LABORATORIO DIDATTICO 6 Collegamento wireless di due sensori di temperatura con Arduino e invio dei dati a una rete locale

Per monitorare la temperatura in diversi ambienti di una casa si vuole realizzare una rete di sensori di temperatura (per semplicità solo due sensori), collegati wireless con il dispositivo centrale, a sua volta connesso a una rete locale Ethernet.

Soluzione

Si sceglie di gestire la rete mediante una scheda Arduino, dotata di Ethernet Shield per il collegamento alla rete locale, e di raccogliere i dati dai sensori mediante moduli radio XBee; lo schema a blocchi è rappresentato nella Figura 1.

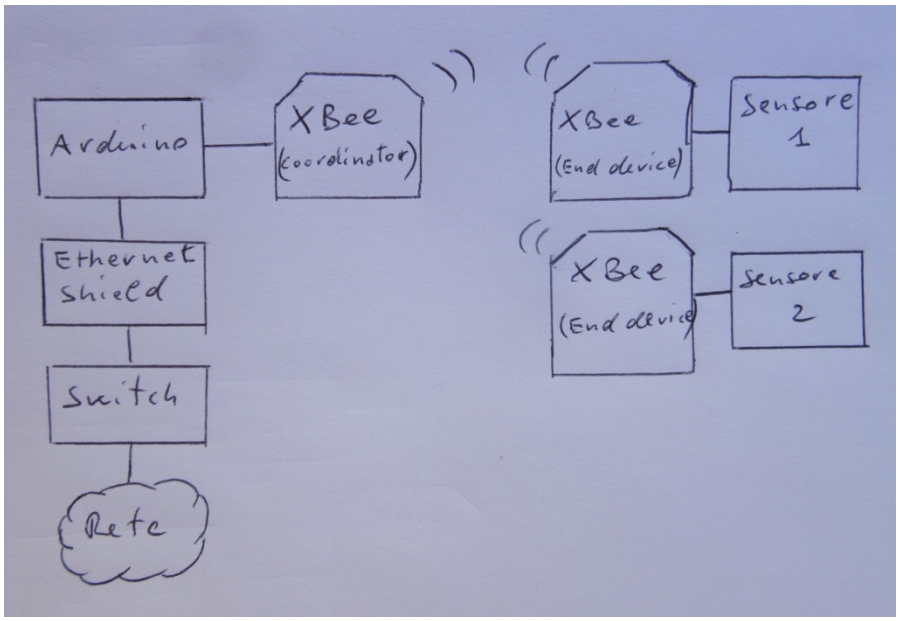

Figura 1 – Struttura a blocchi della rete di sensori di temperatura.

L'XBee del nodo centrale deve essere impostato, mediante il software XCTU, come *coordinator* API, mentre quelli remoti vanno configurati come *end device*:

Ogni *dispositivo remoto* (Figura 2) è costituito da un sensore di temperatura LM35 e un modulo XBee (configurato come *end device*) inserito su una scheda *SparkFun XBee Explorer Regulated*, alimentati con una tensione V_{cc} compresa tra 4 e 5 V.
La tensione d'uscita del sensore, che ha una sensibilità di 10 mV/°C, varia da 0 V a 1,2 V nel campo di temperature 0 °C - 120 °C, quindi può essere inviata direttamente al convertitore ADC di XBee, che ha un campo di lavoro 0 V - 1,2 V. Usiamo l'ingresso analogico AD1 (pin 19), da configurare opportunamente tramite XCTU.

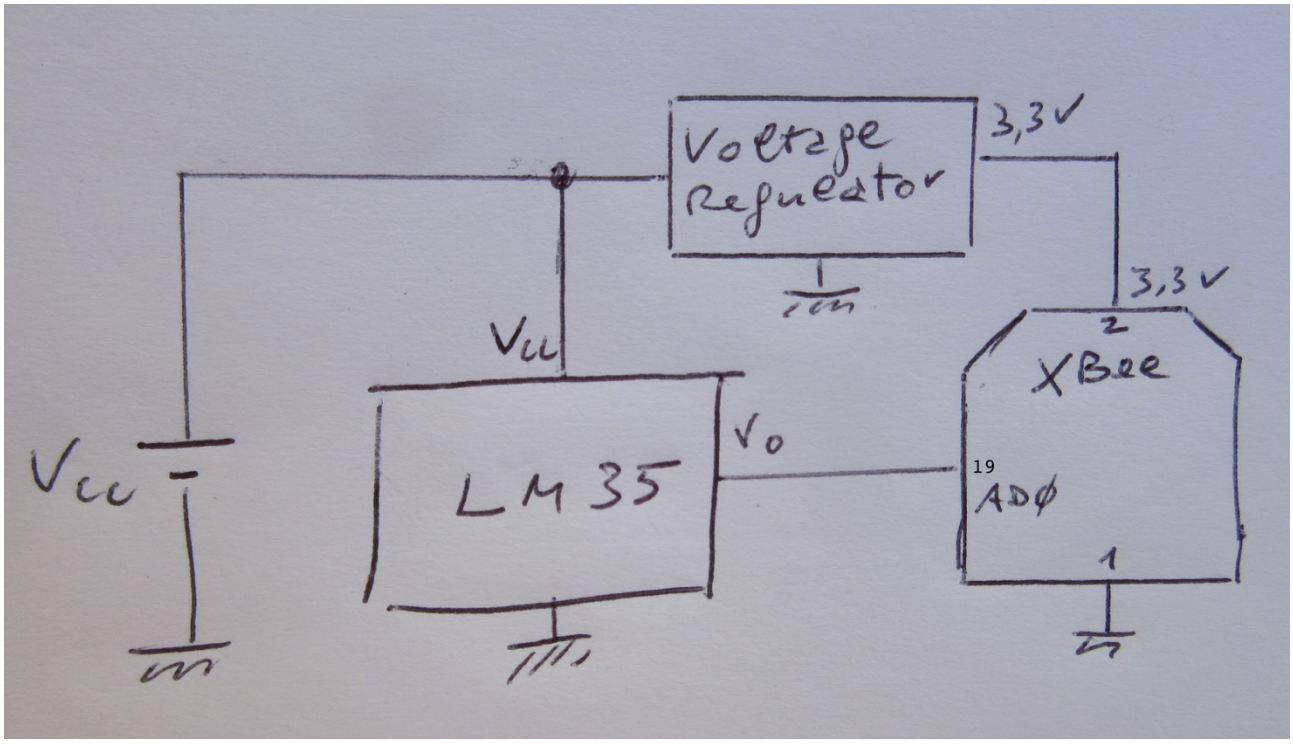

Figura 2 – Schema elettrico di un dispositivo remoto.

Il *nodo centrale* è costituito da Arduino, dotato di Ethernet Shield per il collegamento alla rete locale e una XBee Shield su cui è alloggiato l'XBee configurato come *coordinator*.
Nell'esercitazione 1 la comunicazione avveniva solo tra due XBee e si è quindi utilizzata la configurazione AT (trasparente), già predisposta per default negli XBee Series 1.
Per realizzare una rete è invece necessario utilizzare la modalità API (*Application Programming Interface*), in cui i dati trasmessi sono organizzati in *frame* che consentono, tra l'altro, di individuare i nodi con un indirizzo e di rivelare gli errori di trasmissione.
La struttura di un *frame* è rappresentata nella Figura 3 ed è costituita dai seguenti campi:

- 1 byte di start ($7E_H = 0111110_2$);
- 2 byte che contengono il numero dei byte che seguono (byte di dati più quello di *checksum*);
- *n* byte di dati (lunghezza variabile): in base alla prima parte del campo (CMD ID) la parte restante (CMD DATA) può contenere informazioni di routing (indirizzi di partenza e destinazione, richieste di conferma trasmissione, ecc..), eventuali comandi (richieste di trasmissione, comandi di configurazione) o dati veri e propri, come letture di sensori;
- 1 byte di controllo (*checksum*) per la rivelazione degli errori di trasmissione: il byte è ottenuto sottraendo al numero $FF_H = 1111111_2$, gli 8 bit meno significativi risultanti dalla somma degli *n* byte di dati più quello di *checksum*.

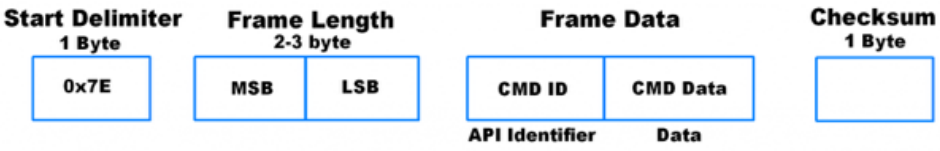

Figura 3 – Struttura di un frame API.

Configurando i tre moduli tramite il software XCTU si deve impostare per ognuno:

- il ruolo (un *coordinator API* e due *end device API*);

- la rete (stesso valore di PAN ID);
- gli indirizzi DL e DH: assegnare valori scambiati, per farli comunicare tra loro;
- il tempo di campionamento IR (l'intervallo di tempo tra due letture del sensore);
- configurare il pin AD1 come ingresso analogico;
- per aumentare la potenza e quindi la portata settare i parametri "*Power Mode*" con "*boost enable*" e "*Power Level*" con "HIGH".

La tabella in Figura 4, tratta dalla XBee Quick Reference Guide, riporta il significato dei byte che costituiscono un frame contenente dati (cioè di tipo 0x92, come specificato dal byte 3)

API format for I/O Data Sample RX Indicaton	Byte	Example	Description
	0	0x7e	Start byte – Indicates beginning of data frame
	1	0x00	Length – Number of bytes (ChecksumByte# – 1 – 2)
	2	0x14	
	3	0x92	Frame type - 0x92 indicates this will be a data sample
	4	0x00	64-bit Source Address (Serial Number)
	5	0x13	MSB is byte 4, LSB is byte 11
	6	0xA2	
	7	0x00	
	8	0x40	
	9	0x77	
	10	0x9C	
	11	0x49	
	12	0x36	Source Network Address – 16 Bit
	13	0x6A	
	14	0x01	Receive Opts. 01=Packet Acknowledged. 02=Broadcast packet
	15	0x01	Number of sample sets. Always set to 1 due to XBEE limitations
	16	0x00	Digital Channel Mask – Indicates which pins are set to DIO
	17	0x20	
	18	0x01	Analog Channel Mask – Indicates which pins are set to ADC
	19	0x00	Digital Sample Data (if any) – Reads the same as Digital Mask
	20	0x14	
	21	0x04	Analog Sample data (if any)
	22	0x25	There will be two bytes here for every pin set for ADC
	23	0xF5	Checksum(0xFF - the 8 bit sum of the bytes from byte 3 to this byte)

Figura 4 –Significato dei 24 byte che costituiscono un frame contenente dati (cioè di tipo 0x92

Il significato dei byte più importanti è il seguente (si confronti con la Figura 5 che rappresenta la stampa su Serial Monitor dei cinque frame ricevuti in cinque letture alternate dei due sensori):

- **byte 0**: start ($7E_H = 0111110_2$);
- **byte 1 - 2**: numero dei byte di dati più quello di *checksum*; infatti nella colonna 2 di Figura 5 si rileva $12_H=18_{10}$, tolti i primi 3 byte i restanti sono 18 (si faccia attenzione che mancano i byte 19 e 20 perché nella *Digital Channel Mask* non è attivato nessun input digitale);

- **byte 3**: tipo di frame; $0x92=92_H$ ® frame contenente dati;
- **byte 4 - 11**: Serial Number del trasmettitore (64 bit) (ogni XBee ha un SN differente); in Figura 5 il byte 11 distingue i due XBee che stanno trasmettendo i dati alternativamente ($E0_H$ e DF_H);
- **byte 12 - 13**: indirizzo di rete del trasmettitore;
- **byte 16 - 17**: maschera dei pin settati come digitali; nel nostro caso (Figura 5) nessun ingresso è settato come digitale (0_H e 0_H)
- **byte 18**: maschera dei pin settati come analogici: nel nostro caso (Figura 5) il byte 18 ha valore $2_H=00000010_2$ e indica che A1 (pin 19) è settato come ingresso analogico;

- **byte 19 - 20**: riportano i valori rilevati sugli ingressi digitali; nel nostro caso (Figura 5) sono assenti perché la maschera dei byte 16 e 17 è tutta a zero;
- **byte 21 - 22**: rappresentano i valori analogici rilevati, una coppia di byte (MSB e LSB) per ogni ingresso analogico attivato;
- **byte 23**: *checksum*.

0	1	2	**3**	4	5	6	7	8	9	10	**11**	12	13	14	15	16	17	**18**	21	**22**	23
7E,	0,	12,	92,	0,	13,	A2,	0,	40,	F4,	A0,	E0,	A3,	95,	1,	1,	0,	0,	2,	0,	4C,	7C,
7E,	0,	12,	92,	0,	13,	A2,	0,	40,	F4,	A0,	DF,	99,	15,	1,	1,	0,	0,	2,	0,	67,	EC,
7E,	0,	12,	92,	0,	13,	A2,	0,	40,	F4,	A0,	E0,	A3,	95,	1,	1,	0,	0,	2,	0,	5C,	6C,
7E,	0,	12,	92,	0,	13,	A2,	0,	40,	F4,	A0,	DF,	99,	15,	1,	1,	0,	0,	2,	0,	70,	E3,
7E,	0,	12,	92,	0,	13,	A2,	0,	40,	F4,	A0,	E0,	A3,	95,	1,	1,	0,	0,	2,	0,	5A,	6E,

Figura 5 – Stampa sul Serial Monitor dei frame ricevuti in cinque letture alternate dei due sensori (mancano i byte 19 e 20 perché nella *Digital Channel Mask* non è attivato nessun input digitale).

Si riporta lo **sketch** caricato su Arduino con i relativi commenti.

```
#include <SPI.h>              // si includono le librerie
#include <Ethernet.h>

float temp1, temp2, v, vref=1.2;
int i, msb, lsb, data1, data2, err=0, dis, ind;
char data_rx;              // variabile char per memorizzare i byte letti dal client
byte mac[] = { 0xDE, 0xAD, 0xBE, 0xEF, 0xFE, 0xED };
IPAddress ip(10,0,0,111);        // indirizzo della shield Ethernet

EthernetServer arduino_server(80);   // creo un oggetto server che rimane in ascolto
                                     // sulla porta specificata (80)
EthernetClient pc_client;

void setup() {
 Serial.begin(9600);
 Ethernet.begin(mac, ip);   // inizializzo il chip WIZnet con il mac e l'ip
 arduino_server.begin();   // avvio l'oggetto server per eventuali richieste dal client
}

void loop() {
 if (Serial.available()>21) {      // controllo se è arrivato tutto il frame
  if (Serial.read()==0x7E) {     // controllo se il primo byte è 7E

    for(i=1; i<11; i++)  dis=Serial.read();   // leggo ed elimino 10 byte
     ind=Serial.read();     // leggo e salvo l'undicesimo (serial number del tx)

    for(i=1; i<8; i++)  dis=Serial.read();     // leggo ed elimino 7 byte
     msb=Serial.read();  // leggo e salvo l'ottavo (byte 21, MSB della temperatura)
     lsb=Serial.read();    // leggo e salvo il nono (byte 22, LSB della temperatura)
```

```
   if(ind==0xDF) {        // se l'indirizzo è quello del primo sensore
     data1=msb*256 + lsb;              // combino MSB e LSB per ricavare il valore
     temp1=(data1*vref/1023)*100;   // converto il valore in temperatura
   }
   else {                    // se l'indirizzo è quello del secondo sensore
     data2=msb*256 + lsb;
     temp2=(data2*vref/1023)*100;}
    }
  }
        // leggi client con richiesta
pc_client=arduino_server.available();
  if (pc_client) {
   while(pc_client.connected())  {   // mentre è connesso ripeti
    if (pc_client.available()) {   // controllo se ci sono byte disponibili per la lettura
      data_rx=pc_client.read();    // se ci sono li leggo

          // verifica fine richiesta client
      if (data_rx=='\n')  {  // attendo che tutti i byte siano stati letti, se data_rx contiene
                             // il carattere "new line" significa che tutti i byte sono arrivati

          // scrivo pagina HTML sul client
      pc_client.print("<html>");
      pc_client.print("<head> <title> Server Web Arduino </title></head>");
      pc_client.print("<body bgcolor='#87CEEB'>");
      pc_client.print("<p align='center'>");
      pc_client.print("<h1><p align='center'>Lettura temperature da sensori remoti via xbee </h1>");
      pc_client.print("<br>");
      pc_client.print("<p align='center'><table width='50%' border='3'>");
      pc_client.print ("<tr><td><h2><p align='center'> N Sensore </h2></td> ");
      pc_client.print("<td><h2><p align='center'> Temperatura C </h2> </td> ");
      pc_client.print("</tr>");
      pc_client.print ("<tr>");
      if(temp1>27) {  // se la temperatura del sensore 1 è >27°C, colora in rosso la cella html
        pc_client.print ("<tr bgcolor='red'>");
        pc_client.print(" <td > <h2><P align='center'> Sensore 1 </h2> </td> ");
        pc_client.print("<td> <h2><P align='center'>");
        pc_client.print(temp1);
      }
      else   {         // se <27°C ripristino standard della cella html sensore 1
        pc_client.print ("<tr bgcolor='#87CEEB'>");
        pc_client.print("<td > <h2><P align='center'> Sensore 1 </h2> </td> ");
        pc_client.print("<td> <h2> <P align='center'>");
        pc_client.print(temp1);
      }
      if(temp2>27) {   // se la temperatura del sensore 2 è >27°C, colora in rosso la cella html
        pc_client.print ("<tr bgcolor='red'>");
        pc_client.print(" <td > <h2><P align='center'> Sensore 2 </h2> </td> ");
        pc_client.print("<td> <h2><P align='center'>");
        pc_client.print(temp2);
      }
```

```
      else {        // se <27°C ripristino standard della cella html sensore 2
        pc_client.print ("<tr bgcolor='#87CEEB'>");
        pc_client.print("<td > <h2><P align='center'> Sensore 2 </h2> </td> ");
        pc_client.print("<td> <h2><P align='center'>");
        pc_client.print(temp2);
        pc_client.print("</td>");
        pc_client.print("</tr>");
       }
       pc_client.print("</table>");
       pc_client.print(" </body>");
       pc_client.print("</html>");
       break;
      }
    }
  }
  delay(5); // pausa per dare tempo al web browser di ricevere i dati
  pc_client.stop();    // chiudi la connessione
 }
}
```