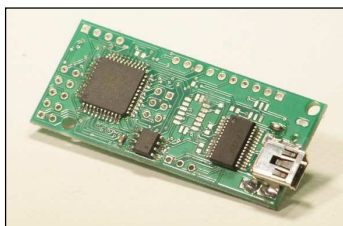


MODULO PROGRAMMABILE I/O

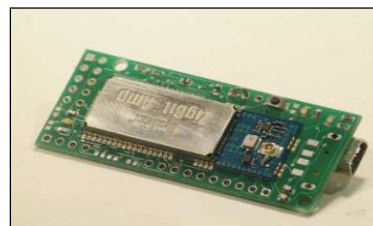
MANUALE PER LA PROGRAMMAZIONE



MZA-30



MZBi-30



MZBa-30

| | |
|---------------------------------------|----|
| INTRODUZIONE: | 3 |
| Programmazione in MZ-BASIC..... | 3 |
| Il linguaggio MZ Basic | 4 |
| Regole del linguaggio..... | 5 |
| Istruzioni Set | 6 |
| Sintassi MZ-Basic..... | 6 |
| Commenti | 6 |
| Dichiarazioni di Variabili..... | 7 |
| Assegnazione variabili | 7 |
| Operatori..... | 7 |
| Funzione Lineare | 9 |
| Etichette..... | 9 |
| JUMP | 10 |
| Costrutto IF | 10 |
| WAIT..... | 13 |
| Comunicazione seriale | 13 |
| Utilizzo periferiche su bus I2C | 14 |
| Comunicazione di Rete: | 16 |
| Lo standard ZigBee | 16 |
| Implementazione in MZ_Basic..... | 18 |
| LINGUAGGIO C | 21 |
| Sviluppo nativo su MZBI/A-30 | 21 |
| Sviluppo nativo su MZA30..... | 22 |

| | |
|---|----|
| Installazione librerie..... | 23 |
| Documentazione Librerie..... | 23 |
| APPENDICI | 24 |
| Tabella 1 - Variabili..... | 24 |
| Tabella 2 – Error Register..... | 25 |
| 1.1 - Installazione del Framework..... | 26 |
| Ambiente GNU/Linux..... | 26 |
| Installazione del Framework in ambiente Windows | 28 |

INTRODUZIONE:

Scopo del presente manuale e' quello di fornire le istruzioni necessarie per potere programmare i moduli MZA-30, MZBi-30 e MZBa-30. Tutti i moduli possono venire programmati in due differenti modalit :

- ➔ **MZ BASIC:** Linguaggio semplice, ideale per utilizzo nella didattica e per prendere familiarit  con i moduli. Si utilizza una **Virtual Machine** precaricata e il codice di programmazione e' Basic-Like; grazie ad una semplice interfaccia grafica   possibile:
 - Compilare il codice rendendolo compatibile con la **Virtual Machine** pre-installata
 - Scrivere il codice sui moduli collegati tramite la porta USB
- ➔ **LINGUAGGIO C:** dedicata ai programmatori piu' esperti, viene programmata in linguaggio C utilizzando le specifiche ATMEL. In questo modo si pu  programmare a proprio piacere il microprocessore, sfruttando a pieno tutte le sue potenzialit .

Programmazione in MZ-BASIC

Tutti i moduli venduti hanno precaricata l'ultima versione di **Virtual Machine** necessaria per interpretare i comandi. Per ripristinarla consultare il capitolo dedicato "**Installazione del Framework**"

  possibile utilizzare funzionalit  wireless basata sul protocollo standard di comunicazione di rete ZigBee per poter comunicare con altri dispositivi presenti nella rete, in modo altrettanto semplice. Questa funzionalit  e' supportata esclusivamente dai moduli MZBi-30 e MZBa-30.



Nel DATASHEET dei moduli MZ   descritto come utilizzare il compilatore per aggiornare i moduli con i nuovi firmware. In questo documento sono contenuti solamente le specifiche di programmazione.

Il linguaggio MZ Basic

Il linguaggio **MZ Basic** è un linguaggio molto simile al BASIC e permette di utilizzare più di un centinaio di registri (che vengono visti dal codice come variabili) che permettono di accedere in modo semplice ed immediato a svariati dispositivi hardware come ADC, digital I/O, canali PWM, semplicemente leggendo o scrivendo un valore nel dato registro.

Seguono alcuni comandi principali per gestire il modulo MZ. L'elenco completo è nell'APPENDICE 2):

| Nome | Descrizione |
|-----------------------------|-----------------------------|
| VAR 'nome' AS 'tipo' | Definizione variabile |
| AVR_IN_0 .. AVR_IN_4 | Ingressi analogici AVR 1..4 |
| AVR_IO_0 .. AVR_IO_9 | AVR GPIO 1..10 |

Ecco qualche esempio:

- Per **definire una variabile**: `VAR "NomeVariabile" AS SHORT oppure VAR "NomeVariabile" AS AVR_IN_0`
- Per **assegnare** il valore 100 alla variabile: `"NomeVariabile" = 100`
- Per **mettere allo stato logico 1 un piedino di GPIO** (ingresso-uscita digitale) ed accendere un eventuale LED collegato, basta scrivere, una volta definita la variabile del tipo AVR_IO_X, `C1 = 1` Per spegnerlo basta metterlo a 0: `C1 = 0`
- Per **leggere il segnale analogico** presente sul terzo canale, basta definire due variabili una di tipo AVR_IN l'altra di tipo generico SHORT quindi scrivere: `"NomeVariabileGenerica" = "NomeVariabileIN"`

Segue un primo semplice esempio di codice :

```
# Prima prova utilizzo VM per accndere e spegnere un LED

# Definizione variabile associata al piedino 1
VAR ControlloLED AS AVR_IO_0

# Accensione LED
ControlloLED = 1
```

```
# Pausa di 1000 ms
```

```
WAIT 1000
```

```
# Spegnimento LED
```

```
ControlloLED = 0
```

```
# fine del codice
```

Regole del linguaggio

Ecco le poche, ma essenziali regole da seguire:

- Tutti i COMANDI devono essere in MAIUSCOLO
- Ogni elemento presente nella linea di comando (variabile, operatore, registro, ..) deve separarsi dagli altri con un carattere vuoto *SPAZIO*
 - $x=1$ → **ERRORE**
 - $x =1$ → **ERRORE**
 - $x = 1$ → **CORRETTO**
- L'indentazione non e' necessaria. Se si vuole indentare usare gli SPAZI e non i TAB
- Prima di utilizzare *variabili* o *registri* vanno dichiarati con il comando VAR

Istructions Set

Segue l'elenco completo delle istruzioni utilizzabili in MZ-BASIC.

| Nome | V/C/E* | Semantica |
|------------------|--------|--|
| COMMENTO | | # commento |
| Assegnazione | V | <var1> = <var2> |
| Assegnazione | C | <var> = <const> |
| Somma | V | <var1> = <var1> + <var2> |
| Somma | C | <var> = <var> + <const> |
| Sottrazione | V | <var1> = <var1> - <var2> |
| Sottrazione | C | <var> = <var> - <const> |
| Moltiplicazione | V | <var1> = <var1> * <var2> |
| Moltiplicazione | C | <var> = <var> * <const> |
| Divisione | V | <var1> = <var1> / <var2> |
| Divisione | C | <var> = <var> / <const> |
| Incremento | V | <var> = <var> + 1 |
| Decremento | V | <var> = <var> - 1 |
| And logico | V | <var1> = <var1> & <var2> |
| And logico | C | <var> = <var> & <const> |
| Or logico | V | <var1> = <var1> <var2> |
| Or logico | C | <var> = <var> <const> |
| Not logico | V | <var> = ~<var> |
| Funzione Lineare | V,C | <var1> = <var1> * <const1> / <const2> + <const3> |
| Aspetta (delay) | V | WAIT <var> |
| Aspetta (delay) | C | WAIT <const> |
| Salta | E | JUMP <etichetta> |
| Chiama | V | CALL <funzione> |
| Trasmetti | V,C | TX <id> AT <channel> VALUE <dato> |

Nota: La colonna V/C/E definisce se l'operazione è effettuata tra variabili (V), tra una variabile ed una costante (C) o nel caso del salto, con una etichetta (E)

Sintassi MZ-Basic

Commenti

I commenti si definiscono inserendo all'inizio della riga di commento il carattere #.

Per commenti da più righe aggiungere all'inizio di ogni riga il carattere #.

ESEMPIO:

```
# Commento valido
```

```
#Commento Valido
```

```
a = b #Commento NON VALIDO
```

Dichiarazioni di Variabili

Le variabili vengono dichiarate usando la seguente sintassi:

VAR <name> AS <type>

<name>: Deve essere un insieme di caratteri alfanumerici (almeno una lettera) compreso anche il carattere “_”, il nome deve cominciare con una lettera

<type>: Il tipo di una variabile deve essere unicamente uno di quelli contenuti in **tabella 1 (paragrafo 3.11)**

ESEMPIO:

```
VAR pippo AS SHORT
VAR s AS NET_CTRL_RX8
VAR 1 AS SHORT (ERRORE nome variabile non può essere 1)
```

Assegnazione variabili

L'assegnazione dei valori alle variabili avviene tramite l'operatore =.

<var> = <var1>

<var> = costante (numero)

ESEMPIO:

```
ciao = 0
pippo = ciao
pippo =ciao ( Errore, manca SPAZIO )
```

Operatori

Di seguito la sintassi per utilizzare gli operatori. **<var>** indica una variabile che deve essere dichiarata per poter essere utilizzata e **<const>** indica una costante numerica.

Operatori Binari

| Operatore | tra variabili: | tra costanti: |
|-----------|----------------|---------------|
|-----------|----------------|---------------|

| | | |
|-----|--|---|
| AND | $\langle \text{var1} \rangle = \langle \text{var1} \rangle \& \langle \text{var2} \rangle$ | $\langle \text{var} \rangle = \langle \text{var} \rangle \& \langle \text{const} \rangle$ |
| OR | $\langle \text{var1} \rangle = \langle \text{var1} \rangle \langle \text{var2} \rangle$ | $\langle \text{var} \rangle = \langle \text{var} \rangle \langle \text{const} \rangle$ |
| NOT | $\langle \text{var} \rangle = \sim \langle \text{var} \rangle$ | |

ESEMPIO:

```

vara = vara & varb
vara = vara & 128
vara = ~ vara
    
```

Operatori Aritmetici

| <i>Operatore:</i> | <i>tra variabili:</i> | <i>tra costanti:</i> |
|-------------------|--|--|
| SOMMA | $\langle \text{var} \rangle = \langle \text{var} \rangle + \langle \text{var} \rangle$ | $\langle \text{var} \rangle = \langle \text{var} \rangle + \langle \text{const} \rangle$ |
| SOTTRAZIONE | $\langle \text{var} \rangle = \langle \text{var} \rangle - \langle \text{var} \rangle$ | $\langle \text{var} \rangle = \langle \text{var} \rangle - \langle \text{const} \rangle$ |
| MOLTIPLICAZIONE | $\langle \text{var} \rangle = \langle \text{var} \rangle * \langle \text{var} \rangle$ | $\langle \text{var} \rangle = \langle \text{var} \rangle * \langle \text{const} \rangle$ |
| DIVISIONE | $\langle \text{var} \rangle = \langle \text{var} \rangle / \langle \text{var} \rangle$ | $\langle \text{var} \rangle = \langle \text{var} \rangle / \langle \text{const} \rangle$ |
| INCREMENTO | $\langle \text{var} \rangle = \langle \text{var} \rangle + 1$ | |
| DECREMENTO | $\langle \text{var} \rangle = \langle \text{var} \rangle - 1$ | |

ESEMPIO:

```

somma = A + B
incremento = A + 1
risultato = A * B + 1
    
```

Funzione Lineare

Permette di esprimere una varietà di funzioni numeriche a condizione che venga mantenuto il seguente costrutto, si rispetchi l'ordinamento degli operatori e che questi non vengano duplicati:

<var> = <var> * <const1> / <const2> + <const3>

ESEMPIO:

```
VAR a AS SHORT
```

```
VAR b AS SHORT
```

```
a = 32
```

```
a = a * 10 / 20 + 1
```

```
a = b * 1 / 2 + 3
```

```
a = b * c * d      Errore: duplicati gli operatori
```

```
a = a * 10 + 1 / 20  Errore: invertito l'ordinamento
```

Etichette

Le etichette possono essere usate per i salti nel codice grazie all'istruzione **JUMP** e alla condizione di VERO del costrutto **IF**. Devono essere definite nel codice tramite la parola chiave **LABEL** seguita dal nome dell'etichetta e dal simbolo ":"

La sintassi è : **LABEL <etichetta>:**

ESEMPIO

```
.. codice ..
```

```
.. codice ..
```

```
JUMP salto:
```

```
.. codice ..
```

```
.. codice ..
```

```
LABEL salto:
```

```
.. codice ..
```

```
.. codice ..
```

JUMP

Come visto in precedenza il comando JUMP serve per effettuare un salto ad una LABEL definita in un'altra area del codice. Vede una equivalenza con il comando GOTO del linguaggio Basic tradizionale. L'utilizzo piu' frequente e' quello di rilanciare l'esecuzione del codice quando si e' arrivati al termine, in modo che non venga eseguito una volta soltanto.

La sintassi è : **JUMP <etichetta>**

ESEMPIO

```
# Inizio codice
.. codice ..
.. codice ..

# Inizio parte ciclica del codice
LABEL Inizio_codice:
.. codice ..
.. codice ..

JUMP Inizio_codice:
```

Costrutto IF

Sono supportati due tipologie di controllo:

- Controllo uguaglianza
- Controllo intervallo, maggiore o minore e differenza

In entrambi i casi ogni istruzione va definita su una singola riga, e l'operazione che segue il **then** deve essere un salto ad una label <etichetta> o una sub <funzione>. Se quindi il controllo effettuato da esito positivo, viene invocata la chiamata ad una LABEL o SUB che tipicamente contengono lo stesso codice che nei linguaggi tradizionali seguono il suffisso *THEN*.

Controllo Uguaglianza

Permette di controllare il valore di una variabile confrontandola con un'altra variabile o con una costante.

Sintassi:

IF <nome_variabile1> = <nome_variabile2> THEN <etichetta>

IF <nome_variabile1> = <costante> THEN <etichetta>

ESEMPIO

```

VAR a AS SHORT
VAR b AS SHORT
a = 10
IF a = b THEN <etichetta>

IF a = 30 THEN <etichetta>

```

Controllo intervallo, maggiore o minore e differenza:

Occorre prestare attenzione agli operatori di confronto utilizzati: **prima il > e poi <**

Sintassi:

IF <var1> > <var2> AND <var3> <var4> THEN <etichetta>

IF <var1> > <costante> AND <var2> <costante> THEN <etichetta>

ESEMPI:

```

VAR a AS SHORT
VAR b AS SHORT
VAR c AS SHORT
VAR d AS SHORT

IF a > b THEN <etichetta>
IF a > 5 THEN <etichetta>
IF c < d THEN <etichetta>
IF a > b AND c < d THEN <etichetta>
IF a > b AND c < 5 THEN <etichetta>
IF a > b AND a < b THEN <etichetta>           → Equivale al <>
IF a < b AND c > d THEN <etichetta>         ERRORE: operatori > e < invertiti !

```

ESEMPIO di Equivalenza con comando IF <condizione> THEN <istruzioni> ELSE <istruzioni>

```

IF <condizione> THEN etichetta_vero

```

```
#se falso esegue questo codice
```

```
....
```

```
JUMP endif
```

```
LABEL etichetta_vero:
```

```
#se vero esegue questo
```

```
...
```

```
LABEL endif:
```

WAIT

Consente di mettere in attesa il programma per il numero di millisecondi specificato.

La sintassi è:

WAIT <var>

WAIT <const>

ESEMPIO

```

VAR Timer AS SHORT
Timer = 50
WAIT Timer
WAIT 10
WAIT Timer + 10  ERRORE: non consentite operazioni

```

Comunicazione seriale

Le MZ hanno una porta seriale a 38400 BAUD 8N1; il canale di accesso possiede una coda di ricezione di 32byte e come nella comunicazione via rete, se viene ricevuto un nuovo byte con coda piena viene scartato il messaggio più vecchio.

Sono disponibili tre registri per la comunicazione seriale:

1. **SERIAL_IO**: Fornisce l'accesso alla porta seriale, ogni byte scritto in questa variabile viene spedito tale e quale sulla porta seriale, ogni lettura di questo registro legge l'ultimo byte presente nella coda di ricezione seriale.
2. **SERIAL_OUT**: ogni 16bit scritti in questo registro vengono spediti via seriale in esadecimale ASCII. Utile per spedire ad una console messaggi di debug.
3. **SERIAL_CTRL_RX**: come i vari NET_CTRL_RX_X di rete, contiene il numero di byte ricevuti dalla porta seriale e presenti in coda.

ESEMPIO Trasmissione Seriale:

```

var a as short
var out as serial_out

label main:
a = 10 * 10 / 5
out = a

```

```
jump main
```

ESEMPIO Ricezione Seriale:

```
#Variabile del registro della porta seriale
var read as SERIAL_IO
#variabile di controllo ricezione
var ctrl as SERIAL_CTRL_RX
var temp as short

label main:
#Appena ricevo un byte via rete
if ctrl = 1 then readSeriale
jump main
#Leggo il byte ricevuto sulla porta seriale
label readSeriale:
temp = read
# In temp ho il byte ricevuto via seriale
jump main
```

Utilizzo periferiche su bus I2C

Il bus I2C puo' essere utilizzato in diverse modalità in base al tipo di dispositivo slave che si vuole controllare. E' possibile scrivere/leggere semplicemente un byte o una word sul bus I2C oppure leggere/scrivere particolari registri del dispositivo I2C da controllare.

Il bus I2C viene gestito attraverso 4 differenti variabili:

- **I2C_MODE**: modalità di gestione del bus I2C (vedi oltre)
- **I2C_ADDRESS**: Indirizzo I2C a 7bit del device da controllare
- **I2C_REG**: Registro da leggere o scrivere
- **I2C_DATA**: Dati (8 o 16bit) da leggere o scrivere.

Le possibili modalità di gestione sono:

- Legge / Scrive 8bit sul bus I2C
 - <start> <send address> <R/W I2C_DATA LSB> <stop>

- Legge / Scrive una word (16bit) sul bus I2C
 - Step: <start> <send address> <R/W I2C_DATA MSB> <R/W I2C_DATA LSB> <stop>
- Legge / Scrive 8bit in registro indirizzato a 8bit sul bus I2C
 - <start> <send address> <W I2C_REG LSB> <restart> <send address> <R/W I2C_DATA LSB> <stop>
- Legge / Scrive 16bit in un registro indirizzato a 8bit sul bus I2C
 - <start> <send address> <W I2C_REG LSB> <restart> <send address> <R/W I2C_DATA MSB> <R/W I2C_DATA LSB> <stop>
- Legge / Scrive 8bit in un registro indirizzato a 16bit sul bus I2C
 - <start> <send address> <W I2C_REG MSB> <W I2C_REG LSB> <restart> <send address> <R/W I2C_DATA LSB> <stop>
- Legge / Scrive 16bit in un registro indirizzato a 16bit sul bus I2C
 - <start> <send address> <W I2C_REG MSB> <W I2C_REG LSB> <restart> <send address> <R/W I2C_DATA MSB> <R/W I2C_DATA LSB> <stop>

Di seguito un esempio di scrittura di un byte, 208, nella cella 10 (registro 10) di una Eeprom I2C con indirizzo 84 e indirizzamento di memoria a 8bit:

```
VAR mode AS I2C_MODE
```

```
VAR address AS I2C_ADDRESS
```

```
VAR memorylocation AS I2C_REG
```

```
VAR data AS I2C_DATA
```

```
VAR temp AS SHORT
```

```
# Imposto l'indirizzo I2C della EEprom
```

```
address = 84
```

```
# Modalità 5: leggo/scrivo 8 bit in un registro indirizzato a 8bit.
```

```
mode = 5
```

```
# Locazione di memoria o registro da gestire.
```

```
memorylocation = 10
```

```
# Scrivo 208 nella locazione di memoria
```

```
data = 208
```

```
# Rileggo dalla eeprom il byte letto.
```

```
temp = data
```

Comunicazione di Rete:

Le schede MZBA30 e MZB30 dispongono di un'interfaccia di rete wireless ZigBee operante a 2.4Ghz.

Lo standard ZigBee

ZigBee intende operare in tutti quei minuti oggetti che potrebbero non avere nulla a che fare con l'informatica e le reti. Oggetti magari semplici e scontati - quali per esempio gli interruttori della luce, le serrature, i sensori ambientali o gli allarmi - a cui fornire la capacità di integrarsi in una rete informatica. A tutti questi oggetti non serve velocità, in quanto hanno poche informazioni da scambiare.

Lo standard ZigBee opera nella banda ISM, la stessa utilizzata da WiFi. Questo garantisce la possibilità di installare gli apparati in maniera libera, senza richiedere autorizzazioni statali. La semplicità in ZigBee implica anche un costo basso per singolo chip.

Caratteristiche salienti del protocollo:

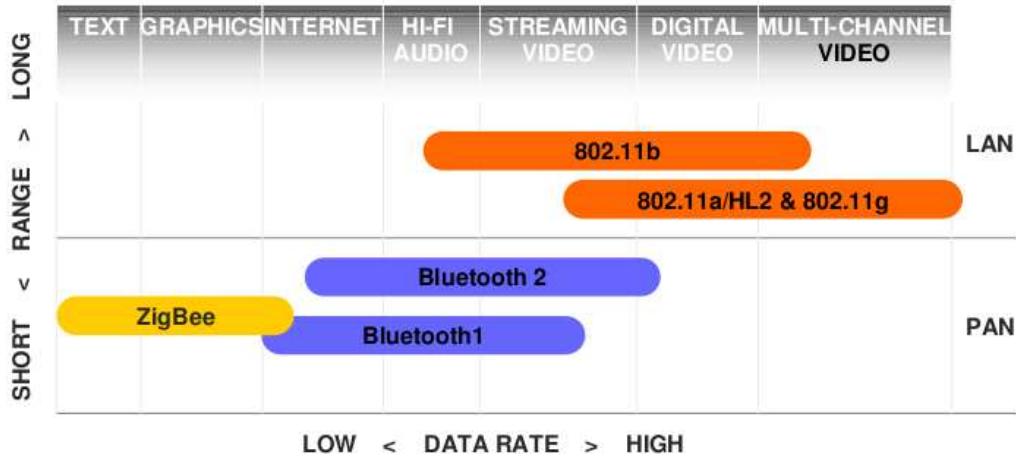
- Bassi consumi
- Semplicità nel codice di controllo
- Tempo di accesso alla rete basso
- Elevato numero di nodi della rete, nell'implementazione di MZ_Basic sono possibili teoricamente 65535 nodi, ma realisticamente qualche centinaio.

Lo standard ZigBee definisce tre livelli di apparati.:

- Il più complesso di tutti è il **Coordinator**. Questo elemento è il punto focale della rete. Opera come nodo principale di coordinamento e di raccolta dati ed è in grado di operare come bridge verso altre reti.
- Il **Full Function Device (Router)** è invece un dispositivo che può essere definito client, in grado cioè di generare informazioni e inviarle al nodo centrale. Può però funzionare anche come intermediario per altri dispositivi.
- Infine si ha il **Reduced Function Device (End-point)**, in grado di comunicare sulla rete ma incapace di fare da intermediario per altri dispositivi. Questo è l'elemento più semplice ed economico, perciò il candidato migliore per gli

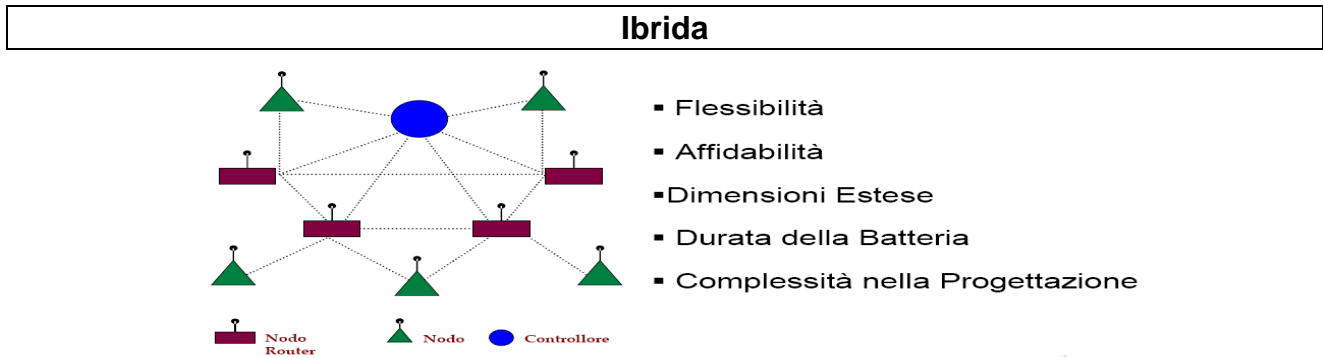
oggetti più semplici come lampadine, interruttori, ventole e via dicendo.

Nell'implementazione in MZ_Basic tutti i nodi sono Router tranne il nodo con ID = 0 il quale sarà il coordinator della rete. Di seguito un raffronto tra i principali protocolli di comunicazione Wireless:



| Feature(s) | IEEE 802.11b | Bluetooth | ZigBee |
|---------------|--------------------------------------|-----------------------------|--|
| Power Profile | Hours | Days | Years |
| Complexity | Very Complex | Complex | Simple |
| Nodes/Master | 32 | 7 | 64000 |
| Latency | Enumeration upto 3 seconds | Enumeration upto 10 seconds | Enumeration 30ms |
| Range | 100 m | 10m | 70m-300m |
| Extendability | Roaming possible | No | YES |
| Data Rate | 11Mbps | 1Mbps | 250Kbps |
| Security | Authentication Service Set ID (SSID) | 64 bit, 128 bit | 128 bit AES and Application Layer user defined |

Di seguito un esempio di rete implementabile utilizzando MZ Basic:



Implementazione in MZ_Basic

Ogni scheda viene identificata nella rete ZigBee attraverso un **ID** a 16bit (da 0 a 65535) che viene specificato al momento dell'aggiornamento del firmware scritto con l'MZCompiler. Il proprio ID è poi accessibile dal codice utilizzando la variabile di tipo **NET_ADDRESS**; utile per riutilizzare il codice su più dispositivi.

Le MZB dispongono di 8 differenti **canali** di ricezione, ognuno dei quali dispone di un buffer di 8 messaggi, ogni messaggio è composto da 16bit.

NB: Per fare un'analogia con il protocollo internet TCP-IP è possibile pensare all'ID come all'indirizzo IP e i canali come alle porte di comunicazione IP.

Trasmissione

La trasmissione via rete avviene attraverso il comando TX:

TX <id> AT <channel> VALUE <data>

Dove:

- **id:** (variabile) è l'indirizzo logico del ricevente del messaggio; se = 0 identifica il coordinator (ogni rete ZigBee deve avere almeno un coordinator), se > 0 identifica un end-point.
- **channel:** è un numero da 1 a 8 che identifica il canale logico in cui avverrà la comunicazione
- **data:** sono i 16bit da spedire.

ESEMPIO (spedisco 2000 all'MZB di indirizzo 0 sul canale 1 ogni 0,5 secondi):

```
var id as short
var channel as short

# Destinatarario (coordinator)
id = 0
```

```
# Canale di comunicazione
```

```
channel = 1
```

```
label main:
```

```
tx id at channel value 2000
```

```
wait 500
```

```
jump main
```

Ricezione

La ricezione viene implementata utilizzando le variabili di tipo NET_CTRL_RX1.. 8 e NET_RX_1 .. 8, dove:

- **NET_CTRL_RX1 .. 8:** sono variabili di controllo per un dato canale di ricezione e contengono il numero di messaggi presenti nella coda del dato canale (*i.e.* NET_CTRL_RX1 contiene il numero di messaggi presenti nel canale 1)
- **NET_RX_1 .. 8:** sono le variabili che permettono di accedere effettivamente al messaggio presente nella coda del dato canale (*i.e.* NET_RX_1 conterrà il più vecchio messaggio ricevuto nel canale 1).

ESEMPIO (leggo il messaggio ricevuto dall'esempio di trasmissione sopra):

```
#Variabile del registro di ricezione sul canale 1
```

```
var read as NET_RX_1
```

```
#variabile di controllo del canale 1
```

```
var ctrl as NET_CTRL_RX1
```

```
#Variabile per scrivere via seriale
```

```
var out as serial_out
```

```
label main:
```

```
# Controllo se sono presenti messaggi di rete
```

```
if ctrl = 1 then readnet
```

```
wait 500
```

```
jump main
```

```
#Scrivo il byte ricevuto sulla porta seriale
```

```
label readnet:
```

```
    out = read
```

```
    jump main
```

LINGUAGGIO C

Sviluppo nativo su MZBI/A-30

I moduli **MZBI/A-30** sono basati su microcontrollori Meshnetics che incorporano un core Atmel AVR **ATMega1281** come microcontrollore e un transceiver AT86RF230 per l'accesso alla rete ZigBit.

Di seguito alcune caratteristiche dell'MCU montato sulle schede MZBI/A-30:

4. 128Kbyte flash memory per la memorizzazione del firmware
 5. 8Kbyte di SRAM
 6. 4Kbyte di EEPROM (in parte occupata dai parametri di rete)
 7. Clock interno a 4Mhz, circa 8MIPS di throughput
- GPIO / ADC / PWM / I2C / IRQ / UART

Sono programmabili in C tramite il compilatore standard Open Source gcc utilizzando il framework di sviluppo fornito dalla Atmel per la gestione della rete ZigBit, framework nato per essere utilizzato in ambiente Windows utilizzando come compilatore avr-gcc fornito da WinAVR. Tale framework è chiamato **BitCloud** ed è scaricabile previa registrazione gratuita dal sito Atmel all'indirizzo: <http://www.atmel.com/BitCloud>.

In base al modello è necessario scaricare lo specifico framework, più precisamente:

- **MZBI-30**: scaricare il framework **BitCloud PRO SDK for ZigBit Development Kit**
- **MZBA-30**: scaricare il framework **BitCloud PRO SDK for ZigBit Amp Development Kit**

Sempre allo stesso indirizzo è disponibile tutta la documentazione necessaria.

Oltre al framework BitCloud è possibile programmare le MZBI/A-30 anche utilizzando il framework **Open Source OpenMac** (scaricabile da <http://www.meshnetics.com/opensource/mac/>) che gestisce il livello MAC ZigBit.

Sviluppo nativo su MZA30

Il modulo **MZA30** è basato su micro controllore Atmel ATmega644P ed è programmabile in linguaggio C utilizzando una versione apposita del compilatore *Open Source gcc* sia in ambiente Windows sia in ambiente GNU/Linux.

A differenza della programmazione Frameworked, la programmazione nativa permette di accedere a basso livello a tutte le caratteristiche del microcontrollore permettendo di sviluppare firmware molto versatili e potenti.

Per facilitare lo sviluppo vengono forniti tutti gli strumenti per poter iniziare a sviluppare in modo efficiente, più precisamente:

1. Libreria **libHAL644p** per la gestione delle periferiche del microcontrollore e da un micro framework per la gestione dei timer.
2. **Bootloader seriale** per la scrittura del firmware sul microcontrollore stesso
3. Tool di sviluppo Open Source **avr-gcc**, compilatore C per AVR, **avr-libc**, librerie standard C per AVR e **avrdude**, programmatore seriale per AVR.

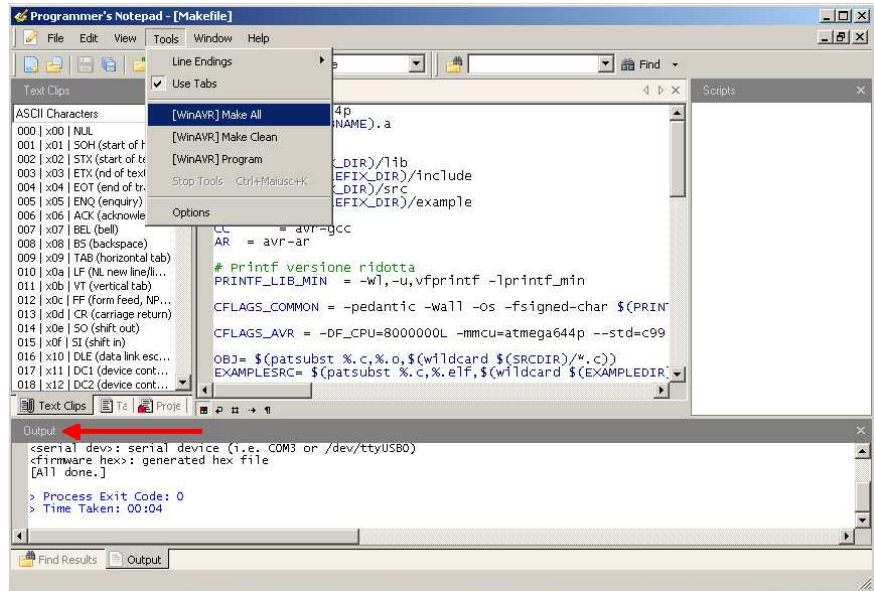
Di seguito alcune delle caratteristiche del microcontrollore ATmega644P:

- 64Kbyte flash memory per la memorizzazione del firmware
- 4Kbyte di SRAM
- 2Kbyte Eeprom, per la memorizzazione dei dati
- Clock interno a 8Mhz, circa 8MIPS di throughput.
- GPIO / ADC / PWM / I2C / IRQ / UART
- Basso consumo (QUANTO)

Installazione librerie

Una volta installati i tool di sviluppo è sufficiente scaricare dal sito www.nuzoo.it il pacchetto `libhal644p-<versione>.zip`, quindi decomprimerlo; verrà creata la directory `HAL644p` contenente i sorgenti delle librerie.

Dal menu `start->programmi->WinAvr` eseguire `Programmers Notepad`. Nel menu `File` selezionare `Open` e aprire il file `Makefile` nella directory `HAL644p` creata dalla decompressione del pacchetto delle librerie. Selezionare quindi dal menu `Tools` di `Programmers Notepad` (vedi immagine a destra) la voce `[WinAVR] Make All`; nella sezione `output` (freccia rossa nell'immagine a destra)



verrà visualizzato il log di compilazione. Al termine, se tutto è andato a buon fine, comparirà la stringa `[All Done]` e nella directory `HAL644p` verranno creati: la libreria `libHAL644p.a` e, nella directory `example`, alcuni esempi di firmware.

Descrizione e utilizzo

Le librerie `HAL644p` forniscono accesso a tutte le interfacce hardware i `MZA30`; `avr-gcc` assieme alle `avr-libc` permettono la compilazione del firmware e forniscono le interfacce alle librerie standard C; `avrdude` permette di scrivere il firmware appena creato sulla scheda `MZA30`. Al termine della compilazione delle librerie con il comando `Make All` vengono compilati anche alcuni esempi di firmware all'interno della directory `Example`, esempi molto utili per comprendere meglio il framework di sviluppo e per comprendere i parametri necessari alla compilazione.

Documentazione Librerie

All'interno della directory `HAL644p`, oltre alle librerie e agli esempi di codice, e' presente la directory `Documentation->html`; in questa directory e' presente la documentazione (generata da `Doxygen`) `html` dei moduli e delle funzioni della libreria `libHAL644p.a`, fondamentale strumento per lo sviluppo.

Serial Bootloader

Il processo di compilazione genera un file `<firmware>.hex` che contiene i codice macchina da scrivere sulla scheda

APPENDICI

Tabella 1 - Variabili

| Tipo Variabile | Descrizione |
|--|--|
| SHORT | Tipo Generico, 32 Registri da 16 bit ciascuno Utilizzabili come variabili temporanee |
| DIGITAL_IO_1 DIGITAL_IO_2 DIGITAL_IO_3 DIGITAL_IO_4 | I/O Digitale del 1°Niom I/O Digitale del 2°Niom I/O Digitale del 3°Niom I/O Digitale del 4°Niom |
| AVR_IO_0 AVR_IO_1 AVR_IO_2 AVR_IO_3 AVR_IO_4 AVR_IO_5 AVR_IO_6 AVR_IO_7 AVR_IO_8 AVR_IO_9 | AVR GPIO numero 0 (condiviso con PWM) AVR GPIO numero 1 (condiviso con PWM) AVR GPIO numero 2 (condiviso con PWM) AVR GPIO numero 3 (condiviso con PWM) AVR GPIO numero 4 AVR GPIO numero 5 AVR GPIO numero 6 AVR GPIO numero 7 AVR GPIO numero 8 AVR GPIO numero 9 |
| AVR_IN_0 .. 3 | Ingressi Analogici AVR (0...3) |
| SERIAL_IO SERIAL_OUT SERIAL_CTRL_RX | Porta Seriale RAW Porta seriale ASCII output Canale di Controllo Ricezione Seriale |
| PWM_FREQ_0 PWM_FREQ_1 PWM_FREQ_2 PWM_FREQ_3 | PWM Frequency 0 (GPIO 0 unsigned) PWM Frequency 1 (GPIO 1 unsigned) PWM Frequency 2 (GPIO 2 unsigned) PWM Frequency 3 (GPIO 3 unsigned) |
| PWM_DUTY_0 PWM_DUTY_1 PWM_DUTY_2 PWM_DUTY_3 | PWM Duty Cycle 0 (GPIO 0) PWM Duty Cycle 1 (GPIO 1) PWM Duty Cycle 2 (GPIO 2) PWM Duty Cycle 3 (GPIO 3) |
| I2C_ADDRESS I2C_MODE I2C_REG I2C_DATA | Indirizzo slave I2C La modalità I2C da usare Registro I2C da leggere/scrivere Dato I2C da leggere/scrivere |
| NET_ADDRESS | Contiene il netaddress del dispositivo |
| NET_RX_1 .. 8 NET_CTRL_RX1 .. RX8 | Canali di Ricezione via Rete (1...8) Canali di Controllo Ricezione via Rete (1...8) |
| VM_ERROR | Registro Contenente gli Errori della Virtual Machine |
| FALSE TRUE | Registro Contenente Sempre False (0) Registro Contenente Sempre True (1) |
| ANALOG_IN_1_1 .. 8 ANALOG_IN_2_1 .. 8 ANALOG_IN_3_1 .. 8 ANALOG_IN_4_1 .. 8 | Ingressi Analogici del 1°NIOM (1.. .8) Ingressi Analogici del 2°NIOM (1.. .8) Ingressi Analogici del 3°NIOM (1.. .8) Ingressi Analogici del 4°NIOM (1.. .8) |
| ANALOG_OUT_1_1 .. 4 ANALOG_OUT_2_1 .. 4 ANALOG_OUT_3_1 .. 4 ANALOG_OUT_4_1 .. 4 | Uscite Analogiche del 1°NIOM (1...4) Uscite Analogiche del 2°NIOM (1...4) Uscite Analogiche del 3°NIOM (1...4) Uscite Analogiche del 4°NIOM (1...4) |

Tabella 2 – Error Register

L' Error Register (E) contiene gli errori della Virtual Machine causati dall' ultima operazione eseguita.

Questo registro può risultare utile per un maggior controllo di correttezza del codice in quanto ogni operazione, al termine della sua esecuzione, imposta questo registro (E) con un valore corrispondente all' eventuale errore generato durante la sua esecuzione.

N.B.: Ciò significa che se vengono riscontrati due errori, ma il registro viene letto solo dopo la seconda operazione errata, l' errore riportato sarà relativo a quest' ultima e non alla prima !!

Ogni bit dell' Error Register corrisponde ad un dato errore; se l' i-esimo bit del registro è uguale ad 1 allora si è verificato l' errore corrispondente, con eccezione del bit 15.

| E | BIT | Nome | Descrizione |
|--|-----|--------------|---|
| r r o r R e g i s t e r | 0 | Undef | Undef |
| | 1 | Undef | Undef |
| | 2 | Bad Reg | Il registro utilizzato è sconosciuto alla Virtual Machine |
| | 3 | RO Reg | Richiesta di scrittura su registro Read-Only |
| | 4 | Net Error | Errore di connessione nella rete ZigBee |
| | .. | Undef | Undef |
| | .. | Undef | Undef |
| | 15 | Net Presence | Presenza della rete ZigBee (del coordinator), se = 0 → rete non presente |

Tabella 3 – Specifiche di comunicazione

| Protocollo | | Specifiche |
|---------------|------------------|--|
| I2C | | Frequenza impostata a 125Khz. |
| Porta Seriale | | UART 38400 BAUD 8N1 |
| MZB30 | Canali PWM | Frequenze da 0 a 65335 Hz definizione a 16bit |
| MZA30 | Canali PWM 0 e 1 | Frequenze da 0 a 65335 Hz definizione a 16bit |
| | Canali PWM 2 e 3 | Frequenze ammesse: 15600Hz, 1920Hz, 480Hz, 241Hz, 121Hz, 61Hz, 15Hz, 0Hz definizione a 8bit. |

1.1 - Installazione del Framework

Ambiente GNU/Linux

Installazione tool di sviluppo

E' necessario inizialmente installare i tool di sviluppo per microcontrollori AVR, più precisamente:

- **gcc-avr**: compilatore per chip AVR
- **avr-libc**: librerie standard c per chip AVR
- **avrdude**: programmatore seriale o ISP

E' necessario avere installato anche **Makefile** per la compilazione dei sorgenti.

Debian/Ubuntu (pacchetti deb)

In ambiente Debian / Ubuntu eseguire i seguenti comandi (come utente *root*):

```
aptitude install gcc-avr avr-libc avrdude
```

Fedora (pacchetti rpm)

In ambiente Fedora o in caso di distribuzioni con sistema di packaging basato sugli rpm eseguire, come utente *root*, i seguenti comandi:

```
yum install avr-gcc avr-libc avrdude
```

GNU/Linux standard (sorgente)

Altri sistemi GNU/Linux:

- Seguire le indicazioni presenti in http://www.nongnu.org/avr-libc/user-manual/install_tools.html
Per la compilazione del framework di sviluppo per AVR.
- Scaricare ed installare avrdude da <http://mirror.lhnidos.org/GNU/savannah/avrdude/>

Installazione librerie

Una volta installati i tool di sviluppo è sufficiente scaricare dal sito www.nuzoo.it il pacchetto **libhal644p-<versione>.zip** ed eseguire i seguenti passi:

```
utente@pejo:~/develop$ unzip libHAL644p-1.0.zip
```

```
utente@pejo:~/develop$ cd HAL644p/
```

```
utente@pejo:~/develop/HAL644p$ make all
```

Se tutto è andato a buon fine al termine dell'esecuzione di make all comparirà il seguente testo:

To write firmware to avr device use command:

```
avrdude -c avr109 -b 19200 -P <serial dev> -p atmega644p -U flash:w:<firmware hex>
```

<serial dev>: serial device (i.e. COM3 or /dev/ttyUSB0)

<firmware hex>: generated hex file

[All done.]

Installazione del Framework in ambiente Windows


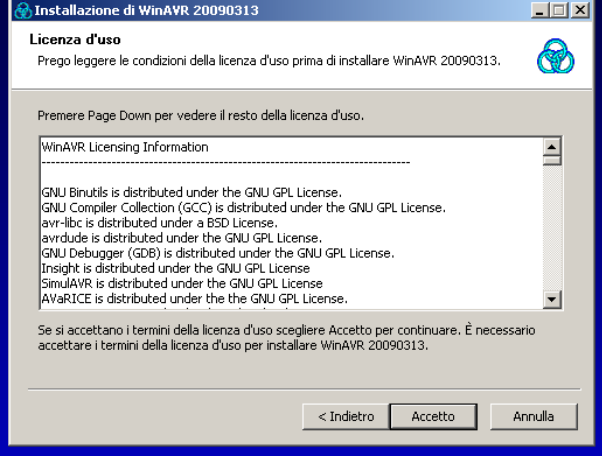
Installazione tool di sviluppo

Il tool di sviluppo è composto dalle seguenti componenti:

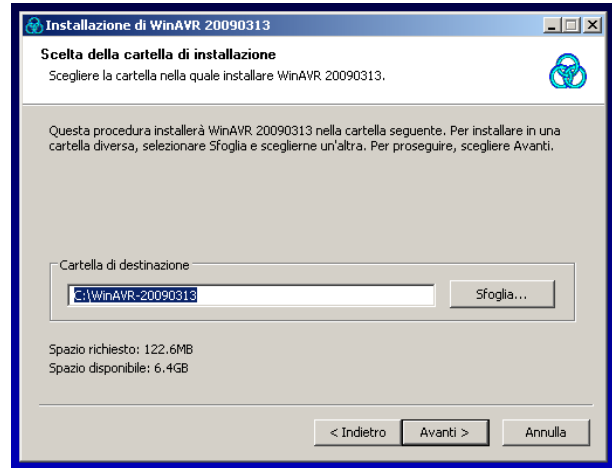
- **gcc-avr**: compilatore per chip AVR
- **avr-libc**: librerie standard c per chip AVR
- **avrdude**: programmatore seriale o ISP
- altri tool che facilitano lo sviluppo.

In ambiente Windows questi sono stati raggruppati in un unico pacchetto, **WinAvr**, rendendo semplice l'installazione.

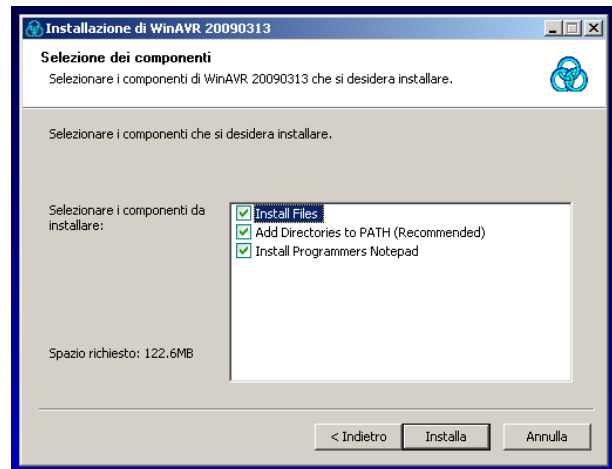
E' sufficiente scaricare WinAvr dal sito <http://www.nuzoo.it>, quindi lanciare **WinAVR-20090313-install.exe** e seguire i passi dell'installazione:

| | |
|---|--|
| <p>1) Alla finestra di benvenuto premere avanti.</p> |  |
| <p>2) Viene richiesto di accettare la licenza GPL. Premere avanti.</p> |  |

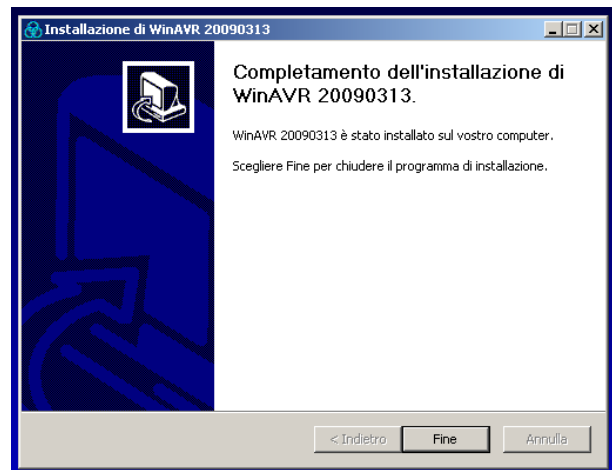
3) Viene richiesta la directory di installazione. Mantenere quella di default e premere **avanti**.



4) Vengono richiesti i componenti da installare. Di default sono tutti selezionati, premere **Installa**.



5) L'installatore installerà tutto il software necessario e imposterà tutte le variabili di sistema richieste. Al termine dell'installazione verrà visualizzata la finestra qui a destra. Premere **fine** per completare.



Le MZA30 sono tutte dotate di un boot-loader seriale (standard di programmazione avr109) che permette di scrivere codice compilato semplicemente utilizzando un cavo usb senza aver bisogno di programmatori esterni.

Una volta compilato il codice e generato il file <firmware>.hex è sufficiente:

- Collegare la MZA30 alla porta USB del PC (avendo precedentemente installato, in ambiente Windows, i driver di gestione della scheda)
- Identificare la porta seriale associata alla MZ30: premere tasto destro su **Risorse del Computer**, quindi **Proprietà->tab Hardware->Gestione periferiche**, quindi andare su **Porte (COM e LPT)**
- Riavviare la MZ30 premendo l'apposito pulsante di reset.
- Lanciare il comando

```
avrdude -c avr109 -b 19200 -P <porta seriale (i.e. COM2 o /dev/ttyUSB0)> -p
atmega644p -U flash:w:<percorso><firmware>. Hex
```

(Dal momento in cui viene riavviata la MZ30 al momento in cui viene lanciato avrdude non deve passare piu' di un secondo, nel caso in cui avrdude non riuscisse a contattare la MZ30, ripetere dal passo 3)

- Se tutto è andato a buon fine verrà visualizzata la stringa:
avrdude done. Thank you.

Link utili

- <http://www.nongnu.org/avr-libc/>: Homepage del progetto avr-libc contenente la documentazione delle librerie standard C per Avr.
- <http://www.atmel.com/products/avr/default.asp>: Sito del produttore dei microcontrollori AVR fonte per datasheet e numerosi application note.
- <http://www.avrfreaks.net/>: AVR Freaks comunita' di sviluppo per progetti basati su AVR, fonte di molte informazioni.