UNIVERSITÀ
DI SIENA
1240

RELAZIONE DEL PROGETTO DI ARCHITETTURA DEI CALCOLATORI

SVILUPPO DI UN MONITOR PER PROCESSI FERMENTATIVI, STUDIO DEL PROTOCOLLLO I2C, CREAZIONE DI UNA LIBRERIA PER UN SENSORE IN C++

AUTORI

RICCARDO NOCELLA
LUCIANO BIGIOTTI

A/A 2015/2016

OBIETTIVI

questo progetto si prefigge come obiettivi:

- *sviluppare un monitor per il controllo di processi fermentativi*

  Verrà sviluppato un monitor per il controllo e l'analisi di processi fermentativi, anche da remoto. Infatti attraverso la board UDOO verranno costantemente campionati i dati di alcool, temperatura, pressione. Tali dati sono visualizzabili anche attraverso internet sul sito appositamente creato. Inoltre, tramite un led, si simulerà l'avvio di una cintura riscaldante se la temperatura della bevanda scendesse al di sotto di una certa temperatura prefissata (22 °C)

- *lo sviluppo di una libreria per un sensore, usato nel monitor, nel linguaggio C++*

  Verrà realizzata una libreria in C++ per il sensore di temperatura/pressione utillizato nel monitor.

- *lo studio del protocollo I2C*

  Sarà presentato il protocollo I2C utilizzato nel progetto sopracitato per la comunicazione tra board e sensori.

INDICE

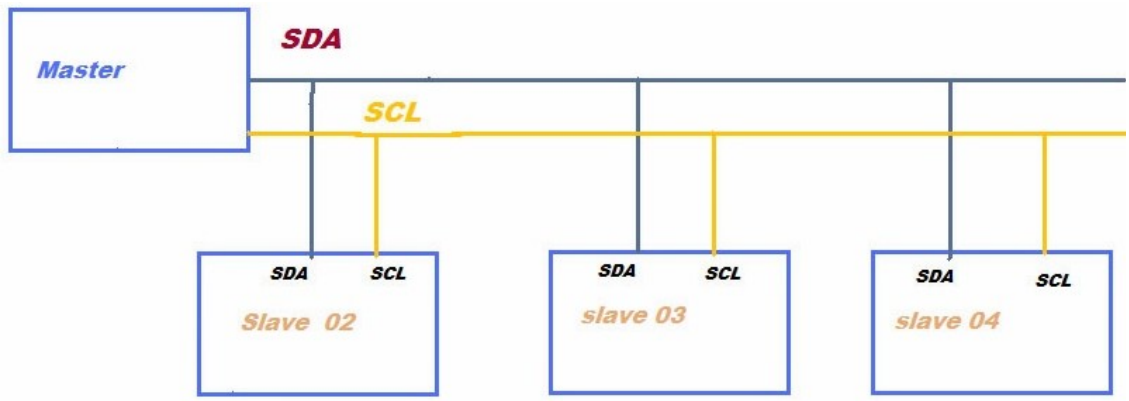
IL PROTOCOLLO I2C

Introduzione

In un sistema di elaborazione i vari sottosistemi (processore, memoria, dispositivi di I/O) devono essere interfacciati l'uno con l'altro per comunicare tra loro. Tale obiettivo è normalmente ottenuto tramite l'uso di un BUS.
Il BUS è un canale di comunicazione condiviso che usa un insieme di fili per collegare diversi sottosistemi.
Affinché gli elementi interagiscano correttamente, è necessario, inoltre, stabilire delle specifiche di comunicazione. L'insieme delle regole di trasmissione prende il nome di protocollo. La Philips ha brevettato un protocollo di comunicazione seriale, multi-master (in cui cioè più dispositivi possono assumere il controllo del bus) con velocità di comunicazione che raggiunge i 100 Kbit/s nella modalità normale (standard mode), i 400 Kbit/s nella modalità veloce (fast mode) e i 3.4 Mbit/s nella modalità ad alta velocità (High Speed Mode I2C v.2.0) chiamato I2C (Inter Integrated Circuit), il quale consente il colloquio fra più dispositivi collegati fra loro attraverso un bus composto da due linee.

Caratteristiche del protocollo I2C

Il protocollo utilizza un BUS formato da due linee bidirezionali. Una linea serve a trasportare il segnale di sincronizzazione (clock), l'altra serve per il trasferimento di dati. La prima viene chiamata "scl" (Serial Clock), la seconda "sda" (Serial Data).
I segnali che transitano sul BUS possono avere valore di "1" o "0" e le tensioni che le rappresentano sono quelle d'alimentazione, di voltaggio pari a 5V di norma, e di massa rispettivamente.



Ogni dispositivo collegato a queste linee è dotato di un indirizzo univoco, di 7 o 10 bit (la versione da 10 bit è relativa all'ultima revisione dello standard) e può agire sia da master che da slave, secondo le funzioni previste al suo interno. Il master si occupa di iniziare la trasmissione e di generare la tempistica del trasferimento, mentre lo slave è quel dispositivo indirizzato dal master. Entrambe le due categorie appena descritte possono assumere il ruolo di trasmittente o ricevente.

La modalità di trasferimento dati puo essere schematizzata come:

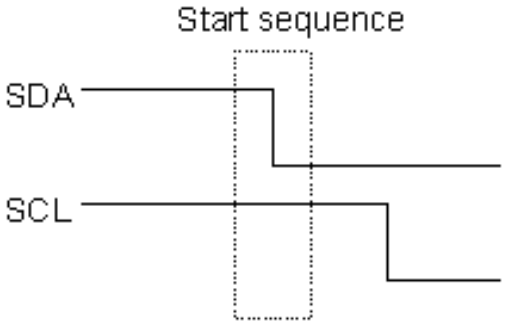
D1 vuole ricevere (inviare) i dati a D2

1. D1 (master) spedisce l'indirizzo di D2 (slave) sul bus
2. D2 (D1) (slave-transmitter) trasmette i dati ad D1(D2) (master-receiver)
3. D1 termina il trasferimento

Tale schema è valido sia nel caso in cui il master voglia ricevere che trasmettere dati .

Descrizione della trasmissione dati

Dopo un controlllo sull'occupazione del BUS, la trasmissione inizia con la generazione di un segnale di start, da parte del master, sulla linea. La condizione di start consiste nel lasciare la linea "scl" allo stato "alto", mentre la linea "sda", subisce una transizione dallo stato '1' allo stato '0'.



Dopo la generazione del segnale di start inizia la trasmissione dei dati vera e propria. Ciascun pacchetto dati consta di 8 bit più un ulteriore bit detto di acknowledgement. Nel primo pacchetto dati inviato, dopo la condizione di start, gli otto bit non sono dati, infatti i primi sette bit rappresentano l'indirizzo dello slave mentre l'ottavo serve a specificare la funzione che deve svolgere il ricevente. Tale bit in base allo standard previsto dal protocollo può assumere due valori con i seguenti significati:

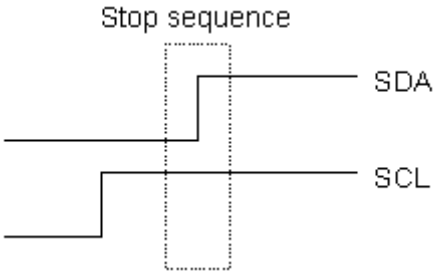
- '0' : ricezione dati;

- '1' : trasmissione dati;

Dopo la trasmissione d'ogni byte chi trasmette ha l'obbligo di lasciare la linea "sda" allo stato "alto", in modo da permettere a chi riceve dei dati, di darne conferma tramite il meccanismo dell'acknowledgement (riconoscimento): esso consiste nell'abbassare la linea "sda" in corrispondenza del nono impulso (9 bit detto appunto di acknowledgement) presente sulla linea "scl".
Se tale bit fosse posto a zero (ack) da parte del dispositivo ricevente vuole dire che ha ricevuto

correttamente i dati ed è pronto a riceverne degli altri, se invece il bit non venisse generato o fosse settato ad uno (nack) significa che il ricevente non può accettare altri dati o la trasmissione non si è svolta in modo corretto e il master deve terminare il trasferimento generando una sequenza di stop sulla linea.

Il segnale di stop è sempre generato dal master, come quello dello start. Esso consiste nel far variare la linea sda dallo stato “basso” a quello “alto” in corrispondenza del periodo “alto” della linea “scl. In figura un esempio di segnale di stop:



Numero massimo di device collegabili sul BUS con I2C

Se i primi sette bit del primo byte servono a identificare l'indirizzo di un device sulla linea allora il numero di device massimo collegabili sono 128 (2^7). Tuttavia ciò non è corretto in quanto due gruppi ciascuno di otto indirizzi (0000 XXX e 1111 XXX) sono riservati per funzioni predefinite.

La seguente tabella illustra gli indirizzi e le relative funzioni:

X = don't care; 1 = HIGH; 0 = LOW.

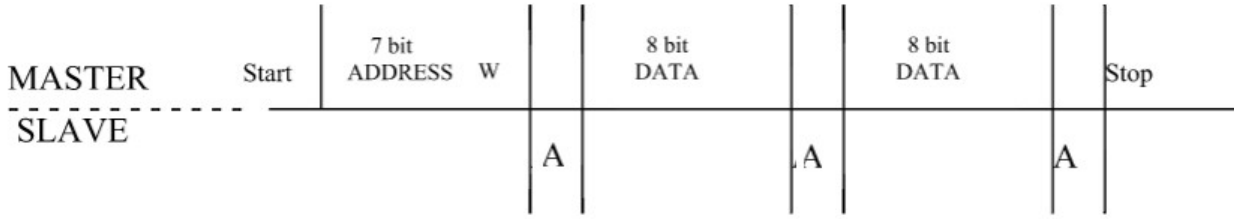
Slave address	R/$\overline{W}$ bit	Description
0000 000	0	general call address[1]
0000 000	1	START byte[2]
0000 001	X	CBUS address[3]
0000 010	X	reserved for different bus format[4]
0000 011	X	reserved for future purposes
0000 1XX	X	Hs-mode master code
1111 1XX	1	device ID
1111 0XX	X	10-bit slave addressing

Possibili modalita del trasferimento dati

I device collegati sul bus possono effettuare vari tipi di operazione come ad esempio la scrittura, la lettura o scrittura/lettura.

Operazione di scrittura

Nell'operazione di scrittura il master trasmette i dati allo slave

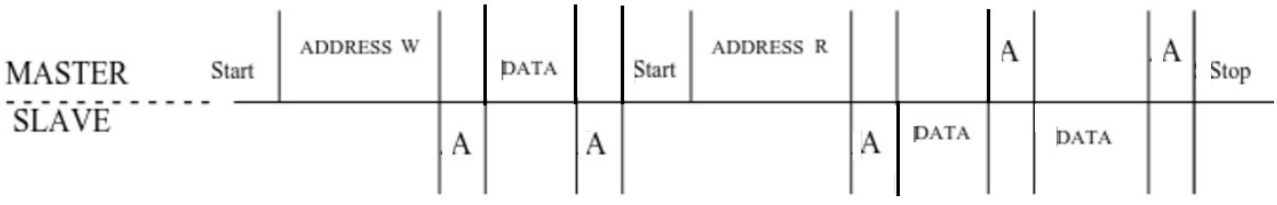


Operazione di lettura

Nell'operazione di lettura il master riceve i dati dallo slave

Operazione di scrittura/lettura

Nell'operazione di scrittura/lettura il master trasmette e poi riceve i dati dallo slave

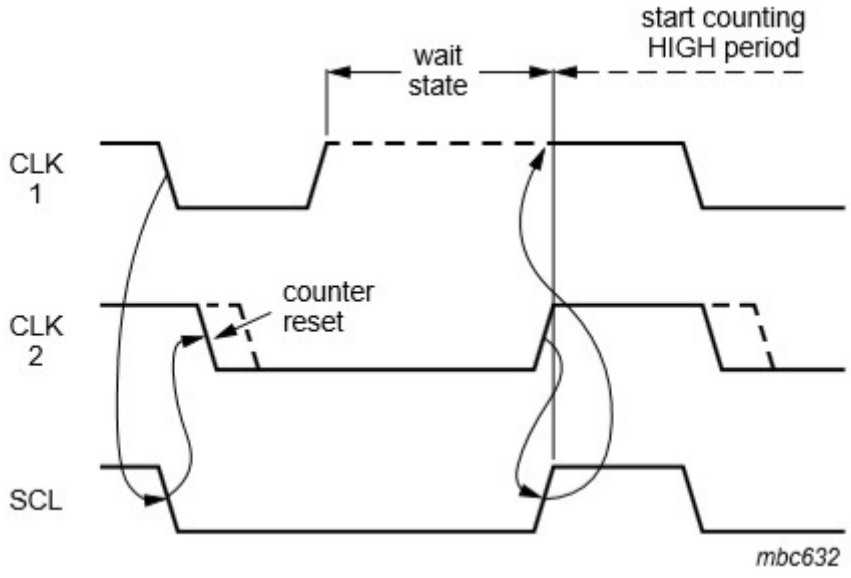


L'arbitragio nell' I2C

L'arbitraggio è la funzione che gestisce il possesso del BUS per evitare ambiguità quando più master richiedono contemporaneamente il suo utilizzo; l'arbitro decide a quale dispositivo concederlo per prima.
Due master possono iniziare a trasmettere dati contemporaneamente su un bus libero, per farlo ci devono essere dei metodi per decidere chi tra i due prenda il controllo del BUS e porti a termine la propria trasmissione. I metodi qui utilizzati prendono il nome di clock synchronization e arbitration. Ovviamente in caso di singolo master di tali metodi non ve n'è bisogno.

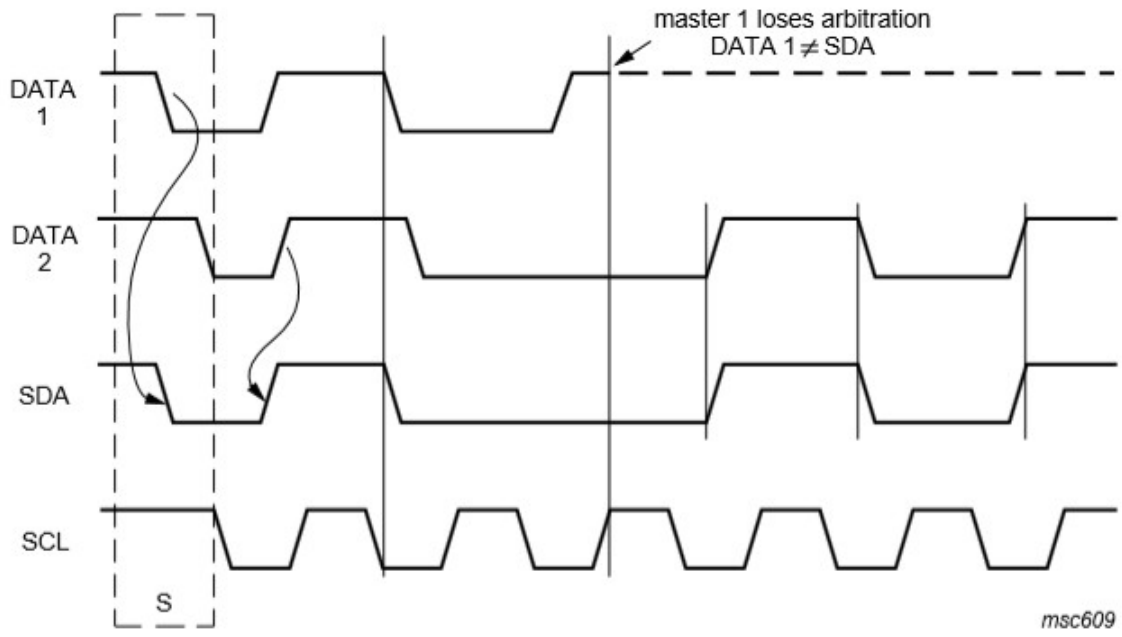
Clock synchronization

Nella comunicazione tra un master e uno slave, possono insorgere due condizioni potenzialmente dannose per il corretto scambio di dati:
 *a livello di byte, lo slave è in grado di ricevere i bytes provenienti dal master alla velocità da questo impostata, ma necessita di un certo tempo per gestire quanto ricevuto oppure per preparare i bytes da trasmettere al master, tempo che non viene concesso dal master che continua a comandare la linea SCL alla sua frequenza;
 *a livello di bit, lo slave potrebbe non essere sufficientemente veloce nelle sue operazioni interne per mantenere la sincronizzazione con il master.
Per porre rimedio a questa situazione si usa il metodo del clock synchronization.
La sincronizzazione dei segnali sfrutta il criterio di "prevalenza" fra i valori '1' e '0' ovvero il segnale scl viene posto a 1 solo quando entrambi i valori dei clock valgono '1' mentre vale zero anche solo se uno dei due clock si imposta a '0'. Lo slave quindi, in condizioni critiche, può mantenere bloccata a livello basso la linea 'SCL' forzando il master in uno stato di attesa, il quale non può procedere all'emissione di un livello alto sulla linea SCL finchè questa viene tenuta a '0' da un altro dispositivo. Pertanto nel primo caso lo slave mantiene a livello basso SCL finchè non ha correttamente riconosciuto il byte arrivato o prodotto il byte da trasmettere. Nel secondo caso lo slave mantiene a livello basso SCL, estendendone il semiperiodo, e in definitiva rallentando il clock del bus: la velocità del master è così adattata alla velocità interna dello slave più lento.
La figura sottostante mostra il funzionamento.



Arbitraggio

Un master può iniziare una comunicazione solamente se il bus è libero; due o più master potrebbero però generare la condizione di START “in contemporanea”. In tal caso ha inizio un processo, denominato arbitraggio, che opera sulla linea SDA quando SCL si trova a livello alto. Il processo, che sfrutta l’and cablato, consiste nel paragonare ciò che si trasmette con quello che effettivamente si trova sulla linea SDA. Quando due dispositivi trasmettono due livelli differenti, quello prevalente risulta essere il livello basso. Se un master trasmette su SDA un livello ‘0’ e un altro trasmette su SDA un livello ‘1’, quest’ultimo disabilita il suo stato d’uscita, poiché sulla linea vede un valore diverso rispetto a quello che tenta di trasmettere.
In figura il funzionamento dell'arbitraggio nell'I2C



LA LIBRERIA

Il file udoosensor.cpp contiene la libreria per il sensore di temperatura e pressione.

```
int udooSensor::udooSensor_INIT(unsigned int I2C_ADDR){

fd=0;
 /* check if the device is connected to the board */

 if ((fd = open(PATH_DEV , O_RDWR)) < 0 ){
                                cout<<fd<<endl;
                                perror("Can't find the device\n");
                                exit(1); }

/* check the comunication with the connected device */
          if ((ioctl(fd, I2C_SLAVE, I2C_ADDR)) < 0){
                                perror("Can't talk with the device\n");
                                exit(1);
                        }

          if (I2C_ADDR==udooSensor::I2C_ADDRESS){


  /* Enable Data Flags in PT_DATA_CFG */

          i2c_smbus_write_word_data(fd, udooSensor::PT_Data_Config_Reg, 0x07);
```

0x13	PT Data Configuration Register (**PT_DATA_CFG**)(1)(3)	0x00	No	R/W	0x14	Data event flag configuration

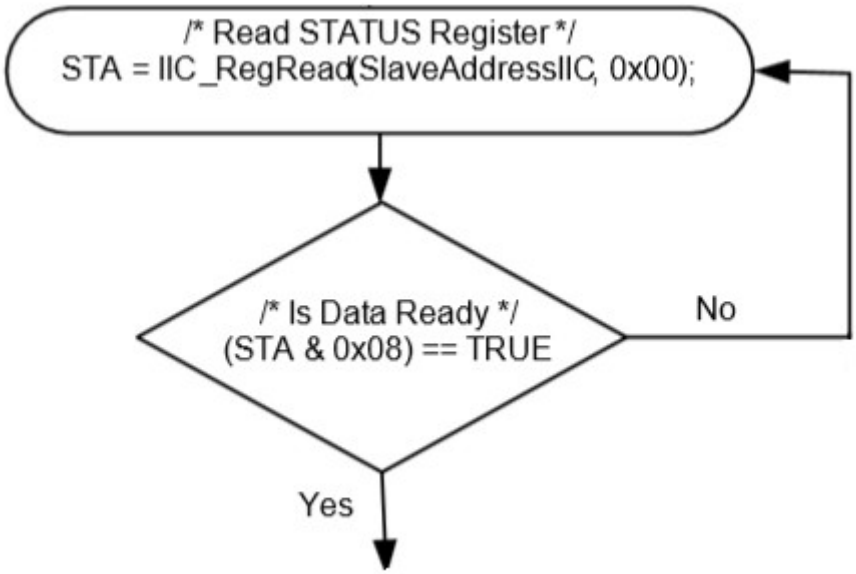
```
/*  Set One Shot *//* Set Active */

          i2c_smbus_write_word_data(fd, udooSensor::CTRL_REG1, 0xB9);
```

Note: Except for standby mode selection and OST (One Shot Mode), the device must be in STANDBY mode to change any of the fields within CTRL_REG1 (0x26).

```
          }

return fd; };

          void udooSensor::GetRawData(int fd, unsigned int I2C_ADDR){

                    cout << "Reading from "<< PATH_DEV << " at address: 0x"<<  I2C_ADDR <<endl;

                    if(I2C_ADDR==udooSensor::I2C_ADDRESS){

                    /* the data oversampling ratio, High Resolution Mode, oversample = 128 */

                   i2c_smbus_write_word_data(fd, udooSensor::CTRL_REG1, (0x38 | 0x01 | 0x00));
```

0x26	Control Register 1 (**CTRL_REG1**)(1)(4)	0x00	No	R/W	0x27	Modes, Oversampling

```
STATUS = 0;

/* check DATA is available on the bus  */
do{
        STATUS = i2c_smbus_read_byte_data(fd, udooSensor::status);

        } while((STATUS & 0x08) == 0);
```



screenshot del datasheet per la decodifica dell'utilizzo del sensore

```
buffer_Press[0] = i2c_smbus_read_word_data(fd, udooSensor::pressure_MSB); //pressure_MSB
buffer_Press[1] = i2c_smbus_read_word_data(fd, udooSensor::pressure_CSB); //pressure_CSB
buffer_Press[2] = i2c_smbus_read_word_data(fd, udooSensor::pressure_LSB); //pressure_LSB

cout<<"Pressure raw data get from register: 0x"<< buffer_Press[0]<<buffer_Press[1]<< buffer_Press[2]<<endl;
buffer_Temp[0] = i2c_smbus_read_word_data(fd, udooSensor::temperature_MSB); //temperature_MSB
buffer_Temp[1] = i2c_smbus_read_word_data(fd, udooSensor::temperature_LSB); //temperature_LSB

//la funzione che setta e ricostruisce  il valore della temperatura all'interno della classe
        void udooSensor::SetTemperature(int fd, unsigned int I2C_ADDR){
//riordino i bit per la temperatura
data_order =  buffer_Temp[0];
data_order = data_order << 8; //SLL
data_order = data_order | buffer_Temp[1]; // OR inclusivo
data_order = data_order >> 4; //SRL
udooSensor::temperature  = data_order * 0.062500;
```

Table 17. OUT_T_MSB Register

	7	6	5	4	3	2	1	0
R	TD11	TD10	TD9	TD8	TD7	TD6	TD5	TD4
W								
Reset	0	0	0	0	0	0	0	0

Table 18. OUT_T_LSB Register

	7	6	5	4	3	2	1	0
R	TD3	TD2	TD1	TD0	0	0	0	0
W								
Reset	0	0	0	0	0	0	0	0

```
void udooSensor::SetPressure_ALT(int fd, unsigned int I2C_ADDR){
//ricostruisco e setto il valore della pressione

buffer_Press[0] = i2c_smbus_read_word_data(fd, udooSensor::pressure_MSB); //P_MSB
buffer_Press[1] = i2c_smbus_read_word_data(fd, udooSensor::pressure_CSB); //P_CSB
buffer_Press[2] = i2c_smbus_read_word_data(fd, udooSensor::pressure_LSB); //P_LSB

//riordino i bit per la pressione
data_order1=0;
pressure = 0.0;

data_order1 = buffer_Press[0];
data_order1 <<= 12;

CSB_BIT = buffer_Press[1];
CSB_BIT <<= 4;
data_order1 |= CSB_BIT;

LSB_BIT = buffer_Press[2];
LSB_BIT >>= 4;
data_order1 |= LSB_BIT;

pressure = data_order1 * 0.00025; // Moltipiplicare per il fattore  di scala (0.0002)
 // per ottenere kPa

udooSensor::pressure=pressure;
```

Table 14. OUT_P_MSB Register

	7	6	5	4	3	2	1	0
R	PD19	PD18	PD17	PD16	PD15	PD14	PD13	PD12
W								
Reset	0	0	0	0	0	0	0	0

Table 15. OUT_P_CSB Register

	7	6	5	4	3	2	1	0
R	PD11	PD10	PD9	PD8	PD7	PD6	PD5	PD4
W								
Reset	0	0	0	0	0	0	0	0

Table 16. OUT_P_LSB Register

	7	6	5	4	3	2	1	0
R	PD3	PD2	PD1	PD0	0	0	0	0
W								
Reset	0	0	0	0	0	0	0	0

IL MONITOR

La parte del programma che si occupa del monitoraggio e ricezione dei dati dei sensori salvandoli in un file csv e mettendoli a disposizione sulla pagina web è il file monitor.cpp.

```
//creazione delle istanze degli oggetti
//smonta il modulo per prenderne il possesso
system ("rmmod mpl3115");

        Birra *peroni;
        peroni = new Birra();

        udooSensor *sensore_udoo;
        sensore_udoo = new udooSensor;

        SerialUdoo *comunication ;
        comunication = new SerialUdoo;


/*############-sensorUDOO init-##############*/
        FD_u=sensore_udoo-> udooSensor_INIT(0x60);

/*################# init serial comunication ############# */
        FD_a=comunication->open_port();
        control=comunication->set_interface_attribs(FD_a,B115200,0);

//ricezione dati

sensore_udoo->GetRawData(FD_u,0x60);
                sensore_udoo->SetTemperature(FD_u,0x60);
                sensore_udoo->SetPressure_ALT(FD_u,0x60);
                temp=sensore_udoo->getTemperature();
                press=sensore_udoo->getPressure();

//per evitare errori di lettura sulla seriale da parte dell'A9

do{
        FD_a=comunication->read_port(buffer_alcohol,FD_a);
        alcohol_val=atof(buffer_alcohol);

   } while(alcohol_val < 100.0  && alcohol_val > 1023  );
```

// scrivo i dati nel file

	A	B	C	D	E	F	G
1	Temperature:	Alcohol:	Pressure:	Hour:	Minute:	Second:	Date:(Y.M.D)
2	23.3125	12	96.6225	12	1	44	15/2/2016
3	23.3125	0	96629	12	1	56	15/2/2016
4	23.3125	9	96.6277	12	2	8	15/2/2016
5	23375	16	96.6263	12	2	19	15/2/2016
6	23.5	0	96.6275	12	2	31	15/2/2016
7	23.5625	11	96.6335	12	2	43	15/2/2016
8	23875	12	96628	14	5	0	15/2/2016
9	23875	0	96.6275	14	5	2	15/2/2016
10	23875	9	96.6298	14	5	5	15/2/2016
11	23875	0	96.6298	14	5	8	15/2/2016
12	23875	6	96.6318	14	5	11	15/2/2016
13	23875	48	96627	14	5	14	15/2/2016
14	23875	0	96.6305	14	5	17	15/2/2016
15	23875	17	96.6302	14	5	20	15/2/2016
16	23875	0	96.6308	14	5	23	15/2/2016
17	23875	8	96629	14	5	26	15/2/2016
18	23875	26	96629	14	5	29	15/2/2016
19	23875	0	96.6288	14	5	32	15/2/2016
20	23875	45	96.63	14	5	35	15/2/2016
21	23.9375	0	96.6285	14	5	38	15/2/2016

SCREENSHOT DELLA SIMULAZIONE

immagine del sito

immagine dell'esecuzione del programma

RIFERIMENTI BIBLIOGRAFICI

Materiale di consultazione per la stesura della relazione sul protocollo I2C:

http://www2.units.it/marsi/elettronica2/tesine_elettro/i2c/i2c.pdf
http://www.nxp.com/documents/user_manual/UM10204.pdf
http://i2c.info
http://www.datsi.fi.upm.es/docencia/Micro_C/i2c.pdf
https://learn.sparkfun.com/tutorials/i2c

Materiale di consultazione per la realizzazione della libreria:

http://www.biteyourconsole.net/2013/01/08/impariamo-a-programmare-guida-come-creare-una-libreria-in-c
http://www.freemedialab.org/wiki/doku.php?id=programmazione:c:creare_librerie
http://www.nxp.com%2Ffiles%2Fsensors%2Fdoc%2Fdata_sheet
%2FMPL3115A2.pdf&h=GAQEbICWl
https://www.sparkfun.com%2Fdatasheets%2FSensors%2FMQ-3.pdf&h=GAQEbICWl

Materiale di consultazione per la creazione del monitor:

http://www.udoo.org/tutorial/vnc-server-with-udoo/
http://www.udoo.org/docs-neo