차 례

로봇 센서 인터페이스 개발 학습모듈의 개요

학습모듈의 목표

로봇이 계획된 동작을 수행하고 주변 환경을 인식하기 위한 다양한 센서 인터페이스 프로그램을 개발할 수 있다.

선수학습

로봇공학, 전자기학, 회로이론, 센서공학, 신호처리

학습모듈의 내용 체계

학습	학습 내용	NCS 능력단위 요소	
		코드번호	요소 명칭
1. 로봇 센서 사양 분석하기	1-1. 로봇 센서 사양 분석	1903080303_14v1.1	로봇 센서 사양 분석하기
2. 로봇 센서 프로토콜 설계하기	2-1. 로봇 센서 프로토콜 설계	1903080303_14v1.2	로봇 센서 프로토콜 설계하기
3. 로봇 센서 드라이버 구현하기	3-1. 로봇 센서 드라이버 구현	1903080303_14v1.3	로봇 센서 드라이버 구현하기

핵심 용어

폴링, 인터럽트, RS-232, GPS, 조도 센서, 온습도 센서, 스택, 링버퍼, 스타터키트, 개발환경 구축, 비트 연산자, AVR studio, JTAG

학습 1	로봇 센서 사양 분석하기
학습 2	로봇 센서 프로토콜 설계하기
학습 3	로봇 센서 드라이버 구현하기

1-1. 로봇 센서 사양 분석

학습 목표

- 로봇 구동을 위해 필요한 센서를 파악하고 사용 목적과 제원을 도출할 수 있다.
- 선정된 로봇 센서의 작동 원리를 정리할 수 있다.
- 로봇 센서 인터페이스를 위한 하드웨어의 제원을 분석하고 인터페이스 프로그램 작성에 필요한 사항들을 정리할 수 있다.
- 로봇 센서 관련 기술정보 및 관련 데이터시트를 해석할 수 있다.

필요 지식 /

① 센서 선정

로봇에서 사용되는 센서는 여러 가지 범주로 나눌 수 있다. 다음은 종류별 센서들의 분류를 나열한 것이다.

1. 사람의 오감에 따라 대응하는 센서 분류
   (1) 시각: 포토 다이오드, 컬러 센서, 카메라
   (2) 청각: 마이크
   (3) 후각: 가스 센서
   (4) 미각: 바이오 센서, 미각 센서
   (5) 촉각: 온도 센서, 터치 센서, 압력 센서
2. 계측 물리량의 종류에 따른 센서 분류
   (1) 기계: 위치, 형상, 속도, 힘, 토크, 압력, 진동, 질량 센서
   (2) 전기: 전압, 전류, 전하, 전기 전도도 센서
   (3) 자기: 자기장, 자속, 투자율 센서
   (4) 온도: 온도, 열전달, 열전도도, 비열 센서
   (5) 기타: 음향, 근접도, 화학 조성, 광전기, 방사, 레이저, 광학, 촉각, 음성, 시각 센서

3. 출력 방식에 따른 센서 분류

(1) 전압 출력형: 측정값을 전압으로 출력함

(2) 전류 출력형: 측정값을 전류로 출력함. 저항을 연결하여 전압으로 변환함

(3) 디지털 신호형: 측정값을 디지털 신호로 출력함

(4) 아날로그 신호형: 측정값을 아날로그 신호로 출력함. ADC를 통해 디지털 신호로 변환함

(5) 통신 신호형: 정해진 방식의 디지털 통신을 통하여 일련의 신호가 전송됨. Field bus(DeviceNet, CAN, ProfiBus, Ethernet 등)는 ISO 15735 국제 표준으로 정해져 있음. 변환기를 통하여 USART(RS-232)로 MCU에 전달되거나, 전용 통신칩으로 회로를 따로 구성하여 MCU에 연결됨

4. 응답 방식에 따른 센서 분류

(1) 일반형: 센서의 측정값이 항상 준비되어 있음

(2) 요구-응답형: MCU가 요청 시에 측정값을 전송함

(3) 주기적 응답형: 초기 설정 이후에는 MCU의 요청이 없어도 주기적으로 측정값을 전송함

(4) 계속 응답형(continuous stream): MCU의 요청이 없어도 센서의 측정값이 준비되면 센서가 계속하여 측정값을 전송함

2 로봇 센서 신호의 처리

1. 일반형 디지털 센서 신호 확인

센서의 측정값이 항상 준비되어 있는 경우, 로봇 MCU는 그것을 단지 필요한 순간에 읽어오기만 하면 된다. MCU의 디지털 입력 핀에 연결되어 있는 ON/OFF 센서들의 경우, 예를 들어 버튼 스위치, 근접 스위치, 포토 다이오드 등의 신호는 MCU가 필요한 때에 신호를 바로 확인할 수 있다.

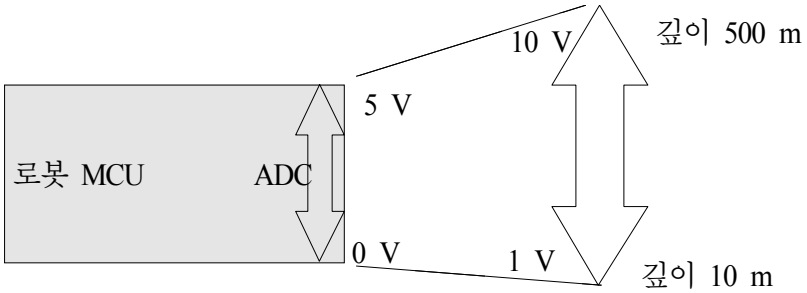
2. 일반형 아날로그 센서 신호의 오차 분석

로봇 MCU는 ADC 포트를 통하여 아날로그 센서의 측정값을 확인할 수 있다. MCU가 아날로그 입력 핀에 연결된 센서 측정값을 읽어오기 위하여, 먼저 MCU가 ADC로 하여금 아날로그 신호를 디지털 신호로 변환하는 동작을 개시시키고, 그 결과가 안정되는 시간만큼 기다린 후, 변환 완료된 신호를 읽어오는 단계가 필요하다.

혹은 ADC가 끊임없이 계속하여 신호 변환을 하도록 설정하고, 특정 메모리에 저장된 값을 필요할 때에 읽어올 수도 있는데, 이 경우는 측정값이 필요하지 않은 때에도 MCU의 리소스와 전력을 계속 사용하게 되므로 배터리 등 제한된 전력만을 사용하는 경우에는 피해야 한다.

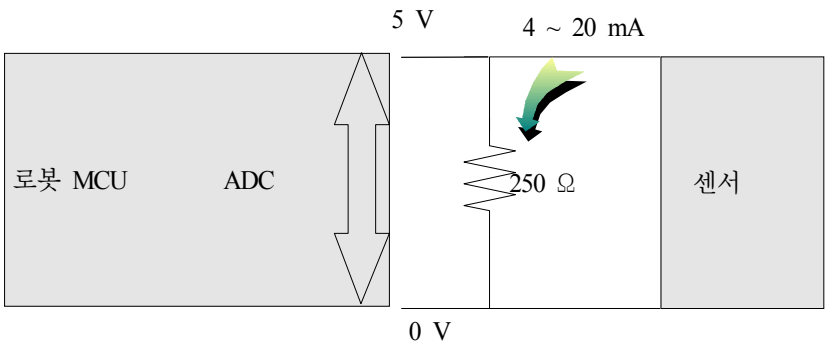
일반적으로 ADC 포트의 입력값 범위는 0 V ~ 5 V (또는 3.3 V)이므로, 모든 아날로그 센서

의 측정값은 이 범위에 맞도록 최종적인 조정이 필요하다. 예를 들어 주어진 압력에 비례하여 전압을 출력하는 압력 센서의 경우, 그 값이 미세하기 때문에 op 앰프를 사용한 증폭기를 이용하여 적절한 값을 출력하게 된다. 또 [그림 1-1]과 같이 출력값의 크기가 1V~10V로 ADC 포트의 입력 범위를 넘어서는 경우, 예를 들어 1/2 전압 분배기 회로를 이용하여 0.5V~5V로 조정할 수 있다.



[그림 1-1] 아날로그 센서 전압 출력값의 조정

산업 현장에서는 전압값이 아니라 전류값(4~20mA)을 출력하는 센서를 선호하는데, 이는 노이즈의 영향이 적어 긴 거리에도 안정적인 측정값을 전송할 수 있기 때문이다. 이 경우에도 센서의 신호는 [그림 1-2]와 같이 최종적으로는 250 Ω의 저항을 연결하여 로봇 MCU의 ADC 포트에 전압으로 입력된다. 측정값이 0mA인 경우는 전선이 단선되었거나 센서가 고장임을 의미한다.

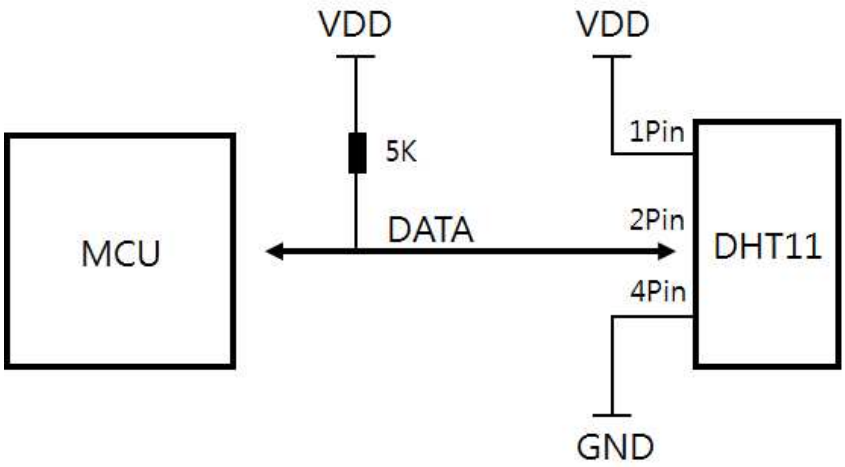


[그림 1-2] 센서 전류 출력의 전압 변환

또 op 앰프의 이득값이나 저항값은 제조상의 오차를 가지고 있기 때문에, 로봇 MCU에서 읽은 값과 실제 측정값이 서로 같도록 확인하고 교정(calibration)하는 작업이 필요하다.

3. 디지털 신호의 내용 및 에러 분석

센서가 아날로그 측정값을 로봇 MCU에 전달할 때에 전선을 통하여 발생되는 노이즈나 ADC 변환 오차를 배제하기 위하여, 센서 내부에서 측정값을 ADC 변환한 후, 이를 디지털 통신을 통하여 전달하는 경우가 있다.



[그림 1-3] 디지털 신호의 송수신

예를 들면, [그림 1-3]과 같이 온습도 센서로 자주 사용되는 DHT11 온습도 모듈의 경우, 아래와 같은 형태의 총 40 비트의 디지털 신호를 송신한다.

데이터 형식(data format): 8 비트 습도값(A) + 8 비트 습도값 소수점 이하(B) + 8 비트 온도값(C) + 8 비트 온도값 소수점 이하(D) + 체크섬(E)

이 경우 센서를 통해 측정된 습도는 A.B %이고, 온도는 C.D °이며, 체크섬은 E이다. 체크섬은 디지털 통신을 통하여 전달된 패킷에 대한 오류 유무를 확인하기 위하여 측정값 외에 추가로 전달되는 값인데, DHT11 센서는 데이터의 합, 즉 E = A + B + C + D로 계산하여 하위 8 비트를 전송한다. 로봇 MCU는 전달받은 A, B, C, D로 계산한 체크섬을 전달받은 E와 비교하여 동일한 경우에만 센서값을 사용하도록 하면 된다.

4. 디지털 통신 신호의 내용 및 에러 분석

디지털 통신으로 전달되는 신호의 크기나 주기 등을 표준화된 통신 방법을 사용하여 전달하는 센서들도 많은데, 주로 RS-232(recommended standard 232, version C)방식의 통신을 사용한다. RS-232는 -15 ~ 15 V를 사용하는 1 : 1 방식의 통신 방식, RS-422은 -7 ~ 7 V를 사용하는 1 : N 방식, RS-485는 -7 ~ 12 V를 사용하는 N : N 방식의 통신 규약이다. 로봇 MCU는 이 신호를 0 V와 5 V (또는 3.3 V)로 변환하기 위하여 주변 회로 장치를 필요로 한다. 대표적인 것이 RS-232를 지원하는 [그림 1-4]의 MAX 232 칩이다.

[그림 1-4] MAX232

(1) RS-232C의 주요 사양

single ended mode(송신선이 송신 신호선 1개와 공통 접지선으로 이루어지고, 수신선이 수신 신호선 1개와 공통 접지선으로 이루어짐)

1개의 송(수)신기와 1개의 수(송)신기

최대 15 m 내외의 통신에 사용

최대 20 Kbps의 통신 속도

full duplex(송신과 수신이 동시에 이루어짐)

-15 V ~ 15 V 신호 사용

(2) RS-422의 주요 사양

differential mode(송수신선이 각각 +와 -의 2개의 선으로 이루어짐)

1개의 송신기와 다수의 수신기

최대 1.2 Km 내외의 통신에 사용

10 Mbps의 통신 속도

full duplex

-7 V ~ 7 V 신호 사용

(3) RS-485의 주요 사양

differential mode(송수신선이 각각 +와 -의 2개의 선으로 이루어짐)

다수의 송신기와 다수의 수신기

최대 1.2 Km 내외의 통신에 사용

10 Mbps의 통신 속도

half duplex(송신할 때는 수신할 수 없고, 수신할 때는 송신할 수 없음)

-7 V ~ 12 V 신호 사용

일반적으로 로봇 MCU가 수신한 디지털 통신 패킷의 내용은 센서의 제조사 별로 각각 다르다. 그러므로 이를 확인하기 위하여 센서의 매뉴얼을 살펴보는 것이 반드시 필요하다. GPS 센서의 경우에는 대부분의 제조사가 NMEA0183(national marine electronics association)이라 불리는 국제 표준에 따라 데이터 형식을 따르고 있다. 여기에는 [그림 1-5]와 같이 GPRMC, GPGGA, GPGSV, GPVTG 등 수십여 개의 문장들의 형식이 정해져 있다.

```
$GPGGA,112135.000,3508.0750,N,12906.2135,E,1,4,1.93,78.0,M,25.4,M,,*6F
$GPGSA,A,3,02,15,21,29,,,,,,,,,2.16,1.93,0.96*0D
$GPGSV,3,1,09,13,83,344,16,15,55,252,37,42,47,161,37,02,35,154,40*7D
$GPGSV,3,2,09,29,28,255,32,30,23,052,21,21,20,316,30,07,05,034,23*77
$GPGSV,3,3,09,18,01,287,*45
$GPRMC,112135.000,A,3508.0750,N,12906.2135,E,0.91,200.31,270816,,,A*6C
$GPVTG,200.31,T,,M,0.91,N,1.69,K,A*3B

$GPGGA,112136.000,3508.0749,N,12906.2135,E,1,4,1.93,77.5,M,25.4,M,,*6E
$GPGSA,A,3,02,15,21,29,,,,,,,,,2.16,1.93,0.96*0D
$GPRMC,112136.000,A,3508.0749,N,12906.2135,E,0.95,202.01,270816,,,A*62
$GPVTG,202.01,T,,M,0.95,N,1.75,K,A*33
```

[그림 1-5] GPS 신호의 예 (2016. 8. 27, 부경대학교에서 수신)

여기에서 GPRMC 문장의 내용을 해석한 예는 다음과 같다. NMEA0183에서 체크섬은 $와 *사이의 모든 문자들을 XOR한 것으로 정의된다.

```
$GPRMC,112135.000,A,3508.0750,N,12906.2135,E,0.91,200.31,270816,,,A*6C
```

112135.000	수신 시각 11:21:35 UTC
A	수신기 신뢰성 A = OK, V = warning
3508.0750,N	위도 35 도 08.0750 분 North
12906.2135,W	경도 129 도 06.2135 분 West
0.91	지상 속도, Knots
200.31	진행 방위
270816	수신 날짜 27 Aug. 2016
6C	체크섬

GPGGA 문장의 내용 해석 예는 다음과 같다.

$GPGGA,112135.000,3508.0750,N,12906.2135,E,1,4,1.93,78.0,M,25.4,M,,*6F	
112135.000	수신 시각 11:21:35.000
3508.0750,N	위도 35도 08.0750분 North
12906.2135,E	경도 129도 06.2135분 East
1	Fix의 종류, 0 = Invalid, 1 = GPS, 2 = DGPS
4	계산에 사용한 위성의 개수
1.93	horizontal dilution of Position
78.0,M	해수면 기준 고도
25.4,M	WGS-84 타원체 기준 고도
6F	체크섬

수행 내용 / 로봇 센서 사양 분석하기

재료・자료

- 센서 데이터시트
- RS-232C, RS-485, 이더넷, USB 등의 규격서
- 로봇에 대한 사용자 요구 사양서

기기(장비・공구)

- 컴퓨터, 프로젝트, 프린터

안전・유의 사항

- 해당 사항 없음

수행 순서

1 로봇 구동을 위하여 필요한 센서를 파악하여, 사용 목적과 제원을 도출한다.

로봇 구동을 위하여 필요한 센서를 파악하여, 사용 목적과 제원을 도출한다. 여기에서는 센서 신호 처리를 위하여 다음과 같이 간단한 상황을 가정한다.

로봇은 초기화 버튼과 3개의 버튼을 가진다. 각 버튼은 눌러지면 아래의 작업을 하는 것으로 가정한다.

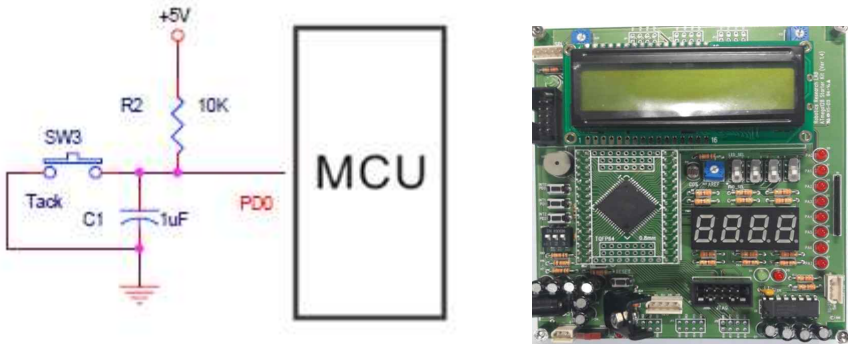
- 버튼 1: 주변의 온도와 습도를 감지한다.
- 버튼 2: 주변의 밝기를 감지한다.
- 버튼 3: 현재 위치를 파악한다.

2 선정된 로봇 센서의 제원을 파악하고, 작동 원리를 정리한다.

1. 사용자가 버튼을 눌렀는지 여부를 감지하기 위하여 스위치를 사용한다.

누름 버튼 스위치로 사용하는 BL150-N-A(LOCK)의 특징은 다음의 웹사이트에서 확인할 수 있다(http://www.devicemart.co.kr/12701).

로봇 MCU는 ATmega128을 사용하는 [그림 1-6]의 스타터키트를 사용하고, MCU의 PORT D를 사용하기로 한다. PORT D의 0~3번 핀은 인터럽트가 가능한 디지털 입력으로 사용 가능하다. 해당 스타터키트는 3개의 누름 버튼 스위치가 PORT D의 0~2번 핀에 연결되어 있다.



(1) MCU와의 연결도 (2) 스타터 키트

[그림 1-6] 스위치와 MCU의 연결 방법

[그림 1-6]의 회로는 스위치를 누르면 0, 누르지 않으면 1을 로봇 MCU에 전달하는 회로이다. 스위치가 눌러지지 않은 경우에는 접지로 흐르는 직류 전류는 단전되어 1(5 V)의 값이 디지털 입력 D PORT의 0번 핀에 그대로 입력된다. 스위치가 눌러진 경우 저항에 흐르는 전류값과 전력은

$$\frac{5}{10K} = 0.5m(A),$$

$$5\,V \times 0.5mA = 2.5mW$$

이기 때문에 1/4 W의 일반적인 저항을 사용할 수 있다.

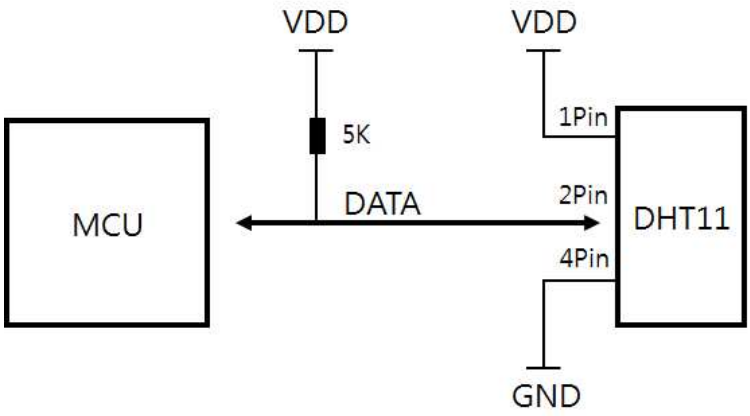
커패시터는 스위치 접점의 기계적인 진동 등으로 생길 수 있는 채터링(chattering)에 의한 고주파 성분을 접지로 통과시켜, 최종적으로는 직류 성분만 MCU에 전달하는 역할을 한다.

2. 온도와 습도를 감지하기 위하여 DHT11을 사용한다.

DHT11은 디지털 출력을 가진 센서로, 자세한 인터페이스 방식은 매뉴얼을 참고한다 (http://www.micropik.com/PDF/dht11.pdf).

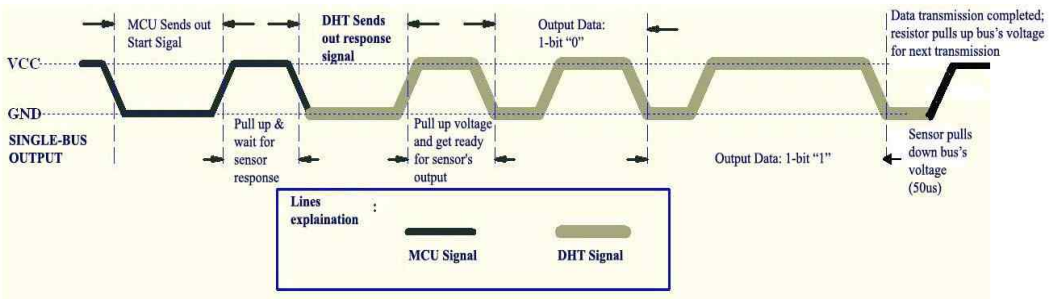
매뉴얼에 의하면, DHT11은 4개의 핀을 가지고 있는데, [그림 1-7]과 같이 1번 핀은 전원(3 V ~ 5.5 V)에, 4번 핀은 GND에 연결한다. 2번 핀은 MCU의 디지털 입력 핀에 연결하되 전선의 길이가 20 m 이내인 경우 5 KΩ의 저항 전원 사이에 연결한다. 3번 핀은 사용하지 않는다.

로봇 MCU는 ATmega128을 사용하는 스타터키트를 사용하고, D 포트의 7번 핀을 사용하기로 한다.



[그림 1-7] DHT11과 MCU의 연결 방법

[그림 1-8]과 매뉴얼에 의하면, DHT11은 다음과 같은 순서로 동작된다.



출처: DHT11 sensor manual(http://www.micropik.com/PDF/dht11.pdf). 2016. 10. 3. 스크린샷.
[그림 1-8] DHT11 센서의 전체적인 동작 방식

- MCU의 D 포트 7번 핀은 DHT11의 2번 핀과 연결되어 있으며, 이 신호선 하나를 입력 및 출력으로 사용한다.
- MCU가 신호선에 0(18 ms 이상)에서 1(20 ~ 40 us)로 변화하는 신호를 출력하면 통신이 시작된다.
- DHT11은 신호선에 0(80 us)에서 1(80 us)로 변화하는 신호를 출력하여 대답한다.
- DHT11은 이후부터 0(50 us)에서 1로 변화하는 40개의 신호를 출력하는데, [그림 1-9]에서 나타낸 바와 같이 1로 유지되는 기간이 26 ~ 28 us이면 0을 의미하고, 70 us이면 1을 의미한다.
- MCU는 이 신호를 순서대로 읽어서 데이터 패킷을 형성한다.
  데이터 형식(Data Format): 8 비트 습도값(A) + 8 비트 습도값 소수점 이하(B) + 8 비트 온도값(C) + 8 비트 온도값 소수점 이하(D) + 체크섬(E)
- 이 경우, 센서를 통해 측정된 습도는 A.B %이고, 온도는 C.D °이고, 체크섬은 E이다. 체크섬은 데이터의 합, 즉 E = A + B + C + D로 계산하여 하위 8 비트로 정의된다. 로봇 MCU는 전달받은 A, B, C, D로 계산한 체크섬을 전달받은 E와 비교하여 동일한 경우에 제대로 센서값을 받았다고 판단한다.

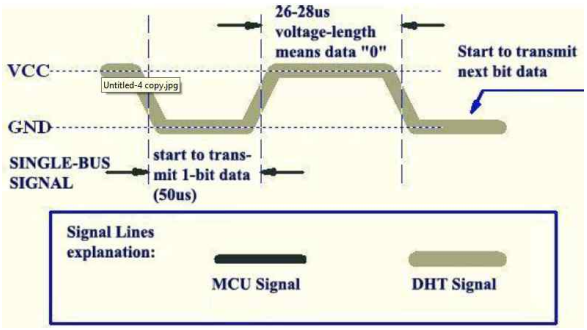


Figure 4 Data "0" Indication

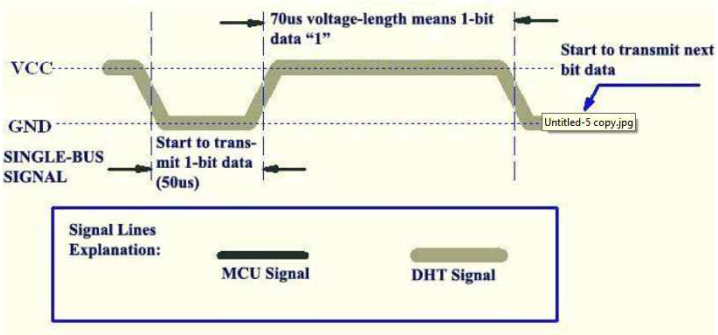


Figure 5 Data "1" Indication

출처: DHT11 sensor manual(http://www.micropik.com/PDF/dht11.pdf). 2016. 10. 3. 스크린샷.
[그림 1-9] DHT11 센서의 0 신호와 1 신호

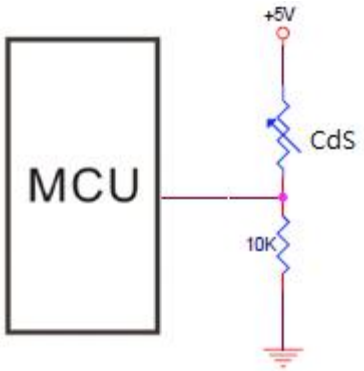
3. 주변의 밝기를 감지하기 위하여 CdS 센서를 사용한다.

CdS 센서는 밝기에 반비례하여 저항이 바뀌는 소자로 이를 이용하여 아날로그 출력을 만들 수 있다. 자세한 내용은 매뉴얼을 참고한다.

(http://www.devicemart.co.kr/include/down.php?file=/data/goods/goodsfile/33219_file_0.pdf&mode=goods&name=D7-CdS-e.pdf)

센서 사양에 의하면 10 Lux의 빛이 있는 경우 10~20 KΩ의 저항을 가지고, 반대의 경우에 1 MΩ을 나타낸다.

로봇 MCU로는 ATmega128을 사용하는 스타터키트를 사용하고, 해당 스타터키트의 F PORT의 0번 핀이 CdS 센서와 연결되어 있다. PORT F의 0번 핀은 ADC0로도 사용 가능하므로, 해당 핀을 아날로그 입력 핀으로 설정하여 CdS의 신호를 수신한다.



[그림 1-10] CdS 조도 센서와 MCU의 연결 방법

빛이 없는 경우에는 CdS의 저항이 1 MΩ이므로 10 KΩ 저항에 걸리는 전압은 아래와 같이 계산된다.

$$\frac{10K}{10K+1M}\times 5\,V = 0.05\,V$$

빛이 있는 경우, CdS의 저항이 20 KΩ인 경우는

$$\frac{10K}{10K+20K}\times 5\,V = 1.67\,V$$

이고, 더 강한 빛의 경우에 CdS의 저항이 10 KΩ으로 낮아진 경우

$$\frac{10K}{10K+10K}\times 5\,V = 2.5\,V$$

가 출력된다. 이때 CdS와 저항에 흐르는 전류와 전력은 각각

$$\frac{5}{10K+10K} = 0.25\,mA,$$

$$2.5\,V \times 0.25\,mA = 0.625\,mW$$

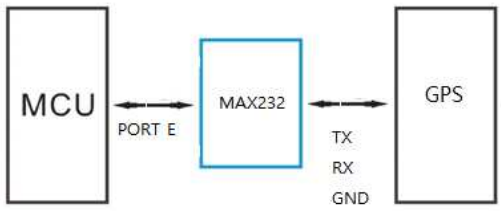
이므로 매뉴얼에 의하면 100 W까지 견디는 CdS 소자와 흔히 사용하는 1/4 W 저항을 사용하면 안전하다.

4. 현재 로봇의 위치를 파악하기 위하여 GPS를 사용한다.

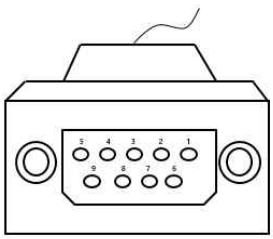
GPS는 NMEA0183 프로토콜에 따라 정보를 전송하며, RS-232를 사용하는 것으로 선정하였다. 자세한 내용은 매뉴얼을 참고한다.
(http://ascenkor.img.or.kr/downloads/Ascen-GPS631A-Datasheet.pdf)

로봇 MCU로는 ATmega128을 사용하는 스타터키트를 사용하고, PORT E의 0번과 1번 핀을 사용하기로 한다. ATmega128의 E PORT의 0번 핀은 USART의 RXD0로, 1번 핀은 USART의 TXD0로 사용 가능하므로, 해당 핀을 시리얼 통신용으로 설정하여 GPS로부터의 신호를 수신한다.

선정된 GPS는 내부에 RS 232 모듈을 내장하고 있어 -15 ~ 15 V의 신호를 송출하므로 [그림 1-11]과 같이 MAX 232를 사용하여 MCU가 받아들일 수 있는 0 ~ 5 V의 신호로 변환하여야 한다.



(1) GPS와 MCU의 연결도



번호	설명
1	VCC
2	RX
3	TX
5	GND
4, 6, 7, 8, 9	not connected

(2) 핀 배열

[그림 1-11] GPS와 MCU의 연결 방법

이 GPS의 데이터는 다음과 같은 방식으로 한 글자씩 비동기 전송된다.

- baud rate: 9600
- 데이터 비트: 8
- 스톱 비트: 1
- 흐름 제어: N (없음)

매뉴얼에 의하면 해당 GPS는 NMEA 출력 문장 중 GPGGA, GPGSA, GPGSV, GPRMC를 제공한다. GPGGA의 경우, $GPGGA로 시작되는 데이터를 [그림 1-12]의 예제를 참고하여 적절히 해석하여야 한다.

$GPGGA,064951.000,2307.1256,N,12016.4438,E,1,8,0.95,39.9,M,17.8,M,,*65

Table-2: GGA Data Format

Name	Example	Units	Description
Message ID	$GPGGA		GGA protocol header
UTC Time	064951.000		hhmmss.sss
Latitude	2307.1256		ddmm.mmmm
N/S Indicator	N		N=north or S=south
Longitude	12016.4438		dddmm.mmmm
E/W Indicator	E		E=east or W=west
Position Fix Indicator	1		See Table-3
Satellites Used	8		Range 0 to 14
HDOP	0.95		Horizontal Dilution of Precision
MSL Altitude	39.9	meters	Antenna Altitude above/below mean-sea level
Units	M	meters	Units of antenna altitude
Geoidal Separation	17.8	meters	
Units	M	meters	Units of geoid separation
Age of Diff. Corr.		second	Null fields when DGPS is not used
Checksum	*65		
<CR> <LF>			End of message termination

Table-3: Position Fix Indicator

Value	Description
0	Fix not available
1	GPS fix
2	Differential GPS fix

출처: GPS-631A Datasheet(http://ascenkor.img.or.kr/downloads/Ascen-GPS631A-Datasheet.pdf). 2016. 10. 3. 스크린샷.
[그림 1-12] GPGGA 신호의 해석

GPGSA의 경우, $GPGSA로 시작되는 데이터를 [그림 1-13]의 예제를 참고하여 적절히 해석하여야 한다.

$GPGSA,A,3,29,21,26,15,18,09,06,10,,,,,2.32,0.95,2.11*00

Table-4: GSA Data Format			
Name	Example	Units	Description
Message ID	$GPGSA		GSA protocol header
Mode 1	A		See Table-5
Mode 2	3		See Table-6
Satellite Used	29		SV on Channel 1
Satellite Used	21		SV on Channel 2
....	....	....	....
Satellite Used			SV on Channel 12
PDOP	2.32		Position Dilution of Precision
HDOP	0.95		Horizontal Dilution of Precision
VDOP	2.11		Vertical Dilution of Precision
Checksum	*00		
<CR> <LF>			End of message termination

Table-5: Mode 1	
Value	Description
M	Manual—forced to operate in 2D or 3D mode
A	2D Automatic—allowed to automatically switch 2D/3D

Table-6: Mode 2	
Value	Description
1	Fix not available
2	2D (<4 SVs used)
3	3D (≥4 SVs used)

출처: GPS-631A Datasheet(http://ascenkor.img.or.kr/downloads/Ascen-GPS631A-Datasheet.pdf). 2016. 10. 3. 스크린샷.
[그림 1-13] GPGSA 신호의 해석

GPGSV의 경우, $GPGSV로 시작되는 데이터를 [그림 1-14]의 예제를 참고하여 적절히 해석하여야 한다.

$GPGSV,3,1,09,29,36,029,42,21,46,314,43,26,44,020,43,15,21,321,39*7D

$GPGSV,3,2,09,18,26,314,40,09,57,170,44,06,20,229,37,10,26,084,37*77
$GPGSV,3,3,09,07,,,26*73

Table-7: GSV Data Format			
Name	Example	Units	Description
Message ID	$GPGSV		GSV protocol header
Number of Messages	3		Range 1 to 3 *(Depending on the number of satellites tracked, multiple messages of GSV data may be required.)*
Message Number1	1		Range 1 to 3
Satellites in View	09		
Satellite ID	29		Channel 1 (Range 1 to 32)
Elevation	36	degrees	Channel 1 (Maximum 90)
Azimuth	029	degrees	Channel 1 (True, Range 0 to 359)
SNR (C/No)	42	dBHz	Range 0 to 99, (null when not tracking)
....	....	....	....
Satellite ID	15		Channel 4 (Range 1 to 32)
Elevation	21	degrees	Channel 4 (Maximum 90)
Azimuth	321	degrees	Channel 4 (True, Range 0 to 359)
SNR (C/No)	39	dBHz	Range 0 to 99, (null when not tracking)
Checksum	*7D		
<CR> <LF>			End of message termination

출처: GPS-631A Datasheet(http://ascenkor.img.or.kr/downloads/Ascen-GPS631A-Datasheet.pdf). 2016. 10. 3. 스크린샷.
[그림 1-14] GPGSV 신호의 해석

GPRMC의 경우, $GPRMC로 시작되는 데이터를 [그림 1-15]의 예제를 참고하여 적절히 해석하여야 한다.

$GPRMC,064951.000,A,2307.1256,N,12016.4438,E,0.03,165.48,260406, 3.05,W,A*55

Table-8: RMC Data Format			
Name	**Example**	**Units**	**Description**
Message ID	$GPRMC		RMC protocol header
UTC Time	064951.000		hhmmss.sss
Status	A		A=data valid or V=data not valid
Latitude	2307.1256		ddmm.mmmm
N/S Indicator	N		N=north or S=south
Longitude	12016.4438		dddmm.mmmm
E/W Indicator	E		E=east or W=west
Speed over Ground	0.03	knots	
Course over Ground	165.48	degrees	True
Date	260406		ddmmyy
Magnetic Variation	3.05,W	degrees	E=east or W=west (Need AscenKorea Customization Service)
Mode	A		A= Autonomous mode D= Differential mode E= Estimated mode
Checksum	*55		
<CR> <LF>			End of message termination

출처: GPS-631A Datasheet(http://ascenkor.img.or.kr/downloads/Ascen-GPS631A-Datasheet.pdf). 2016. 10. 3. 스크린샷.
[그림 1-15] GPRMC 신호의 해석

수행 tip

- 사용 센서의 매뉴얼과 데이터시트를 읽는 것은 대단히 중요하다.
- 통신으로 전달되는 센서의 내용은 정확하고 세밀하게 분석해야 한다.

2-1. 로봇 센서 프로토콜 설계

학습 목표

- 로봇 센서 인터페이스 하드웨어를 분류할 수 있다.
- 로봇 센서 인터페이스를 위한 프로토콜을 분석할 수 있다.
- 센서 사양에 맞는 통신 프로토콜을 설계할 수 있다.

필요 지식 /

① 로봇 센서의 출력 확인

센서의 출력값에 따라 다음과 같이 센서를 분류할 수 있다.

- 전압 출력형: 측정값을 전압으로 출력함
- 전류 출력형: 측정값을 전류로 출력함. 저항을 연결하여 전압으로 변경함

결국은 전압값이 로봇 MCU의 입력 핀으로 출력된다. 이때 전압값의 신호에 따라 다음과 같이 나눌 수 있다.

- 디지털 신호형: 측정값을 디지털 신호로 출력함
- 아날로그 신호형: 측정값을 아날로그 신호로 출력함. ADC를 통해 디지털 신호로 변경함
- 통신 신호형: 정해진 방식의 디지털 통신을 통하여 일련의 신호가 전송됨. Field bus(DeviceNet, CAN, ProfiBus, Ethernet 등)는 ISO 15735 국제 표준으로 정해져있음. 변환기를 통하여 USART(RS-232)로 MCU에 전달되거나, 드라이브 회로를 따로 구성하여 MCU에 연결됨

한편, 센서값을 확인하기 위한 절차에 따라서는 다음과 같이 센서를 분류할 수 있다.

- 스탠바이형: 센서의 측정값이 항상 준비되어 있으므로 MCU가 읽어오기만 하면 됨
- 요구-응답형: MCU가 요청 시에 센서가 측정값을 전송함
- 주기 응답형: MCU의 요청이 없어도 센서가 정해진 주기마다 측정값을 전송함

- 연속 스트림형(continuous stream): 센서의 측정이 완료 되는대로 센서가 계속하여 측정 값을 전송함

2 로봇 센서 신호의 확인 방식

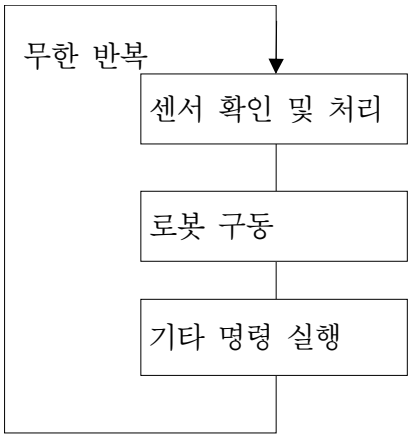
로봇 MCU가 센서로부터의 신호를 확인하는 방법은 폴링 방식과 인터럽트 방식의 두 가지로 나뉜다.

1. 폴링(polling)

폴링 방식이란, 로봇 MCU에서 실행되는 프로그램이 연결된 센서들의 신호를 지속적으로 감시하여 상태를 확인하는 방식이다. 프로그램의 작성이 간단하고 폴링 프로그램이 MCU의 하드웨어와 무관하여 프로그램의 이식성이 높다.

그러나 센서의 신호가 없을 때나 변화가 없는 경우에도 계속하여 MCU의 리소스를 소비하여야 한다는 점과, 폴링 이후의 작업들을 모두 수행한 이후에야 다시 폴링의 차례가 돌아오기 때문에 생기는 폴링 간 시간 지연 등을 고려하여 프로그램해야 한다.

[그림 2-1]은 일반적인 폴링 방식에 의하여 센서값 확인 및 처리, 로봇 구동, 기타 명령의 실행을 반복하는 로봇 MCU 프로그램의 메인 루프를 나타낸다.



[그림 2-1] 폴링 방식의 센서 신호 처리

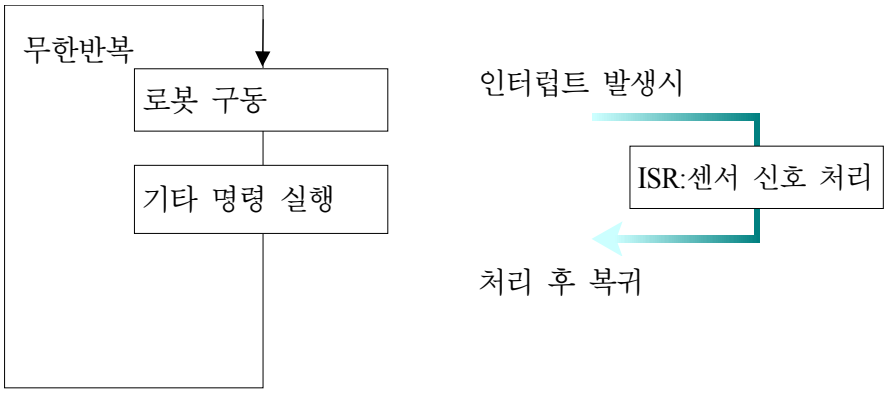
2. 인터럽트(interrupt)

인터럽트 방식이란, 센서로부터의 신호 변경 또는 요청이 있을 때에만 로봇 MCU가 하던 일을 멈추고 정해진 인터럽트 서비스 루틴(ISR, interrupt service routine)을 실행한 후, 다시 하던 일을 계속 하는 방식이다. 인터럽트가 발생하는 순간에 시간 지연 없이 바로 해당 루틴이 실행되므로, 센서 신호에 즉각적인 반응을 할 수 있어 내외부의 서비스 요청에 대하여 MCU가 가장 빨리 대응할 수 있는 방식이다.

그러나 MCU별로 사용 가능한 인터럽트 종류와 처리 방법에 차이가 있으므로, 폴링 방식

에 비해 사용하는 MCU에 대한 보다 자세한 이해를 필요로 한다는 점과 타 MCU로의 이식이 어렵다는 점을 고려하여야 한다.

[그림 2-2]는 로봇 MCU의 메인 루프가 로봇 구동과 기타 명령 실행만을 반복하여 수행하고, 인터럽트 발생 시에만 센서 신호 처리를 한 다음, 원래의 메인 루프로 복귀하는 것을 나타낸다.



[그림 2-2] 인터럽트 방식의 센서 신호 처리

인터럽트를 요청할 수 있는 내외 이벤트(interrupt source)들은 아래의 것들을 포함하며, MCU 별로 지원하는 인터럽트의 종류도 조금씩 차이가 있다.

- RESET: 리셋 버튼, 파워 온, 브라운 아웃(전압 강하 리셋), watch dog, JTAG 리셋 요청.
- 외부 인터럽트: 인터럽트 처리가 가능한 입력 핀에 대한 신호 입력.
- TIMER COMP: 타이머값이 미리 정한 한계치에 다다랐음(최종값을 지정).
- TIMER OVF: 타이머값이 최고치를 넘음(초기값을 지정).
- TIMER CAPT: 정해진 입력값이 들어왔음.
- SPI/STC: SPI 송신 완료.
- TWI: TWI 통신(=I2C) 관련 이벤트.
- USART RX: USART 수신 완료.
- UDRE(usart data register empty); 데이터 레지스터가 비어있음.
- USART TX: USART 송신 완료.
- ADC: ADC 변환이 완료되었음.
- EE READY: EEPROM Ready.
- ANALOG COMP: 아날로그 비교기.
- SPM READY: 프로그램 메모리 준비 완료.
- FIFO FULL: FIFO가 가득 참.
- TRAP: TRAP 명령이 실행됨, 디버거 만들 때 유용함.

SWI(software interrupt): 알 수 없는 명령어.

인터럽트는 개발자의 프로그램에 의하여 실행 여부를 설정할 수 있는 인터럽트(maskable interrupt)와 그렇지 않은 NMI(non maskable interrupt)로 나누어진다. NMI는 기억 장치의 심각한 오류 상황이나 입력 전압의 강하 등 긴급한 상황에서 하드웨어에 의하여 발생되는 인터럽트이다.

또 동시에 여러 개의 인터럽트가 발생되었을 때를 위하여 모든 인터럽트에는 우선순위가 정해져 있다.

MCU는 인터럽트가 발생하였을 때 각 인터럽트별로 정해진 메모리 내의 번지로 실행 순서를 옮기게 되는데 이 주소 공간을 인터럽트 벡터(interrupt vector)라고 부른다. MCU는 인터럽트 벡터에 지정된 인터럽트 서비스 루틴을 실행한다.

3 통신 버퍼의 구성

일반적으로 시리얼 통신, CAN 통신 등의 통신을 통하여 수신되는 데이터들은 비주기적으로 MCU에 도착한다. 폴링 방식으로는 다음 폴링이 실행되는 한 주기보다 빠른 속도로 들어오는 통신 데이터들을 처리할 수 없으므로, 통신 데이터가 수신되는 순간에 인터럽트를 발생시켜 수신 데이터를 저장한 다음, 원래의 작업을 계속하는 것이 더 좋은 방식이다.

이때 데이터를 저장하는 구조는 아래의 세 가지로 나뉜다.

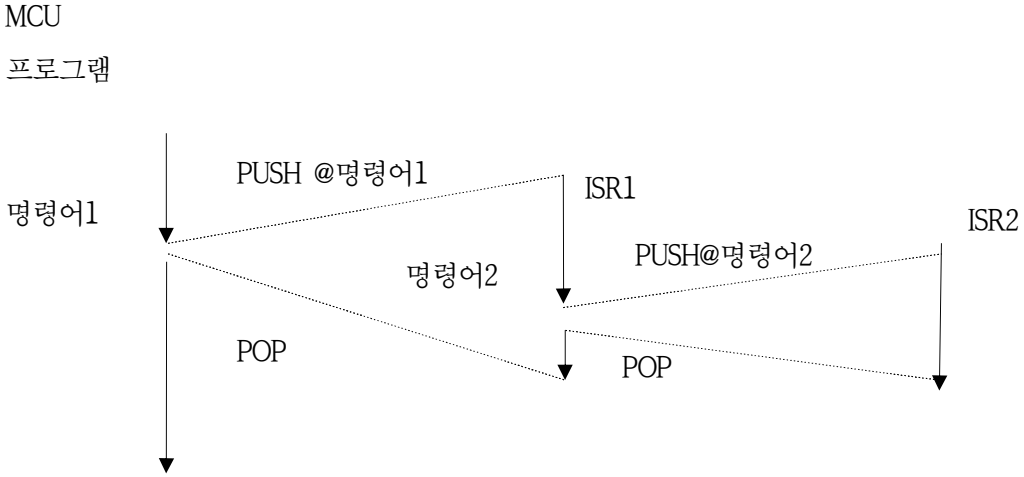
- 스택(stack): 나중에 들어간 데이터가 먼저 출력되는 LIFO(last in first out) 구조의 버퍼
- 큐(queue): 먼저 들어간 데이터가 먼저 출력되는 FIFO(first in first out) 구조의 버퍼
- 링버퍼(ring buffer): 버퍼의 끝과 처음이 연결되어 있는 원형 구조의 큐

1. 스택(stack)

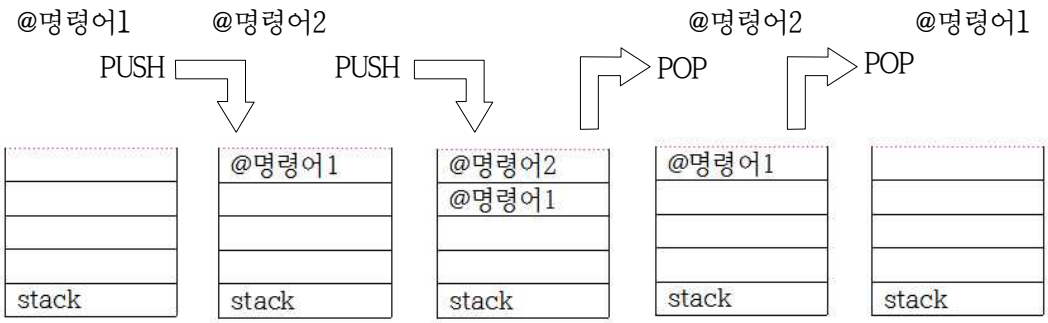
스택은 나중에 들어간 데이터가 먼저 출력되는 LIFO 구조의 버퍼를 말한다. 데이터를 스택에 넣는 것을 푸시(push)라고 하고 스택에서 데이터를 가져오는 것을 팝(pop)이라고 부르기 때문에 푸시-팝 구조라고도 한다. 입구와 출구가 같은 동전 케이스를 생각하면 된다. 스택 구조는 주로 인터럽트나 중첩된 함수의 호출을 위하여 사용되는 구조이다. 예를 들어 [그림 2-3]과 같은 상황을 가정하자. 스택 내용은 변화는 [그림 2-4]에 나타내었다.

- 로봇 MCU가 (명령어1)을 실행하고 있는데, 인터럽트가 발생한다.
- 로봇 MCU는 현재 실행하고 있던 명령어의 주소(@명령어1)를 스택에 저장(PUSH @명령어1)하고 인터럽트 서비스 루틴(ISR1)을 실행한다. 여기에서 @는 메모리의 주소를 나타낸다.
- 인터럽트 서비스 루틴 ISR1 내의 (명령어2)를 실행하고 있는데, 우선순위가 더 높은 인터럽트가 발생한다.
- 현재의 인터럽트 명령어의 위치(@명령어2)를 스택에 저장(PUSH @명령어2)하고 새로운 인터럽트 서비스 루틴(ISR2)을 실행한다.
- ISR2의 실행을 완료한다.

- 스택에 나중에 저장된 내용을 받아(POP @명령어2) 이전의 인터럽트 서비스 루틴(ISR1)을 재개하여 완료한다.
- 스택에 남아있는 명령어의 위치를 받아(POP @명령어1) 기존에 실행하던 로봇 MCU의 명령을 실행한다.



[그림 2-3] 인터럽트 중복 발생 시의 처리



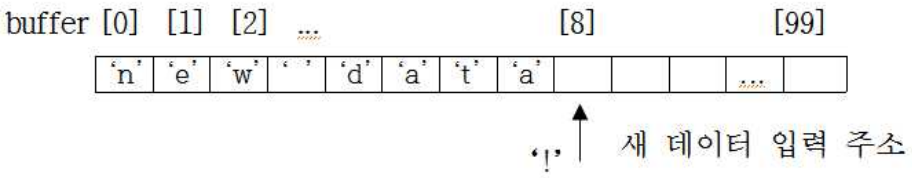
[그림 2-4] 인터럽트 중복 발생 시 스택의 내용 변화

2. 큐(queue)

로봇 MCU의 통신 데이터 수신 인터럽트 서비스 루틴은 가능한 짧고 간결하게 작성되어야 한다. 이것은 인터럽트 서비스 루틴이 실행되는 동안에는 다른 작업을 할 수 없을 뿐만 아니라 동등한 우선순위를 가진 새로운 데이터 수신 인터럽트를 실행할 수 없으므로, 인터럽트 서비스 루틴을 실행하는 동안 수신되는 새 데이터를 처리할 수 없기 때문이다.
그러므로 데이터 수신 인터럽트 서비스 루틴은 수신된 데이터를 가능한 빨리 메모리에 저장하기만 하고, 수신된 데이터를 해석하고 실행하는 것은 데이터의 전문이 완성되었을 때에 MCU의 메인 루틴에서 실행하는 것이 좋은 방법이다.
큐는 먼저 들어간 데이터가 먼저 출력되는 FIFO 구조의 버퍼를 말한다. 시리얼 통신의 수

신 인터럽트 발생 시, 큐 구조를 사용하는 인터럽트 서비스 루틴은 수신된 데이터를 직전 데이터의 다음 장소에 차례대로 저장한다. 예를 들어 [그림 2-5]와 같이 센서로부터 "new data!" 라는 문자열이 수신된다면 새로 도착한 '!' 는 'a' 다음 위치인 buffer[8]에 저장된다.

저장된 데이터를 꺼내는 순서는 먼저 저장된 데이터 순서대로이다. 데이터를 확인하는 순서는 항상 제일 처음 도착한 buffer[0]부터이며, 이 경우에 원래의 수신 데이터를 순서대로 확인할 수 있다. 수신 데이터를 확인하고 나면 로봇 MCU는 다시 buffer[0]부터 수신 데이터를 채워나간다.

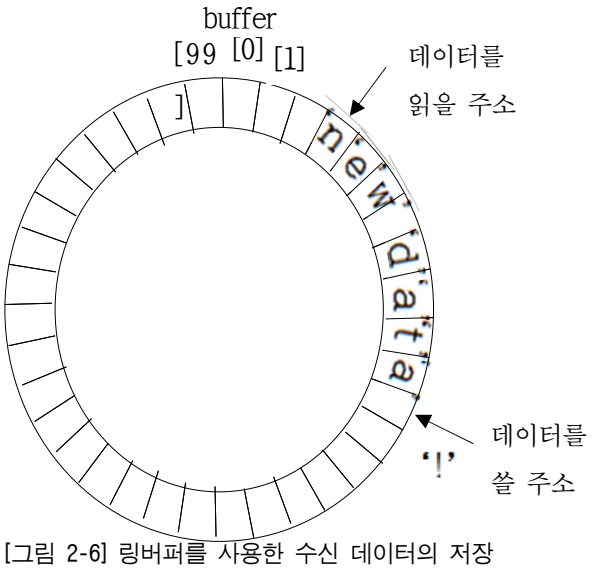


[그림 2-5] 큐를 사용한 수신 데이터의 저장

3. 링버퍼(ring buffer)

큐 구조는 출력을 위한 버퍼의 시작 주소가 항상 0이고, 입력되는 데이터의 버퍼 주소만 하나씩 증가한다. 입력되는 데이터의 길이는 제공된 큐의 길이를 초과할 수 없다.

링버퍼는 큐의 마지막과 시작이 연결된 것으로 가정하는 구조이다. [그림 2-6]에서 buffer의 마지막 공간인 buffer[99]의 다음은 buffer[0]으로 연결됨을 볼 수 있다. 링버퍼는 데이터의 입력 버퍼 주소만을 관리하는 큐와는 달리 2개의 포인터를 사용하는데, 보통 데이터의 입력 버퍼 주소를 pHead로, 데이터의 출력 버퍼 시작 주소를 pTail로 표기한다.



[그림 2-6] 링버퍼를 사용한 수신 데이터의 저장

수신된 데이터는 큐에서와 마찬가지로 최종 수신된 데이터의 다음 장소인 pHead가 가리키는 곳에 저장된다. 큐 구조에서 데이터를 읽을 주소가 buffer[0]로 고정되어 있는 것과 달리, 링버퍼 구조에서는 데이터를 읽을 주소가 이전에 읽은 데이터의 다음 주소인 pTail이 가리키는 곳이 된다.
링버퍼에 저장되어 있는 데이터를 읽을 때마다 pTail은 pHead쪽으로 다가가게 된다. 데이터를 이미 읽어낸 저장 장소는 새 데이터를 쓸 수 있는 공간이 된다. 그러므로 수신 인터럽트 서비스 루틴을 통하여 링버퍼에 저장되는 통신 데이터는, MCU에서 적절한 간격으로 링버퍼에서 읽어내기만 하면 수신 데이터의 길이에 제한이 없게 된다.
링버퍼의 쓰기 주소인 pHead는 다음과 같이 변화시키면 된다. 이것은 pTail에도 마찬가지로 적용된다.

$$pHead = (pHead + 1)\%BUFFER\_SIZE$$

여기에서 BUFFER_SIZE는 버퍼의 크기이고 %는 나머지 연산을 뜻한다.
예를 들어 현재의 데이터 쓰기 주소인 $pHead = 10$이라면, 그 다음 쓰기 주소는

$$pHead = (10+1)\%100 = 11,$$

현재의 쓰기 주소가 $pHead = 99$이라면, 그 다음 쓰기 주소는

$$pHead = (99+1)\%100 = 0$$

이므로 버퍼가 처음으로 연결된다.

링버퍼에 들어있는 데이터의 개수는 다음과 같이 구할 수 있다.

$$numData = (pHead - pTail + BUFFER\_SIZE)\%BUFFER\_SIZE$$

예를 들어 그림과 같이 '!' 데이터를 수신하기 직전이라면 버퍼 안의 데이터는 "new data" 이고,

$$pHead = 11$$

$$pTail = 3$$

이므로,

$$numData = (11-3+100)\%100 = 8$$

이다.

만약, 같은 데이터가 buffer[95], buffer[96], buffer[97], buffer[98], buffer[99], buffer[0], buffer[1], buffer[2]에 들어있다면

$$pHead = 3$$

$$pTail = 95$$

$$numData = (3-95+100)\%100 = 8.$$

수행 내용 / 로봇 센서 프로토콜 설계하기

재료・자료

- 센서 데이터시트
- 전기·전자 부품 데이터시트
- 전기·전자 부품 어플리케이션 노트
- RS-232C, RS-485, 이더넷, USB 등의 규격서

기기(장비・공구)

- 컴퓨터, 프린터
- 펌웨어 개발환경. http://www.atmel.com/tools/atmelstudio.aspx 에서 Atmel Studio 최신 버전 다운로드
- ATmega128 스타터키트. http://www.coenfirm.co.kr/bbs/board.php?bo_table=prod4&wr_id=7
- ISP 또는 JTAG 장비(윈도우 및 Atmel Studio 최신 버전과 호환성 확인 필요) http://www.atmel.com/tools/AVRJTAGICEMKII.aspx?tab=overview
- 문서 작성용 소프트웨어

안전・유의 사항

- 펌웨어 개발환경으로 사용되는 Atmel Studio의 경우, 파일의 경로나 이름에 한글이 허용되지 않음을 유의한다.
- 컴퓨터 환경에 따라 일부 ISP와 JTAG 에뮬레이터가 지원되지 않을 수 있으므로, 지원 여부를 반드시 확인하여야 한다.
- 실습 장비에 대한 전원 투입 시, 정확한 전압과 극성을 인가하여야 한다.

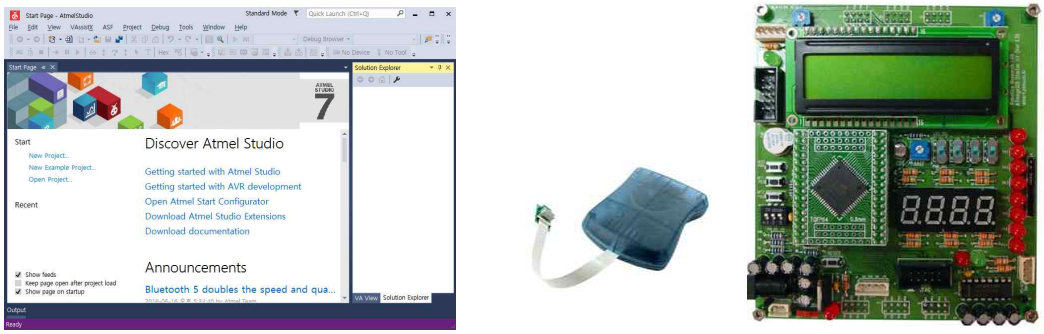
수행 순서

1 로봇 센서의 종류에 따른 인터페이스 하드웨어를 분류하고, 하드웨어에 맞는 입력 모듈을 구상한다.

1. 로봇 센서 인터페이스에 따라 로봇 MCU의 입력 모듈을 구상한다. 여기에서는 센서 신호 처리를 위하여 다음과 같이 간단한 상황을 가정한다.
   (1) 버튼 1: 인터럽트 방식에 의하여 스위치 입력을 확인한다. ISR1이 호출되었음을 LCD에 표시한다.
   (2) 버튼 2: 인터럽트 방식에 의하여 스위치 입력을 확인한다. ISR2가 호출되었음을 LCD에 표시한다.
   (3) 버튼 3: 폴링 방식에 의하여 스위치 입력을 확인한다. 폴링 프로시저가 호출되었음을 LCD에 표시한다.

2 선정된 로봇 센서에 맞는 입력 모듈의 방식을 익히기 위하여, 폴링 및 인터럽트 방식에 의한 디지털 입력 확인 방식을 연습한다.

1. 사용자가 버튼을 눌렀는지를 감지하기 위하여 스위치 입력을 감시하는 모듈을 설계한다. 여기에서는 로봇 MCU로는 ATmega128을 사용하는 스타터키트를 사용하고, MCU의 PORT D를 사용하기로 한다. ATmega128의 D PORT의 0~3번 핀은 인터럽트 처리가 가능한 디지털 입력으로 사용 가능하다. 스타터키트는 3개의 누름 버튼 스위치가 PORT D의 0~2번 핀에 연결되어 있다.



출처: JTAG(http://www.atmel.com/tools/AVRJTAGICEMKII.aspx?tab=overview). 2016. 10. 3. 스크린샷.
Coenfirm(http://www.coenfirm.co.kr/bbs/board.php?bo_table=prod4&wr_id=7). 2016. 10. 3. 스크린샷.
[그림 2-7] Atmel Studio, JTAG 에뮬레이터, ATmega128 스타터키트

2. Atmel Studio에서 [New Project...]를 생성한다.

- GCC C Excutable Project
- Name: Test2 (Atmel Studio 7.0 버전에서는 파일 경로와 이름에 한글이 있으면 안 됨)
- Device Selection: ATmega128

[그림 2-8]과 같이 main.c의 내용을 입력한다.

```
/*
 * Test2.c
 *
 * Created: 2016-07-24 오후 7:15:19
 * Author : gabi http://cal.pknu.ac.kr
 */

#define F_CPU       16000000UL          // 16M Hz
#include <avr/io.h>
#include <util/delay.h>
#include <avr/interrupt.h>
#include "lcd.h"

// 입출력 포트 초기화
void ioport_set(void)
{
        DDRA = 0xFF;     // PORT A는 8핀 모두 출력(LED 0~7)      (0b 1111 1111)
        DDRD = 0xF8;     // PORT D는 0~2핀은 입력 (Button 0~2)  (0b 1111 1000)

        DDRC = 0xFF;     // PORT C는 8핀 모두 출력(LCD)
        DDRE = 0xE0;     // PORT E는 5~7핀이 출력 (LCD)
}

// 인터럽트 초기화 서브루틴
void init_interrupt(void)
{
```

```
        EIMSK = 0x03;     // INT0, 1 활성화, (0b 0000 0011)
        EICRA = 0x0F;     // INT0, 1 상승 에지 선택, (0b 0000 1111)
//      EIFR = 0x00;

        sei();                          // 인터럽트 가능
}

// INT0 외부 인터럽트 루틴
ISR(INT0_vect)
{
        lcd_gotoxy(0,2); lcd_puts(">>Button1 ISR          "); // 20자
}

// INT1 외부 인터럽트 루틴
ISR(INT1_vect)
{
        lcd_gotoxy(0,2); lcd_puts("..Button2 ISR          "); // 20자
}

// 메인 루틴
int main(void)
{
        ioport_set();
        init_interrupt();

        lcd_init();

        lcd_gotoxy(0,1); lcd_puts("   [ robot MCU ]   ");
        lcd_gotoxy(0,2); lcd_puts("   ATmega128 Go! ");

        while(1){
                PORTA = PIND | 0xF8;
                //PORT D에 연결된 버튼에 따라 PORT A에 연결된 LED를 켬
                if (~PIND & 0b00000100){
                        lcd_gotoxy(0,2); lcd_puts("   ATmega128 Go! ");
                }
        }
}
```

[그림 2-8] Starter kit의 예제 프로그램(test1.c) 입력

[그림 2-9]와 같이 lcd.c의 내용을 입력한다.

```
/*
 * lcd.c
 *
 * Created: 2016-07-24 오후 7:16:39
 * Author : gabi http://cal.pknu.ac.kr
 */
```

```c
#define F_CPU       16000000UL          // 16M Hz
#include <avr/io.h>
#include <util/delay.h>
#include "lcd.h"

// LCD 커서 안보이기
void cursor_off(void)
{
        PORTE &= 0x7F;
        _delay_ms(200);             // power on delay
        lcd_write(0x0C);        // display on, cursor off, blink off
        _delay_ms(100);
}

// LCD 커서 보이기
void cursor_on(void)
{
        PORTE &= 0x7F;
        _delay_ms(200);             // power on delay
        lcd_write(0x0F);        // display on, cursor on, blink on
        _delay_ms(100);
}

// LCD 지우기
void lcd_clear(void)
{
        PORTE &= 0x7F;
        _delay_us(1);
        lcd_write(0x01);
        _delay_ms(4);
}

// LCD에 한글자 보내기
void lcd_write(unsigned char c)
{
        PORTC = c;
        PORTE |= 0x20;
        _delay_us(1);
        PORTE &= 0xDF;
        _delay_us(250);
}

// LCD 초기화
void lcd_init(void)
{
        PORTE &= 0x7F;
        _delay_ms(200);       // power on delay
        lcd_write(0x38);      // 8 bit mode, 1/16 duty, 5x8 font
        lcd_write(0x0F);      // display on, cursor on, blink on
        lcd_write(0x01);      // clear screen
        _delay_ms(100);
```

```
}

// LCD 글자 위치 설정
void lcd_gotoxy(unsigned char x, unsigned char y)
{
        PORTE &= 0x7F;
        _delay_us(1);

        if (y==1) lcd_write(0x80+x);
        else lcd_write(0xC0+x);
}

// LCD에 문장 쓰기
void lcd_puts(char *s)
{
        PORTE |= 0x80;
        _delay_us(1);

        while(*s) lcd_write(*s++);
}
```

[그림 2-9] Starter kit의 예제 프로그램(test1.c) 입력

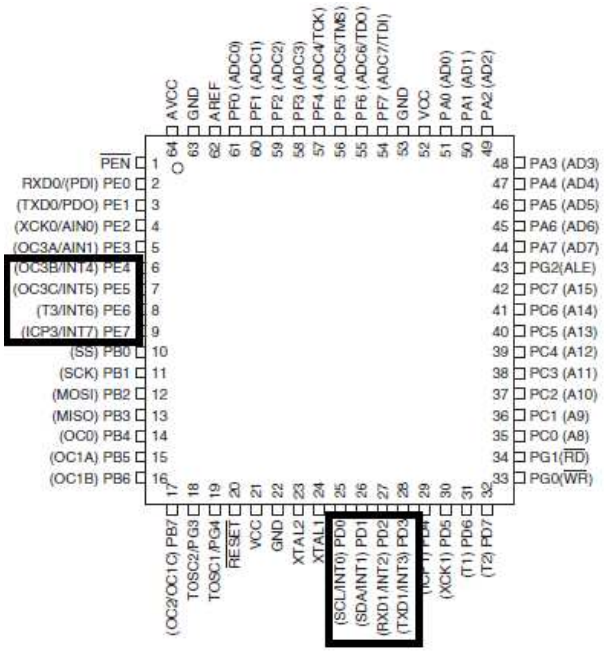
[그림 2-10]과 같이 lcd.h를 입력한다.

```
/*
 * lcd.h
 *
 * Created: 2016-07-24 오후 7:22:20
 * Author : gabi http://cal.pknu.ac.kr
 */
#ifndef __LCD__
#define __LCD__

// LCD 커서 안보이기
void cursor_off(void);

// LCD 커서 보이기
void cursor_on(void);

// LCD 지우기
void lcd_clear(void);

// LCD에 한글자 보내기
void lcd_write(unsigned char c);

// LCD 초기화
void lcd_init(void);

// LCD 글자 위치 설정
```

```
void lcd_gotoxy(unsigned char x, unsigned char y);

// LCD에 문장 쓰기
void lcd_puts(char *s);

#endif
```

[그림 2-10] Starter kit의 예제 프로그램(test1.c) 입력

일반적으로 MCU의 핀은 두 가지 이상의 용도로 사용할 수 있으며, ATmega128의 경우는 [그림 2-11]에서 보이듯이 PORT D의 0~3번 핀과, PORT E의 4~7번 핀을 인터럽트 소스로 사용 가능하다.



출처: ATmega128 manual(http://www.atmel.com/images/doc2467.pdf). 2016. 10. 3. 스크린샷.
[그림 2-11] 외부 인터럽트 핀의 배치

ATmega128은 EIMSK, EICRA, EICRB, SREG 레지스터 값의 설정에 의하여 인터럽트의 특징을 정의한다. [그림 2-12]의 EIMSK(external interrupt mask) 레지스터는 외부 인터럽트의 활성화를 결정한다.

- 해당 비트가 1인 경우 외부 인터럽트를 활성화
- 예를 들면 INT0, INT1이 1인 경우 (0b 0000 0011) PORT D의 0번, 1번 핀을 외부 인터럽트로 사용함

Bit	7	6	5	4	3	2	1	0	
	INT7	INT6	INT5	INT4	INT3	INT2	INT1	IINT0	EIMSK
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

출처: ATmega128 manual(http://www.atmel.com/images/doc2467.pdf). 2016. 10. 3. 스크린샷.
[그림 2-12] EIMSK 레지스터

[그림 2-13]의 EICRA(external interrupt control register A)와 EICRB(external interrup control register B)는 인터럽트를 발생시키는 조건에 관한 것이다.

- 해당 비트가 00, 01, 10, 11이면 각각 0 신호, 신호 변화, 하강 엣지, 상승 엣지일 때 인터럽트를 요청
- 예를 들면 ISC11, ISC10, ISC00, ISC01이 모두 1인 경우 (EICRA = 0b 0000 1111) INT1과 INT0은 상승 엣지(0에서 1로 바뀌는 순간)일 때에 인터럽트가 요청됨

Bit	7	6	5	4	3	2	1	0	
	ISC31	ISC30	ISC21	ISC20	ISC11	ISC10	ISC01	ISC00	EICRA
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Bit	7	6	5	4	3	2	1	0	
	ISC71	ISC70	ISC61	ISC60	ISC51	ISC50	ISC41	ISC40	EICRB
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

출처: ATmega128 manual(http://www.atmel.com/images/doc2467.pdf). 2016. 10. 3. 스크린샷.
[그림 2-13] EICRA, EICRB 레지스터

인터럽트의 실행 가능 여부는 [그림 2-14]의 SREG(status register)의 최상위 비트를 설정함으로써 정해진다.

- SREG의 7번 비트가 1인 경우 모든 인터럽트를 사용할 수 있는 상태가 됨
- sei() 명령은 7번 비트를 1로, cli()는 7번 비트를 0으로 만듦

Bit	7	6	5	4	3	2	1	0	
	I	T	H	S	V	N	Z	C	SREG
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

출처: ATmega128 manual(http://www.atmel.com/images/doc2467.pdf). 2016. 10. 3. 스크린샷.
[그림 2-14] SREG 레지스터

각 포트의 입출력은 [그림 2-15]의 입출력의 방향을 나타내는 DDR(data direction register)

을 설정함으로써 정해진다. 즉, A 포트의 입출력 방향은 DDRA를 사용하고, B 포트의 입출력은 DDRB를 사용한다.

- 각 비트가 1인 경우 해당 핀은 출력, 0인 경우 입력
- 예를 들면 DDRA = 0xFF인 경우, 포트 A의 8개 핀 모두 출력으로 사용함
- 예를 들면 DDRD = 0xF8인 경우, 포트 D의 0, 1, 2번 핀은 입력, 나머지는 출력

Bit	7	6	5	4	3	2	1	0	
	DDA7	DDA6	DDA5	DDA4	DDA3	DDA2	DDA1	DDA0	DDRA
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

출처: ATmega128 manual(http://www.atmel.com/images/doc2467.pdf). 2016. 10. 3. 스크린샷.
[그림 2-15] DDR 레지스터

출력으로 정해진 각 핀에 출력값을 설정하기 위한 레지스터는 [그림 2-16]의 PORT이다. DDR과 마찬가지로 A 포트의 출력값은 PORTA를 사용하고 B 포트의 출력값은 PORTB를 사용한다.

- 각 비트가 1인 경우, 해당 핀은 HIGH 상태(5 V 또는 3.3 V), 0인 경우 LOW 상태(0 V)
- 예를 들면 PORTA = 0xAA인 경우, 포트 A의 7, 5, 3, 1번 핀은 HIGH를, 나머지는 LOW를 출력함

Bit	7	6	5	4	3	2	1	0	
	PORTA7	PORTA6	PORTA5	PORTA4	PORTA3	PORTA2	PORTA1	PORTA0	PORTA
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

출처: ATmega128 manual(http://www.atmel.com/images/doc2467.pdf). 2016. 10. 3. 스크린샷.
[그림 2-16] PORT 레지스터

입력으로 정해진 각 핀의 입력값은 [그림 2-17]의 PIN 레지스터를 확인한다. DDR과 마찬가지로 A 포트의 입력값은 PINA를 사용하고 B 포트의 입력값은 PINB를 사용한다.

- 각 비트가 1인 경우, 해당 핀은 HIGH 상태(5 V 또는 3.3 V), 0인 경우 LOW 상태(0 V)
- 예를 들면 PIND = 0x05인 경우, 포트 D의 입력 상태는 2번, 0번 핀은 HIGH, 1번 핀은 LOW, 나머지는 관련 없음(DDRD = 0xF8의 경우).

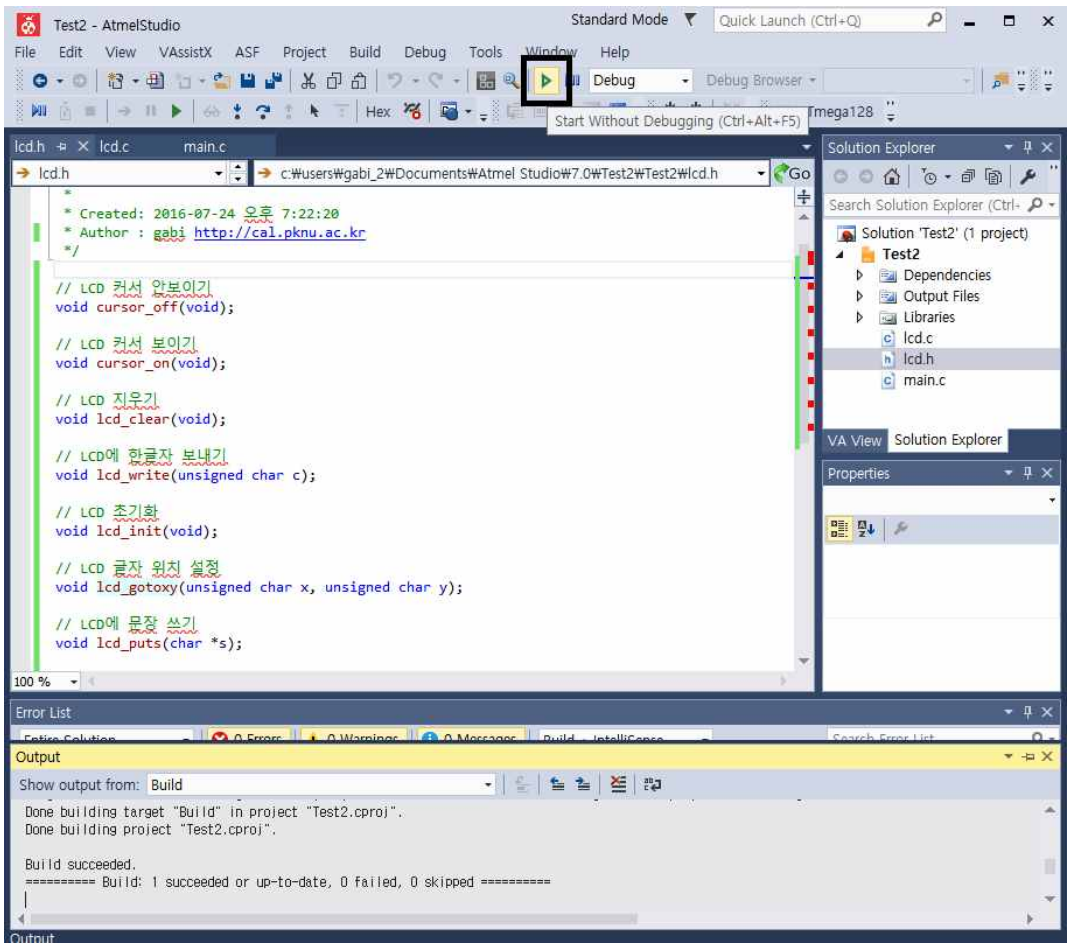
Bit	7	6	5	4	3	2	1	0	
	PINA7	PINA6	PINA5	PINA4	PINA3	PINA2	PINA1	PINA0	PINA
Read/Write	R	R	R	R	R	R	R	R	
Initial Value	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	

출처: ATmega128 manual(http://www.atmel.com/images/doc2467.pdf). 2016. 10. 3. 스크린샷.
[그림 2-17] PIN 레지스터

Test2 모듈은 아래 그림과 같이 3개의 소스 파일로 이루어져 있다.

- main.c
- lcd.c
- lcd.h

Build > Build Solution(F7)을 실행하여 프로그램에 오류가 없음을 확인한다. [그림 2-18]을 참고하여 Debug > Start Without Debugging(Ctrl-Alt-F5)를 이용하여 MCU 키트에 펌웨어를 다운로드한 후 실행한다.



[그림 2-18] 로봇 MCU 펌웨어 컴파일 및 다운로드

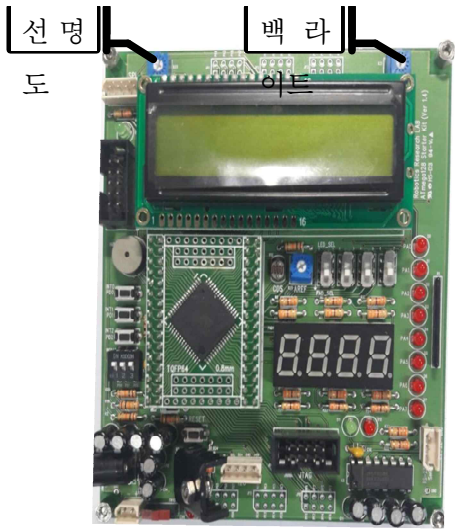
- 버튼 1은 인터럽트로 동작하며, 눌렀다 떼는 순간 LCD에 “>>Button1 ISR” 이라는 문자열을 보여준다.
- 버튼 2는 인터럽트로 동작하며, 눌렀다 떼는 순간 LCD에 “..Button2 ISR” 이라는 문자열을 보여준다.
- 버튼 3은 폴링으로 동작하며 LCD에 “ATmega128 Go!” 라는 문자열을 보여준다.

스타터키트에 전압이 인가되면 LCD에 글자가 나타난다. 만약 LCD에 아무것도 나타나지 않으면 [그림 2-19]의 화면 밝기 조절용 가변 저항과 백라이트 조절용 가변 저항을 좌우로 돌려 LCD의 밝기를 조절한다.

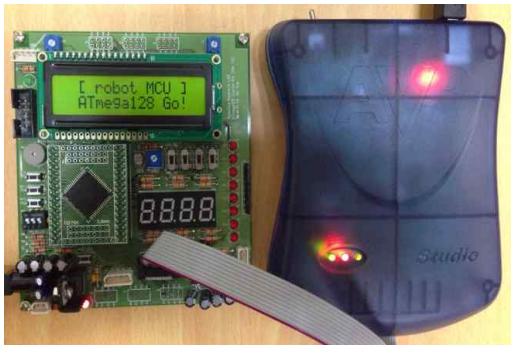
[그림 2-20]은 로봇 MCU의 펌웨어 실행 장면을 나타낸다. 초기 상태에서 버튼 1을 누르면, 스위치가 떨어지는 순간 인터럽트 INT0의 인터럽트 서비스 루틴이 실행된다. 버튼은 누르는 순간 1에서 0으로 출력이 변화하고, 떼는 순간 0에서 1로 변하기 때문에 로봇 MCU에서 상승 에지로 설정된 인터럽트는 손을 떼는 순간에 단 한 번 실행된다.

버튼 2를 누르면 스위치가 떨어지는 순간 인터럽트 INT1의 인터럽트 서비스 루틴이 실행된다. 버튼은 누르는 순간 1에서 0으로 출력이 변화하고, 떼는 순간 0에서 1로 변하기 때문에 로봇 MCU에서 상승 에지로 설정된 인터럽트는 손을 떼는 순간에 단 한 번 실행된다.

버튼 3은 인터럽트가 아니라 폴링으로 동작되며 초기 상태로 복귀한다. 메인 루프의 폴링으로 인하여 버튼을 누르고 있는 동안은 계속하여 초기화 동작을 반복 실행하게 된다.



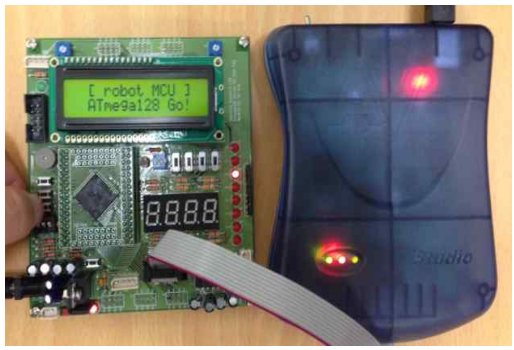
[그림 2-19] CdS 센서와 센서 감도 조절을 위한 가변 저항의 위치



(1) 초기 상태

(2) 1번 버튼, ISR 실행

(3) 2번 버튼, ISR 실행



(4) 3번 버튼, 폴링, 초기화

[그림 2-20] 실행 예

3 통신을 통하여 연결되는 로봇 센서에 대응하기 위하여 링버퍼를 구현한다.

1. 버튼 1: 인터럽트 방식에 의하여 스위치 입력을 확인한다. 링버퍼에 가상의 데이터가 한 글자 저장된다.
2. 버튼 2: 인터럽트 방식에 의하여 스위치 입력을 확인한다. 링버퍼에 저장된 데이터를 한 글자 읽어 들인다.
3. 버튼 3: 폴링 방식에 의하여 스위치 입력을 확인한다.

lcd.c와 lcd.h는 그대로 사용한다. 링버퍼의 구현을 위하여 [그림 2-21]의 ringBuffer.h를 입력한다.

```
/*
 * ringBuffer.h
 *
 * Created: 2016-07-24 오후 8:41:23
 * Author : gabi http://cal.pknu.ac.kr
 */
#ifndef __RINGBUFFER__
```

```
#define __RINGBUFFER__

#define BUFFER_SIZE 10

void init_Buffer();
int getBufferNumData();
int saveBufferData(char ch);
int getBufferData(int numData, char *buf);
int showBuffer(char *str);
int isBufferFull();

#endif
```

[그림 2-21] Starter kit의 예제 프로그램(ringBuffer.h) 입력

ringBuffer.c의 내용을 [그림 2-22]와 같이 프로그램한다.

```
/*
 * ringBuffer.c
 *
 * Created: 2016-07-24 오후 8:31:54
 * Author : gabi http://cal.pknu.ac.kr
 */
#include "ringBuffer.h"

char buffer[BUFFER_SIZE];
int pHead=0;
int pTail = 0;

// 링버퍼 초기화
void init_Buffer()
{
        pHead = 0;
        pTail = 0;
}

// 버퍼안에 저장된 데이터의 갯수
int getBufferNumData()
{
        return (pHead - pTail + BUFFER_SIZE)%BUFFER_SIZE;
}

// 버퍼가 꽉 찼는가?
int isBufferFull()
{
        if (getBufferNumData() == BUFFER_SIZE-1) return 1; // 꽉 찼음
        return 0;  // 다 차지 않았음
}
```

```
// 버퍼에 데이터 한개를 저장
int saveBufferData(char ch)
{
	if (isBufferFull()) return -1;

	buffer[pHead] = ch;
	pHead = (pHead + 1)%BUFFER_SIZE;
	return 0;
}

// 버퍼에서 데이터를 빼냄
int getBufferData(int numData, char *buf)
{
	int i = 0;
	int numGetData = 0;

	for (i=0; i<numData; i++){
		if (getBufferNumData()!=0){
			buf[numGetData++] = buffer[pTail];
			pTail = (pTail + 1)%BUFFER_SIZE;
		}
	}

	buf[numGetData] = 0;

	return numGetData;
}

// 버퍼안의 내용 보기
int showBuffer(char *str)
{
	int i=0;
	int pData=pTail;

	while (pData != pHead){
		str[i++] = buffer[pData];
		pData = (pData + 1)%BUFFER_SIZE;
	}

	str[i] = 0;
	return i;
}
```

[그림 2-22] Starter kit의 예제 프로그램(ringBuffer.c) 입력

Test3.c의 내용을 [그림 2-23]과 같이 입력한다.

```
/*
 * Test3.c
```

```
 *
 * Created: 2016-07-24 오후 7:15:19
 * Author : gabi http://cal.pknu.ac.kr
 */

#define F_CPU       16000000UL          // 16M Hz
#include <avr/io.h>
#include <util/delay.h>
#include <avr/interrupt.h>
#include "lcd.h"
#include "ringBuffer.h"

// 입출력 포트 초기화
void ioport_set(void)
{
        DDRA = 0xFF;    // PORT A는 8핀 모두 출력(LED 0~7)      (0b 1111 1111)
        DDRD = 0xF8;    // PORT D는 0~3핀은 입력 (Button 0~2)  (0b 1111 1000)

        DDRC = 0xFF;    // PORT C는 8핀 모두 출력(LCD)
        DDRE = 0xE0;    // PORT E는 5~7핀이 출력 (LCD)
}

// 인터럽트 초기화 서브루틴
void init_interrupt(void)
{
        EIMSK = 0x03;   // INT0, 1 활성화, (0b 0000 0011)
        EICRA = 0x0F;   // INT0, 1 상승 에지 선택, (0b 0000 1111)

        sei();                          // 인터럽트 가능
}

// INT0 외부 인터럽트 루틴
ISR(INT0_vect)
{
        char bufString[BUFFER_SIZE] = "Buf Full";
        if (saveBufferData('@') != -1) showBuffer(bufString);

        lcd_clear();
        lcd_gotoxy(0,1); lcd_puts(bufString);
        lcd_gotoxy(0,2); lcd_puts(">>Button1 ISR          ");  // 20자
}

// INT1 외부 인터럽트 루틴
ISR(INT1_vect)
{
        char ch;
        int numData;
        char bufString[BUFFER_SIZE];

        numData = getBufferData(&ch);
        showBuffer(bufString);
```

```
        lcd_clear();
        lcd_gotoxy(0,1); lcd_puts(bufString);
        lcd_gotoxy(0,2); lcd_puts("..Button2 ISR          "); // 20자
}

// 메인 루틴
int main(void)
{
        ioport_set();
        init_interrupt();

        init_Buffer();
        lcd_init();

        lcd_gotoxy(0,1); lcd_puts("   [ robot MCU ]   ");
        lcd_gotoxy(0,2); lcd_puts("   ATmega128 Go! ");

        while(1){
                PORTA = PIND | 0xF8;
                //PORT D에 연결된 버튼에 따라 PORT A에 연결된 LED를 켬
                if (~PIND & 0b00000100){
                        lcd_gotoxy(0,2); lcd_puts("   ATmega128 Go! ");
                }
        }
}
```

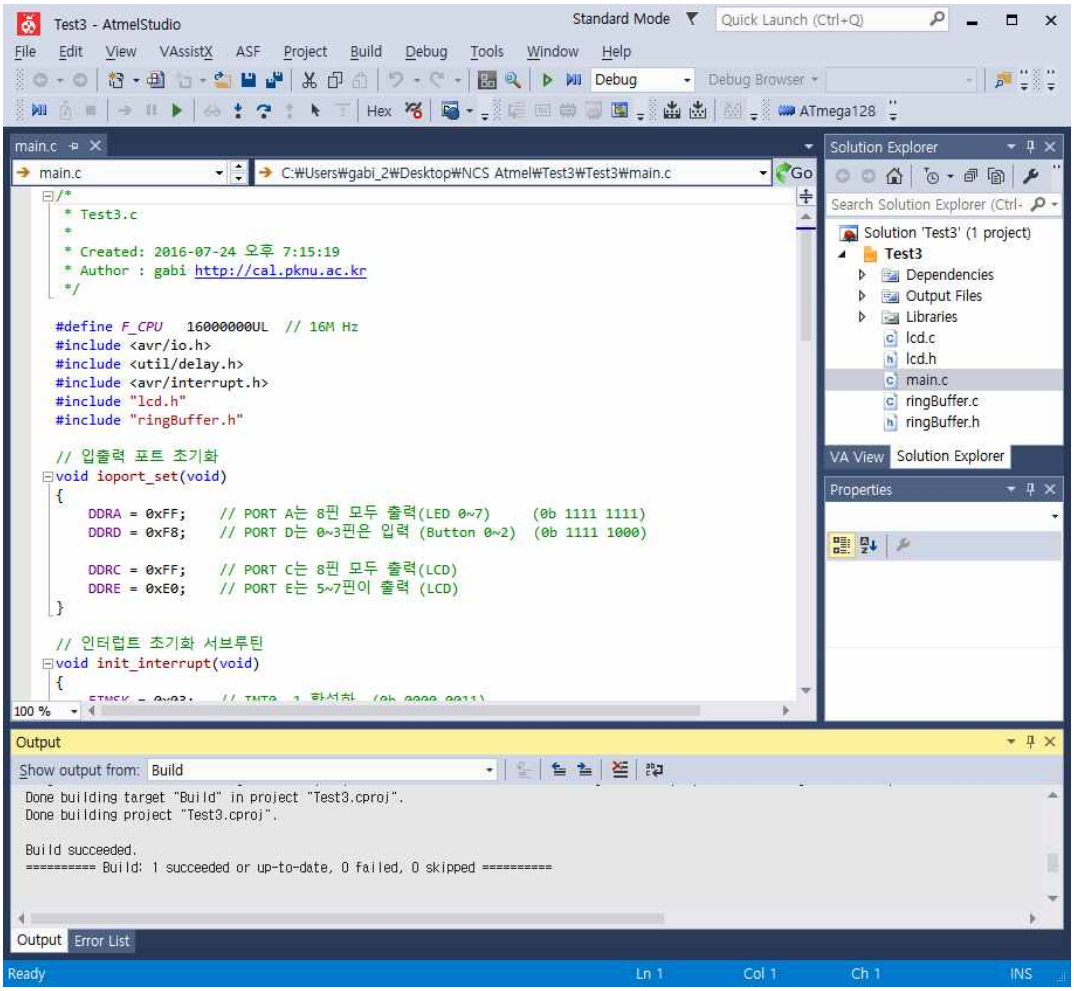
[그림 2-23] Starter kit의 예제 프로그램(Test3.c) 입력

Test3 모듈은 [그림 2-24]와 같이 5개의 소스 파일로 이루어져 있다.

- main.c
- lcd.c
- lcd.h
- ringBuffer.c
- ringBuffer.h

Build > Build Solution(F7)을 실행하여 프로그램에 오류가 없음을 확인한다. Debug > Start Without Debugging(Ctrl-Alt-F5)를 이용하여 MCU 키트에 펌웨어를 다운로드 후 실행한다.

- 버튼 1은 인터럽트로 동작하며 눌렀다 떼는 순간 링버퍼에 가상의 데이터('@')가 저장된다. 링버퍼의 내용을 LCD에 표시한다.
- 버튼 2는 인터럽트로 동작하며 눌렀다 떼는 순간, 링버퍼에서 한 개의 데이터를 읽어들인다. 링버퍼의 내용을 LCD에 표시한다.
- 버튼 3은 폴링으로 동작하며, 눌러져있는 동안, 초기화 동작을 반복하여 실행한다.



[그림 2-24] 로봇 MCU 펌웨어 컴파일 및 다운로드

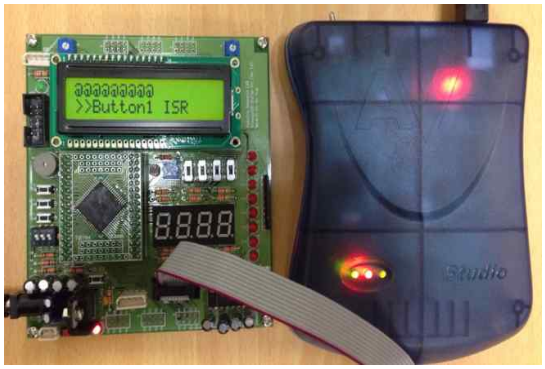
로봇 MCU의 실행 예는 [그림 2-25]와 같다. 초기 상태에서 버튼 1을 누르면, 스위치가 떨어지는 순간 인터럽트 INT0의 인터럽트 서비스 루틴이 한 번 실행된다. 이때 인터럽트 서비스 루틴은 가상의 데이터 '@'를 링버퍼에 저장한다. 링버퍼의 동작을 LCD에서 보여주기 위하여 지정된 버퍼 사이즈(BUFFER_SIZE = 10)로 인하여 최대 9개의 @를 링버퍼에 저장할 수 있다. 10개째의 버튼이 눌러지면 "Buffer Full" 메시지를 나타내고 더 이상의 데이터를 저장할 수 없다.

버튼 2를 누르면 스위치가 떨어지는 순간 인터럽트 INT1의 인터럽트 서비스 루틴이 한 번 실행된다. 이때 인터럽트 서비스 루틴은 링버퍼로부터 하나의 데이터를 가져온다.

버튼 1과 버튼 2의 동작이 연달아 실행될 때, 버튼 2의 동작이 충분히 빠르다는 가정 하에서는 버튼 1로 인한 통신 데이터의 수신 가능한 길이에는 제한이 없게 된다.

버튼 3은 인터럽트가 아니라 폴링으로 동작되며 "ATmega128 Go!" 라는 메시지를 LCD에 나타낸다. 메인 루프의 폴링으로 인하여 버튼을 누르고 있는 동안은 계속하여 초기화 동작을 반복하여 실행한다.

(1) 초기 상태



(2) 1번 버튼으로 데이터 저장

(3) 링버퍼가 가득 참

(4) 2번 버튼으로 데이터를 빼냄

[그림 2-25] 링버퍼를 사용한 데이터의 저장과 인출 예

수행 tip

- Atmel Studio 7.0 버전에서는 파일 경로와 이름에 한글이 있으면 펌웨어를 다운로드할 수 없으므로 주의하여야 한다.

학습 3 로봇 센서 드라이버 구현하기

3-1. 로봇 센서 드라이버 구현

학습 목표

- 로봇 센서 요구 사항을 만족하는 센싱 알고리즘을 작성할 수 있다.
- 센싱 알고리즘에 기반을 둔 드라이버를 설계할 수 있다.
- 로봇 센서 인터페이스를 위한 함수를 코딩하고 동작 확인을 할 수 있다.

필요 지식 /

1 C 언어와 로봇 MCU

로봇 MCU의 펌웨어를 제작하기 위하여 가장 널리 쓰이는 컴퓨터 언어는 C 언어이다. 예전에는 소스 코드의 길이와 실행 속도에서 유리한 어셈블러(assembler)가 펌웨어 제작 도구로 쓰였으나, 현재는 소스 코드 최적화 기능을 가진 컴파일러의 발전으로 인하여 알아보기 힘든 어셈블러를 사용할 필요가 없어졌다.

로봇 MCU의 펌웨어를 기술하는 데에 있어, 유용하게 쓰이는 표현들을 알아두기로 한다. [그림 3-1]의 소스 코드는 Atmel Studio에서 예제 코드로 제공하는 mega_gpio_example.c의 일부를 나타낸 것이다. 처음 커멘트 부분은 라이선스 구문으로 인용 시에 반드시 포함해야 하는 부분이다.

```
/**
 * \file
 *
 * \brief megaAVR STK600 GPIO Example
 *
 * Copyright (c) 2014-2015 Atmel Corporation. All rights reserved.
 *
 * \asf_license_start
 *
 * \page License
 *
```

```
 * Redistribution and use in source and binary forms, with or without
 * modification, are permitted provided that the following conditions are met:
 *
 * 1. Redistributions of source code must retain the above copyright notice,
 *     this list of conditions and the following disclaimer.
 *
 * 2. Redistributions in binary form must reproduce the above copyright notice,
 *     this list of conditions and the following disclaimer in the documentation
 *     and/or other materials provided with the distribution.
 *
 * 3. The name of Atmel may not be used to endorse or promote products derived
 *     from this software without specific prior written permission.
 *
 * 4. This software may only be redistributed and used in connection with an
 *     Atmel microcontroller product.
 *
 * THIS SOFTWARE IS PROVIDED BY ATMEL "AS IS" AND ANY EXPRESS OR IMPLIED
 * WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF
 * MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NON-INFRINGEMENT ARE
 * EXPRESSLY AND SPECIFICALLY DISCLAIMED. IN NO EVENT SHALL ATMEL BE LIABLE FOR
 * ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL
 * DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS
 * OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION)
 * HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT,
 * STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN
 * ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE
 * POSSIBILITY OF SUCH DAMAGE.
 *
 * \asf_license_stop
 *
 */

... 중략 ...

/**
 * \brief Pin change interrupt handler for PB0
 *
 * Create breakpoint here and then run. Press SW0 in order to test
 * this.
 */
ISR(EXAMPLE_PCINT_vect)
{
        /* Toggle PD0 (LED0)
         *
         * Note that it will toggle both on button press and release.
         */
        PIND = (1 << PIND0);
}

.. 중략 ...

int main(void)
{

.. 중략 ...

        /* It's also possible to control a single or a set of pins on a port.
         *
```

```
             * Setting pin PD0 high in order to turn off LED0.
             */
            PORTD |= (1 << PORTD0);

            // Setting pin PD1 low in order to turn on LED1.
            PORTD &= ~(1 << PORTD1);

            // Toggle both pin PD0 and PD1, which toggles LED0 and LED1.
#if defined(__AVR_ATmega16__) || defined(__AVR_ATmega32__) \
                    || defined(__AVR_ATmega64__) || defined(__AVR_ATmega128__)
            /* For older megaAVR devices read-modify-write PORT register.
             * This isn't safe for interrupts.
             */
            PORTD ^= (1 << PIND0) | (1 << PIND1);
#else
            // Use PIN register to toggle on newer megaAVR devices
            PIND = (1 << PIND0) | (1 << PIND1);
#endif

.. 생략 ...
```

[그림 3-1] mega_gpio_example.c: megaAVR GPIO example application

하나의 포트가 8개의 입출력 핀을 가지고 있는 8비트 로봇 MCU의 경우, 연산은 8비트를 최소 단위로 하여 실행된다. 16비트 로봇 MCU의 경우에는 16비트를 최소 단위로 하여 한번에 16비트씩 실행된다. 그러므로 하나의 포트 내에 포함된 각 비트의 비트 연산이 필요하다.

1. 비트 연산자(&, ¦)의 연산 방법

'&' 와 '¦' 는 각각 비트별 AND와 OR 연산을 의미한다. 8비트를 기준으로 다음과 같이 8비트 데이터가 있다고 가정한다.

num1 = 0x14 (즉, 0b 0001 0100)
num2 = 0x2C (즉, 0b 0010 1100)

이때 두 데이터의 '&' 연산과 '¦' 연산의 결과는 다음과 같다. 연산은 비트별로 이루어졌음을 확인한다.

num1 & num2 = 0x04 (즉, 0b 0000 0100)
num1 ¦ num2 = 0x3C (즉, 0b 0011 1100)

2. 비트 연산자(<<, >>)의 연산 방법

'<<' 와 '>>' 은 각각 좌우로 이동하는 shift 연산을 의미한다. '<<' 의 경우 한 칸

이동할 때마다 원래 데이터의 2배가 되고 '>>' 의 경우 한 칸 이동할 때마다 원래 데이터의 반이 된다는 의미도 가지고 있다. 8비트를 기준으로 다음과 같이 8비트 데이터가 있다고 가정한다.

num1 = 0x14 (즉, 0b 0001 0100)
num2 = 0x2C (즉, 0b 0010 1100)

이때 '<<' 과 '>>' 의 연산 예는 다음과 같다. 연산자 뒤의 숫자는 shift 횟수를 나타낸다.

num1 << 1 = 0x28 (즉, 0b 0010 1000)
num2 >> 2 = 0x0B (즉, 0b 0000 1011)

3. 비트 연산자(^)의 연산 방법

'^' 은 비트별 XOR 연산을 의미한다. 8비트를 기준으로 다음과 같이 8비트 데이터가 있다고 가정한다.

num1 = 0x14 (즉, 0b 0001 0100)
num2 = 0x2C (즉, 0b 0010 1100)

이때 두 데이터의 '^' 연산의 결과는 다음과 같다. 연산은 비트별로 이루어졌음을 확인한다.

num1 ^ num2 = 0x38 (즉, 0b 0011 1000)

4. 비트 연산자(~)의 연산 방법

'~' 은 비트별 NOT 연산을 의미한다. 8비트를 기준으로 다음과 같이 8비트 데이터가 있다고 가정한다.

num1 = 0x14 (즉, 0b 0001 0100)

이때 데이터의 '^' 연산의 결과는 다음과 같다. 연산은 비트별로 이루어졌음을 확인한다.

~num1 = 0xEB (즉, 0b 1110 1011)

5. 비트 연산자의 활용

비트 연산자를 활용한 프로그램의 예를 들어 보도록 하자. 예제 코드 내부에 다음과 같은 문장이 있다. 여기에서 PIND0, PIND1, ... , PIND7는 헤더 파일에 각각 0, 1, ..., 7로 정의되어 있다.

```
PIND = (1 << PIND0);
```

먼저 (1 << 0)은 1 (즉, 0b 0000 0001)을 0번 좌로 이동한다는 뜻으로 0번 핀의 입력을 1 (high) 상태로 설정한다는 의미이다.
같은 과정을 거쳐 다음의 문장을 해석해보자.

```
PIND = (1 << PIND0) | (1 << PIND1);
```

우변의 의미는 0번 좌로 이동시킨 '0b 0000 0001'과 1번 좌로 이동시킨 '0b 0000 0010'을 '|' (즉, OR) 연산하였으므로, 0번과 1번 비트가 각각 1인 '0b 0000 0011'이고, 이 값을 D 포트의 입력값으로 설정한다는 의미이다.

```
PIND = (1 << PIND3) | (0 << PIND2) | (1 << PIND1) | (0 << PIND0);
```

위 문장은 D 포트의 3번 핀을 1로, 2번 핀을 0으로, 1번 핀을 1로, 0번 핀을 0으로 만든다는 의미이다.

이제 아래의 세 문장을 예로 들어보자.

```
PORTD ^= (1 << PIND0) | (1 << PIND1);
PORTD |= (1 << PORTD0);
PORTD &= ~(1 << PORTD1);
```

첫 번째는, 0번과 1번 비트가 1인 '0b 0000 0011'이 이전의 D 포트 출력값과 '^' (즉, XOR) 연산을 실행한다. 예를 들어 PORTD의 이전값이 '0b 1010 0101'이었다고 가정하면,

```
0b10100101 ^ 0b00000011 = 0b10100110
```

으로 D 포트의 0번과 1번 핀의 값만 반전된 ‘0b 1010 0110’이 된다.

계속하여 두 번째 연산을 실행하면 D 포트에 0번 비트를 1인 상태로 기존의 D 포트 출력 값에 더하여 ‘0b 1010 0111’을 출력한다. 즉, 지정된 비트를 1로 만들기 위한 명령이다.

여기에 세 번째 연산을 실행하면

0b10100111 & ~(0b00000010) = 0b10100111 & 0b11111101 = 0b10100101

으로, D 포트의 1번 비트를 0으로 바꾼 ‘0b 1010 0101’을 출력한다. 즉, 지정된 비트를 0으로 만들기 위한 명령이다.

수행 내용 / 로봇 센서 드라이버 구현하기

재료・자료

- 센서 데이터시트
- 전기·전자 부품 데이터시트
- 전기·전자 부품 어플리케이션 노트
- RS-232C, RS-485, 이더넷, USB 등의 규격서

기기(장비・공구)

- 컴퓨터, 프린터
- 펌웨어 개발환경(http://www.atmel.com/tools/atmelstudio.aspx)에서 Atmel Studio 최신 버전 다운로드
- ATmega128 스타터키트(http://www.coenfirm.co.kr/bbs/board.php?bo_table=prod4&wr_id=7)
- ISP 또는 JTAG 장비(윈도우 및 Atmel Studio 최신 버전과 호환성 확인 필요) http://www.atmel.com/tools/AVRJTAGICEMKII.aspx?tab=overview
- 문서 작성용 소프트웨어

안전 · 유의 사항

- 펌웨어 개발환경으로 사용되는 Atmel Studio의 경우, 파일의 경로나 이름에 한글이 허용되지 않음을 유의한다.
- 컴퓨터 환경에 따라 일부 ISP와 JTAG 에뮬레이터가 지원되지 않을 수 있으므로, 지원 여부를 반드시 확인하여야 한다.
- 실습 장비에 대한 전원 투입 시, 정확한 전압과 극성을 인가하여야 한다.

수행 순서

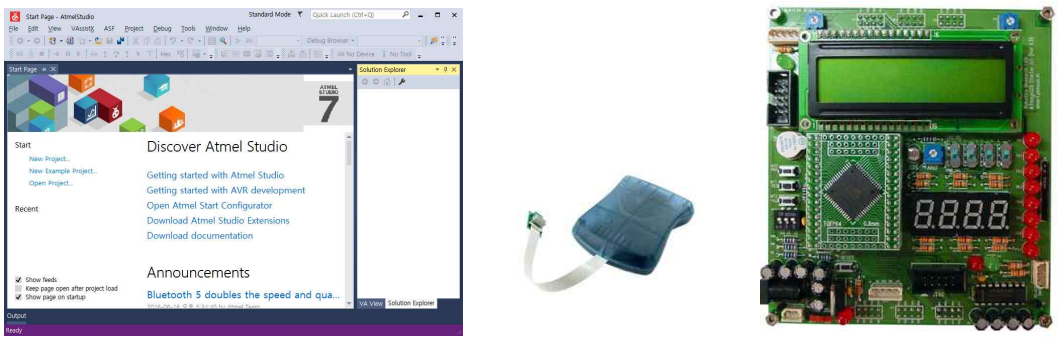
1 로봇 센서의 요구 사항을 만족하는 센싱 알고리즘을 작성하고 이에 맞는 드라이버를 설계한다.

로봇 센서 인터페이스에 따라 로봇 MCU의 입력 모듈을 구상하고, 이에 맞는 드라이버를 설계한다. 여기에서는 다음과 같은 상황을 가정한다.

1. 버튼 1: 인터럽트 방식에 의하여 스위치 입력을 확인한다. 이때 메인 루프에서 LCD에 로봇 주변의 조도를 표시한다.
2. 버튼 2: 인터럽트 방식에 의하여 스위치 입력을 확인한다. 이때 메인 루프에서 LCD에 로봇 주변의 습도 및 온도를 표시한다.
3. 버튼 3: 인터럽트 방식에 의하여 스위치 입력을 확인한다. 이때 메인 루프에서 LCD에 로봇의 위치(경도, 위도)를 표시한다.

2 선정된 로봇 센서에 맞는 아날로그 센서의 드라이버를 작성한다.

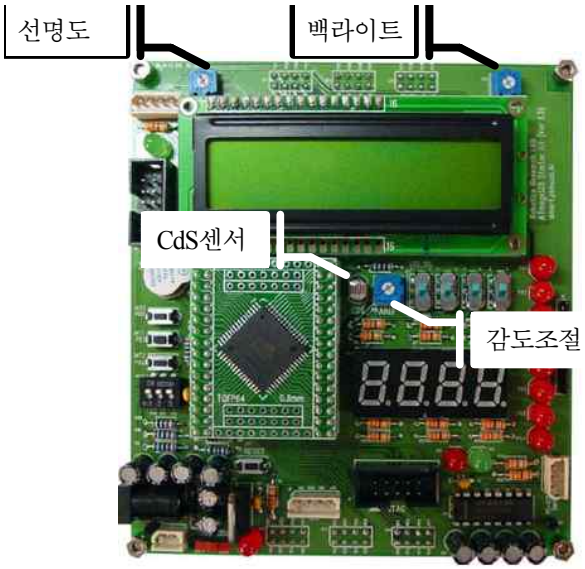
1. 학습 2의 연습에서와 같은 [그림 3-2]의 개발환경을 사용한다.



출처: JTAG(http://www.atmel.com/tools/AVRJTAGICEMKII.aspx?tab=overview). 2016. 10. 3. 스크린샷.
Coenfirm(http://www.coenfirm.co.kr/bbs/board.php?bo_table=prod4&wr_id=7). 2016. 10. 3. 스크린샷.
[그림 3-2] Atmel Studio, JTAG 에뮬레이터, ATmega128 스타터키트

2. 아날로그 출력을 가진 센서를 위한 드라이버 프로그램을 작성한다.

여기에서는 ATmega128 스타터키트의 F 포트 0번 핀에 연결되어 있는 CdS 조도 센서를 사용하기로 한다. [그림 3-3]을 참고한다.



[그림 3-3] Atmel Studio, JTAG 에뮬레이터, ATmega128 스타터키트

3. Atmel Studio에서 [New Project...]를 생성한다.

(1) GCC C Executable Project

(2) Name: Test4 (Atmel Studio 7.0 버전에서는 파일 경로와 이름에 한글이 있으면 안 됨)

(3) Device Selection: ATmega128

4. lcd.c, lcd.h, ringBuffer.c, ringBuffer.h는 이전의 것을 그대로 사용한다.

[그림 3-4]와 같이 main.c의 내용을 입력한다.

```
/*
 * Test4.c
 *
 * Created: 2016-07-24 오후 7:15:19
 * Author : gabi http://cal.pknu.ac.kr
 */

#define F_CPU       16000000UL          // 16M Hz
#include <avr/io.h>
#include <util/delay.h>
#include <avr/interrupt.h>
```

```
#include "lcd.h"
#include "ringBuffer.h"
#include "cds.h"

int buttonNum = -1;// 누른 버튼 번호

// 입출력 포트 초기화
void ioport_set(void)
{
	DDRA = 0xFF;	// PORT A는 8핀 모두 출력(LED 0~7)	(0b 1111 1111)
	DDRD = 0xF8;	// PORT D는 0~3핀은 입력 (Button 0~2)	(0b 1111 1000)

	DDRC = 0xFF;	// PORT C는 8핀 모두 출력(LCD)
	DDRE = 0xE0;	// PORT E는 5~7핀이 출력 (LCD)
}

// 인터럽트 초기화 서브루틴
void init_interrupt(void)
{
	// INT0, 1, 2 활성화, (0b 0000 0111)
	EIMSK=(0<<INT7)|(0<<INT6)|(0<<INT5)|(0<<INT4)|
	    (0<<INT3)|(1<<INT2)|(1<<INT1)|(1<<INT0);

	// INT0, 1, 2 상승 에지 선택, (0b 0011 1111)
	EICRA=(0<<ISC31)|(0<<ISC30)|(1<<ISC21)|(1<<ISC20)|
	                (1<<ISC11)|(1<<ISC10)|(1<<ISC01)|(1<<ISC00);

	sei();				// 인터럽트 가능
}

// INT0 외부 인터럽트 루틴
ISR(INT0_vect)
{
	buttonNum = 0;
}

// INT1 외부 인터럽트 루틴
ISR(INT1_vect)
{
	buttonNum = 1;
}

// INT2 외부 인터럽트 루틴
ISR(INT2_vect)
{
	buttonNum = 2;
}

// 메인 루틴
int main(void)
{
	char strCdS[20];
	char strDHT[20];
	char strGPS[20];
```

```
        ioport_set();
        init_interrupt();

        init_Buffer();
        init_CdS();
        lcd_init();

        lcd_gotoxy(0,1); lcd_puts("   [ robot MCU ]   ");
        lcd_gotoxy(0,2); lcd_puts("   ATmega128 Go!  ");

        read_CdS(strCdS);
        //read_DHT11(strDHT);
        //read_GPS(strGPS);

        switch(buttonNum){
                case 0: // CdS 조도 센서값을 표시함(ADC)
                        lcd_clear();
                        lcd_gotoxy(0,1); lcd_puts(strCdS);
                        lcd_gotoxy(0,2); lcd_puts(">>Analog Sensor");
                        break;
                case 1: // DHT 온습도 값을 표시함 (Digital)
                        lcd_clear();
                        lcd_gotoxy(0,1); lcd_puts(strDHT);
                        lcd_gotoxy(0,2); lcd_puts(">>Digital Sensor");
                        break;
                case 2: // GPS 위도, 경도를 표시함(시리얼 통신)
                        lcd_clear();
                        lcd_gotoxy(0,1); lcd_puts(strGPS);
                        lcd_gotoxy(0,2); lcd_puts(">>Serial Comm.");
                        break;
        }
}
```

[그림 3-4] Starter kit의 예제 프로그램(Test4.c) 입력

init_interrupt 함수 내부가 비트 연산자를 사용한 구문으로 변경되었음을 확인한다.

```
// INT0, 1, 2 활성화, (0b 0000 0111)
EIMSK = (0<<INT7)|(0<<INT6)|(0<<INT5)|(0<<INT4)|
        (0<<INT3)|(1<<INT2)|(1<<INT1)|(1<<INT0);

// INT0, 1, 2 상승 에지 선택, (0b 0011 1111)
EICRA = (0<<ISC31)|(0<<ISC30)|(1<<ISC21)|(1<<ISC20)|
        (1<<ISC11)|(1<<ISC10)|(1<<ISC01)|(1<<ISC00);
```

작성된 펌웨어는 푸시 스위치의 입력을 인터럽트로 받는다. 인터럽트 서비스 루틴은 가능한 짧고 빠르게 작성되어야 하므로 눌러진 스위치의 번호만을 저장한다. main 함수의

while 문 내부에서 저장된 스위치의 번호를 확인한 후, ADC를 사용하여 CdS 조도 센서의 값을 읽는다. 읽어온 값을 LCD에 표시한다. 이것은 로봇 MCU가 센서값 확인을 실행하는 동안에 수신될 지도 모를 시리얼 통신 데이터를 빠뜨리지 않기 위한 것이다.

DHT11 온습도 센서와 GPS 센서는 이 단계에서 구현하지 않는다.

[그림 3-5]와 같이 cds.h를 입력한다.

```
/*
 * cds.h
 *
 * Created: 2016-07-27 오후 5:36:26
 * Author : gabi http://cal.pknu.ac.kr
 */
#ifndef __CDS__
#define __CDS__

void init_CdS();
void read_CdS(char *strResult);

#endif
```

[그림 3-5] Starter kit의 예제 프로그램(cds.h) 입력

[그림 3-6]과 같이 cds.c의 내용을 입력한다.

```
/*
 * cds.c
 *
 * Created: 2016-07-27 오후 5:36:14
 * Author : gabi http://cal.pknu.ac.kr
 */
#define F_CPU       16000000UL          // 16M Hz
#include <avr/io.h>
#include <util/delay.h>
#include <avr/interrupt.h>
#include <stdio.h>

void init_CdS(void)
{
        //ADMUX 레지스터 설정
        //Bit(7-6) : 외부 AREF 단자로 입력된 전압을 사용
        //Bit5 : 데이터 우측 정렬
        //Bit(4-0) : 단일 전압 입력중에서 ADC0와 연결되어 있음
        //ADMUX=0b00000000;
        ADMUX=(0<<REFS1)|(0<<REFS0)|(0<<ADLAR)|(0<<MUX4)
                |(0<<MUX3)|(0<<MUX2)|(0<<MUX1)|(0<<MUX0);

        //ADCSRA 레지스터 설정
        //Bit7 : ADC Enable 비트 이므로 1로 설정
        //Bit6 : ADC 변환을 시작
        //Bit5 : Free Running 모드로 동작을 위하여 1로 설정
```

```
        //Bit(4-3) : 현재는 인터럽트 관련 비트는 사용하지 않으므로 0으로 설정
        //Bit(2-0) : 분주비를 128로 설정(111)
        //ADCSRA=0b11100111;
        ADCSRA=(1<<ADEN)|(1<<ADSC)|(1<<ADFR)|(0<<ADIF)
                |(0<<ADIE)|(1<<ADPS2)|(1<<ADPS1)|(1<<ADPS0);
}

void read_CdS(char *strResult)
{
        // Free Running 모드에서는 ADC값을 아무때나 읽어올 수 있음
        sprintf(strResult, "CdS: %d", ADC);
}
```

[그림 3-6] Starter kit의 예제 프로그램(cds.c) 입력

ADC를 사용하기 위한 ATmega128의 설정은 ADMUX와 ADCSRA 레지스터를 사용한다. 변환 결과값은 ADCH, ADCL 두 레지스터에 저장된다.

[그림 3-7]의 ADMUX(ADC Multiplexer) 레지스터는 다음과 같은 기능을 설정한다.

- Reference Selection을 위한 REFS1, REFS0가 00, 01, 11인 경우, 각각 AREF 핀 입력 전압, AVCC 핀 입력 전압, 내부 기준 전압을 기준으로 설정한다. 10인 경우는 사용하지 않는다.
- ADLAR(ADC left adjust result)이 1인 경우는 10비트로 변환된 값이 16비트(ADCH, ADCL로 명명된 2개의 8비트 레지스터) 저장소에 좌측 정렬, 0인 경우는 우측 정렬된다.
- MUX4 ~ MUX0의 값에 의하여 8개의 ADC 채널 중 하나를 선택한다. 단극성 입력과 차동 입력을 선택할 수 있고, 이득값도 선택할 수 있다.
- 기타 더 자세한 내용은 매뉴얼을 확인한다.

Bit	7	6	5	4	3	2	1	0	
	REFS1	REFS0	ADLAR	MUX4	MUX3	MUX2	MUX1	MUX0	ADMUX
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

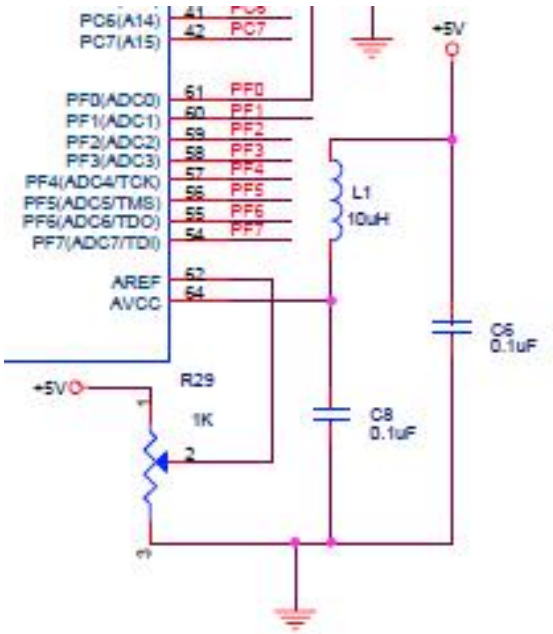
출처: ATmega128 manual(http://www.atmel.com/images/doc2467.pdf). 2016. 10. 3. 스크린샷.
[그림 3-7] ADMUX 레지스터

예를 들어 위 프로그램 내부의 설정을 확인해 보자.

```
ADMUX = (0<<REFS1)|(0<<REFS0)|(0<<ADLAR)|(0<<MUX4)
        |(0<<MUX3)|(0<<MUX2)|(0<<MUX1)|(0<<MUX0);
```

이 설정에 의하면 ADC 기준 전압으로 AREF 핀 입력 전압을 사용하는데, 스타터키트의 경우, AREF 핀에 입력되는 전압은 [그림 3-8]에서 보이는 것처럼 최대 5 V의 전압이 가변

저항에 의하여 분배되어 입력된다. AVCC는 노이즈 필터를 통하여 5 V에 연결되어 있다. ADLAR이 0이므로 우측 정렬이다. MUX4 ~ MUX0의 값이 00000인데, 이것은 ADC0 핀과 공통 접지를 사용하는 단극성 입력을 의미한다.



[그림 3-8] Atmel Studio, JTAG 에뮬레이터, ATmega128 스타터키트

[그림 3-9]의 ADCSRA(ADC control and status register A)는 ADC의 동작 방법에 대한 것을 설정한다.

- ADEN(ADC enable)은 1인 경우, ADC 사용이 가능하다.
- ADSC(ADC start conversion)을 1로 설정하면 ADC 변환이 시작된다. MCU가 ADC 변환을 완료하면 이 비트가 0으로 바뀐다.
- ADFR(ADC free running select)가 1로 설정된 경우, MCU는 ADC 변환이 한 번 실행되면 그 이후 계속하여 실행한다. 0으로 설정하면 멈춘다.
- ADIF(ADC interrupt flag)이 1인 경우, 해당 인터럽트 서비스 루틴이 한 번 실행된다.
- ADIE(ADC interrupt enable)이 1인 경우, ADC 변환 완료 인터럽트를 사용 가능하다.
- ADPS2~0(ADC prescaler select bits)는 클럭 분주비를 설정한다.
- 기타 더 자세한 내용은 매뉴얼을 확인한다.

Bit	7	6	5	4	3	2	1	0	
	ADEN	ADSC	ADFR	ADIF	ADIE	ADPS2	ADPS1	ADPS0	ADCSRA
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

출처: ATmega128 manual(http://www.atmel.com/images/doc2467.pdf). 2016. 10. 3. 스크린샷.
[그림 3-9] ADCSRA 레지스터

[그림 3-10]은 변환 결과값이 저장되는 ADCH, ADCL 레지스터의 내용을 보여준다.

Bit	15	14	13	12	11	10	9	8	
	–	–	–	–	–	–	ADC9	ADC8	ADCH
	ADC7	ADC6	ADC5	ADC4	ADC3	ADC2	ADC1	ADC0	ADCL
	7	6	5	4	3	2	1	0	

(1) ADLAR = 0, 우측 정렬

Bit	15	14	13	12	11	10	9	8	
	ADC9	ADC8	ADC7	ADC6	ADC5	ADC4	ADC3	ADC2	ADCH
	ADC1	ADC0	–	–	–	–	–	–	ADCL
	7	6	5	4	3	2	1	0	

(2) ADLAR = 1, 좌측 정렬

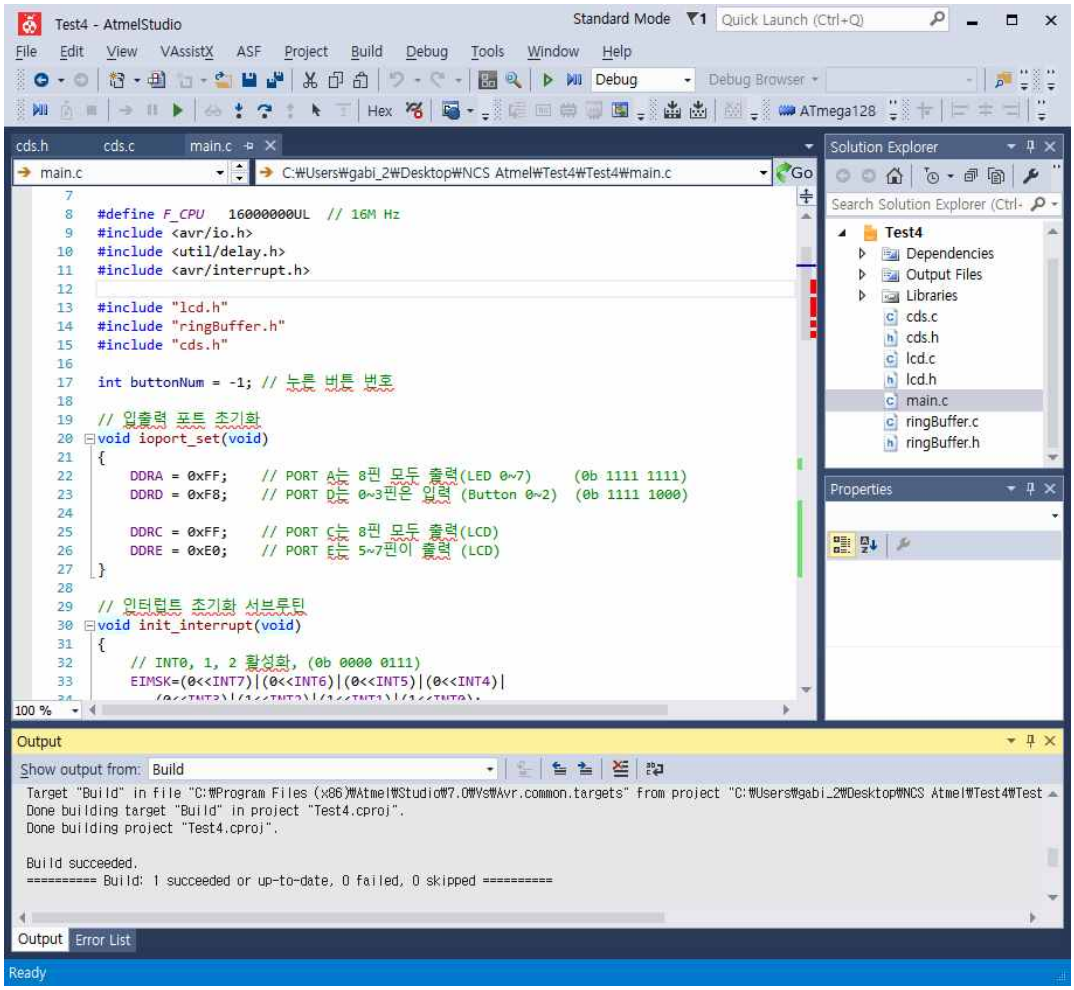
출처: ATmega128 manual(http://www.atmel.com/images/doc2467.pdf). 2016. 10. 3. 스크린샷.
[그림 3-10] ADC 레지스터

Test4 모듈은 [그림 3-11]과 같이 7개의 소스 파일로 이루어져 있다.

- main.c
- lcd.h
- lcd.c
- ringBuffer.h
- ringBuffer.c
- cds.h
- cds.c

Build > Build Solution(F7)을 실행하여 프로그램에 오류가 없음을 확인한다. Debug > Start Without Debugging(Ctrl-Alt-F5)를 이용하여 MCU 키트에 펌웨어를 다운로드한 후 실행한다.

- 버튼 1: 버튼을 누르면 LCD에 CdS 조도 센서값을 표시한다.
- 버튼 2: 버튼을 누르면 LCD에 DHT11 온습도 센서값을 표시한다. 현재 상태에서는 데이터값이 없다.
- 버튼 3: 버튼을 누르면 LCD에 로봇의 위치(경도, 위도)를 표시한다. 현재 상태에서는 데이터값이 없다.



[그림 3-11] 로봇 MCU 펌웨어 컴파일 및 다운로드

로봇 MCU의 실행 예는 [그림 3-12]와 같다.

- 초기 상태에서 버튼을 누르면, 스위치가 떨어지는 순간 해당 버트의 인터럽트 서비스 루틴이 실행된다. 버튼들은 누르는 순간 1에서 0으로 출력이 변화하고, 떼는 순간 0에서 1로 변하기 때문에 로봇 MCU에서 상승 에지로 설정된 인터럽트는 손을 떼는 순간에 단 한 번만 실행된다.
- 전압이 인가되면 LCD에 글자가 나타난다. 만약 LCD에 아무것도 나타나지 않으면 화면 밝기 조절용 가변 저항과 백라이트 조절용 가변 저항을 좌우로 돌려 LCD의 밝기를 조절한다.
- 버튼 1을 누르면 LCD에 조도 센서의 출력값이 그대로 표시된다. 만약 이 값이 1023으로 변하지 않고 멈추어 있다면, 센서 옆의 가변 저항을 좌우로 돌려 AREF 기준 전압을 변화시킨다.
- 버튼 2와 3을 누르면 DHT11 온습도 센서와 GPS 위치값을 표시하기 위하여 대기한다. 두 센서의 드라이버 프로그램은 이후 학습 내용에서 하나씩 구현한다.

(1) 초기 상태

(2) 버튼 1, 조도값 표시

(3) 버튼 2, 온습도 표시(미완성)

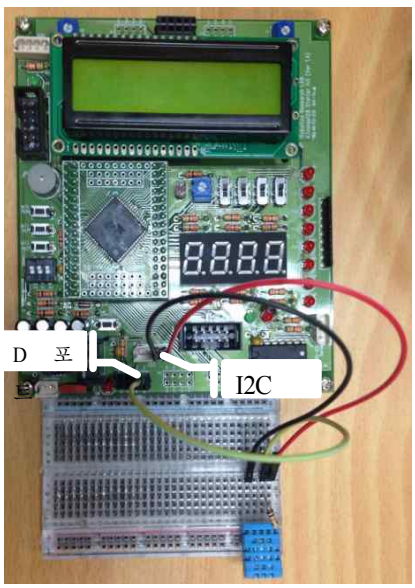
(4)버튼 3, 위치 표시(미완성)

[그림 3-12] 실행 예

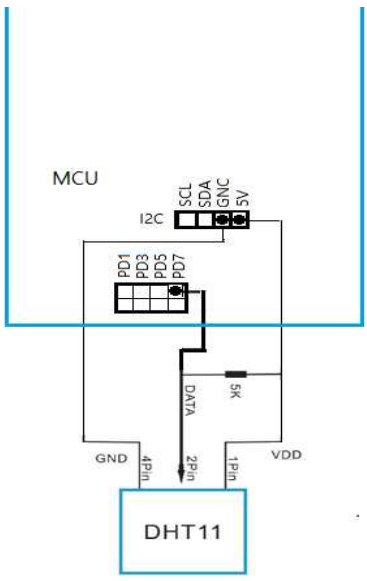
3 선정된 로봇 센서에 맞는 디지털 센서의 드라이버를 작성한다.

1. 디지털 출력을 가진 센서를 위한 드라이버 프로그램을 작성한다.

여기에서는 [그림 3-13]과 같이 DHT11 온습도 센서를 ATmega128 스타터키트의 B 포트에 연결하여 사용하기로 한다.



(1) 연결 예



(2) 연결도

[그림 3-13] DHT11과 스타터키트의 연결

스타터키트에는 DHT11 센서가 장착되어 있지 않으므로, 브레드보드를 사용하여 연결한다. 센서에 인가할 VDD와 GND는 스타터키트의 I2C 커넥터로부터 VDD와 GND를 연결하고 센서의 데이터 핀을 MCU의 D 포트 7번 입력 핀에 연결하였다.

2. Atmel Studio에서 [New Project...]를 생성한다.

- GCC C Executable Project
- Name: Test5 (Atmel Studio 7.0 버전에서는 파일 경로와 이름에 한글이 있으면 안 됨)
- Device Selection: ATmega128

[그림 3-14]와 같이 main.c의 내용을 입력한다. 이전 프로그램과 다른 점은 dht11.h을 포함시키고, 메인 루틴 내부 switch 위의 read_DHT11(strSensor) 부분을 활성화한 것이다. GPS 센서는 이 단계에서 구현하지 않는다.

```
/*
 * Test5.c
 *
 * Created: 2016-07-24 오후 7:15:19
 * Author : gabi http://cal.pknu.ac.kr
 */

#define F_CPU       16000000UL          // 16M Hz
#include <avr/io.h>
#include <util/delay.h>
#include <avr/interrupt.h>

#include "lcd.h"
#include "ringBuffer.h"
#include "cds.h"
#include "dht11.h"                      // read_DHT11 함수의 선언

... 생략(Test4.c와 동일) ...

                read_CdS(strCdS);
                read_DHT11(strDHT);
                //read_GPS(strGPS);

                switch(buttonNum){
                        case 0: // CdS 조도 센서값을 표시함(ADC)
                                lcd_clear();
                                lcd_gotoxy(0,1); lcd_puts(strCdS);
                                lcd_gotoxy(0,2); lcd_puts(">>Analog Sensor");
                                break;
                        case 1: // DHT 온습도 값을 표시함 (Digital)
                                lcd_clear();
                                lcd_gotoxy(0,1); lcd_puts(strDHT);
                                lcd_gotoxy(0,2); lcd_puts(">>Digital Sensor");
```

```
                                    break;
                            case 2: // GPS 위도, 경도를 표시함(시리얼 통신)
                                    lcd_clear();
                                    lcd_gotoxy(0,1); lcd_puts(strGPS);
                                    lcd_gotoxy(0,2); lcd_puts(">>Serial Comm.");
                                    break;
                    }

...후략...(Test4.c와 동일)
```

[그림 3-14] Starter kit의 예제 프로그램(Test5.c) 입력

[그림 3-15]와 같이 dht11.h를 입력한다.

```
/*
 * dht11.h
 *
 * Created: 2016-07-27 오후 4:34:41
 * Author : gabi http://cal.pknu.ac.kr
 */
#ifndef __DHT11__
#define __DHT11__

void read_DHT11(char *strResult);

#endif
```

[그림 3-15] Starter kit의 예제 프로그램(dht11.h) 입력

[그림 3-16]과 같이 dht11.c의 내용을 입력한다.

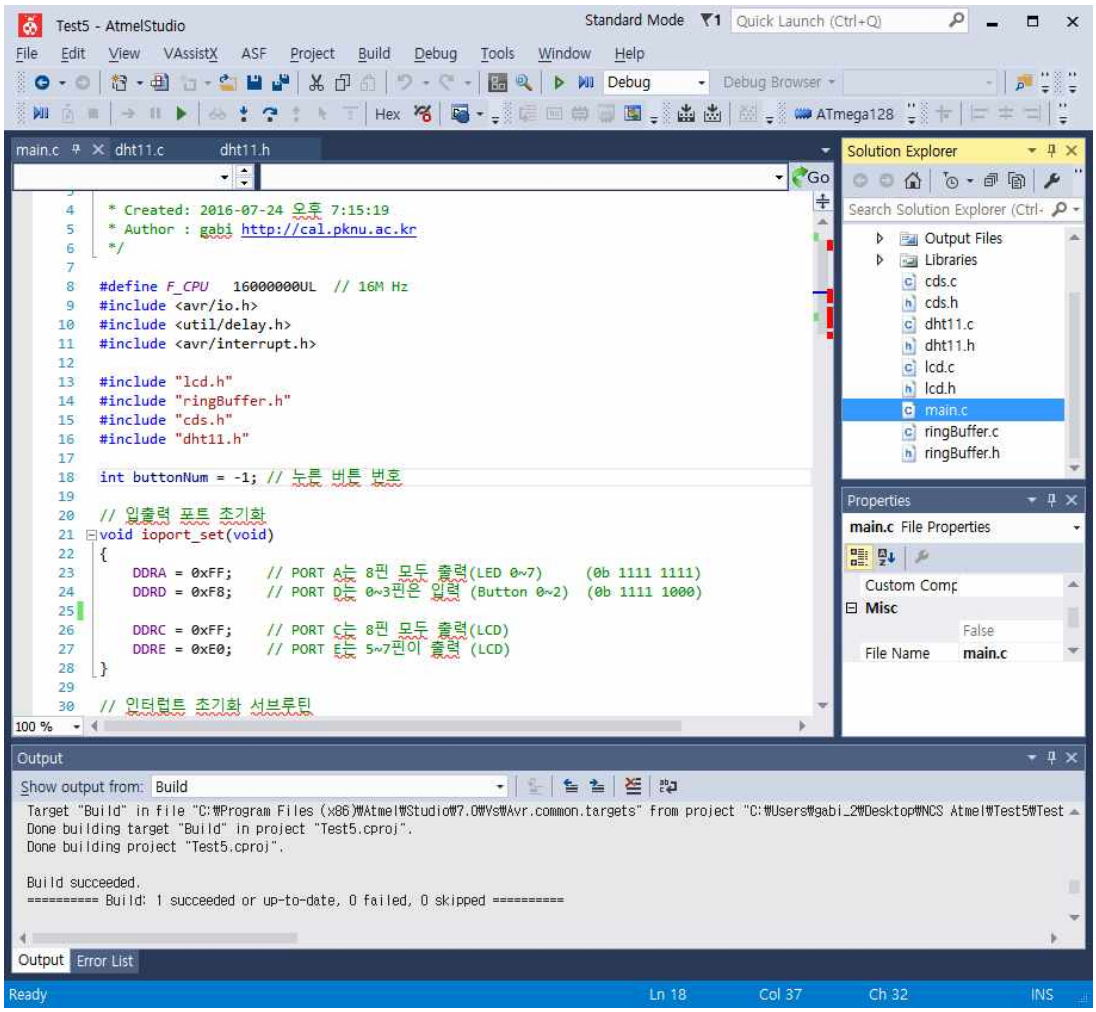
```
/*
 * dht11.c
 *
 * Created: 2016-07-24 오후 8:31:54
 * Author : gabi http://cal.pknu.ac.kr
 */
#define F_CPU       16000000UL          // 16M Hz
#include <avr/io.h>
#include <util/delay.h>
#include <avr/interrupt.h>
#include <stdio.h>

#include "dht11.h"
#define MAX_COUNT 85

void read_DHT11(char *strResult)
{
```

```
        char dhtVal[5] = {0, 0, 0, 0, 0};       // 습도.습도, 온도.온도, 체크섬
        char checkSum;

        unsigned char val;                      // 현재 신호
        unsigned char state = 1;                // 기준 상태

        int counter = 0;
        int j = 0, i;

        // MCU의 데이터 요청 출력
        DDRD |= (1<<PIND7);                     // D 포트 7번 핀을 출력으로 설정
        PORTD &= ~(1<<PIND7);                   // D 포트 7번 핀에 0 출력
        _delay_ms(18);
        PORTD |= (1<<PIND7);                    // D 포트 7번 핀에 1 출력
        _delay_us(40);

        // DHT 응답 입력
        DDRD &= ~(1<<PIND7);                    // D 포트 7번 핀을 입력으로 설정
        for(i = 0; i < MAX_COUNT; i++) {
                counter = 0;

                while((val = (PIND & 0b10000000)) == state) {
                        counter++;              // 센서 신호가 바뀌지 않으면 카운트 증가
                        _delay_us(1);
                        if(counter == 255) break;
                };

                state = val;                    // 바뀐 신호를 새 기준으로 설정
                if(counter == 255) break;

                // 처음 3개는 무시(0->1로 응답 시작, 그다음 0은 데이터 시작)
                if((i >= 4) && (i%2 == 0)) {
                        dhtVal[j/8] <<= 1;   // 1칸씩 왼쪽으로 이동, 즉 다음 비트 (8 비트)
                        if(counter > 16) dhtVal[j/8] |= 1;
                                                // 1로 유지되는 시간이 28us 보다 크면 1
                                                // trial and error로 16 이 적당함을 찾음
                        j++;
                }
        }

        // 체크섬 검사, 측정값 리턴
        checkSum = (dhtVal[0] + dhtVal[1] + dhtVal[2] + dhtVal[3]) & 0xFF;
        if((j >= 40) && (dhtVal[4] == checkSum)) {
                sprintf(strResult, "H%+2d.%d,T%2d.%d", dhtVal[0], dhtVal[1], dhtVal[2], dhtVal[3]);
                // 습도, 온도
        }
}
```

[그림 3-16] Starter kit의 예제 프로그램(dht11.c) 입력

Test5 모듈은 [그림 3-17]과 같이 9개의 소스 파일로 이루어져 있다.

- main.c
- lcd.h
- lcd.c
- ringBuffer.h
- ringBuffer.c
- cds.h
- cds.c
- dht11.h
- dht11.c



[그림 3-17] 로봇 MCU 펌웨어 컴파일 및 다운로드

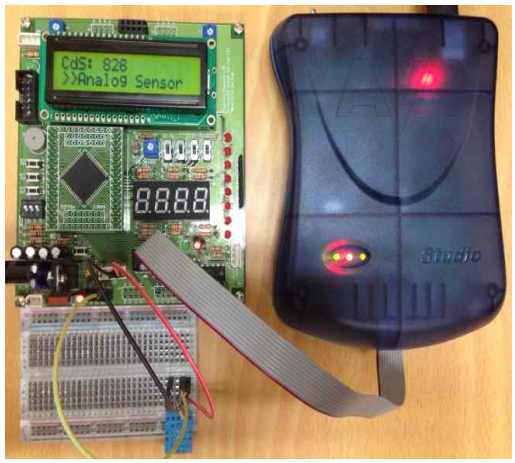
Build > Build Solution(F7)을 실행하여 프로그램에 오류가 없음을 확인한다. Debug > Start Without Debugging(Ctrl-Alt-F5)를 이용하여 MCU 키트에 펌웨어를 다운로드한 후 실행한다.

- 버튼 1: 버튼을 누르면 LCD에 CdS 조도 센서값을 표시한다.

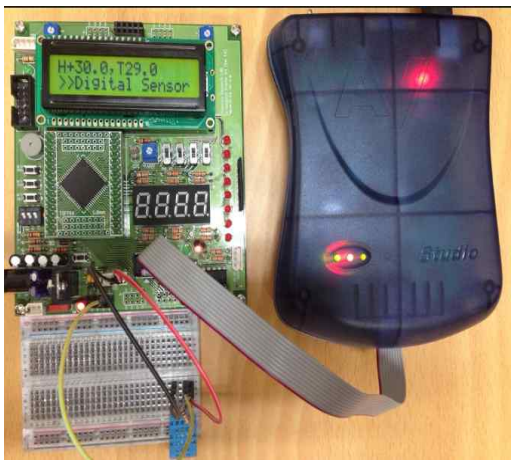
- 버튼 2: 버튼을 누르면 LCD에 DHT11 온습도 센서값을 표시한다.
- 버튼 3: 버튼을 누르면 LCD에 로봇의 위치(경도, 위도)를 표시한다. 현재 상태에서는 데이터값이 없다.

로봇 MCU의 실행 예는 [그림 3-18]과 같다.

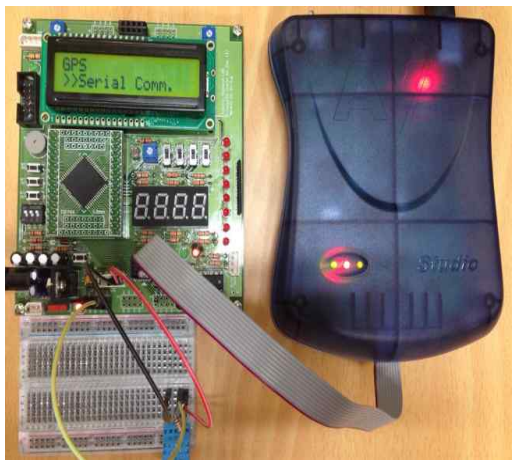
(1) 초기 상태



(2) 버튼 1, 조도값 표시



(3) 버튼 2, 온습도 표시



(4)버튼 3, 위치 표시(미완성)

[그림 3-18] 실행 예

- 초기 상태에서 버튼을 누르면, 스위치가 떨어지는 순간, 해당 버트의 인터럽트 서비스 루틴이 실행된다. 버튼들은 누르는 순간 1에서 0으로 출력이 변화하고, 떼는 순간 0에서 1로 변하기 때문에 로봇 MCU에서 상승 에지로 설정된 인터럽트는 손을 떼는 순간에 단 한 번 실행된다.
- 전압이 인가되면, LCD에 글자가 나타난다. 만약 LCD에 아무것도 나타나지 않으면 화면 밝기 조절용 가변 저항과 백라이트 조절용 가변 저항을 좌우로 돌려 LCD의 밝기를 조절한다.
- 버튼 1을 누르면 LCD에 조도 센서의 출력값이 표시된다. 만약 이 값이 1023으로 변하지 않고 멈추어 있다면, 센서 옆의 가변 저항을 좌우로 돌려 AREF 기준 전압을 변화시킨다.

- 버튼 2를 누르면 LCD에 온습도 센서의 출력값이 표시된다.
- 버튼 3을 누르면 GPS 위치값을 표시하기 위하여 대기한다. 시리얼 포트를 사용하는 GPS의 드라이버 프로그램은 이후 학습 내용에서 구현한다.

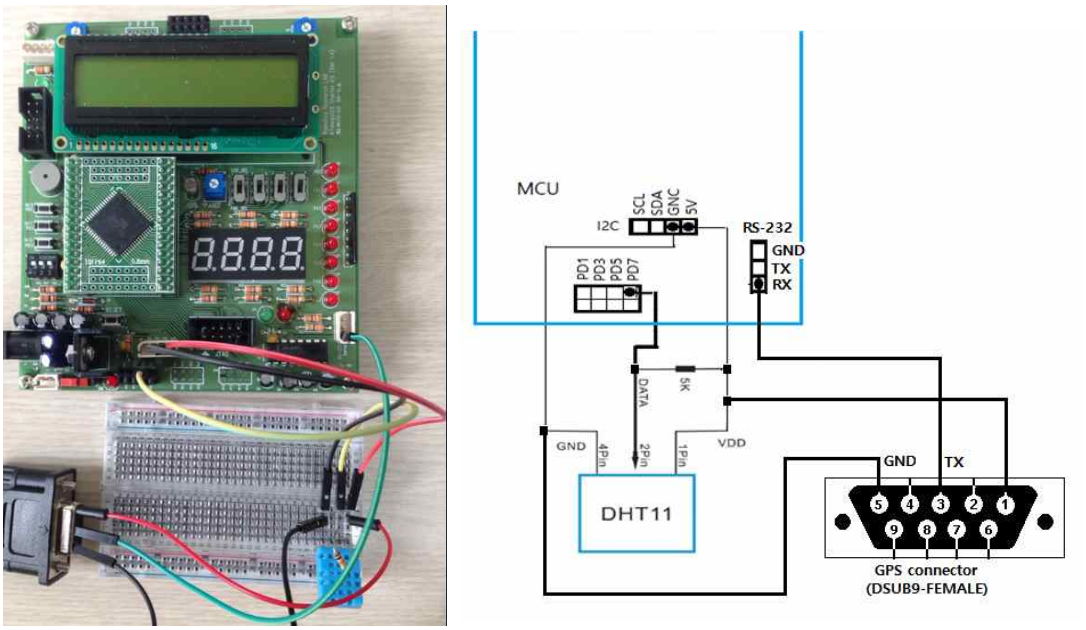
4 선정된 로봇 센서에 맞는 시리얼 통신 센서의 드라이버를 작성한다.

1. 시리얼 통신 출력을 가진 센서를 위한 드라이버 프로그램을 작성한다.

여기에서는 [그림 3-19]와 같이 GPS 수신기를 ATmega128 스타터키트의 시리얼 포트에 연결하여 사용하기로 한다.

GPS 수신기는 [그림 1-11]과 같은 9핀 커넥터(암)를 제공하는데, 이 중 사용되는 핀은 GPS에 전원을 공급하기 위한 1번 핀(5 V), 전원과 신호의 공통 접지선인 5번 핀(GND), GPS 신호선인 3번 핀(TX)이다.

이전 실습에서 사용된 온습도 센서의 연결을 유지한 채로 GPS 수신기도 연결하기 위하여 브레드보드로부터 전원선과 접지선을 연결하면 간편하다. GPS의 신호선은 스타터키트의 RS-232 시리얼 포트의 수신 단자(RX)에 연결한다.



(1) 연결 예 (2) 연결도

[그림 3-19] GPS와 스타터키트의 연결

2. Atmel Studio에서 [New Project...]를 생성한다.

- GCC C Executable Project
- Name: Test6 (Atmel Studio 7.0 버전에서는 파일 경로와 이름에 한글이 있으면 안 됨)
- Device Selection: ATmega128

[그림 3-20]과 같이 main.c의 내용을 입력한다. 이전 프로그램과 다른 점은 gps.h을 포함시키고, 메인 루틴 내부 switch 위의 read_GPS(strSensor) 부분을 활성화한 것이다.

```
/*
 * Test6.c
 *
 * Created: 2016-07-24 오후 7:15:19
 * Author : gabi http://cal.pknu.ac.kr
 */

#define F_CPU       16000000UL          // 16M Hz
#include <avr/io.h>
#include <util/delay.h>
#include <avr/interrupt.h>

#include "lcd.h"
#include "ringBuffer.h"
#include "cds.h"
#include "dht11.h"
#include "gps.h"                        // read_GPS 함수의 선언

... 생략(Test4.c와 동일) ...

                    read_CdS(strCdS);
                    read_DHT11(strDHT);
                    read_GPS(strGPS);

                    switch(buttonNum){
                            case 0: // CdS 조도 센서값을 표시함(ADC)
                                        lcd_clear();
                                        lcd_gotoxy(0,1); lcd_puts(strCdS);
                                        lcd_gotoxy(0,2); lcd_puts(">>Analog Sensor");
                                        break;
                            case 1: // DHT 온습도 값을 표시함 (Digital)
                                        lcd_clear();
                                        lcd_gotoxy(0,1); lcd_puts(strDHT);
                                        lcd_gotoxy(0,2); lcd_puts(">>Digital Sensor");
                                        break;
                            case 2: // GPS 위도, 경도를 표시함(시리얼 통신)
                                        lcd_clear();
                                        lcd_gotoxy(0,1); lcd_puts(strGPS);
                                        lcd_gotoxy(0,2); lcd_puts(">>Serial Comm.");
                            break;
                    }

...후략...(Test4.c와 동일)
```

[그림 3-20] Starter kit의 예제 프로그램(Test6.c) 입력

[그림 3-21]과 같이 ringBuffer.h를 수정한다. 링버퍼의 크기를 충분히 잡아준다.

```
/*
 * ringBuffer.h
 *
 * Created: 2016-07-24 오후 8:41:23
 * Author : gabi http://cal.pknu.ac.kr
 */
#ifndef __RINGBUFFER__
#define __RINGBUFFER__

#define BUFFER_SIZE 1000

void init_Buffer();
int getBufferNumData();
int saveBufferData(char ch);
int getBufferData(int numData, char *buf);
int showBuffer(char *str);
int isBufferFull();

#endif
```

[그림 3-21] Starter kit의 예제 프로그램(ringBuffer.h) 입력

[그림 3-22]와 같이 gps.h를 입력한다.

```
/*
 * gps.h
 *
 * Created: 2016-07-31 오후 10:58:59
 * Author : gabi http://cal.pknu.ac.kr
 */

#ifndef __GPS__
#define __GPS__

void init_GPS();
void read_GPS(char *);

#endif
```

[그림 3-22] Starter kit의 예제 프로그램(gps.h) 입력

[그림 3-23]과 같이 gps.c의 내용을 입력한다.

```
/*
 * gps.c
 *
 * Created: 2016-07-31 오후 10:57:33
```

```
 * Author : gabi http://cal.pknu.ac.kr
 */
#define F_CPU       16000000UL          // 16M Hz
#include <avr/io.h>
#include <util/delay.h>
#include <avr/interrupt.h>
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include "ringBuffer.h"
#include "gps.h"

char GPS_Buffer[BUFFER_SIZE];                                   // GPS 신호 저장 버퍼
int GPS_copy_flag = 0;                                          // 링버퍼 복사 신호
int numCopyData = 0;                                            // 링버퍼 복사 데이터 수

//USART 관련*********************************************
void init_GPS(void)
{
//*UBRR0H/L 레지스터 설정
        //Baud Rate 9600이므로 UBRR0L레지스터만 103으로 설정
        UBRR0H=0x00;
        UBRR0L=103;

        //*UCSR0B 레지스터 설정
        //수신 부분 인터럽트를 사용하므로
        //인터럽트 관련 비트중 Bit7만 enable
        //수신부, 송신부 동작 enable (bit4~3)
        //8비트 전송 이므로 UCZ02 bit =0 (bit2)
        //8비트 전송 이므로 9비트 전송시 사용되는 비트 disable(bit1~0)
        UCSR0B=(1<<RXCIE0)|(0<<TXCIE0)|(0<<UDRIE0)|(1<<RXEN0)|(1<<TXEN0)|
                (0<<UCSZ02)|(0<<RXB80)|(0<<TXB80);

        //*UCSR0C 레지스터 설정
        //비동기 전송 모드 이므로 UMSEL0=0(bit6)
        //패리티 기능을 사용하지 않으므로 UPM01~0=0(bit5~4)
        //stop bit를 1개 사용하므로 USBS0=0(bit3)
        //8비트 전송 이므로 UCZ01~0=0(bit2~1)

UCSR0C=(0<<UMSEL0)|(0<<UPM01)|(0<<UPM00)|(0<<USBS0)|(1<<UCSZ01)|(1<<UCSZ00);
}

ISR(USART0_RX_vect)
{
        char rxChar;
        static char rxChar_1 = 0;
        static int numSerialData = 0;

        rxChar = UDR0;

        if(!isBufferFull())     {                               // 링버퍼가 꽉 차있지 않을 경우
```

```
                saveBufferData(rxChar);                 // 링버퍼에 수신된 data 저장
                numSerialData++;
                // data가 0x0D와 0x0A로 끝날 경우 copy_flag set
                if ((rxChar_1 == 0x0D) && (rxChar == 0x0A)){
                        numCopyData = numSerialData;  // 0D0A까지 copy
                        numSerialData = 0;
                        GPS_copy_flag++;
                }
        }
        else                                            // GPS 링버퍼 꽉 차있을 경우
        {
                GPS_copy_flag = -1;
                numSerialData = 0;
                init_Buffer();
        }

        rxChar_1 = rxChar;
}

void Tx_Char(char data)                                 // USART0을 사용한 문자 전송
{
        while(!(UCSR0A & 0x20));
        UDR0 = data;
}

void Tx_String(char *string)                            // USART0을 사용한 문자열 전송
{
        char *p = string;

        while(*p != 0)
        {
                Tx_Char(*p++);
        }

}

//GSP Data Passing ********************************************
char token[16]={0};
char *GPS_strtok(char *pt)
{
        int n=0;

        //데이터 passing
        while((*pt!=',')&&(*pt!='*')&&(*pt!='\n')&&(*pt!='\r')) token[n++] = *pt++;

        token[n]='\0';
        return ++pt;
}

char Longitude[16];                                     // 경도
char Latitude[16];                                      // 위도
```

```
void GPGGA(char *pt)
{
  // $GPGGA,114455.532,3735.0079,N,12701.6446,E,1,03,7.9,48.8,M,19.6,M,0.0,0000*48
        int i=0;

        pt=GPS_strtok(pt);

        while(i++ < 15){
                pt=GPS_strtok(pt);

                switch(i){
                  case 2:           //위도 출력
                                    strcpy(Longitude, token);
                        break;
                  case 4:           //경도 출력
                                    strcpy(Latitude, token);
                                    break;
                }
        }
}

void NMEA0183_Parser(char *strIn)
{
        //들어오는 GPS데이터가 $GPGGA 로 시작된다면
        if (strncmp(strIn, "$GPGGA",6) == 0) GPGGA(strIn);
}

void read_GPS(char *strResult)                  // GPS 센서의 data 수신
{

        if(GPS_copy_flag<=0) return;            // GPS_copy_flag = 0 종료
        GPS_copy_flag = 0;                      // 복사 완료 하여, copy_flag clear

        getBufferData(numCopyData, GPS_Buffer);
        NMEA0183_Parser(GPS_Buffer);

        sprintf(strResult, "GPS:%d,%d", atoi(Latitude)/100, (int)atoi(Longitude)/100);
}
```

[그림 3-23] Starter kit의 예제 프로그램(gps.c) 입력

시리얼 통신을 사용하기 위한 ATmega128의 설정은 UCSRnA, UCSRnB, UCSRnC, UBRRnH, UBRRnL을 사용한다. 송수신 데이터는 UDRn에 저장된다.

[그림 3-24]의 UDRn(USART n I/O data register)는 n번 USART의 송수신 데이터를 저장하는 레지스터이다. 외부적으로는 송수신 시에 UDRn을 공통으로 사용하지만, 내부적으로는 송신 시에는 UDRn의 데이터가 TXBn(transmit data buffer n)에 저장되고, 수신 시에는 RXBn(receive data buffer n)의 데이터가 UDRn에 저장된다. 송신 시에는 UDREn 플래그가 1로 설정된 상태에서만 가능하다.

Bit	7	6	5	4	3	2	1	0	
	RXBn[7:0]								UDRn (Read)
	TXBn[7:0]								UDRn (Write)
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

출처: ATmega128 manual(http://www.atmel.com/images/doc2467.pdf). 2016. 10. 3. 스크린샷.
[그림 3-24] UDR 레지스터

[그림 3-25]의 UCSRnA (USART n control and status register A)는 n번 USART의 송수신 동작 제어와 상태를 설정하기 위한 레지스터이다.

- RXCn (USART receive complete)가 1이면 수신 버퍼에 데이터가 있음을 의미한다. 이때 수신 완료 인터럽트(RXCIEn)가 요청된다.
- RXCn (USART transmit complete)가 1이면 송신 버퍼에 보낼 데이터가 있음을 의미한다. 이때 송신 완료 인터럽트(TXCIEn)가 요청된다.
- UDREn (USART data register empty)가 1이면 새로운 데이터를 수신할 수 있음을 의미한다. 이때 송신 데이터 준비 완료 인터럽트(UDRIEn)가 요청된다.
- FEn (frame error)가 1이면 프레임 오류가 생겼음을 나타내는데, 수신 데이터의 스톱 비트(stop bit)가 0인 경우이다.
- DORn (data overrun)가 1이면 수신 데이터가 처리되기 전에 새로운 데이터가 검출되는 경우를 의미한다.
- UPEn (parity error)가 1이면 패리티 오류(parity error)가 검출되었음을 의미한다.
- U2Xn (double the USART transmission speed)는 비동기 모드에서 전송속도를 2배 높이는 기능을 설정한다.
- MPCMn (multi-processor communication mode)는 다중 프로세서 통신 모드를 설정한다.
- 더 자세한 내용은 매뉴얼을 참조한다.

Bit	7	6	5	4	3	2	1	0	
	RXCn	TXCn	UDREn	FEn	DORn	UPEn	U2Xn	MPCMn	UCSRnA
Read/Write	R	R/W	R	R	R	R	R/W	R/W	
Initial Value	0	0	1	0	0	0	0	0	

출처: ATmega128 manual(http://www.atmel.com/images/doc2467.pdf). 2016. 10. 3. 스크린샷.
[그림 3-25] UCSRnA 레지스터

[그림 3-26]의 UCSRnB (USART n control and status register B)는 송수신 동작 제어와 9비트 데이터 전송 설정 시에 9번째 비트를 저장하기 위한 레지스터이다.

- RXCIEn (RX complete interrupt enable)가 1이면 수신 완료 인터럽트를 사용할 수 있다.
- TXCIEn (TX complete interrupt enable)가 1이면 송신 완료 인터럽트를 사용할 수 있다.

- UDRIEn (USART data register empty interrupt enable)가 1이면 송신 데이터 레지스터 준비 완료 인터럽트를 사용할 수 있다.
- RXENn (receiver enable)가 1이면 해당 핀을 기존의 입출력 핀으로 사용하는 것이 아니라 시리얼 통신의 수신용으로 사용한다는 것을 의미한다.
- TXENn (transmit enable)가 1이면 해당 핀을 기존의 입출력 핀으로 사용하는 것이 아니라 시리얼 통신의 송신용으로 사용한다는 것을 의미한다.
- UPEn (parity error)가 1이면 패리티 오류(parity error)가 검출되었음을 의미한다.
- UCSZn2 (character size)는 UCSRnC의 UCSZn1, UCSZn0과 함께 송수신 문자의 데이터 수를 설정한다.
- RXB8n (receive data bit 8)은 전송 문자의 데이터 수가 9비트로 설정되었을 때, 수신된 9번째 비트(0~8번 비트 중 8번 비트)를 저장한다.
- TXB8n (transmit data bit 8)은 전송 문자의 데이터 수가 9비트로 설정되었을 때, 전송할 9번째 비트(0~8번 비트 중 8번 비트)를 저장한다.
- 더 자세한 내용은 매뉴얼을 참조한다.

Bit	7	6	5	4	3	2	1	0	
	RXCIEn	TXCIEn	UDRIEn	RXENn	TXENn	UCSZn2	RXB8n	TXB8n	UCSRnB
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R	R/W	
Initial Value	0	0	0	0	0	0	0	0	

출처: ATmega128 manual(http://www.atmel.com/images/doc2467.pdf). 2016. 10. 3. 스크린샷.
[그림 3-26] UCSRnB 레지스터

[그림 3-27]의 UCSRnC (USART n control and status register C)는 송수신 동작 제어를 설정하기 위한 레지스터이다.

- UMSELn (USART mode select)가 1이면 동기 통신, 0이면 비동기 통신을 의미한다.
- UPMn1, UPMn0 (parity mode)가 00, 10, 11이면 각각 패리티 없음, 짝수 패리티, 홀수 패리티를 의미한다. 01은 사용하지 않는다.
- USBSn (USART stop bit select)가 1이면 1 비트의 스톱 비트, 0이면 2 비트의 스톱 비트를 사용한다.
- UCSZn1, UCSZn0 (character size)는 UCSRnB의 UCSZn2와 함께 사용된다. UCSZn2~0이 000, 001, 010, 011, 111이면 각각 데이터 크기가 5비트, 6비트, 7비트, 8비트, 9비트임을 나타낸다. 100, 101, 110은 사용하지 않는다.
- UCPOLn (clock parity)는 동기 모드에서만 사용된다. 이 비트가 1이면 하강 에지에서 데이터를 송신하고 상승 에지에서 데이터를 수신한다. 이 비트가 0이면 반대로 동작한다.

Bit	7	6	5	4	3	2	1	0	
	–	UMSELn	UPMn1	UPMn0	USBSn	UCSZn1	UCSZn0	UCPOLn	UCSRnC
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	1	1	0	

출처: ATmega128 manual(http://www.atmel.com/images/doc2467.pdf). 2016. 10. 3. 스크린샷.
[그림 3-27] UCSRnC 레지스터

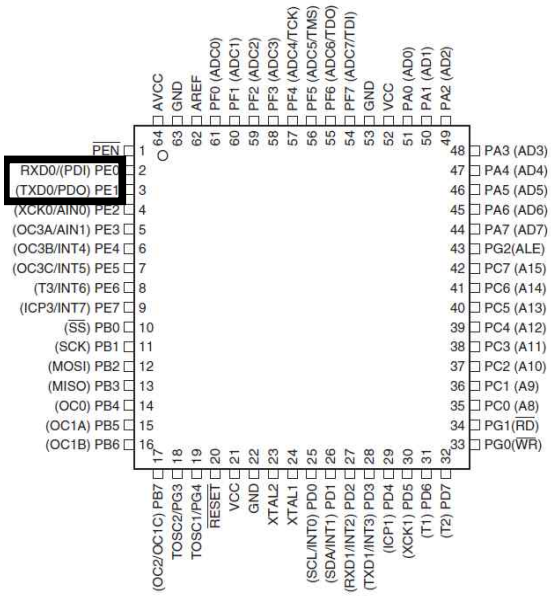
예를 들어, 0번 USART를 위한 프로그램 내부의 설정을 확인해 보자.

```
UCSR0B = (1<<RXCIE0)|(0<<TXCIE0)|(0<<UDRIE0)|(1<<RXEN0)|(1<<TXEN0)|
         (0<<UCSZ02)|(0<<RXB80)|(0<<TXB80);
UCSR0C = (0<<UMSEL0)|(0<<UPM01)|(0<<UPM00)|(0<<USBS0)|(1<<UCSZ01)|(1<<UCSZ00);
```

이 설정에 의하면 수신 완료 인터럽트(RXCIE0=1)를 사용하고 송신 완료 인터럽트(TXCIE0=0)와 송신 데이터 레지스터 준비 완료 인터럽트(UDRIE0=0)는 사용하지 않는다. [그림 3-28]의 E포트의 0번과 1번 핀을 각각 수신 핀(RXEN0=1)과 송신 핀(TXEN0=1)으로 사용한다.

UCSZ02, UCSZ01, UCSZ00이 011이므로 송수신 데이터의 크기는 8비트이며, 9번째 비트를 저장할 필요가 없으므로 RXB80, TXB80은 0으로 설정되어 있다.

UMSEL0이 0이므로 비동기 통신 모드로 동작한다. UPM01, UPM00이 00이므로 패리티는 사용하지 않는다.



출처: ATmega128 manual(http://www.atmel.com/images/doc2467.pdf). 2016. 10. 3. 스크린샷.
[그림 3-28] USART0의 송수신 핀

[그림 3-29]의 UBRRn (USART baud rate register n)는 시리얼 포트의 전송 속도인 보레이트를 설정하기 위한 레지스터이다.

- [그림 3-30]에 의하면, ATmega128이 16 MHz로 동작할 때, 9600 bps의 보레이트는 UBRR을 103으로 설정한다.
- 기타 더 자세한 내용은 매뉴얼을 확인한다.

Bit	15	14	13	12	11	10	9	8	
	–	–	–	–	UBRRn[11:8]				UBRRnH
	UBRRn[7:0]								UBRRnL
	7	6	5	4	3	2	1	0	
Read/Write	R	R	R	R	R/W	R/W	R/W	R/W	
	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	
	0	0	0	0	0	0	0	0	

출처: ATmega128 manual(http://www.atmel.com/images/doc2467.pdf). 2016. 10. 3. 스크린샷.
[그림 3-29] UBRR 레지스터

Baud Rate (bps)	f_{osc} = 16.0000 MHz				f_{osc} = 18.4320 MHz				f_{osc} = 20.0000 MHz			
	U2X = 0		U2X = 1		U2X = 0		U2X = 1		U2X = 0		U2X = 1	
	UBRR	Error	UBRR	Error	UBRR	Error	UBRR	Error	UBRR	Error	UBRR	Error
2400	416	-0.1%	832	0.0%	479	0.0%	959	0.0%	520	0.0%	1041	0.0%
4800	207	0.2%	416	-0.1%	239	0.0%	479	0.0%	259	0.2%	520	0.0%
9600	103	0.2%	207	0.2%	119	0.0%	239	0.0%	129	0.2%	259	0.2%
14.4k	68	0.6%	138	-0.1%	79	0.0%	159	0.0%	86	-0.2%	173	-0.2%
19.2k	51	0.2%	103	0.2%	59	0.0%	119	0.0%	64	0.2%	129	0.2%
28.8k	34	-0.8%	68	0.6%	39	0.0%	79	0.0%	42	0.9%	86	-0.2%
38.4k	25	0.2%	51	0.2%	29	0.0%	59	0.0%	32	-1.4%	64	0.2%
57.6k	16	2.1%	34	-0.8%	19	0.0%	39	0.0%	21	-1.4%	42	0.9%
76.8k	12	0.2%	25	0.2%	14	0.0%	29	0.0%	15	1.7%	32	-1.4%
115.2k	8	-3.5%	16	2.1%	9	0.0%	19	0.0%	10	-1.4%	21	-1.4%
230.4k	3	8.5%	8	-3.5%	4	0.0%	9	0.0%	4	8.5%	10	-1.4%
250k	3	0.0%	7	0.0%	4	-7.8%	8	2.4%	4	0.0%	9	0.0%
0.5M	1	0.0%	3	0.0%	–	–	4	-7.8%	–	–	4	0.0%
1M	0	0.0%	1	0.0%	–	–	–	–	–	–	–	–
Max (1)	1 Mbps		2 Mbps		1.152 Mbps		2.304 Mbps		1.25 Mbps		2.5 Mbps	

출처: ATmega128 manual(http://www.atmel.com/images/doc2467.pdf). 2016. 10. 3. 스크린샷.
[그림 3-30] UBRR 레지스터의 내용

GPS의 데이터는 시리얼 통신(9600, 8, 1, None)을 통하여 로봇 MCU에 전달된다. 수신 데이터가 있으면 인터럽트 서비스 루틴인 ISR(USART0_RX_vect)가 실행된다. ISR은 링버퍼에 저장할 공간이 남아있으면 수신된 8비트 데이터를 링버퍼에 저장한다.

이때 직전 데이터가 0x0d이고 현재 데이터가 0x0a인 경우는 NMEA0183의 한 문장이 끝났

다는 것을 의미하므로, 링버퍼의 내용을 복사하라는 신호인 GPS_copy_flag를 1로 설정한다. NMEA0183의 모든 문장은 '$'로 시작하여 <CR><LF>로 끝나도록 정의되어 있다.
로봇 MCU 펌웨어의 메인 루프에서 계속하여 실행되는 read_GPS() 함수는 GPS_copy_flag가 0이 아닌 경우에만 링버퍼의 내용을 GPS_buffer에 저장하고, 그 내용을 분석한다.
로봇의 위치 데이터는 경도와 위도를 포함한 NMEA0183의 GPGGA 문장을 분석하면 얻을 수 있다.

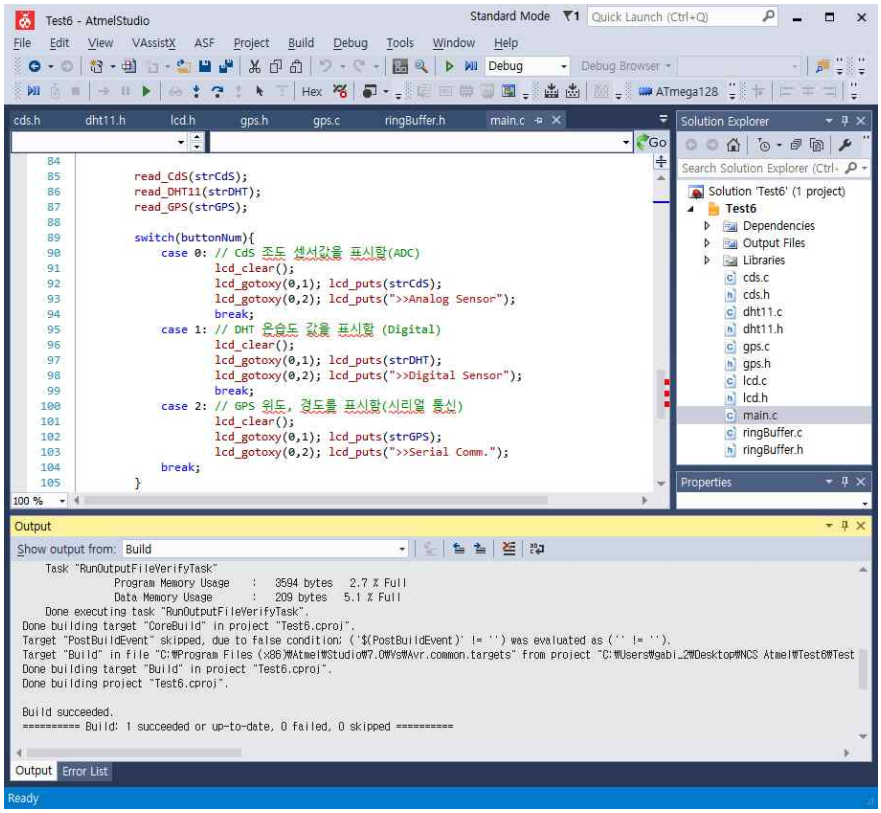
예를 들어 아래와 같은 문장을 해석한다고 가정하자.

$GPGGA,114455.532,3735.0079,N,12701.6446,E,1,03,7.9,48.8,M,19.6,M,0.0,0000*48<CR><LF>

쉼표를 기준으로 토큰(token)을 생성시키면, 2번째 토큰인 "3735.0079"와 3번째 토큰인 'N'을 합쳐 북위 37도 35.0079분, 4번째 토큰인 "12701.6446"과 5번째 토큰인 'E'를 합쳐 동경 127도 01.6446분이다.

Test6 모듈은 [그림 3-31]과 같이 11개의 소스 파일로 이루어져 있다.
- main.c
- lcd.h
- lcd.c
- ringBuffer.h
- ringBuffer.c
- cds.h
- cds.c
- dht11.h
- dht11.c
- gps.c
- gps.h

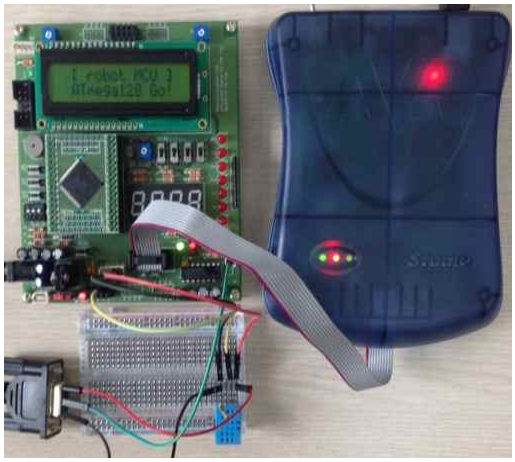
Build > Build Solution(F7)을 실행하여 프로그램에 오류가 없음을 확인한다. Debug > Start Without Debugging(Ctrl-Alt-F5)를 이용하여 MCU 키트에 펌웨어를 다운로드한 후 실행한다.
- 버튼 1: 버튼을 누르면 LCD에 CdS 조도 센서값을 표시한다.
- 버튼 2: 버튼을 누르면 LCD에 DHT11 온습도 센서값을 표시한다.
- 버튼 3: 버튼을 누르면 LCD에 로봇의 위치(경도, 위도)를 표시한다.



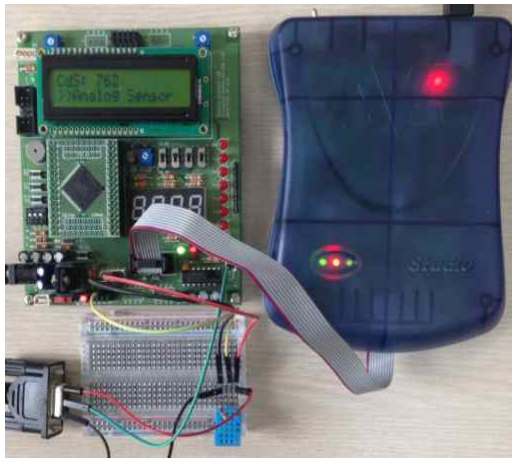
[그림 3-31] 로봇 MCU 펌웨어 컴파일 및 다운로드

로봇 MCU의 실행 예는 [그림 3-32]와 같다.

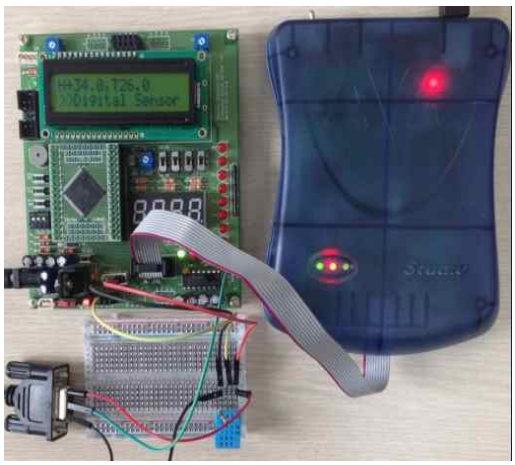
- 초기 상태에서 버튼을 누르면, 스위치가 떨어지는 순간 해당 버트의 인터럽트 서비스 루틴이 실행된다. 버튼들은 누르는 순간 1에서 0으로 출력이 변화하고, 떼는 순간 0에서 1로 변하기 때문에 로봇 MCU에서 상승 에지로 설정된 인터럽트는 손을 떼는 순간에 단 한 번 실행된다.
- 전압이 인가되면, LCD에 글자가 나타난다. 만약 LCD에 아무것도 나타나지 않으면 화면 밝기 조절용 가변 저항과 백라이트 조절용 가변 저항을 좌우로 돌려 LCD의 밝기를 조절한다.
- 버튼 1을 누르면 LCD에 조도 센서의 출력값이 표시된다. 만약 이 값이 1023으로 변하지 않고 멈추어 있다면, 센서 옆의 가변 저항을 좌우로 돌려 AREF 기준 전압을 변화시킨다.
- 버튼 2를 누르면 LCD에 온습도 센서의 출력값이 표시된다.
- 버튼 3을 누르면 LCD에 마지막으로 수신한 GPS 위치값이 표시된다.



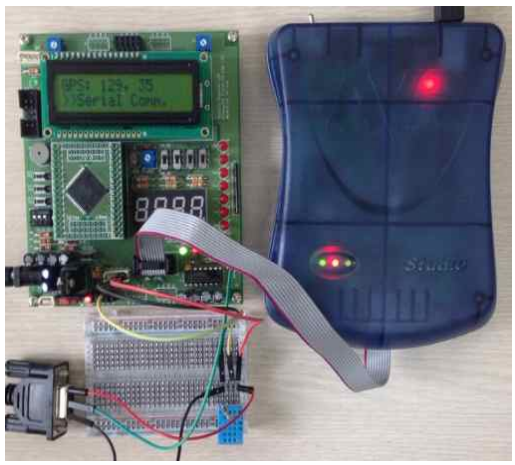
(1) 초기 상태



(2) 버튼 1, 조도값 표시



(3) 버튼 2, 온습도 표시



(4)버튼 3, 위치 표시

[그림 3-32] 실행 예

수행 tip

- Build 실행 후의 결과 창에는 오류나 경고 등의 메시지가 나타난다. 이 메시지를 잘 확인하여 프로그래밍 중 생기는 실수나 오류를 수정한다.
- 인터럽트 서비스 루틴은 가능한 짧고 실행 시간이 짧도록 프로그램한다.