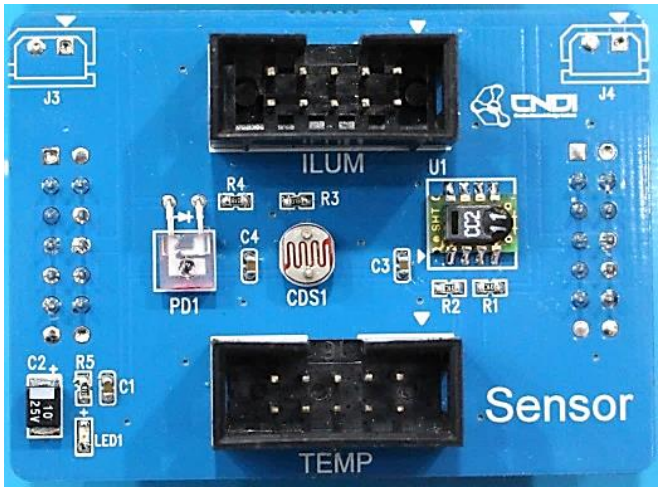
온도/습도/조도 Sensor Module



1. 사양

1.1. Description

온/습도 센서는 온도와 습도를 측정하는 센서이다. 온도계나 습도계를 보면 특정 부분을 통해 온도나 습도의 정도를 표시해 주는 것을 보았을 것이다. 다만, 온/습도 센서의 경우 이렇게 받아들인 정보를 하드웨어를 통하여 받아들여 디지털화된 결과를 도출한다.

조도 센서는 쉽게 말해 빛의 양을 측정하는 센서이다. 사용 예로, 요즘 스마트폰을 보면 사진 찍을 때 빛의 양을 조절해주는 기능이 있는데, 빛의 양이 어느 정도인지 확인하기 위하여 조도 센서가 쓰인다. 혹은, 주변이 어두운 경우 스마트폰에서 자동으로 LCD의 밝기를 더 밝게 조절하는 기능에도 사용된다. 측정하는 방법은 빛의 세기에 따라 저항 값이 변하므로 전압 값을 측정하여 이를 수치화 하면 그 정도를 알 수 있도록 한다. 조도 센서는 동작 방식이나 구성에 따라 다양한 종류가 있다.

이 모듈은 빛을 검출할 수 있는 포토 다이오드와, CDS 센서 그리고 온/습도를 검출 할 수 있는 SH11 센서가 장착된 모듈이다. 포토 다이오드와 CDS센서는 A/D 변환을 통하여 검출 되고 온/습도 센서는 I2C를 이용하여 검출 한다. 자세한 타이밍이나 검출 방법은 제공하는 데이터시트를 참조 하기 바란다. 동작 전압은 5V 이다.

1.2. 구성

1 x Sensor Module

1 x Power Cable

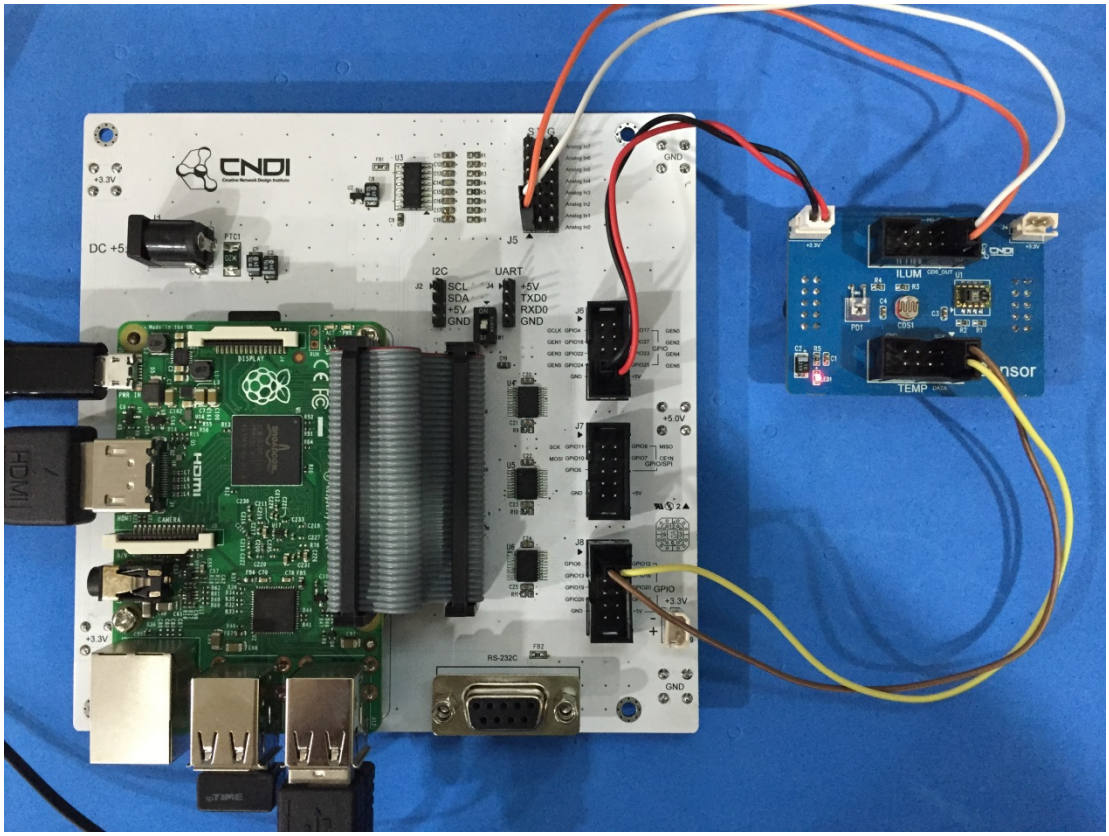
Module Cable

1 x RaspberryPi Adapter Module(별도구매)

2. User Guide

2.1. 결선

모듈 별 결선 방법은 다음의 그림 및 표와 같다.



Module	Raspberry Pi	Sensor
Pin	Analog In0 S	ILUM PD_OUT
Pin	Analog In1 S	ILUM CDS_OUT
Pin	GPIO6	TEMP SCK
Pin	GPIO12	TEMP DATA

모듈의 장치 및 회로에 대한 상세한 내용은 데이터시트 및 회로도를 참고한다.
2핀 전원 케이블을 사용하여 라즈베리파이 어댑터의 +5V 핀과 GND핀에 연결하고 J3 또는 J4의 전원 컨넥터와 연결한다. 이 때, 화살표로 표시된 핀이 전원이며, 나머지 한 핀이 GND 이다.

2.2. 예제프로그램

각 센서를 통해 측정한 값은 어댑터 보드를 통해 라즈베리파이로 전달되어 주기적으로 표시한다. 조도 값은 밝을수록 수치가 높게 측정된다.

- lib_sensor.h

```
#ifndef  __LIB_SENSOR_H__
#define  __LIB_SENSOR_H__

#include <stdio.h>
#include <string.h>
#include <errno.h>

#include <wiringPi.h>
#include <wiringPiSPI.h>

#define CS_MCP3208      8                   //GPIO 8
#define SPI_CHANNEL     0                   //SPI Channel
#define SPI_SPEED       1000000             //spi speed

#define SET_DATA()      digitalWrite(SDA, 1)
#define CLR_DATA()      digitalWrite(SDA, 0)
#define SET_SCK()       digitalWrite(SCK, 1)
```

```
#define CLR_SCK()          digitalWrite(SCK, 0)
#define READ_DATA()        digitalRead(SDA)
#define READ_SCK()         digitalRead(SCK)

#define  SCK               6
#define  SDA               12

#define NOACK   0
#define ACK     1
                                        // Addr      Code(command)    r/w
#define MEASURE_TEMP            0x03    // 000         0001             1
#define MEASURE_HUMI            0x05    // 000         0010             1
#define READ_STATUS_REG         0x07    // 000         0011             1
#define WRITE_STATUS_REG        0x06    // 000         0011             0
#define RESET                   0x1e    // 000         1111             0

enum { TEMP, HUMI };

void SHT11_Init (void);
void Connection_reset (void);
void Transmission_start (void);
float get_SHT11_data (unsigned char type);
unsigned char Write_byte (unsigned char value);
unsigned char Read_byte (unsigned char ack);
unsigned char Measure (unsigned short *p_value,
unsigned short *p_checksum,        unsigned char mode);
void calc_SHT11 (unsigned short p_humidity ,unsigned short p_temperature);

int ReadMcp3208ADC(unsigned char adcChannel);

#endif   /* __LIB_SENSOR_H__ */
```

- lib_sensor.c

```
#include "lib_sensor.h"
```

```
float      val_temp, val_humi;
unsigned short SHT11_humi, SHT11_temp;
unsigned short error, checksum;
unsigned char sensing_type;

static void I2C_data_output (void)  { pinMode(SDA, OUTPUT); }
static void I2C_data_input (void)   { pinMode(SDA, INPUT); }
static void I2C_sck_output (void)   { pinMode(SCK, OUTPUT); }
static void I2C_sck_input (void)    { pinMode(SCK, INPUT); }

static void SET_I2CDATA_PIN (void) { digitalWrite(SDA, 1); }
static void CLR_I2CDATA_PIN (void){ digitalWrite(SDA, 0); }
static void SET_I2CSCK_PIN (void)  { digitalWrite(SCK, 1); }
static void CLR_I2CSCK_PIN (void)  { digitalWrite(SCK, 0); }
static int READ_I2CDATA_PIN (void){ return digitalRead(SDA); }
static int READ_I2CSCK_PIN (void)  { return digitalRead(SCK); }

void SHT11_Init (void)
{
        I2C_data_output ();                 // DDRF |= 0x02;
        I2C_sck_output ();          // DDRF |= 0x01;
        Connection_reset ();
}

void Connection_reset (void)
{
        unsigned char i;
        SET_DATA();                                 // Initial state
        CLR_SCK();                                  // Initial state
        delayMicroseconds(1);
        for (i=0; i<9; i++) {               // 9 SCK cycles
                SET_SCK();
                delayMicroseconds(1);
        CLR_SCK();
```

```
                delayMicroseconds(1);
        }
}

void Transmission_start (void)
{
        SET_DATA();                     //Initial state
        CLR_SCK();                      //Initial state
        delayMicroseconds(1);

        SET_SCK();
        delayMicroseconds(1);

        CLR_DATA();
        delayMicroseconds(1);

        CLR_SCK();
        delayMicroseconds(1);

        SET_SCK();
        delayMicroseconds(1);

        SET_DATA();
        delayMicroseconds(1);

        CLR_SCK();
}

float get_SHT11_data (unsigned char type)
{
        sensing_type    =       type;
        error   =       0;

        if (sensing_type == HUMI) {
```

```
                // measure humidity
                error   +=      Measure (&SHT11_humi, &checksum, HUMI);
                if (error != 0)                 // [Error] connection reset
                        Connection_reset ();
                else                            // Calculate humidity, temperature
                        calc_SHT11 (SHT11_humi, SHT11_temp);
                return  val_humi;
        }
        else if (sensing_type == TEMP) {
                // measure temperature
                error   +=      Measure (&SHT11_temp, &checksum, TEMP);
                if (error != 0)                 // [Error] connection reset
                        Connection_reset ();
                else                            // Calculate humidity, temperature
                        calc_SHT11 (SHT11_humi, SHT11_temp);
                return  val_temp;
        }
        else {
                return  0;
        }
}

unsigned char Measure (unsigned short *p_value, unsigned short *p_checksum,
                        unsigned char mode)
{
        unsigned short error    =       0;
        unsigned short SHT11_msb, SHT11_lsb;

        switch (mode)                           //send command to sensor
        {
                case TEMP :
                        error   +=      Write_byte (MEASURE_TEMP);
                        break;
                case HUMI :
```

```
                        error   +=      Write_byte (MEASURE_HUMI);
                        break;
                default :
                        break;
        }
        if (error != 0)
                return  error;

        I2C_data_input ();
        while (READ_DATA());
        I2C_data_input();

        SHT11_msb       =       Read_byte (ACK); // read the first byte (MSB)
        SHT11_lsb       =       Read_byte (ACK); // read the second byte (LSB)
        *p_value =      (SHT11_msb * 256) + SHT11_lsb;
        *p_checksum     =       Read_byte (NOACK);      // read checksum

        return error;
}

unsigned char Write_byte (unsigned char value)
{
        unsigned char i, error  =       0;
        I2C_data_output ();
        for (i=0x80; i>0; i/=2)     {
                if (i & value)      SET_DATA ();
                else                CLR_DATA ();

                delayMicroseconds(1);
                SET_SCK ();
                delayMicroseconds(1);
                CLR_SCK ();
                delayMicroseconds(1);
        }
```

```
        SET_DATA ();
        I2C_data_input ();
        delayMicroseconds(1);
        SET_SCK ();
        error    =        READ_DATA ();

        CLR_SCK ();
        I2C_data_output ();

        return error;
}

unsigned char Read_byte(unsigned char ack)
{
        unsigned char i, val        =        0;
        I2C_data_input ();
        SET_DATA();
        delayMicroseconds(1);

        for (i=0x80; i>0; i/=2)     {
                SET_SCK();
                delayMicroseconds(1);
                if (READ_DATA())
                        val = (val | i);
                CLR_SCK();
                delayMicroseconds(1);
        }
        I2C_data_output();

        if (ack)   CLR_DATA();
        else                SET_DATA();

        SET_SCK();
        delayMicroseconds(1);
```

```
        CLR_SCK();
        delayMicroseconds(1);
        SET_DATA();

        return val;
}

void calc_SHT11 (unsigned short humidity, unsigned short temperature)
{
        const float C1    =     -2.0468;          // for 12 Bit
        const float C2    =     0.0367;           // for 12 Bit
        const float C3    =     -0.0000015955;    // for 12 Bit
        const float T1    =     0.01;                   // for 12 Bit
        const float T2    =     0.00008;                // for 12 Bit

        float rh_lin;             // Relative Humidity
        float rh_true;            // Humidity Sensor RH/Temperature compensation
        float t_C;                        // Temperature
        float rh  =     (float)humidity;
        float t           =     (float)temperature;

        t_C                       =     ((t * 0.01) - 40.1) - 5;
        rh_lin            =     (C3 * rh * rh) + (C2 * rh) + C1;
        rh_true   =     (t_C - 25) * (T1 + (T2 * rh)) + rh_lin;

        if (rh_true > 100) rh_true = 100;
        if (rh_true < 0.1)  rh_true = 0.1;

        val_temp          =     t_C;
        val_humi=       rh_true;
}

int ReadMcp3208ADC(unsigned char adcChannel)
{
```

```
        unsigned char buff[3];
        int nAdcValue = 0;

        buff[0] = 0x06 | ((adcChannel & 0x07) >> 2);
        buff[1] = ((adcChannel & 0x07)<<6);
        buff[2] = 0x00;

        digitalWrite(CS_MCP3208,0);                    //low cs Active

        wiringPiSPIDataRW(SPI_CHANNEL, buff, 3);

        buff[1] = 0x0F & buff[1];
        nAdcValue = (buff[1]<<8) | buff[2];

        //spi chip Select command
        digitalWrite(CS_MCP3208, 1);

        return nAdcValue;
}
```

- sensor.c

```
#include "lib_sensor.h"

volatile float temp;
volatile float humi;

int main (void)
{
        int nCdsChannel = 0;
        int nPhotoCellChannel = 1;

        int nCdsValue = 0;
        int nPhotoCellValue = 0;

        if(wiringPiSetupGpio() == -1) {
```

```
                fprintf(stdout,"Not start wiringPi: %s₩n",strerror(errno));
                return 1;
        }

        if(wiringPiSPISetup(SPI_CHANNEL, SPI_SPEED) == -1) {
                fprintf(stdout, "wiringPiSPISetup Failed: %s₩n", strerror(errno));
                return 1;
        }
        pinMode(CS_MCP3208, OUTPUT);
        SHT11_Init();

        while (1)
        {
                Transmission_start();
                temp = get_SHT11_data (TEMP);    // Sensing Temp
                delay(100);

                Transmission_start();
                humi = get_SHT11_data (HUMI);    // Sensing Humi
                delay(100);

                printf("Temp = %2.2f [C], Humi = %2.2f [%]₩n", temp, humi);

                //sensorRead
                nCdsValue = ReadMcp3208ADC(nCdsChannel);
                nPhotoCellValue = ReadMcp3208ADC(nPhotoCellChannel);
                printf("CSD Sensor Value = %u₩n", nCdsValue);
                printf("Photocell Sensor Value = %u₩n", nPhotoCellValue);
                delay(500);
        }
        return 0;
}
```