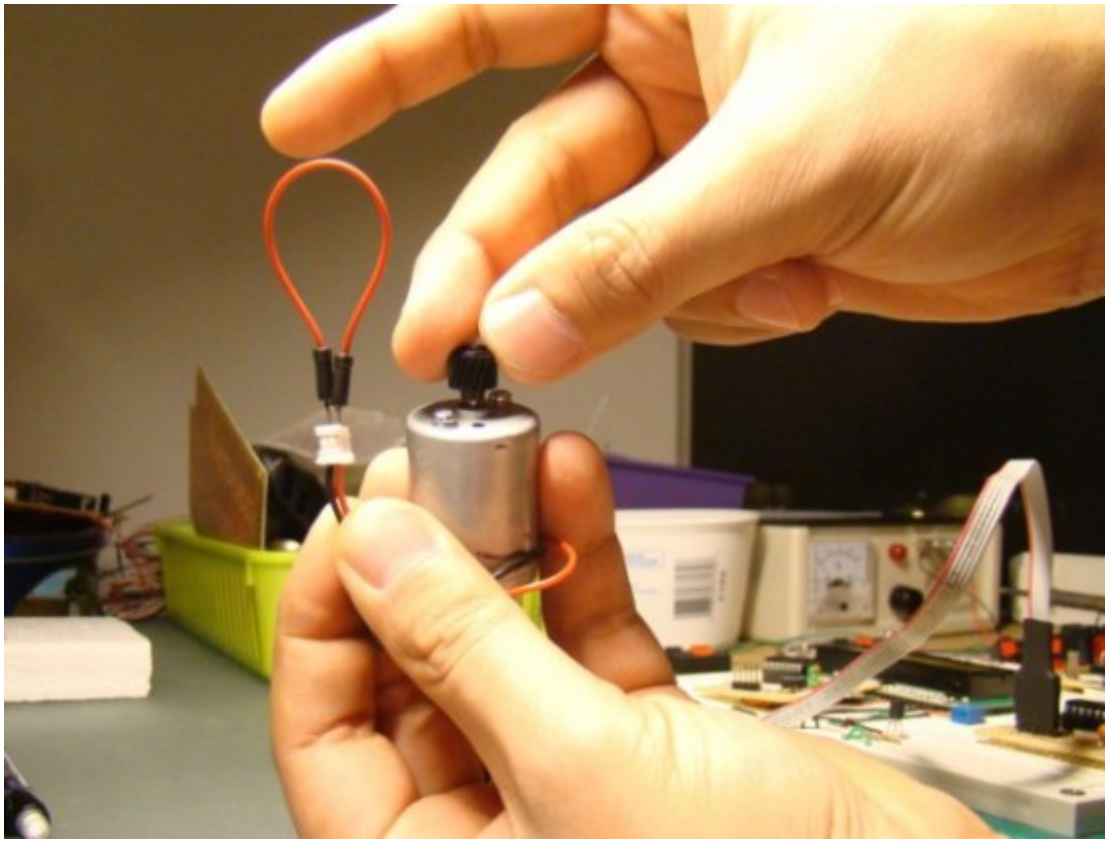
coast모드는 모터의 free spinning 모드를 의미합니다. 만약 모터의 축을 손가락으로 돌려본다면 관성에 의해 축이 좀더 도는 것을 볼 수 있을 것입니다. coast모드에서는 모터를 멈출때 모터가 천천히 멈추게 합니다.

brake모드는 모터의 양극 터미널을 쇼트시켜 모터를 빠르게 멈추게 하는데 사용됩니다. 모터는 회전할때 발전기처럼 동작합니다. 만약 모터의 터미널이 쇼트된다면, 모터는 무한부하처럼 동작하여서 모터를 빨리 멈추게 합니다.

coast모드와 brake모드를 구별하기 위한 간단한 실험을 하여볼수 있습니다. 브러쉬드 DC모터를 가지고 손가락으로 모터의 축을 회전시켜봅니다. 모터의 축이 자유롭게 회전하는것을 볼수 있습니다. 이번에는 양쪽 터미널을 쇼트시키고 다시 모터의 축을 회전시켜보면 모터가 coast모드일때보다 더 느려진것을 발견할 수 있습니다.



Coast mode: 모터가 자유롭게 돔.



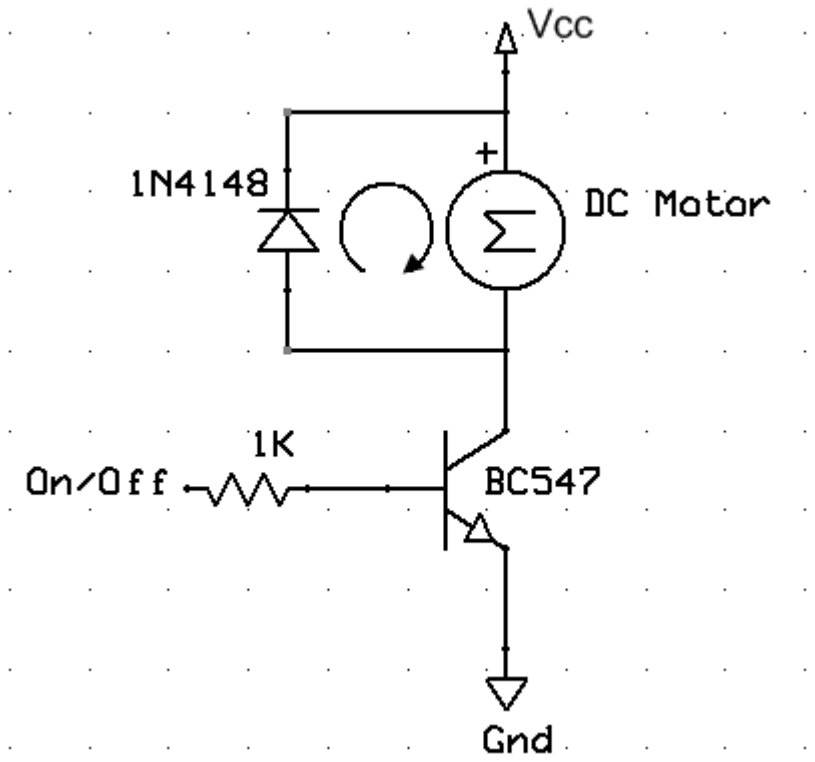
Brake mode: 모터가 좀 더 느려짐

모터 드라이버

모터는 단순하고 전력을 많이 필요로하는데 반해 마이크로컨트롤러는 데이터 프로세싱을 위해 디자인 되었습니다. 마이크로컨트롤러의 내부 구조상 MCU는 모터를 직접 동작시킬 충분한 전류를 공급할 수 없습니다. 그래서 MCU가 제어 할수 있고 모터에 전기적인 힘을 전달하기 위해 외부 드라이버 회로가 필요 합니다. 이런 회로는 트랜지스터 하나처럼 간단할 수 있지만 IC칩처럼 복잡할수 도 있습니다.

싱글 트랜지스터 모터 드라이버

아주 기본적인 모터 드라이버 회로는 모터를 on/off만 하는 싱글 트랜지스터 스위치입니다. 싱글 트랜지스터는 모터를 오직 한 방향으로만 움직이게 할수 있으며 또 모터에 전기적으로 brake를 줄수 없습니다. 싱글 트랜지스터 모터 드라이버의 회로도는 다음과 같습니다.



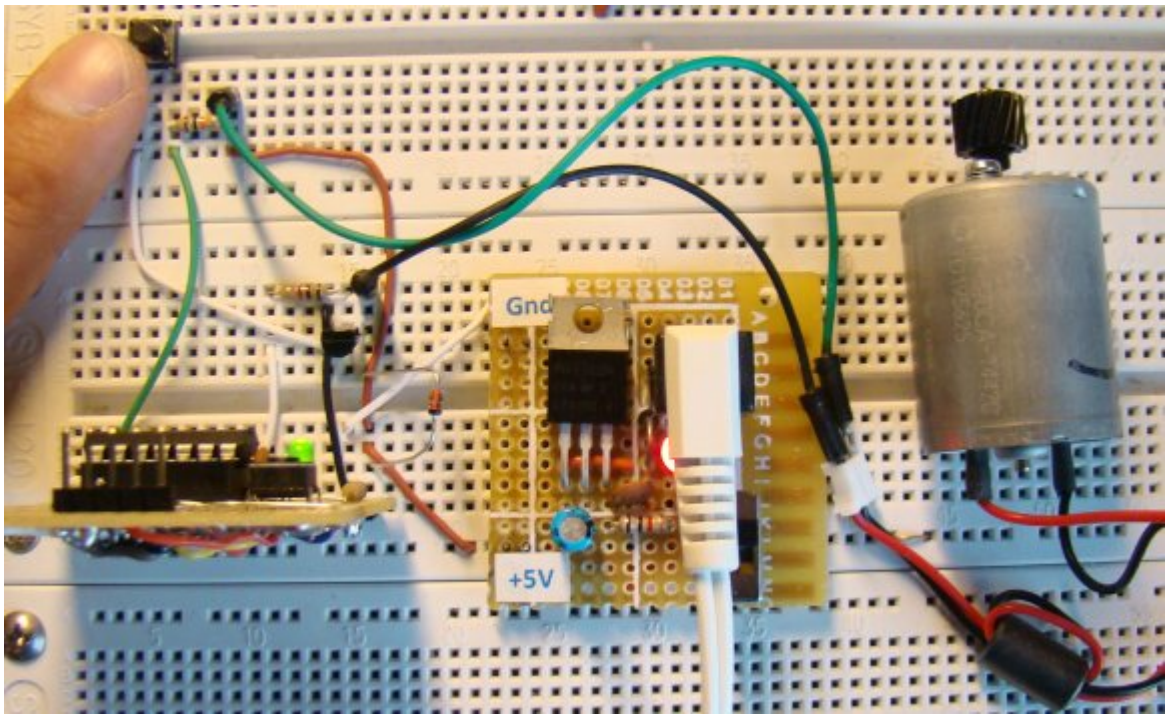
이 예제에서 범용 NPN 트랜지스터인 BC547이 사용되었습니다. BC547을 선택한 이유는 비싸지도 않고 구하기도 쉽기 때문입니다만 트랜지스터가 파워풀하지는 않습니다. 최대 100mA의 전류를 공급할 수 있어 작은 DC모터를 구동하는데는 여전히 좋은 트렌지스터입니다.

이 회로의 작동 원리는 간단합니다. 트랜지스터가 모터의 전원 스위치 역활을 하는 것입니다. 스위치는 ON/OFF 터미널에 인가된 로직 전압을 통해 제어될 수 있습니다. ON/OFF핀이 로직1(+5V)이면 트랜지스터는 ON이고 모터는 Vcc와 GND를 통해 연결됩니다. Vcc전압은 5V보다 클수 있습니다. 그래서 이 싱글 트랜지스터 모터 드라이버의 한가지 장점은 MCU에서 나오는 5V 로직 출력을 가지고 5V이상의 전압을 가지는 모터를 제어 할수 있다는 것입니다. ON/OFF터미널에서 로직 0은 트랜지스터를 OFF로 만들고 모터는 멈추게 됩니다. 1K 저항이 트렌지스터의 베이스와 연결되어 베이스의 전류를 안전한 레벨로 제한하여 줍니다.

모터의 양극 터미널사이에 연결된 다이오드는 트랜지스터를 보호하기 위한 용도입니다. 스위치가 OFF될때 모터내에 떨어지는 전자기장이 모터의 양극단에 역극성을 띈 고전압을 생성시키게 됩니다. 이 전압은 트랜지스터를 망가트리기에 충분한 전압입니다. 다이오드는 모터코일로 통해 다시 오는 전류패스를 제공하기때문에 고전압 형성을 방지합니다. 보통의 경우, 다이오드는 역 바이어스이며 트랜지스터의 스위치 동작에 영향을 미치지 않습니다.

이 회로를 테스트하기 위해 두번째 게시물"기본적인 디지털 입력과 출력"에서 나왔던 회로의 LED를 교체하고 PIC16F688의 RC0핀을 위의 트랜지스터 드라이버 회로의 ON/OFF터미널에 연결합니다. 두번째 게시물에서 작성하였던

같은 HEX파일을 PIC16F688에 로드합니다. 버튼 스위치를 눌름에 따라서 모터가 ON/OFF하는 것을 볼수 있을 것입니다. 모터가 회전하는 방향은 모터의 양극이 어떻게 연결되었느냐에 따라 결정이 됩니다.



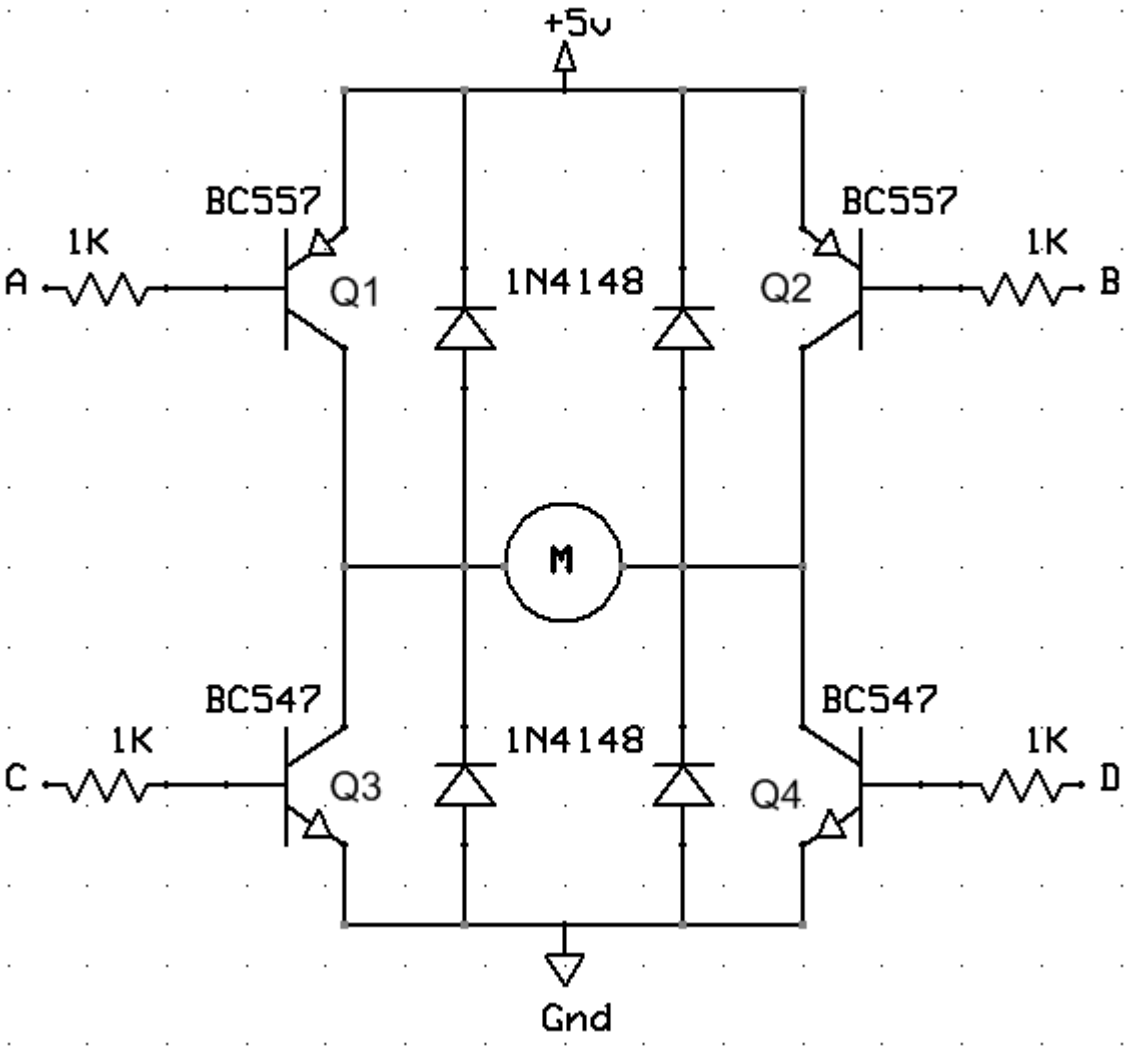
싱글 트렌지스터를 이용하여 DC모터의 ON/OFF동작을 제어

H-Bridge 모터 드라이버

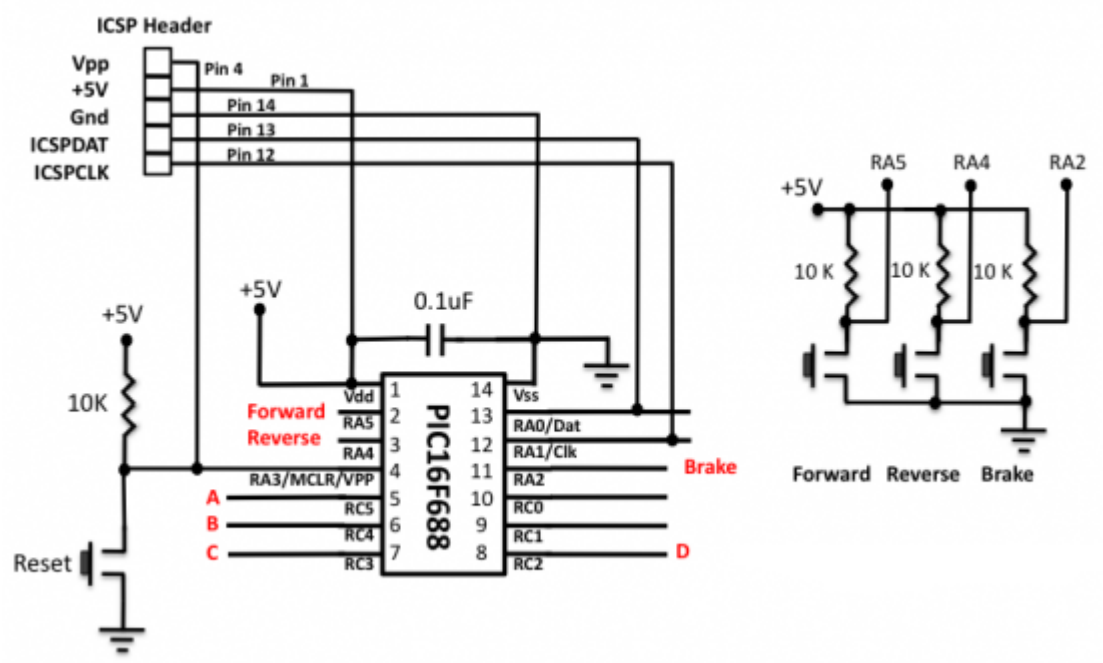
싱글 트랜지스터 모터 드라이버는 제약사항들이 있습니다. 예를 들어 모터를 한방향만으로만 제어할수 있다는 점과 전기적인 brake를 줄수 없다는 점이 그렇습니다. H-bridge 모터 드라이버는 4개의 DC모터 모드를 다 제공합니다. 이 드라이버는 H-bridge로 불리는데 회로가 H와 닮아서 그렇습니다.

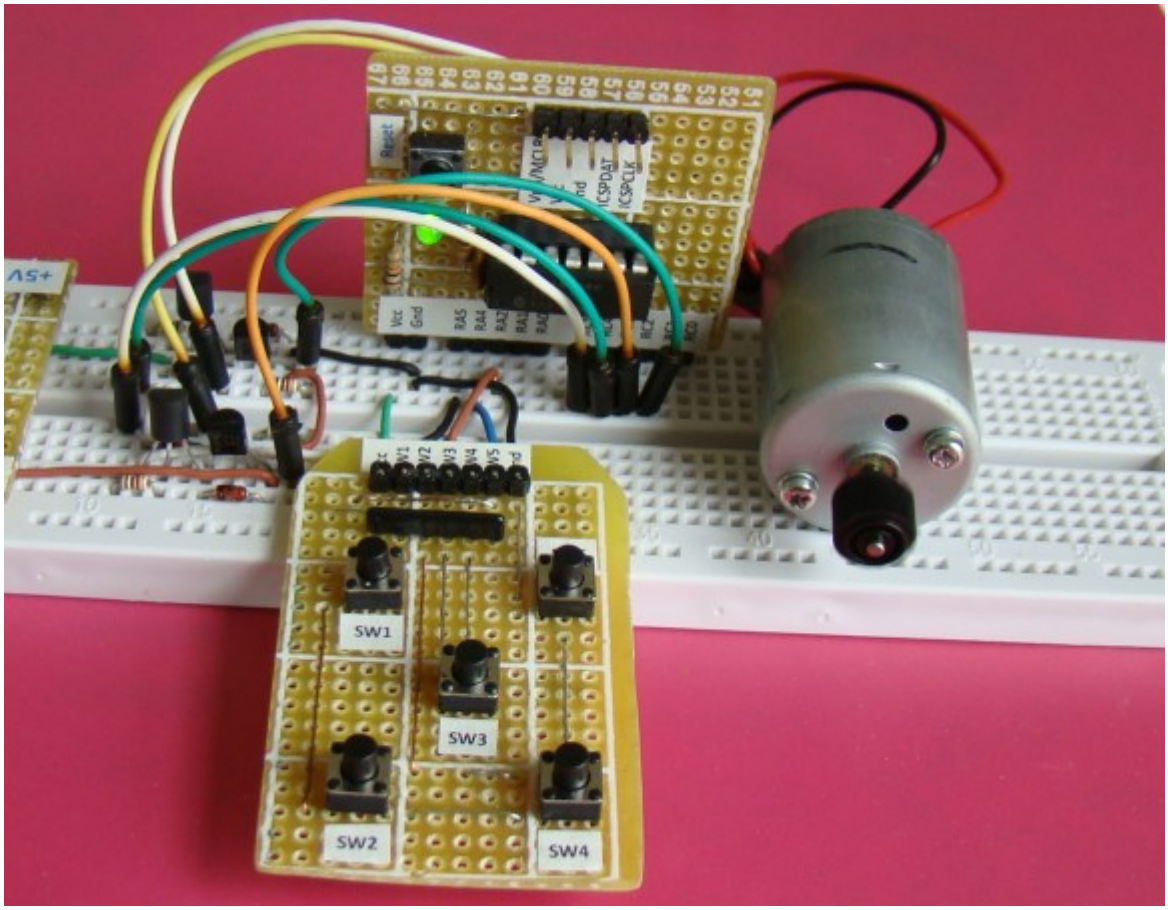
표준 H-bridge는 4개의 트랜지스터(아래의 그림과 같이 2개의 PNP가 위에 2개의 NPN이 아래에)로 구성되어 있습니다. 좌측과 오른쪽은 서로 동일한 형태이며 드라이버 회로의 4개의 입력은 A,B,C,D로 이름 붙여져 있습니다. 터미널 B가 그라운드되고 터미널 C가 +5V로 HIGH 상태가 되면, 트랜지스터 Q2와 Q3에 전류가 흐릅니다. 이것은 모터를 한방향으로 돌게 만듭니다. 만약 터미널 A와 D가 각각 그라운드와 +5V에 연결되면 Q1, Q4는 ON 상태가 되어 "Vcc(+5V)-Q1-모터-Q4-그라운드" 의 전류 패스를 만들게 됩니다. 모터는 이 전류에 의해 먼저회전한 방향의 역방향으로 회전하게 됩니다.

Brake모드는 로직 1(+5V)를 핀C, D에 인가하거나 A,B를 그라운드 시킴으로 만들수 있습니다.
Coast 모드는 4개의 트랜지스터가 OFF, 즉 A,B에는 로직 1(+5V) 상태이고 C,D는 그라운드되어 있을때 만들수 있습니다. 이때는 모터가 자유롭게 회전을 합니다.



본 H-bridge는 DC모터와 PIC16F688모듈로 테스트하였습니다. DC모터를 시계방향, 반시계방향, 브레이크를 제어하기 위해서는 PIC16F688의 4개의 I/O핀이 필요합니다. 4개의 핀은 적절한 로직 레벨을 A,B,C,D에 인가하여 원하는 모터 모드를 만들것입니다. 본 테스트에서는 3개의 푸쉬버튼을 추가하여 시계방향 회전, 반시계방향 회전, 브레이크모드를 선택할 것입니다. 회로도는 아래와 같습니다.





브레드보드에 H-bridge 셋업하기

중요

H-bridge를 구현할시, 올바른 트랜지스터에 올바른 시간에 스위치on할수 있도록 주의하여야 합니다. Q1, Q3(혹은 Q2, Q4)에 스위치ON에 부주의 할 경우 쇼트서킷이 날수 있으며 트랜지스터가 타버릴수 있습니다. 16개의 가능한 A,B,C,D입력 조합중에서 7개의 조합이 쇼트회로로 반드시 피해야합니다. A와C(AC터미널)를 묶고, B와 D(BD터미널)를 묶어 모터를 컨트롤할 두개의 터미널만 만들수도 있습니다. AC에 1D을 BD에 0을 인가하면 모터는 한방향으로 회전합니다. AC와 BD의 로직레변을 바꾸면 모터는 역방향으로 회전합니다. 0이나 1을 AC와 BD에 주면 모터를 멈추게 합니다. 중요한 것은 AC나 BD터미널을 플로팅상태로 놓지 않는것입니다.

AC터미널을 좀더 자세히 보도록 하겠습니다. A와 C는 서로 연결되어 있으므로 만약 그것이 left float 이라면 Q1 콜렉터를 통하고, 베이스를 통하고 Q3의 베이트를 통하고, 에미터를 통해 그라운드로 가는 전류 경로가 생기게 될것입니다. 그래서 Q1과 Q3는 쇼트회로상에 있게 됩니다. 그래서 AC, BD로 컨트롤 핀을 줄일것을 생각하고 있다면 이런 것들을 주의해야합니다. 묶어놓은 터미널을 open, float상태로 놓지 말고 그라운드 시키거나 high시켜야 합니다.

AC	BD	Mode
0	1	Clockwise
1	0	Counterclockwise
1	1	Brake
0	0	Brake
Float	Float	Short circuit

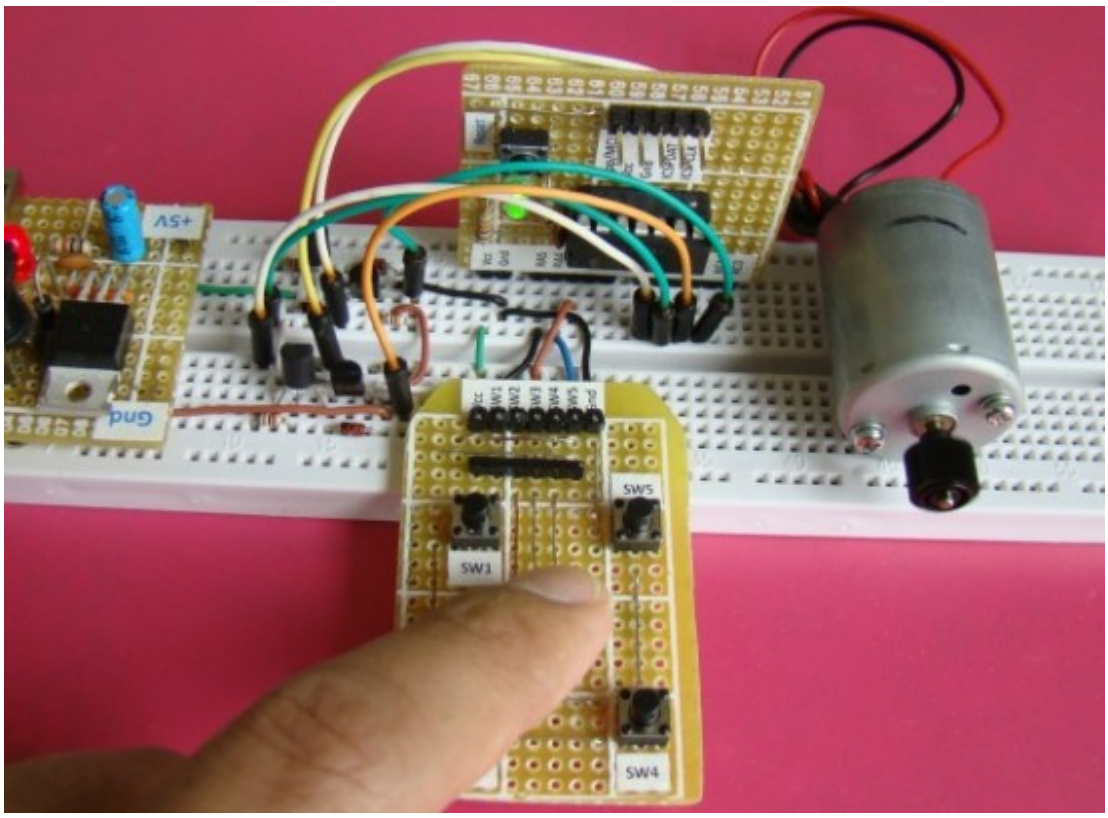
다시말하지만, 본 회로는 작은 모터를 테스트하기 위한 회로입니다. 부하가 걸리는 모터를 동작하기 위해서는 좀더 높은 전류등급을 가진 트렌지스터를 사용하여 회로를 구성해야합니다. BC557을 2907A로 BC547을 2222A 트랜지스터로 교체하면 600mA까지 캐파를 늘릴수 있습니다. 이것보다 더 높은 사양이 필요하다면 BJTs대신 MOSFET사용을 추천하여 드립니다.

H-Bridge 테스트 소프트웨어

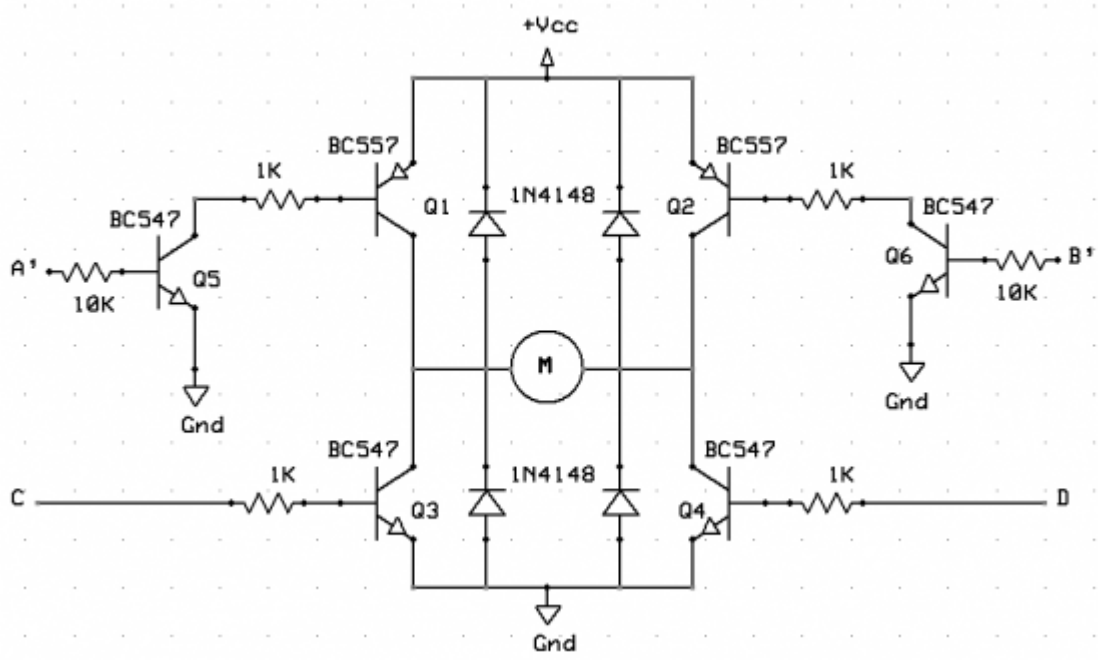
```
/*
 Lab 10: DC Motor control (H-Bridge)
 Description: Forward and reverse motor control
```

```
 MCU: PIC16F688
 Oscillator: Internal clock at 4.0 MHz, MCLR Enabled
*/

sbit Forward at RA5_bit;
sbit Reverse at RA4_bit;
sbit Brake at RA2_bit;
sbit PinA at RC5_bit;
sbit PinB at RC4_bit;
sbit PinC at RC3_bit;
sbit PinD at RC2_bit;
void GetDelay(){
 Delay_ms(300);
}

void main() {
 CMCON0 = 0x07; // Disable comparators
 ANSEL = 0b00000000;
 PORTC = 0x00;
 TRISC = 0b00000000; // PORTB all output
 TRISA = 0b00111100; // RA5, RA4, RA2 inputs

 do {
  if (!Forward){
  GetDelay();
  PORTC = 0;
  GetDelay();
  PinA = 0;
  PinB = 1;
  PinD = 1;
 }

 if (!Reverse) {
  GetDelay();
  PORTC = 0;
```

```
  GetDelay();
  PinA = 1;
  PinB = 0;
  PinC = 1;
 }

 if (!Brake) {
  GetDelay();
  PORTC = 0;
 }
 } while(1);
}  // END main()
```



위의 H-bridge회로는 5V(로직1)이상의 전압으로 모터를 동작하지 못하는 단점을 가지고 있습니다. 만약 9V 모터를 동작시키려 Vcc를 9V로 바꾼다면, 터미널A,B가 HIGH(+5V)일시 트랜지스터 Q1과 Q2는 ON입니다. 이런 경우 MCU와 H-bridge회로 사이에 5V를 9V로 바꾸는 추가적인 전압변환회로가 필요합니다. 아니면 두개의 추가적인 NPN 트랜지스터를 H-bridge에 사용할 수 있습니다.(아래 참조) 컨트롤핀은 A', B', C, D가 됩니다.



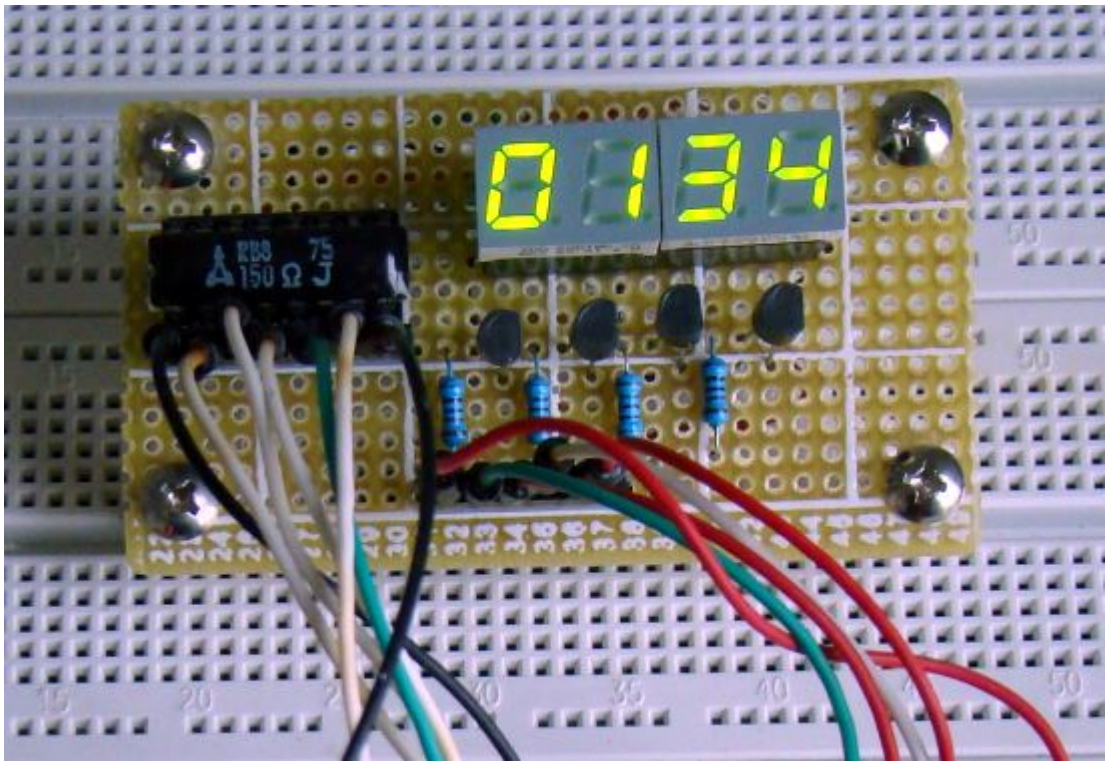
가치창조기술 | www.vctec.co.kr

[PIC 마이컴 실험하기] 11. 7-Seg 디스플레이 멀티플렉싱

마이컴 실험실

2011/10/12 15:13 http://blog.naver.com/ubicomputing/150121215708

지난 게시물중 여섯번째 게시물에서는 7-Seg LED 디스플레이를 PIC MCU와 연결하는 것을 살펴보았는데요. 7개의 세그먼트들이 MCU의 I/O핀을 통해서 개별적으로 동작이 되었습니다. 만약에 그러한 동작을 4개의 7-Seg LED에 대해서 수행을 한다고 하면 28개의 I/O핀이 필요할 것인데 28개의 핀은 꽤 큰 리소스로 중급 PIC MCU에 의해서는 지원되기 힘듭니다. 그렇기 때문에 여러개의 7-Seg LED를 멀티플렉싱하는 기술이 필요한 것입니다. 본 게시물에서는 4개의 양극타입 7-Seg LED와 PIC16F628A를 가지고 어떻게 멀티플렉싱하는지를 살펴볼 것입니다.

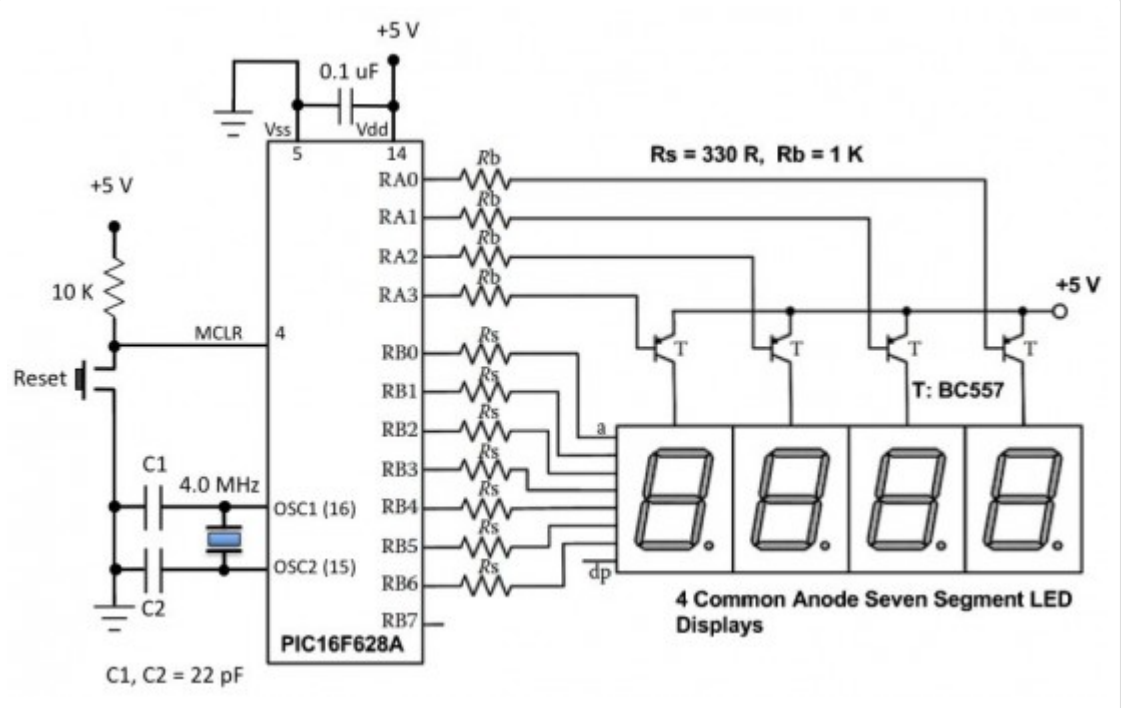


4개의 7-Seg LED를 멀티플렉싱

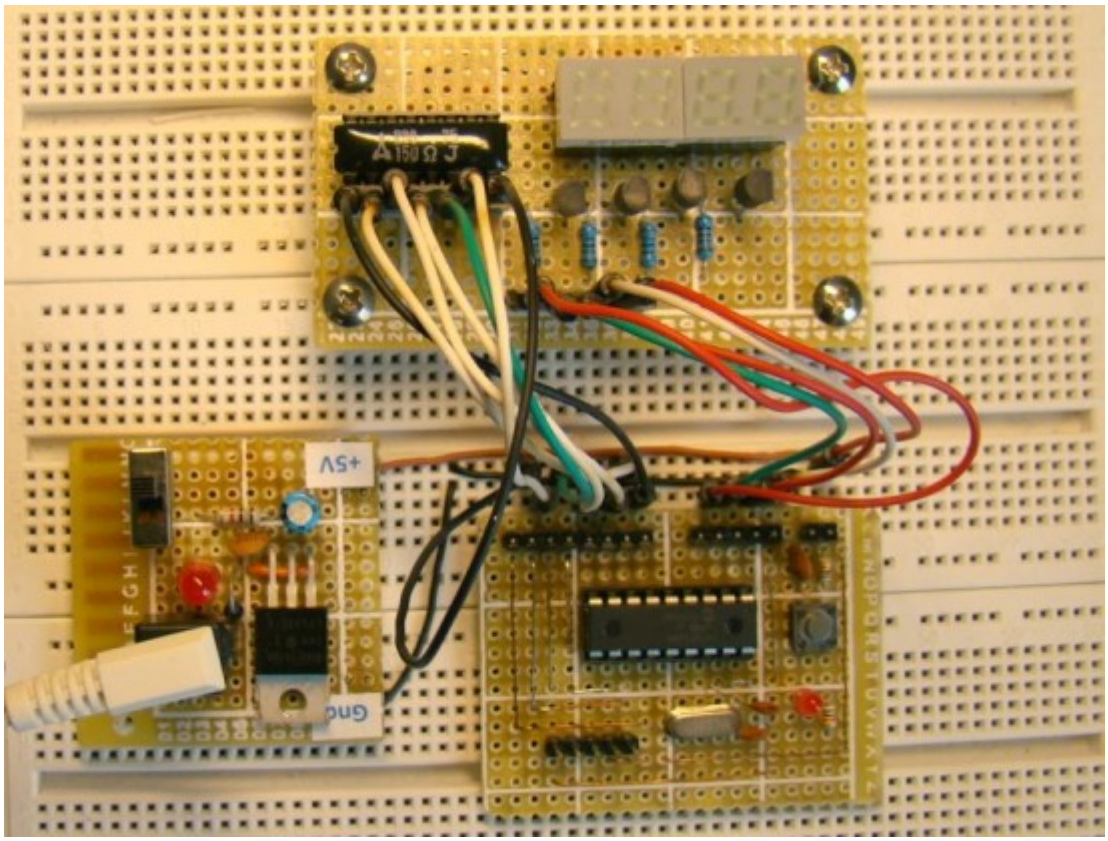
회로도

멀티플렉싱 기술의 원리는 간단합니다. 여러개의 LED 디스플레이 중 모든 비슷한 세그먼트들은 함께 연결되어 한개의 I/O핀을 통하여 동작됩니다. 아래의 회로에서 7-Seg는 전류제한 저항 Rs을 통해 PORTB에 연결되어 있습니다. PORTB핀이 LOW일때 해당 세그먼트는 활성화 되고 양극이 Vcc에 연결되어 있으면 빛이 납니다. 아래의 그림에서 4개 LED의 양극이 Vcc에 직접 연결되어 있지 않은 것을 볼 수 있는데, 대신 4개의 PNP 트랜지스터가 양극 터미널을 Vcc에서 연결하였다 안하였다하는 스위치로 사용이 됩니다. 트랜지스터의 베이스가 HIGH 일때 트랜지스터에는 전류가 흐를수있게 되고 해당 자리수의 공통 양극이 Vcc에 연결이 됩니다. 그래서 트랜지스터는 어떤 디스플레이를 활성화 시킬지 선택을 할수 있습니다. 트랜지스터는 PORTA의 RA0-RA3에 의해 제어됩니다. 우리가 7을 디스플레이하고 싶다고 가정했을때, 세그먼트

a,b,c는 먼저 on되어야 하고(RB0, RB1, RB2는 0, RB3-RB6는 1) RA0는 LOW가 되어야 합니다(RA1-RA3는 HIGH). 4개의 자리수를 전부 표시하기 위해서 각각의 7-Seg 디스플레이는 적당한 refresh freqency를 이용하여 순차적으로 활성화 되게하여 동시에 모든 것이 on으로 나타나게 합니다.



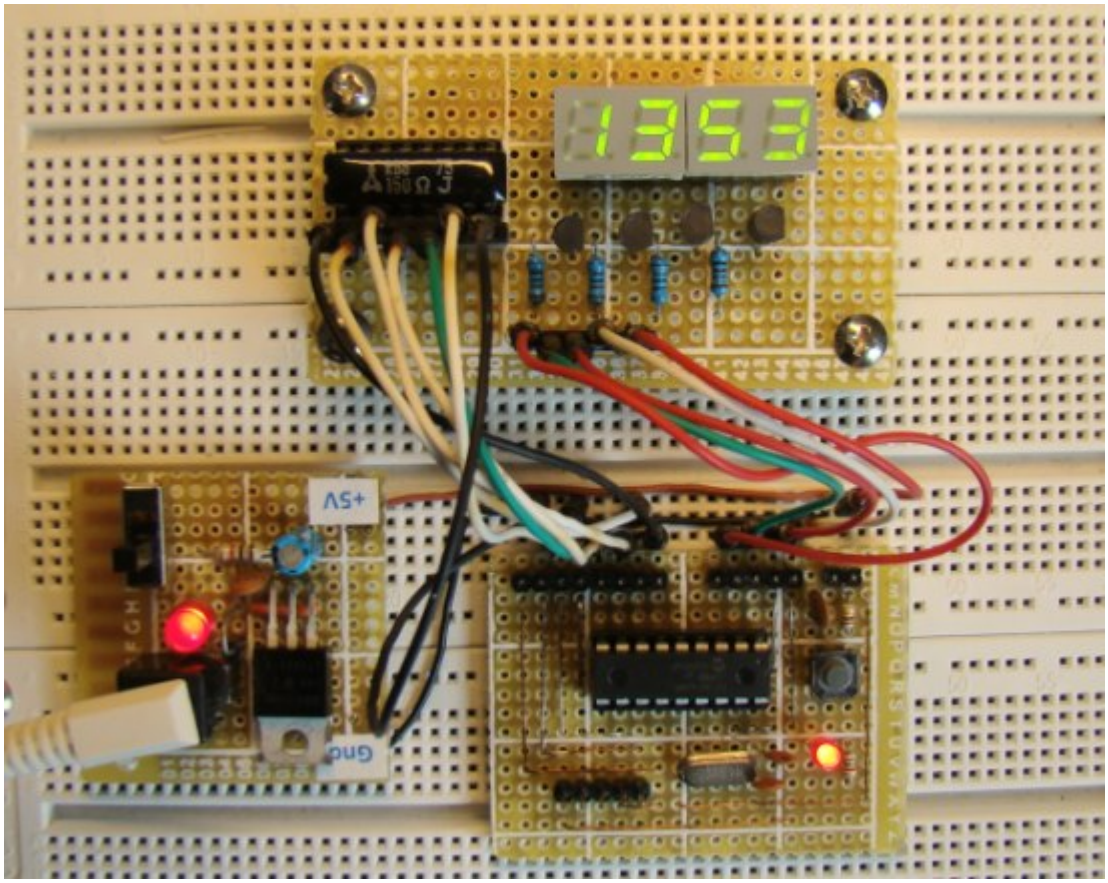
멀티플렉싱 기술을 이용하여 4개의 7-Seg LED 디스플레이 동작하기



멀티플렉스된 7-Seg LED 디스플레이 회로 셋업

소프트웨어

mikroC 컴파일러를 이용하여 4 digit 카운터 샘플 프로그램을 개발하였습니다. 카운터는 0부터 시작하여 9999까지 매초마다 증가합니다. 9999가 되면 0으로 리셋됩니다. 카운터의 값이 7-Seg LED에 나타나게 합니다. 프로그램이 mcu에 로딩되면 회로에 전원을 인가하고 카운터가 작동되는 것을 살펴보세요.



4-Digit 카운터

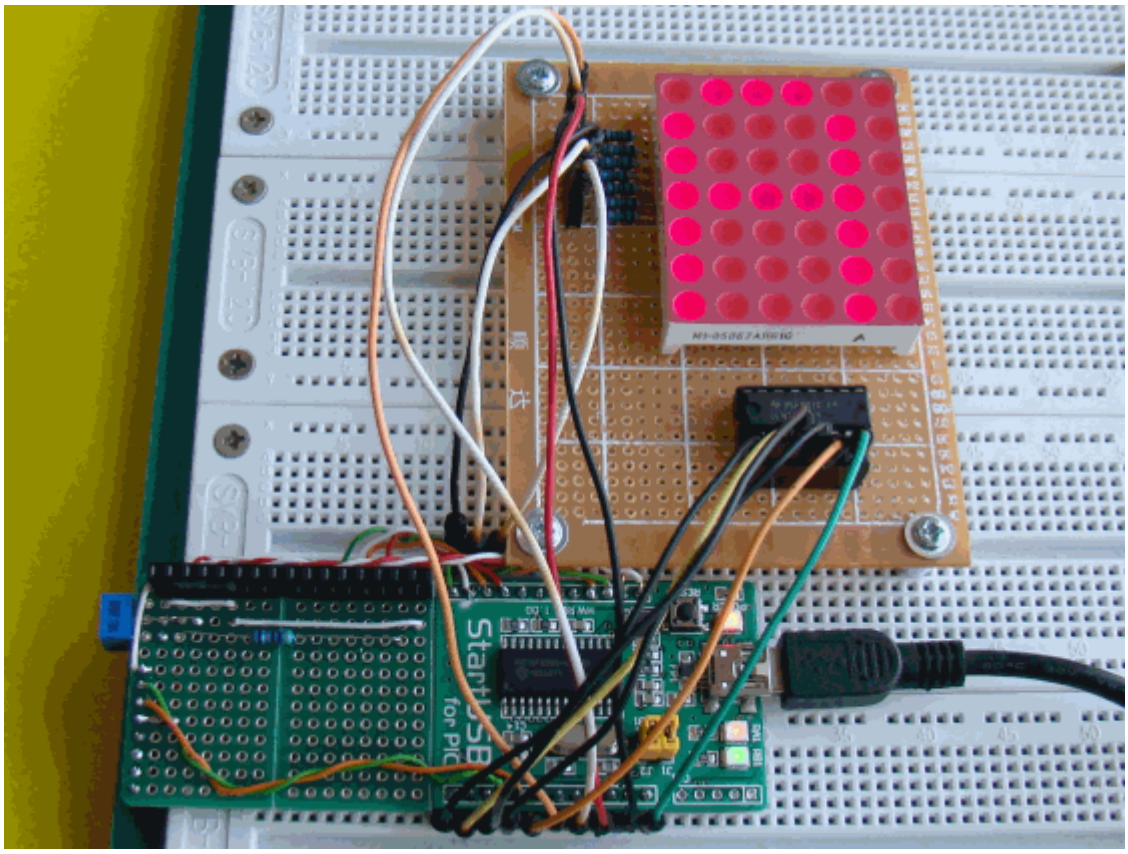
가치창조기술 | www.vctec.co.kr

[PIC 마이컴 실험하기] 12. LED Dot Matrix 디스플레이 기본

마이컴 실험실

2011/10/13 15:04 http://blog.naver.com/ubicomputing/150121299291

7-Seg LED 디스플레이를 PIC 마이컴에 연결하는 방법에 대해 살펴보았는데요. 이번 게시물에서는 LED 도트 매트릭스 디스플레이를 연결하는 방법에 대해 살펴보겠습니다. LED 도트 매트릭스는 정보를 표현하는데 아주 인기 있는 방법중에 하나인데요. 문자나 그림을 에니메이션화하여 표현할수 있기 때문입니다. 도로위의 LED광고판 등이 이러한 LED 도트 매트릭스입니다. 본 실험에서는 한개의 색만 표현하는 LED 도트 매트릭스를 PIC 마이컴에 연결하여 문자와 기호를 디스플레이하는 방법을 알아보겠습니다. 데모를 위해서 StartUSB for PIC 미니개발보드(PIC18F2550장착)을 사용하도록 하겠습니다.

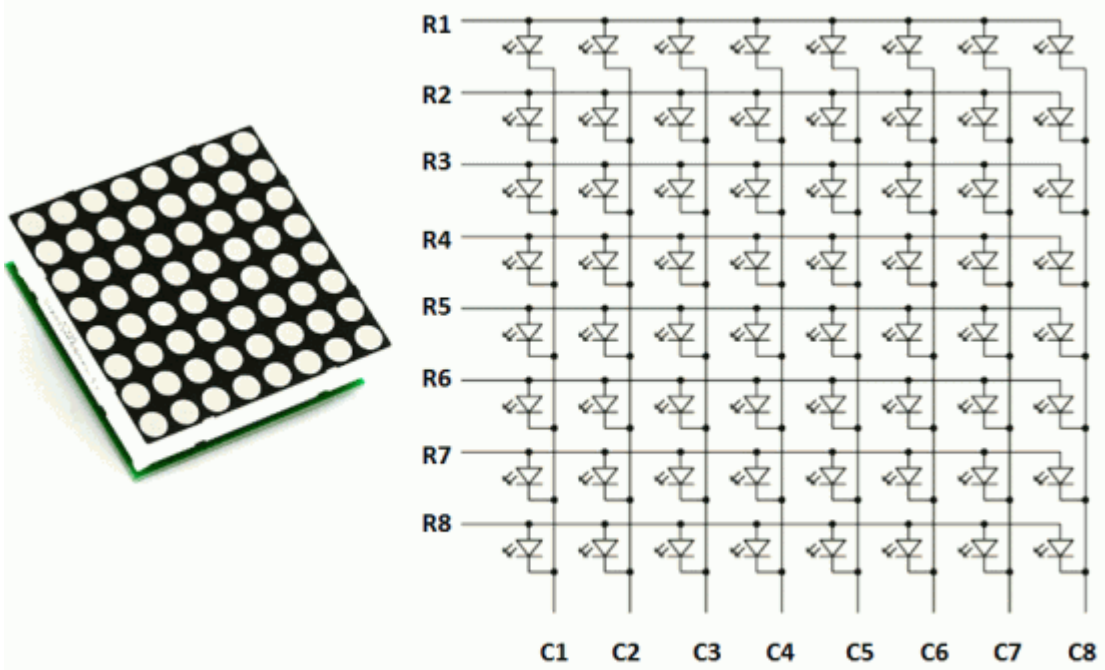


LED 도트 매트릭스와 PIC 마이컴 인터페이싱하기

LED 도트매트릭스 이론

도트매트릭스 디스플레이에는 여러개의 LED들이 격자모양으로 서로 연결이 되어 있습니다. 이렇게 격자모양으로 연결하여 묶어 놓은 이유는 도트매트릭스를 동작시키기 위한 핀의 숫자를 줄이기 위해서 입니다. 예를 들어, 8X8 매트릭스의 LED는 각각의 LED 픽셀당 1개씩 64개의 I/O핀이 필요합니다만 모든 양극을 행으로 함께 묶고(R1-RB), 모든 음극을 열로 열로 함께 묶으면(C1-C8) 필요한 I/O핀은 16개로 줄어 듭니다. 각각의 LED픽셀은 행과 열 조합으로 접근할수 있습니다.

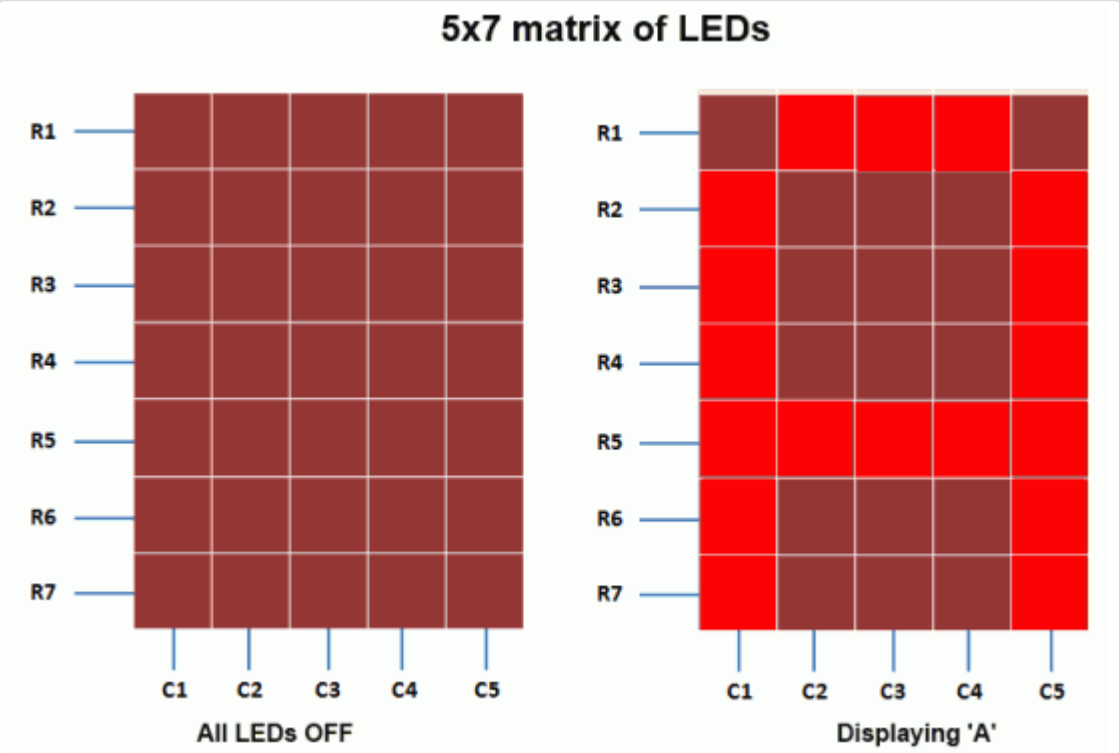
아래의 그림에서 R4를 HIGH로 C3를 LOW로 만들었을 때 (4, 3) 위치에 있는 LED가 ON됩니다. 문자는 이렇게 행과 열을 빨리 스캔하여가면서 디스플레이를 할 수 있습니다. 본 게시물에서는 column scanning에 대해서 설명하겠습니다.



8x8 LED 도트 매트릭스 구조

여기서는 5X7 사이즈의 LED 매트릭스를 사용하여 문자를 디스플레이하는 법을 다루어 보겠습니다. 아래의 그림은 알파벳 A를 표시하기 위하여 어떤 LED가 ON되어야 하는지 보여줍니다. 7개의 열과 5개 행이 MCU의 핀을 통해 제어 됩니다.

알파벳 A를 디스플레이 하고자 할때 먼저 C1을 선택(C1에 LOW적용)하고 다른 열은 그라운드 패스를 막아(C2-C5핀에 로직 HIGH적용) 선택을 취소합니다. 이제 첫번째 열은 활성화 되었고, forward bias전압을 적용하여 R2-R7까지 LED를 ON시킵니다. 다음으로 C2열을 선택하고 다른 열들은 선택을 취소합니다. 그리고 R1-R5에 foward bias를 적용합니다. 이런식으로 나머지도 진행을 합니다. 1초에 100번이상의 속도로 각 행을 빠르게 스캐닝하면서 각각의 행에 있는 해당 LED를 ON 시키면, 눈에 의한 잔상효과로 인해 우리는 하나의 이미지 "A"로 인식하게 됩니다.

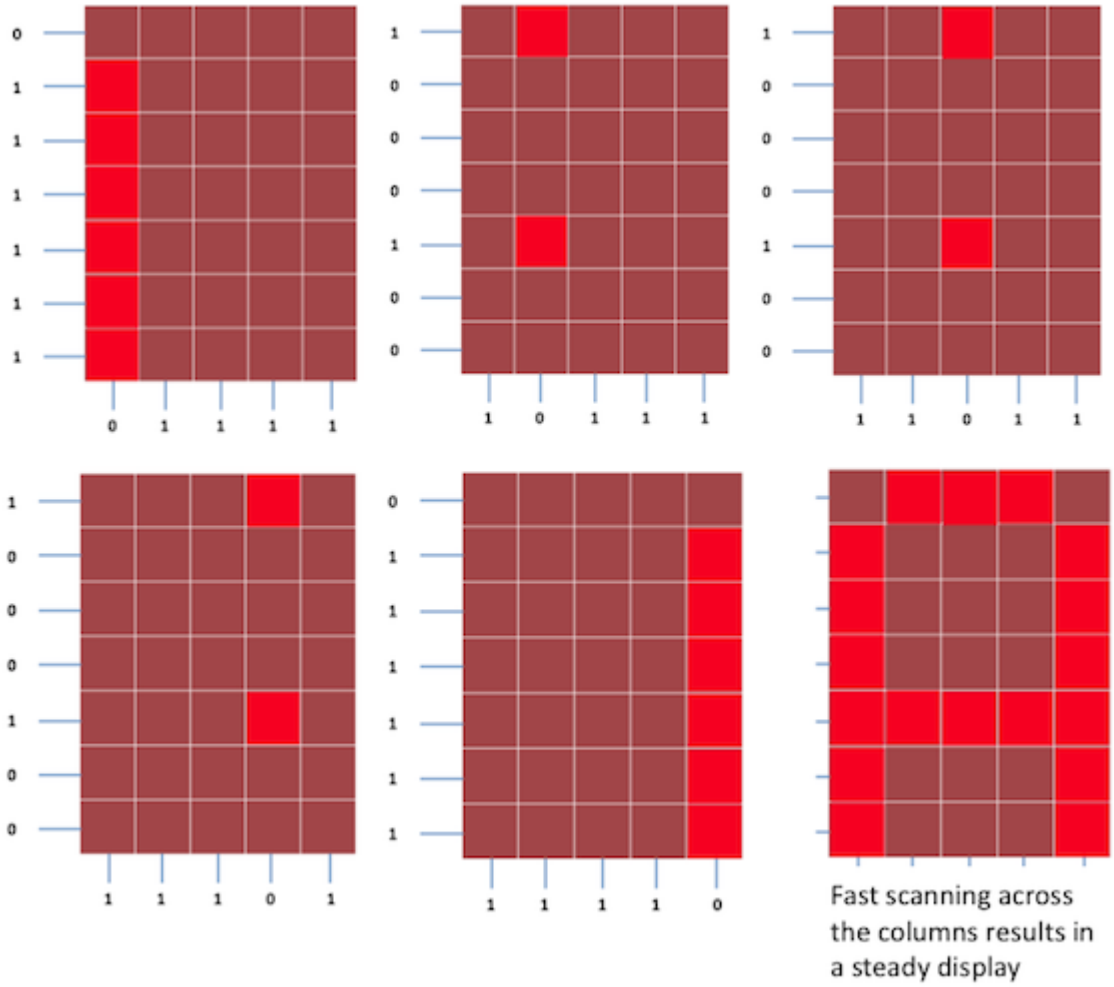


표준 5x7 LED 도트 매트릭스 구조

아래의 테이블은 알파벳 A를 디스플레이 하기 위해서 R1-R7에 적용되는 로직 레벨을 보여줍니다.

Row\Col	C1	C2	C3	C4	C5
R1	0	1	1	1	0
R2	1	0	0	0	1
R3	1	0	0	0	1
R4	1	0	0	0	1
R5	1	1	1	1	1
R6	1	0	0	0	1
R7	1	0	0	0	1

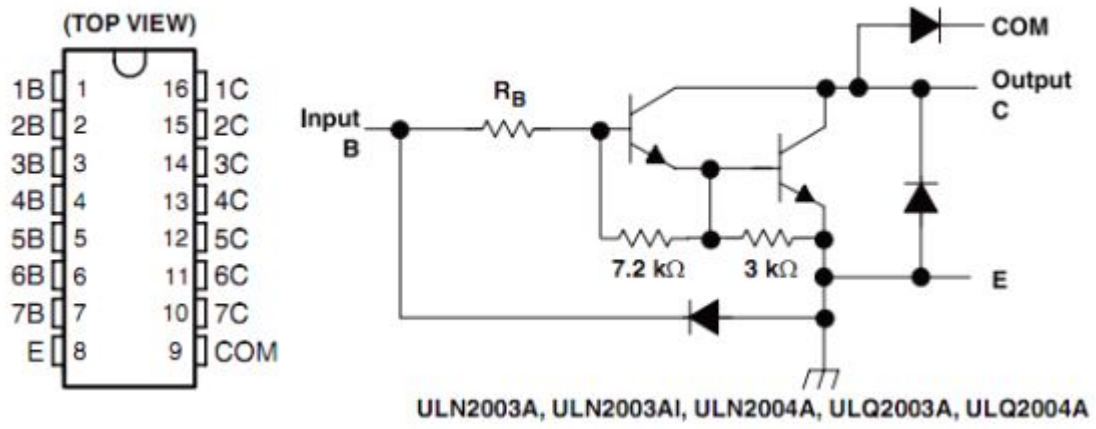
알파벳 A를 디스플레이하기 위한 각 열에 대한 행의 값



각 열을 스캐닝하고 행을 선택한 각각의 모습.

알아두어야 한것은 각각의 행에서 한개의 핀은 한번에 오직 한개의 LED에만 전류를 공급할수 있습니다만 열에 있는 핀은 한개 이상의 LED의 전류 sink입니다. 예를 들어, 알파벳 A를 디스플레이할때 C1열은 6개의 LED로부터의 전류를 싱크하여야 합니다. 마이크로컨트롤러의 I/O핀은 이 정도량의 전류를 싱크할수 없습니다. 그래서 외부 트랜지스터 어래이가 필요합니다. 여기서는 7개의 내장 Darlington 트랜지스터 어레이를 가지고 있는 ULN2003A IC를 이용합니다.

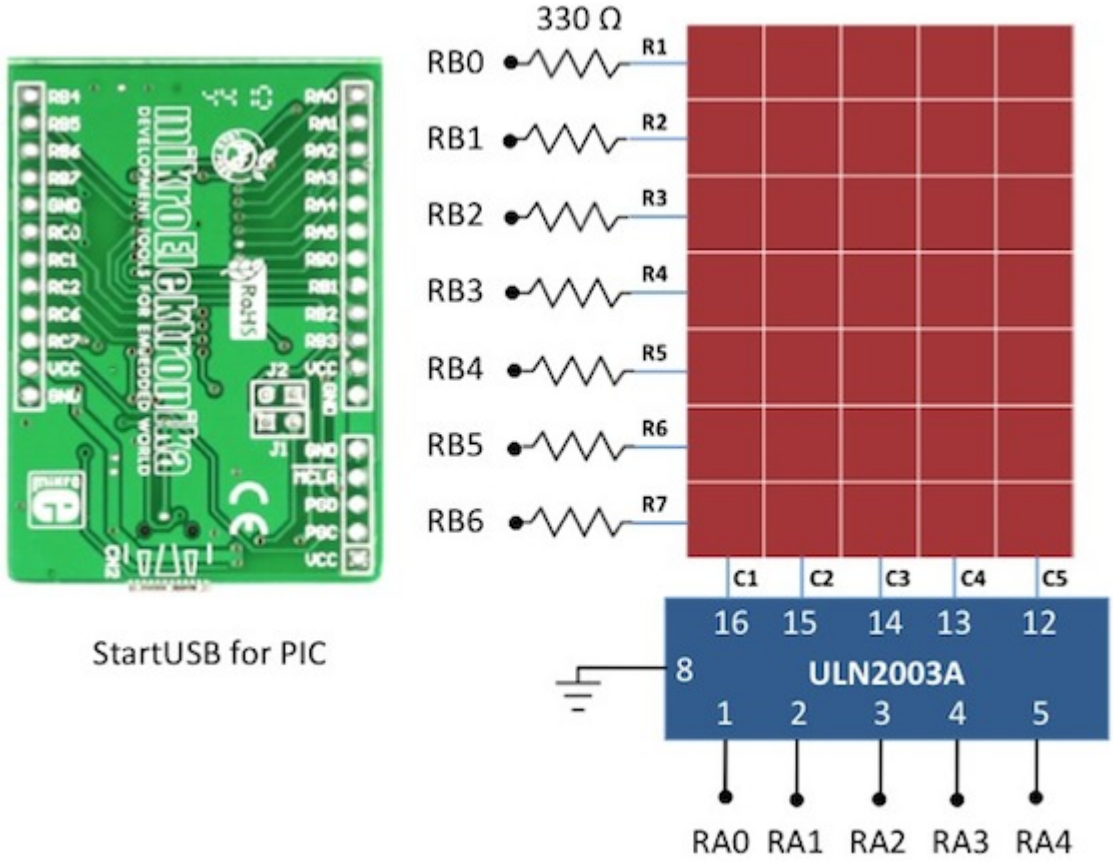
ULN2003A의 입력은 Active high입니다. 이 말은 해당 출력핀을 그라운드시키기위해서는 입력핀에 로직 high를 주어야 한다는 말입니다. ULN2003A내부의 Darlington 트랜지스터 어레이의 스키마틱은 아래와 같습니다.



ULN2003A의 핀다이어그램과 스키마틱 (Darlington transistor arrays)

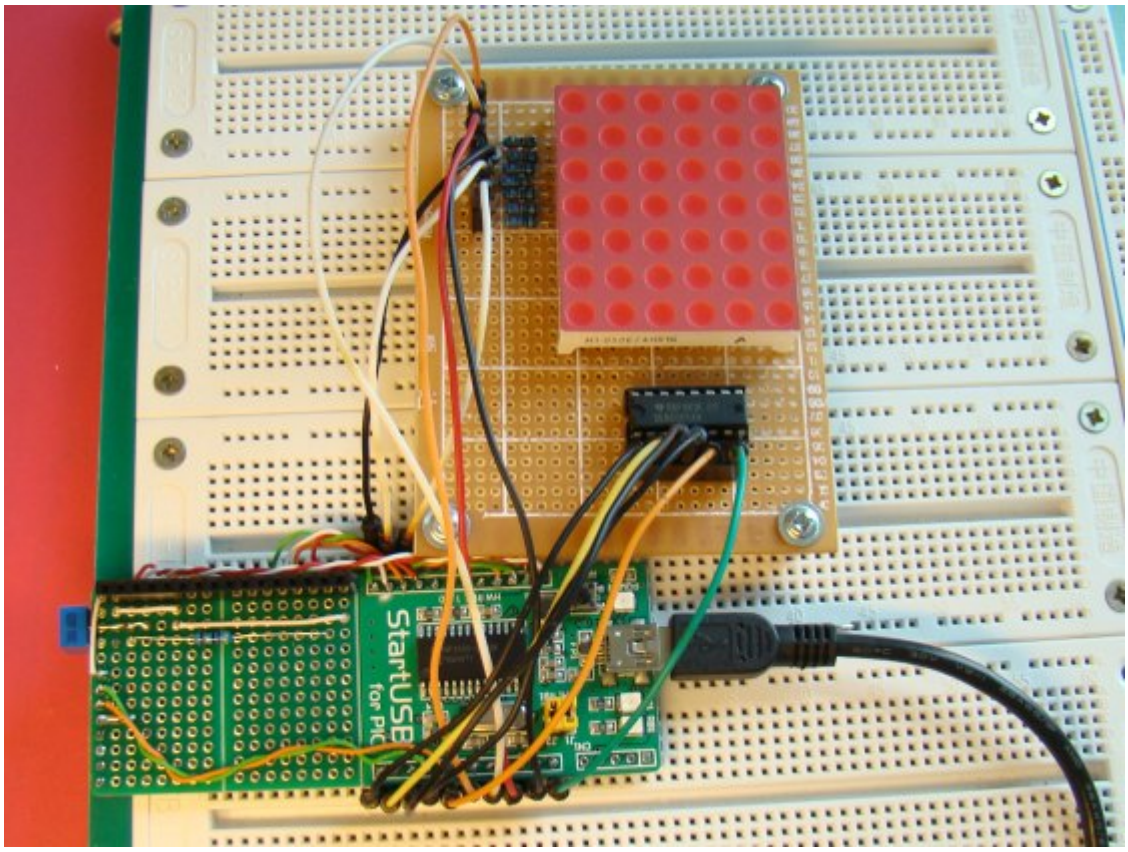
회로셋업

이 실험을 위한 회로의 셋업을 위해서는 7개의 330 Ω을 R1-R7과 직열로 연결하여 전류를 제한하여 주어야 하며 PIC18F2550의 RB0-RB6을 통해 행을 제어합니다. 열은 ULN2003A의 5개의 출력에 연결이 되고 해당하는 ULN2003A IC의 다섯개의 입력은 PIC18F2550의 RA0-RA4에에 의해 제어됩니다. MCU PORTA에 적절한 비트를 보냄으로써 열을 스캔할수 있습니다. 예를들어, RA0을 1로 셋팅하고 RA1-RA4을 클리어하면 첫번째 열이 선택이 될것입니다. MCU는 1ms정도 기다린뒤 다음 열으로 옮겨갑니다. 각각의 열에서 MCU는 원하는 LED를 켜기위해 PORTB에 해당 값을 출력하게 됩니다. 이렇게 행간 옮겨다니며 LED를 출력하는 속도는 사람의 눈에는 인식이 되지 않을 정도로 빨라 하나의 문자로 보여지게 됩니다.



.

5 X 7 LED 도트매트릭스와 StartUSB for PIC(PIC18F2550)을 연결하기 위한 회로도



StartUSB 개발보드와 LED도트 매트릭스 연결 회로의 셋업

소프트웨어

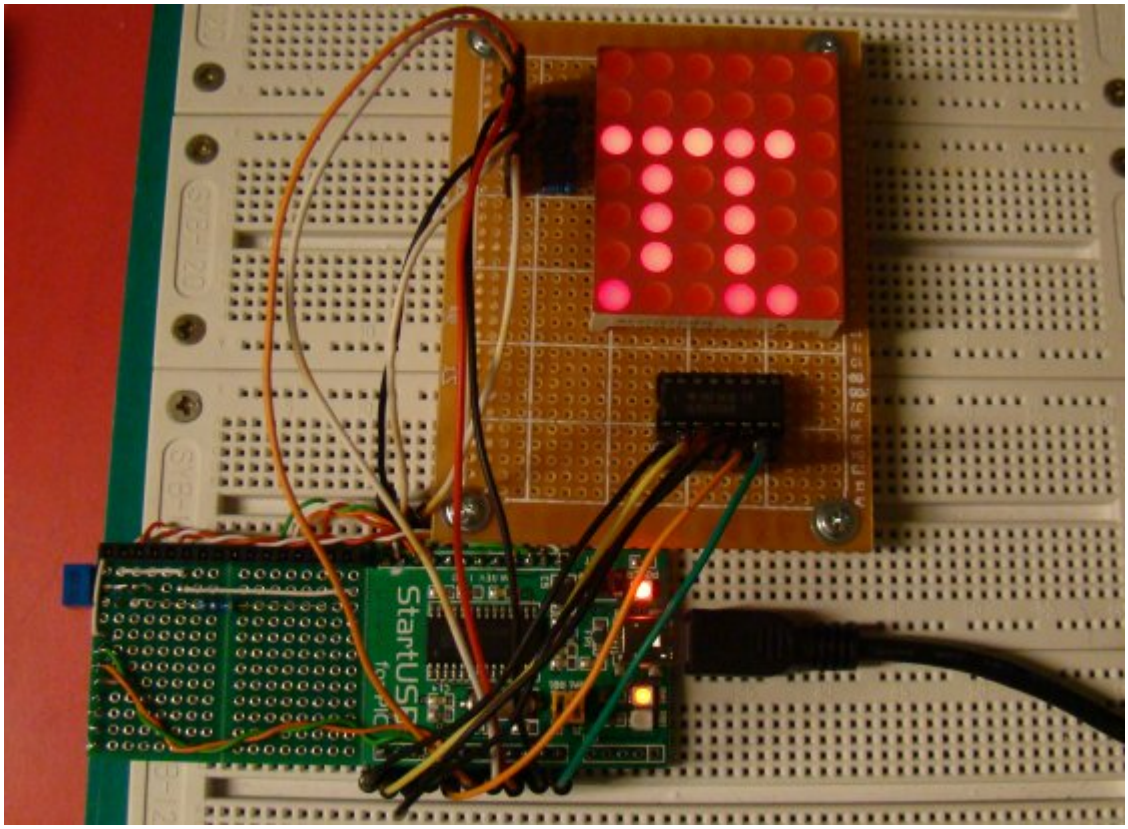
본 실험의 많은 부분은 사실 열을 스캔하고 적절한 값을 행에 주는 소프트웨어 루틴입니다. 문자를 디스플레이하기 위한 행별 열의 값은 램이나 램이 충분하지 않을 경우 프로그램 메모리에 저장할수 있습니다. mikroC 컴파일러에서는 변수들이 램에 저장되고 상수는 프로그램 메모리에 저장됩니다. PIC 마이컴이 충분한 램용량을 가지고 있지 않다면 constant array를 정의하여 행의 값을 프로그램 메모리에 저장할수 있습니다. PIC18F2550은 2KB의 램을 가지고 있어 여기서는 램에 알파벳 A부터 Z에 해당하는 행의 값을 저장하였습니다. 아래는 mikroC 컴파일러에서 행의 값을 정의한 것입니다.

```
unsigned short Alphabets[130]={ 0x7e, 0×09, 0×09, 0×09, 0x7e, // A
0x7f, 0×49, 0×49, 0×49, 0×36,  // B
0x3e, 0×41, 0×41, 0×41, 0×22,
0x7f, 0×41, 0×41,0×22, 0x1c,
0x7f, 0×49, 0×49, 0×49, 0×63,
0x7f, 0×09, 0×09, 0×09, 0×01,
0x3e, 0×41, 0×41, 0×49, 0x7a,
0x7f, 0×08, 0×08, 0×08, 0x7f,
0×00, 0×41, 0x7f, 0×41, 0×00,  // I
0×20, 0×40, 0×41, 0x3f, 0×01,
0x7f, 0×08, 0×14, 0×22, 0×41,
0x7f, 0×40, 0×40, 0×40, 0×60,
0x7f, 0×02, 0×04, 0×02, 0x7f,
0x7f, 0×04, 0×08, 0×10, 0x7f,
0x3e, 0×41, 0×41, 0×41, 0x3e,
0x7f, 0×09, 0×09, 0×09, 0×06,
0x3e, 0×41, 0×51, 0×21, 0x5e,
0x7f, 0×09, 0×19, 0×29, 0×46,
0×46, 0×49, 0×49, 0×49, 0×31,  // S
0×01, 0×01, 0x7f, 0×01, 0×01,
0x3f, 0×40, 0×40, 0×40, 0x3f,
0x1f, 0×20, 0×40, 0×20, 0x1f,
0x3f, 0×40, 0×30, 0×40, 0x3f,
0×63, 0×14, 0×08, 0×14, 0×63,
0×07, 0×08, 0×70, 0×08, 0×07,
0×61, 0×51, 0×49, 0×45, 0×43 // Z
};
```

아래와 같이 정의를 하면 프로그램 메모리에 해당값이 저장이 됩니다.

```
const unsigned short characters[30]={
0×24, 0x2A, 0x7f, 0x2A, 0×12, // $
0×08, 0×14, 0×22, 0×41, 0×00, // <
0×41, 0×22, 0×14, 0×08, 0×00, // >
0×14, 0×14, 0×14, 0×14, 0×14, // =
0×36, 0×49, 0×55, 0×22, 0×50, // &
0×44, 0x3c, 0×04, 0x7c, 0×44, // PI
};
```

알파벳 A부터 Z까지 디스플레이하는 간단한 프로그램을 mikroC 컴파일러를 이용하여 작성하여 보았습니다. 첨부된 프로젝트 파일을 참조하세요.



PI 기호 디스플레이

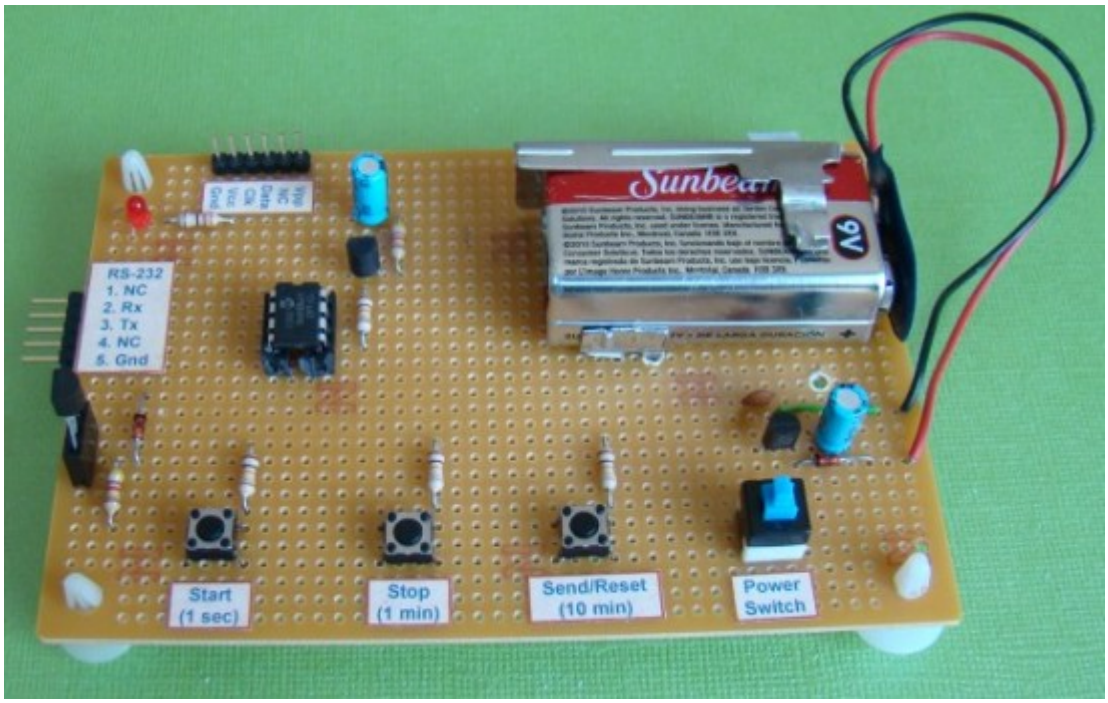
가치창조기술 | www.vctec.co.kr

PIC12F683 마이크로컨트롤러를 이용한 간단한 데이타 로거(data logger) 만들기

마이컴 실험실

2011/10/14 12:42 http://blog.naver.com/ubicomputing/150121372826

PIC12F683 마이컴을 이용한 간단한 데이타 로거를 만들어 보겠습니다. 마이컴은 온도값을 온도센서로 주기적으로 읽어 그 데이터를 EEPROM에 저장합니다. 저장된 온도값은 나중에 시리얼인터페이스를 통해 PC로 전송됩니다.



완성된 온도 데이터 로거 + 9V 배터리

이론

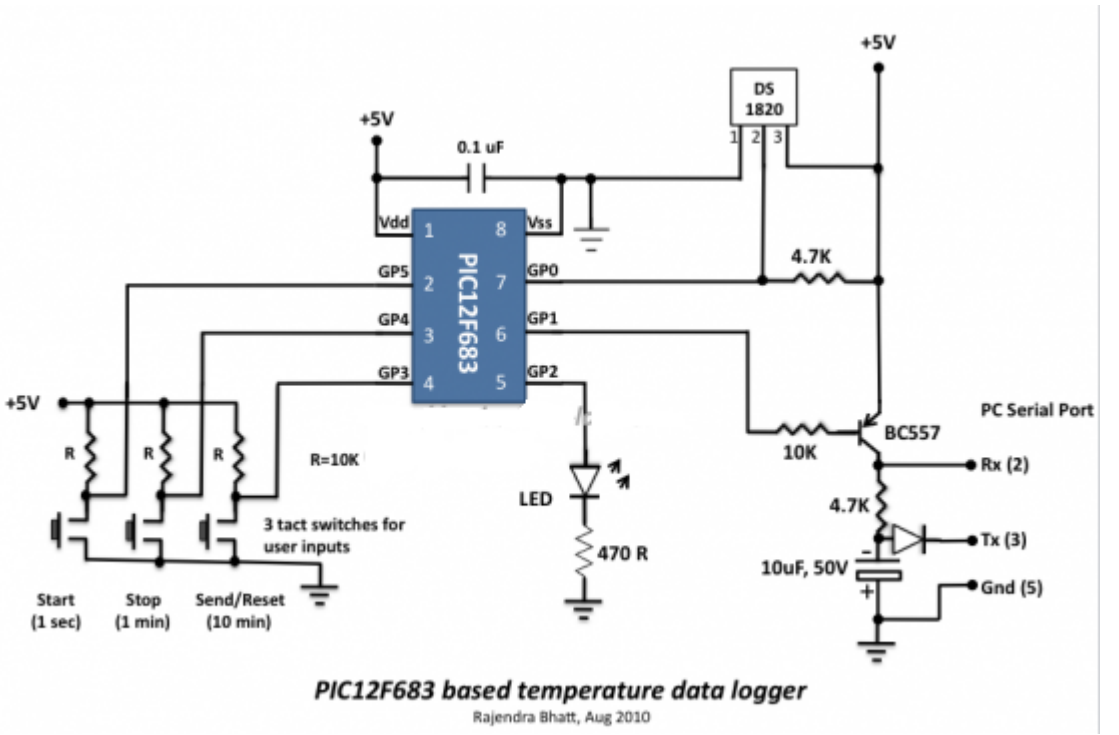
이 게시물에서 사용할 센서는 DS18B20으로 MAXIM사에서 생산된 제품으로 -55도~125도까지의 온도를 +- 0.5도의 정확성(-10도~85도)를 가지는 제품입니다. 센서는 원하는 분해능(0.5, 0.25, 0.125, 0.0625도)에 따라 출력을 9, 10, 11, 12비트 데이터로 설정할수 있습니다. 센서는 mcu와 1개의 와이어로 통신을 합니다. 자세한 내용은 MAXIM의 datasheet 를 참고하시고 이 센서는 세가지 버전(DS1820, DS18S20, DS18B20)이 있다는 것과 몇몇 구조적인 다른 점이 있다는 것을 알아 두시기 바랍니다.

PIC12F683은 256바이트의 EEPROM을 가지고 있습니다. 각각의 온도 데이터는 바이트 단위로 저장이되므로 DS18B20의 상위 8비트 출력만 저장이 됩니다. 이것은 결과적으로 온도 분해능을 1도로 떨어 뜨리게 되죠. 데이타 로거는 254 개의 온도 값(254바이트)을 EEPROM에 저장할수 있습니다. 나머지 두개의 바이트는 샘플링 시간과 샘플된 숫자 정보를 저

장하는데 사용하게 됩니다. 세개의 택트 스위치는 데이터로거의 동작을 제어하는 사용자 입력을 주기 위하여 사용이 됩니다.

회로도

PIC MCU는 4.0MHz의 내부 클럭를 사용합니다. DS18B20센서는 MCU의 GP0핀(7)에 연결이 됩니다. GP2핀에 연결된 LED는 데이터로거의 동작을 표시하는 디스플레이입니다. 예를 들면 샘플이 EEPROM에 레코딩 될때마다 LED를 깜박이는 것입니다. 회로는 9V배터리에서 LM78L05 레귤레이터를 사용하여 유도한 5V로 공급이 됩니다. LM78L05 회로는 매우 일반적인 회로이므로 여기서는 생략하겠습니다.



데이터로거 회로도

세개의 택트 스위치는 다음과 같은 기능을 가집니다.

- Start: 데이터로깅 시작
- Stop: 데이터로깅 중지
- Send/Reset: 시리얼포트를 통해 데이터를 PC에 전송. 2초이상 누르고 있을 경우 EEPROM초기화.

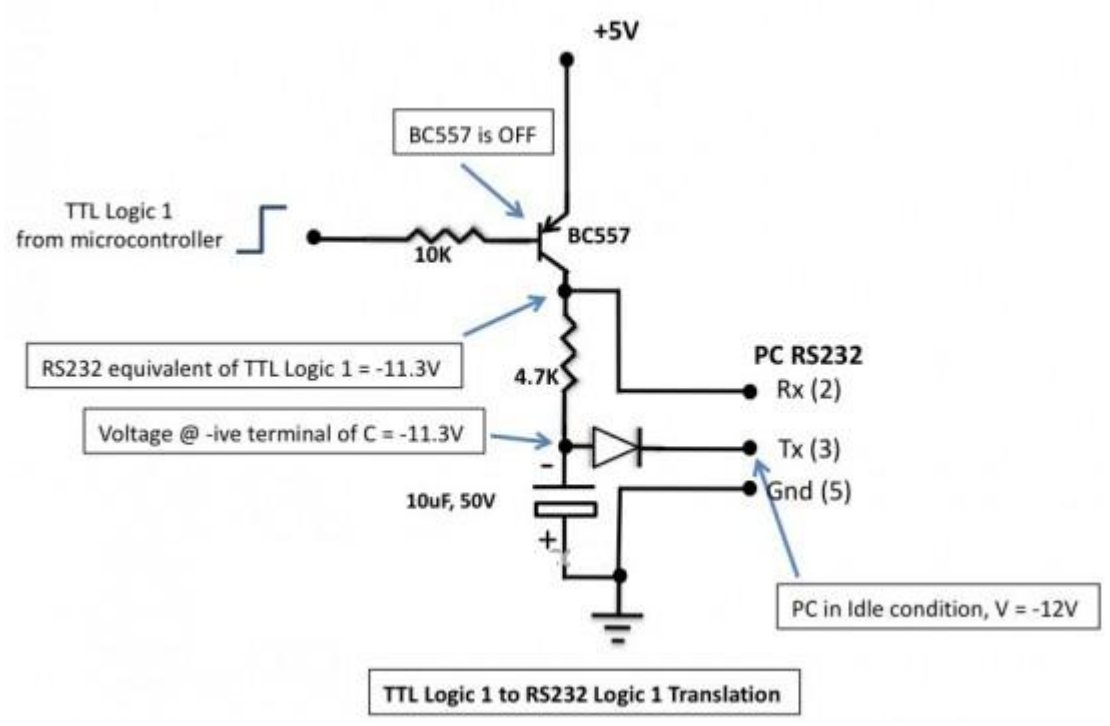
샘플링 시간 선택

이 데이터 로거는 1초, 1분, 10분의 세개의 샘플링 구간 옵션을 제공합니다. 이 옵션의 선택은 스위치를 통해서 할수 있습니다. 만약 10분간격의 샘플링 시간이 필요하다면, 전원을 끄고 "Send/Reset버튼을 누른상태에서 전원을 켭니다. LED가 켜질때가지 기다렸다가 LED가 켜지면 누른 버튼에서 손을 땝니다. 그러면 샘플링 시간이 10분으로 설정이 됩니다. 이 설

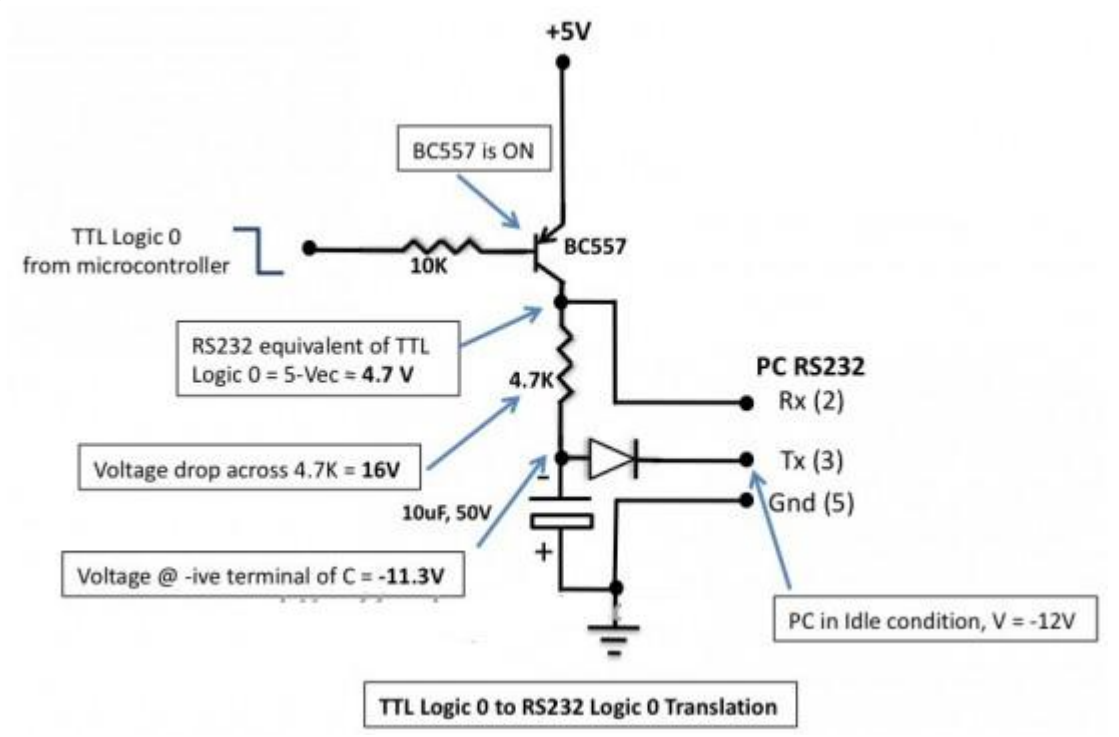
정은 EEPROM의 0번 주소위치에 저장이되어 전원이 없어지더라도 이전에 설정한 샘플링 시간을 되살릴수 있습니다. "Send/Reset"버튼 대신 "Start"나 "Stop"버튼을 누르고 있을 경우 각각 1초, 1분의 기간이 설정됩니다. 10분씩 샘플링 하도록 설정할 경우 이 데이타 로거는 42시간동안 온도를 샘플링할수 있게 됩니다.

PC와의 시리얼 인터페이스

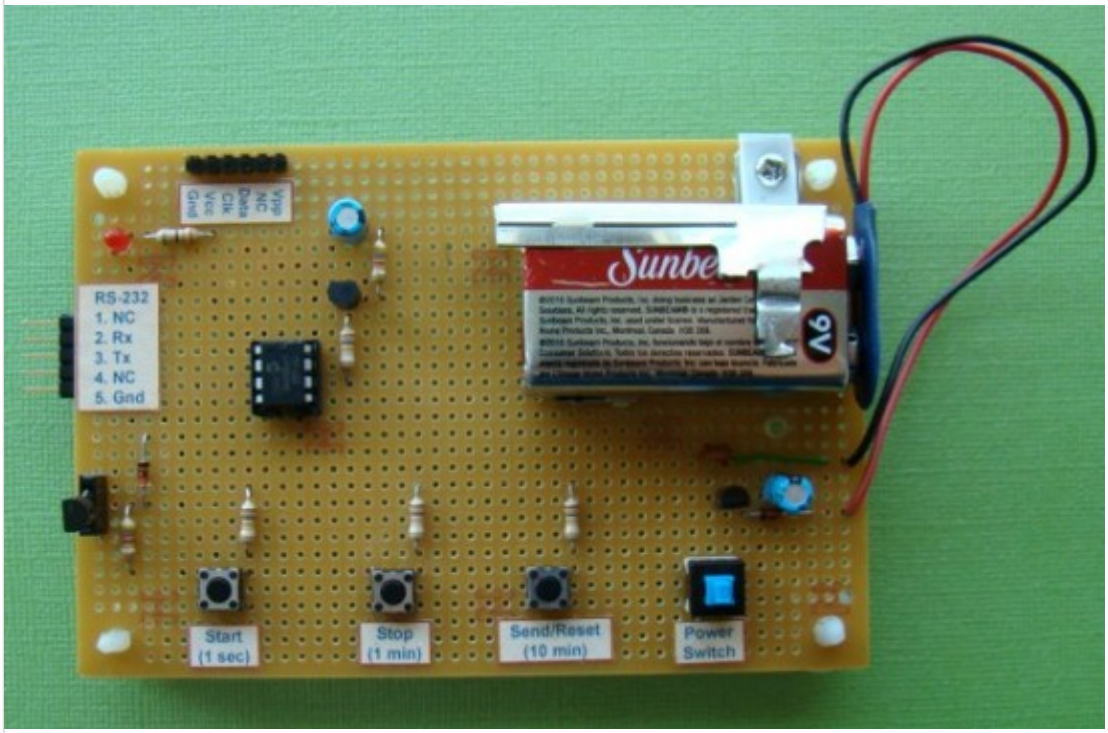
시리얼포트를 통해 데이터를 PC로 전송하는 것은 MCU에서 나오는 TTL로직레벨을 RS232 전압레벨로 변환하여 주는 전압변환회로가 필요합니다. PNP트랜지스터와 몇몇 수동소자를 이용하여 전압변환이 가능합니다. RS232표준은 -3V~-12V를 로직1로 사용하고 +3V~+12V를 로직0으로 사용합니다. 필요한 네가티브 전압은 PC쪽 RS232포트의 TX 핀에서 가져올수 있습니다(PC에서 PIC12F683으로 데이터를 전송하지 않으므로). 대기상태에서 PC쪽의 TX핀은 HIGH(-12V)입니다. 아래의 그림은 TTL레벨의 1과 0을 RS232레벨의 1과 0으로 변환하는 동작을 보여줍니다. 10uF 캐패이시터의 양극은 음극이 좀더 많은 음전압을 다루게 하기 위해 그라운드 되었습니다.



TTL로직 1을 RS232 로직1로 변환



TTL로직0을 RS232로직 0으로 전환



완성된 프로젝트 보드

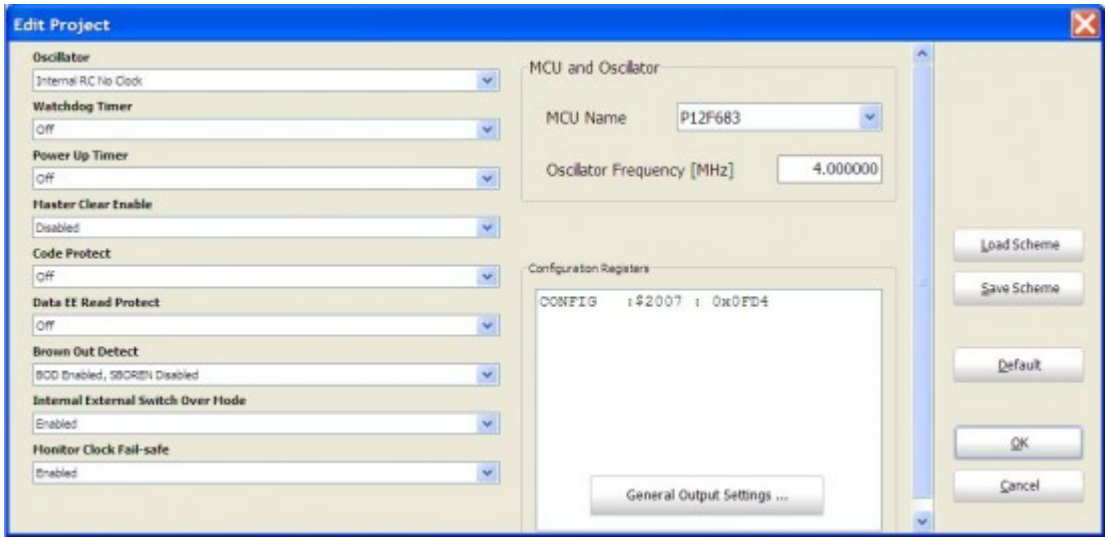
소프트웨어

PIC12F683의 펌웨어는 mikroC 컴파일러로 개발되었습니다. PIC12F683은 내장 하드웨어 UART모듈을 가지고 있지 않지만 mikroC 컴파일러는 PIC 마이컴의 디지털 I/O핀을 통하여 구현된 소프트웨어 UART용 내장 라이브러리 루틴이 있습니다. DS18B20센서가 많이 사용이 되기 때문에 mikroC 컴파일러도 이 센서를 제어하기 위해 1-wire 라이블러리 루틴을 제공하고 있습니다. 1-wire 센서와 통신하기 위한 내장 함수들은 다음과 같습니다.

- Ow_Reset is used for reseting sensor;
- Ow_Read is used for receiving data from sensor; and
- Ow_Write is used for sending commands to sensor.

mikroC 컴파일러는 내장 EEPROM을 읽고 쓰는 연산을 하기 위해 EEPROM 라이브러리 역시 지원합니다. 이러한 mikroC 컴파일러의 내장라이브러리를 이용하면 전체적인 프로그램이 훨씬 쉽게 됩니다. 소스코드는 첨부파일에서 다운받을 수 있습니다.

PIC12F683의 configration bit 셋업은 mikroC 컴파일러의 Edit Project창에서 할수 있습니다. 아래가 필요한 셋팅입니다.

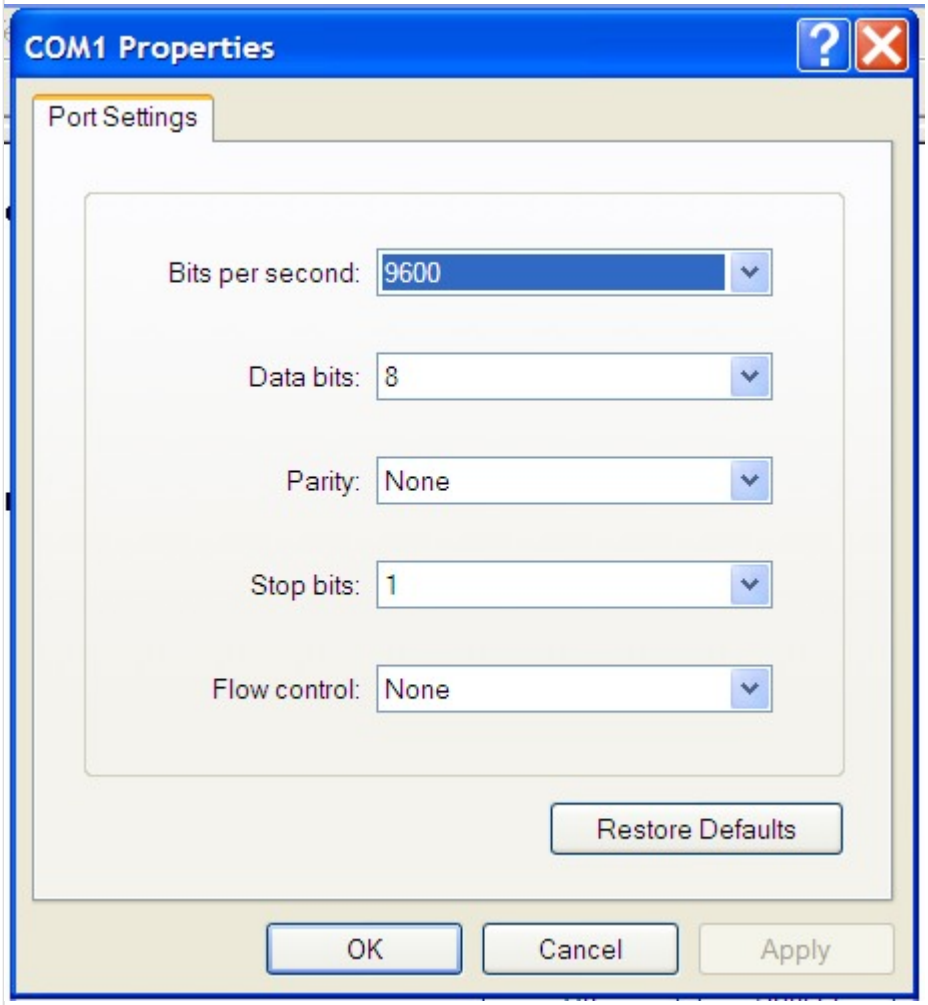


프로그래밍 시퀜스

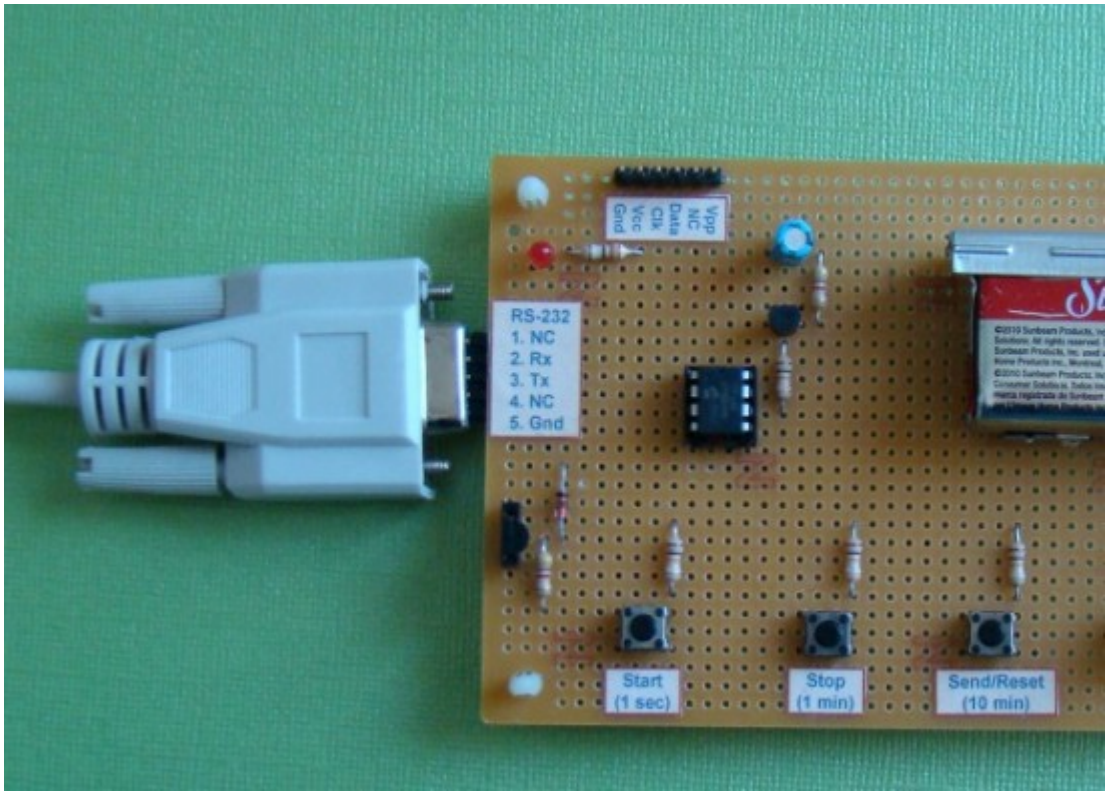
회로에 전원이 들어오면, LED가 3번 깜빡이면서 데이터로거에 전원이 들어오고 초기화 되었다는 것을 알려줍니다. 프로그램은 샘플인터벌을 셋팅하기 위한 키가 눌린 것이 있는지 체크하고 있다면 어떤 키가 눌렸는지 확인하고 그 키에 맞는 샘플링 주기를 EEPROM 0번 주소위치에 저장합니다. LED가 ON이 되면서 샘플링 주기가 설정되었다는 것을 알려주면 버튼을 릴리즈하여 줍니다. 프로그램의 실행은 이제 메인 루프안에 위치하게 됩니다.

세개의 스위치는 Interrupt-on-change 모드에서 동작합니다. 이말은 버튼이 눌리면 언제나 인터럽트가 생성된다는 의미입니다. Start버튼을 누르면 데이타 레코딩이 시작됩니다. 온도 샘플이 매 주기마다 EEPROM에 저장되고 LED는 로깅이 되고 있다는 것을 깜빡이며 알립니다. Stop버튼을 누르면 샘플링이 멈추어지는 반면 Send버튼은 저장된 샘플링 값을 GP1핀(6)을 통해 시리얼 전송하게 합니다. 만약 Send버튼이 2초 이상 눌려있다면 EEPROM은 완전히 클리어 될것입니

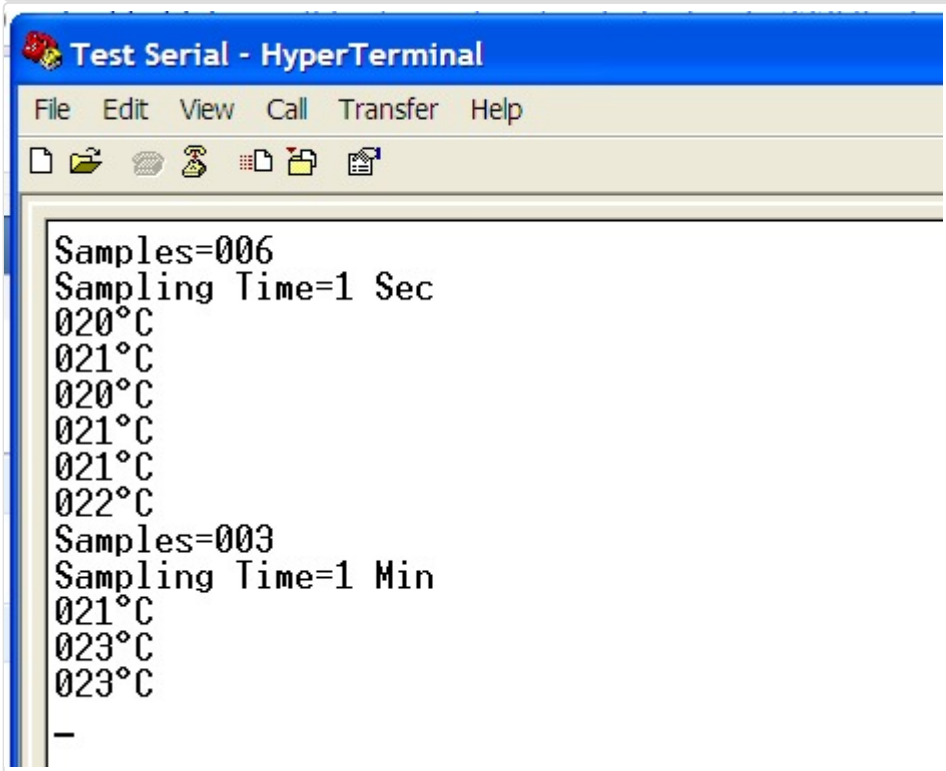
다. PC쪽에서는 하이퍼터미널을 이용하여 PIC12F683에 저장된 데이터를 받아 볼수 있습니다. 시리얼 데이터 전송속도는 9600이며 아래의 화면은 하이퍼터미널의 셋팅을 보여줍니다.



PPC의 하이퍼터미널 셋팅



시리얼 포트 연결



하이퍼터미널에서 수신된 데이터

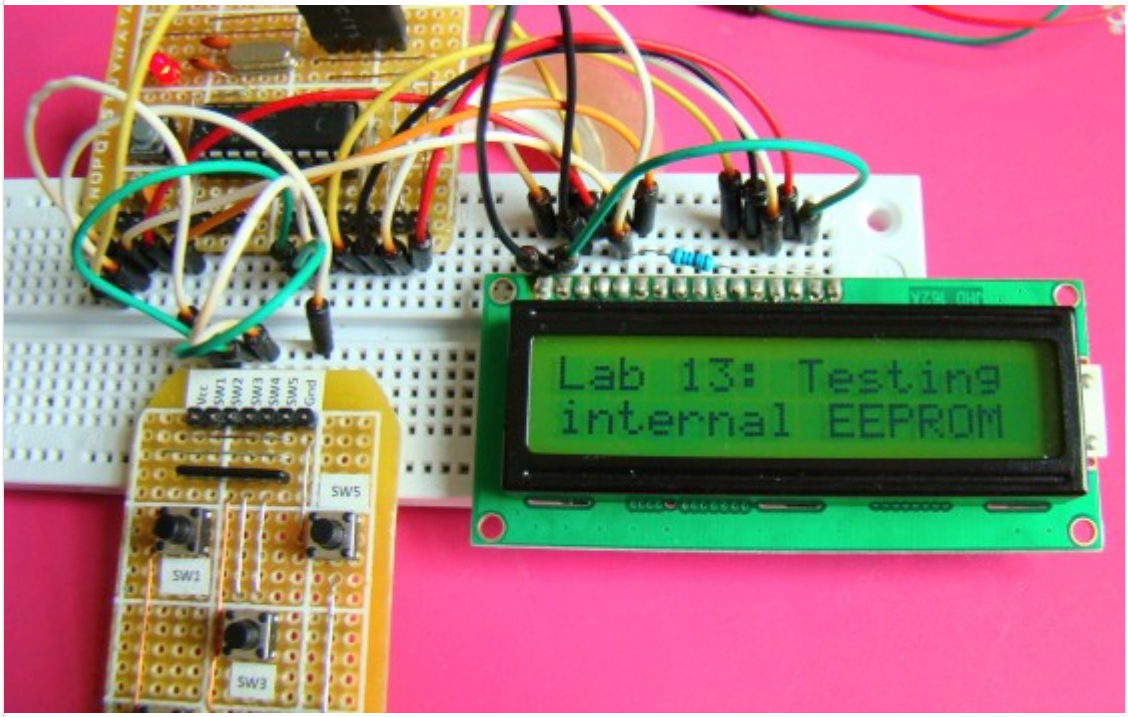
가치창조기술 | www.vctec.co.kr

[PIC 마이컴 실험하기] 13. 내부 EEPROM 읽고 쓰기

마이컴 실험실

2011/10/17 10:56 http://blog.naver.com/ubicomputing/150121609385

EEPROM (*Electrically Erasable Programmable Read-Only Memory*) 은 비활성 메모리의 한종류로 메모리가 보드상에 있을때에도 프로그래밍하고 지울수 있는 메모리입니다. 대부분의 PIC 마이컴은 내장 EEPROM을 가지고 있어 전원이 들어오지 않아도 데이터를 보관할수 있는 좋은 장소를 가지고 있다 하겠습니다. 이전 게시물 "PIC12F683 마이크로컨트롤러를 이용한 간단한 데이타 로거(data logger) 만들기"에서 PIC12F683의 내장 EEPROM을 측정된 온도값을 저장하는데 사용하였습니다. 이번 게시물에서는 PIC16F628A의 내부 EEPROM을 어떻게 읽고 쓰는지 살펴보도록 하겠습니다.



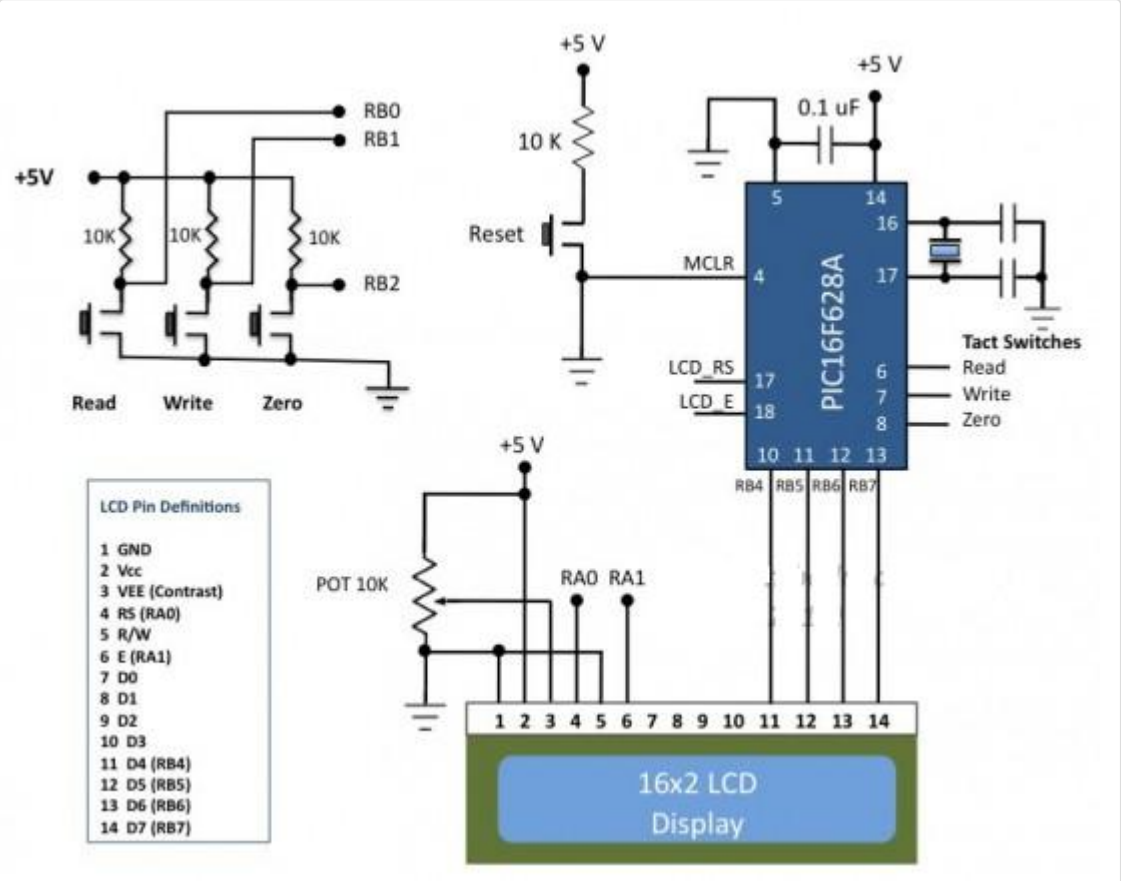
PIC 마이컴 내장 EEPROM을 읽고 쓰기

이론

PIC16F628A 마이컴은 00h부터 7Fh까지의 128바이트의 EEPROM을 가지고 있습니다. EECON1, EECON2, EEDATA, EEADR이라는 4개의 SFR레지스터가 있어서 이 메모리를 읽고 쓰는데 사용이 됩니다. EEADR은 읽고 쓰고 싶은 EEPROM의 주소를 저장하고, EEDATA는 EEPROM에서 읽은 데이터나 EEPROM에 쓸 데이터를 저장하는데 사용이 됩니다. 여기서는 이러한 레지스터에 대한 자세한 설명을 건너뛸 것인데 이는 우리가 사용하는 mikroC컴파일러가 이러한 레지스터를 설정하여 주는 내장 EEPROM라이브러리 루틴을 가지고 있기때문입니다. 이 레지스터에 대해 자세히 알고 싶다면 PIC16F628A DATASHEET를 참고하세요.

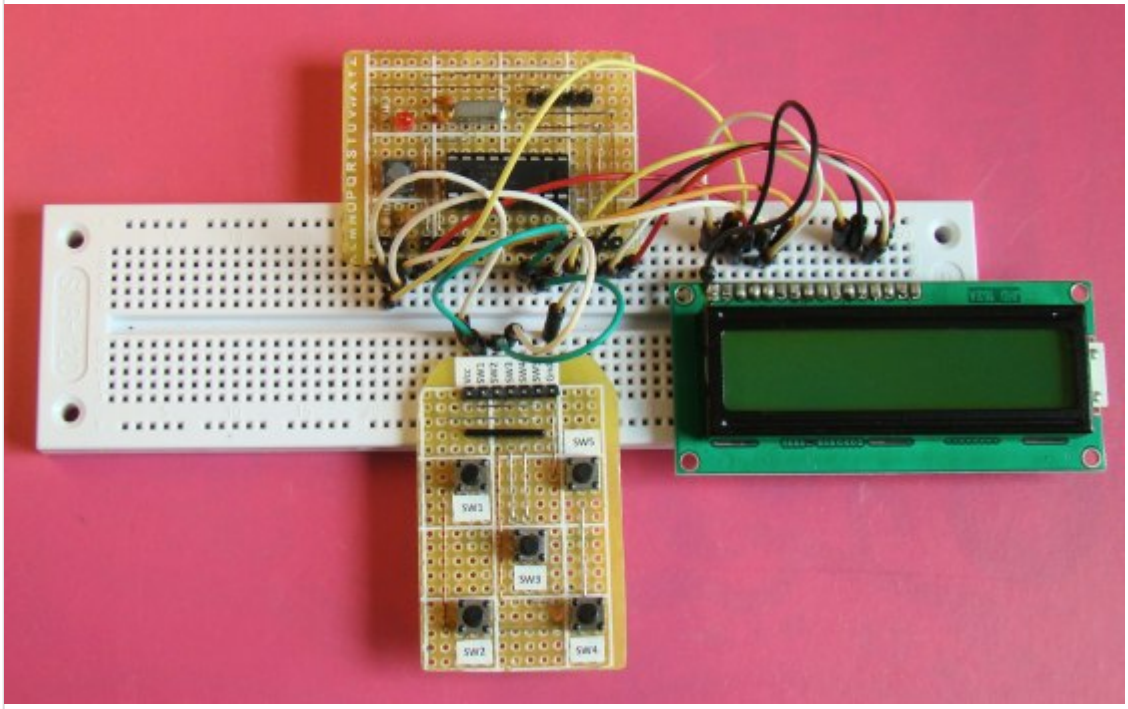
회로셋업

본 실험은 16 X 2 character LCD와 입력을 위한 세개의 스위치가 필요합니다. LCD, 스위치, PIC16F628A를 연결하여 주는 회로도는 아래와 같습니다.



회로도

세개의 택트 스위치는 RB0, RB1, RB3핀에 연결됩니다. LCD는 4비트모드에서 작동하며 데이타 핀은 RB4-RB7핀에 의해 동작됩니다. LCD RS와 E 컨트롤 핀은 RA0와 RA1핀에 연결됩니다. 세개의 택트 스위치에 대한 기능은 밑에서 좀더 언급하겠습니다.



브레드보드에 회로도 셋업

소프트웨어

PIC16F628A의 내부 EEPROM(00h-0Fh)에 데이터를 읽고 쓰고 지우는 연산을 하는 간단한 어플리케이션을 작성하여 보도록 하겠습니다. Read버튼이 눌리면 00h-0Fh에 저장되어 있는 16바이트의 정보를 읽어 LCD에 디스플레이 표시하고 Write버튼을 누르면 문자열을 저장하고 반면에 Zero버튼이 눌리면 저장된 데이터를 0으로 대치하여 보도록 하겠습니다.

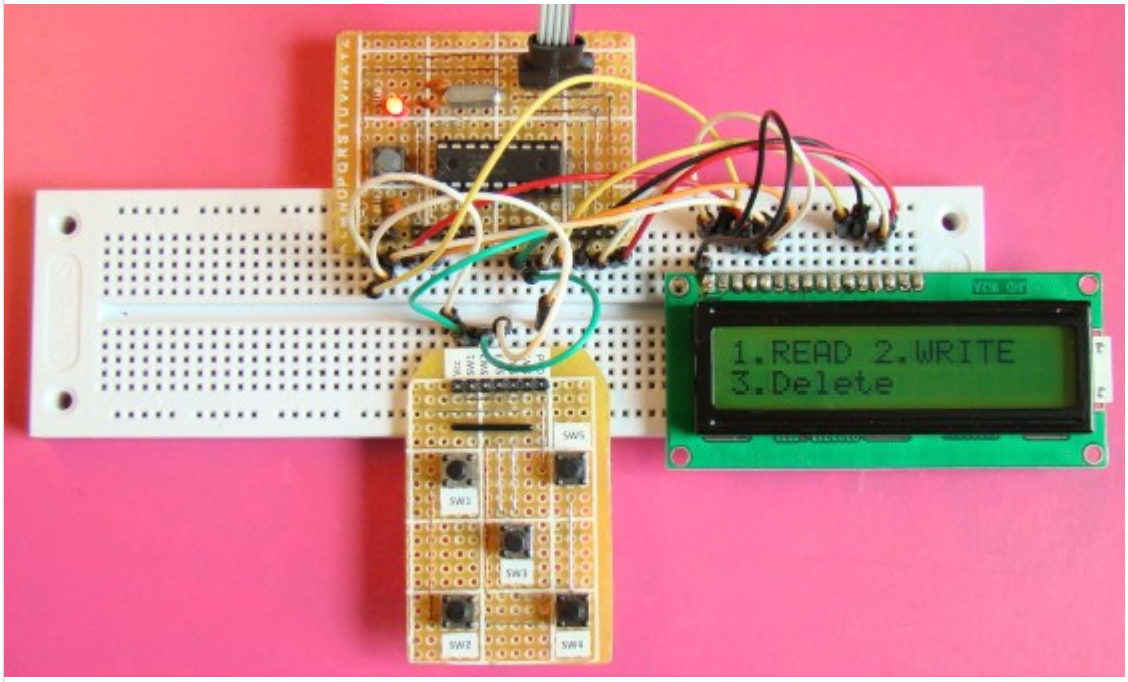
mikroC컴파일러는 EEPROM을 읽고/쓰기를 하기 우해 아래의 함수를 지원합니다.

unsigned short EEPROM_Read(**unsigned short** address) :- returns a byte from the specified address
void EEPROM_Write(**unsigned short** address, **unsigned short** data) :- writes the data at the specified address

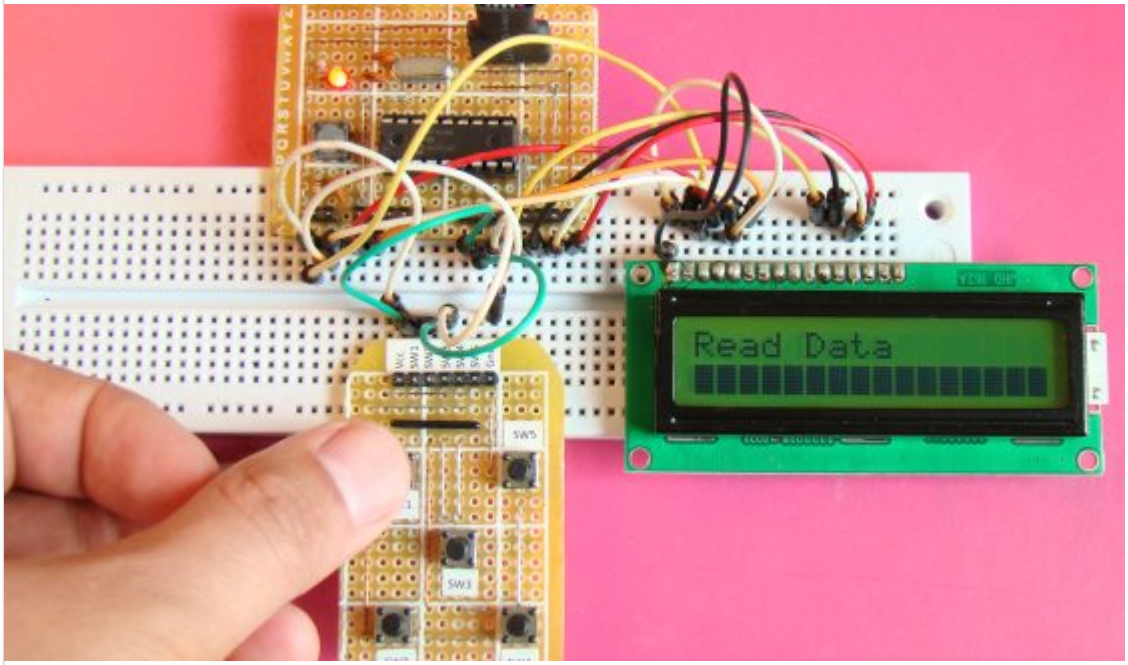
프로젝트 파일은 첨부된 파일에서 다운로드 받을 수 있습니다.

결과

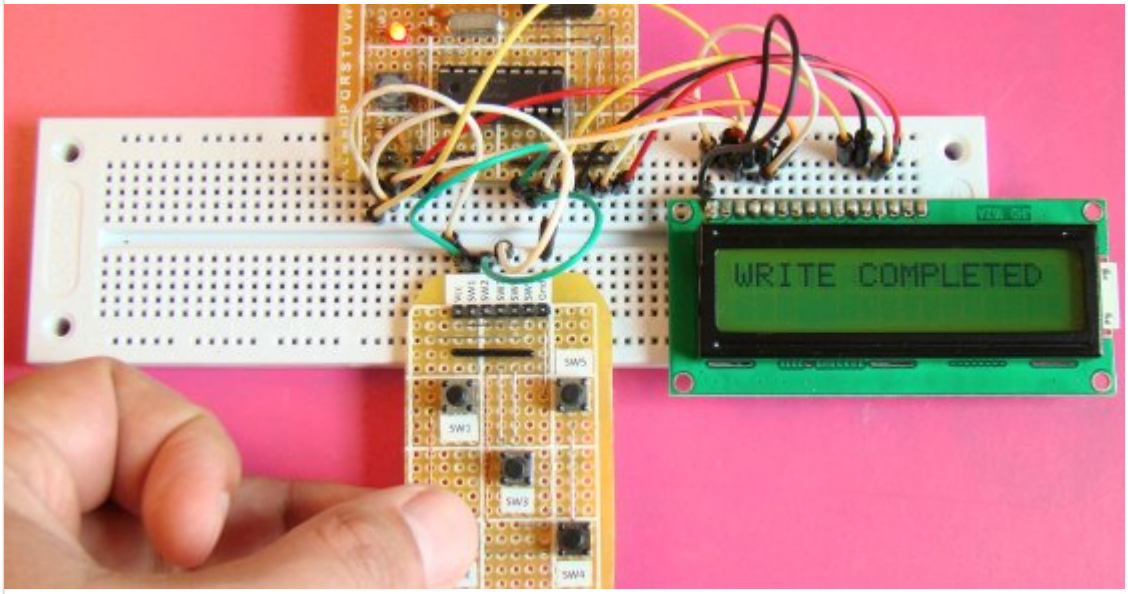
데이터가 없는 EEPROM안의 기본값은 모두 FFh(255)입니다. FFh값이 LCD에 표시될때 5X10 도트들이 보여집니다. 그래서 맨처음 EEPROM을 읽을 때 아래의 그림과 같이 FFh값을 읽게 됩니다. 쓰기 연산을 수행해 보고 전원을 끄고 몇초뒤에 다시 켜지면 Read버튼을 누르면 전원을 꺼도 데이터가 사라지지 않고 읽혀지는 것을 볼수 있습니다.



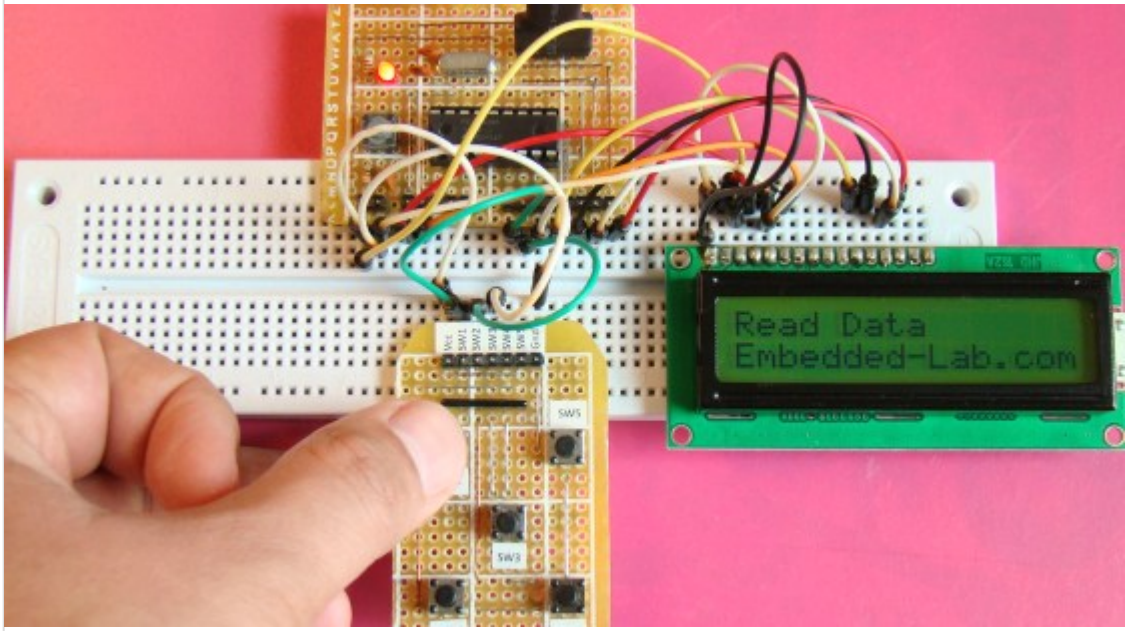
메인 메뉴



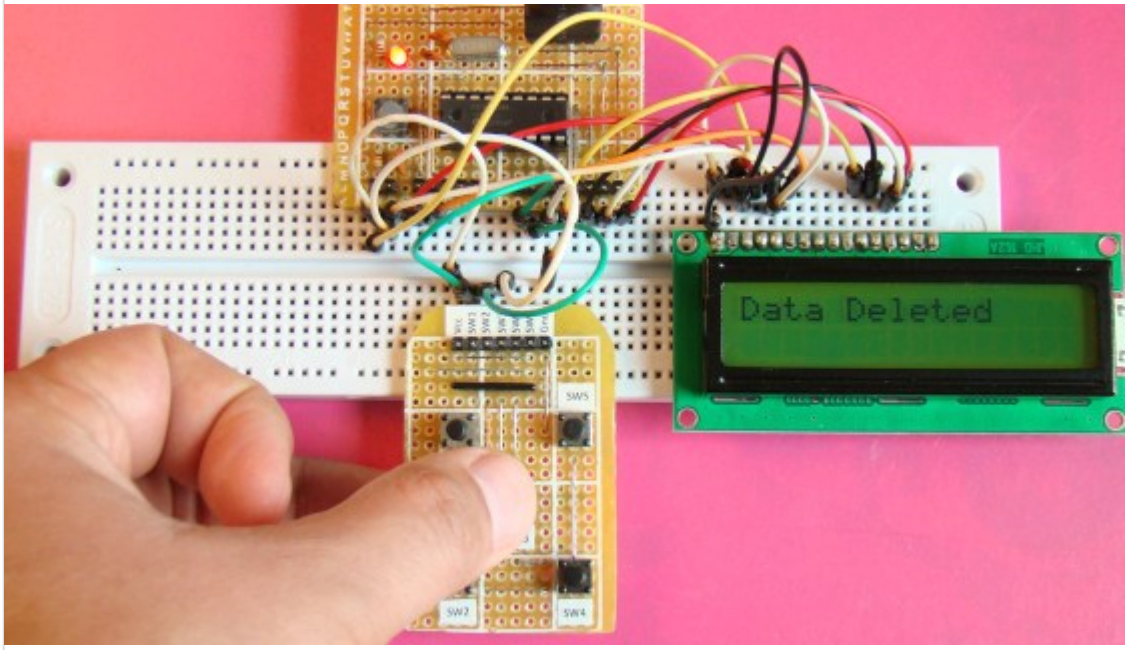
맨처음 FF 데이터를 읽기



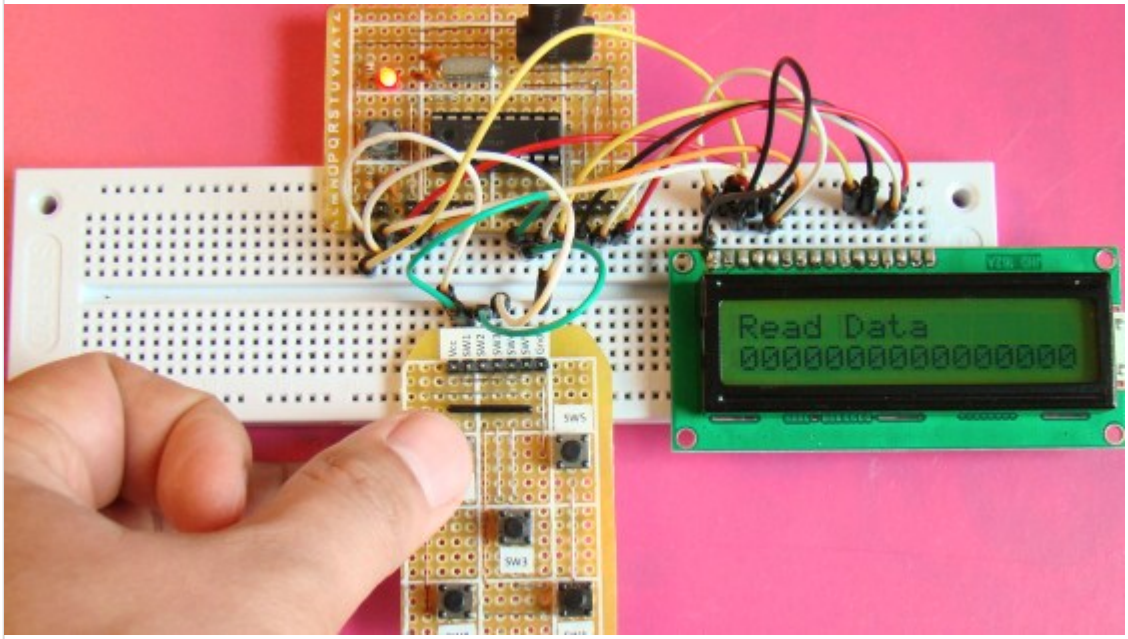
쓰기 연산



저장된 값을 다시 읽음



저장된 값을 0으로 대치



제로 값 읽기

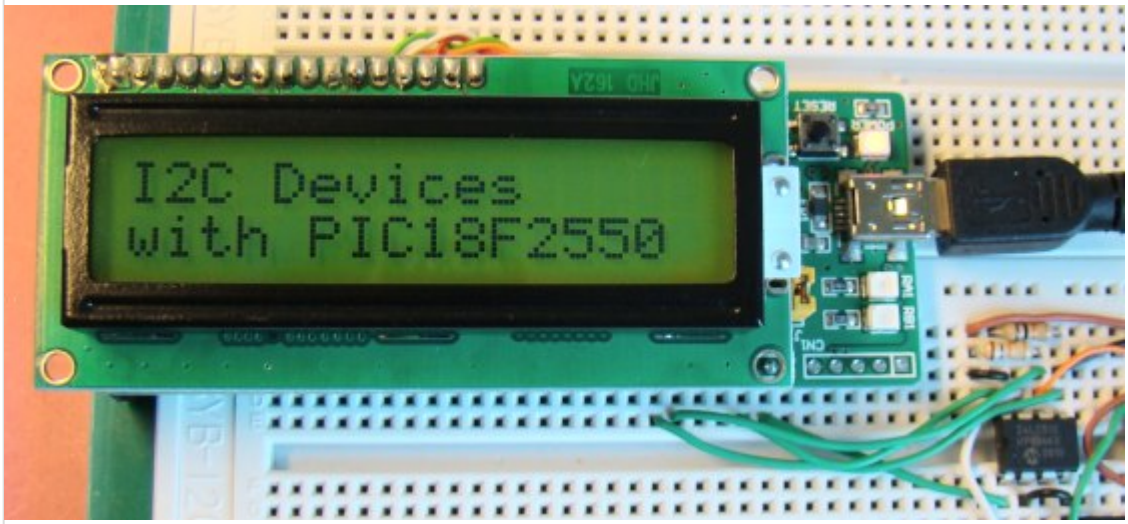
가치창조기술 | www.vctec.co.kr

[PIC 마이컴 실험하기] 14. I2C 통신(PIC18F2550과 DS1631온도센서, EEPROM의 통신)

마이컴 실험실

2011/10/18 14:51 http://blog.naver.com/ubicomputing/150121713210

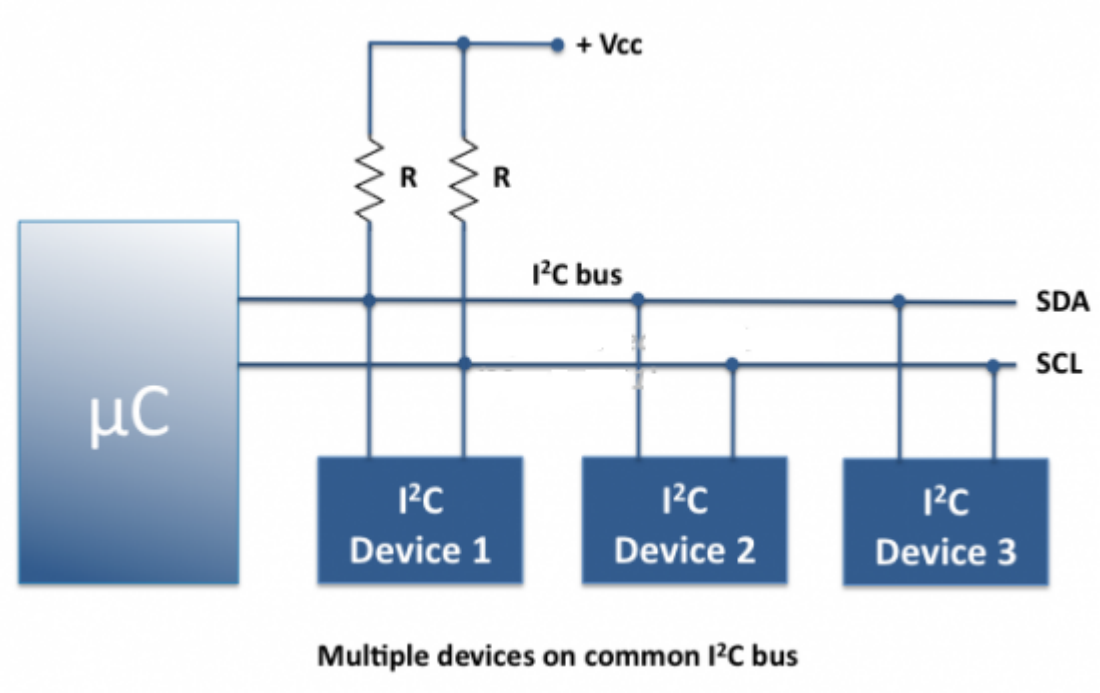
I2C (Inter-Integrated Circuit)은 잛은 통신거리용 시리얼 인터페이스로 데이터 전송을 위해 오직 2개의 버스라인만이 필요합니다. 1980년대 필립스에 의해 개발되었는데 원래 TV 보드상의 cpu와 주변 장치가 쉽게 통신을 할 수 있게 하기 위해 개발되었습니다. 오늘날에는 많은 임베디드 시스템에서 컨트롤러와 저속 주변장치(외장 EEPROM, 디지털 센서, LCD 드라이버 등)과 통신용도로 사용이 되고 있습니다. 이번 게시물에서는 I2C 통신에 대해서 전반적으로 살펴보고 PIC 마이컴에서는 어떻게 구현이 되는지, 그리고 I2C버스상에서 한개 혹은 여러개의 장치를 연결하는 방법에 대해 살펴보겠습니다. 여기서는 두개의 I2C EEPROM칩(24LC512)과 I2C호환 온도센서(DS1631)을 PIC18F2550 마이컴과 연결하여 보도록하겠습니다.



PIC마이컴과 I2C

이론

I2C버스는 SDA(Serial data)라인과 SCL(Serial Clock Line)라인의 두개의 라인을 가지고 있습니다. 한개의 장치에서 보내진 데이터는 SDA라인을 통해 다른 장치로 보내지고, 반면에 SCL라인은 데이터 전송에 필요한 동기화 클럭을 제공합니다. I2C 버스상에 있는 장치는 마스터 아니면 슬레이브장치입니다. 오직 마스터많이 데이터 전송을 시작할수 있고 슬레이브는 마스터에 응답하는 구조입니다. 한개의 공통 버스에 여러개의 마스터를 가지는 것이 가능하지만 동시에는 한개의 마스터만이 사용될수 있습니다. SCL 클럭라인은 항상 마스터에 의해서 동작합니다. 이 게시물에서는 마스터가 한개인 경우만 다룰 것이며 이 마스터는 PIC18F2550 마이컴이 됩니다. 아래의 그림은 한개의 마스터와 세개의 슬레이브장치가 있는 I2C 버스를 보여줍니다. 슬레이브는 데이터 전송을 먼저 시작할수 없고 항상 마스터에 의해 제어 받습니다.



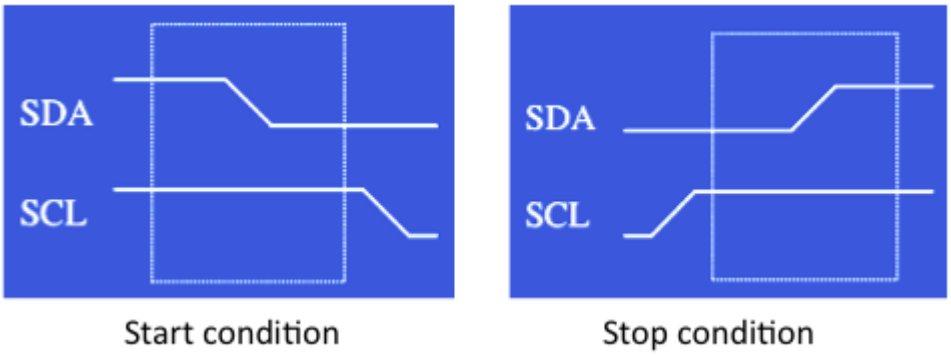
Multiple devices on common I²C bus

SCL과 SDA라인은 둘다 오픈 드레인 드라이버입니다. 그래서 풀업 레지스터를 통해 양전압 전원에 연결이 되어 있어야 합니다. 이것은 I2C 장치는 라인을 LOW만 시킬수 있고 HIGH시킬수 없다는 의미입니다. 아무 장치도 라인을 LOW하지 않으면 풀업레지스터를 통해 HIGH상태가 될것입니다. 이것이 I2C통신에서 풀업 레지스터가 중요하나 이유입니다.

I2C장치의 오픈 드레인 출력은 버스상에서 wired-AND를 수행하는데 도움을 줍니다. I2C버스상의 데이터는 표준모드에서 100Kbps의 속도로 전송이 가능하며, fast mode에서 400kbps, high-speed mode에서 3.4Mbps가 가능합니다.

Start와 Stop 조건

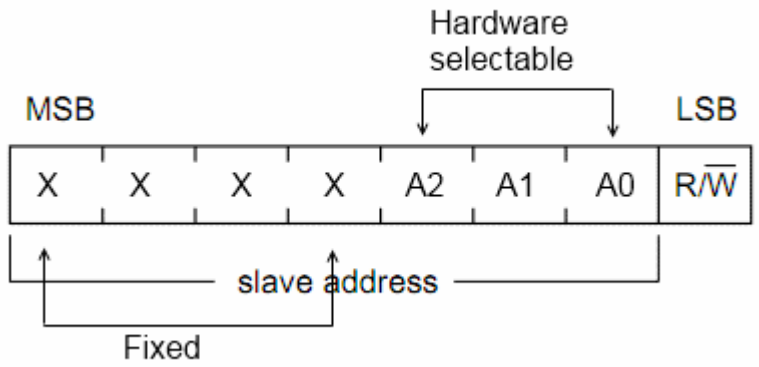
버스상에서 데이터 전송을 시작하기 전에 마스터는 모든 슬레이브 장치에 Start 조건을 만들어 버스상에 무엇인가가 전송이 될것임을 알립니다. 그러면 모든 슬레이브 장치는 어떤 명령이 올지 시리얼 라인을 듣고 있게 됩니다. Start조건은 SDA라인을 low시킴으로 만들수 있습니다. 이후 데이터 전송을 시작하고 끝나면 마스터는 stop조건을 다른 장치들에게 보내 버스를 릴리즈 하게게 됩니다. 스탑 조건을 보내는 신호는 SCL라인과 SDA라인을 순차적으로 릴리즈함으로 만들수 있습니다. 풀업저항에 의해 릴리즈시 HIGH로 되므로 START, STOP 시퀜스는 슬레이브는 데이터전송의 시작과 끝을 표시하여 줄수 있습니다.



START, STOP 조건의 신호

I2C device addressing

버스에 연결되어 있는 각각의 장치는 7비트나 10비트의 주소로 접근이 가능합니다. 10비트 주소체계는 잘 사용이 되지 않으므로 여기서는 생략하도록 하겠습니다. 스타트 조건 이후에 첫번째 바이트는 CONTROL 바이트로 알려져 있습니다. 컨트롤 바이트의 처음 7비트는 슬레이브의 주소로 이루어져 있는 반면, 8번째 비트(LSB)는 데이터방향(Read/Write)을 알려주는 비트입니다. 만약 8번째 비트에 0이 있을 경우 마스터가 선택한 슬레이브에 정보를 쓰겠다는 것을 표시합니다. 반면에 1일 경우는 마스터가 슬레이브로부터 데이터를 읽겠다는 것을 의미합니다. 7비트체계에서는 앞의 4개의 비트는 고정이 되고 나머지 3개 비트가 하드웨어 어드레스 핀(A0, A1, A2)에 의해 셋팅이 됩니다. 이는 사용자가 I2C버스상에서 8개의 장치 작동시킬수 있다는 의미이기도 합니다. 이 핀들은 VCC에 연결되어 HIGH이거나 GND에 연결되어 LOW됩니다.

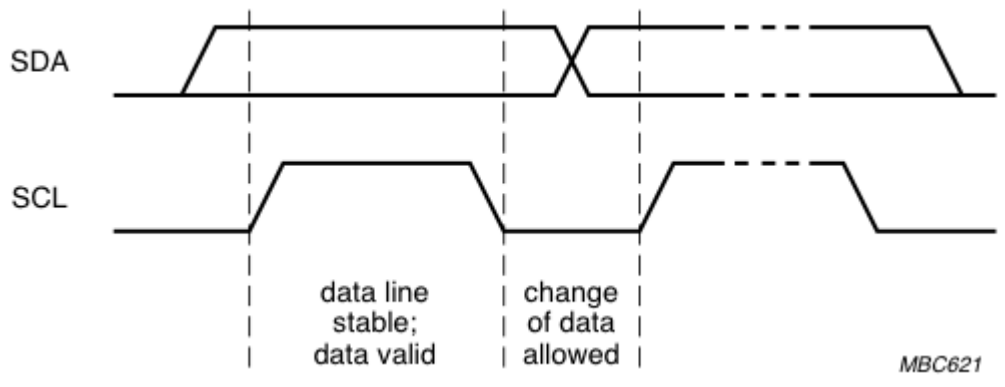


컨트롤바이트는 슬레이브주소를 포함하고 있다.

컨트롤바이트가 보내질때에, 시스템의 각각의 장치는 첫 7비트를 자신의 주소와 비교하여 일치할 경우, 8번째 비트에 따라 자신을 슬레이브-리시버, 슬래이브-트랜스미터로 인식합니다.

데이터 전송

SDA라인에 보내지는 모든 데이터 바이트는 8비트 길이 여야 합니다. SDA라인에 보내지는 데이터는 MSB(Most Siginificant Bit)를 먼저 보내고 SCL라인은 동기화 클럭을 생성합니다. SDA라인 상의 데이터는 SCL이 HIGH일경우에만 유효한것으로 간주됩니다. 그래서 클럭이 HIGH일경우 데이터는 반드시 안정적인 상태가 되어야 합니다. 데이터라인의 HIGH나 LOW상태는 SCL이 LOW경우에만 변경이 될 수 있고, 이것이 각각의 비트의 타이밍입니다.



SCL이 HIGH일경우만 데이터가 유효함

만약 슬레이브가 인터럽트를 수행하거나 등의 다른 기능을 수행하느라 데이터를 주소받을 수 없는 상태에 있을 경우 SCL 라인을 LOW로 만들어 마스터를 대기상태로 만들수 있습니다. 슬레이브가 데이타를 다시 받을 준비가 되어 클럭 라인을 릴리즈하여 줄 경우 다시 데이터 전송이 시작됩니다. 데이터 전송은 항상 마스터에 의해 생성된 STOP조건에 의해서 끝이 납니다. 하지만 마스터가 다른 슬레이브와 통신을 계속하기를 원한다면 Repeated-Start조건을 생성하고 다른 슬레이브를 addressing합니다.

Acknowledgment

데이터를 주고 받는 중인 리시버는 각각의 바이트가 수신된 후 acknowledge(ACK)를 생성하여 주어야 합니다. ACK는 트랜잭션중 8번째 비트가 전송이 되면 발생하고 이 상태에서 트랜스미터는 SDA버스를 릴리즈하여 리시버가 SDA버스를 동작 할 수 있게 하여주어야 합니다. 리시버는 바이트수신이 끝나면 SDA를 LOW시킴으로 ACK를 보냅니다. 만약 리시버가 SDA를 LOW로 동작시키지 않는다면, 이상태는 no-acknowlege(NACK)가 되며 동작이 중지됩니다. 만약 보내진 바이트가 컨트롤바이트(슬레이브 주소+R/W비트)라면, 오직 주소가 일치하는 슬레이브만이 ACK로 응답할수 있습니다.

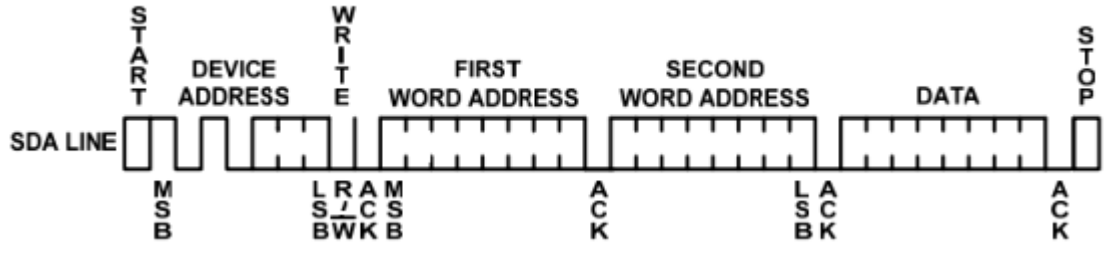
그럼 PIC18F2550마이컴과 24LCS512 EEPROM, DS1631온도 센서가 I2C버스상에서 통신할때 이런 다양한 신호가 어떻게 발생하는지 살펴보도록 하겠습니다. PIC18F2550마이컴의 MSSP(Master Synchronous Serial Port)모듈은 두개의 I/O핀 RB0/SDA(21), RB1/SCL(22)을 통해 I2C 통신을 할 수 있게 합니다. 이 모듈의 자세한 사항은 mikroC 컴파일러가 I2C통신 라이브러리 루틴을 제공하고 있기때문에 여기서는 언급하지 않습니다.

Serial EEPROM (24LC512)

24LC512는 I2C를 지원하는 마이크로칩에서 나온 시리얼 EEPROM으로 64K x 8 (512Kbits)의 용량을 가지고 있습니다. 이 IC칩의 핀다이어 그램은 아래에 나와 있습니다. 이 장치의 7비트 주소의 앞의 4개 비트는 '1010'으로 고정되었습니다만 그 다음의 세개 비트는 A0, A1, A2핀으로 설정이 가능합니다. 예를들어, A0는 HIGH, A1, A2는 그라운드된 24LC512은 '1010001'의 7비트 주소를 가지게 됩니다. 이러한 주소 방식은 총 8개의 I2C 장치를 사용할수 있게 합니다.

바이트 쓰기 연산

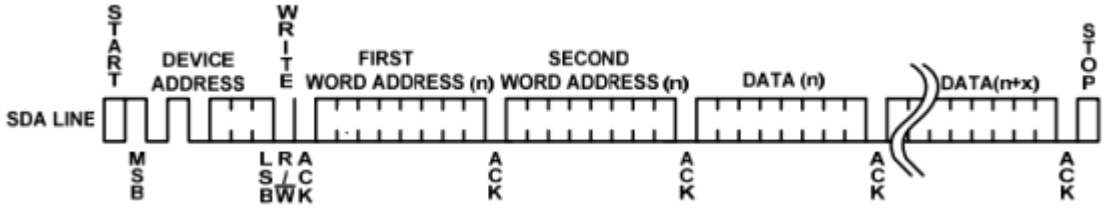
바이트 쓰기 연산을 위해서는, EEPROM의 65536개의 바이트위치중에 하나를 선택하기 위해 두개 바이트의 주소가 필요합니다. 마스터는 컨트롤바이트를 전송한 후에 두개의 주소바이트를 전송합니다. 마스터는 그후에 메모리에 쓸 데이터 바이트를 전송합니다. 이 데이터를 받으면 24LC512는 ACK펄스를 보내고 마스터는 스탑 신호를 생성하여 데이터전송을 끝냅니다.



Byte write operation

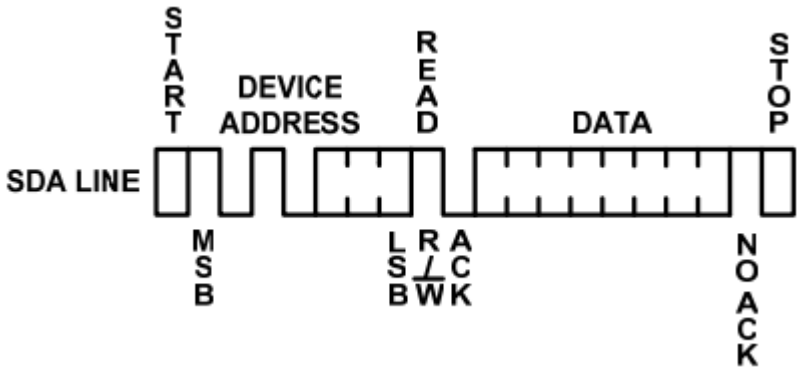
Page Write operation

쓰기에 걸리는 시간을 최소화하기 위하여 24LC512는 Page Write 기능을 제공하는데, 128개의 연속된 바이트를 동시에 쓸수있게 하여줍니다. page write는 바이트를 쓰는 것과 똑같은 방식으로 시작이 되지만 스탑신호를 생성하는 대신 마스터는 127개의 추가적인 바이트를 전송합니다. 이 전송된 되이터는 칩내의 페이지 버퍼에 임시 저장되고 마스터가 스탑 신호를 보내면 메모리로 쓰여집니다. Page write연산에 대한 자세한 사항은 24LC512의 데이터쉬트를 참고하세요.



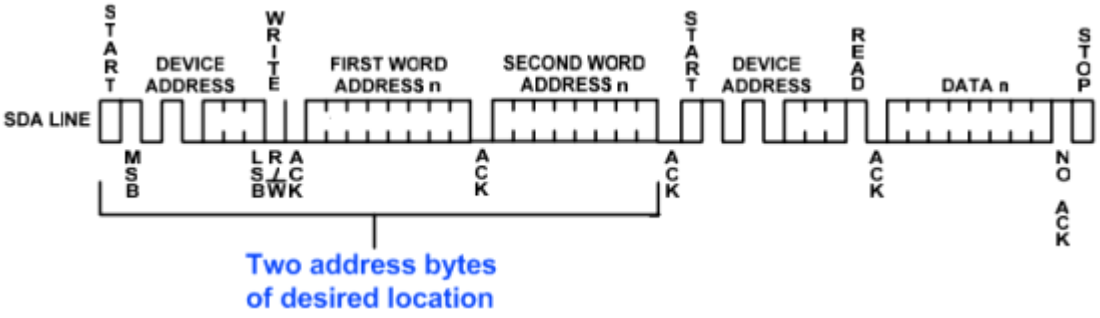
Read operation

읽기 연산도 쓰기연산과 같은 방식으로 시작되지만 컨트롤바이트의 R/W비트가 1로 셋팅이 되는 것이 다릅니다. 24LCS512는 세가지 타입의 읽기 동작을 지원합니다(현재주소 읽기, 무작위 읽기, 순차적 읽기). 내부적으로 EEPROM은 마지막으로 읽힌 주소를 가지고 있는 어드레스 카운터를 가지고 있습니다. 그래서 주소 n을 마지막으로 읽었다면 다음번 현재주소 읽기 연산은 n+1의 데이터를 접근하게 됩니다. R/W비트가 1로 셋팅된 컨트롤 비트를 받게 되면 EEPROM은 ACK를 생성하고 현재 주소의 데이터 바이트를 전송하게 됩니다. 마스터는 ACK를 보내지는 않지만 STOP신호를 보내게 되고 EEPROM은 전송을 멈추게 됩니다.



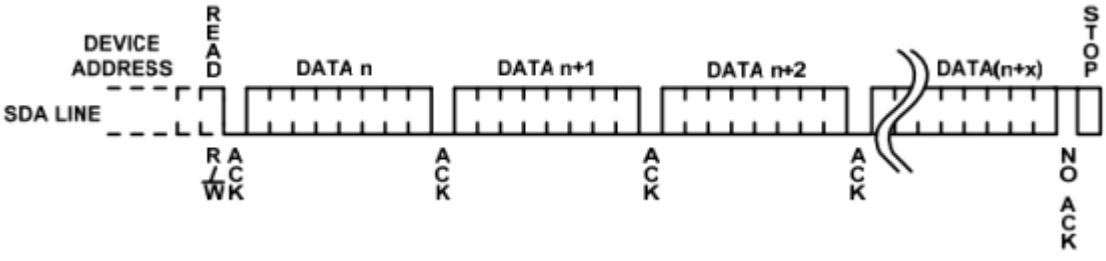
Current address read signal

무작위 읽기 연산은 마스터가 메모리를 무작위로 접근할수 있게 합니다. 이런 연산을 수행하기 위해서는 어드레스 워드가 먼저 셋팅이 되어 있어야 하고 이것을 쓰기연산(R/W비트가 0)사용하여 24LCS512에게 주소데이터를 보냄으로써 이루어집니다. 주소word가 보내지면 마스터는 repeated-Start신호를 ACK뒤에 보내게 됩니다. 이것은 쓰기연산을 종료하게 되고 그러면 마스터는 R/W비트를 1로 셋팅하여 컨트롤바이트를 다시 생성합니다. EEPROM은 다시 ACK를 생성하고 8비트 데이터를 보냅니다. 마스터는 ACK를 보내지 않고 정지신호를 보내 전송을 중지합니다.



Random read operation

순차읽기는 현재주소 읽기나 무작위 읽기 모드에의해 시작될수 있습니다. 24LC512의 첫번째 바이트를 받은 후에 마스터는 현재주소 읽기나 무작위 읽기 모드에서 사용된 정지신호대신에 ACK를 생성합니다. 이 ACK는 24LC512에게 다음의 8비트 워드를 전송하게 명령합니다. 맨 마지막 바이트가 마스터에 전송이 되면 마스터는 ACK를 생성하지 않고 정지신호를 생성합니다.



Sequential read operation

Write Protect (WP)

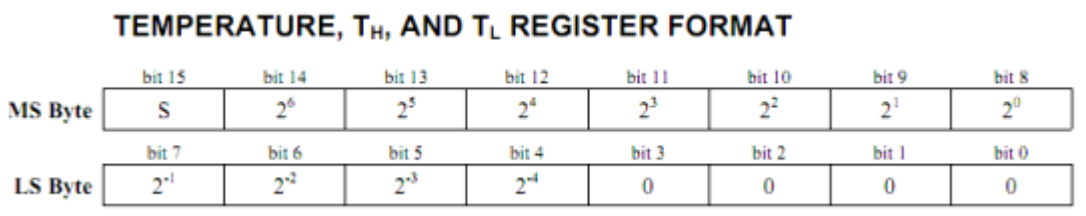
24LC512의 7번핀은 하드웨어 Write Protect입력입니다. 만약 이 핀이 Vcc에 묶여 있다면 쓰기연산은 금지됩니다. 하지만 읽기 연산은 아무상관 없습니다. 본 게시물에서는 이 핀을 그라운드 시키겠습니다.

I2C 온도센서(DS1631)

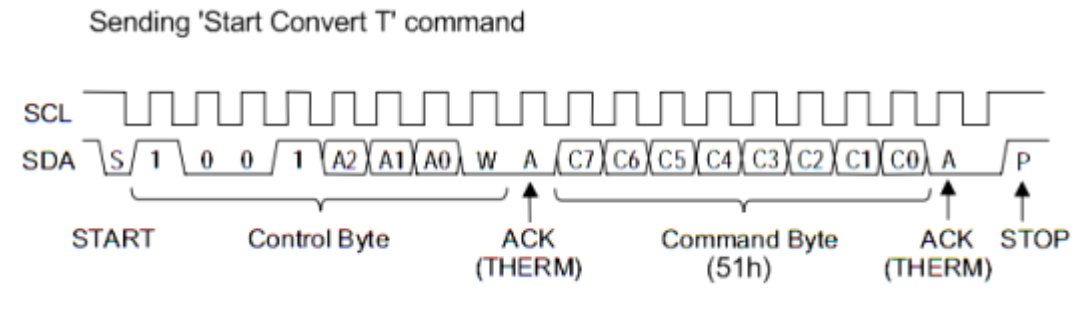
DS1631은 맥심사에서 제조한 디지털 온도측정계로 -55 °C ~125 °C 구간에서 9, 10, 11, 12비트 온도 분해능을 보유하고 있습니다. 기본 분해능은 12비트로 0.0625도씩 온도가 증가합니다. DS1631과의 통신은 I2C인터페이스를 통해서 이루어지며 세개의 어드레스 핀(A0, A1, A2)는 8개의 장치와 통신할 수 있게 하여 줍니다. (핀다이어그램 참조) 각각의 슬레이브의 7비트 주소는 1 0 0 1 A2 A1 A0의 형태이며 A0,A1,A2는 해당 입력 핀을 통해 선택이 가능합니다.

온도 측정

DS1631의 구조와 온도변환 프로세스를 위해서 DS1631의 데이터쉬트를 읽어보기를 권하여 드립니다. 여기서는 온도변환의 one-shot mode에 대해서만 설명하도록 하겠습니다. 장치가 방금 켜졌고 분해능이 12비트이라고 가정합시다. one-shot 모드에서 DS1631 센서는 마스터로부터 명령 바이트, 51h,를 받은 후부터 온도를 12비트 디지털 워드로 변환하기 시작합니다. 이것은 Start Convert T명령으로 알렸 있습니다. 이 변환이 끝나면, 디지털로 변환된 온도는 16비트 2의 보수 형태로 2바이트의 온도레지스터 TH, TL에 저장이 됩니다(아래 참조). Sign비트(S)는 온도가 양(S=0)인지 음(S=1)인지 알려줍니다.



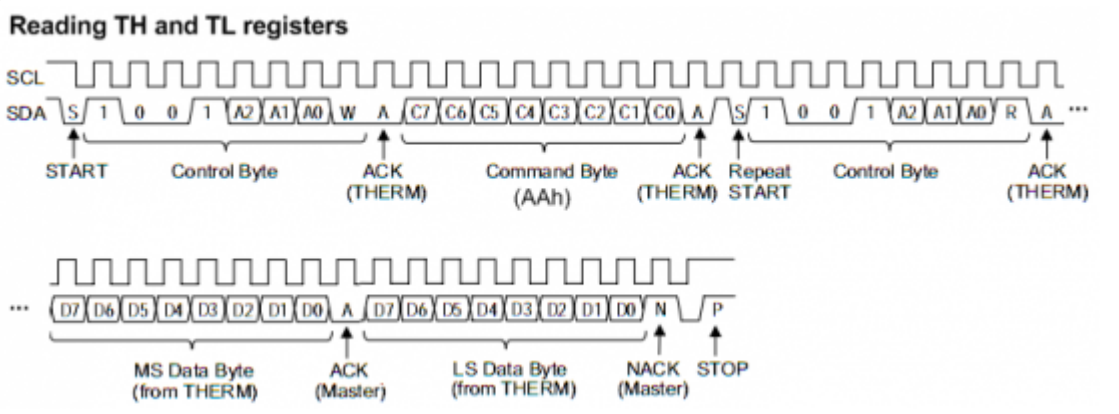
DS1631내부의 온도 레지스터



Start Convert T 명령 보내기

마스터는 Read Temperature(AAh)명령을 보냄으로서 DS1631에서 온도 데이터를 읽을 수 있습니다. 명령에 대한 응답으로 ACK를 받은 후에, 마스터는 같은 슬레이브 주소를 가지고 있는 컨트롤 바이트와 repeated Start신호를 반드시 보내야 합니다. 하지만 이번에는 R/W비트를 1로 셋팅하여 DS1631에 읽기 연산이 실행될 것을 알려줍니다. DS1631은 이 컨트롤바이트에 대한 응답으로 ACK를 보내고 다음 클럭 사이클에 요청받은 데이터를 전송하기 시작합니다. TH, TL레지스터 두개 바이트를 읽을때, 마스터는 첫번째 데이터 바이트에 ACK로 반드시 응답하고, 두번째 바이트에 NACK으로 대응

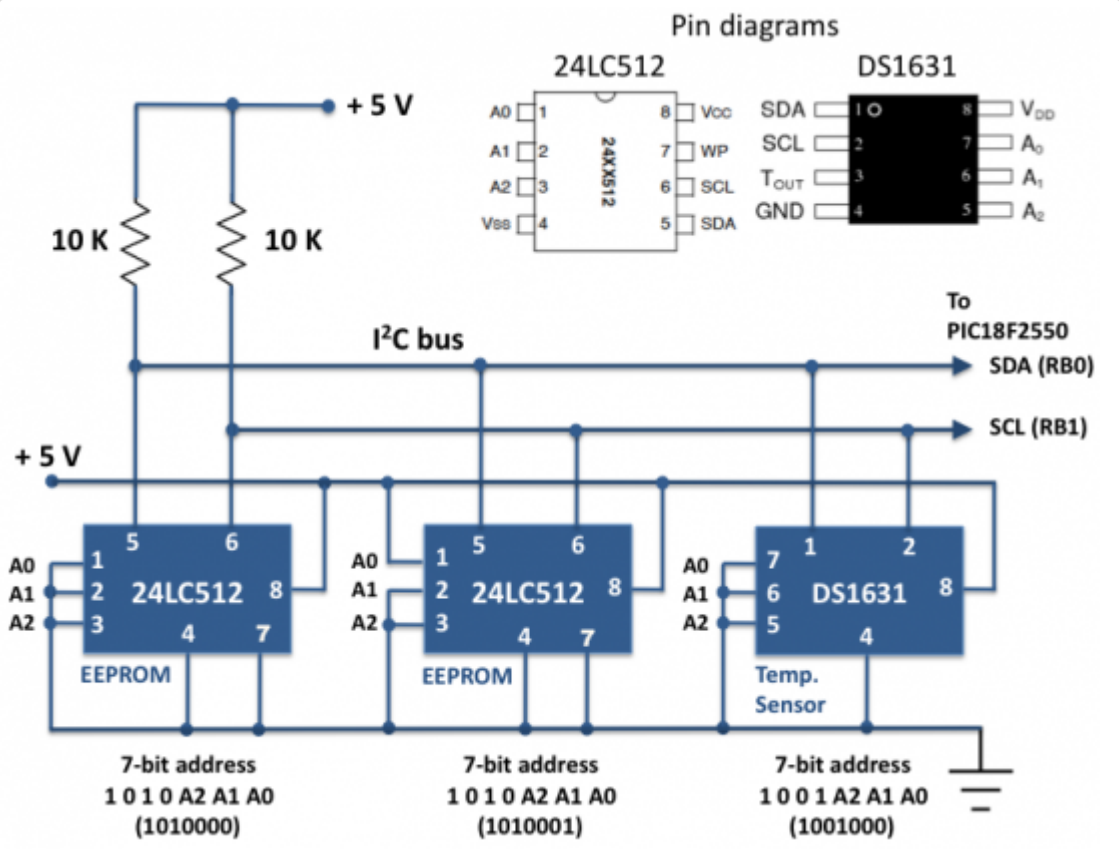
하고 STOP신호를 보냅니다. 만약 데이터의 MSB(Most Significant Byte)만 필요하다면 마스터는 첫번째 데이터 바이트를 받은 뒤 NACK으로 응하고 STOP신호를 보낼수 있습니다.



DS1631로부터 16비트 온도 데이터를 읽기

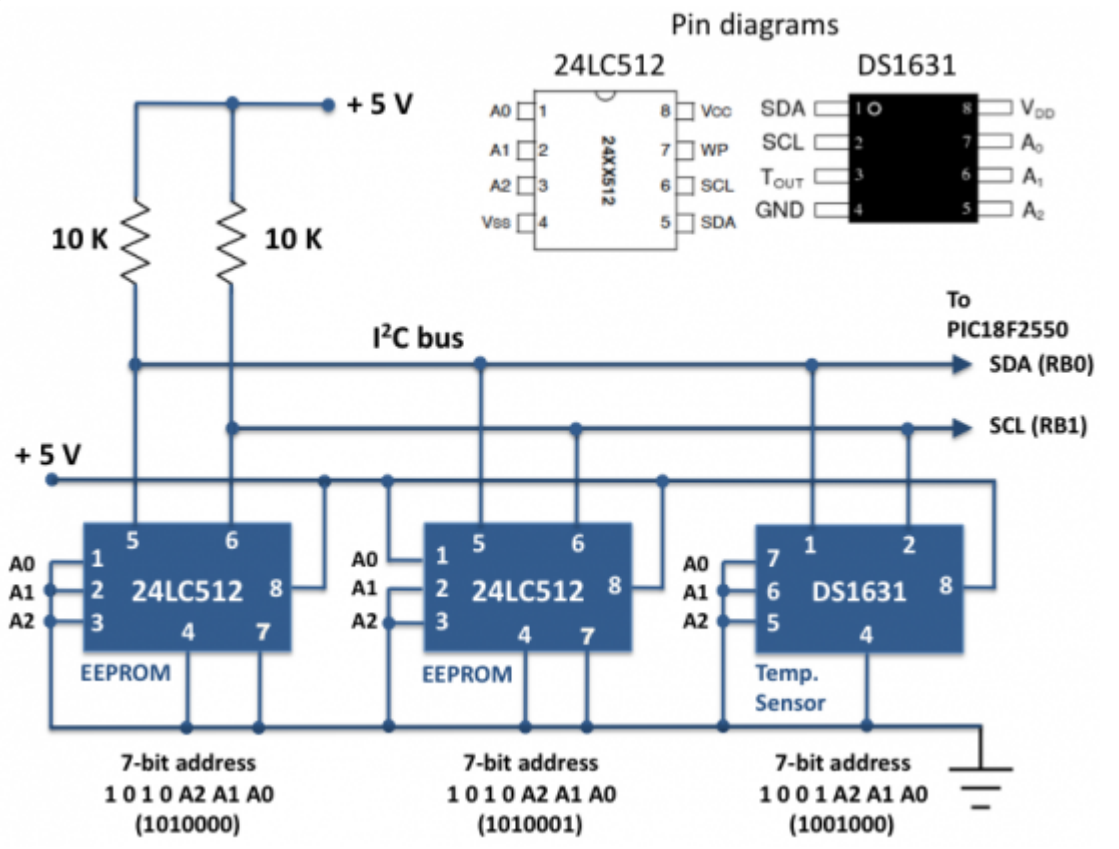
회로 셋업

I2C 실험을 위한 회로도는 아래와 같습니다. 두개의 시리얼 EEPROM(24LC512)와 DS1631 온도 센서는 공용 I2C버스에 연결되어 있습니다. 두개의 EEPROM간의 주소 충돌을 피하기 위하여 하나의 A0핀을 그라운드에 다른 하나를 Vcc에 연결하였습니다. 이 세개의 I2C 장치의 7비트 주소는 아래 다이어 그램에 표시되었습니다. 두개의 10K저항이 I2C버스를 위한 풀업 저항입니다. SDA와 SCL라인은 PIC18F2550의 RB0와 RB1핀에 연결되었습니다. PIC18F2550을 위해서는 StartUSB for PIC 보드를 사용하였습니다.

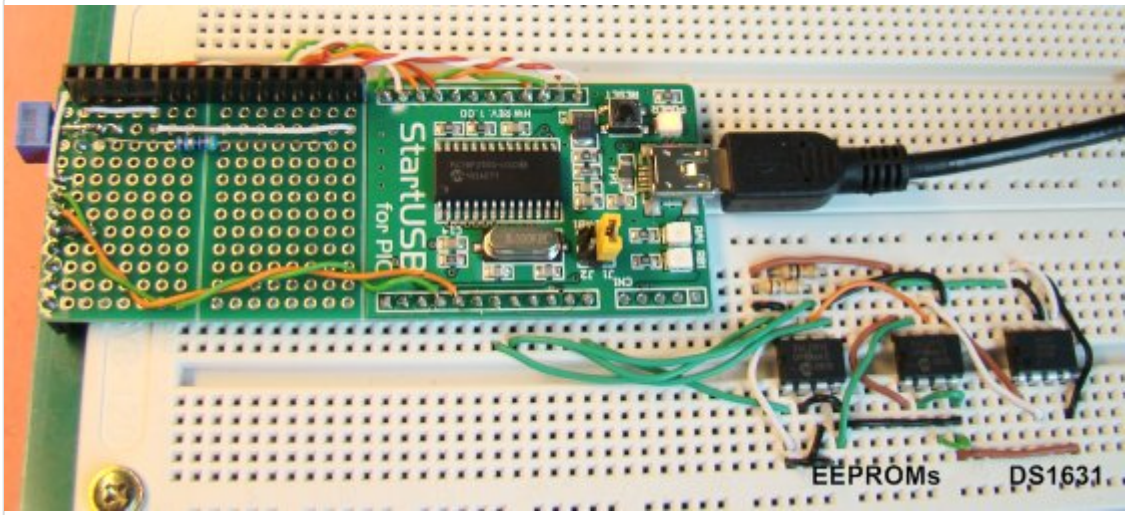


두 개의 EEPROM과 온도센서를 연결한 회로도

표준 16x2 LCD는 I2C 읽기/쓰기 연산을 디스플레이 하기 위해 사용되었습니다. LCD를 동작시키기 위한 MCU핀은 아래와 같습니다.



I2C 버스와 LCD를 PIC18F2550와 연결



브레드보드상에 회로 셋업

소프트웨어

PIC18F2550 마이컴과 공용 I2C버스상에 멀티드롭된 세개의 장치간의 I2C통신을 테스트 하기 위한 소프트웨어 어플리케이션을 작성하여 보도록하겠습니다. 이 프로그램은 먼저 세개의 장치가 버스상에 존재하는지 체크한 후에 온도변환 명령을 DS1631에 보냅니다. 두바이트의 온도 데이터가 읽혀지고 두개의 EEPORM에 저장이 됩니다(하나 EEPROM의 상위바이트, 다른 하나의 EEPROM에는 하위바이트, 0부터 시작). 다섯개의 온도 샘플이 기록되면 EEPROM에서 순차적으로

읽어 온도단위로 변환한뒤 LCD에 디스플레이 합니다. mikroC 컴파일러는 I2C통신을 다루는 라이브러리루틴을 가지고 있습니다. 간단히 설명하면 아래와 같습니다.

void I2C1_Init(const unsigned long clock): I2C통신을 위해서는 제일처음 MSSP모듈을 초기화 하여야 합니다. 예를 들어 *I2C_Init(100000)* 명령은 100KHz로 MSSP모듈을 설정합니다.
unsigned short I2C1_Start(void) : I2C버스가 Free한지 확인하고 시작신호를 생성할지 결정합니다. 에러가 없으면 제로.
id I2C1_Repeated_Start(void) : repeated-start 신호 생성
unsigned short I2C1_Is_Idle(void) : I2C버스가 Free한지 체크하고 free하면 1을 리턴, 아니면 0을 리턴
unsigned short I2C1_Rd(unsigned short ack) : 슬레이브에서 한바이트를 읽고, 파라메터 ACK가 0이면 NACK신호를 보냄. 그렇지 않으면 ACK를 보냄. 예를 들어 마이컴이 DS1631에서 두바이트의 온도를 읽을려면, 첫번째 바이트를 읽기 위해 I2C1_Rd(1)을 두번째 바이트를 읽기 위해 I2C1_Rd(0)을 사용하여야 합니다.
unsigned short I2C1_Wr(unsigned short data) : I2C버트를 통해 데이터를 보냄. 에러가 없으면 0을 리턴
void I2C1_Stop(void) : 정지 명령 생성

프로그램에서는 PIC18F2550이 먼저 각각의 주소를 세개의 장치에 보내 ACK를 확인함으로써 세개의 장치가 I2C버스상에서 사용가능한지를 체크합니다. mikroC 컴파일러 는 아래와 같이 장치를 체크합니다.

```
void check_device(unsigned short dev_address){
I2C1_Start();
if (I2C1_Wr(dev_address)){
  Lcd_Out(2,1,"Device not found");
}
else Lcd_Out(2,1,"Device is OK");
I2C1_Stop();
}
```

아래는 PIC18F2550이 Start Convert T명령을 DS1631에 보내고, 그후에 12비트 온도변환이 끝날때까지 750ms를 기다렸다가 2바이트의 온도값을 받는 프로그램입니다.

```
// Read temperature
 I2C1_Start();
 I2C1_Wr(DS1631);          // Send device address
 I2C1_Wr(0x51);           // Start Convert Temp command
 I2C1_Stop();
```

```
 Delay_ms(750);
 I2C1_Start();
 I2C1_Wr(DS1631);              // DS1631 Address again
 I2C1_Wr(0xAA);               // Read Temperature command
 I2C1_Repeated_Start();
 I2C1_Wr(DS1631+1);            // Address with Read
 MS_Byte = I2C1_Rd(1);         // Read MSB and send ACK
 LS_Byte = I2C1_Rd(0);         // Read LSB and send NAK
 I2C1_Stop();
```

아래의 두개루틴은 두개의 EEPROM에 대해 읽기/쓰기 연산을 수행합니다.

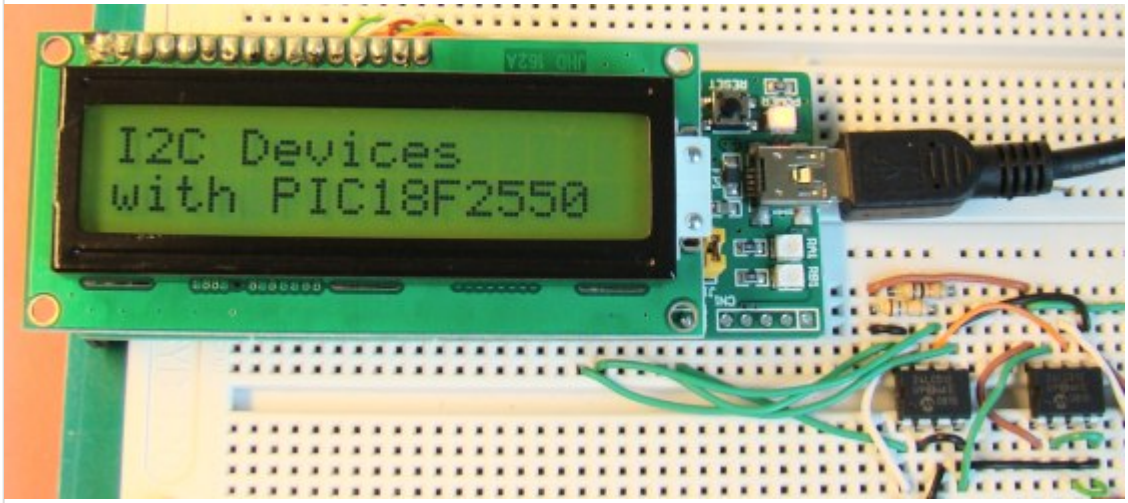
```
//--------------- Reads data from 24LC512 EEPROM - single location
unsigned short EEPROM_ReadByte(unsigned short EEPROM_Select, unsigned short rAddress) {
 unsigned short rAddrH, rAddrL, result;
 rAddrH = 0;
 rAddrL = rAddress;
 I2C1_Start();            // issue I2C1 start signal
 I2C1_Wr(EEPROM_Select);    // send byte via I2C1  (device address + W)
 I2C1_Wr(rAddrH);           // send Higher address byte
 I2C1_Wr(rAddrL);           // send Lower address byte
 I2C1_Repeated_Start();    // issue I2C1 signal repeated start
 I2C1_Wr(EEPROM_Select+1);   // send byte (device address + R)
 result = I2C1_Rd(0u);     // Read the data (NO acknowledge)
 while (!I2C1_Is_Idle())
 asm nop;               // Wait for the read cycle to finish
 I2C1_Stop();
 return result;
}

//--------------- Writes data to 24LC512 EEPROM - single location
void EEPROM_WriteByte(unsigned DeviceSelect, unsigned short wAddress, unsigned short wData) {
 unsigned short wAddrH, wAddrL;
 wAddrH = 0;
 wAddrL = wAddress;
```

```
 I2C1_Start();        // issue I2C1 start signal
 I2C1_Wr(DeviceSelect);   // send control byte
 I2C1_Wr(wAddrH);     // send higher address byte
 I2C1_Wr(wAddrL);     // send lower address byte
 I2C1_Wr(wData);      // send data to be written
 I2C1_Stop();
}
```

[프로젝트파일 다운로드]

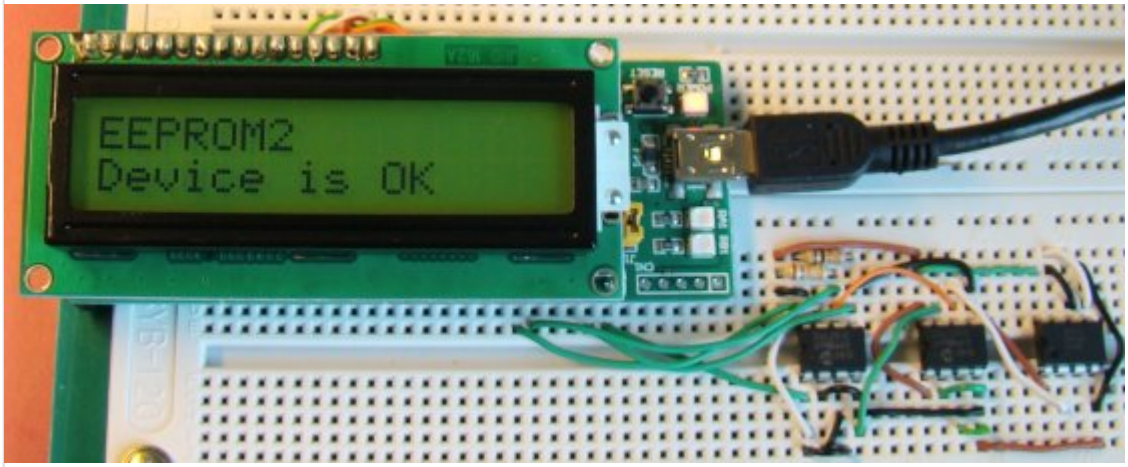
결과

아래의 그림은 I2C테스트 프로그램을 실행하면서 연산을 수행한 것을 보여줍니다.



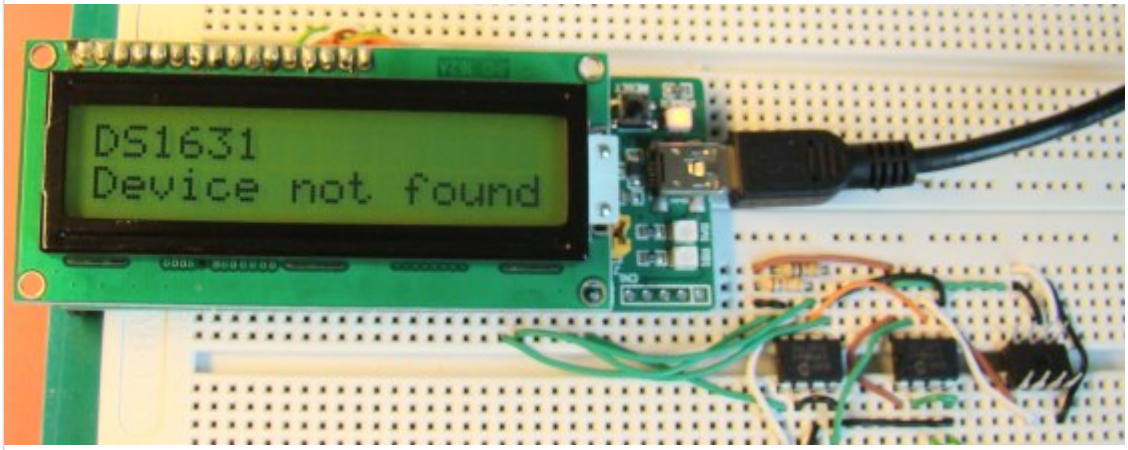
시작 메세지

EEPROM1 테스팅

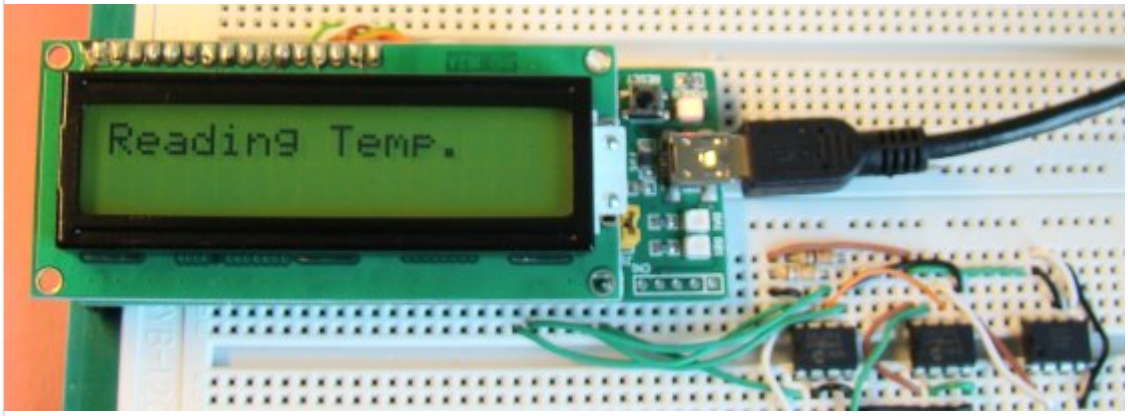


EEPROM2 테스팅

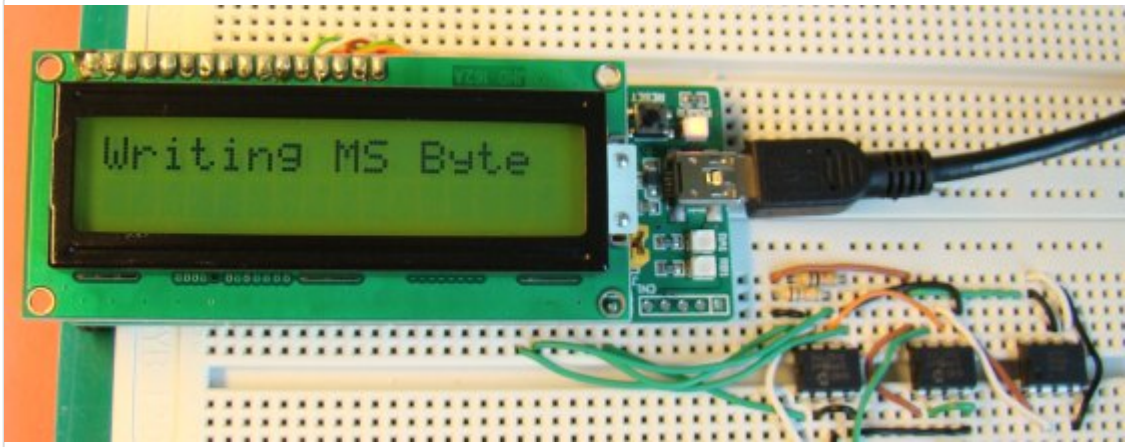
DS1631 온도센서 테스트



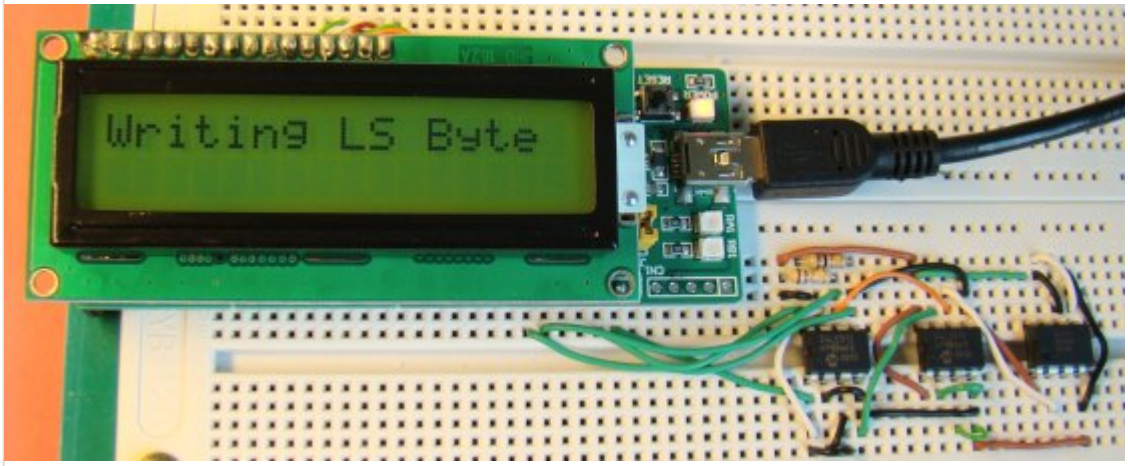
DS1631를 떼어 냈을때



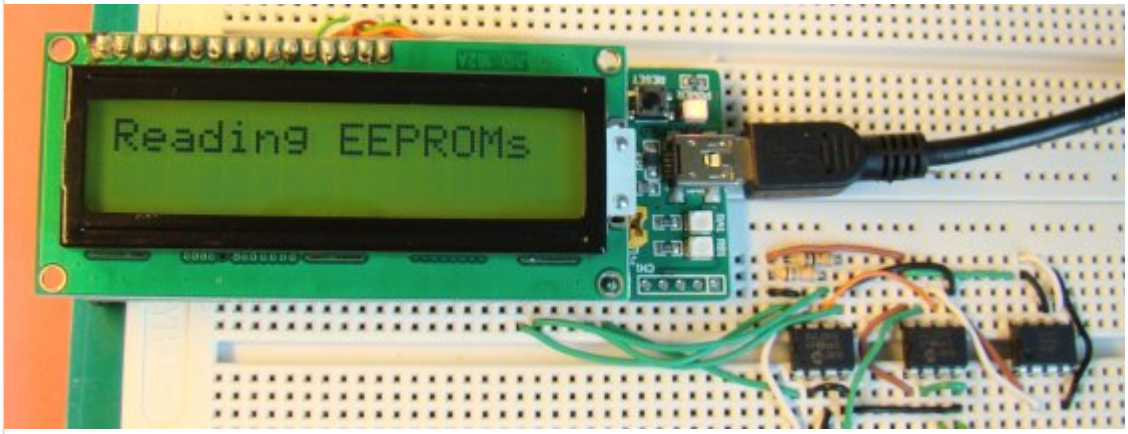
DS1631에서 온도 읽기



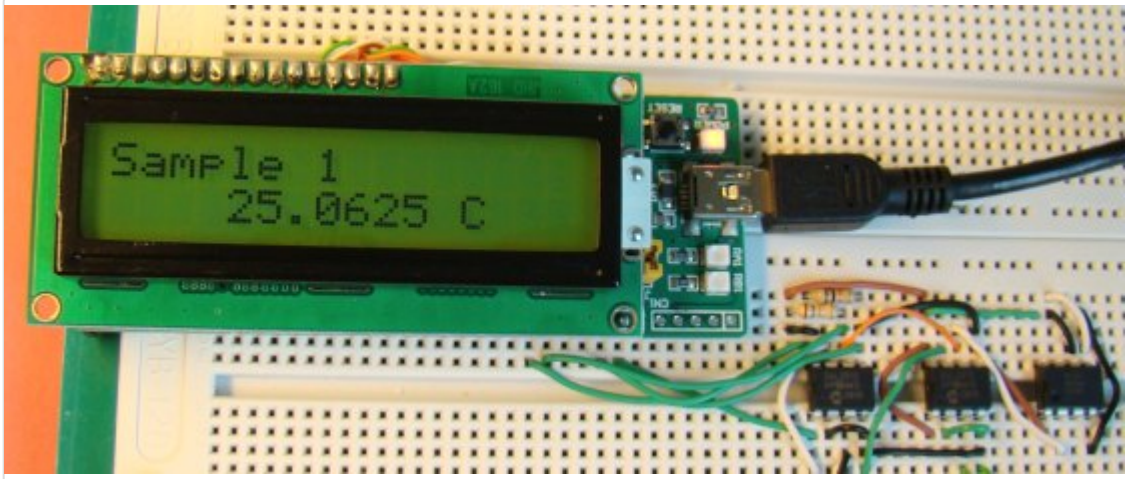
EEPROM1에 온도 MS바이트 쓰기



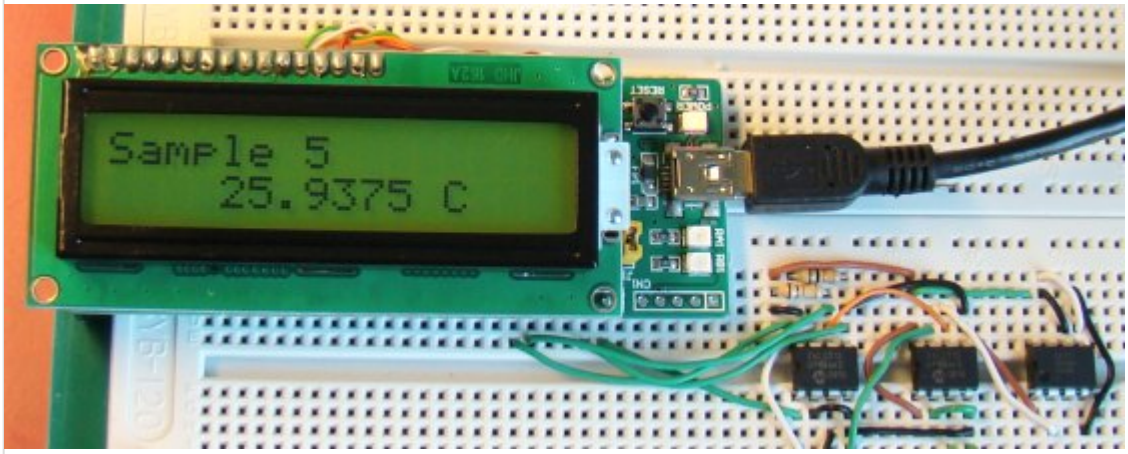
EEPROM2에 온도 LS바이트 쓰기



기록된 온도 샘플 읽기



첫번째 온도 샘플



5번째 샘플

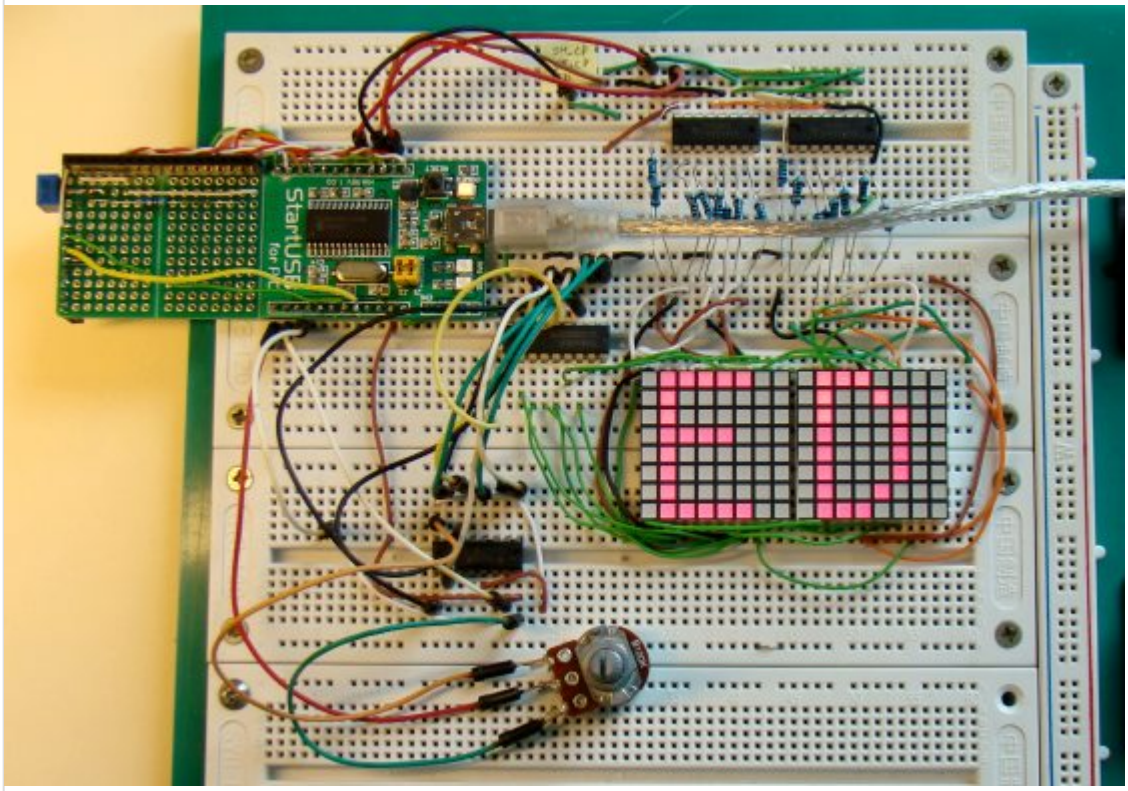
가치창조기술 | www.vctec.co.kr

[PIC 마이컴 실험하기] 15. 도트 매트릭스 LED에서 텍스트를 움직이며 디스플레이하기

마이컴 실험실

2011/10/20 17:25 http://blog.naver.com/ubicomputing/150121889366

지난 12번째 실험에서 LED도트 매트릭스의 기본적인 구조와 마이컴과의 연결에 대해서 살펴보았는데요. 오늘은 16X8 LED 도트 매트릭스에서 텍스트를 옆으로 스크롤링하는 방법에 대해 실험해 보도록 하겠습니다. 사용된 MCU는 PIC18F2550(StartUSB for PIC보드에 탑재)입니다. LED의 16개 열은 두개의 쉬프트 레지스터(74HC595)에 의해 개별적으로 동작되는 반면 8개의 행은 decade 카운터(CD4017)에서 디코딩된 출력에 의해 동작이 됩니다. 지난 12번째 실험에서는 열이 스캔이 되었었는데 여기에서는 행을 스캔하고 열에 적절한 로직레벨을 공급하도록 하겠습니다. 포텐티오미터로부터의 아날로그 입력은 마이컴에 의해 읽혀 텍스트를 옆으로 스크롤하는 속도를 결정하게 합니다. 스크롤은 오른쪽에서 왼쪽으로 움직이도록 하겠습니다. PIC18F2550용 프로그램은 mikroC 컴파일러로 개발되었습니다.

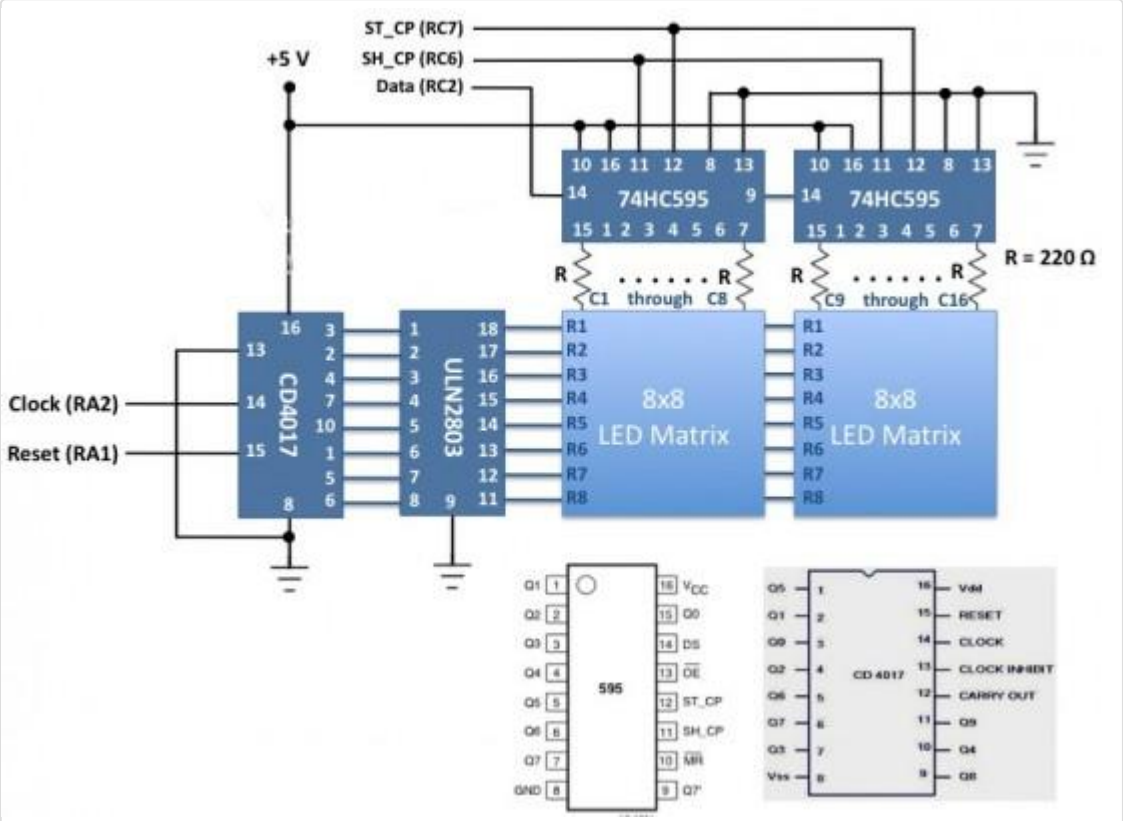


16x8 LED 도트 매트릭스에서 텍스트 스크롤하기

회로도

LED도트 매트릭스 디스플레이의 내부 구조는 지난 12번째실험에서 다루었기 때문에 여기서는 다시 다루지 않겠습니다. 두개의 8x8 LED 매트릭스가 이번 실험에 사용되었고, 두개 LED의 각각의 행을 연결하여 총 8개의 행으로 결합하였고 열은 개별적으로 동작되도록하여 결과적으로 16개의 열이 만들어 졌습니다. 각각의 행에 LED의 합쳐진 전류는 ULN2803

IC내부의 darlington pair 트랜지스터 어레이를 통하여 싱크됩니다. 16개의 열(양극)은 두개의 쉬프트레지스터(74HC595)에 의해 동작하며 전류제한 저항(220 Ω)과 직렬로 아래와 같이 연결됩니다.



회로도

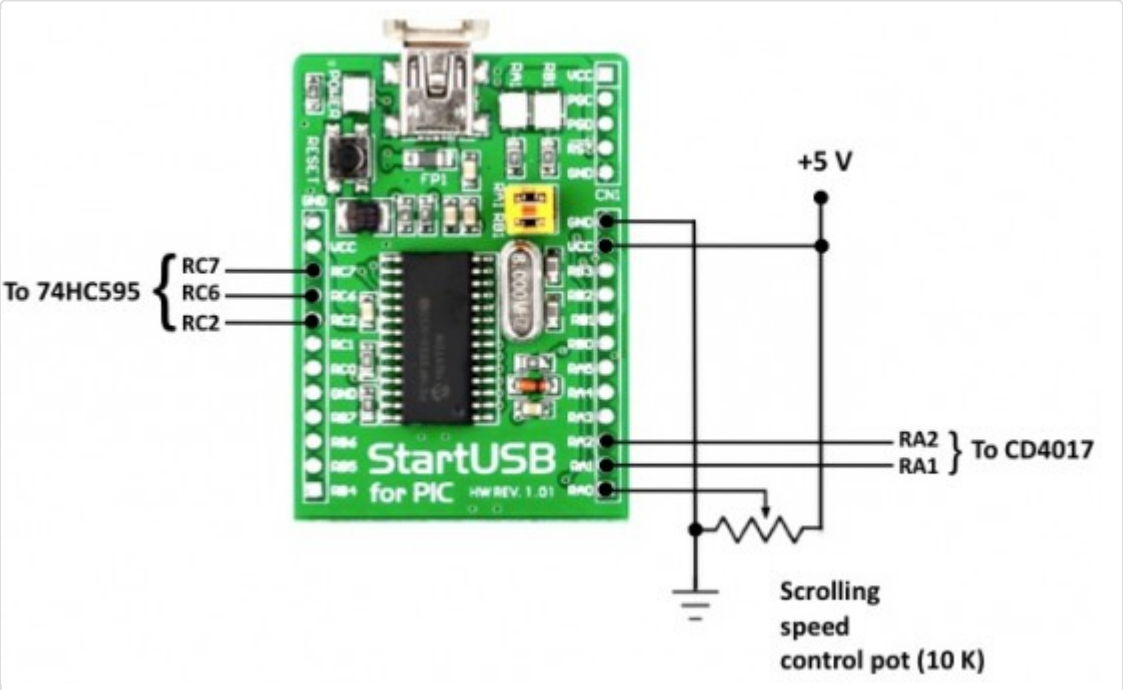
쉬프트레지스터(74HC595)의 역활

쉬프트레지스터를 사용하면 LED 매트릭스를 동작시키는데 필요한 I/O핀의 숫자를 최소화 할수 있습니다. 16개의 열을 각각 동작시키려면 16개의 MCU I/O핀이 필요하지만 두개의 74HC595 IC를 사용함으로써 필요한 I/O핀을 3개로 줄일 수 있습니다. 74HC595는 스토레지 레지스터를 가진 8단계 시리얼-IN, 시리얼-OUT 혹은 패러럴-OUT 쉬프트 레지스터입니다. 쉬프트 레지스터와 스토레지 레지스터는 별도의 클럭 SH_CP(핀11)과 ST_CP(핀12)을 가지고 있습니다. 데이터는 DS핀(14)을 통하여 레지스터에 시리얼하게 공급이 되며 SH_CP입력의 양 전이 시에 쉬프트됩니다. 그러나 각각의 레지스터의 데이터는 스토레지 레지스터로 전송되기 이전에는 74HC595의 출력 핀에 나오지 않습니다. 74HC595 표준 시리얼 출력 Q7(9번핀)을 cascading을 위해 제공하는데, 이것이 이번 실험에 필요한 부분입니다. 첫번째 쉬프트 레지스터의 시리얼 출력은 두번째 쉬프트레지스터의 시리얼 입력(DS핀)에 연결됩니다. 그래서 16비트 column 데이터는 첫번째 쉬프트 레지스터의 DS핀을 통하여 시리얼하게 전송이 가능합니다. 이것은 ST_CP상의 클럭펄스 뒤에 따라오는 SH_CP상의 16개의 클럭 펄스를 필요로 합니다. 비동기 리셋 핀(MR)은 항상 HIGH(비활성화)된 반면 OE(Output Enable)핀은 그라운드(활성화)됩니다.

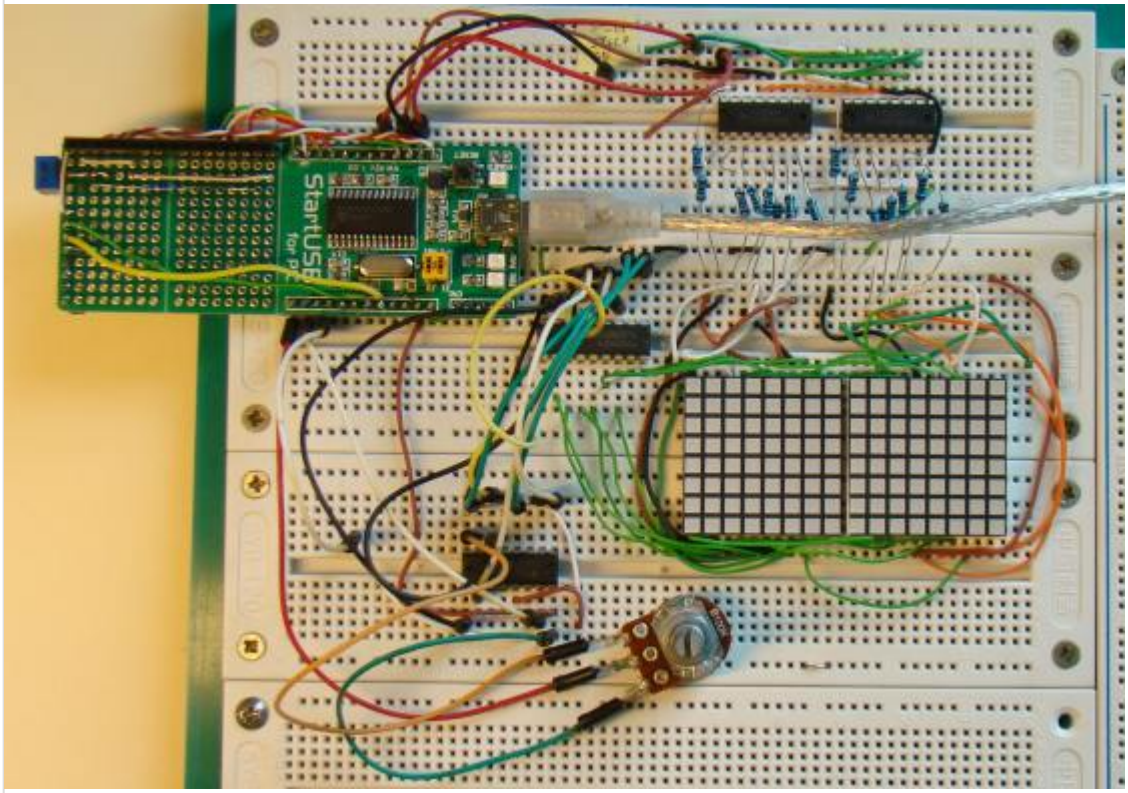
카운터의 역활(CD4017)

처음 잠시 이야기를 하였지만, 이번 실험에서는 행이 위에서부터 아래로 빠르게 스캔이 될 것입니다. 8개의 행을 순차적으로 스캔하기 위해 8개의 I/O핀이 필요합니다. PIC18F2550의 PORTB핀을 이런 용도로 사용할수 있습니다. 하지만

PORTB 핀들을 interrupt-on-change기능과 같은 다른 용도로 사용하고자 한다면 CD4017과 같은 포트 확장기를 사용할수 있으며 MCU의 두개의 I/O핀만을 필요로 합니다. CD4017은 5-stage divided-by-10 Jonhnson counter로 10개의 디코딩된 출력과 캐리아웃비트를 가지고 있습니다. 카운터는 카운터 리셋라인에 로직1을 줌으로써 제로카운트로 클리어 할수 있습니다. clock inhibit pin(13번핀)이 그라운드되어 있을때, 카운터는 클럭시그널(14번핀)의 positive edge시에 증가합니다. 10개의 디코딩된 출력은 보통 로직0 상태에 있고 각각의 타임 슬록시에만 로직 1상태로 갑니다. 각각의 디코딩된 출력은 1개의 완전한 클럭사이클 동안 high로 남아 있게 됩니다. 캐리아웃 신호는 매 10클럭 입력 사이클마다 한개의 완전한 사이클을 완성하고 다음단계의 ripple carry signal로 사용이 됩니다. LED매트릭스의 8개열은 순차적으로 ULN2803 IC를 통해 CD4017의 디코딩된 출력 Q0-Q7에 연결이 됩니다. ULN2803 I는 8개의 Darlington pair로 각각은 LED의 결합된 전류를 싱크하기 위한 그라운드 패스를 제공합니다. 매 8번째 클럭사이클에서는 마이크로컨트롤러는 리셋핀(15번핀)에 로직 1을 주어 카운터를 리셋하게 됩니다. 74HC595와 CD4017을 동작시키기 위해 사용하는 마이컴의 핀은 아래와 같습니다. 10K 포텐티오미터가 PIC18F2550의 RA0핀에 연결되어 텍스트 스크롤의 속도를 제어하게 됩니다.



LED도트 매트릭스를 작동시키기 위한 MCU I/O핀



StartUSB for PIC보드를 이용하여 브레드보드에 회로 셋업

소프트웨어

LED 도트 매트릭스에 글씨를 디스플레이하는 방법을 잘 모르신다면 지난 12번째 실험을 먼저 참고하시길 권해드립니다. 행을 스캐닝할때는 각각의 행은 매우 짧은 시간(대략 1ms)동안 선택이 되고 열에 해당 로직 레벨을 주게 됩니다. 열들을 1초에 100번이상 빠르게 스캔하면서 LED를 ON시키면 잔상 효과로 인해 글자가 정지되어 있는 것처럼 보입니다. 아래의 그림은 8X8도트 매트릭스에 A 문자를 디스플레이하기 위해 ON되어야 할 LED를 보여줍니다. 모든 프린트 가능한 ASCII 문자는 2차원 constant arrary CharData[][8]에 저장이 되고 이것은 PIC18F2550의 프로그램 메모리에 저장이 됩니다.

CharacterRow[1]	0	0	0	0	1	1	1	0
CharacterRow[2]	0	0	0	1	0	0	0	1
CharacterRow[3]	0	0	0	1	0	0	0	1
CharacterRow[4]	0	0	0	1	0	0	0	1
CharacterRow[5]	0	0	0	1	1	1	1	1
CharacterRow[6]	0	0	0	1	0	0	0	1
CharacterRow[7]	0	0	0	1	0	0	0	1
CharacterRow[8]	0	0	0	1	0	0	0	1

문자 A를 표시하기 위한 값

```
const unsigned short CharData[][8] ={
{0b00000000, 0b00000000, 0b00000000, 0b00000000, 0b00000000, 0b00000000, 0b00000000, 0b00000000}, //Space
{0b00000100, 0b00000100, 0b00000100, 0b00000100, 0b00000100, 0b00000100, 0b00000000, 0b00000100}, // !
{0b00001010, 0b00001010, 0b00001010, 0b00000000, 0b00000000, 0b00000000, 0b00000000, 0b00000000}, // "
{0b00000000, 0b00001010, 0b00011111, 0b00001010, 0b00011111, 0b00001010, 0b00011111, 0b00001010}, // #
{0b00000111, 0b00001100, 0b00010100, 0b00001100, 0b00000110, 0b00000101, 0b00000110, 0b00011100}, // $
{0b00011001, 0b00011010, 0b00000010, 0b00000100, 0b00000100, 0b00001000, 0b00001011, 0b00010011}, // %
{0b00000110, 0b00001010, 0b00010010, 0b00010100, 0b00001001, 0b00010110, 0b00010110, 0b00001001}, // &
{0b00000100, 0b00000100, 0b00000100, 0b00000000, 0b00000000, 0b00000000, 0b00000000, 0b00000000}, // '
{0b00000010, 0b00000100, 0b00001000, 0b00001000, 0b00001000, 0b00001000, 0b00000100, 0b00000010}, // (
{0b00001000, 0b00000100, 0b00000010, 0b00000010, 0b00000010, 0b00000010, 0b00000100, 0b00001000}, // )
{0b00010101, 0b00001110, 0b00011111, 0b00001110, 0b00010101, 0b00000000, 0b00000000, 0b00000000}, // *
.....
{0b00000000, 0b00000000, 0b00000000, 0b00001010, 0b00011110, 0b00010100, 0b00000000, 0b00000000} // ~
};
```

8X8포맷에서 프린트 가능한 ASCII 문자를 위해 열의 값 정의하기

한가지 떠오르는 질문은 "이 배열에서 어떻게 특정문자의 올바른 인덱스 값을 찾아내는가?" 입니다. 해답은 간단합니다. 이 배열은 '스페이스' (10진값 32)부터 시작해서 '~' (10진값 126)까지 ASCII문자를 순차적으로 넣어 만든 배열입니다. 그래서 ASCII값에서 32를 빼면 올바른 해당 행인덱스 값을 얻을수 있습니다. 예를들어 만약 $를 디스플레이하고 싶을때 $의 10진값은 36입니다. 여기서 32를 빼면 4를 얻고 이것이 $의 행 인덱스 값이 됩니다. (위의 그림을 살펴보세요)

그럼 글자를 좌측으로 스크롤링하는 방법에 대해 살펴 보겠습니다. 16X8 LED 매트릭스의 비트정보를 저장하기 위해서 디스플레이버퍼를 정의할 것입니다. integer타입(16bit)을 8개 가지고 있는 크기의 배열인데 이 배열의 내용은 아래 매트릭스에서 보여지는 그림과 같습니다. 아래 그림은 빈화면을 보여주는 버퍼의 비트값들입니다.

Display Buffer with all zeros (displaying nothing)																
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	DisplayBuffer[1]
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	DisplayBuffer[2]
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	DisplayBuffer[3]
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	DisplayBuffer[4]
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	DisplayBuffer[5]
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	DisplayBuffer[6]
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	DisplayBuffer[7]
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	DisplayBuffer[8]

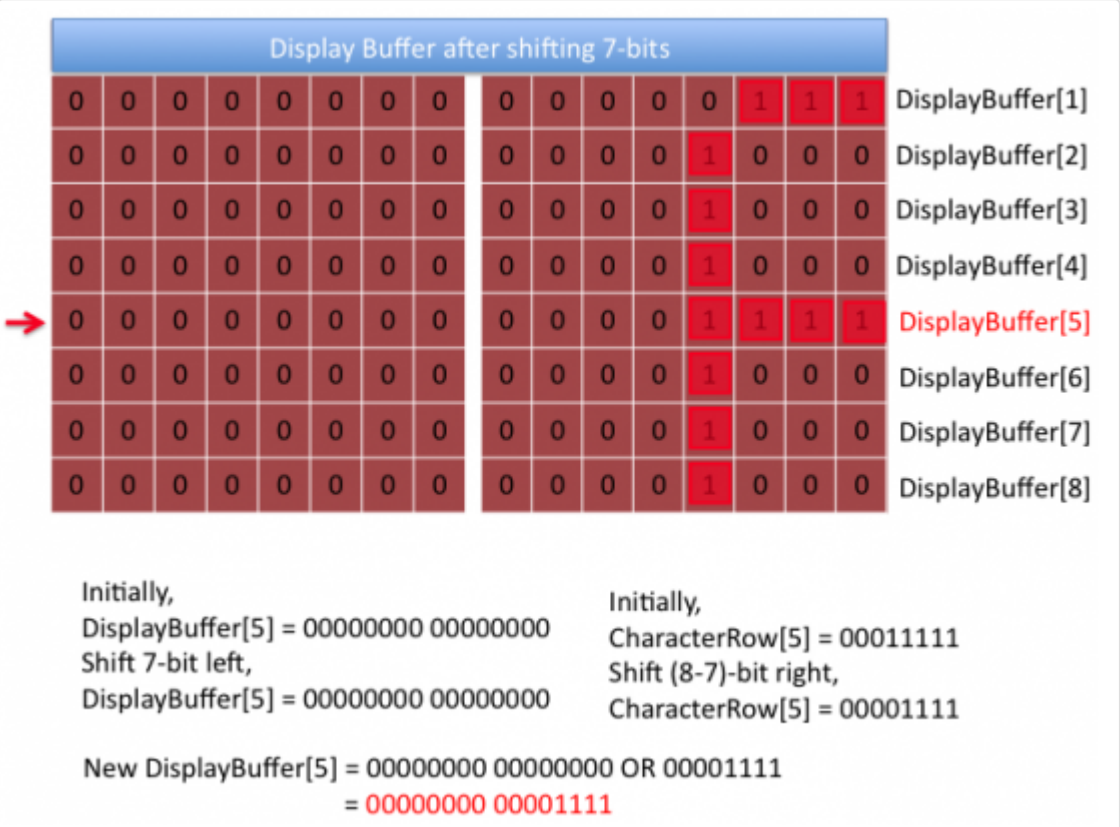
8x16 bit 디스플레이 버퍼

매트릭스상에서 글자를 디스플레이하기 위해서는, 열을 재빨리 바꾸어가며 각각의 행에 적절한 로직레벨을 주어야합니다. 문자를 오른쪽에서 좌측으로 옮기려고할때는 모든 행의 열의 값을 적절한 거리만큼 옆으로 쉬프트 시켜야 합니다. 문자가 충분히 쉬프트되었으면, 메세지의 다음 문자를 디스플레이하기 위한 열의 값을 줍니다. 각각의 쉬프트에선 디스플레이버퍼를 업데이트하여야 합니다. 디스플레이버퍼를 업데이트하기 위한 아래 공식이 오른쪽에서 왼쪽으로의 스크롤 효과를 내게 만들어 줍니다.

DisplayBuffer[i] = (DisplayBuffer[i] << ShiftAmount) BIT OR (CharacterRow[i] >> (8- ShiftAmount)

최초에 디스플레이가 비어 있다고 가정하고 A를 오른쪽에서 왼쪽으로 한번에 한 컬럼씩 스크롤링한다고 가정합시다. 만약 위의 그림중에 하나에 나와 있는 A에 해당하는 매트릭스패턴 을 본다면, 좌측 3개 컬럼은 비어 있기때문에 4번째 컬럼이 디스플레이버퍼에 쉬프트 되기 전에는 아무것도 보지 못할 것입니다. 아래의 그림은 최초의 디스플레이 버퍼에서 7번 좌로 쉬프트된 단계(ShiftAmount = 7)를 보여줍니다. 1번 행은 최초에 '00000000 00000000'이였는데 '00000000 00000111로 A문자의 일부가 표시 되었습니다. 이 값은 다음과 같이 얻어집니다. 최초에 디스플레이버퍼('00000000 00000000')를 좌측으로 7비트 쉬프트합니다. 여전히 디스플레이버퍼는 모두 '0'입니다. A문자를 저장하고 있는 8x8 매트릭스의 첫번째 행의 값은 '00001110'입니다. 이 값을 오른쪽으로 1번(8-7=1) 쉬프트하면 '00000111'의 값을 얻고 이값을 DisplayBuffer[1]과 bit-OR합니다. 그러면 '000000000 00000111'의 값을 얻게 됩니다.

8x8의 문자가 디스플레이 버퍼에 완전히 로딩되고 ShiftAmount 8까지 순차적으로 증가되어야 합니다. 그러면 다시 ShiftAmount는 1부터 시작되고 오르쪽에 다음 문자를 로딩합니다. 이런식으로 모든 문자들이 로딩될때까지 계속 합니다.



7비트를 좌측으로 쉬프트한 후의 디스플레이 버퍼

아래의 루틴은 16비트 열의 값을 시리얼하게 두개의 쉬프트 레지스터에 공급하게 합니다.

```
void send_data(unsigned int temp){
unsigned int Mask = 0x0001, t, Flag;
for (t=0; t<16; t++){
  Flag = temp & Mask;
  if(Flag==0) Serial_Data = 0;
  else Serial_Data = 1;
  SH_Clk = 1;
  SH_Clk = 0;
  Mask = Mask << 1;
}
// Apply clock on ST_Clk
ST_Clk = 1;
```

```
ST_Clk = 0;
}
```

그리고 아래의 코드는 도트 매트릭스 디스플레이에 메세지를 스크롤하게 만들어 줍니다. *shift_step* 의 값이 한번에 얼마만큼 쉬프트 시킬것인지를 결정합니다. 여기서는 1을 셋팅하였습니다.

```
for (k=0; k<StringLength; k++){
  for (scroll=0; scroll<(8/shift_step); scroll++) {
   for (ShiftAmount=0; ShiftAmount<8; ShiftAmount++){
    index = message[k];
    temp = CharData[index-32][ShiftAmount];
    DisplayBuffer[ShiftAmount] = (DisplayBuffer[ShiftAmount] << shift_step)|

   (temp >> ((8-shift_step)-scroll*shift_step));
   }
  speed = 10+ADC_Read(0)/10;
  for(l=0; l<speed;l++){
   for (i=0; i<8; i++) {
    send_data(DisplayBuffer[i]);
    CD4017_Clk = 1;
    CD4017_Clk = 0;
    Delay_ms(1);
   }  // i
  CD4017_Rst = 1;
  CD4017_Rst = 0;
  } // l
  } // scroll
 }
```

mikroC 컴파일러로 작성된 프로젝트 파일 다운로드

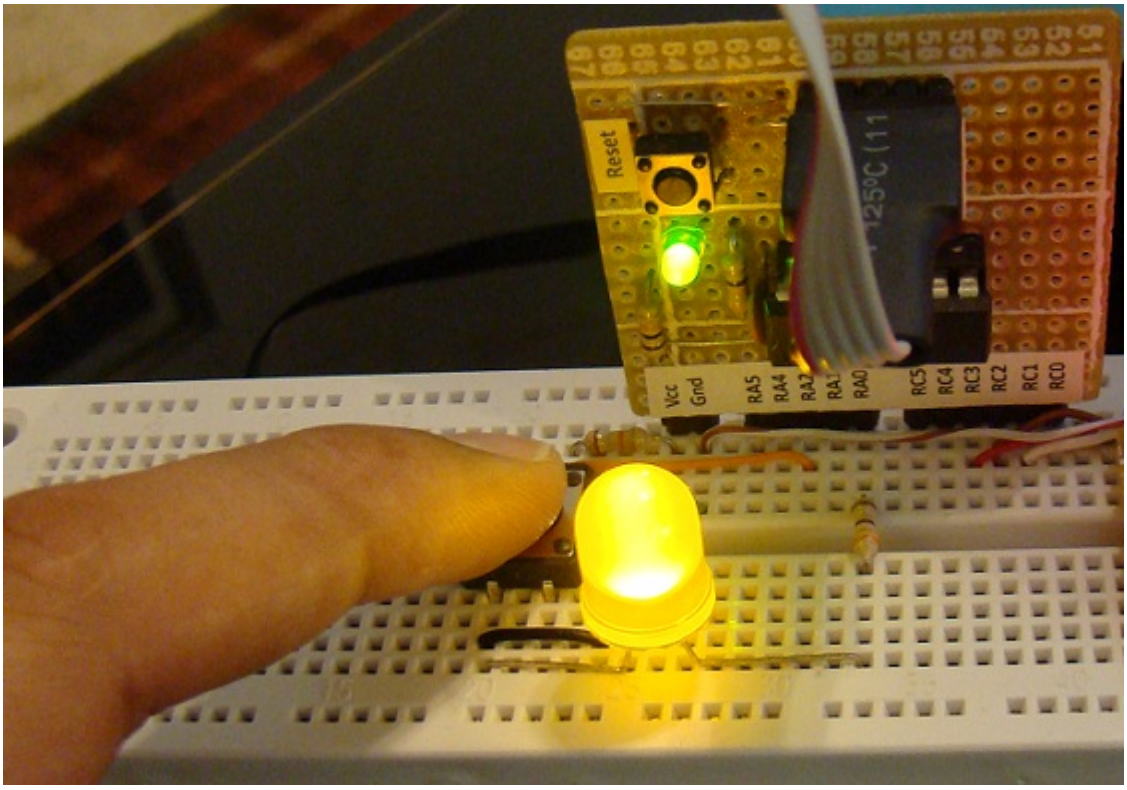
가치창조기술 | www.vctec.co.kr

[PIC 마이컴 실험하기] 16. 인터럽트 이해하기

마이컴 실험실

2013/01/16 12:22 http://blog.naver.com/ubicomputing/150156622605

실시간으로 데이터를 처리해야 하는 임베디드 시스템의 환경에서 인터럽트는 이벤트를 제어하기 위한 강력한 도구 입니다. 전형적인 임베디드 시스템에서 마이크로컨트롤러와 같은 임베디드 프로세서는 보통 한가지 일 이상을 해야합니다. (실제로 MCU는 한번에 한가지 일을 합니다만,,) 예를 들어 방에서 사용하는 디지털 온도조절 장치의 경우, MCU는 방안의 온도를 모니터링 해야하고, 히터를 ON/OFF 시켜야 하며, LCD 디스플레이를 제어하고 사용자가 온도를 셋팅할때에 이에 대하여 적절한 반응을 보여야 합니다. 처음 세가지 일은 MCU에게 있어 time-critical하지 않은 일이지만 사용자가 설정 버튼을 눌렀을때 반응하는 것은 바로바로 일어 나야하기 때문에 time-critical 한 일이 되는 것이며, MCU가 무슨 일을 하고 있던지 간에 사용자가 버튼을 누르면 하던일을 멈추고 보다 높은 우선순위의 일에 대하여바로 반응해야하는 것입니다. 이러한 것이 바로 인터럽트의 사용에 의하여 가능합니다. 이번 게시물에서는 인터럽트 시스템을 대략적으로 설명하고 PIC MCU에서 인터럽트를 어떻게 처리하는지 설명할 것입니다.



PIC 마이컴의 인터럽트 이해하기

이론

위에서 언급했듯이 인터럽트는 time-critical한 이벤트와 그렇지 않은 이벤트를 구분하는 기본이 되며, 일을 우선순위에 따라 처리하여 실행하는 도구입니다. 인터럽트는 MCU에게 현재 하고 있는 일을 멈추고 다른 프로그램을 실행하라는 명

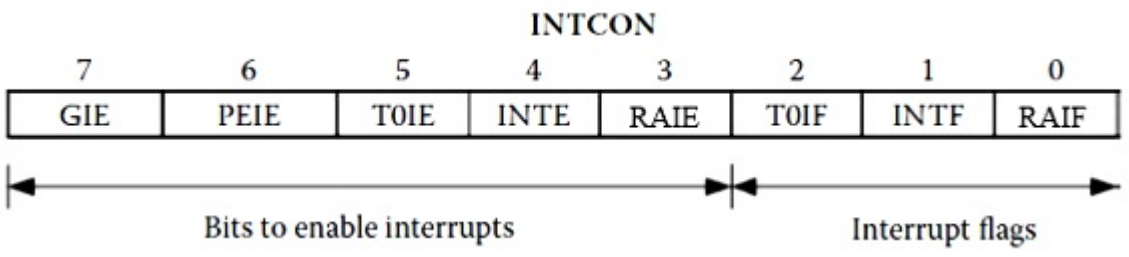
령이며, 실행되는 다른 프로그램은 프로그램 메모리의 미리 정의된 영역에 저장된 인터럽트 서비스 루틴(ISR)을 의미합니다. 인터럽트는 비동기식 이벤트인데, 이말은 인터럽트 요청이 프로그램실행 어느때나 일어 날 수 있다는 말입니다. 하지만 인터럽트 요청이 일어날때마다 CPU는 현재의 인스트럭션을 마치고 다음번 실행한 인스트럭션의 주소(Program Counter)를 스택에 저장하게 하여 인터럽스 서비스 루틴의 실행이 끝나게 되면 다시 실행할 수 있게 합니다. 그런다음 CPU는 인터럽트 서비스 루틴이 실행되는 메모리의 특정 영역으로 점프하게 됩니다. PIC MCU의 경우는 0x0004입니다. 인터럽트 서비스 루틴이 끝날때의 return 명령의 실행은 스택의 제일 위에 놓여진 주소를 다시 Program Counter로 로드하고 인터럽트가 처음 걸렸을때와 같은 지점부터 명령을 다시 실행합니다.
MCU는 여러가지 인터럽트 소스를 가지고 있는데 이러한 인터럽트 소스는 외부가 될수 도 있고 내부가 될 수도 있습니다. 아주 일반적인 인터럽트 소스는 타이머, EEPROM, ADC 모듈, comparator 입니다. 외부 인터럽트는 주변장치등으로부터 오며 MCU의 핀이나 관련된 포트를 통하여 MCU에 전달됩니다.

중간급의 PIC MCU의 인터럽트들은 고정되어 있고 maskable 인터럽트입니다. 이것이 의미하는 것은 인터럽트들이 전역적으로 활성화 되거나 비활성화 될 수 있고 각각의 인터럽트 소스가 각각 활성화 되거나 비활성화 될 수 있다는 의미입니다. 여기서는 PIC16F688 MCU의 인터럽트를 이야기 하여 보도록 하겠습니다.

PIC16F688 MCU는 아래와 같은 인터럽트 소스를 가지고 있습니다:

- External Interrupt at RA2/INT pin
- TMR0 Overflow Interrupt
- PORTA Change Interrupts
- 2 Comparator Interrupts
- A/D Interrupt
- Timer1 Overflow Interrupt
- EEPROM Data Write Interrupt
- Fail-Safe Clock Monitor Interrupt
- EUSART Receive and Transmit interrupts

언급된 인터럽트들은 각각 활성화하거나 비활성화 할수 있고 한번에 한개 이상의 인터럽트들이 활성화 될 수 있습니다. 인터럽트가 요청되어 지면 MCU는 현재의 인스트럭션을 실행하고 Program Counter값을 스택에 저장한 후 프로그램메모리의 0004h로 점프합니다. 이 주소는 인터럽트 서비스 루틴이 위치한 곳의 주소 입니다. 모든 인터럽트 소스들은 프로그램을 0004h의 주소로 점프하게 만들기 때문에, 인터럽트 서비스 루틴을 작성하는 프로그래머는 반드시 인터럽트를 요청하는 소스를 알아내야 인터럽트에 맞는 루틴을 작성할 수 있습니다. 인터럽트 소스를 알아내는 것은 인터럽트 시스템과 관련된 speical function register INTCON과 PIR1의 비트를 체크함으로서 알아 낼수 있습니다. 아래의 그림은 INTCON 인터럽트 컨트롤 레지스터의 비트들을 보여줍니다.



PIC16F688의 INTCON 레지스터

각각의 비트에 대한 설명은 아래와 같습니다:

GIE (global interrupt enable): 이 비트는 모든 인터럽트에 대해 on/off 스위치와 같습니다. GIE 비트를 0으로 셋팅하면 모든 인터럽트들이 각각의 활성화 설정 비트와 상관없이 비활성화 됩니다. GIE=1 으로 셋팅하면 인터럽트는 활성화 됩니다. 단 각각의 인터럽트는 각각의 활성화 비트가 설정되어 있어야 합니다. 인터럽트가 요청되면 GIE 비트는 자동으로 0으로 셋팅이 되어 더이상의 인터럽트 요청을 비활성화 시켜버립니다. 요청된 인터럽트의 서비스 루틴이 실행이 마무리되어 return명령이 실행될때 GIE 비트는 다시 1로 설정이 되어 전체 인터럽트 시스템을 활성화 시킵니다.

PEIE (peripheral interrupt enable) 비트는 peripheral interrupt 그룹에 대한 미니 마스터 스위치입니다. 이 비트는 peripheral itnerrupt들을 활성화 시키기 위해선 반드시 셋팅되어야 합니다. 참고로, peripheral interrupt들은 PIE1 레지스터에 각각의 활성화 비트를 가지고 있습니다.

T0IE, T0IF: 이 비트들은 Timer0 오버플로우 인터럽트와 관련이 있습니다. T0IE = 1로 설정이 될경우, Timer0 인터럽트는 활성화 됩니다. T0IF는 TMR0가 오버플로우 되어 255에서 0으로 변할때마다 1로 셋팅이 됩니다. 이 내용은 PIC타이머와 카운터 게시물에서도 언급이 되었습니다.

INTE, INTF: 이 비트들은 PIC16F688 MCU의 RA2/INT핀의 상태 종속된 외부 인터럽트와 관련이 있습니다. INTE 비트는 인터럽트를 활성화 시킵니다. 인터럽트는 INT핀의 rising혹은 falling edge에 인터럽트를 발생 시킬 수 있으며, 이것은 OPTION 레지스터의 6번 비트(INTEDG)를 사용하여 설정이 가능합니다. 아래 참조. INTF 비트는 플래그 비트로 INT 핀의 rising이나 falling edge를 검출하였을 시 셋팅됩니다.

RAIE, RAIF: 이 비트들은 PORTA핀의 로직레벨 변화에 따른 인터럽트와 관련이 있습니다. RAIE =1 일때 인터럽트가 활성화 되며 RBIF 비트는 플래그 비트로 PORTA핀중 하나에 변화가 생겼음을 의미합니다. PORTA핀의 어떤 핀이 인터럽트를 트리거할 수 있는지를 선택하려면 IOCA(INTERRUPT-ON-CHANGE-PORTA) 레지스터의 해당 비트가 반드시 셋팅되어 있어야 합니다.

OPTION레지스터의 각각의 비트 PIE1과 IOCA 레지스터는 아래와 같습니다.

OPTION_REG							
RBPU	INTEDG	T0CS	T0SE	PSA	PS2	PS1	PS0
Bit 7							Bit 0

OPTION_REG의 INTEDG 비트는 INT핀의 인터럽트 에지를 선택하는데 사용됩니다.

IOCA (Interrupt-On-Change PORTA) register							
-	-	IOCA5	IOCA4	IOCA3	IOCA2	IOCA1	IOCA0
Bit 7							Bit 0

IOCA 비트를 셋팅하면 PORTA의 해당 핀에 interrupt-on-change 인터럽트를 활성화 시킵니다.

PIE1: PERIPHERAL INTERRUPT ENABLE REGISTER 1

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
EEIE	ADIE	RCIE	C2IE	C1IE	OSFIE	TXIE	TMR1IE
bit 7							bit 0

Legend:			
R = Readable bit	W = Writable bit	U = Unimplemented bit, read as '0'	
-n = Value at POR	'1' = Bit is set	'0' = Bit is cleared	x = Bit is unknown

bit 7 **EEIE:** EE Write Complete Interrupt Enable bit
1 = Enables the EE write complete interrupt
0 = Disables the EE write complete interrupt

bit 6 **ADIE:** A/D Converter (ADC) Interrupt Enable bit
1 = Enables the ADC interrupt
0 = Disables the ADC interrupt

bit 5 **RCIE:** EUSART Receive Interrupt Enable bit
1 = Enables the EUSART receive interrupt
0 = Disables the EUSART receive interrupt

bit 4 **C2IE:** Comparator 2 Interrupt Enable bit
1 = Enables the Comparator C2 interrupt
0 = Disables the Comparator C2 interrupt

bit 3 **C1IE:** Comparator 1 Interrupt Enable bit
1 = Enables the Comparator C1 interrupt
0 = Disables the Comparator C1 interrupt

bit 2 **OSFIE:** Oscillator Fail Interrupt Enable bit
1 = Enables the oscillator fail interrupt
0 = Disables the oscillator fail interrupt

bit 1 **TXIE:** EUSART Transmit Interrupt Enable bit
1 = Enables the EUSART transmit interrupt
0 = Disables the EUSART transmit interrupt

bit 0 **TMR1IE:** Timer1 Overflow Interrupt Enable bit
1 = Enables the Timer1 overflow interrupt
0 = Disables the Timer1 overflow interrupt

PIE1 레지스터 비트는 peripheral interrupt들을 활성화 시킵니다.

PIR1: PERIPHERAL INTERRUPT REQUEST REGISTER 1

R/W-0	R/W-0	R-0	R/W-0	R/W-0	R/W-0	R-0	R/W-0
EEIF	ADIF	RCIF	C2IF	C1IF	OSFIF	TXIF	TMR1IF
bit 7							bit 0

Legend:			
R = Readable bit	W = Writable bit	U = Unimplemented bit, read as '0'	
-n = Value at POR	'1' = Bit is set	'0' = Bit is cleared	x = Bit is unknown

bit 7 **EEIF**: EEPROM Write Operation Interrupt Flag bit
1 = The write operation completed (must be cleared in software)
0 = The write operation has not completed or has not been started

bit 6 **ADIF:** A/D Converter Interrupt Flag bit
1 = A/D conversion complete (must be cleared in software)
0 = A/D conversion has not completed or has not been started

bit 5 **RCIF:** EUSART Receive Interrupt Flag bit
1 = The EUSART receive buffer is full (cleared by reading RCREG)
0 = The EUSART receive buffer is not full

bit 4 **C2IF:** Comparator C2 Interrupt Flag bit
1 = Comparator output (C2OUT bit) has changed (must be cleared in software)
0 = Comparator output (C2OUT bit) has not changed

bit 3 **C1IF:** Comparator C1 Interrupt Flag bit
1 = Comparator output (C1OUT bit) has changed (must be cleared in software)
0 = Comparator output (C1OUT bit) has not changed

bit 2 **OSFIF:** Oscillator Fail Interrupt Flag bit
1 = System oscillator failed, clock input has changed to INTOSC (must be cleared in software)
0 = System clock operating

bit 1 **TXIF:** EUSART Transmit Interrupt Flag bit
1 = The EUSART transmit buffer is empty (cleared by writing to TXREG)
0 = The EUSART transmit buffer is full

bit 0 **TMR1IF:** Timer1 Overflow Interrupt Flag bit
1 = The TMR1 register overflowed (must be cleared in software)
0 = The TMR1 register did not overflow

PIR1 레지스터는 peripheral interrupt의 플래그 비트를 포함하고 있으며, 인터럽트 동안 셋팅됩니다.

이러한 레지스터에 대한 자세한 사항은 PIC16F688의 데이터쉬트를 읽어 보십시오. 남은 게시물에서는 인터럽트를 작성하는 법, 특히 PIC16F688 MCU의 RA2/INT에서 발행하는 인터럽트를 어떻게 프로그램하는지 다루어 볼 것 입니다.

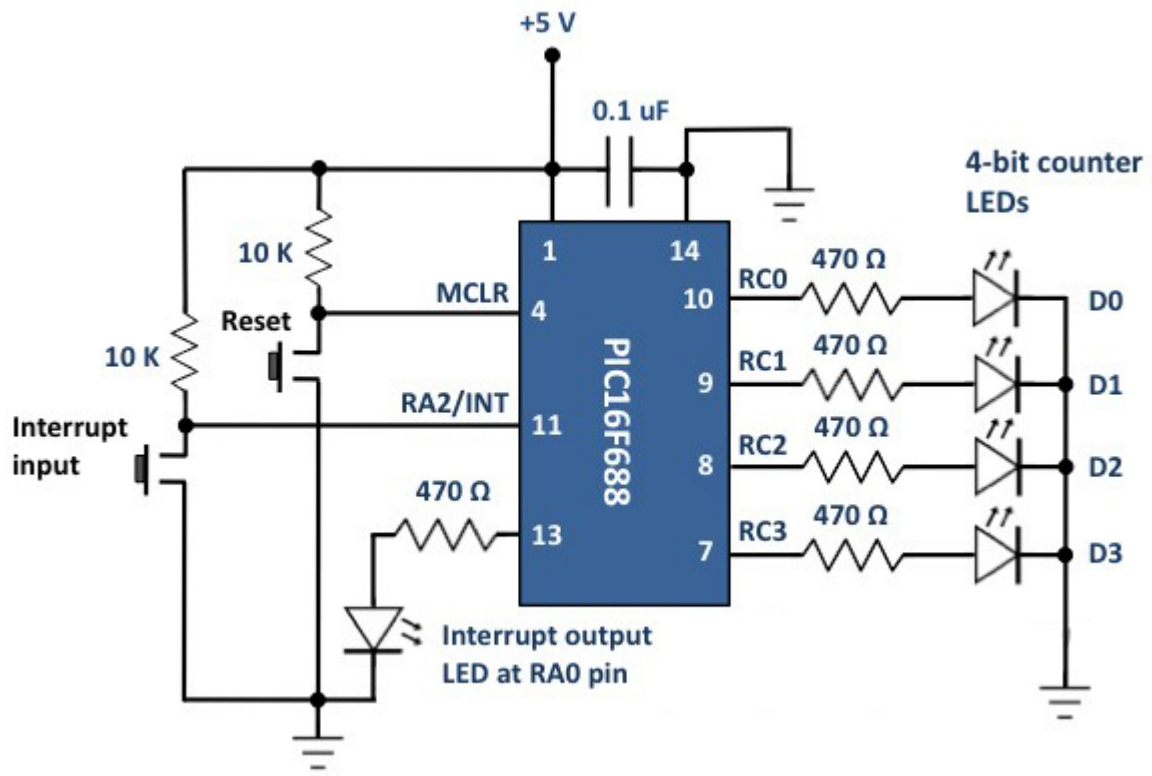
RA2/INT 인터럽트를 초기화 하기 위해서는 아래와 같은 작업이 반드시 이루어져야 합니다:

1. Port-A, line 2 (RA2) 는 반드시 입력으로 초기화.
2. 인터럽트 소스는 반드시 신호의 falling 혹은 rising 에지에서 발생하도록 셋팅. OPTION_REG 레지스터의 INTEDG비트를 이용.
3. 외부 인터럽트 플래그(INTCON 레지스터의 INTF 비트)는 반드시 클리어 되어야 함.
4. 글로벌 인터럽트는 반드시 셋팅되어야 함. INTCON레지스터의 GIE비트 이용.
5. RA2/INT 의 외부 인터럽트는 반드시 활성화 되어야 함. INTCON 레지스터의 INTE 비트 이용.

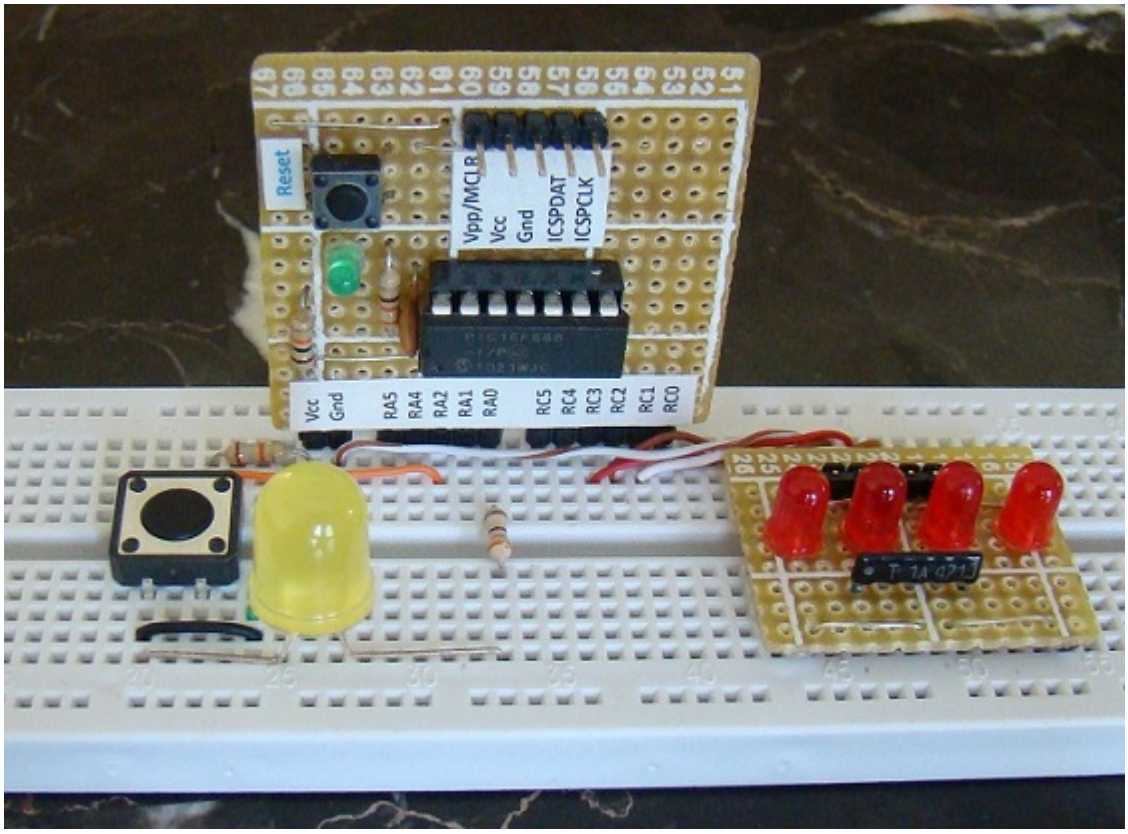
RA2/INT 인터럽트를 이용하는 샘플 프로그램은 게시물 아래에 있습니다.

회로도

아래의 회로도에서 보듯이, 푸쉬 버튼 스위치가 PIC16F688의 RA2포트에 연결되었습니다. 이 스위치는 눌렸을때 인터럽트를 발생시키도록 합니다. 4개의 LED가 RC0~RC3 포트에 연결되었고 또다른 LED가 RA0에 연결되었습니다. 메인 프로그램은 대략 1초 단위로 숫자를 증가시키는 4비트 바이너리 카운터입니다. 카운터의 값은 PORTC에 보내지고 네개의 LED에 값이 표시됩니다. 나머지 다른 LED는 푸쉬버튼이 눌렸을때 on/off됩니다. 스위치는 active low 이기때문에 인터럽트는 신호의 falling edge(INTEDG=0)에 프로그래밍 됩니다. PIC16F688은 내부 4.0Mhz 클럭에 의해 동작합니다.



외부인터럽트 동작확인을 위한 회로도



브레드보드에 회로 셋업

소프트웨어

인터럽트 서비스 루틴은 케이스별로 다르지만 일반적으로 모든 인터럽트에 적용되는 단계들이 있습니다. 예를 들어 인터럽트 서비스 루틴은 인터럽트를 시작하게 한 어떠한 이벤트를 검출하게 되면서 시작된다는 점이 그렇습니다. 본 예제의 경우, RA2/INT 핀에서 온 인터럽트 소스를 확인하기 위해서 INTCON레지스터의 INTF 플래그 비트가 체크되어야 한다는 점이 그렇고, 비슷하게 인터럽트 플래그(INTCON레지스터의 INTF 플래그 비트)가 인터럽트 서비스 루틴이 종료될때 서비스 루틴에 의해 클리어 되어야 한다는 점이 그렇습니다. 만약 클리어 되지 않으면 CPU는 인터럽트가 또 발생한 줄 알고 계속 인터터럽스 서비스 루틴으로 점프하게 됩니다.

PIC MCU에서는 인터럽트가 발생하면 INTCON레지스터의 GIE비트를 자동으로 클리어 합니다. 이 말은 인터럽트 서비스 루틴이 실행중일때는 즉 인터럽트가 발생하여 인터럽트에 대한 처리가 끝날때까지는 다른 인터럽트가 발생할 수 없다는 것을 의미합니다. 인터럽트 서비스루틴이 종료될때 GIE비트는 다시 셋팅되어 인터럽트가 활성화 됩니다.

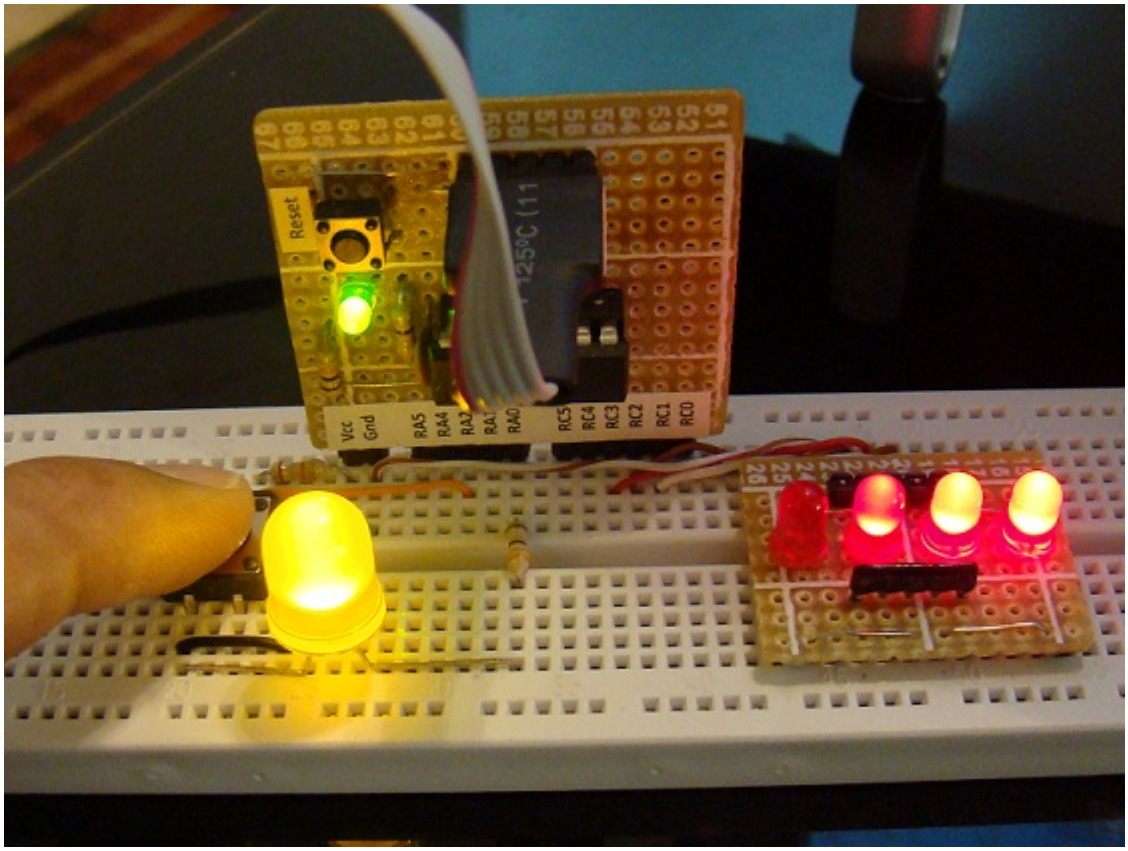
본 실험에서의 인터럽트 프로그래밍은 mikroC for PIC 컴파일러로 작성되었습니다. mikroC에서 인터럽트 서비스 루틴은 다른 서브루틴처럼 작성되지만 루틴의 이름이 interrupt로 정해져 있습니다. 인터럽트 서비스 루틴내에서, 적절한 플래그 비트를 검사하여 인터럽트가 어디서 발생했는지 알아내야 합니다. 소스코드는 아래에 있습니다.

```
/*
Lab 16: Understanding Interrupts
MCU: PIC16F688
Internal Clock @ 4MHz, MCLR Enabled, PWRT Enabled, WDT OFF
Copyright @ Rajendra Bhatt
Jul 19, 2011
*/

sbit LED at RA0_bit;  // Interrupt LED
unsigned short count;

// Interrupt service routine
void interrupt(){
 if(INTCON.INTF == 1) LED = ~LED;
 delay_ms(200);   // Key debounce time
 INTCON.INTF = 0;
}

void main() {
 ANSEL = 0b00000000;    // All I/O pins are configured as digital
 CMCON0 = 0x07 ;        // Disbale comparators
 TRISC = 0b00010000;    // PORTC all outputs except RC4
 TRISA = 0b00000100;    // PORTA all outputs except RA2
 INTCON = 0b10010000;   // Set GIE and INTE bits
 OPTION_REG.INTEDG = 0; // External interrupt on falling edge of RA2/INT pin
 count = 0;
 LED = 0;
 PORTC = count;
 do {
  count ++;
  if (count ==16) count =0;
  PORTC = count;
  delay_ms(1000);
 }while (1); // infinite loop
}
```

소스코드 다운로드

결과

MCU가 0에서 15까지 계속 카운트 하는 것을 볼 수 있습니다. 버튼을 누를때마다 MCU가 노랑 LED의 상태를 변화 시키는 것으로 인터럽트에 반응하고 다시 원래의 일인 숫자를 카운트하는 것을 볼 수 있습니다.



INT핀에서 인터럽트가 발생하였을때의 데모화면

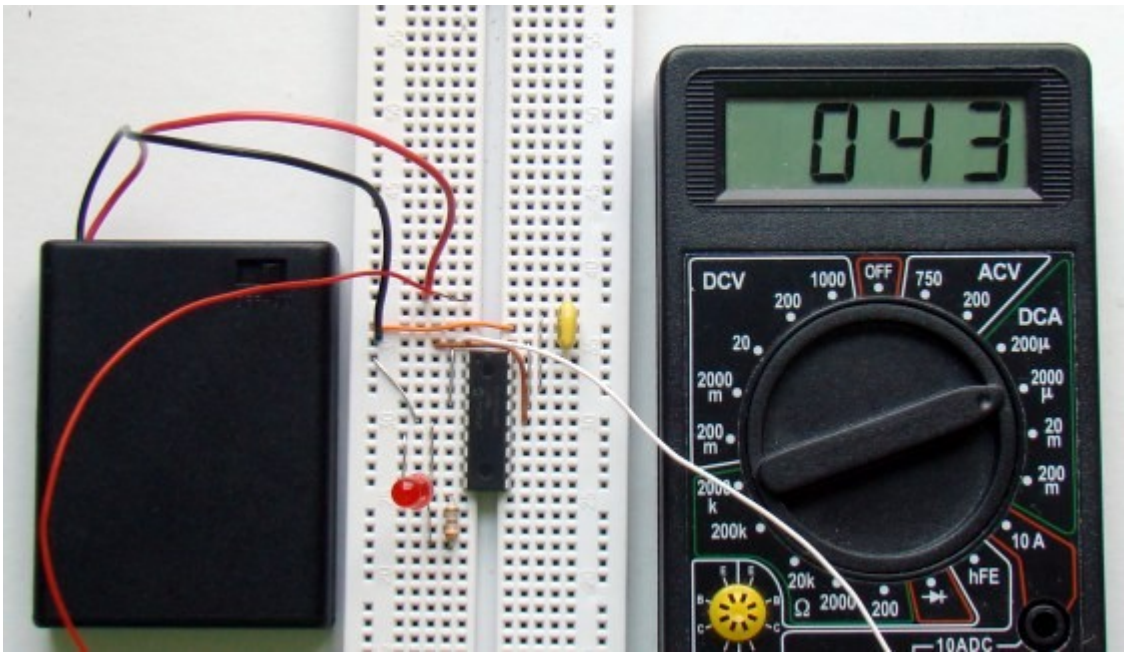
가치창조기술 | www.vctec.co.kr

[PIC 마이컴 실험하기] 17. PIC 마이크로컨트롤러의 슬립모드와 웨이크업모드

마이컴 실험실

2013/01/17 11:17 http://blog.naver.com/ubicomputing/150156726952

PIC 마이크로컨트롤러의 sleep기능은 배터리를 사용해야하는 어플리케이션에 필수적인 아주 유용한 기능힙니다. 슬립모드에서는 PIC MCU의 일반적인 동작이 멈추어지게 되고 클럭 오실레이터도 off 되게 되어 전력소비가 최소화되는 상태가 됩니다. 이런 상태의 MCU는 외부 리셋, 와치독 타이머 리셋, INT0 핀의 인터럽트 호은 포트변화 인터럽트 등에 의해 다시 깨어나게 되죠. 본 게시물에서는 PIC MCU가 어떻게 슬립모드로 진입하는지 살펴보고 슬립모드와 정상모드에서 전류 소비량이 어떻게 차이가 나는지 확인해 보도록 하겠습니다.



슬립모드 이해하기

이론

슬립모드에서 PIC MCU는 전력소비가 가장 작은 상태로 상태를 변경시킵니다. SLEEP 인스트럭션을 실행함으로써 슬립모드로 진입할 수 있으며 장치의 오실레이터는 OFF 되어 디바이스내에 아무런 시스템 클럭도 발생하지 않게 됩니다. 하지만 I/O 포트는 슬립상태가 되기전의 상태로 남아 있기 때문에, 전력소비를 최소화 하려면 슬립모드로 가기전에 출력포트를 전류의 소스나 싱크로 두는 것을 반드시 막아야 합니다. 게다가 사용되지 않는 모든 I/O 핀은 입력으로 설정되어야하고 풀업(V_{DD})시키거나 풀다운(V_{SS})시켜야 합니다.

MCU를 슬립모드에서 깨어나게 하기 위해서는 여러가지 이벤트로 가능한데,

1. 디바이스 리셋
2. Watchdog Timer Wake-up (WDT은 활성화 되어 있어야 함).
3. 슬립모드 중에서도 자신의 인터럽트 플래그를 셋팅할 수 있는 주변 모듈. 예를 들면:

- External INT pin
- Change on port pin
- Comparators
- A/D conversion
- Timer1
- LCD
- SSP
- Capture

첫번째 디바이스 리셋 이벤트는 디바이스를 리셋시켜 깨어나게 합니다만 나머지 두개의 이벤트는 MCU를 깨운후 프로그램 실행을 재개합니다. SLEEP 명령이 실행되고, 다음 명령(PC+1)이 pre-fetch되면, 프로세서가 SLEEP 명령 이후에 깨어났을때 다음 명령을 실행할 수 있게 됩니다. 인터럽트에 의해 디바이스가 깨어날려면, 해당하는 인터럽트 활성화 비트를 셋팅하여 인터럽트를 활성화 하여 주어야 합니다. Wake-Up은 GIE 비트의 상태에 상관없습니다. 만약 GIE 비트가 비활성화 되어 있다면, MCU는 단순히 슬립모드에서 깨어나 명령을 다시 실행할 것이고, 만약 GIE 비트가 활성화 되어 있다면 프로세서는 SLEEP명령 이후의 인스트럭션을 실행한 후 바로 인터럽트 주소(0004h)로 점프할 것입니다. 그래서 인터럽트를 단지 PIC MCU를 깨우려는 용도로만 사용할려면 GIE비트는 슬립모드전에 반드시 클리어해야 합니다.

Watchdog Timer (WDT)

Watchdog Timer 혹은 WDT 는 자신의 고유한 클럭 소스를 가진 독립적인 타이머 입니다. WDT는 정상적인 동작을 가로막는 소프트웨어 에러(예를들면 무한루프)가 발생했을 시 PIC MCU가 그 에러로부터 복구될 수 있는 방법을 제공합니다. WDT는 PIC MCU를 리셋시키는 타이머로 타임아웃 발생시 리셋을 수행하며, 디바이스 설정 비트 WDTE를 통하여 활성화하거나 비활성화 할 수 있습니다. 만약 활성화 시켰다면 소프트웨어 코드로 이 기능을 비활성화 시킬수는 없습니다. 때문에 WDT의 타임아웃을 막기 위해서 소프트웨어는 주기적으로 WDT를 리셋하여 주어야 합니다. PIC16F628A에서 와치독타이머는 18ms의 타임아웃 기간을 가지고 있고 이 기간은 1:128 프리스케일러를 사용하여 2.3초까지 연장 시킬 수 있습니다. 프리스케일러 rate는 OPTION레지스터의 PS0-PS2비트를 통하여 설정이 가능합니다. WDT에 프리스케일러를 할당하기 위해서는 PSA 비트도 반드시 셋팅해야하는 것을 명심하십시오. 그렇지 않으면 PSA비트는 timer0에 의해 사용이 될 것입니다.

OPTION_REG – OPTION REGISTER (ADDRESS: 81h, 181h)

R/W-1	R/W-1	R/W-1	R/W-1	R/W-1	R/W-1	R/W-1	R/W-1
$\overline{\text{RBPU}}$	INTEDG	T0CS	T0SE	PSA	PS2	PS1	PS0
bit 7							bit 0

$\overline{\text{RBPU}}$: PORTB Pull-up Enable bit
1 = PORTB pull-ups are disabled
0 = PORTB pull-ups are enabled by individual port latch values

INTEDG: Interrupt Edge Select bit
1 = Interrupt on rising edge of RB0/INT pin
0 = Interrupt on falling edge of RB0/INT pin

T0CS: TMR0 Clock Source Select bit
1 = Transition on RA4/T0CKI/CMP2 pin
0 = Internal instruction cycle clock (CLKOUT)

T0SE: TMR0 Source Edge Select bit
1 = Increment on high-to-low transition on RA4/T0CKI/CMP2 pin
0 = Increment on low-to-high transition on RA4/T0CKI/CMP2 pin

PSA: Prescaler Assignment bit
1 = Prescaler is assigned to the WDT
0 = Prescaler is assigned to the Timer0 module

PS<2:0>: Prescaler Rate Select bits

Bit Value	TMR0 Rate	WDT Rate
000	1 : 2	1 : 1
001	1 : 4	1 : 2
010	1 : 8	1 : 4
011	1 : 16	1 : 8
100	1 : 32	1 : 16
101	1 : 64	1 : 32
110	1 : 128	1 : 64
111	1 : 256	1 : 128

Set PSA bit of OPTION register to assign prescaler to WDT

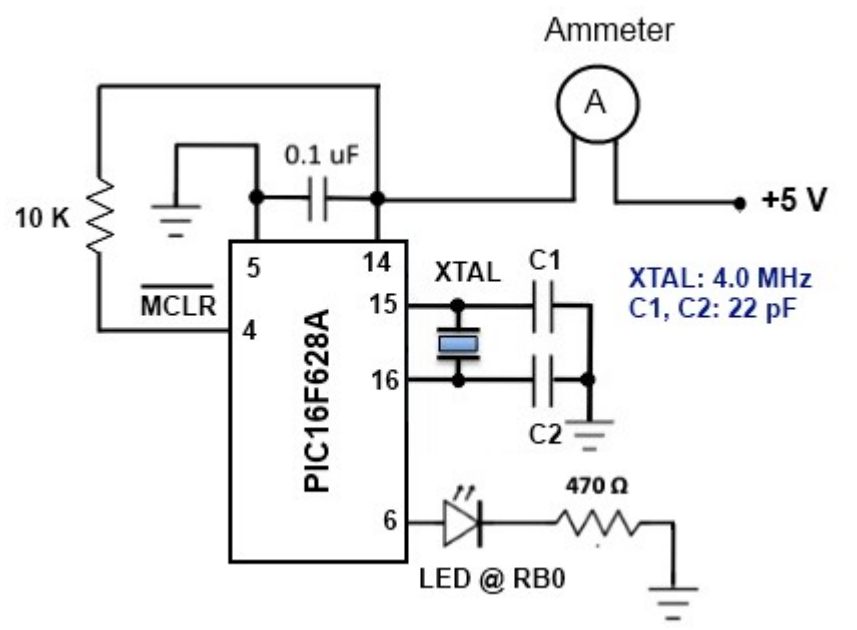
정상모드에서 와치독 타이머의 타임아웃 리셋을 방지하기 위해서, WDT 리셋 인스트럭션(CLRWDT)을 WDT 오버플로우를 방지하기에 충분히 자주 실행되는 위치의 프로그램 메인루프 안에 위치 시키게 됩니다. 만약 프로그램이 에러로 멈추게 된다면 CLRWDT 인스트럭션이 실행되지 않게 되고 program counter는 0000으로 리셋되어 프로그램이 다시 시작되게 됩니다.
하지만 PIC MCU가 슬립모드에 있다면 WDT 타임아웃은 디바이스를 리셋시키지 않고 wake-up 시키게 되고 (이러한 wake-up을 WDT wake-up이라고 합니다) MCU는 슬립모드에서 깨어나 명령을 다시 실행합니다.

주의: sleep 인스트럭션이 실행될때, WDT 타이머는 클리어됩니다. 하지만 WDT는 계속 동작합니다. (WDT가 활성화 되어 있을 경우)

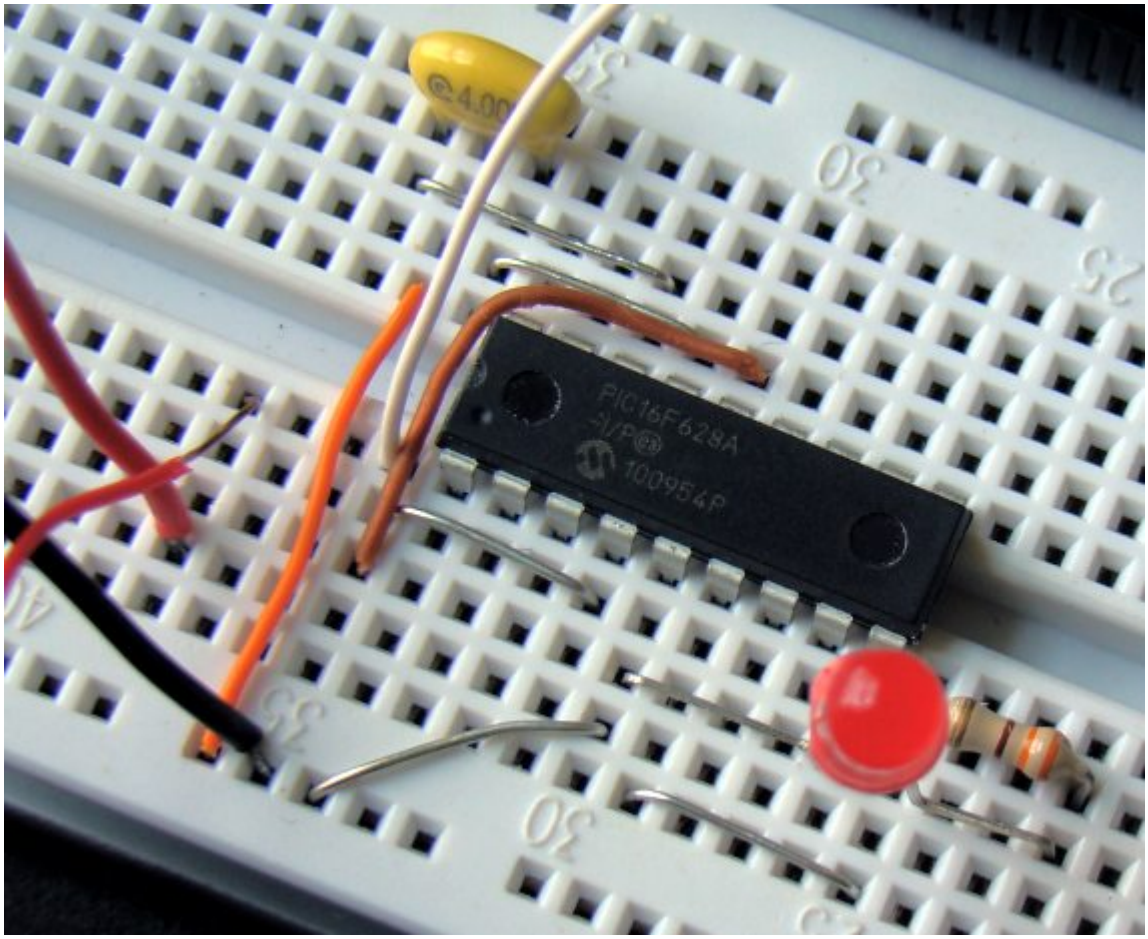
슬립모드는 주기적으로 데이터를 측정하는 데이터로거 어플리케이션 등에 꼭 필요한 기능힙니다. 배터리로 동작하는 데이터로거의 경우 측정을 마친후 슬립모드로 진입하였다가 다시 깨어나 측정을 하고 슬립모드로 진입하게 하여 배터리를 절약할 수 있습니다.

실험 셋업

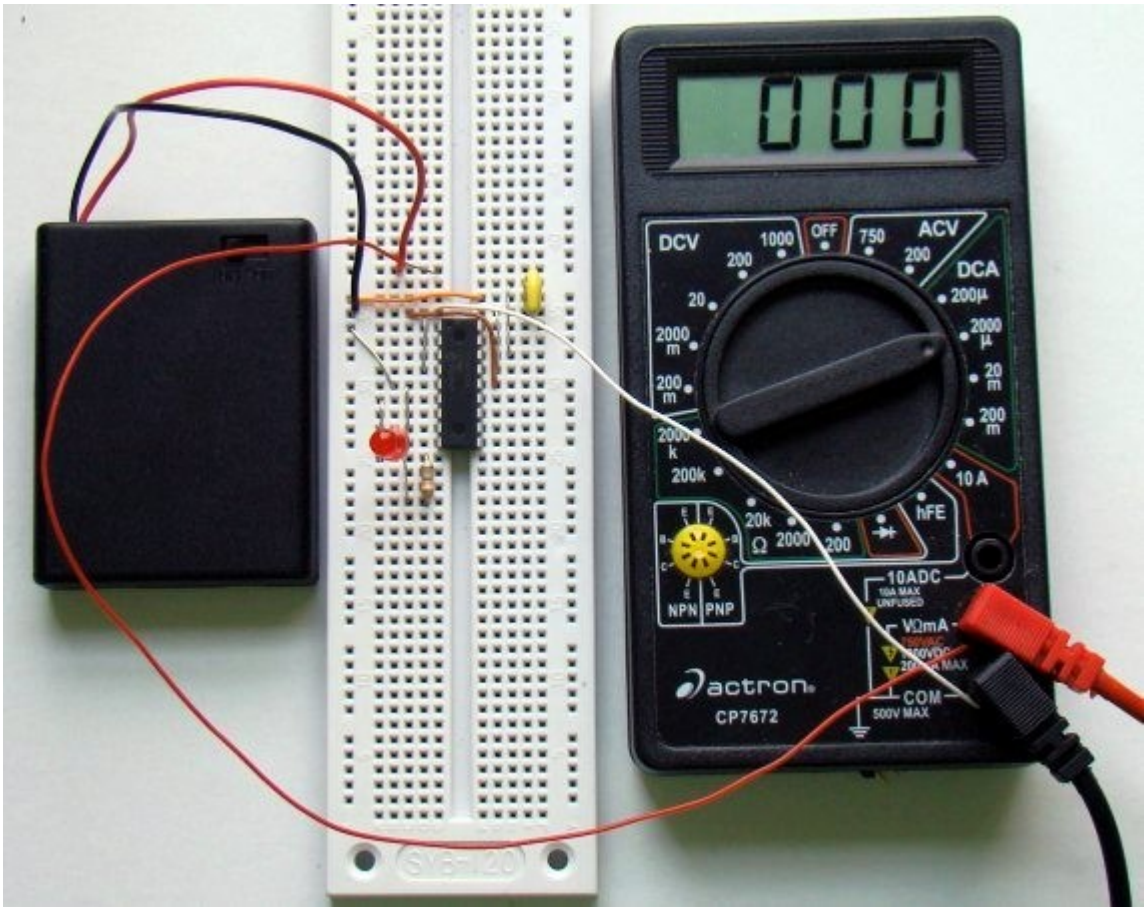
본 실험에 대한 회로도는 아래와 같습니다. PIC16F628A MCU는 내부 4.0Mhz 클럭으로 동작합니다. LED는 RB0 포트의 핀에 연결되었습니다. LED는 4.3초 딜레이 후에 0.5초동안 빛을 냅니다. 딜레이의 절반시간(2.3초)동안 MCU는 슬립모드 상태이고 WDT는 MCU를 wakeup시킵니다. 나머지 2초 딜레이는 소프트웨어 루틴(NOP명령)에 의해 생성됩니다. 전류측정기를 전원과 MCU회로 사이에 직렬로 연결하여 회로의 전류소비를 측정합니다. MCLR핀은 10k 저항을 사용하여 풀업 시켰습니다.



슬립모드 실험을 위한 회로도



회로도 셋업



회로도 셋업

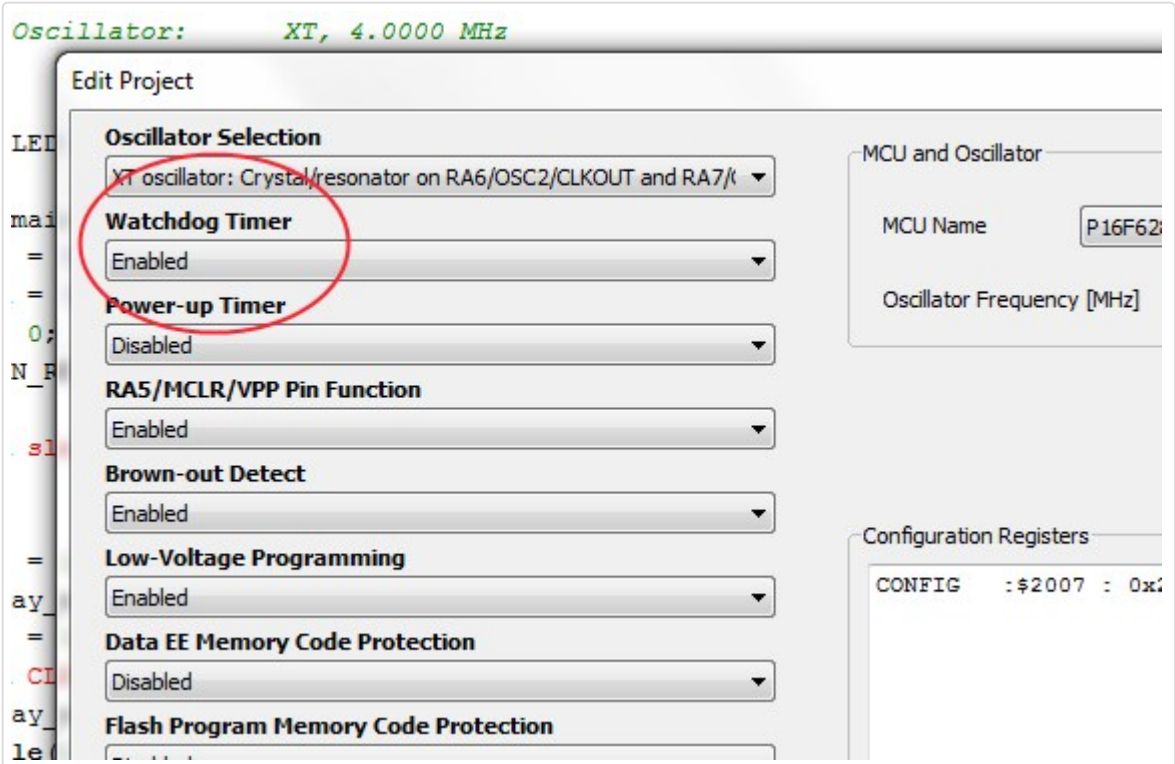
소프트웨어

아래의 소프트웨어는 mikroC for PIC 컴파일러에 의해 작성되었습니다. OPTION 레지스터는 WDT에 프리스케일러를 할당하도록 셋팅되었습니다. 1:128 비율의 프리스케일러는 WDT 타임아웃 기간을 약 2.3초로 늘려 줍니다. 추가적인 소프트웨어 딜레이 2초는 Delay_ms() 라이브러리 루틴을 이용하여 생성하였습니다. 슬립에 의한 딜레이와 정상모드에서 발생하는 딜레이 중 소비되는 전류를 측정하기 위해 전류계가 연결되었습니다.

```
/* Project name:
   Understanding sleep mode in PIC microcontrollers
 * Copyright:
   (c) Rajendra Bhatt
 * Test configuration:
   MCU:          PIC16F628A
   Oscillator:   XT, 4.0000 MHz
*/

sbit LED at RB0_bit;      // LED is connected to PORTB.0

void main() {
TRISB = 0b00000000;
TRISA = 0b00100000;
LED = 0;
OPTION_REG = 0b00001111;  // Assign 1:128 prescaler to WDT
```

```
do {
  asm sleep;            // Sleep mode, WDT is cleared,
                    // and time out occurs at Approx. 2.3 Sec
                    // Current is minimum in Sleep mode
  LED = 1;            // After wake-up the LED is turned on
  Delay_ms(500);
  LED = 0;
  asm CLRWDT;         // Clear WDT
  Delay_ms(2000);     // Measure current here and compare with Sleep mode
 }while(1);           // current
}
```

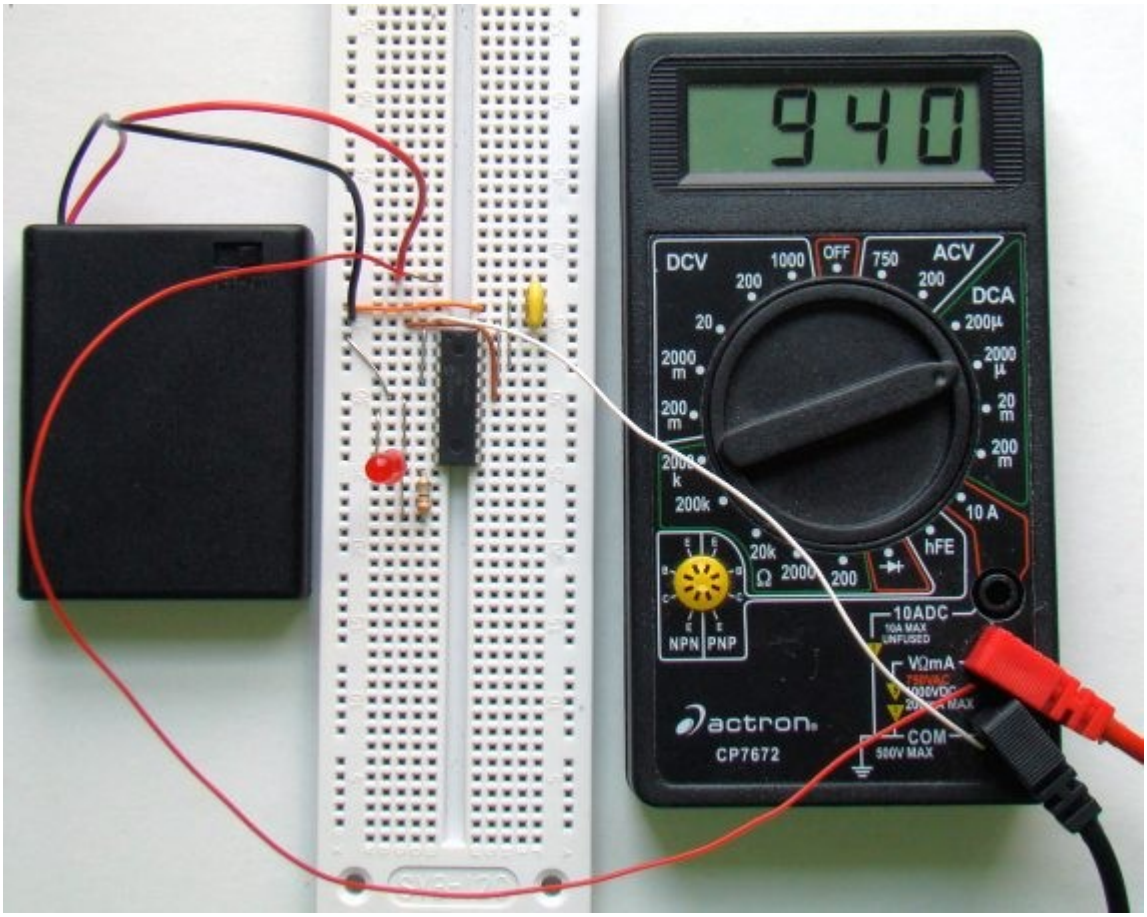
와치독 타이머는 configuration 레지스터에 설정이 되어 있어야 합니다. mikroC 컴파일러에서는 Edit Project 창에서 설정이 가능합니다. (아래 참조)



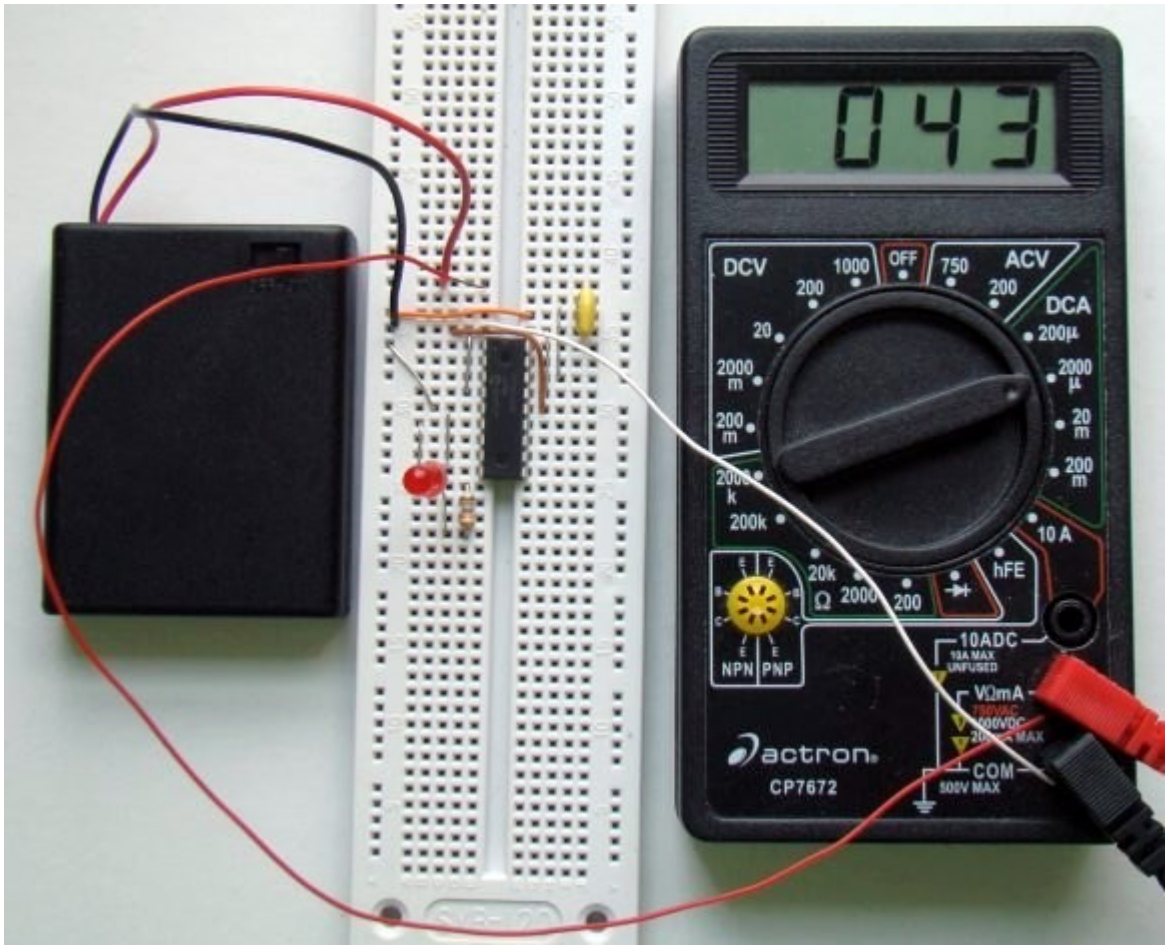
Enabling WDT from Project Edit window in mikroC Pro for PIC

결과

정상모드 상에서 소프트웨어 딜레이시(LED OFF)에 발생하는 전류 소비는 940 μA 인 반면 슬립모드에서는 43 μA 의 전류만이 소비되는 것을 확인 할 수 있습니다. 보시는대로 전류 소비차가 매우 높습니다.


940
DCV
OFF
ACV
DCA
hFE
10 A
NPN
PNP
10ADC
VΩmA
COM
actron
CP7672

정상모드시의 전류소비



슬립모드시의 전류소비

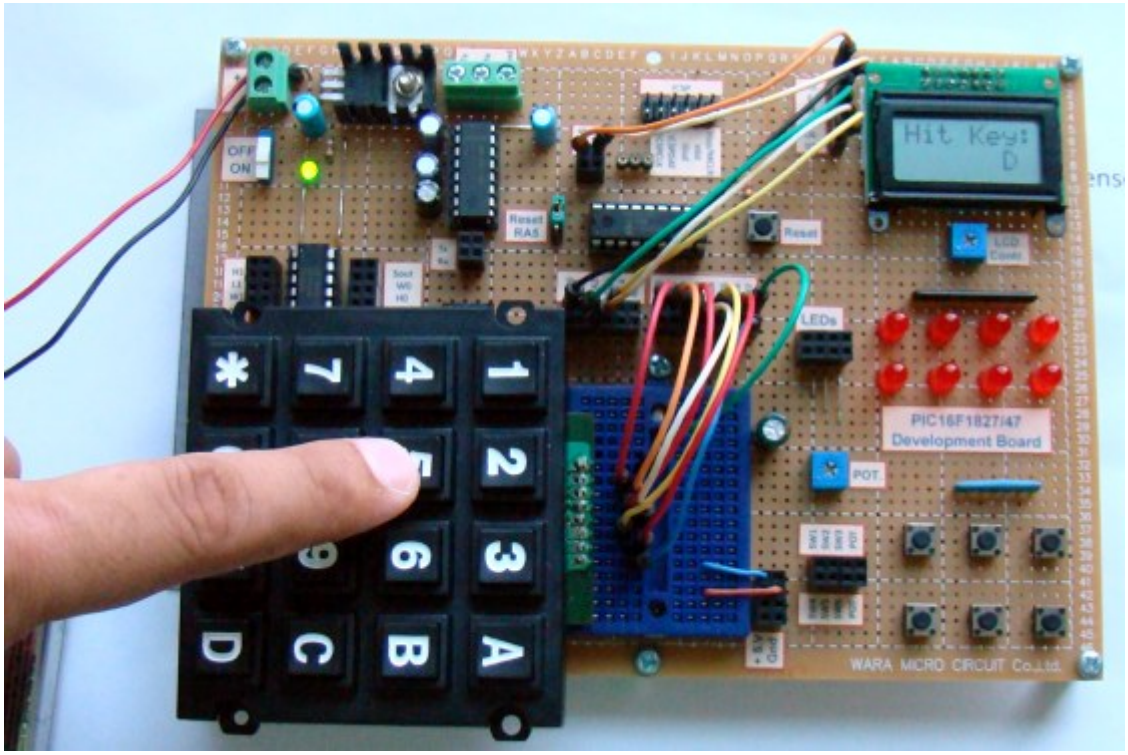
가치창조기술 | www.vctec.co.kr

[PIC 마이컴 실험하기] 18. 매트릭스 키패드 인터페이싱

마이컴 실험실

2013/01/18 10:48 http://blog.naver.com/ubicomputing/150156838609

매트릭스 키패드는 임베디드 시스템에서 자주 사용되는 입력장치로 간단한 구조를 가지고 있으며 쉽게 연결하여 사용할 수 있습니다. 좋은 장점중 하나는 MCU의 I/O 자원을 최소한도로 사용하면서 많은 키 입력을 사용할 수 있다는 점에 있습니다. 본 게시물에서는 PIC MCU에 연결되어 있는 4x4 매트릭스 키패드로부터 데이터 입력을 읽어 들이는 두개의 서로 다른 접근은 설명하여 보겠습니다.
눌린 키값은 캐릭터 LCD에 표시되고 이 예제에서 사용된 MCU는 PIC17f1827입니다.

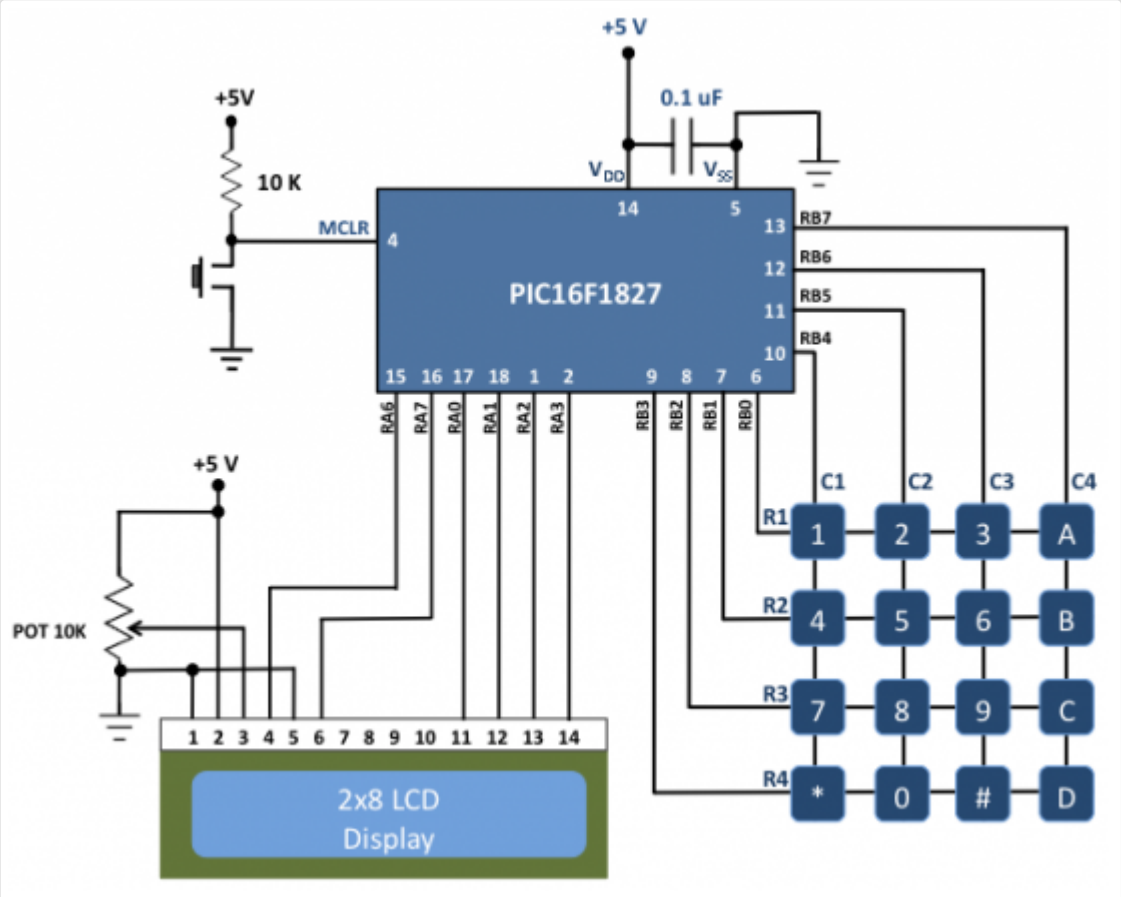


PIC 에 연결된 4x4 매트릭스 키패드

이론

매트릭스 키패드는 간단히 말하면 택트 스위치의 확장판으로 매트릭스 형태로 연결된 키들로 구성이 되어 있습니다. 각각의 키는 간단한 기계적인 스위치로 열과 행의 교차하는 지점에 위치하고 있습니다. 키가 눌리게 되면 키의 열과 행은 전기적인 접촉을 이루게 됩니다. 이러한 열과 행은 MCU 포트 핀에 연결이 되는데, 열 4개와 행4개를 MCU의 I/O핀에 연결(8라인 연결)하게 되면 총 16개의 키를 사용할 수 있게 된다는 점이 큰 장점입니다. 즉 16개의 라인을 MCU의 I/O핀에 연결할 필요가 없다는 이야기지요. 아래의 회로도를 보면 4x4 매트릭스 키패드가 PIC16F1827의 PORTB에 연결되어 있는 것

을 확인 할 수 있습니다. MCU는 내부 클럭을 사용하여 500Khz로 동작되고 있습니다. PORTA 핀은 눌린 키 정보를 표시하기 위한 캐릭터 LCD를 구동하기 위해 사용됩니다.



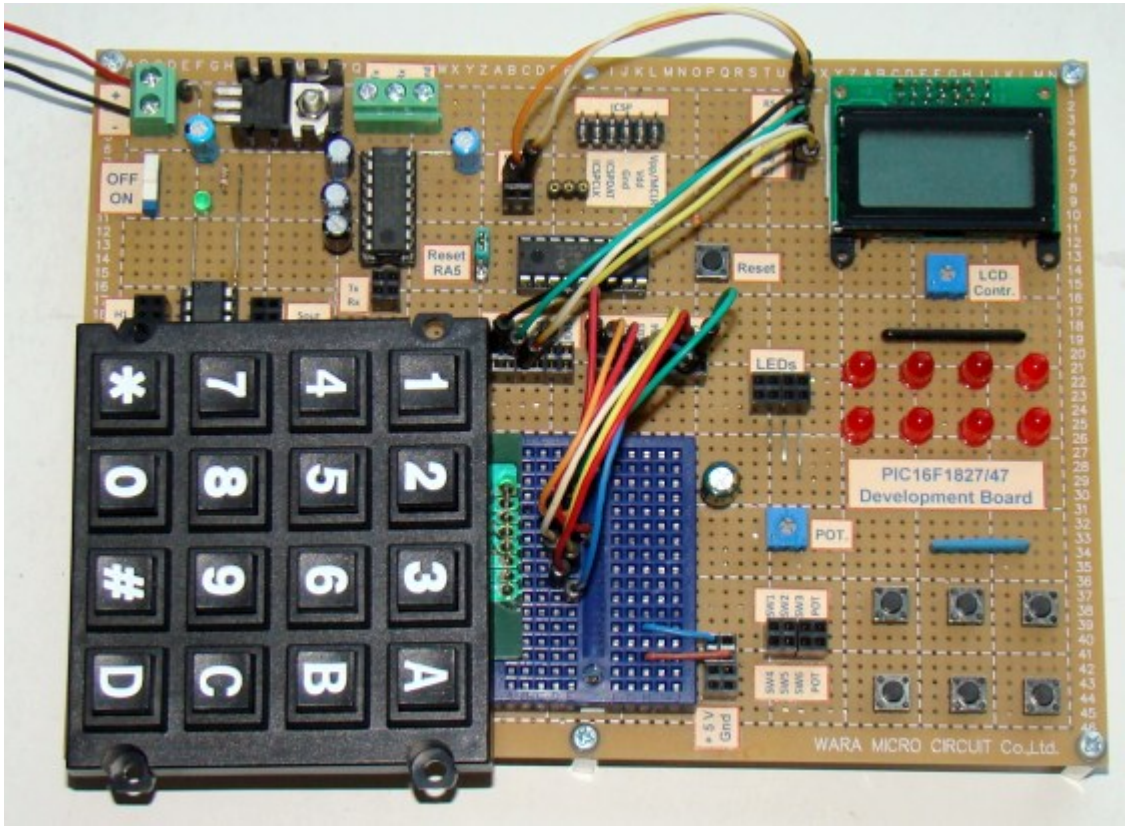
PIC 에 매트릭스 키패드 연결하기

이제 어떤 버튼이 눌렸는지 알아내기 위해 키패드를 스캐닝하는 두개의 접근법을 이야기하도록 하겠습니다.

행 순차 탐색: 이 방법에서는 행에 연결되어 있는 모든 I/O핀들이 출력으로 설정되어 있고 열에 연결되어 있는 모든 I/O핀들은 입력으로 설정되어 있습니다. MCU에 입력으로 연결되어 있는 열에 있는 라인들은 내부 풀업저항을 사용하여 풀업되어 있어, 이 라인들의 기본 입력값은 1입니다. 키패드의 상태는 각각의 행을 순차적으로 한번에 한 행씩 탐색하면서 열의 값을 읽으면 알 수 있습니다. 예를들어, 첫번째 행을 0으로 설정하고 모든 열을 읽습니다. 첫번째 열의 어떤 키가 눌렸다면 그 키의 열의 값은 반드시 0으로 읽히게 됩니다. 이러한 식으로 다음 행을 0으로 설정하고 열의 값들을 다시 읽는 방법을 반복하여 어떠한 키가 눌렸는지 알아 냅니다.

행과 열 동시 탐색: 이 방법에서는 모든 행과 열이 두개의 단계를 거쳐 동시에 탐색됩니다. 먼저 행 라인들이 출력으로, 열 라인들은 입력으로 설정합니다. 내부 풀업저항으로 열 라인을 풀업시킵니다. 그 후 모든 행을 0으로 셋팅하고 열을 읽습니다. 만약 어떤 키가 눌렸다면 해당하는 열은 0으로 읽히게 됩니다. 이렇게 하면 버튼이 눌린 열은 찾아 낼수 있지만 행은

찾지 못하게 되어, 이번에는 좀전에 수행하였던 방법을 반대로 수행합니다. 행을 입력으로, 열을 출력으로 설정하고 행을 풀업 시킵니다. 모든 열을 0으로 설정하고 행을 읽습니다. 이러한 접근방법은 순차적으로 탐지하는 방법보다 빠른 방법입니다.
만약 MCU가 I/O포트에 내부 풀업저항을 지원하지 않는다면 반드시 외부에 풀업저항을 연결하여 주어야 합니다.

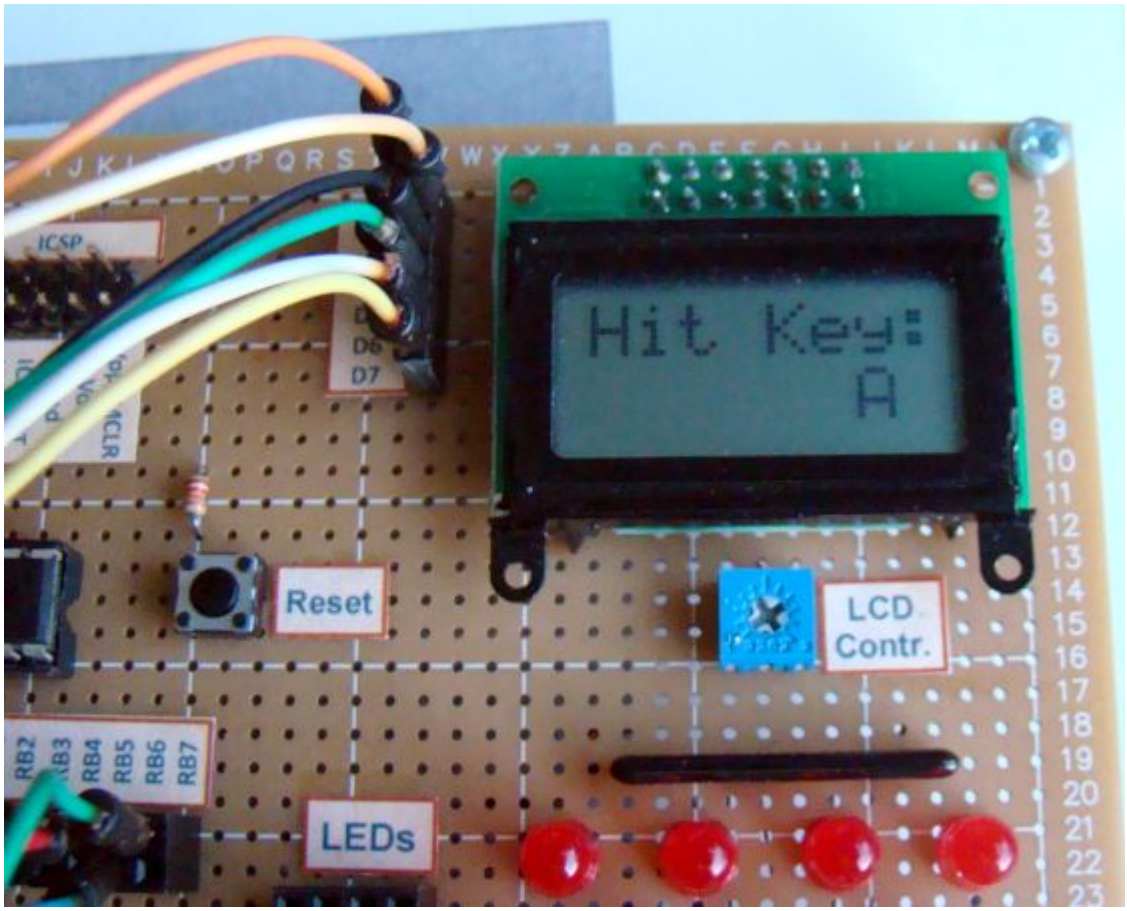


PIC16F1827 의 PORTB에 연결된 키패드의 모습

소프트웨어

아래는 mikroC for PIC 컴파일러로 작성된 키패드 스캔 루틴입니다. 언급된 두개의 접근 방법에 대한 코드입니다. PORTB핀에 약한 풀업저항을 주기 위하여 OPTION_REG의 WPUEN(Weak Pull-Up on PORTB)핀을 반드시 클리어 해야하고 PORTB의 WPUB(Weak Pull-Up on PORTB) 레지스터의 해당 비트를 반드시 설정해주어야 합니다. mikroC에서는 내장된 키패드 라이브러리가 있기 때문에 이러한 루틴이 필요없지만, 여기서는 내장된 라이브러리 루틴을 사용하지 않고 작성하였습니다.

mikroC 소스코드



눌린 키가 디스플레이되는 화면

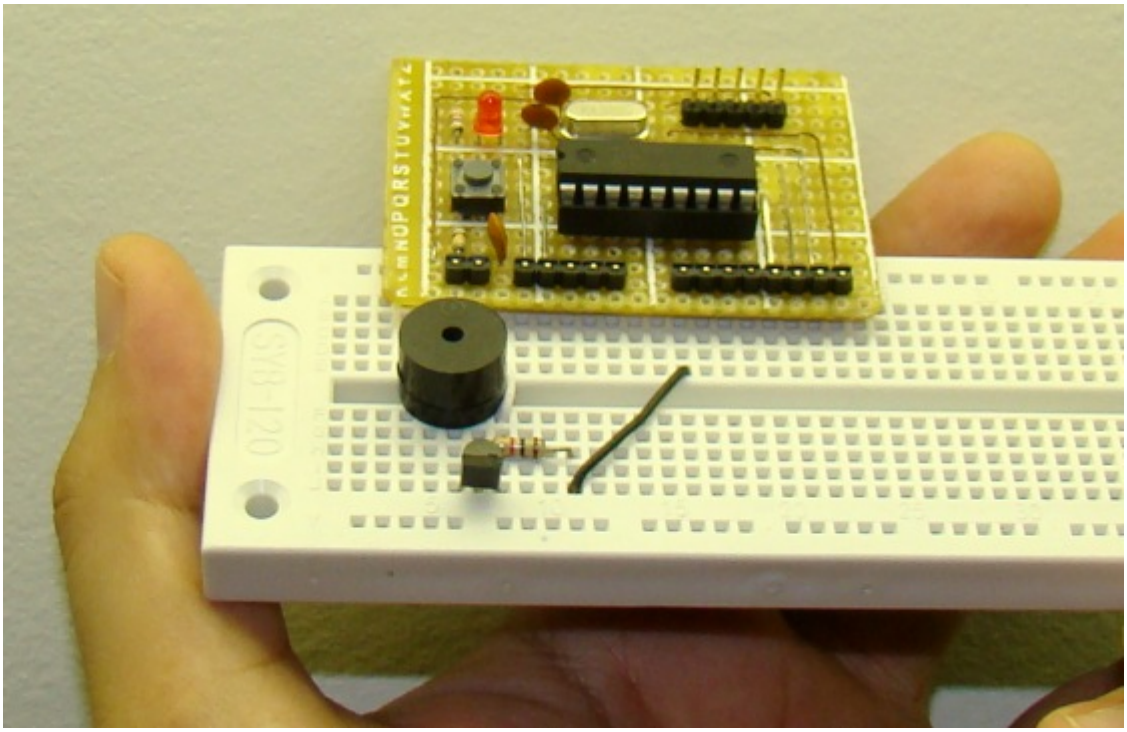
가치창조기술 | www.vctec.co.kr

[PIC 마이컴 실험하기] 19. 음계 연주하기

마이컴 실험실

2013/01/21 13:42 http://blog.naver.com/ubicomputing/150157129289

음계는 특정 주파수의 소리 파동입니다. 음계의 주파수를 정확히 알수있다면 PIC MCU가 같은 주파수의 스퀘어 웨이브를 I/O핀중에 하나에 생성시킴으로 음계를 연주 할 수 있습니다. 이번 게시물에서는 PIC16F628 MCU와 버저로 생일축하곡을 연주하는 방법에 대해 살펴보도록 하겠습니다.



PIC MCU를 이용하여 음악 연주하기

이론

음악을 연주하기 위해서는 음계를 알아야 합니다. 각각의 음이 일정 기간 연주되어야 하며, 음간에는 일정시간동안의 간격이 있어야 합니다. 아래의 테이블은 C부터 시작하는 음계의 주파수를 보여줍니다. 중간 C는 C4라고도 하는데 이는 표준 88키 피아노 키보드의 4번째 C 건반이기 때문입니다.

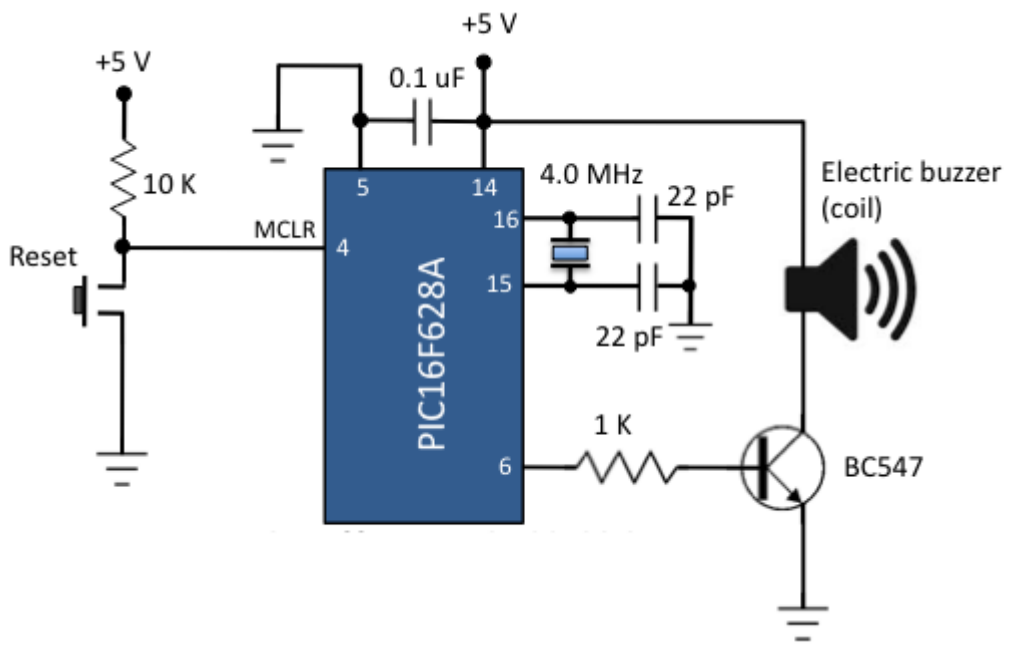
Notes	C	C#	D	D#	E	F	F#	G	G#	A	A#	B
Hz	262	277	294	311	330	349	370	392	415	440	466	494

음도와 각각의 주파수

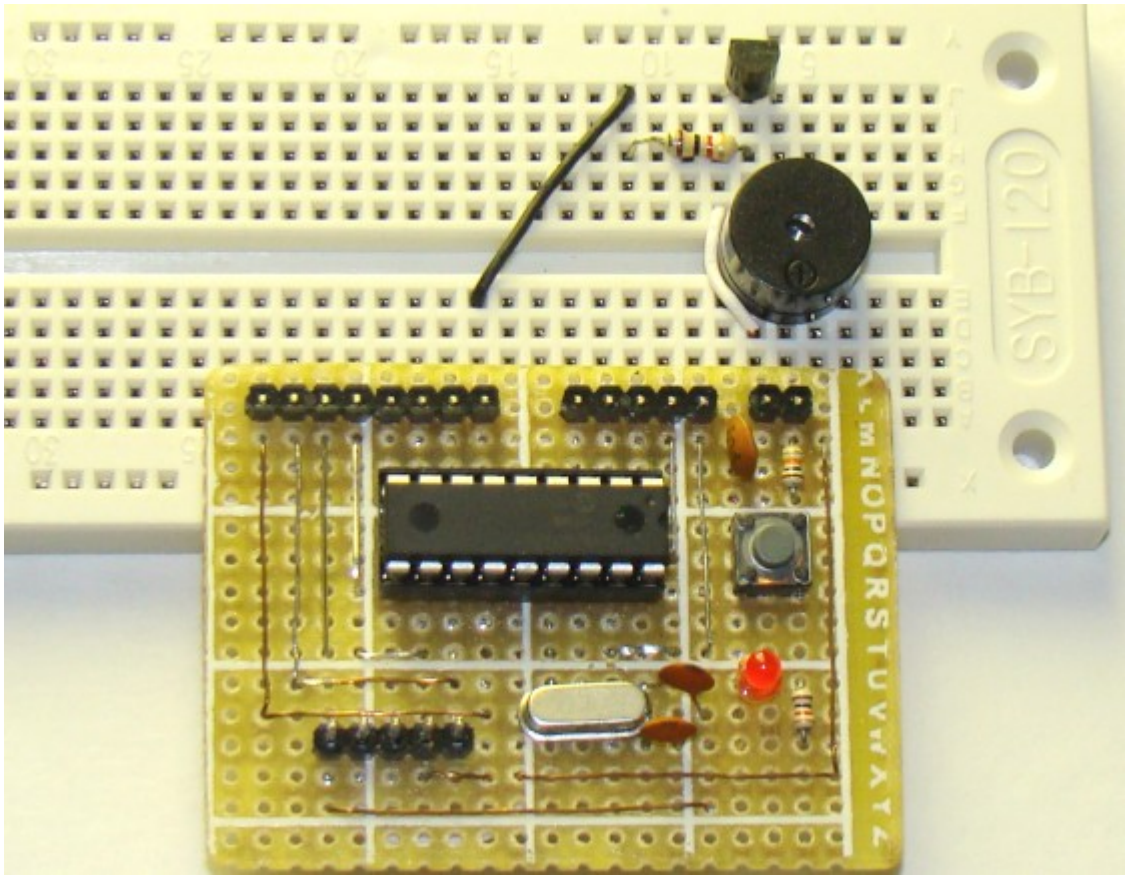
다른 옥타브의 음들은 위의 테이블에 2를 곱하거나 나눔으로써 구할 수 있습니다. 예를 들어 C4의 다음번 C 음은 524Hz 입니다.
음은 음의 주파수의 스퀘어 웨이브를 이용하여 생성할 수 있습니다. 그래서 노래를 연주하기 위해서는 알아야 모든 것은 노래의 음도와 타이밍 정보이며 나머지는 다 프로그래밍과 관련된 것들입니다. 스퀘어 웨이브는 MCU의 I/O핀을 하이나 로우로 변환시킴으로써 얻을 수 있습니다. 또다른 방법은 PWM 모듈을 사용하는 것인데 여기서는 첫번째 것을 사용하도록 하겠습니다.

회로도

본 실험을 위한 회로도는 매우 간단합니다. PIC16F628A의 RB0핀은 원하는 주파수의 스퀘어 웨이브를 만들기 위해 비트 변환을 만들어줍니다. PIC16F628A는 I/O핀은 25mA의 전류를 내보낼 수 있는데, 이것은 버저를 직접적으로 동작시키기에 충분하지 않을 수 있습니다. 그래서 전류를 높이기 위해 BC547 NPN 트랜지스터를 사용하였습니다. 오디오의 음질을 향상시키기 위해서 RC필터를 사용할 수 있습니다만 여기서는 사용하지 않고 실험을 진행합니다. 참고로 스퀘어 웨이브는 사인 웨이브만큼 좋은 소리가 나지 않습니다.



RB0핀과 트랜지스터를 사용하여 코일 버저를 동작



회로셋업

위의 그림에서 사용된 모듈은 다음 링크를 참고하세요. 브레드보드 모듈 링크

소프트웨어

mikroC 컴파일러에서 오디오 톤을 생성하는 것은 매우 쉬운데, 이러한 목적으로 내장된 라이브러리를 포함하고 있기 때문입니다. 라이브러리는 아래의 두개의 함수를 가지고 있습니다.

Sound_Init(**char** *snd_port, **char** snd_pin): 소리 생성을 위해 적당한 MCU핀을 설정하는 함수. 예를 들어, Sound_Init(&PORTB, 0) 함수의 실행은 RB0핀을 소리출력 핀으로 설정합니다.

Sound_Play(**unsigned** freq_in_hz, **unsigned** duration_ms): 핀에 스퀘어 웨이브 신호를 생성합니다.

노래의 음도에 대한 주파수는 MCU의 롬이나 메모리에 가변배열로 정의 할 수 있습니다. 생일 축하 노래는 크기가 크지 않으므로 mikroC에서 integer 타입으로 아래와 같이 정의 되었습니다.

```
/*                Hap  py  Birth Day  to  you,  Hap py   birth day  to
                 C4   C4   D4   C4   F4   E4   C4   C4   D4   C4   G4 */
```

```
unsigned int notes[] = { 262, 262, 294, 262, 349, 330, 262, 262, 294, 262, 392,

/*                you, Hap py  Birth Day  dear  xxxx      Hap  py   birth
                 F4   C4   C4   C5   A4   F4   E4   D4   B4b  B4b  A4 */
                 349, 262, 262, 523, 440, 349, 330, 294, 466, 466, 440,

/*                day  to  you
                 F4   G4   F4   */
                 349, 392, 349
                };
```

비슷하게 각각의 음도에 대한 연주기간도 인티저 형태의 배열에 정의 될수 있습니다. 소스코드는 아래의 링크에서 다운로드 받을 수 있습니다. 소스코드 다운로드

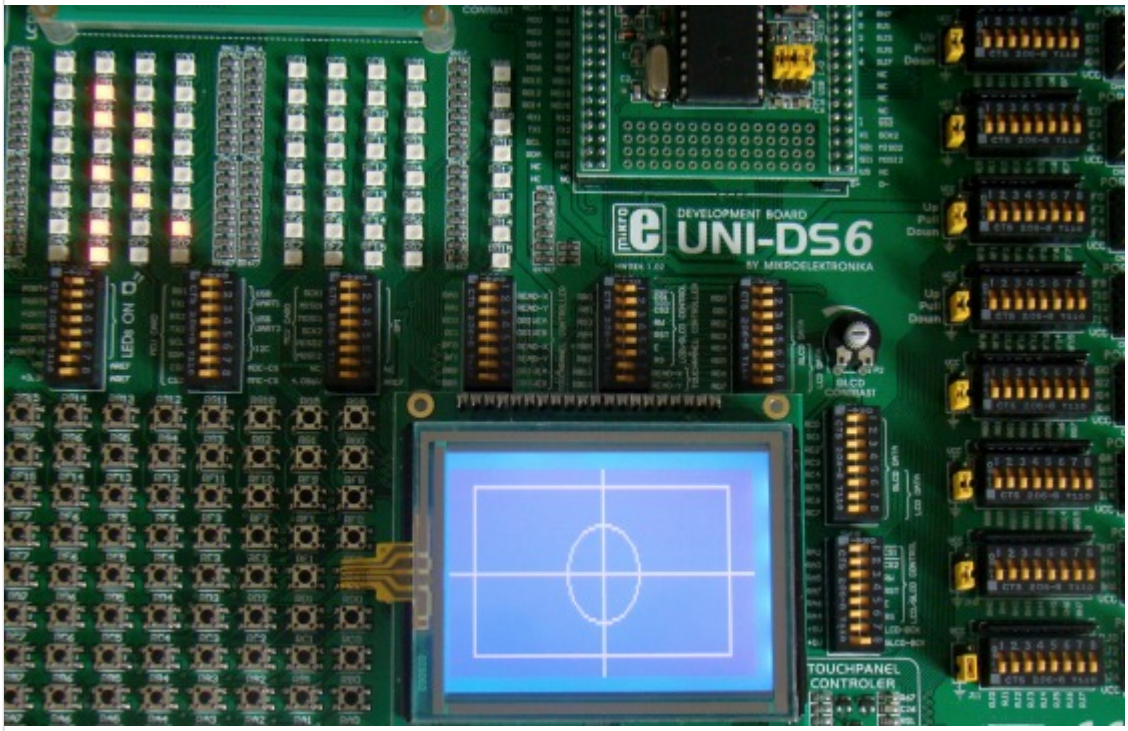
가치창조기술 | www.vctec.co.kr

[PIC 마이컴 실험하기] 20-1. KS0108기반의 그래픽 LCD 인터페이싱 하기

마이컴 실험실

2013/01/22 11:42 http://blog.naver.com/ubicomputing/150157235434

개발하고자 하는 프로젝트에 GLCD(그래픽 LCD)를 사용할 경우에는 HD44870 기반의 문자 LCD를 사용하는 것보다 데이타를 표현하는데 있어 좀 더 큰 자유로움을 경험할 수 있습니다. 이 번 게시물에서는 KS0108 기반의 GLCD를 PIC MCU에 인터페이싱 하는 방법에 대해여 설명하도록 하겠습니다. 참고로 KS1018은 GLCD의 컨트롤러 칩이름입니다. 이 번 게시물은 크게 두 파트로 나뉘어 있는데 첫번째 파트에서는 GLCD를 초기화 하고 점선을 디스플레이하는 펌웨어를 PIC MCU에 어떻게 쓰는가 하는 부분을 설명하고, 두번째 파트는 좀더 복잡한 텍스트와 객체를 디스플레이 하기 위한 mikroC 컴파일러의 라이브러리에 대해 살펴보도록 하겠습니다. GLCD는 I/O핀과 메모리를 많이 사용해야하는 장치이기 때문에 36개의 I/O와 14KB의 플래쉬 메모리를 가지고 있는 PIC16F887가 이 실험에서 사용이 되었고, 타켓보드로는 마이크로일렉트로니카의 개발보드가 사용되었지만 회로는 브레드보드에서도 셋업이 가능합니다.

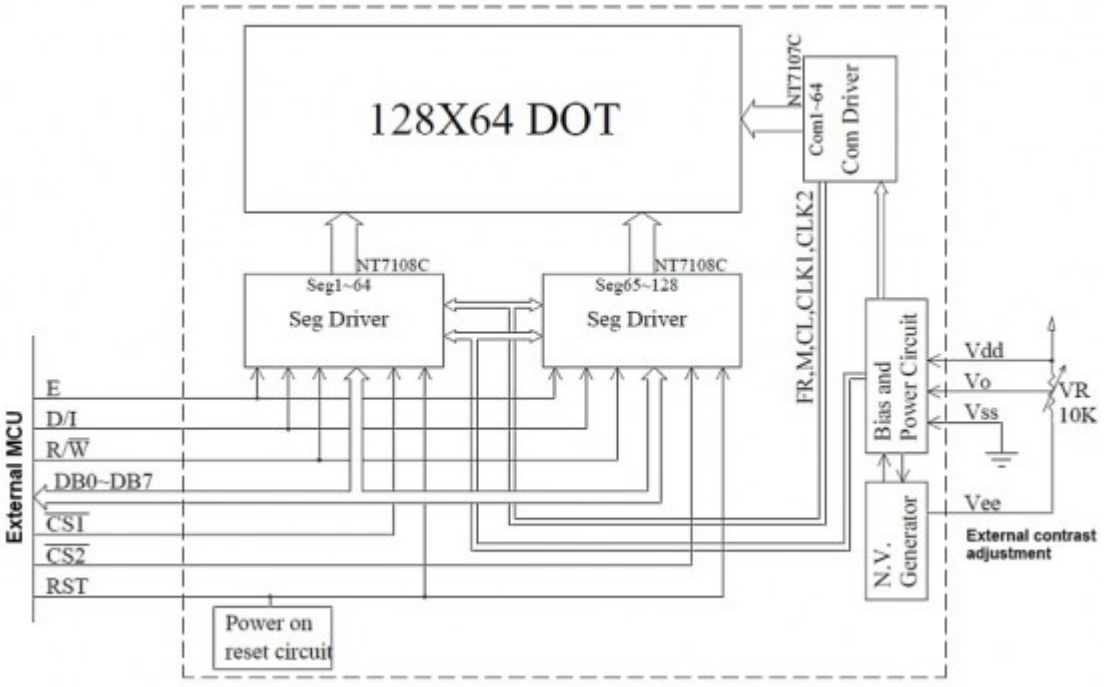


128x64 픽셀 GLCD 연결하기

이론

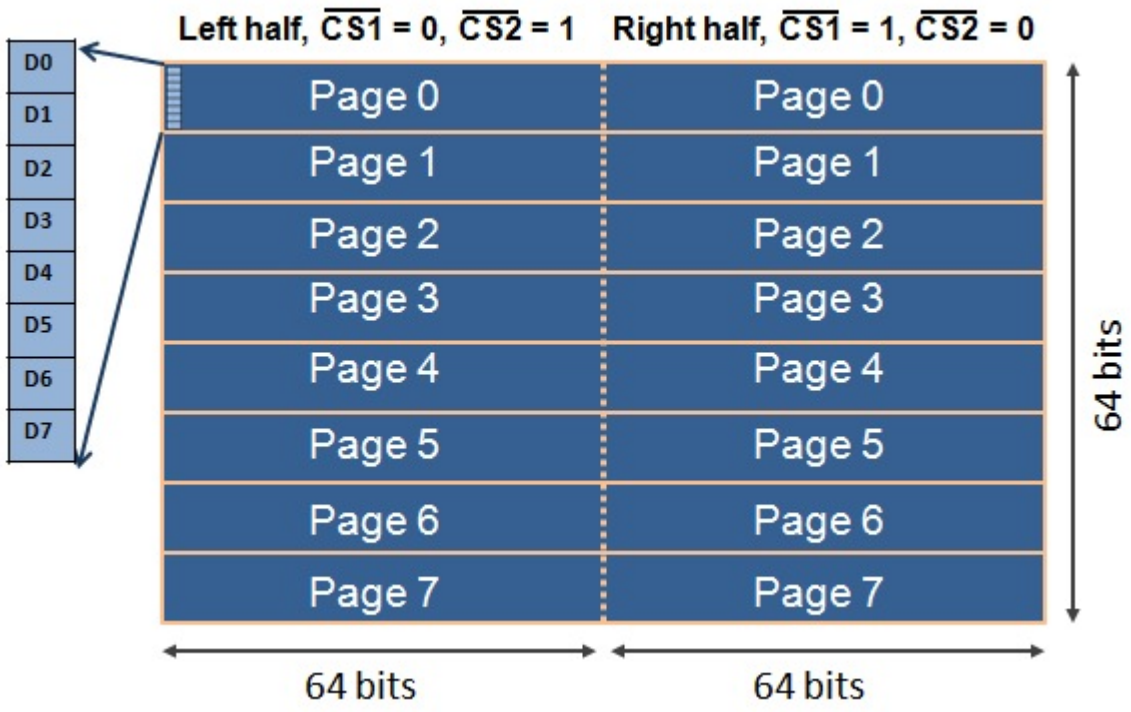
여기서 사용된 그래픽 LCD는 Winstar의 WDG0151-TMI 모듈로 128x64픽셀의 단색 디스플레이입니다. NT7108C와 NT7107C 두개의 컨트롤러 칩을 사용하는데 이 칩은 삼성 KS0108B와 KS0107B 컨트롤러와 호환됩니다. KS0108B(혹

은 NT7108C) 컨트롤러는 64개의 채널 출력을 가진 도트 매트릭스 LCD 세그먼트 드라이버이기때문에 WDG0151 모듈은 128 세그먼트를 동작시키기 위해 두개의 컨트롤러를 가집니다. 다시말하면 KS0107B(혹은 NT7107C)는 64 채널을 가진 드라이버로 두개의 KS0108B 세그먼트 드라이버를 동작시키기위한 타이밍 신호를 생성합니다. KS0108B와 KS0107B는 매우 인기 있는 컨트롤러로 많은 그래픽 LCD에서 사용이 됩니다. 아래는 WDG0151 GLCD 모듈의 내부 블럭 다이어그램입니다.



KS0108B (NT7108C)기반 128x64 픽셀 GLCD모듈의 내부 블럭 다이어그램

NT1707C 칩은 COM1에서 COM64까지의 64개의 디스플레이 라인을 동작시킵니다. 첫번째 NT7108C 칩은 좌측 절반의 세그먼트(SEG1~SEG64)를 동작시키고 나머지 NT7108C칩은 우측 절반의 세그먼트(SE65~SEG128)을 동작시킵니다. 디스플레이의 각각의 절반은 NT7108C 드라이버의 칩 셀렉트 핀(CS1과 CS2)을 통해 개별적으로 접근이 가능합니다. 각각의 절반은 8비트 높이의 8개의 페이지로 아래의 그림과 같이 구성되어 있습니다.



GLCD 페이지 구조와 칩 셀렉트

좌측 절번의 페이지 0부터 시작해서 만약 한 바이트의 데이터를 전송한다면, 전송된 데이터는 페이지 0의 첫번째 열에 나타날 것입니다. 한바이트 전송을 64번 반복한다면, 좌측에서 우측 절반으로 넘어가게 되고 128번째 위치에 다다를때까지 반복되게 됩니다. 그러므로 128x64픽셀의 디스플레이 프레임에 필요한 총 바이트는 2 * 64 pixels * 8 bits = 1024 바이트 입니다.

Winstar WDG0151-TMI 모듈은 내부에 음전압 발생회로를 가지고 있는데 V_{EE} 외부 핀에 음전압을 제공할 수 있습니다. 외부 포텐티오미터(보통 10K)를 Vcc와 Vee핀 사이에 연결하여 Vo 핀에 LCD 동작 전압(contrast)을 설정할 수 있습니다. KS0108기반 GLCD의 핀 다이어그램은 표준화 되어 있지 않기때문에 GLCD를 올바르게 연결하기 위해서는 제조사의 데이터쉬트를 꼭 참조하셔야 합니다. 아래의 테이블은 윈스타 WDG0151-TMI 모듈의 핀 설명입니다. 총 20개의 핀을 가지고 있으며, 처음 두개의 핀은 좌측 및 우측 컨트롤러를 선택하는 칩 셀렉트 핀으로 active low입니다.(다른 모델에서는 active high일수 있습니다.) WDG0151-TMI 모듈은 5v에서 동작합니다. 6번 핀은 Data/Instruction 선택 핀으로 RS(Register Select)핀으로도 알려져 있습니다. GLCD의 D0-D7핀에 공급되는 8비트 데이터는 D/I = 0일경우 LCD 컨트롤러는 명령으로 그것을 인식하며, D/I=1일 경우 데이터로 인식합니다. R/W와 E핀은 HD44780기반의 문자LCD와 비슷한 기능을 가지고 있습니다. 백라이트 LED 19번과 20번핀에는 전류의 양을 제한하기 위해 고정된 값을 가진 저항이 반드시 직렬로 연결되어야 합니다.

Pin no.	Symbol	Level	Description
1	$\overline{CS1}$	L	Select segments 1-64
2	$\overline{CS2}$	L	Select segments 65-128
3	V_{SS}	0 V	Ground
4	V_{DD}	5.0 V	Supply voltage for logic
5	Vo	Variable	Contrast adjustment
6	D/I or RS	H/L	H : Data, L: Instruction
7	R/W	H/L	H: Read data, L: Write data
8	E	H	Enable signal
9-16	D0-D7	H/L	Data bus
17	RST	L	Reset the LCD module
18	V_{EE}		Negative voltage output
19	A		Anode (+) of B/L LED
20	K		Cathode (-) of B/L LED

Winstar WDG01510 GLCD 모듈의 핀 설명

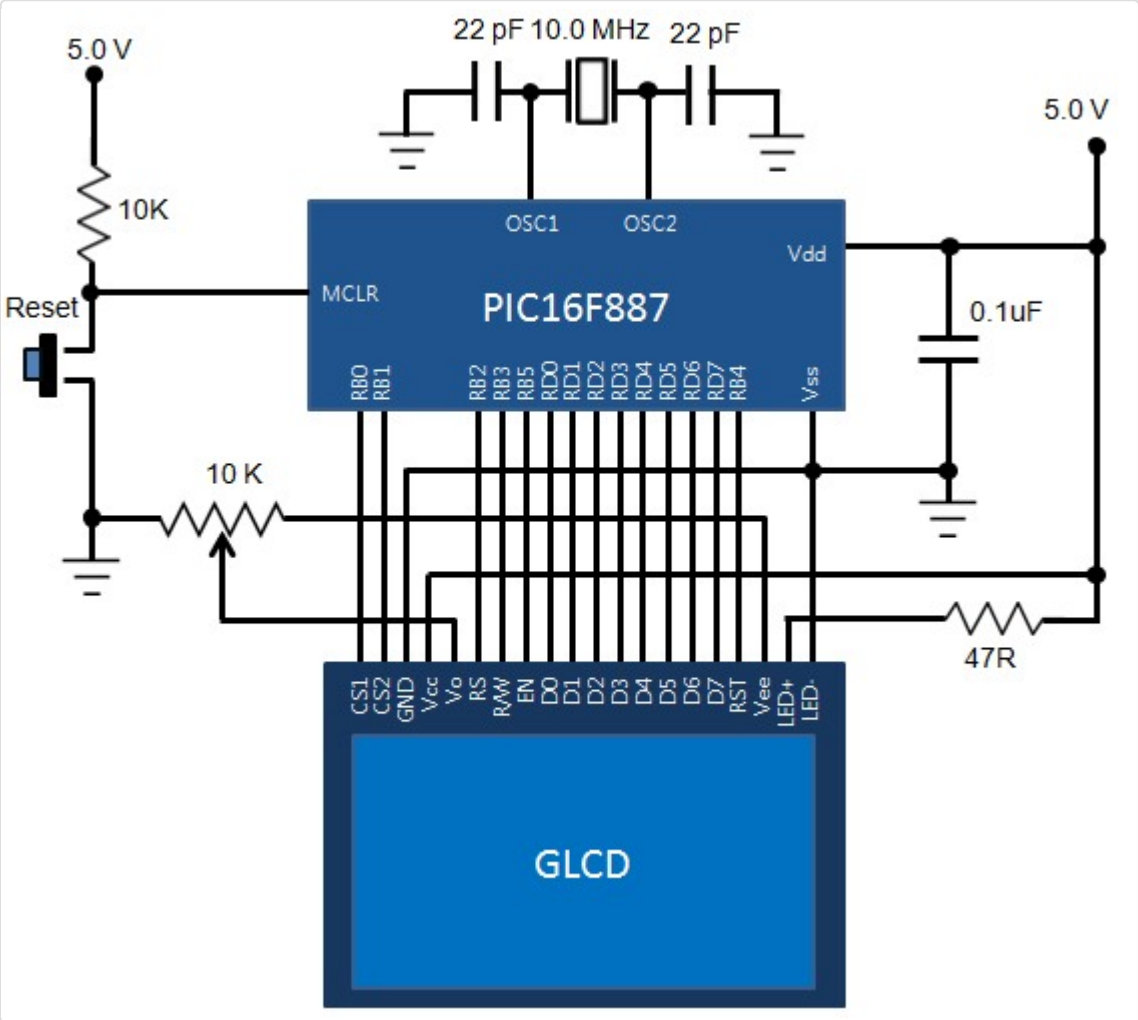
KS0107/KS0108 는 문자 생성기를 가지고 있지 않기 때문에 문자생성기는 반드시 MCU의 펌웨어에 구현되어야 합니다. LCD 컨트롤러는 아래의 테이블과 같은 인스트럭션을 제공합니다. RS(D/I)핀은 데이터를 읽고 쓸경우만 하이상태로 만들어 놓고 명령을 전송할때는 LOW로 만들어 놓아야 하는 것을 주의 하십시오.

Instruction	RS	R/W	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0	Function
Display on/off	L	L	L	L	H	H	H	H	H	L/H	Controls the display on or off. Internal status and display RAM data is not affected. L:OFF, H:ON
Set address (Y address)	L	L	L	H	Y address (0-63)						Sets the Y address in the Y address counter.
Set page (X address)	L	L	H	L	H	H	H	Page (0-7)			Sets the X address at the X address register.
Display Start line (Z address)	L	L	H	H	Display start line (0-63)						Indicates the display data RAM displayed at the top of the screen.
Status read	L	H	Bus y	L	On/ Off	Reset	L	L	L	L	Read status. BUSY L: Ready H: In operation ON/OFF L: Display ON H: Display OFF RESET L: Normal H: Reset
Write display data	H	L	Write data								Writes data (DB0: 7) into display data RAM. After writing instruction, Y address is increased by 1 automatically.
Read display data	H	H	Read data								Reads data (DB0: 7) from display data RAM to the data bus.

디스플레이 제어 명령

회로도

실험을 위한 회로도는 아래와 같습니다. PIC16F887 MCU가 윈스타 WDG0151-TMI GLCD를 동작시키기 위하여 사용되었으며, 데이터 핀은 PORTD에 연결되었습니다. 다른 제어 신호는 PORTB핀들에 연결되었습니다.



GLCD를 연결하기 위한 회로도

소프트웨어

테스트 프로그램은 mikroC 컴파일러로 작성되었습니다. 컴파일러에서 작업을 편하게 하기 위해서 GLCD 라이브러리를 제공하지만 이러한 라이브러리를 사용하지 않고 PIC16F887에서 GLCD로 데이터를 전송하는 코드를 작성하여 보겠습니다. 그 후에 mikroC 컴파일러의 GLCD 라이브러리를 사용하여 좀더 복잡한 연산을 수행하고 살펴보도록 하겠습니다. 아래의 코드는 GLCD 화면에 11개의 점선을 출력하는 프로그램입니다. 점선간 간격은 6줄입니다. 아래의 코드에서 사용한 서브루틴에 대한 간략한 설명입니다.

GLCD_ON() : 디스플레이를 켠다. 컨트롤러들에 3Fh 명령을 전송함으로써 디스플레이를 키게 되고 명령을 보낼때는 CS1, CS2는 반드시 로우 상태로 만들어 주어야 한다. 비슷하게 RS핀도 로우 상태로 만들어 전송된 바이트가 명령어 임을 알려준다.

Set_Start_Line() : 이 함수는 화면 상단에 디스플레이되는 라인 넘버를 변경한다. 0에서 63까지의 숫자를 셋팅 가능하며 디스플레이의 RAM데이터에는 영향을 미치지 않습니다.

GOTO_COL() : 0에서 127 열중 하나의 열로 커서를 옮깁니다.

GOTO_ROW() : 특정 행이나 페이지(0~7)로 커서를 옮김니다.

GOTO_XY() : 커서를 특정 위치로 옮깁니다.

GLCD_Write() : 현재 위치에 데이터 바이트를 씁니다.

GLCD_Read() : 현재 위치에서 한바이트를 읽어 리턴합니다. 이 서브루틴의 코드를 보면 두개의 읽기 연산이 관련되어 있는 것을 볼 수 있는데. 첫번째 읽기연산은 디스플레이 램에서 데이터를 읽어 KS0108B의 출력 레지스터로 옮겨오는 것이고, 두번째 읽기 연산은 MCU가 실제 데이터를 읽어오는 연산입니다.

GLCD_Clrln() : 특정 행(0-7)을 클리어 합니다.

GLCD_CLR() : 화면(총 8개의 페이지)를 클리어 합니다.

Draw_Point() : 특정 위치에 점을 찍습니다.

```
/*
 * Project name: Testing GLCD with PIC16F887
   Embedded Lab 2011.

 * Description:
    This routine demonstrates how to initialize a KS0108 based GLCD and
    activate the pixels on the display. A sub-routine is written to draw a point on
    the GLCD at a given coordinates, and is used to draw dotted lines.
     * Test configuration:
    MCU:            PIC16F887
    Dev.Board:      UNI-DS6
    Oscillator:     HS, 10.0000 MHz
    Ext. Modules:   GLCD 128x64, KS108/107 controller

*/

// Glcd module connections
#define GLCD_Data   PORTD
#define GLCD_Dir    TRISD

sbit GLCD_CS1 at RB0_bit;
sbit GLCD_CS2 at RB1_bit;
sbit GLCD_RS  at RB2_bit;
sbit GLCD_RW  at RB3_bit;
sbit GLCD_RST  at RB4_bit;
sbit GLCD_EN at RB5_bit;
```

```
sbit GLCD_CS1_Direction at TRISB0_bit;
sbit GLCD_CS2_Direction at TRISB1_bit;
sbit GLCD_RS_Direction  at TRISB2_bit;
sbit GLCD_RW_Direction  at TRISB3_bit;
sbit GLCD_RST_Direction  at TRISB4_bit;
sbit GLCD_EN_Direction at TRISB5_bit;
// End Glcd module connections

void Enable_Pulse()
{
  GLCD_EN = 1; //EN high
  delay_us(5);
  GLCD_EN = 0; //EN low
  delay_us(5);
}

void GLCD_ON()
{
   //Activate both chips
   GLCD_CS1 = 0;
   GLCD_CS2 = 0;
   GLCD_RS = 0;          //RS low for command byte
   GLCD_RW = 0;          //RW low for write
   GLCD_Data = 0x3F;     //ON command
   Enable_Pulse();
}

void Set_Start_Line(unsigned short line)
{
   GLCD_RS = 0;            //RS low for command
   GLCD_RW = 0;            //RW low for write
   //Activate both chips
   GLCD_CS1 = 0;
   GLCD_CS2 = 0;
   GLCD_Data = 0xC0 | line;     //Set Start Line command
   Enable_Pulse();
}

void GOTO_COL(unsigned int x)
{
  unsigned short Col_Data;
  GLCD_RS = 0;
  GLCD_RW = 0;
  if(x&lt;64)            //Left Half
  {
    GLCD_CS1 = 0;         //select chip 1
    GLCD_CS2 = 1;         //deselect chip 2
    Col_Data = x;         //put column address on data port
  }
  else                //Right Half
  {

    GLCD_CS2 = 0;
    GLCD_CS1 = 1;
```

```
    Col_Data = x-64;   //put column address on data port
  }
  Col_Data = (Col_Data | 0x40 ) &amp; 0x7F;  //Command format
  GLCD_Data = Col_Data;
  Enable_Pulse();
}

void GOTO_ROW(unsigned int y)
{
  unsigned short Row_Data;
  GLCD_RS = 0;
  GLCD_RW = 0;
  Row_Data = (y | 0xB8 ) &amp; 0xBF; //put row address on data port
  GLCD_Data = Row_Data;
  Enable_Pulse();
}

void GOTO_XY(unsigned int x,unsigned int y)
{
   GOTO_COL(x);
   GOTO_ROW(y);
}

void GLCD_Write(unsigned short b)
{
  GLCD_RS = 1;
  GLCD_RW = 0;
  GLCD_Data = b;
  delay_us(1);
  Enable_Pulse();
}

unsigned short GLCD_Read(unsigned short column)
{
   unsigned short read_data = 0; //Read data here
   GLCD_Dir = 0xFF;            //PORTD as Input
   GLCD_RW = 1;                //Read
   GLCD_RS = 1;               //Data
   GLCD_CS1 = (column&gt;63);
   GLCD_CS2 = !GLCD_CS1;        //Disable/Enable CS2
   delay_us(1);
   GLCD_EN = 1;            //Latch RAM data into ouput register
   delay_us(1);

   //Dummy read to fetch data from the display RAM
   GLCD_EN = 0;            //Low Enable
   delay_us(5);
   GLCD_EN = 1;            //latch data from output register to data bus
   delay_us(1);

   read_data = GLCD_Data;    //Input data
   GLCD_EN = 0;             //Low Enable to remove data from the bus
   delay_us(1);
   GLCD_Dir = 0x00;         //PORTD is Output again
   return read_data;
```

```
}

 void GLCD_ClrIn(unsigned short In)
 {
   int i;
   GOTO_XY(0,In);      //At start of line of left side
   GOTO_XY(64,In);     //At start of line of right side
   GLCD_CS1 = 0;
   for(i=0;i&lt;65;i++)
   GLCD_Write(0);
 }

void GLCD_CLR()
{
  unsigned short m;
  for(m=0;m&lt;8;m++){
   GLCD_ClrIn(m);
  }
}

void Draw_Point(unsigned short x,unsigned short y, unsigned short color)
{
   unsigned short Col_Data;;
   GOTO_XY(x,(y/8));
   switch (color)
   {
      case 0:        //Light spot
         Col_Data = ~(1&lt;&lt;(y%8)) &amp; GLCD_Read(x);
      break;
      case 1:        //Dark spot
         Col_Data = (1&lt;&lt;(y%8)) | GLCD_Read(x);
      break;
   }
   GOTO_XY(x,(y/8));
   GLCD_Write(Col_Data);
}

void main() {
 unsigned short u, v;
 ANSEL  = 0;              // Configure AN pins as digital
 ANSELH = 0;
 C1ON_bit = 0;             // Disable comparators
 C2ON_bit = 0;
 TRISD = 0x00;
 TRISB = 0x00;
 PORTB = 0x00;
 PORTD = 0x00;
 GLCD_CS1 = 1;              // De-Activate both chips
 GLCD_CS2 = 1;
 GLCD_RST = 1;
 GLCD_ON();
 GLCD_CLR();
 Set_Start_Line(0);
 do {
```

```
    for(u=0; u&lt;64; u+=6)
     for (v=0; v&lt;128; v+=2)
     Draw_Point(v, u, 1);
     delay_ms(1000);
     GLCD_CLR();
     delay_ms(1000);

 } while(1);
}
```

MCU가 프로그램을 실행하면 아래와 같이 11개의 점선을 볼 수 있습니다.

점선을 찍은 화면

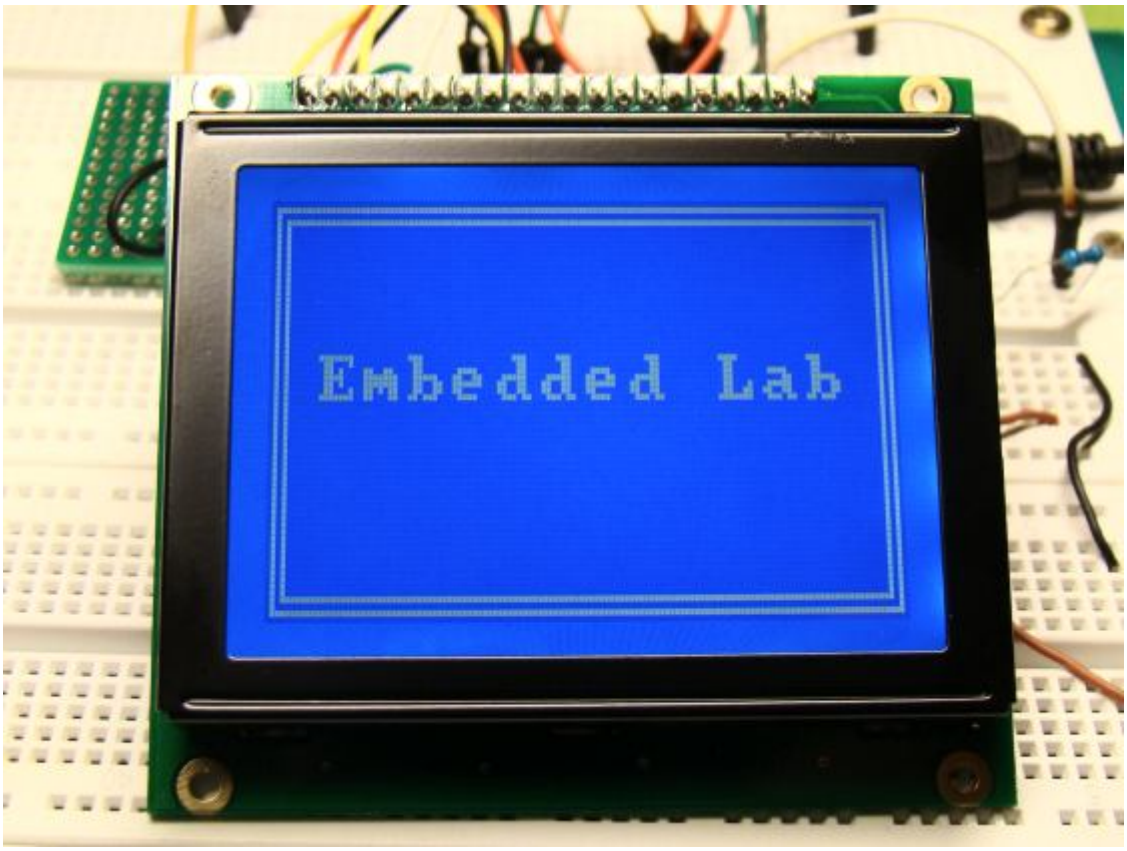
가치창조기술 | www.vctec.co.kr

[PIC 마이컴 실험하기] 20-2. KS0108기반의 그래픽 LCD 인터페이싱 하기

마이컴 실험실

2013/01/23 10:41 http://blog.naver.com/ubicomputing/150157349987

앞선 게시물에서는 윈스타 WDG0151-TMI GLCD 모듈과 PIC16F887 사이의 인터페이싱과 테스트 프로그램을 살펴보았습니다. 살펴본데로 단순히 점선만 표시하는 일에도 많은 노력이 필요하다는 것을 확인하였습니다. 이번 게시물에는 mikroC의 내장 GLCD 라이브러리를 이용하여 GLCD를 쉽게 제어 하는 방법에 대해 이야기 하겠습니다.



mikroC의 GLCD 라이브러리를 이용

mikroC의 GLCD Library

mikroC 컴파일러는 삼성 KS0108/KS0107 컨트롤러 기반의 128x64 픽셀 GLCD를 위한 라이브러리를 제공합니다. 라이브러리는 베이직과 어드밴스로 두개의 타입으로 나뉘어 있습니다. 라이브러리를 사용하기 전에 아래와 같은 핀 정의가 필요합니다. 여기에서는 PORTD가 데이타 핀으로 PORTB가 컨트롤 신호용으로 사용되었습니다. (선행 게시물 참조) 아래

와 같이 GLCD핀이 셋팅되었습니다. 주의해야 할 점 하나는 데이타 라인은 여러개의 포트가 아닌 반드시 한개의 포트에만 할당되어야 하는 점입니다.

```
// glcd pinout settings
char GLCD_DataPort at PORTD;
sbit GLCD_CS1 at RB0_bit;
sbit GLCD_CS2 at RB1_bit;
sbit GLCD_RS at RB2_bit;
sbit GLCD_RW at RB3_bit;<strong></strong>
sbit GLCD_EN at RB5_bit;
sbit GLCD_RST at RB4_bit;
sbit GLCD_CS1_Direction at TRISB0_bit;
sbit GLCD_CS2_Direction at TRISB1_bit;
sbit GLCD_RS_Direction at TRISB2_bit;
sbit GLCD_RW_Direction at TRISB3_bit;
sbit GLCD_EN_Direction at TRISB5_bit;
sbit GLCD_RST_Direction at TRISB4_bit;
```

Basic routines:

- *Glcd_Init* : GLCD 모듈을 초기화한다.
- *Glcd_Set_Side*: GLCD의 좌측 우측을 선택한다. 예를 들면, Glcd_Set_Side(0) 혹은 Glcd_Set_Side(62)는 GLCD의 좌측을 선택하고 Glcd_Set_Side(67)과 같은 64-127의 값은 GLCD의 오른쪽을 선택합니다.
- *Glcd_Set_X* : 선택된 GLCD의 좌우측 안에서 왼쪽을 기준으로 x축 위치를 설정합니다.
- *Glcd_Set_Page* : GLCD의 페이지(0-7)를 선택합니다.
- *Glcd_Read_Data* : 현재 위치의 GLCD 메모리에서 한바이트를 읽고 다음 위치로 옮직입니다.
- *Glcd_Write_Data* : 현재 위치의 GLCD 메모리에서 한바이트를 쓰고 다음 위치로 옮직입니다.

Advanced routines:

- *Glcd_Fill* : 바이트 패턴을 가지고 GLCD 디스플레이의 램을 채웁니다. 만약 바이트 패턴이 0이라면 디스플레이를 클리어 합니다. 0xFF라면 GLCD 전체를 1로 채웁니다.
- *Glcd_Dot* : 주어진 좌표에 정해진 색으로 GLCD 상에 점을 그립니다. Glcd_Dot(x,y, color)의 형태로 사용이되어지며 x는 0-127, y는 0~63, color는 0~2가 사용됩니다. color는 점의 상태를 결정하는 것으로 0은 점을 지우고, 1은 점을 그리며, 2는 점의 상태를 현재와 반대로 변환합니다.
- *Glcd_Line* : 두개의 좌표를 연결하여 하나의 라인을 그립니다. color는 0-2가 쓰일수 있습니다.
- *Glcd_V_Line* : 같은 x좌표에 있는 두개의 점을 연결하여 수직으로 선을 그립니다.
- *Glcd_H_Line* : 같은 y좌표에 있는 두개의 점을 연결하여 수평으로 선을 그립니다.
- *Glcd_Rectangle* : 좌측 상단과 우측 하단의 두개의 점을 기준으로 사각형을 그립니다.
- *Glcd_Box* : 좌측 상단과 우측 하단의 두개의 점을 기준으로 사각형을 그립니다. Glcd_Rectangle함수와 다르게 color 파라메터는사각형 안을 채우는 컬러입니다.
- *Glcd_Circle* : 중심 점을 기준으로 원을 그립니다.

- *Glcd_Set_Font*: 먼저 게시물에서 언급했듯이, KS0108 컨트롤러는 내장된 문자생성기가 없습니다. 그래서 MCU의 펌웨어에 폰트 프로그래밍 되어 있어야 합니다. 이런 작업은 각각의 문자에 대한 데이터 값을 입력해야하는 시간을 잡아먹는 작업입니다. 이러한 점때문에 mikroC 컴파일러는 아래의 데모 폰트를 제공합니다.
  - Font_Glcd_System3x5
  - Font_Glcd_System5x7
  - Font_Glcd_5x7
  - Font_Glcd_Character8x7

  이 폰트들은 Glcd_Write_Char함수와 Glcd_Write_Text 함수들에 사용됩니다. 폰트를 정의하는 문법은 아래와 같습니다.

Glcd_Set_Font (const char **activeFont*, unsigned short *aFontWidth*, unsigned short*aFontHeight*, unsigned int *aFontOffs*);

파라메터 설명:

- activeFont: 설정될 폰트. 문자 배열로 포맷되어 있어야 함
- aFontWidth: 폰트 문자의 도트 너비
- aFontHeight: 폰트 문자의 토드 높이
- aFontOffs: mikroC컴파일러의 문자세트와 표준 ASCII 세트 간의 차이를 나타내는 숫자. 라이브러리에 제공되는 데모폰트는 32의 오프셋을 가지고 있습니다. 그래서 Font_Glcd_5x7의 폰트를 사용하고 싶으면 아래와 같이 폰트를 정의 할 수 있습니다.

```
Glcd_Set_Font(Font_Glcd_5x7, 5, 7, 32);
```

- *Glcd_Write_Char* : GLCD의 페이지(0-7)의 x위치(0-127)에 문자를 씁니다.
- *Glcd_Write_Text* : GLCD의 페이지(0-7)의 x위치(0-127)에 문자열을 씁니다.
- *Glcd_Image*: GLCD에 비트맵이미지를 표시합니다. 이미지의 비트맵 배열을 반드시 펌웨어에 프로그래밍 되어 있어야 합니다. 이 부분은 다음에 오는 게시물을 참고 하십시오.

회로도

선행 게시물 참조

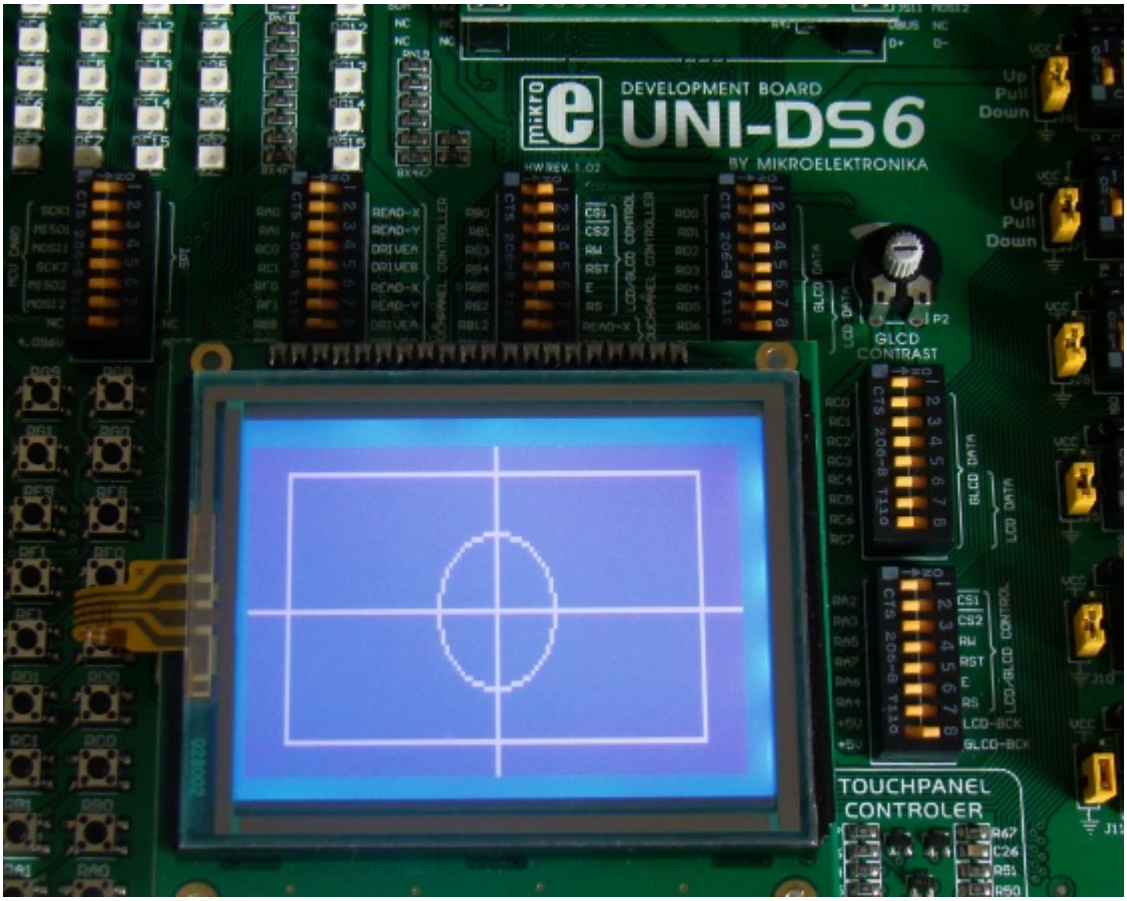
소프트웨어

아래의 프로그램은 mikroC 컴파일러의 GLCD 라이브러리 사용법을 위해 작성되었습니다. 프로그램의 첫부분은 수평 수직 라인, 사각형, 원, 내부 색칠된 박스 등을 그립니다. 두번째 부분은 Font_Glcd_5x7을 사용하여 8개의 페이지(0-7)에 문장을 씁니다. 마지막으로 좀더 큰 폰트(Font_Glcd_Characters_8x7)를 사용하여 GLCD중앙에 "Embedded Lab"이라고 표시합니다.

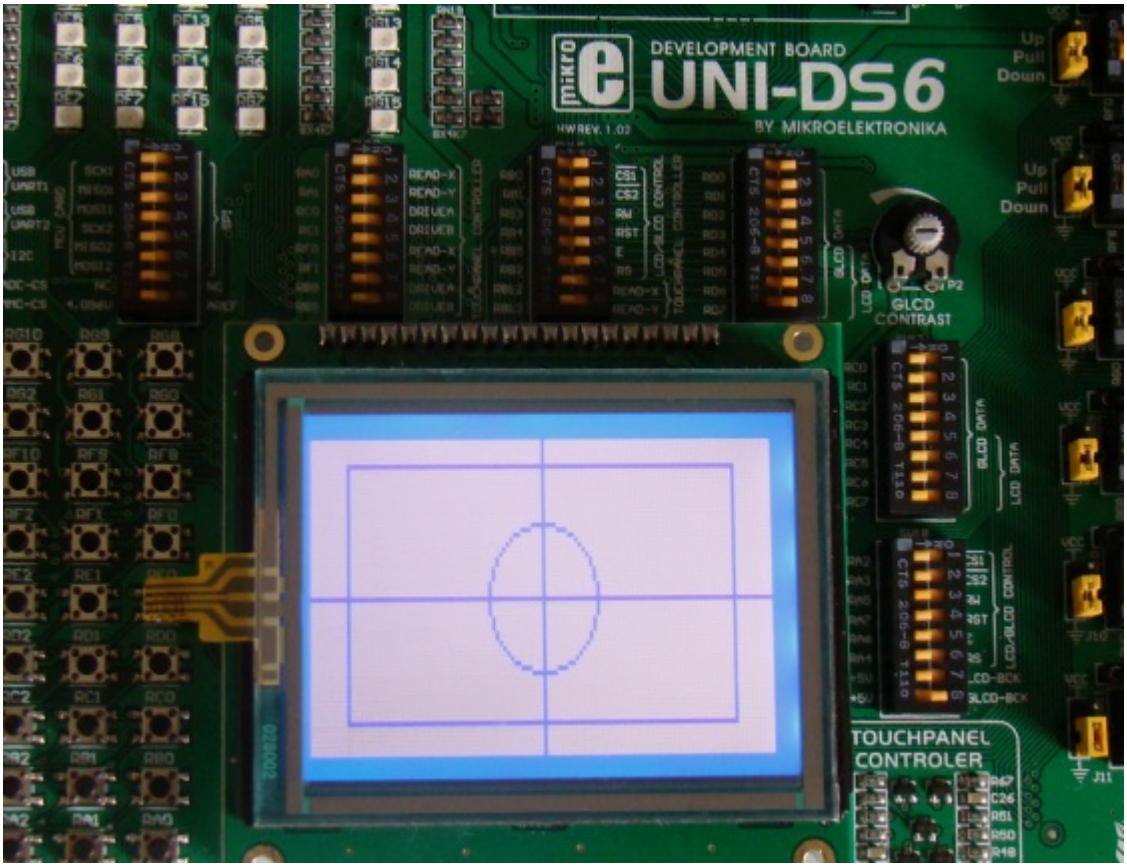
```
/*
 * Project name: Testing GLCD with PIC16F887
   Embedded Jan 02, 2012.

 * Description:
    This routine demonstrates how to use MikroC Pro for PIC GLCD library
    routines for displaying shapes and texts of various font size.
     * Test configuration:
    MCU:            PIC16F887
    Dev.Board:      UNI-DS6
    Oscillator:     HS, 10.0000 MHz
    Ext. Modules:   GLCD 128x64, KS108/107 controller

 */

// Glcd module connections
char GLCD_DataPort at PORTD;

sbit GLCD_CS1 at RB0_bit;
sbit GLCD_CS2 at RB1_bit;
sbit GLCD_RS  at RB2_bit;
sbit GLCD_RW  at RB3_bit;
sbit GLCD_RST  at RB4_bit;
sbit GLCD_EN at RB5_bit;

sbit GLCD_CS1_Direction at TRISB0_bit;
sbit GLCD_CS2_Direction at TRISB1_bit;
sbit GLCD_RS_Direction  at TRISB2_bit;
sbit GLCD_RW_Direction  at TRISB3_bit;
sbit GLCD_RST_Direction  at TRISB4_bit;
sbit GLCD_EN_Direction at TRISB5_bit;
// End Glcd module connections

void Delay2S(){           // 2 seconds delay function
  Delay_ms(2000);
}

void main() {
  ANSEL  = 0;                          // Configure AN pins as digital
  ANSELH = 0;
  C1ON_bit = 0;                        // Disable comparators
  C2ON_bit = 0;
  TRISD = 0x00;
  TRISB = 0x00;
  Glcd_Init();            // Initialize GLCD
  Glcd_Fill(0x00);        // Clear GLCD

  do {
     Glcd_V_Line(0, 63, 64, 1);
     Glcd_H_Line(0, 127, 32, 1);
     Delay2s();
     Glcd_Rectangle(10,5,117,57,1);     // Draw rectangle, color is 1
     Delay2s();
     Glcd_Circle(64,32, 15, 1);
     Delay2s();
```

```
    Glcd_Fill(0xFF);                // Fill Glcd with all 1s
    Delay2s();
    Glcd_V_Line(0, 63, 64, 0);
    Glcd_H_Line(0, 127, 32, 0);
    Delay2s();
    Glcd_Rectangle(10,5,117,57,0);     // Draw rectangle, color is 0 now
    Delay2s();
    Glcd_Circle(64,32, 15, 0);
    Delay2s();
    Glcd_Fill(0x00);                // Clear GLCD
    Delay2s();
    Glcd_Box(10,5,117,57,1);
    Delay2s();
    Glcd_Fill(0x00);                // Clear GLCD
    Delay2s();

    Glcd_Set_Font(Font_Glcd_5x7, 5, 7, 32);
    Glcd_Write_Text("This is Page 0", 10, 0, 1);
    Glcd_Write_Text("This is Page 1", 10, 1, 1);
    Glcd_Write_Text("This is Page 2", 10, 2, 1);
    Glcd_Write_Text("This is Page 3", 10, 3, 1);
    Glcd_Write_Text("This is Page 4", 10, 4, 1);
    Glcd_Write_Text("This is Page 5", 10, 5, 1);
    Glcd_Write_Text("This is Page 6", 10, 6, 1);
    Glcd_Write_Text("This is Page 7", 10, 7, 1);
    Delay2s();
    Glcd_Fill(0x00);        // Clear GLCD
    Glcd_Set_Font(Font_Glcd_Character8x7, 8, 7, 32);
    Glcd_Write_Text("Embedded Lab", 8, 3, 2);   // Write string
    Delay2s();
    Glcd_Fill(0x00);        // Clear GLCD
  } while(1);
}
```

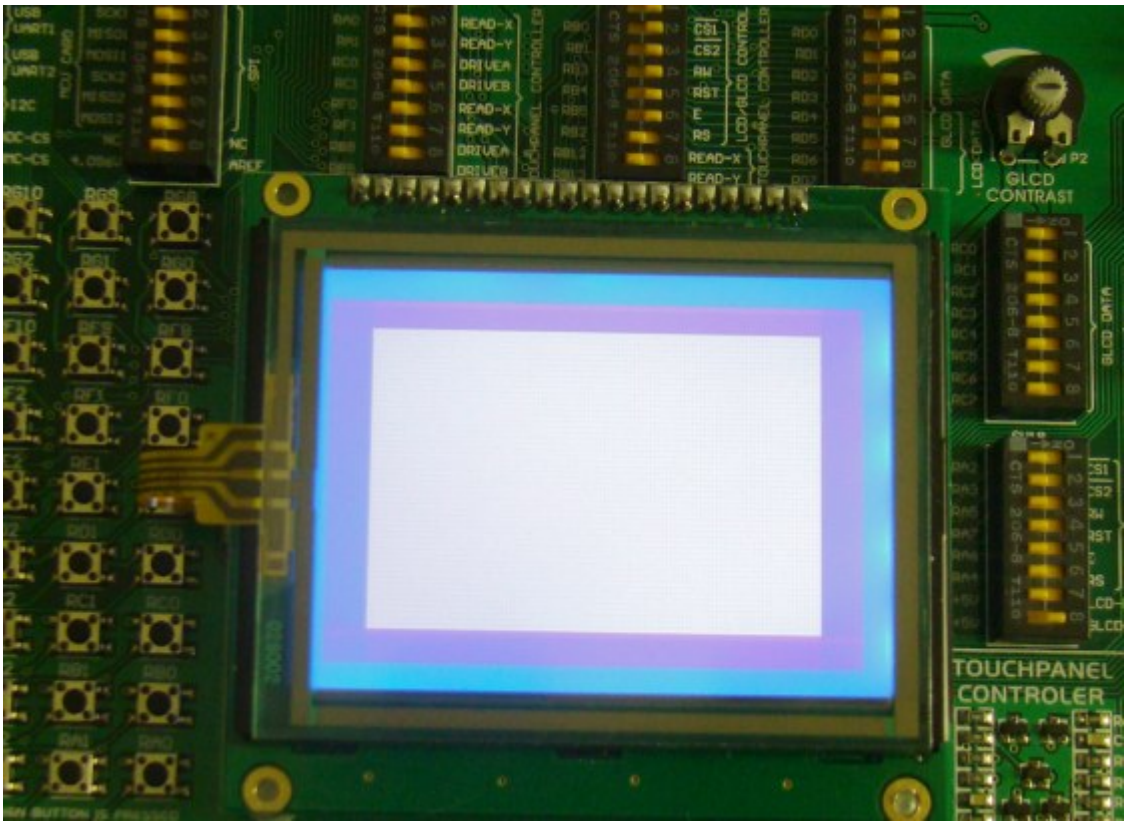
위의 테스트 프로그램을 실행한 화면입니다.


DEVELOPMENT BOARD
UNI-DS6
BY MIKROELEKTRONIKA
GLCD CONTRAST
TOUCHPANEL CONTROLER

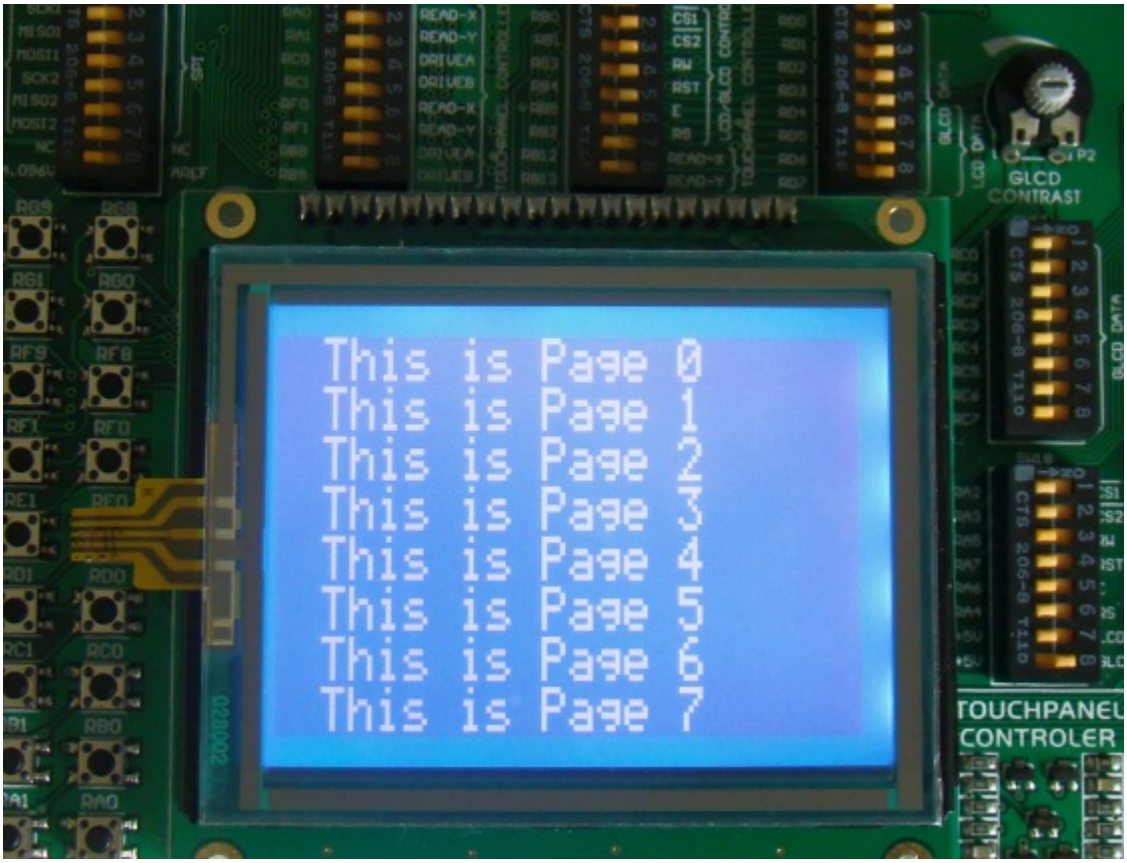
라인, 원, 사각형 그리기



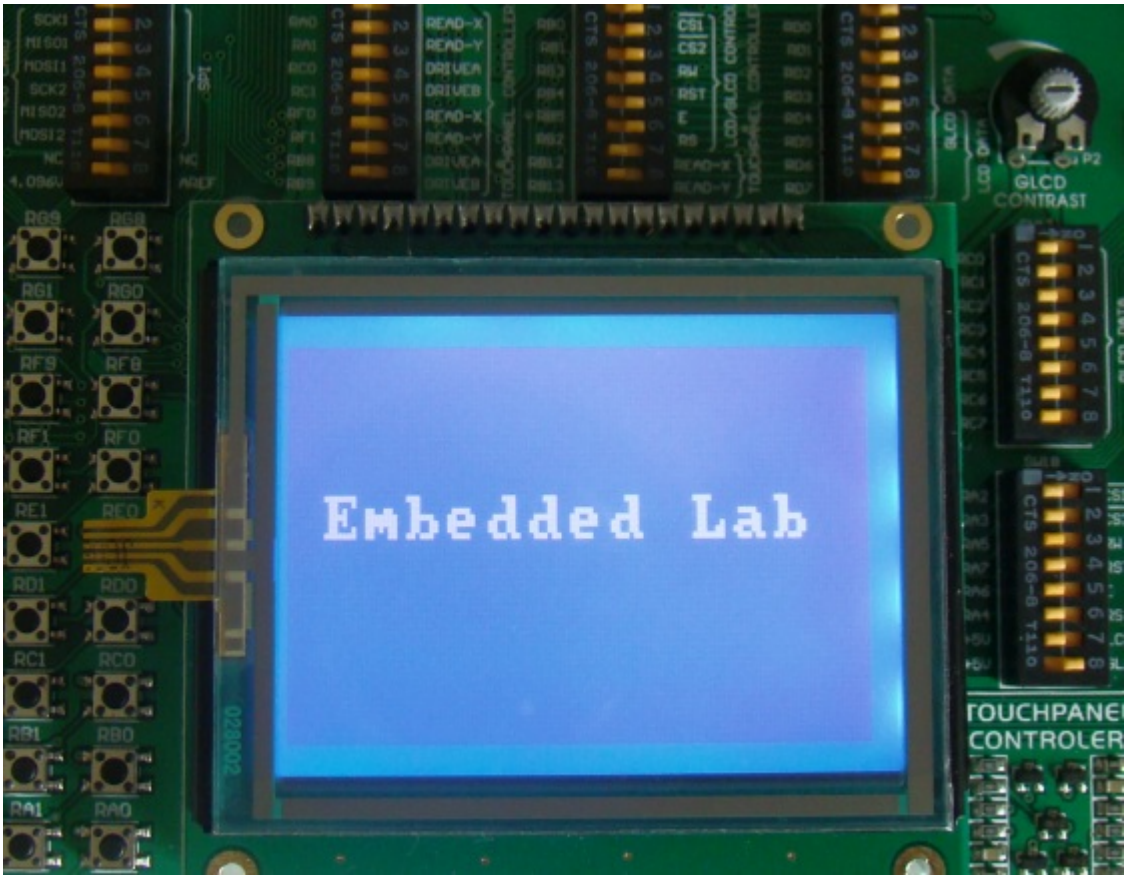
색을 반대로 하여 그린 동일한 그림



색으로 채워진 박스



5x7폰트 크기로 각 페이지에 표시



8x7 폰트 디스플레이

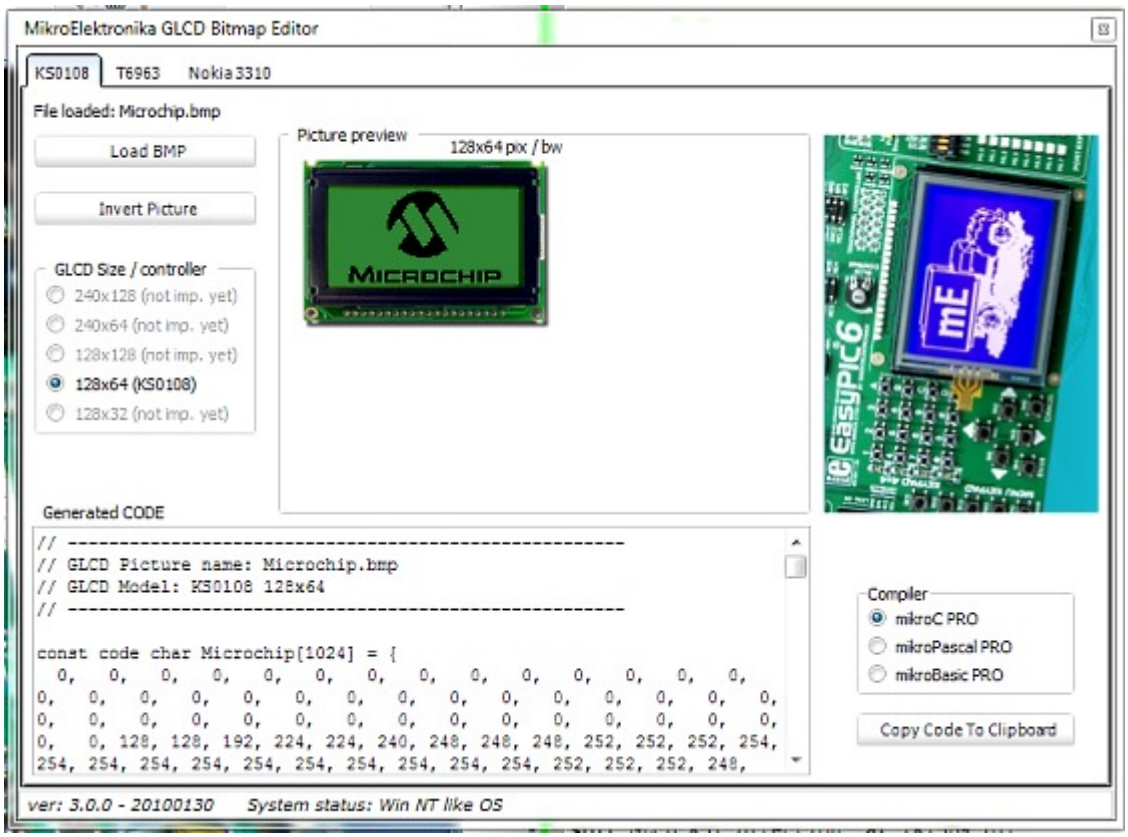
가치창조기술 | www.vctec.co.kr

[PIC 마이컴 실험하기] 20-3. mikroC 컴파일러의 비트맵 GLCD 툴

마이컴 실험실

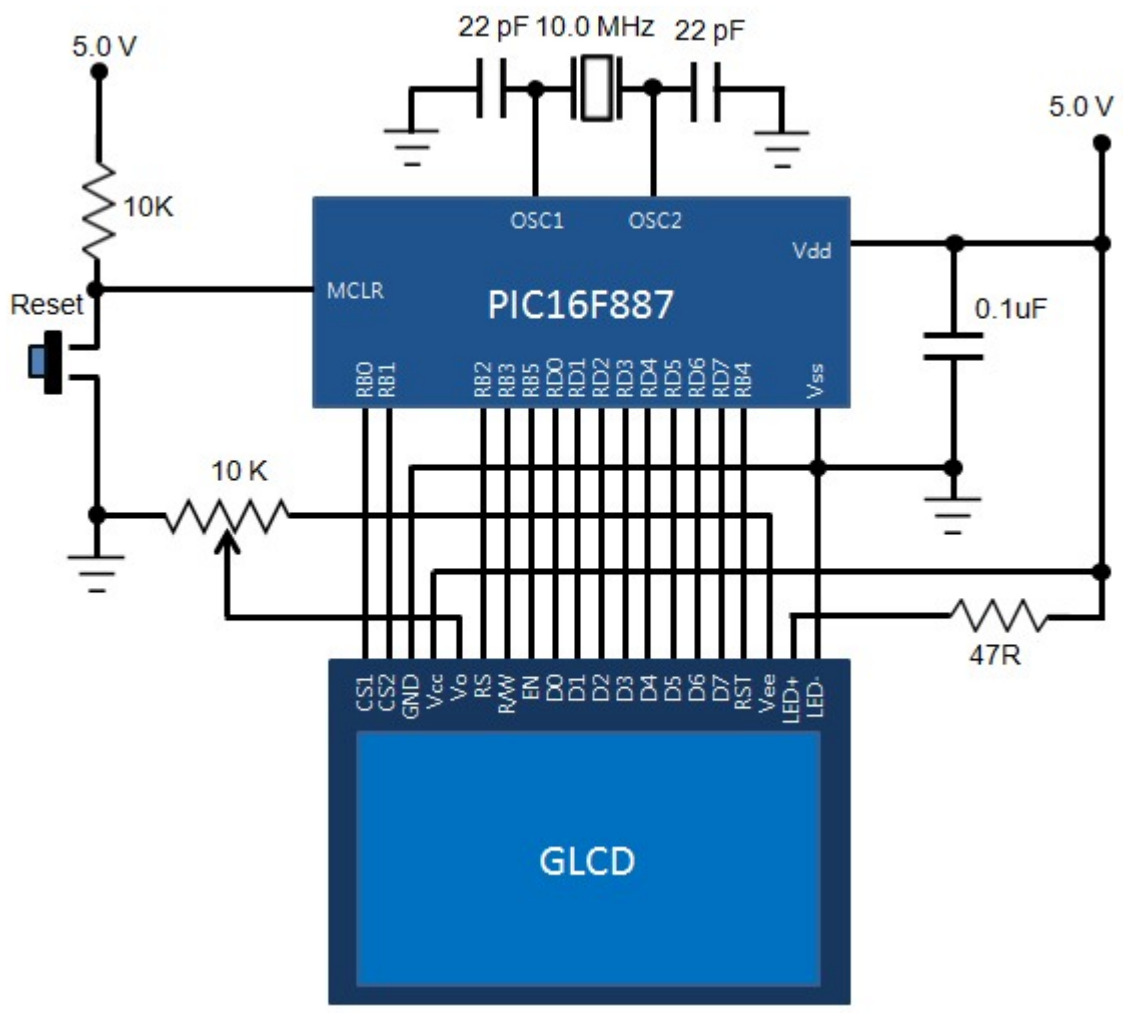
2013/01/23 11:40 http://blog.naver.com/ubicomputing/150157356967

본 게시물은 mikroC 컴파일러의 GLCD bitmap 에디터를 사용하여 비트맵 이미지를 데이터 배열로 변화하여 MCU를 사용하여 GLCD에 비트맵 이미지를 디스플레이 하는 방법에 대하여 설명하였습니다. GLCD 비트맵 에디터는 BMP 이미지와 동일한 코드를 생성하여 MCU 소스코드에 쉽게 삽입할 수 있습니다.



mikroC컴파일러의 GLCD BMP editor

여기서는 NT7107/8 기반(삼성 KS0107/8호환)의 128x64 GLCD를 사용하였습니다. 마이크칩 로고를 데이터 배열로 변환하여 PIC MCU에 의해 동작하는 LCD에 표시하여 보도록 하겠습니다. 회로도는 아래와 같습니다.

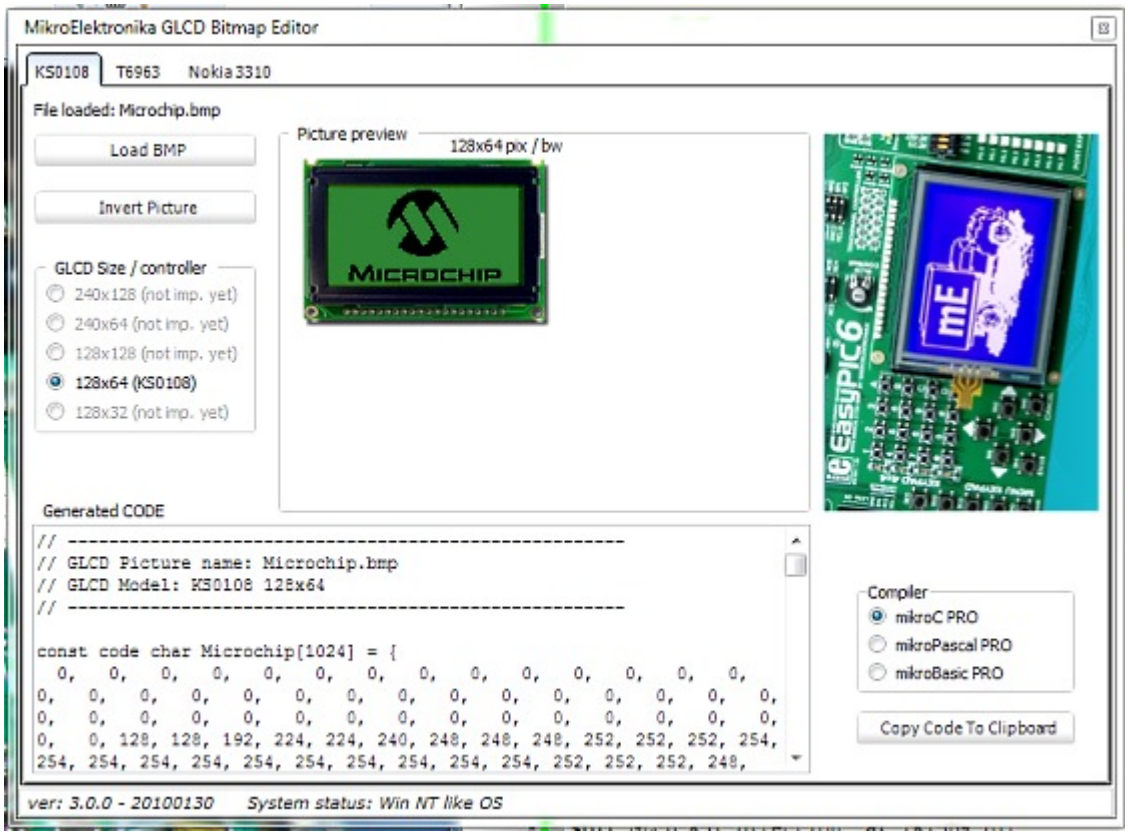


PIC16F887와 GLCD의 연결 회로도

아래의 그림은 128x64 픽셀 마이크로칩 로고입니다. 먼저 단색의 bmp이미지로 변환되어야 하는데 윈도우의 그림판 어플리케이션에서 관련 작업을 할 수 있습니다.

Microchip logo

단색으로 변환한 로고를 mikroC의 Tools 메뉴에서 GLCD bitmap editor 상에서 엽니다. GLCD의 크기와 컨트롤러를 선택합니다. 여기서는 128x64 (KS0108)을 선택하였습니다. 다음으로 Load BMP버튼을 눌러 BMP파일을 선택하여 변환하면 Generated Code 섹션에 코드가 생성된것이 보입니다. 코드를 카피하여 메인 프로그램에 붙여 넣거나 다른 파일로 만들어 메인 프로그램에서 include합니다.



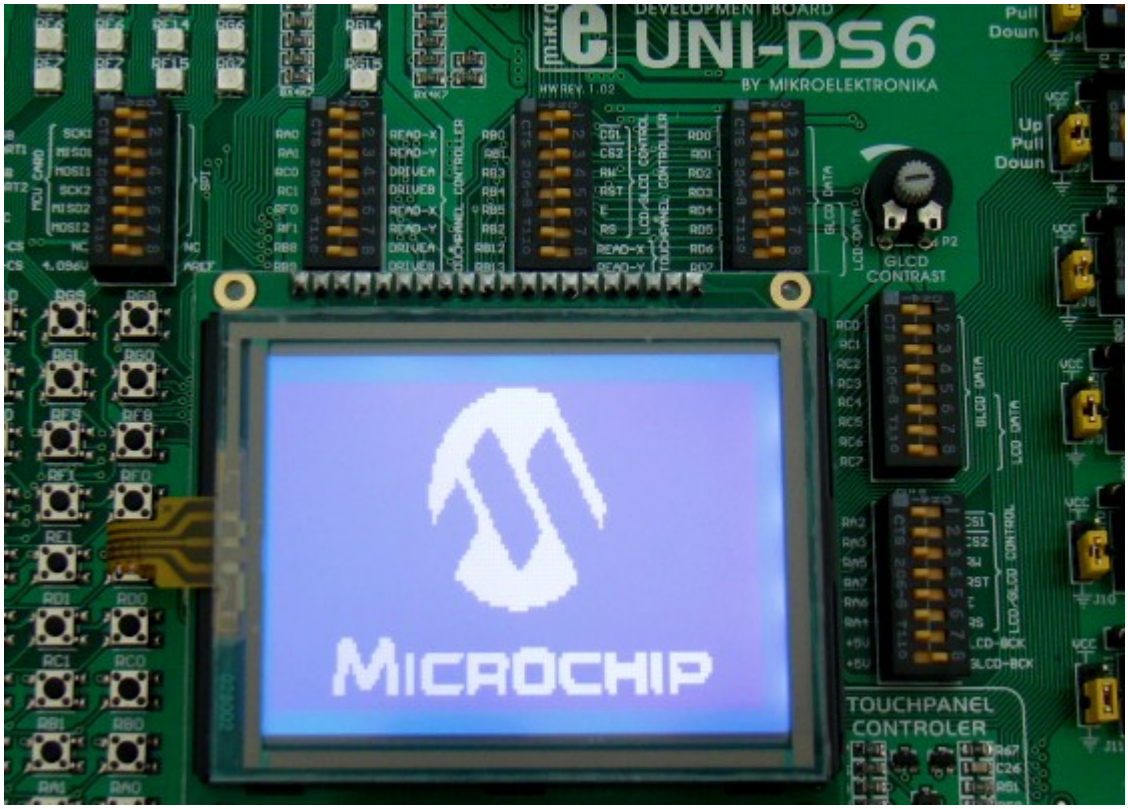
mikroC컴파일러의GLCD BMP editor

생성된 코드는 GLCD의 어떤 픽셀에 이미지를 표시해야 하는지를 표시하는 단순 바이트 배열으로 코드는 아래와 같이 생겼습니다.

```
// ----------
// GLCD Picture name: Microchip.bmp
// GLCD Model: KS0108 128x64
// ----------

const code char Microchip[1024] = {
  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0, 7, 15, 15, 31, 31,
  31, 63, 63, 63, 63, 63, 63, 63, 63, ...
  ... 12, 12, 12, 12, 127, 127, 127,  0,  0, 127, 127, 127,  0,  0, 127, 0
};
```

128x64 크기(128x64 = 8192 픽셀)의 LCD는 1024바이트(1024x8 = 8192 비트)의 배열 크기가 필요합니다. mikroC 컴파일러에서는 내장된 GLCD라이브러리를 이용하여 쉽게 이미지 데이터를 GLCD에 로드할 수 있습니다. Glcd_Image(Micochip) 명령은 로고를 GLCD상에 표시하여 줄 것입니다.

소스코드 다운로드



결과 화면

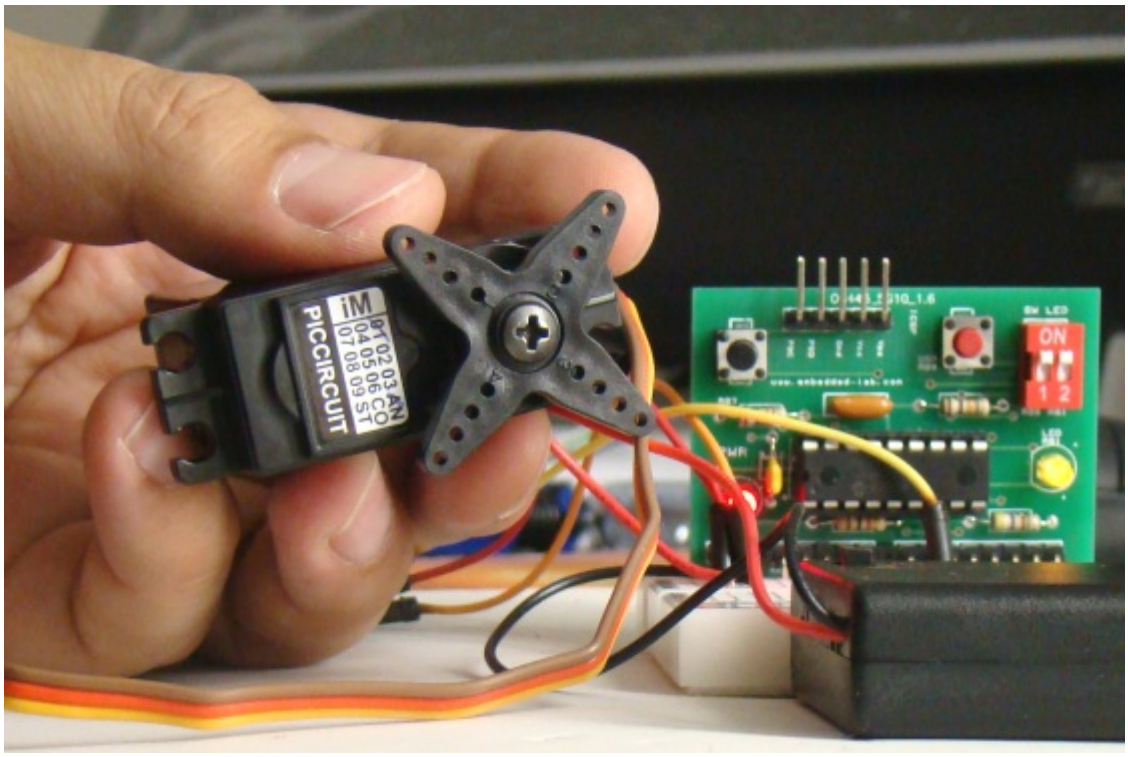
가치창조기술 | www.vctec.co.kr

[PIC 마이컴 실험하기] 21. 서보모터 제어하기

마이컴 실험실

2013/01/24 10:52 http://blog.naver.com/ubicomputing/150157476790

서보 모터는 DC모터의 일종으로 기어, 모터의 방향 및 위치를 제어하기 위한 회로가 장착되어 있는 모터입니다. 서보모터는 정교한 각위치를 제어가 가능하기 때문에 로봇분야나, RC 카 등 모터의 위치를 제어하기 위한 분야에서 사용이 됩니다. 이 게시물에서는 먼저 서보 모터가 어떻게 구성이 되어 있고 어떻게 동작하는지 살펴본 다음 PIC MCU와 연결하는 방법을 살펴보겠습니다.

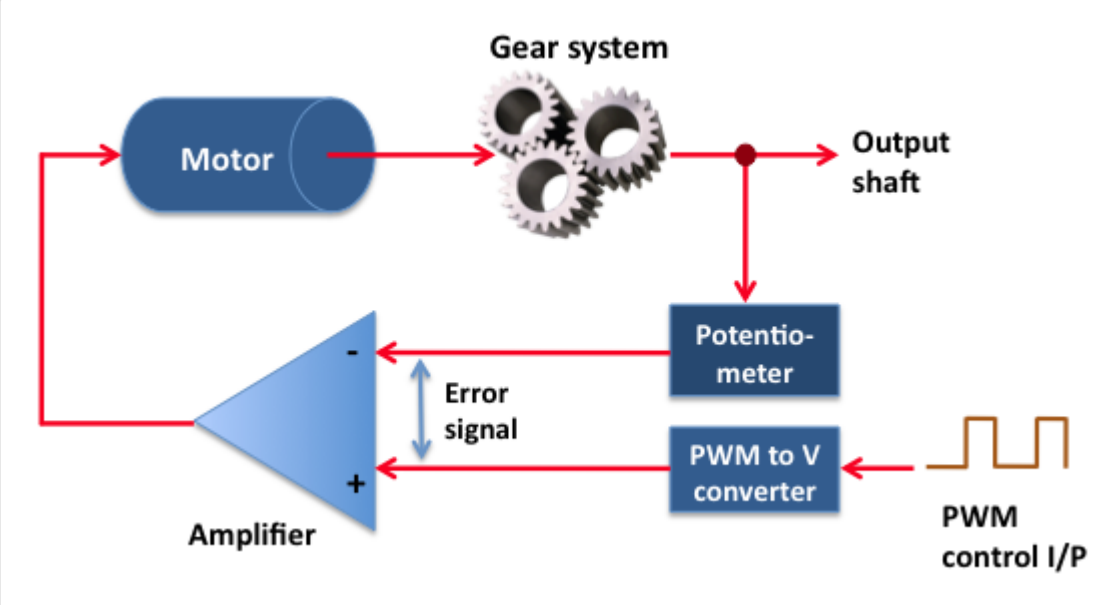


PIC MCU를 이용하여 서보 모터 제어하기

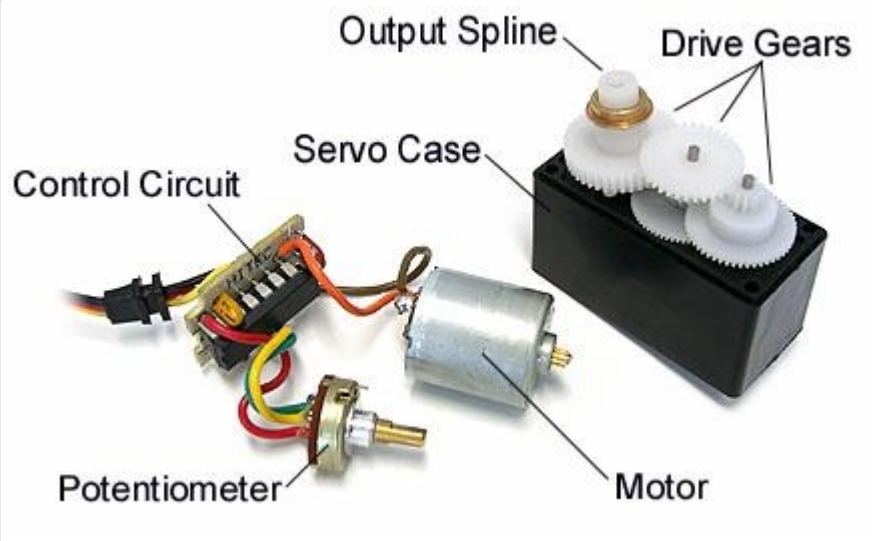
이론

서보모터는 DC모터와 여러 기어를 통해서 모터에 연결되어 있는 출력 샤프트, 샤프트의 위치를 제어하기 위한 전자회로로 구성된 박스입니다. 서보모터를 사용하는 목적은 정교한 각 움직임을 제어하고자 할때 사용됩니다.

서보모터의 움직임 제어를 위해서 출력 샤프트의 위치정보는 트랜스듀서를 사용하여 즉각적으로 제어회로에 공급되는데, 이렇게 하기 위한 가장 간단한 방법은 포텐티오미터를 출력 샤프트나 기어트레인에 붙이는 것입니다. 제어 회로는 포텐티오미터로부터 오는 피드백 신호(샤프트의 현재 위치정보)를 제어입력신호(샤프트가 움직여야 할 위치 정보)와 비교하여 차이(에러신호)가 있다면 DC모터를 움직여 그 차이를 줄여줍니다. 원하는 위치로 모터가 정확하게 움직였을 경우 에러신호는 0 이 됩니다. 아래는 일반적인 서보모터의 기능을 표현한 다이어그램입니다.



서보모터의 동작원리

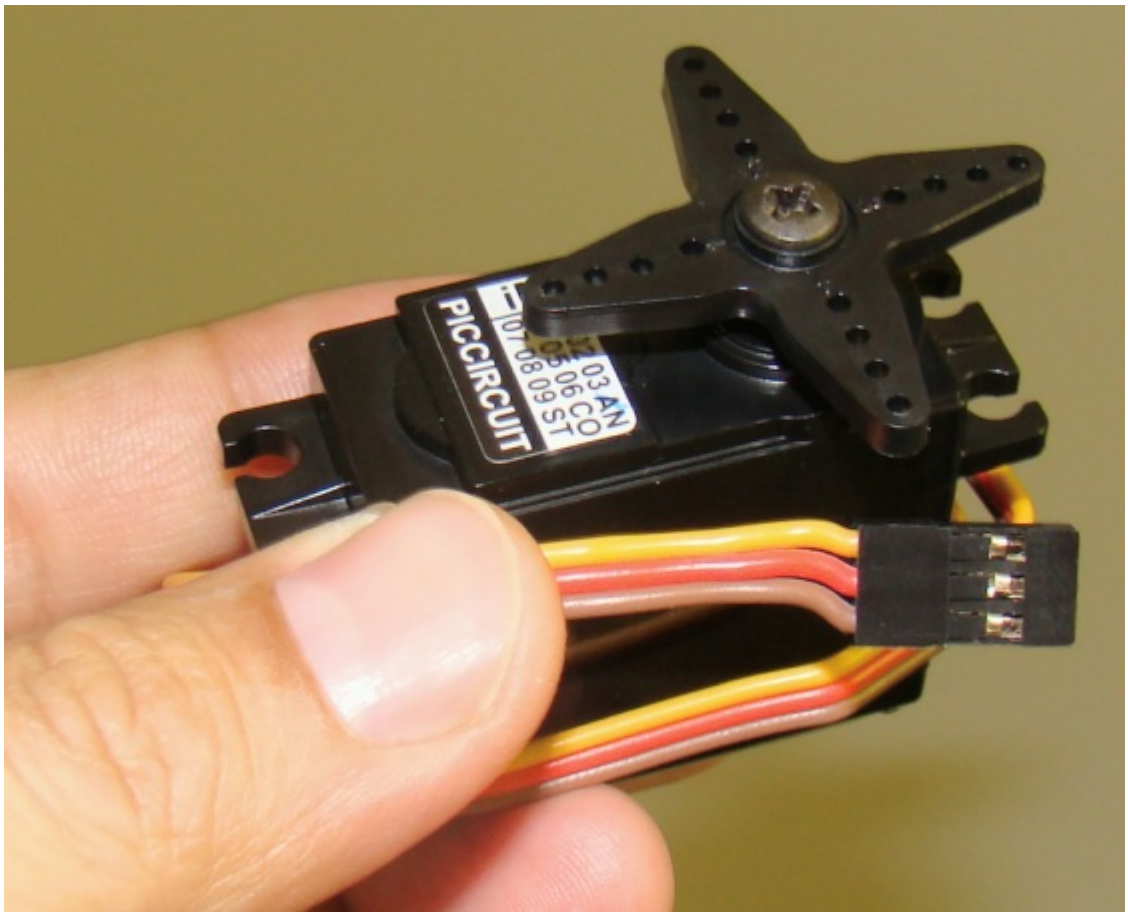


서보모터의 부품

서보모터로의 제어 입력신호는 PWM신호로 일반적으로 50Hz를 사용합니다. 이것은 펄스가 매 20ms마다 반복되어야 한다는 것을 의미합니다. 펄스의 너비는 출력 샤프트의 각위치를 결정합니다. 서보모터 내에 있는 전자회로는 PWM신호를 출력전압신호로 변환하여 줍니다. 이 출력전압신호는 포텐티오미터로부터 받은 피드백전압과 비교됩니다. 만약 두개의 값에 차이가 있다면 차이가 0이 될때까지 적당한 위치로 모터를 움직입니다. 일반적인 펄스너비의 값은 1.0에서 2.0ms 사이입니다. 표준 서보모터에 있어 1.0ms에서 1.5ms의 펄스너비는 시계방향으로 모터를 회전 시키며, 1.5에서 2.0ms는 모터를 반시계방향으로 회전 시킵니다. 1.5ms 펄스너비는 서보모터를 중앙으로 움직이게 합니다. 하지만 이러한 값들은 모터에 따라 다르기 때문에 사용하는 모터의 데이터쉬트를 참조해야합니다.

대부분의 서보모터는 180도를 회전하지만 어떤 것들은 360도 혹은 그 이상을 회전할 수 있습니다. 서보모터는 로봇팔과 같이 움직임을 제어하는 연결부이에 많이 사용됩니다.

서보모터는 세개의 선이 있습니다. 두개는 전원공급용(Vcc와 Gnd)이고 나머지 하나는 제어 신호용입니다. Vcc 라인은 보통 빨간색이고 그라운드는 검정이나 갈색입니다. 제어라인은 하얀색, 노랑색, 혹은 오렌지 색중에 하나로 표시됩니다. 여기서 사용된 서보모터는 빨강, 갈색, 노랑 색이 각각 Vcc, Gnd, 제어신호용으로 사용되었습니다. 5.0v에서 동작하며 180도 회전이 가능합니다.

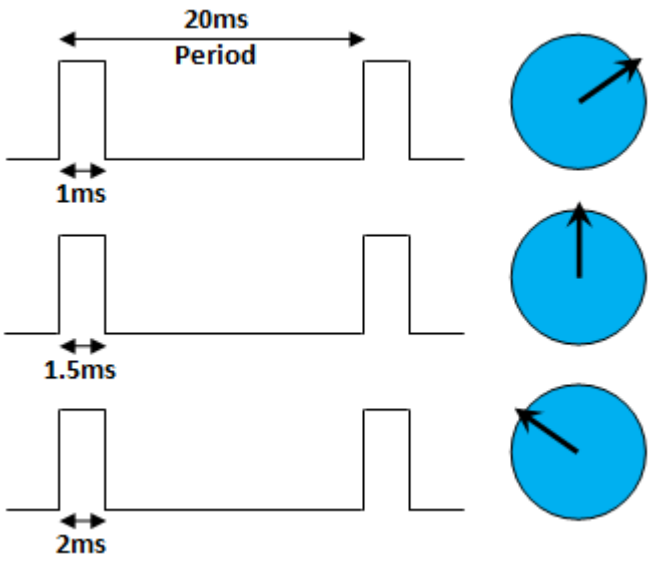


일반적인 서보모터의 생김새

아래 테이블은 사용하는 서보모터의 각위치에 따른 펄스너비 값이 표기되었습니다. 펄스의 반복주기는 50Hz(20ms)이라는 것을 기억하십시오.

Pulse width (ms)	Angular position (° CCW)
0.7	0 (min)
1.1	45
1.5	90
1.9	135
2.3	180 (max)

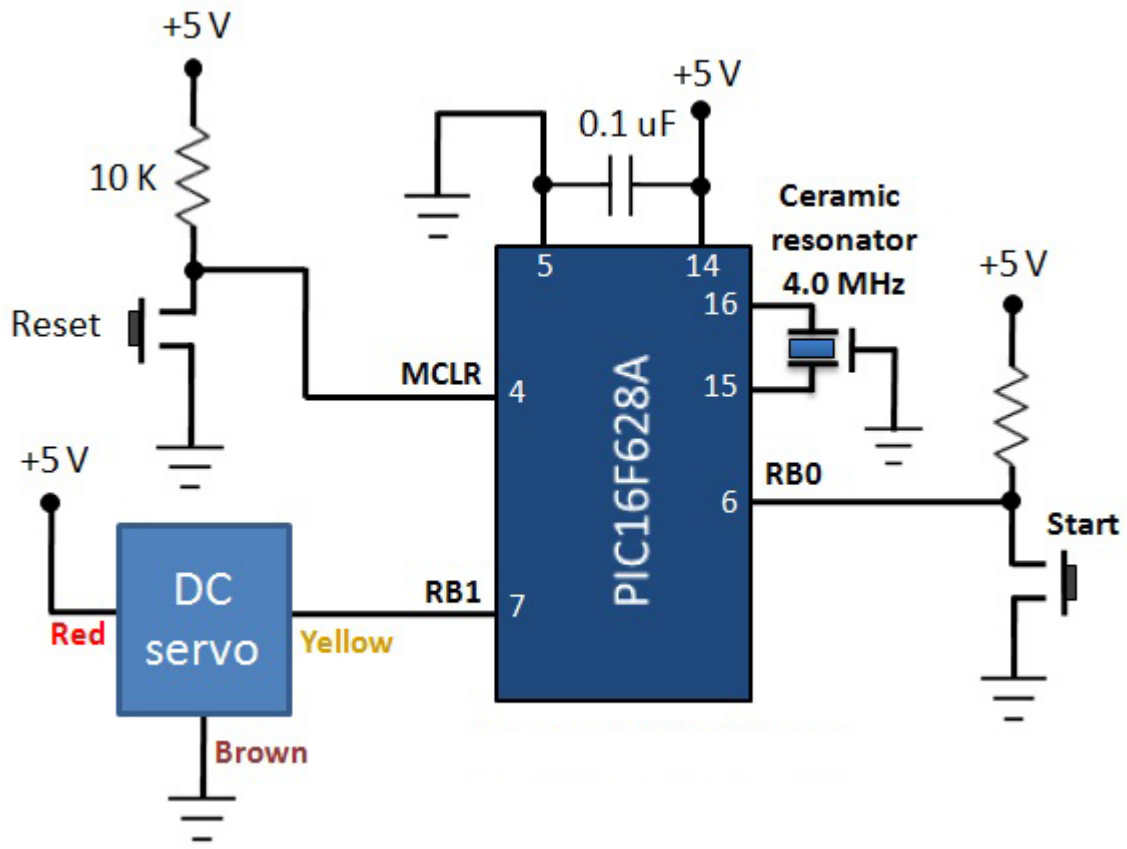
서로 다른 각위치에 대한 서보 타이밍 정보



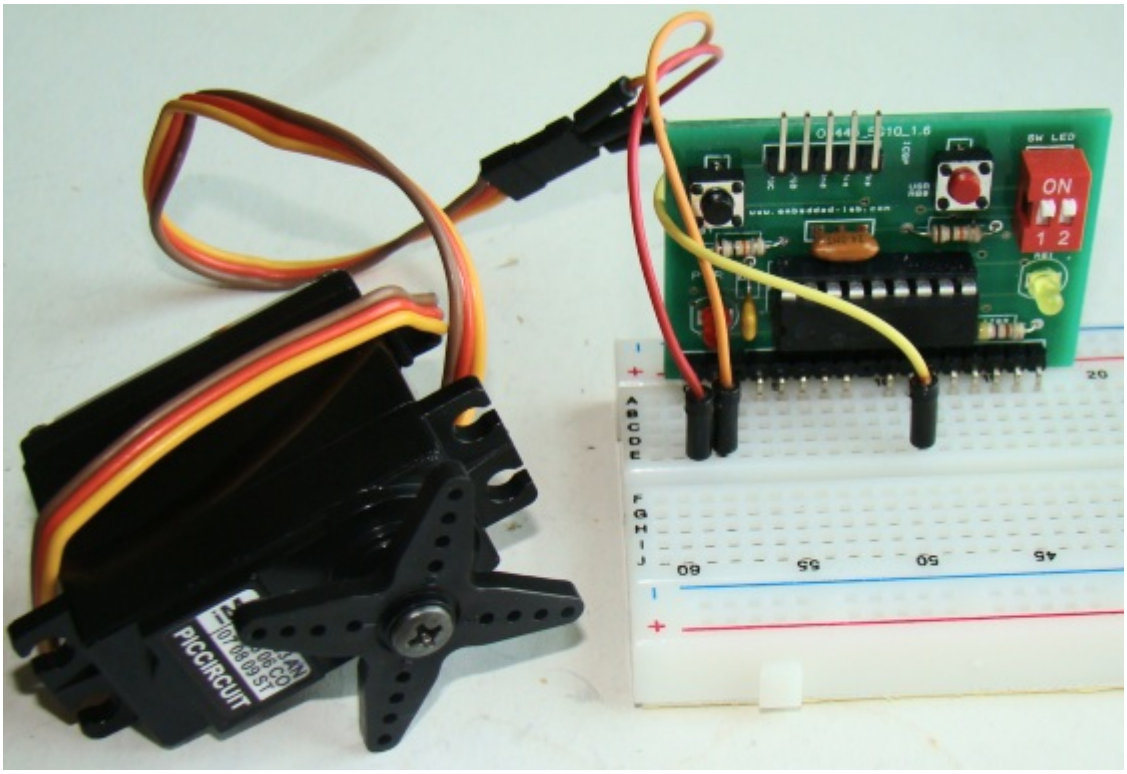
서보arm의 서로다른 각위치

회로

본 실험의 회로도는 아래의 그림과 같습니다. 서보의 제어입력은 PIC16F628A의 RB1핀에 연결되었습니다. 택트 스위치가 RB0핀에 연결되어 서보를 제어하기 위한 입력 신호를 주기 위해 사용되었습니다.



서보모터 제어를 위한 회로도



브레드보드상의 회로셋업

소프트웨어

PIC16F628A의 펌웨어는 mikroC 컴파일러로 작성이 되었습니다. Timer0 모듈은 1:256 프리스케일러가 할당된 타이머로 동작되어 두개의 PWM펄스간 약 20ms의 갭을 만들어 줍니다. 클럭주파수가 4.0Mhz이기 때문에 1μs 머신 싸이클이 되며, Timer0를 사용하는데있어 딜레이를 계산하는 것을 단순화 시켜 줍니다. mikroC 는 Delay_Cyc()이라는 내장 라이브러리 함수를 제공하여 클럭 딜레이를 생성할 수 있게 하여 줍니다. 이 함수는 제어펄스를 0.7에서 2.3ms 사이상으로 변경시키는 데 사용이 됩니다. 회로가 처음 전원이 인가가 되거나 리셋이되면, 0.7ms 펄스너비를 가진 50Hz PWM신호가 RB1핀에 계속 생성이 됩니다. 이 제어 신호는 서보를 시계방향 끝(각위치 0)으로 회전 시키게 만듭니다. RB0핀에 연결된 택트 스위치가 눌리게 되면 펄스의 너비는 0.2ms 증가가 되고 이 것은 약 22.5도 정도 샤프트를 반시계방향으로 돌리게 됩니다. 스위치가 눌릴때마다 펄스 너비는 0.2ms씩 증가가 되므로 샤프트는 다른쪽 끝(각위치 180도)에 도달하게 됩니다. 스위치를 9번 누를때, 펄스 너비는 0.7ms로 리셋되어 모터 샤프트는 각위치가 0이 될때까지 시계방향으로 회전합니다.

```
/*
  Lab 21: Servo motor Control using PIC16F628A
  MCU: PIC16F628A running at 4.0 MHz, MCLR enabled, WDT is OFF,
      Brown out detect disabled
  Written by: Rajendra Bhatt   2012/03/29
  Description: User input switch is connected to RB0 and Servo Control signal
  is generated from RB1 pin.
*/

sbit SW1 at RB0_bit;
sbit Control at RB1_bit;

unsigned short i=7, delay;

void interrupt() {
  delay = i*10;
  Control = 1;
  Delay_Cyc(delay);  // Generates delay equal to (10*delay) clock cycles
  Control = 0;
  TMR0 = 180;        // TMR0 returns to its initial value
  INTCON.T0IF = 0;   // Bit T0IF is cleared so that the interrupt could reoccur
}

void main() {
 CMCON = 0x07;    // Disable Comparators
 TRISB = 0b00000001;
 PORTB = 0;
 OPTION_REG = 0x07; // Prescaler (1:256) is assigned to the timer TMR0
 TMR0 = 180;        // Timer T0 counts from 180 to 255 to create ~20 ms period
 INTCON = 0xA0;     // Enable interrupt TMR0 and Global Interrupts
 do{
  if(!SW1){        // Change pulse width when Switch is pressed
   Delay_ms(300);
   i = i+2;
   if(i&gt;23) i=7;
  }
```

```
}while(1);
}
```

소스코드 다운로드

가치창조기술 | www.vctec.co.kr