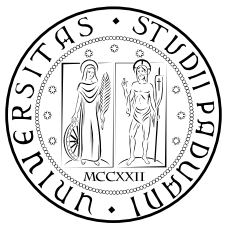


UNIVERSITÀ DEGLI STUDI DI PADOVA

DIPARTIMENTO DI TECNICA E GESTIONE DEI SISTEMI INDUSTRIALI
CORSO DI LAUREA TRIENNALE IN INGEGNERIA MECCATRONICA E MECCATRONICA

TESI DI LAUREA TRIENNALE

SVILUPPO DI UN SOFTWARE DI INTERFACCIA PER UN EMULATORE DI GENERATORI EOLICI

Development of an Interface Software for Wind Generators Emulator

Relatore: Prof. Paolo Mattavelli

Correlatore: Ing. Andrea Petucco

Laureando: Stefano Tonegato
597058-IMM

ANNO ACCADEMICO: 2015-16

"La vita é come un treno che ti passa di fronte.
A volte vedi un vagone e volte ne vedi un altro.
Altre volte vedi il vagone ristorante,
e a stento riesci a scorgere i bicchieri d'acqua posati sopra i tavoli.
L'importante é che, durante lo scorrere del treno,
tu riesca a vedere tutti quei bicchieri
mezzi pieni."

SOMMARIO

Lo scopo di questa tesi é lo sviluppo di un'interfaccia grafica per la comunicazione dati tramite personal computer ad un innovativo dispositivo per l'emulazione del comportamento di sistemi di generazione elettrica da fonte eolica finalizzato all'attivitá di certificazione, previste dalle normative italiane ed internazionali, in modo pratico ed economico. L'emulatore é finalizzato alla simulazione delle applicazioni eoliche cosiddette *full-converter*, ove il generatore eolico é connesso alla rete elettrica attraverso un convertitore *AC/DC/AC*, attraverso il quale viene gestita tutta la potenza del generatore.

La relazione descrive i vari passaggi relativi allo sviluppo dell'interfaccia di comunicazione dati e si divide in sei capitoli che riguardano, rispettivamente, l'introduzione agli impianti eolici, l'emulazione di un generatore eolico, le interfacce di comunicazione seriale, le specifiche e scelte progettuali, la descrizione del software di interfaccia sviluppato ed infine l'analisi dei risultati sperimentali.

INDICE

1 INTRODUZIONE 1
- 1.1 L'Energia Eolica 1
- 1.2 Gli Impianti Eolici 2
- 1.3 Tipologia di turbine 3
  - 1.3.1 Turbine eoliche ad asse verticale 3
  - 1.3.2 Turbine eoliche ad asse orizzontale 3

2 EMULAZIONE DI UN GENERATORE EOLICO 11
- 2.1 Introduzione 11
- 2.2 Descrizione della parte hardware dell'emulatore 16

3 INTERFACCE DI COMUNICAZIONE SERIALE 19
- 3.1 Introduzione 19
- 3.2 Interfaccia di comunicazione I2C 21
- 3.3 Interfaccia di comunicazione SPI 22
- 3.4 Interfaccia di comunicazione SCI 25
  - 3.4.1 Formato di comunicazione dell'interfaccia SCI 27
  - 3.4.2 UART 31
- 3.5 Protocollo di comunicazione Ethernet 31
  - 3.5.1 Transfer Control Protocol 35
  - 3.5.2 User Datagram Protocol 36
  - 3.5.3 Confronto TCP vs UDP 36

4 SPECIFICHE E SCELTE PROGETTUALI 39
- 4.1 Introduzione 39
- 4.2 Specifiche scheda NetBurner 41
- 4.3 Specifiche dell'ambiente di sviluppo 43
- 4.4 Socket 43

5 SVILUPPO DEL SOFTWARE DI INTERFACCIA 45
- 5.1 Introduzione 45
  - 5.1.1 Struttura del codice di programmazione 45
  - 5.1.2 Parametri di simulazione 46
- 5.2 Descrizione del codice di programmazione 48
  - 5.2.1 mainwindow.h 48
  - 5.2.2 main.cpp 49
  - 5.2.3 mainwindow.cpp 50
- 5.3 Descrizione dell'interfaccia grafica 61

6 RISULTATI SPERIMENTALI 67

Conclusioni 71

Appendix 73

A CODICE DI PROGRAMMAZIONE 75

BIBLIOGRAFIA 85

ELENCO DELLE FIGURE

Figura 1 Tipi di turbina ad asse verticale 4
Figura 2 (a) turbina sopravento con orientamento attivo; (b) idem ma passivo; (c) turbina sottovento con orientamento passivo. 5
Figura 3 Esempio di turbina a 3 pale. 5
Figura 4 Tipi di turbina ad asse verticale 6
Figura 5 Interno di una tipica navicella di una turbina ad asse orizzontale [4]. 7
Figura 6 Curva di potenza per una turbina da 600 kW [3]. 9
Figura 7 Emulatore realizzato dalla facoltá di Aachen, schema di base e rendering [10]. 14
Figura 8 Realizzazione dell'emulatore di Aachen [11]. 14
Figura 9 Emulazione meccanica del sistema turbina-riduttore realizzata da Aachen [11]. 15
Figura 10 Emulazione elettrica del sistema turbina-riduttore-generatore in via di sviluppo. 15
Figura 11 Configurazione di una turbina basata su generatore a magneti permanenti e convertitore elettronico di tipo back-to-back full-power. 15
Figura 12 Scheda Texas Instruments LAUNCHXL-F28277S. 16
Figura 13 Microcontrollore-DSP C2000 Delfino TMS320F28377S. 16
Figura 14 Ambiente di sviluppo Code Composer Studio 6. 17
Figura 15 Routine di emulazione. 18
Figura 16 Modalitá di trasmissione. 19
Figura 17 Trasmissione asincrona. 21
Figura 18 Moduli I2C collegati [12]. 21
Figura 19 Bus SPI: connessioni tra un *master* ed un singolo *slave*. 23
Figura 20 Diagramma temporale dei segnali SCLK, SDO, SDI. 24
Figura 21 Esempio di connessione diretta tra un master e 3 slave controllati singolarmente. 25
Figura 22 Esempio di *master* connesso a 3 *slave* collegati in catena. 25
Figura 23 Schema logico interfaccia SCI [12]. 26
Figura 24 Esempi di frame [12]. 27
Figura 25 Schema a blocchi interfaccia SCI [12]. 28
Figura 26 Esempio di formato di comunicazione asincrona SCI [12]. 29

Figura 27 Esempio di ricezione seriale SCI [12]. 29
Figura 28 Esempio di trasmissione seriale SCI [12]. 30
Figura 29 Esempio di porta seriale *RS-232*. 31
Figura 30 Schema di rete Ethernet semplice. 32
Figura 31 Esempio di cavo Ethernet con doppini intrecciati e connettori RJ-45. 32
Figura 32 Esempi di rete con dispositivi di gestione del traffico 33
Figura 33 Esempio di *frame Ethernet* [2]. 33
Figura 34 Esempio di datagramma [1]. 37
Figura 35 Scheda NetBurner SB70LC-100IR. 40
Figura 36 Istantanea di *NetBurner Page Configuration*. 41
Figura 37 Istantanea di *IPSetup*. 42
Figura 38 Istantanea dell'ambiente di sviluppo *Qt Creator*. 44
Figura 39 Struttura del codice nell'ambiente Qt. 46
Figura 40 Confronto tra variabile *float* e *char*. 54
Figura 41 Istantanea dell'ambiente di sviluppo di interfacce grafiche *Qt Design*. 61
Figura 42 Istantanea del software Turbine Control sviluppato. 62
Figura 43 Menú a tendina. 63
Figura 44 Icona e titolo software. 63
Figura 45 Barra di stato. 63
Figura 46 Group Box *Wind Turbine*. 64
Figura 47 Group Box *Generator Characteristics*. 64
Figura 48 Group Box *Gear Box*. 65
Figura 49 Group Box *Power*. 65
Figura 50 Group Box *Output*. 66
Figura 51 Group Box *Simulation*. 66
Figura 52 Realizzazione dell'emulatore sviluppato. 67
Figura 53 Specifiche relative ai buchi di tensione dettate dalle normative. 68
Figura 54 Area relativa alla disconnessione consentita del generatore durante un buco di tensione. 68
Figura 55 Forme d'onda acuisite dagli oscilloscopi. 70

ELENCO DELLE TABELLE

Tabella 1 Riepilogo dei parametri utilizzati per l'emulazione 67

Tabella 2 Notazione delle forme d'onda illustrate 69

1 INTRODUZIONE

1.1 L'ENERGIA EOLICA

L'energia eolica é la conversione dell'energia posseduta dal vento in una forma utilizzabile di energia, generalmente grazie all'utilizzo di vele che spingono in moto le navi, tramite mulini a vento che producono energia meccanica oppure ancora tramite aerogeneratori che producono energia elettrica.

La parola "eolica" deriva da Eolo, antico dio greco del vento, il cui nome *aiolos* significa veloce.

L'energia eolica é una delle prime forme di energia che l'uomo ha imparato a sfruttare; sin dall'antichitá ha impiegato la forza del vento per la navigazione con barche a vela e per muovere le pale dei mulini a vento utilizzati per macinare cereali, spremere olive o per pompare l'acqua.

Durante il secolo scorso questo uso del vento é stato soppiantato dalla disponibilitá di motori a combustione interna e dall'energia elettrica a basso costo fornita da centrali a combustibili fossili e nucleari. Solo negli ultimi decenni, dato l'aumento dell'inquinamento, la diminuzione delle riserve di combustibili fossili e l'aumento dei consumi globali, l'energia eolica viene impiegata per produrre energia elettrica.

Il vento é un fenomeno atmosferico dovuto al riscaldamento del Sole. Il Sole irradia sulla Terra $1,74x10^{17}$ watt di potenza; di questa circa il 2% viene convertita in energia eolica.

La Terra cede all'atmosfera il calore ricevuto dal Sole, ma non lo fa in modo uniforme. Nelle zone in cui viene ceduto meno calore la pressione dei gas atmosferici aumenta, mentre dove viene ceduto piú calore, l'aria diventa calda e la pressione dei gas diminuisce. Si formano cosí aree di alta pressione e aree di bassa pressione, influenzate anche dalla rotazione della Terra. Quando diverse masse d'aria vengono a contatto, la zona dove la pressione é maggiore tende a trasferire aria dove la pressione é minore. Succede la stessa cosa quando lasciamo sgonfiare un palloncino. L'alta pressione all'interno del palloncino tende a trasferire l'aria verso l'esterno, dove la pressione é piú bassa, dando luogo a un piccolo flusso d'aria. Il vento é dunque lo spostamento d'aria, piú o meno veloce, tra zone di diversa pressione. E tanto piú alta é la differenza di pressione, tanto piú veloce sará lo spostamento d'aria, tanto piú forte sará il vento.

1.2 GLI IMPIANTI EOLICI

La piú importante forma di impiego dell'energia eolica é quella relativa alla produzione di energia elettrica attraverso i generatori eolici.

L'energia elettrica si ottiene sfruttando l'energia cinetica del vento: le masse d'aria in movimento fanno girare le pale di un elica; queste a loro volta sono collegate ad un generatore che trasforma l'energia meccanica (rotazione delle pale) in energia elettrica. Questi moderni mulini a vento sono chiamati aerogeneratori.

La produzione di energia elettrica attraverso un'aerogeneratore dipende dall'interazione tra le pale del rotore e il vento trasformando l'energia meccainica di rotazione che viene convertita in energia elettrica per mezzo di un generatore elettrico. La quantitá di energia di una massa m in kg di vento che soffia ad una velocitá v su una turbina eolica é pari a:

$$E_\omega = \frac{1}{2} \cdot m \cdot v^2 \tag{1}$$

in cui la massa m é data da:

$$m = \rho \cdot V_{ol} \tag{2}$$

dove ρ é la densitá dell'aria in kg/m^3 e V_{ol} é il volume dell'aria in m^3, pari a Avt, dove A é la sezione del rotore in m^2. In queste condizioni la potenza meccanica totale del vento é data da:

$$P_w = \frac{1}{2} \cdot \rho \cdot A \cdot v^3 \tag{3}$$

Secondo la legge di Betz, la massima potenza estraibile da una turbina eolica sará sempre inferiore al 59.3% della potenza totale. Viene introdotto perció un termine C_p detto coefficiente di potenza che verrá usato per ottenere la potenza meccanica in uscita da una turbina eolica:

$$P_m = \frac{1}{2} \cdot \rho \cdot A \cdot v^3 \cdot C_p(\beta, \gamma) \tag{4}$$

dove il termine C_p é funzione dell'angolo delle pale della turbina β e del rapporto γ tra velocitá lineare della punta della pala e della velocitá del vento, che viene comunemente nominato *tip speed ratio*, la cui espressione é quindi:

$$\gamma = \frac{\omega_r \cdot R}{v} \tag{5}$$

dove ω_r é la velocitá del rotore della turbina in rad/s ed R il suo raggio. La relazione tra C_p, β e γ é propria di ogni turbina e di solito si ricava da un prototipo a scala ridotta della turbina reale.

1.3 TIPOLOGIA DI TURBINE

Esistono turbine eoliche diverse per forma e dimensione. In base alla loro tecnologia di costruzione possono essere suddivise in due macro-famiglie:

1. turbine eoliche ad asse verticale - (acronimo inglese VAWT)
2. turbine eoliche ad asse orizzontale - (acronimo inglese HAWT)

1.3.1 *Turbine eoliche ad asse verticale*

Sebbene le turbine ad asse verticale esistano da secoli ed in questi ultimi anni le relative tecnologie siano state notevolmente sviluppate, non sono diffuse quanto l'eolico orizzontale e trovano applicazione soprattutto in zone urbanizzate di tipo residenziale o industriale. Il motivo principale della minore diffusione in zone aperte, come ad esempio quelle di campagna, é che, non utilizzando di solito un'alta torre, non si avvantaggiano della maggiore velocitá del vento che si incontra ad altezze dal suolo piú elevate. Gli aerogeneratori ad asse verticale non necessitano di essere orientati nella direzione del vento. Inoltre, poiché l'albero del rotore é verticale, il generatore elettrico e il sistema di trasmissione che interfaccia quest'ultimo al rotore possono venire montati su una torre bassa e leggera, che ha minor costo rispetto alle torri elevate tipiche delle turbine ad asse orizzontale.

Le turbine ad asse verticale si dividono in:

- turbina Savonius
- turbina Darrieus
- turbina ibrida Darrieus-Savonius

Tali turbine sono raffigurate in fig. 1.

1.3.2 *Turbine eoliche ad asse orizzontale*

Le turbine ad asse orizzontale, che costituiscono circa il 99% delle turbine attualmente utilizzate si dividono in due grandi tipi:

- turbine sopravento
- turbine sottovento

Nelle turbine *sopravento*, che sono di gran lunga le piú diffuse, l'albero del rotore deve essere puntato nella direzione del vento, che arriva sul rotore dal davanti. Nel grande eolico, per orientare il rotore nella direzione del vento rilevata da appositi sensori, e mantenerlo entro un opportuno "angolo di attacco" (in quanto non é possibile seguire di continuo il vento nel suo variare di direzione) si usa un

(a) *Esempio di una turbina Savonius..*

(b) *Esempio di una turbina Darrieus..*

(c) *Esempio di turbina ibrida Darrieus-Savonius..*

Figura 1: Tipi di turbina ad asse verticale

sistema di imbardata poggiato su dei cuscinetti e dotato di un motore, il quale agisce in maniera attiva gestito da un controllo elettronico. Nel piccolo eolico, invece, il rotore é sopravento non in maniera attiva bensí passiva, in quanto continuamente orientato nella direzione del vento grazie a una pinna/timone posta alle sue spalle. Nelle turbine *sottovento*, infine, il vento non giunge sul rotore dal davanti bensí dal di dietro, incontrando prima la torre di sostegno della turbina, ed anche qui il rotore viene orientato in maniera passiva.

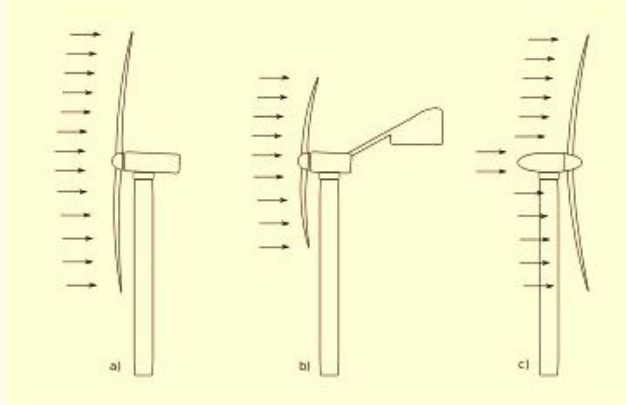


Figura 2: (a) turbina sopravento con orientamento attivo; (b) idem ma passivo; (c) turbina sottovento con orientamento passivo.

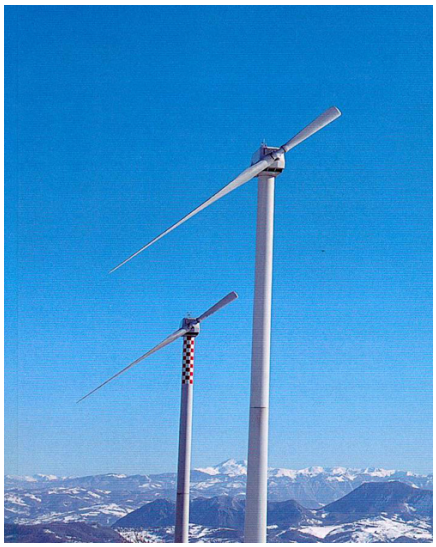
Il piú diffuso aerogeneratore ad asse orizzontale é il tipo avente turbina a tre pale, rappresentato in fig. 3.

Figura 3: Esempio di turbina a 3 pale.

Tuttavia esistono modelli con turbina a 2 pale (fig. 4a), a singola pala con contrappeso (fig. 4b) e modelli con turbina multipala tipica

dei paesaggi nord-americani per applicazioni soprattutto nel eolico a basse potenze (fig. 4c).

(a) *Esempio di turbina a 2 pale..*



(b) *Esempio di turbina a singola pala..*

(c) *Esempio di turbina multipala..*

Figura 4: Tipi di turbina ad asse verticale

La struttura di un aerogeneratore ad asse orizzontale é composta da un sostegno formato da fondamenta e torre , solitamente in acciaio, che reca alla sua sommitá una navicella. L'illustrazione di fig. 5 mostra l'interno della navicella di una tipica turbina ad asse orizzontale di taglia medio-grande. In questo involucro sono contenuti l'albero di trasmissione a bassa velocitá, il moltiplicatore di giri a ingranaggi, l'albero di trasmissione ad alta velocitá, il generatore elettrico e i dispositivi ausiliari composti da un sistema frenante, un sistema

di controllo di imbardata e un sistema di controllo del vento con anemometro.)

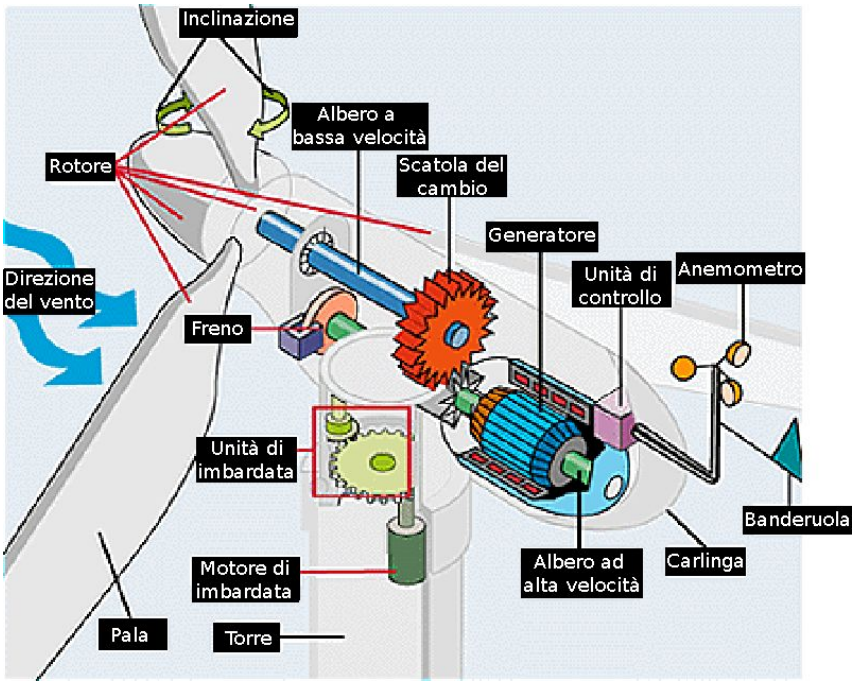


Figura 5: Interno di una tipica navicella di una turbina ad asse orizzontale [4].

All'estremitá dell'albero lento é fissato il rotore costituito dal mozzo sul quale sono montate le pale. La forma delle pale é disegnata in modo che il flusso dell'aria che le investe azioni il rotore. Dal rotore l'energia cinetica del vento viene trasmessa ad un generatore elettrico per mezzo dell'albero di trasmissione a bassa velocitá. Il generatore elettrico converte l'energia meccanica ricevuta in energia elettrica. Esistono due tipi di generatori elettrici: il generatore asincrono e il generatore sincrono.

- il generatore asincrono é essenzialmente un motore trifase ad induzione caratterizzato da una velocitá di sincronismo che dipende dal numero di poli e dalla frequenza di rete. Se la coppia meccanica agente sull'albero rotore é motrice anziché resistente e fa aumentare la velocitá di rotazione fino a superare la velocitá di sincronismo, la macchina elettrica asincrona passa dal funzionamento come motore a quello come generatore immettendo energia elettrica in rete. La differenza relativa tra la velocitá di sincronismo e la velocitá effettiva di rotazione é chiamata scorrimento s che nel funzionamento da generatore diventa quindi negativo.

  Nei generatori asincroni usuali con rotore a gabbia di scoiattolo (rotore in cortocircuito), lo scorrimento é di circa l'1% cosicché tali dispositivi sono di fatto considerati a velocitá di rotazione

costante. La velocitá di rotazione dell'albero principale varia da zero alla velocitá nominale di dimensionamento in funzione della velocitá del vento incidente, ma non puó essere controllata e variata volontariamente da un sistema di controllo come accade per i sistemi a velocitá variabile.

La corrente di magnetizzazione dello statore, la quale crea il campo magnetico rotante al traferro, é fornita dalla rete stessa. Inoltre tale generatore consuma una certa quantitá di potenza reattiva, la quale deve essere fornita da sistemi compensatori quali i condensatori. Quando una raffica di vento colpisce una turbina eolica dotata di un generatore asincrono a rotore in cortocircuito, poiché la velocitá di rotazione é costante, si ha una repentina variazione della coppia e la conseguente rapida variazione della potenza erogata; se la potenza di cortocircuito della rete a cui l'aerogeneratore é connesso é bassa, possono pertanto verificarsi delle fluttuazioni di tensione sui dispositivi elettrici collegati in prossimitá, fluttuazioni che possono creare malfunzionamenti dei dispositivi stessi. Inoltre si puó assistere alla variazione rapida del flusso luminoso emesso dalle lampade elettriche, che genera quel fastidioso "sfarfallio" noto come *flicker*. Anche per tale motivo la ricerca si é spinta verso la realizzazione di sistemi a velocitá variabile che consentono inoltre di ridurre gli "strappi di coppia" sul rotore, che accellerano il processo di usura meccanica del rotore, e di far funzionare lo stesso nel punto di massima efficienza aerodinamica su un ampio range di velocitá del vento. Soluzioni a velocitá variabile realizzate con generatori ad induzione si realizzano interponendo tra lo statore del generatore con rotore gabbia di scoiattolo e la rete un convertitore di frequenza o utilizzando un generatore asincrono a rotore avvolto ad anelli il cui rotore é alimentato da una corrente alternata indipendente, fornita da un convertitore di frequenza: in tal modo la velocitá di sincronismo é funzione della differenza tra la frequenza di rete e la frequenza della corrente rotorica. Si puó raggiungere cosí una variazione di velocitá del 30%.

- nel generatore sincrono, chiamato anche alternatore, il rotore é costituito da un elettromagnete a corrente continua o da magneti permanenti (PMSG). La frequenza della tensione indotta sullo statore (e quindi della corrente prodotta) é direttamente proporzionale alla velocitá di rotazione del rotore. Per consentire un funzionamento a velocitá variabile, si interpone tra alternatore e rete un convertitore elettrico che trasforma dapprima la tensione a frequenza variabile (in funzione della velocitá del rotore e quindi del vento) in uscita dal generatore in corrente continua mediante un raddrizzatore elettronico e successivamente la riconverte in corrente alternata a frequenza di rete tramite un

inverter. Cosí facendo si svincola la frequenza della corrente generata dalla frequenza di rete. L'assorbimento di corrente del convertitore elettronico dipende dal numero di giri del PMSG. Esso é tarato sulle caratteristiche aerodinamiche della turbina in modo da massimizzarne l'efficienza. Quando il sistema é in equilibrio e arriva una raffica di vento aumenta l'energia cinetica con conseguente aumento della velocitá di rotazione delle pale che porta un aumento di assorbimento del convertitore. Si ottiene cosí un nuovo punto di equilibrio. Viceversa quando il vento cessa di soffiare.

Ogni turbina ha una propria caratteristica curva di potenza. La curva di potenza di una macchina eolica mostra il rapporto tra la velocitá di rotazione della turbina e la potenza elettrica istantanea erogata dal generatore. Il grafico di fig. 6 riporta, in maniera esemplificativa, il comportamento di una turbina eolica da 600 kW al variare della velocitá del vento. Nell'asse delle ascisse é indicata la potenza elettrica erogata, mentre l'asse delle ordinate riporta le diverse velocitá del vento.

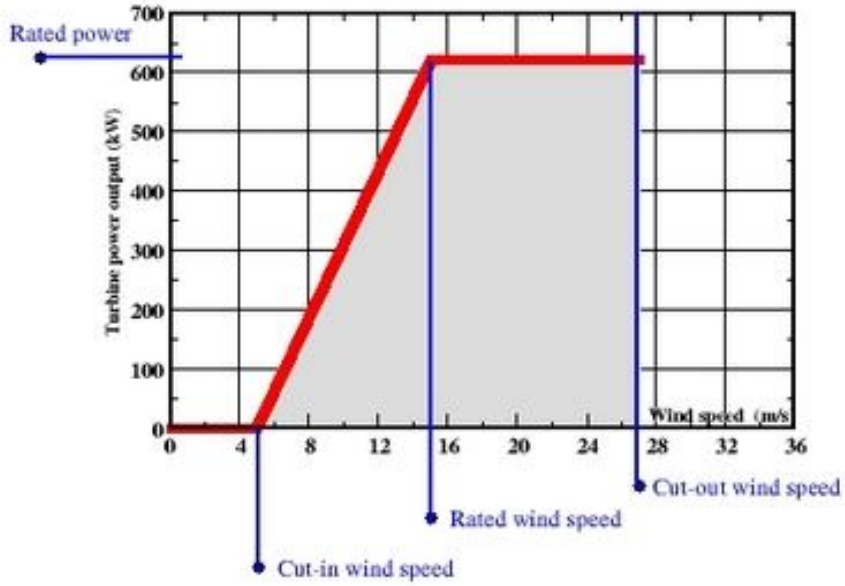


Figura 6: Curva di potenza per una turbina da 600 kW [3].

La soglia minima (cut-in wind speed) di velocitá del vento richiesta per l'avvio della turbina é in questo caso di 5 m/s. La velocitá nominale (rated wind speed), cioé la velocitá del vento nella quale la turbina raggiunge la potenza nominale di targa, é pari a 15 m/s. La potenza erogata rimane costante sul valore nominale fino al raggiungimento della soglia massima (cut-out wind speed) di velocitá del vento, che nel caso di questa turbina é realisticamente localizzata in 27 m/s. Oltrepassata la soglia massima tollerata, l'aerogeneratore si mette in sicurezza e interrompe la produzione di energia elettrica,

per evitare il rischio di danneggiamenti. A queste velocitá sorgono problemi di:

- vibrazioni/oscillazioni meccaniche alla torre di sostegno della navicella.
- sollecitazioni elevate dovute alla forza centripeta a causa dell'eccessiva velocitá di rotazione delle pale della turbina.

I progressi nel disegno dei rotori eolici degli ultimi 15 anni permettono a questi di operare anche a velocitá del vento inferiori, imbrigliando una quantitá maggiore di energia e raccogliendola ad altezze maggiori, aumentando la quantitá di energia eolica sfruttabile. Sono stati messi a punto anche dei rotori con pale "mobili": variando l'inclinazione delle pale al variare della velocitá del vento é possibile limitare la velocitá di rotazione della turbina al valore nominale quando essa tende a superarlo. Tale controllo prende il nome di "regolatore dell'angolo di pitch" (pitch-angle controller). Questo sistema é utilizzato anche in caso di raffiche di vento troppo forti. Le pale vengono fatte ruotare in maniera tale da far aumentare l'angolo di attacco della pala con il vento, cosí da ridurre la coppia estratta.

2 EMULAZIONE DI UN GENERATORE EOLICO

2.1 INTRODUZIONE

L'aumento degli impianti di produzione di energia elettrica da fonte rinnovabile, ad esempio di tipo fotovoltaico e di tipo eolico, ha introdotto una varietà di problematiche relative alla gestione delle reti elettriche di distribuzione e trasmissione. Tali reti, infatti, sono nate nell'ottica che la generazione elettrica avvenisse in modo concentrato attraverso grandi centrali elettriche, mentre gli utenti, siano essi a bassa tensione o a media tensione, fossero principalmente passivi. La presenza di una massiccia generazione distribuita, ad esempio di tipo fotovoltaico, eolico e idroelettrico, ha strutturalmente cambiato lo scenario e la gestione della rete stessa, rendendo necessario l'introduzione di una serie di regolamentazioni tecniche per le nuove piccole o medie sorgenti di energia distribuite.

Le normative in merito alle regole di connessione di sorgenti in bassa e media tensione, come la [8, 7], sono intervenute con lo scopo di fornire delle prescrizioni di riferimento per la loro corretta connessione tenendo conto delle caratteristiche funzionali, elettriche e gestionali delle reti di distribuzione. A questo scopo le normative hanno modificato le modalità con le quali gli inverter devono interagire con la rete. I punti fondamentali trattati dalle norme sono la possibilità di immettere in rete energia reattiva, l'insensibilità ai buchi di tensione, la possibilità di modificare lo stato su precisi comandi inviati dal distributore.

Di conseguenza, prima della connessione di nuovi dispositivi di generazione-conversione, è necessario che la sorgente elettrica sia conferme con le regole e le specifiche richieste dalle normative. Per questo motivo il generatore deve essere sottoposto ad una serie di prove di certificazione presso un laboratorio accreditato EA secondo la Norma ISO 17025, come ad esempio viene descritto nell'Allegato N della CEI 0-21 [8].

Per i generatori statici senza parti in movimento, quali ad esempio quelli di tipo fotovoltaico, le prove di certificazioni sono relativamente piu' semplici in quanto possono essere svolte in laboratori certificati sostituendo la sorgente fotovoltaica con un alimentatore in cc. Diversamente, quando si tratta di una prova di generatori con parti rotanti connesse, come i generatori eolici, sono necessarie delle misure sul campo. Questo richiede ovviamente uno sforzo ed un onere aggiuntivo; inoltre, le prove sul campo sono soggette all'aleatorietà della sorgente, quali ad esempio la presenza del vento.

Le principali prove richieste a livello europeo per effettuare una connessione di un generatore sono riassunte nel seguente elenco:

- Anti-islanding;
- Armoniche;
- Condizioni di connessione e riconnessione;
- Corrente continua di uscita;
- Flicker;
- Limitazione della potenza attiva;
- Low Voltage Fault Ride Through (LVFRT);
- Over Voltage ride Through (OVRT);
- Residual current monitoring (RCMU);
- Richiusura in discordanza di fase;
- Scambio della potenza reattiva e regolazione della tensione.

Questo progetto mira alla realizzazione di un banco di test per l'emulazione del comportamento di turbine eoliche in modo da effettuare test di compatibilità sui convertitori eolici a bordo dell'aerogeneratore. Tra le varie prove necessarie, questa relazione si focalizza sul test di immunità al Low-Voltage-Ride-Through (LVRT), che risulta essere complesso e costoso da effettuare sul campo. Infatti, il test richiede l'utilizzo di un particolare dispositivo che realizza buchi di tensione ai capi del eolico attraverso un partitore ad impedenze [6]. Tale dispositivo è solitamente delle dimensioni di un container e perciò non è sempre semplice raggiungere il luogo dove è necessario effettuare la misura. Il noleggio del dispositivo di tensione è, inoltre, molto costoso, oltre a richiedere un nuovo cablaggio del generatore eolico verso la rete elettrica. Vi è infine da menzionare che il generatore di buchi di tensione introduce delle perturbazioni non trascurabili sulla rete di distribuzione e per il suo utilizzo è necessario un'accordo specifico con l'ente di distribuzione dell'energia, non sempre possibile in certe nazioni. Oltre a queste problematiche, le normative specificano precisi livelli di estrazione di potenza durante il quale effettuare i test. Ciò significa che è possibile effettuare le prove solo nelle giuste condizioni atmosferiche e che il test dipende pesantemente dalle condizioni meteorologiche.

Tramite l'emulatore eolico proposto in questo progetto è possibile sostituire le parti elettromeccaniche che compongono il generatore eolico con un alimentatore puramente elettronico. In questo modo è possibile effettuare le prove di certificazione in condizioni di vento ed estrazione di potenza completamente controllabili. La possibilità

di effettuare test di LVRT in laboratorio risolve quindi, almeno in linea di principio, i problemi descritti precedentemente, rendendo le condizioni di test ripetibili e controllabili e la certificazione molto più economica. Tutto ciò è fattibile per convertitori eolici di potenza nominale fino a quelle raggiungibili dagli alimentatori controllati disponibili nei laboratori di certificazione. Allo stato attuale e per le nostre conoscenze, il livello di potenza attualmente disponibile in laboratori di certificazione è di circa 1 *MVA*.

Analizzando lo stato dell'arte su tale tematica, soprattutto a livello accademico, si sono riscontrati alcuni studi connessi all'analisi del comportamento di generatori eolici durante i buchi di tensione. Ad esempio nell'università di Chalmers è stato proposto un generatore di buchi di tensione basato su un convertitore elettronico bidirezionale [9], giá esistente nelle vicinanze del generatore eolico. Questa soluzione, rispetto a quella tradizionale basata su partitore ad impedenze, offre una maggiore versatilità e flessibilità nella generazione dei buchi di tensione. Tramite questo dispositivo è possibile generare dei buchi di tensione sagomabili a piacimento, ad esempio con ampiezza variabile nel tempo. Lo studio, sebbene interessante, è ovviamente limitato alla particolare configurazione del generatore eolico a disposizione che, nelle sue vicinanze ha un convertitore di alta potenza (dedicato, in realtà, per studi di HVDC) che può essere riconfigurato ed utilizzato come generatore di buchi di tensione.

Un'altro approccio che mira allo sviluppo di una banco di test da laboratorio per il test generatori eolici è quello sviluppato da RWTH institute in Aachen [10, 11]. Questo banco di test si basa su una macchina elettrica controllata da un simulatore real-time, ove un motore elettrico viene controllato tramite un dispositivo hardware in the loop (HIL) in modo da simulare il comportamento aerodinamico e meccanico del rotore della turbina eolica. L'emulazione della parte meccanica è stata realizzata tramite un motore e da un riduttore, come mostrato in fig. 7, fig. 8 e fig. 9. Tale soluzione è indubbiamente molto interessante, ma richiede la realizzazione di un banco prova molto costoso e complesso, dovuto principalmente alla realizzazione della parte meccanica e ai rinforzi necessari nella struttura dell'edificio. Tali banchi prova non sono accessibili in termini di costo a normali laboratori di certificazione. Tale soluzione è stata già realizzata ad Aachen per potenze fino a 1 *MW* e l'estensione a potenze fino a 4 *MW* è prevista nei prossimi anni.

La strategia di emulazione proposta in questo progetto si basa sull'idea di evitare l'utilizzo di parti meccaniche, al fine di semplificare i requisiti del set-up di prova. Si basa, quindi, su una elaborazione digitale real-time che controlla un amplificatore di potenza in modo da emulare il comportamento del sistema composto da turbina eolica, riduttore meccanico, controllore dell'angolo delle pale e generatore a magneti permanenti, come mostrato in fig. 10. Inoltre, la tensione di

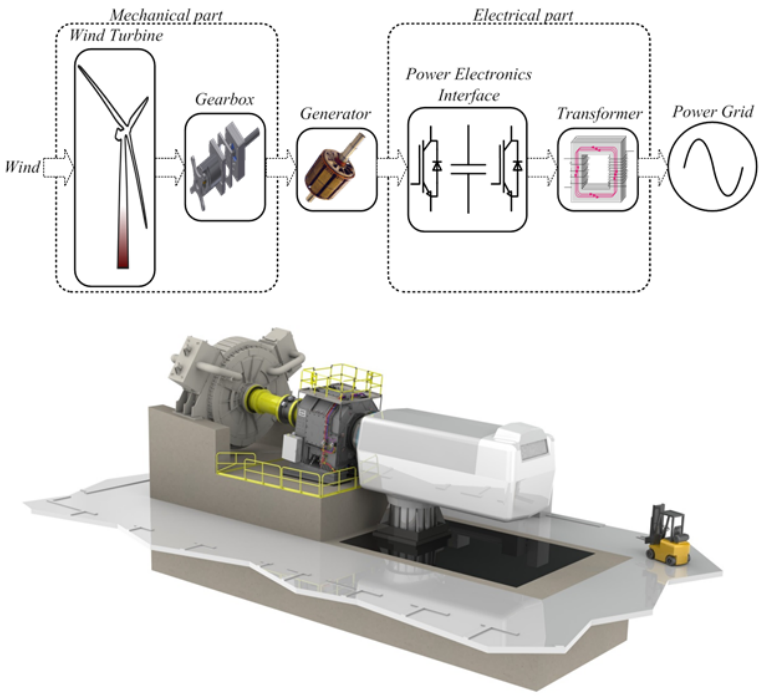


Figura 7: Emulatore realizzato dalla facoltá di Aachen, schema di base e rendering [10].

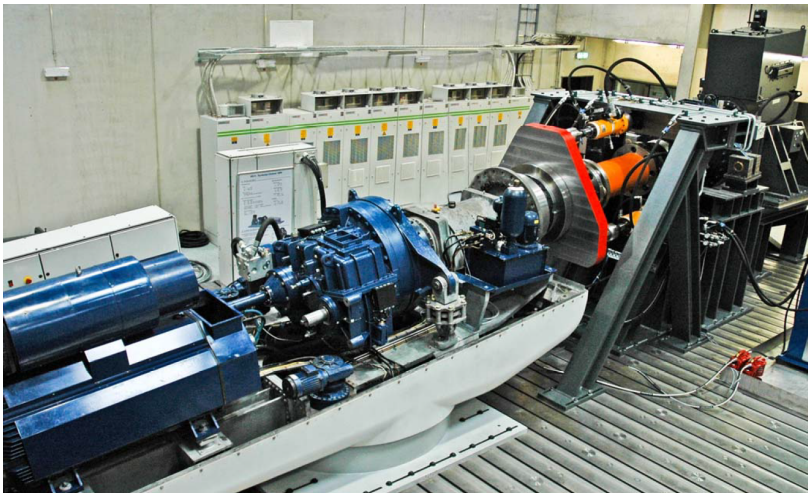


Figura 8: Realizzazione dell'emulatore di Aachen [11].

rete viene emulata da un inverter in modo da poterne controllare le caratteristiche, permettendo di effettuare in laboratorio test di compatibilità relativi a buchi di tensione di rete (low voltage ride through, LVRT) o sovratensioni (high voltage ride through), similmente a quanto già in essere per i test sui generatori di tipo fotovoltaico. Si è scelto

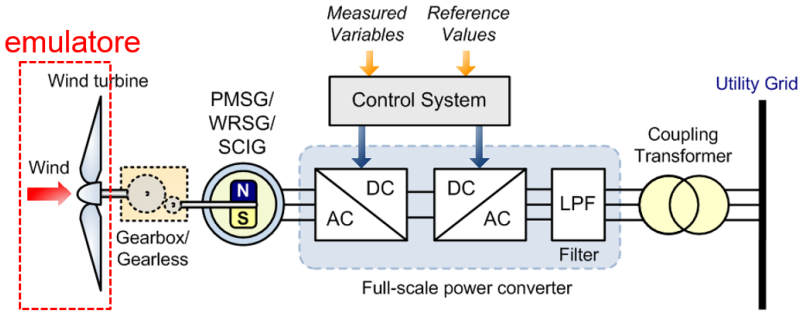


Figura 9: Emulazione meccanica del sistema turbina-riduttore realizzata da Aachen [11].

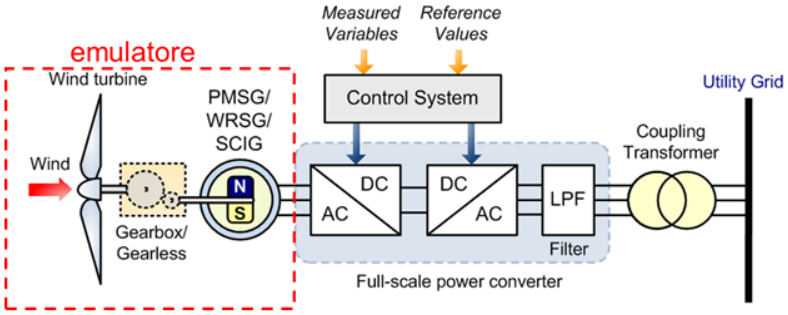


Figura 10: Emulazione elettrica del sistema turbina-riduttore-generatore in via di sviluppo.

di emulare turbine eoliche full-converter realizzate con generatori a magneti permanenti, come indicato in fig. 11. In tali applicazioni, l'interposizione di un convertitore back-to-back tra il generatore e la rete permette di rendere indipendente la velocità di rotazione del rotore del generatore eolico dalla frequenza di rete.

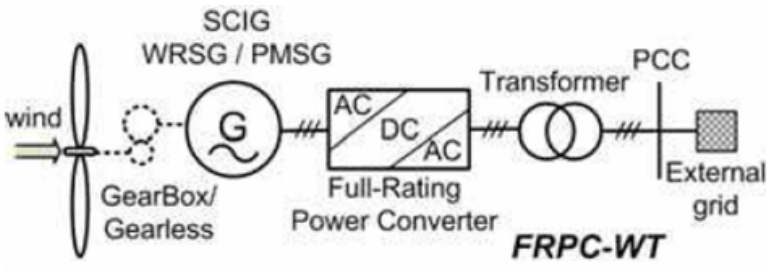


Figura 11: Configurazione di una turbina basata su generatore a magneti permanenti e convertitore elettronico di tipo back-to-back full-power.

Per implementare l'emulatore in esame è necessario, come primo passo, ricavare il modello matematico del sistema da emulare. In seguito tale modello viene implementato, tramite un opportuno software, nel simulatore in tempo reale (real-time) che pilota l'amplificatore di potenza.

2.2 DESCRIZIONE DELLA PARTE HARDWARE DELL'EMULATORE

Una volta sviluppato il modello matematico dell'emulatore del generatore eolico è necessario tradurlo in linguaggio di programmazione e implementarlo nel sistema hardware real-time che si prende carico dell'esecuzione dell'intera simulazione.

Per la realizzazione hardware dell'emulatore si è scelto di utilizzare la piattaforma LAUNCHXL-F28277S realizzata da Texas Instruments, illustrata in fig. 12, e dotata di microcontrollore TMS320F28377S (fig. 13) della famiglia logica C2000 Delfino. Questa scheda, oltre ad

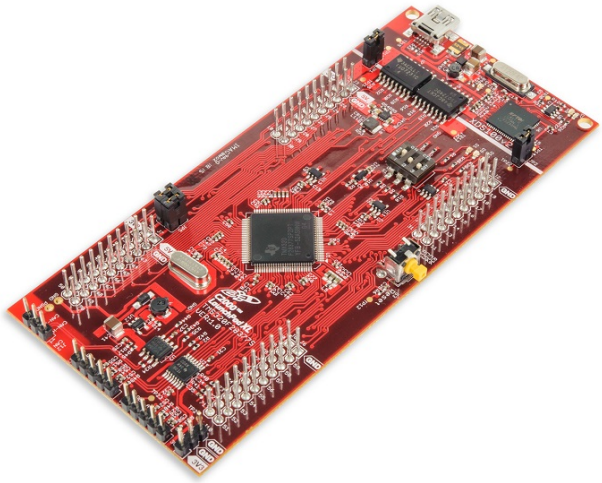


Figura 12: Scheda Texas Instruments LAUNCHXL-F28277S.

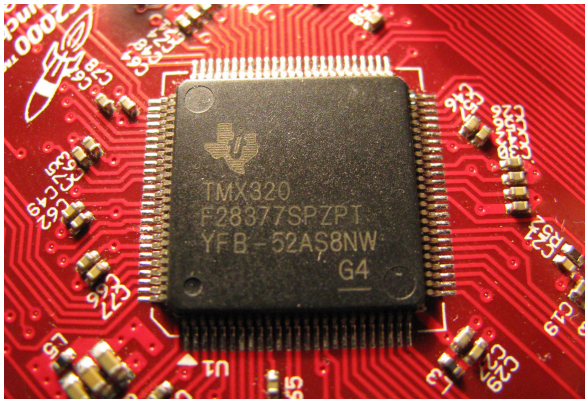


Figura 13: Microcontrollore-DSP C2000 Delfino TMS320F28377S.

avere già un emulatore incorporato che ne facilita la programmazione ed il debug, dispone di 14 ingressi ADC, 3 DAC, 8 uscite PWM, porte di comunicazione seriale SCI UART, SPI e I2C, numerosi I/O riconfigurabili, unità di calcolo floating point, coprocessore e DMA. Il DSP presenta una frequenza di clock massima di 200 MHz e vanta prestazioni pari a 400 MIPS con l'ausilio del coprocessore.

Il costo di questa piattaforma è estremamente basso, si parla di 27*Euro* per singola scheda.

Il microcontrollore è programmabile in linguaggio C tramite l'ambiente di sviluppo integrato Texas Instruments Code Composer Studio 6 basato su ambiente Eclipse (fig. 14). La licenza del software di programmazione Code Composer Studio è gratuita per tutti i processori della famiglia C2000. Oltre alle ragioni legate all'economicità della scheda hardware e dell'ambiente di programmazione è stata scelta questa soluzione anche per motivi di riutilizzabilità del codice. C infatti è un linguaggio di programmazione standardizzato che non dipende dalla versione del software di programmazione e per il quale non si necessita di licenza. Successivamente alla scelta del-

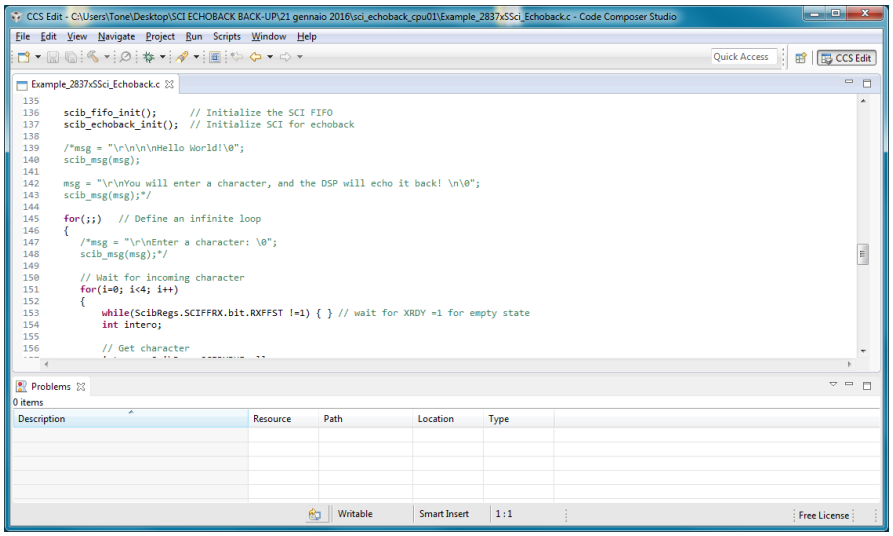


Figura 14: Ambiente di sviluppo Code Composer Studio 6.

la piattaforma hardware il modello dell'emulatore è stato tradotto in linguaggio C, debuggato, compilato e implementato nella memoria FLASH-ROM del microcontrollore mediante l'ambiente di sviluppo Code Composer Studio 6 per sistema operativo Microsoft Windows 7 su personal computer.

Il firmware sviluppato si basa su una routine di emulazione illustrata in fig. 15 e composta da tre fasi principali:

1. Acquisizione ingressi
2. Operazioni di calcolo
3. Aggiornamento uscite

Essendo il simulatore da implementare un sistema real-time, questa serie di operazioni deve necessariamente terminare entro un tempo prestabilito senza mai oltrepassarne il limite. Per questo motivo il sistema deve essere progettato in modo da non introdurre operazioni di durata non prevedibile che potrebbero allungare in modo indefinito la durata totale del ciclo.

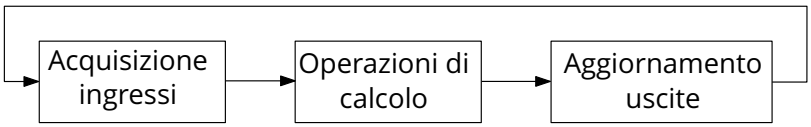


Figura 15: Routine di emulazione.

La caratteristica principale dei sistemi di questo tipo è un sistema di temporizzazione preciso che determina l'inizio del ciclo illustrato in fig. 15. Per implementare la temporizzazione è stato fatto uso di uno dei tre timer di precisione all'interno del processore. Il timer è di base un contatore il quale incremento avviene alla frequenza del clock della CPU, che nel microcontrollore TMS320F377S presenta una frequenza di 200 MHz. In base alla temporizzazione desiderata viene impostato un valore di fine conteggio, quando il contatore raggiunge questo valore viene resettato e viene lanciato un interrupt alla massima priorità. Questo interrupt è stato collegato alla routine di emulazione che esegue le tre operazioni base descritte in fig. 15. In questo modo viene garantita la consistenza del tempo di ciclo del sistema che può essere in questo modo considerato real-time.

3 INTERFACCE DI COMUNICAZIONE SERIALE

3.1 INTRODUZIONE

La trasmissione seriale é una modalitá di comunicazione tra dispositivi digitali nella quale i bit sono comunicati uno di seguito all'altro e giungono sequenzialmente al ricevente nello stesso ordine in cui li ha trasmessi il mittente. Quando due dispositivi sono in comunicazione tra di loro esistono tre possibili modi in cui la comunicazione puó svolgersi:

1. **Simplex:** i dati sono trasmessi in un unica direzione
2. **Half Duplex:** la comunicazione é bidirezionale, ma avviene su un'unica linea
3. **Full Duplex:** la comunicazione avviene attraverso due linee, una per ciascuna direzione

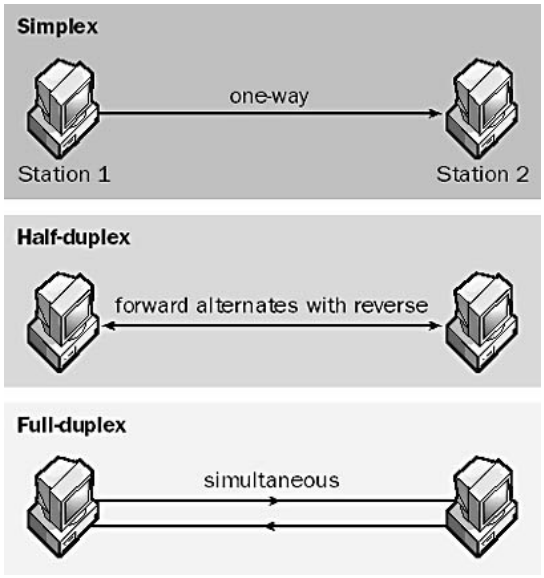


Figura 16: Modalitá di trasmissione.

Le unitá di misura usate per la velocitá di trasmissione su una linea seriale sono:

- *bps:* numero di bit trasmessi per secondo
- *baud:* numero di simboli trasmessi per secondo

Solo se il segnale trasmesso puó assumere simboli di dimensione pari ad 1 bit, allora *baud* e *bps* sono equivalenti. Ad esempio un segnale a 300 *baud* che contiene 4 bit di informazione per ogni simbolo ha una velocitá di trasmissione pari a 1200 *bps*.

La comunicazione seriale puó essere di due tipi: sincrona e asincrona.

- **Trasmissione seriale sincrona:** Nella trasmissione sincrona trasmettitore e ricevitore utilizzano per la trasmissione e la ricezione lo stesso clock. Solitamente si utilizzano due tecniche di sincronizzazione:
  1. il trasmettitore genera ed invia al ricevitore un unico segnale contenente sia le informazioni di temporizzazione che i dati; il ricevitore utilizza questo segnale sia per rifasare il proprio clock tramite un circuito PLL (*Phase Lock Loop*), sia per estrarre i dati.
  2. il trasmettitore ed il ricevitore si scambiano i dati attraverso un canale ed il segnale di temporizzazione su un altro canale.

  Il vantaggio di questo tipo di trasmissione é la possibilitá di raggiungere frequenze molto alte.

- **Trasmissione seriale asincrona:** Nella trasmissione asincrona i due sistemi che stanno dialogando fra loro usano due clock diversi per cui non si puó garantire che i due clock siano in fase o siano alla stessa frequenza. Il ricevitore non ha alcuna informazione sulla temporizzazione con cui arrivano i dati. Per garantire che trasmettitore e ricevitore siano sincronizzati, le informazioni da inviare sono suddivise in piccoli blocchi di 5, 6, 7 o 8 bit che sono precedute e seguite da bit di sincronizzazione detti bit di *start* e di *stop*. É chiaro che l'aggiunta di questi bit di controllo diminuisce l'efficienza della trasmissione poiché vi é un elevato rapporto fra bit di controllo e bit che contengono informazioni. Questo tipo di trasmissione si usa allora quando i dati vanno inviati in maniera sporadica. Ogni blocco é costituito allora da:
  - un bit di start che individua il passaggio dalla condizione di riposo alla condizione di trasmissione.
  - il blocco dei dati che possono essere da 5 a 8 piú un eventuale bit di paritá.
  - 1, 1.5, 2 bit di stop che indicano la fine della trasmissione. Vedi fig. 17.

  Le parole trasmesse quindi possono essere separate da un qualsiasi intervallo di tempo.

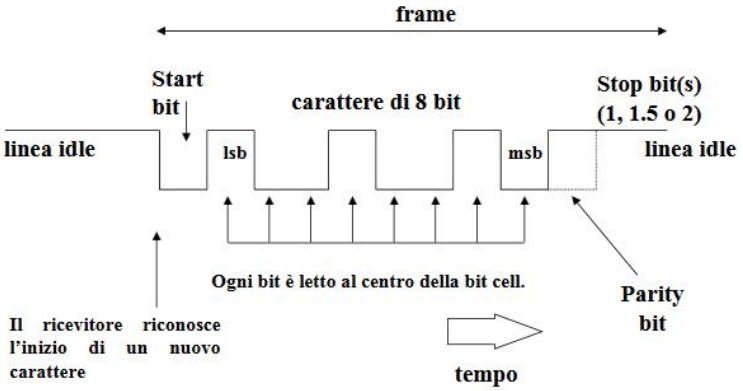


Figura 17: Trasmissione asincrona.

3.2 INTERFACCIA DI COMUNICAZIONE I2C

I2C, abbreviazione di Inter-Integrated Circuit, é un'interfaccia di comunicazione seriale che consente il colloquio fra piú dispositivi collegati fra loro attraverso un bus composto da due linee. Il classico bus I2C é composto da almeno un dispositivo *master* ed uno *slave*. La situazione piú frequente vede un singolo dispositivo *master* e piú *slave*. Il protocollo hardware dell'I2C richiede due linee seriali di comunicazione:

- *SDA* (Serial DAta) per la comunicazione dati
- *SCL* (Serial CLock) per il clock

Per la presenza del segnale di clock l'I2C é un'interfaccia di comunicazione seriale sincrona. La fig. 18 illustra un esempio di piú moduli I2C collegati per un trasferimento bidirezionale da un dispositivo ad altri dispositivi.

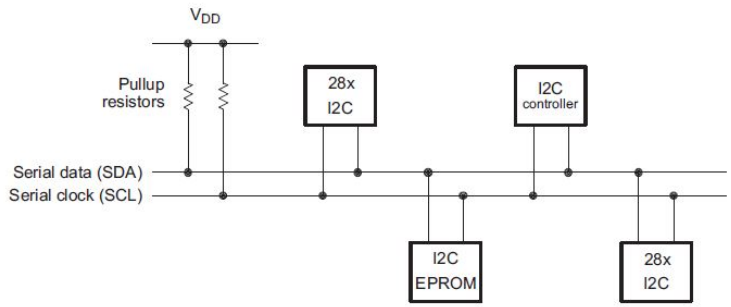


Figura 18: Moduli I2C collegati [12].

Il modulo I2C ha le seguenti caratteristiche:

- supporto per il formato di trasferimento a 8 bit
- modalitá di indirizzamento sia a 7 bit che a 10 bit

- chiamata generale
- modalitá con byte di START
- supporto per piú trasmettitori *master* e ricevitori *slave*
- supporto per piú trasmettitori *slave* e ricevitori *master*
- modalitá combinata di trasmissione/ricezione del *master* e ricezione/trasmissione dello *slave*
- velocitá di trasferimento dati da 10 kbps fino a 400 kbps

Ogni dispositivo collegato ad un bus I2C é riconosciuto da un indirizzo univoco. Ogni dispositivo puó funzionare sia come un trasmettitore o ricevitore, a seconda della funzione del dispositivo. Un dispositivo collegato al bus I2C puó essere considerato come il master o come lo slave quando si eseguono trasferimenti di dati. Viene definito *master* il dispositivo che avvia un trasferimento di dati sul bus e genera i segnali di clock per consentire tale trasferimento. Durante questo trasferimento, qualsiasi dispositivo indirizzato dal *master* viene considerato uno slave. I moduli I2C supportano la modalitá *multi-master*, in cui é possibile collegare uno o piú dispositivi in grado di controllare il bus. Osservando la fig. 18 i moduli I2C utilizzano la linea seriale *SDA* per la comunicazione dei dati e la linea *SCL* per il clock di sincronizzazione. Queste due linee trasportano informazioni ad esempio tra il dispositivo 28x (fig. 18) e gli altri dispositivi collegati al bus. Le linee *SDA* e *SCL* sono entrambe bidirezionali. Ognuna di esse deve essere collegata ad una tensione di alimentazione positiva utilizzando un resistore di pull-up. Quando il bus é libero entrambe le linee sono a livello logico alto. Sono presenti due principali tecniche di trasferimento:

1. **Modalitá Standard:** si inviano esattamente n dati, dove n é un valore di dati programmato nel modulo I2C.
2. **Modalitá Repeat:** mantiene l'invio dei dati fino a quando non si verifica una condizione di STOP o una nuova condizione di START.

3.3 INTERFACCIA DI COMUNICAZIONE SPI

L'acronimo *SPI*, che sta per **S**erial **P**eripheral **I**nterface, si riferisce ad una periferica di interfaccia con caratteristiche seriali. Nella comunicazione SPI sono presenti un unico dispositivo principale detto *master* e un certo numero di dispositivi secondario detti *slave* connessi al master mediante un bus dotato di quattro linee su ci viaggiano i segnali necessari alla comunicazione SPI tra i vari dispositivi. Il master controlla il bus, emette il segnale di clock, decide quando iniziare e terminare la comunicazione. Il bus SPI si definisce:

- di tipo seriale.
- sincrono, per la presenza di un clock che coordina la trasmissione e la ricezione dei singoli bit e determina la velocitá di trasmissione.
- full-duplex, in quanto il colloquio tra i dispositivi puó avvenire contemporaneamente in trasmissione e ricezione.

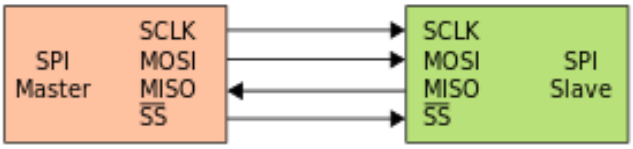


Figura 19: Bus SPI: connessioni tra un *master* ed un singolo *slave*.

L'SPI usa i seguenti segnali per serializzare lo scambio di dati con un altro dispositivo:

- **SS:** questo segnale é conosciuto come **S**lave **S**elect ed é emesso dal *master* per scegliere con quale dispositivo *slave* vuole comunicare. Quando passa a livello logico basso il dispositivo *slave* si mette in attesa pronto ad ascoltare il segnale di clock e i dati.
- **SCLK: S**erial **CL**oc**K**. Clock seriale emesso dal *master* che controlla l'invio e la ricezione dei dati.
- **SDO/MOSI: S**erial **D**ata **O**utput/**M**aster **O**utput **S**lave **I**nput. Linea di uscita seriale dei dati del *master*.
- **SDI/MISO: S**erial **D**ata **I**nput/**M**aster **I**nput **S**lave **O**utput. Linea di ingresso seriale dei dati del *master*.

I dati sono scambiati sempre fra i vari dispositivi collegati al bus. Nel protocollo SPI nessun dispositivo puó essere solo un trasmettitore o solo un ricevitore. Ogni dispositivo ha due linee di dati, una di ingresso SDI e una di uscita SDO. Lo scambio di dati é scandito da un segnale di temporizzazione detto clock presente sulla linea denominata con SCLK (fig. 19). Il clock é generato dal *master* che, tramite esso, sincronizza gli eventi, controlla quando i dati possono essere trasmessi e quando possono essere letti. Soltanto il dispositivo *master* puó controllare la linea di clock SCLK. Questo é un vantaggio perché il clock puó variare senza distruggere o corrompere i dati. La frequenza dei dati (*data-rate*) cambierá semplicemente al variare del clock. Ció rende l'SPI ideale quando si ha un clock impreciso. La sincronizzazione con il segnale di clock SCK dei dati in uscita sul bus SDO avviene in corrispondenza dei fronti di salita o discesa del clock. I dati presenti sul bus d'ingresso SDI vengono letti in corrispondenza del fronte

opposto a quello impostato per i dati in uscita. In fig. 20 viene raffigurato un esempio della sincronizzazione con il segnale di clock. Le

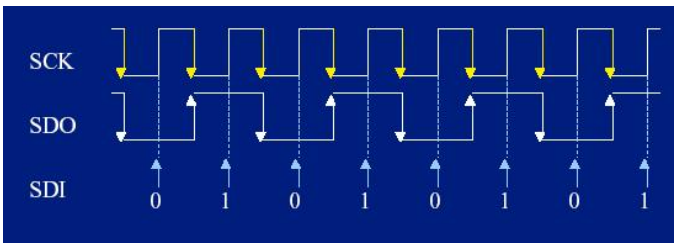


Figura 20: Diagramma temporale dei segnali SCLK, SDO, SDI.

frecce indicano quando si verifica un fronte (salita o discesa) del clock e dei segnali SDO e SDI. Nel caso del SDI il dato é campionato sul fronte del clock opposto a quello in cui cambia l'uscita.

É possibile realizzare comunicazioni con diversi dispositivi *slave* che, collegati allo stesso bus, sono controllati dal clock generato dall'unico *master* che controlla il bus. Quando nel sistema esiste piú di uno slave il master, per selezionare il dispositivo con cui comunicare, attiva il segnale SS. Il segnale é spesso attivo al livello basso per migliorare l'immunitá al rumore del sistema.

Esistono due modalitá per collegare e gestire piú dispositivi *slave* con un *master*:

1. Dispositivi *slave* controllati singolarmente
2. Dispositivi *slave* connessi in catena

Slave controllati singolarmente

La linea SS é dedicata all'abilitazione del dispositivo slave da parte del master, il quale puó abilitare un qualsiasi dispositivo slave connesso a trasmettere. La linea SS, normalmente attiva bassa, in caso di disabilitazione (transizione a livello logico alto) lascia il dispositivo slave con uscita in alta impedenza e quindi isolato completamente dal bus indifferentemente dall'esistenza del segnale di clock. Il numero di dispositivi slave che si possono connettere al bus é limitato esclusivamente dal numero di possibili linee di Slave Select gestibili dal dispositivo master. La frequenza di clock, e di conseguenza la velocitá del bus, puó raggiungere, con questo standard, livelli anche elevati nell'ordine delle decine di MHz ed anche oltre.

- *Vantaggi:* comunicazione piú rapida tra master e singoli slave.
- *Svantaggi* necessitá di avere un pin SS per ogni dispositivo slave.

Slave connessi in catena

In questo caso l'uscita di un dispositivo slave é collegata all'ingresso del dispositivo slave successivo nella catena (fig. 22).

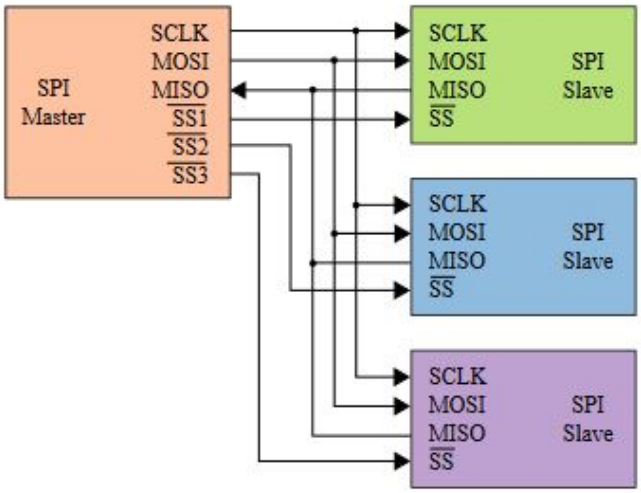


Figura 21: Esempio di connessione diretta tra un master e 3 slave controllati singolarmente.

- *Vantaggi:* uso di un unico pin SS del master per selezionare i diversi dispositivi slave.
- *Svantaggi* minore velocitá di aggiornamento dei singoli dispositivi slave.

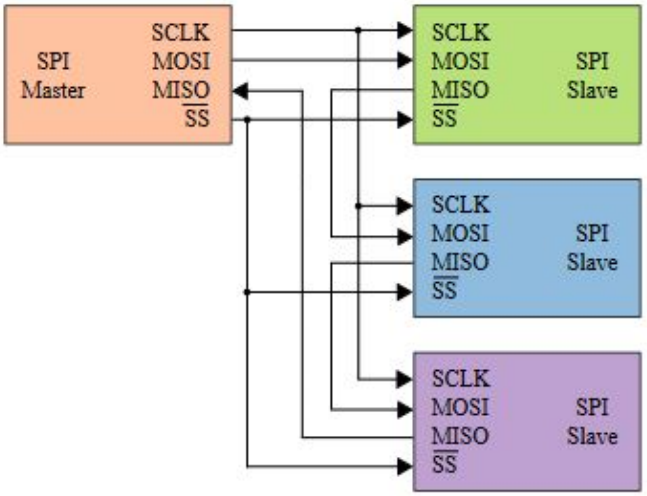


Figura 22: Esempio di *master* connesso a 3 *slave* collegati in catena.

3.4 INTERFACCIA DI COMUNICAZIONE SCI

SCI, acronimo di **S**erial **C**omunications **I**nterface, é una porta seriale asincrona a due linee di cui una dedicata alla trasmissione dati (TX) e la seconda dedicata alla ricezione (RX). I moduli SCI supportano

comunicazioni digitali tra la CPU e altre periferiche asincrone sia nella modalitá half-duplex che nella modalitá full-duplex. Le principali caratteristiche dell'interfaccia SCI sono:

- 2 linee dedicate:
  1. SCITXD: linea riservata alla sola trasmissione
  2. SCIRXD: linea riservata alla sola ricezione
- modalitá operativa sia half- che full-duplex
- utilizza i livelli logici TTL
- 4 flag di rilevamento degli errori di: paritá, overrun, framing e break detection
- formato NRZ (Non Return to Zero)

In fig. 23 é rappresentata la struttura dell'interfaccia SCI contenuta piattaforma LAUNCHXL-F28277S di Texas (fig. 12).

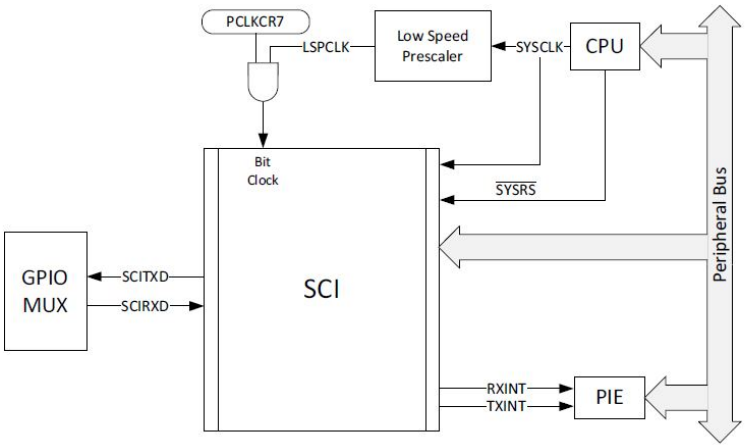


Figura 23: Schema logico interfaccia SCI [12].

Il singolo pacchetto dati utilizzato dall'interfaccia SCI viene detto *carattere*. Esso puó avere lunghezza variabile da 1 a 8 bit. Ciascun *carattere* é accompagnato da altri bit necessari alla comunicazione fra dispositivi. Questo nuovo pacchetto viene chiamato *frame* (fig. 24). Il *frame* é quindi costituito da:

- 1 bit di START
- 1 o 2 bit di STOP
- lunghezza del carattere variabile da 1 a 8 bit
- 1 bit di controllo paritá per rilevamento di eventuali errori di trasmissione
- 1 bit extra per distinguere l'indirizzo dal dato (solo per la modalitá *address-bit*)

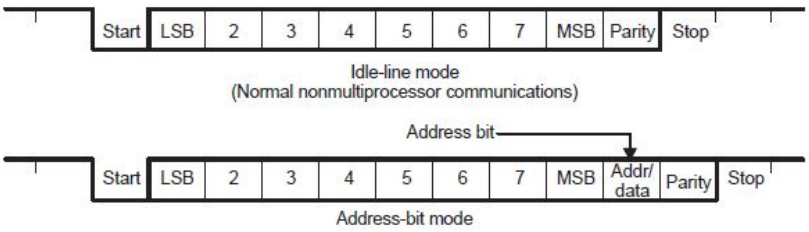


Figura 24: Esempi di frame [12].

Gli elementi principali utilizzati nell'interfaccia SCI nel funzionamento *full-duplex* sono illustrati in fig. 25 e comprendono:

- registri FIFO a 16 livelli sia per trasmissione che per ricezione.
- un trasmettitore TX e i suoi principali registri:
  - SCITXBUF: registro buffer dei dati da trasmetttere. Contiene i dati da trasmettere caricati dal processore di sistema.
  - TXSHF: accetta i dati dal registro SCITXBUF e li sposta nella linea di trasmissione SCITXD un bit alla volta.
- un ricevitore RX e i suoi principali registri:
  - RXSHF: sposta i dati ricevuti in ingresso dal SCIRXD un bit alla volta.
  - SCIRXBUF: registro buffer dei dati ricevuti. Contiene i dati raccolti da RXSHF e possono essere letti dal processore di sistema.
- TXINT: interrupt di trasmissione
- RXINT: interrupt di ricezione
- *baud-generator* programmabile

3.4.1 *Formato di comunicazione dell'interfaccia SCI*

La comunicazione seriale asincrona SCI utilizza due linee per la trasmissione/ricezione dei dati. Come sopra citato il pacchetto dati elementare, detto *frame*, é composto da: 1 bit di START, 1 o 2 bit di STOP, il dato o carattere e 1 bit di controllo paritá (fig. 24). Il clock di sistema (SCICLK) impiega 8 periodi per ogni bit. Il ricevitore dati entra in funzione al rilevamento di un bit di START valido. Un bit di START valido viene identificato da quattro periodi consecutivi di clock SCICLK in un biy di valore 0 (livello logico basso), come mostrato in fig. 26. Se il valore del bit non é 0 il processore di sistema riparte a cercare un altro bit di START.

Per i bit successivi al bit di START, il processore determina il valore del bit facendo tre campioni con il clock nel mezzo del bit. Questi

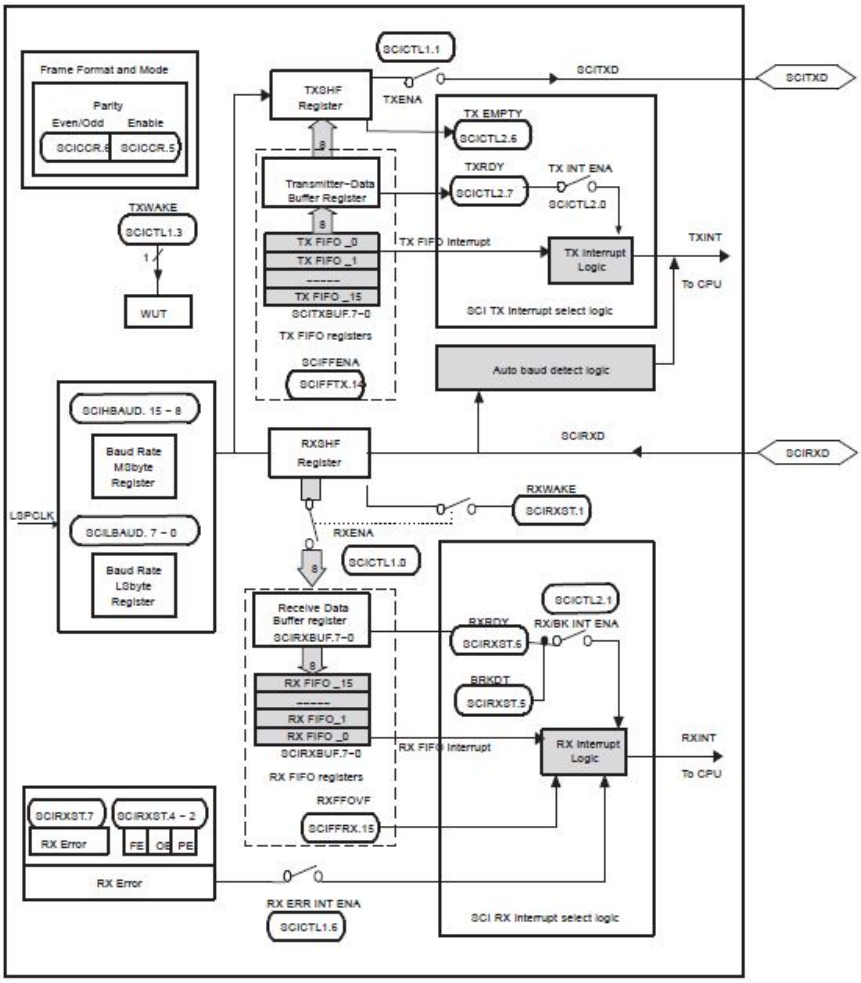


Figura 25: Schema a blocchi interfaccia SCI [12].

campioni vengono fatti in corrispondenza del quarto, quinto e sesto periodo di clock e la determinazione del valore del bit é basata su maggioranza di due campioni su tre. Ad esempio se di questi tre campioni due rilevano un valore basso del bit e uno alto, al bit in esame verrá assegnato il valore basso. La fig. 26 illustra il formato di comunicazione asincrona durante la fase di rilevamento del bit di START e del bit LSB (Least Significant Bit) del carattere.

Dal momento che il ricevitore si sincronizza al *frame* ricevuto, la trasmissione non deve sincronizzarsi con alcun segnale di clock.

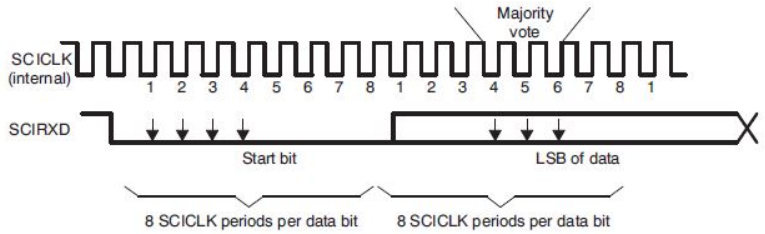


Figura 26: Esempio di formato di comunicazione asincrona SCI [12].

La fig. 27 illustra un esempio di ricezione dati con questi parametri:

- modalitá di risveglio *address-bit*
- carattere avente lunghezza di 6 bit

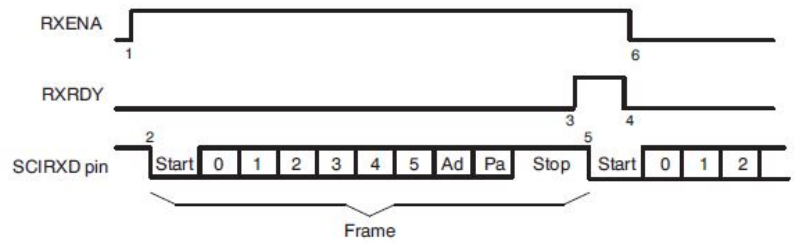


Figura 27: Esempio di ricezione seriale SCI [12].

La ricezione si svolge con i seguenti passi:

1. il bit di flag RXENA passa al livello logico alto e attiva la ricevente
2. i dati giungono sul pin SCIRXD e il bit di START viene rilevato
3. i dati ricevuti vengono shiftati dal RXSHF al registro buffer SCIRXBUF; é richiesto un interrupt. Il bit di flag RSRDY passa a livello alto per segnalare che un nuovo carattere é stato ricevuto
4. il processore di sistema legge il registro buffer SCIRXBUF; il bit di flag RXRDY viene automaticamente resettato

5. il successivo byte di dati arriva sul pin SCIRXD; viene rilevato il bit di START
6. il bit di flag RXENA passa a livello basso per disattivare il ricevitore. I dati continuano ad entrare in RXSHF ma non vengono trasferiti al registro buffer ricevente SCIRXBUF.

La fig. 28 illustra un esempio di trasmissione dati con questi parametri:

- modalitá di risveglio *address-bit*
- carattere avente lunghezza di 3 bit

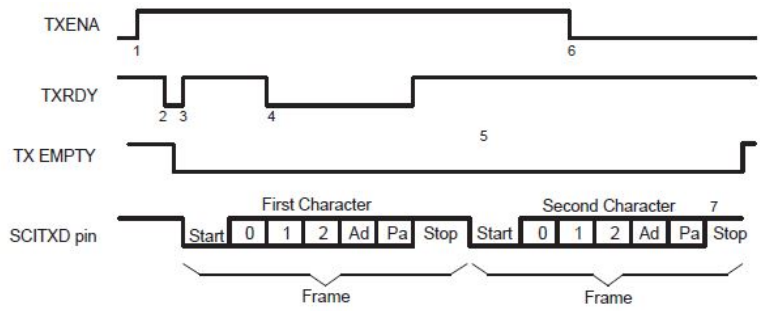


Figura 28: Esempio di trasmissione seriale SCI [12].

La trasmissione si svolge con i seguenti passi:

1. il bit di flag TXENA passa al livello logico alto abilitando la trasmittente
2. i dati da trasmettere vengono scritti nel buffer SCITXBUF dal processore di sistema. Il flag TXRDY passa al livello logico basso
3. i dati vengono trasferiti al registro TXSHF. Il trasmettitore é pronto per un secondo carattere e richiede un interrupt (flag TXRDY passa a livello alto)
4. il processore scrive un secondo carattere nel buffer SCITXBUF dopo aver rilevato il flag TXRDY a livello alto. Una volta scritto il secondo carattere nel buffer SCITXBUF il flag TXRDY ritorna a livello basso
5. la trasmissione del primo carattere é completa. Inizia il trasferimento del secondo carattere nel registro TXSHF
6. il flag TXENA passa a livello basso per disattivare la trasmittente; l'interfaccia finisce di trasmettere il secondo carattere in corso
7. la trasmissione del secondo carattere é completa; la trasmittente é pronta per un nuovo carattere

3.4.2 *UART*

Il comunissimo standard di trasmissione seriale *RS-232* denominato anche *UART* é basato sul protocollo di trasmissione seriale SCI. In particolare l'UART specifica delle caratteristiche elettriche che la comunicazione seriale deve assumere in modo da renderla piú robusta da eventuali disturbi elettromagnetici esterni. In fig. 29 é rappresentato un'esempio di porta seriale RS-232 con connettore DE9P a 9 pin.

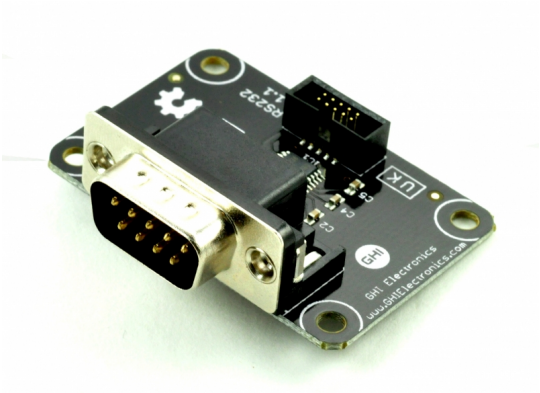


Figura 29: Esempio di porta seriale *RS-232*.

3.5 PROTOCOLLO DI COMUNICAZIONE ETHERNET

Il protocollo *Ethernet* é una famiglia di tecnologie necessarie alla realizzazione e al funzionamento di reti locali, dette *LAN* (Local Area Network) fra dispositivi elettronici. É stata sviluppata a fine anni '70 nei laboratori sperimentali dello Xerox Parc in California e definisce le specifiche tecniche a livello fisico (connettori, cavi, tipo di trasmissione) e del modello architetturale di rete ISO-OSI. L'obbiettivo originale era ottenere una rete locale con una trasmissione afffidabile a 3 Mbit/s su cavo coassiale fra diversi dispositivi in condizioni di traffico contenuto, ma in grado di tollerare con successo occasionali picchi di traffico (fig. 30). Con il passare del tempo il protocollo *Ethernet* é diventato il sistema piú diffuso per la realizzazzione di reti LAN per diverse ragioni:

- é stata una delle prime forme di comunicazione fra apparati telematici. Questo ha facilitato la sua diffusione.
- é economica e facile da usare.
- la rapida diffusione delle componenti hardware ne ha facilitato l'adozione.
- ha un funzionamento molto stabile ed é soggetta a pochi problemi.

Figura 30: Schema di rete Ethernet semplice.

- é adeguata all'utilizzo di protocolli di livello superiore dell'IP (Internet Protocol) quali ad esempio TCP e UDP.

Negli anni sono stati sviluppati 3 modelli di collegamento Ethernet:

1. **10BASE-T:** standard di trasmissione dati per reti LAN ed é caratterizzato da una velocitá di trasferimento di 10 Mbit/s. I cavi sono composti da due doppini di derivazione telefonica intrecciati l'uno con l'altro con connettori di tipo RJ-45, anch'essi di derivazione telefonica (fig. 31).

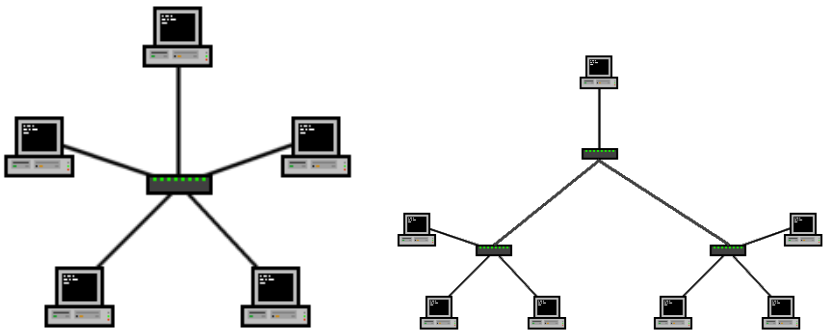
2. **Fast Ethernet:** standard di trasmissione dati per reti LAN in cui la velocitá di trasmissione é di 100 Mbit/s. Anche in questo caso, come nel modello 100BASE-T, il cavo utilizzato nel collegamento é quello con due doppini intrecciati e connettori RJ-45.

3. **Gigabit Ethernet:** rappresenta lo standard in assoluto piú diffuso oggi. Esso é un'evoluzione del Fast Ethernet. Come lascia intuire il nome, il Gigabit Ethernet é caratterizzato da una velocitá di trasferimento massima di 1 Gbit/s, sempre utilizzando il cavo con due doppini intrecciati e connettori RJ-45.

Figura 31: Esempio di cavo Ethernet con doppini intrecciati e connettori RJ-45.

Il cavo Ethernet ha una capacitá limitata sia in lunghezza sia in capacitá di traffico, per cui le LAN di grosse dimensioni vengono suddivise in reti piú ridotte interconnesse tra loro da particolari nodi tra i quali possiamo trovare degli *hub* (fig. 32) o elementi piú sofisticati come *switch* (fig. 32)che hanno il compito di smistare correttamente i dati in transito nella rete.

Lo *switch* é un apparato che consente ad ogni dispositivo collegato alla rete LAN (detto *host*) di essere connesso direttamente allo switch stesso. Allo switch viene anche collegato un cavo Ethernet che permette la comunicazione dello switch verso altri segmenti di rete LAN. In questo modo lo switch intercetta i pacchetti dati (detti *frame*) e li ridireziona ad un host oppure su altri segmenti Ethernet. La gestione dei frame, quindi, é ottimizzata perché questi sono subito reindirizzati alla destinazione evitando, per quanto possibile, collisioni.



(a) *Esempio di rete con hub.*. (b) *Esempio di rete con switch.*.

Figura 32: Esempi di rete con dispositivi di gestione del traffico

L'elemento comune di ogni rete Ethernet é la struttura del pacchetto dati elementare, chiamato *frame*. Composto da 7 elementi differenti, il frame é responsabile di indirizzare e trasportare i dati della comunicazione tra due nodi della stessa rete LAN. In fig. 33 é raffigurato un esempio di *frame Ethernet*. Ciascun *frame* viene trasmesso in modo seriale.

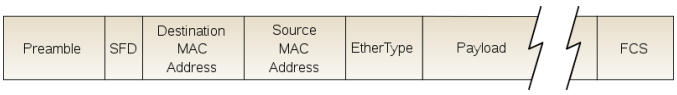


Figura 33: Esempio di *frame Ethernet* [2].

- *Preamble* (preambolo) di 7 byte: questi primi byte hanno valore 10101010 e servono a "svegliare" gli adattatori del ricevente e a sincronizzare gli oscillatori con quelli del mittente.
- *Start Frame Delimiter* (SFD), di 1 byte: serve a delimitare il "preambolo" e segnare l'inizio del pacchetto dati.
- *Destination MAC Address* (indirizzo di destinazione), di 6 byte: corrisponde agli indirizzo MAC del destinatario, esso rap-

presenta l'indirizzo fisico identificativo assegnato in modo univoco dal produttore di ogni scheda di rete e quindi associato univocamente ad ogni dispositivo connesso alla LAN.

- *Source MAC Address* (indirizzo sorgente), anch'esso di 6 byte: corrisponde agli indirizzo MAC del mittente.
- *EtherType* (tipo), di 2 byte: questo campo indica il tipo di protocollo del livello di rete in uso durante la trasmissione.
- *Payload* (campo dati), da 46 byte a 1,5 kbyte: contiene le informazioni vere e proprie scambiate nella comunicazione.
- *Frame Check Sequence* (controllo a ridondanza ciclica *CRC*), di 4 byte: permette di rilevare se sono presenti errori di trasmissione; in pratica, il ricevente calcola il CRC mediante un algoritmo e lo confronta con quello ricevuto in questo campo.

Ethernet é un protocollo di trasmissione seriale che fornisce al livello di rete un servizio senza connessione. In pratica, il mittente invia il frame nella LAN: il frame attraversa tutta la LAN e viene ricevuto da tutti i dispositivi connessi, ma solo l'adattatore che vi riconosce il proprio indirizzo di destinazione lo recepirá, mentre tutti gli altri lo scarteranno.

Il frame ricevuto puó contenere errori, la maggior parte dei quali sono verificabili dal controllo CRC. Un frame che non supera il controllo CRC viene scartato. Ethernet non prevede la ritrasmissione del frame scartato, né una notifica della sua perdita agli strati superiori. Ethernet non é quindi affidabile, ma grazie a ció é semplice ed economica. Il compito di provvedere alla ritrasmissione dei frame perduti viene demandato agli strati superiori del protocollo del livello di rete (ad esempio il protocollo TCP).

La comunicazzione Ethernet consente di utilizzare la suite di protocolli internet definiti dall'IP (Internet Protocol). L'IP indica un insieme di protocolli di rete su cui si basa il funzionamento logico della rete Internet. Esso si occupa di gestire l'indirizzamento dei dispositivi connessi e l'instradamento dei dati trasmessi. A ciascun dispositivo viene infatti assegnato un indirizzo IP che lo identificherá in modo non ambiguo in rete. Le funzionalitá di instradamento, invece, consentono di selezionare il percorso migliore per veicolare un messaggio verso un dato nodo destinatario, noto il suo indirizzo IP.

Gli indirizzi IP possono essere assegnati ai dispositivi connsessi alla rete in due modalitá:

1. **indirizzo statico:** indirizzo assegnato in maniera permanente al dispositivo connesso. Fintanto che il dispositivo resta connesso alla rete l'indirizzo IP statico non varia nel tempo.
2. **indirizzo dinamico:** indirizzo utilizzato per identificare dispositivi non permanenti in una rete LAN. Un server presente nella

LAN (detto *server DHCP*) assegna dinamicamente e automaticamente l'indirizzo scegliendolo casualmente da un range preimpostato di indirizzi IP disponibili. *DHCP* é un protocollo di rete che permette ai dispositivi o terminali di una rete locale di ricevere automaticamente a ogni richiesta di accesso alla rete la configurazione IP necessaria per stabilire una connessione e operare sulla rete basata su Internet Protocol.

I'Internet Protocol mette a disposizione due protocolli per effettuare il trasporto corretto dei dati: il protocollo ***T**rasmission **C**ontrol **P**rotocol* (TCP) e il protocollo ***U**ser **D**atagram **P**rotocol* (UDP).

3.5.1 *Transfer Control Protocol*

Il *Transfer Control Protocol*, anche chiamato *TCP*, é un protocollo di rete a pacchetto di livello di trasporto, appartenente alla suite di protocolli internet IP, che si occupa del controllo di trasmissione ovvero rendere affidabile la comunicazione dati in rete tra mittente e destinatario.

Le principali caratteristiche sono:

- TCP é un protocollo orientato alla connessione, ovvero prima di poter trasmettere dati deve stabilire la comunicazione, negoziando una connessione tra mittente e destinatario, che rimane attiva anche in assenza di scambio di dati e viene esplicitamente chiusa quando non piú necessaria. Esso quindi possiede le funzionalitá per creare, mantenere e chiudere/abbattere una connessione.

- il servizio offerto da TCP é il trasporto di un flusso di byte bidirezionale tra due applicazioni in esecuzione su host differenti. Il protocollo permette alle due applicazioni di trasmettere contemporaneamente nelle due direzioni, quindi il servizio puó essere considerato full-duplex.

- TCP é un protocollo affidabile: garantisce la consegna dei pacchetti a destinazione. Grazie ad appositi segnali di controllo (*detti acknowledgements*) i pacchetti dati che non giungono a destinazione vengono ritrasmessi.

- TCP garantisce che i dati trasmessi, se giungono a destinazione, lo facciano in ordine e uno volta sola ("at most once"). Piú formalmente, il protocollo fornisce ai livelli superiori un servizio equivalente ad una connessione fisica diretta che trasporta un flusso di byte.

- TCP offre funzionalitá di controllo di errore sui pacchetti pervenuti grazie al campo *checksum* contenuto nel pacchetto dati elementare (detto PDU).

- TCP possiede funzionalitá di controllo di flusso tra terminali in comunicazione e controllo della congestione sulla connessione. Questo permette di ottimizzare l'utilizzo dei buffer di ricezione/invio sui dispositivi (controllo di flusso) e di diminuire il numero di pacchetti inviati in caso di congestione della rete.

3.5.2 *User Datagram Protocol*

Lo *User Datagram Protocol*, detto *UDP*, é un protocollo di rete di livello di trasporto, anch'esso appartenente alla suite di protocolli internet IP. A differenza del TCP, l'UDP é un protocollo di tipo connectionless, inoltre non gestisce il riordinamento dei pacchetti né la ritrasmissione di quelli persi, ed é perció generalmente considerato di minore affidabilitá. In compenso é molto rapido (non é presente latenza per riordino e ritrasmissione dei dati) ed efficiente per le applicazioni in tempo reale, anche dette *real-time*. Dato che le applicazioni *real-time* richiedono spesso un *bit-rate* minimo di trasmissione, l'UDP cerca di non ritardare eccessivamente la trasmissione dei pacchetti e si puó tollerare anche qualche perdita di dati.

Il pacchetto dati elementare del protocollo UDP viene detto *datagramma*. In fig. 34 é raffigurato lo schema a blocchi della struttura di un datagramma UDP. Esso é cosí strutturato:

- **Header:** intestazione del datagramma. Essa é costituita da 4 campi contenenti le informazioni di controllo necessarie al funzionamento del protocollo.
  - *Source Port* (16 bit): identifica il numero di porta di trasmissione sull'host del mittente del datagramma
  - *Destination Port* (16 bit): identifica il numero di porta d'ascolto sull'host del destinatario del datagramma
  - *Lenght* (16 bit): contiene la lunghezza totale in byte del datagramma UDP (Header + Payload)
  - *Checksum* (16 bit): contiene il codice di controllo del datagramma
- **Payload:** carico contenente l'effettiva informazione che si vuole trasmettere. E' composto da un solo campo di lunghezza variabile:
  - *Data* (64+ bit): contiene i dati effettivi dell'informazione

3.5.3 *Confronto TCP vs UDP*

Le principali differenze tra TCP e UDP sono:

- TCP é un protocollo orientato alla connessione. Pertanto, per stabilire, mantenere e chiudere una connessione, é necessario

+	Bit 0-15	16-31
0	Source Port (optional)	Destination Port
32	Length	Checksum (optional)
64+	Data	

Figura 34: Esempio di datagramma [1].

inviare pacchetti di servizio i quali aumentano l'overhead di comunicazione. Al contrario, UDP é senza connessione ed invia solo i datagrammi richiesti dal livello applicativo.

- UDP non offre nessuna garanzia sull'affidabilitá della comunicazione ovvero sull'effettivo arrivo dei segmenti, sul loro ordine in sequenza in arrivo; al contrario il TCP tramite i meccanismi di acknowledgement e di ritrasmissione riesce a garantire la consegna dei dati.
- l'oggetto della comunicazione del TCP é il flusso di byte mentre quello dell'UDP é il singolo datagramma.

L'utilizzo del protocollo TCP rispetto a UDP é, in generale, preferito quando é necessario avere garanzie sulla consegna dei dati o sull'ordine di arrivo dei vari pacchetti (come per esempio nel caso di trasferimenti di file). Al contrario UDP viene principalmente usato nel caso si abbiano forti vincoli sulla velocitá (per esempio comunicazioni Voip, oppure video-streaming) e l'economia di risorse della rete.

4

SPECIFICHE E SCELTE PROGETTUALI

4.1 INTRODUZIONE

Per impostare e visualizzare i parametri di simulazione si é deciso di sviluppare un software a interfaccia grafica di comunicazione dati per personal computer. Si é deciso di mettere in comunicazione la scheda Texas dell'emulatore (fig. 12) e l'applicazione di controllo per PC tramite collegamento ethernet. Questo per due motivi:

1. semplicitá di collegamento dato che la connessione ethernet é ormai presente in tutti gli ambienti informatizzati.

2. apre la possibilitá di gestire la simulazione anche da remoto.

Per garantire una comunicazione efficace fra l'emulatore e il PC dov'é installata l'applicazione a interfaccia si é optato per un collegamento cablato diretto dispositivo-dispositivo impiegando un cavo ethernet incrociato anziché un collegamento di rete canonico che prevede una connessione fra i due sistemi gestita da modem e/o router. Il cavo incrociato é un cavo ethernet standard dotato di connettori RJ-45 nella quale, in una di queste due estremitá, si invertono i collegamenti del canale trasmettitore Tx con il canale ricevitore Rx. Questa tecnica permette di ottenere la comunicazione diretta del canale Tx di un dispositivo con il canale Rx del secondo dispositivo senza la necessitá di passare attraverso alcun router che gestisca l'indirizzamento dei pacchetti dati trasmessi. É stata scelta questa soluzione ai fini di aumentare l'affidabilitá e diminuire la complessitá della comunicazione, dato che non essendo presenti modem, router e switch il collegamento risulta piú semplice, diretto e allo stesso tempo non si verificano perdite di dati fra una scheda e l'altra. Per stabilire la connessione diretta mediante cavo incrociato é necessario impostare la modalitá di indirizzamento tramite indirizzo IP statico.

L'emulatore é a tutti gli effetti un sistema di elaborazione real-time. Esso é stato programmato per trasmettere i dati di simulazione al PC dell'applicazione una volta ogni 500 ms, e nella trasmissione opposta l'applicazione é stata programmata per inviare i dati verso l'emulatore una volta ogni 300 ms. Diventa necessario decidere se effettuare la trasmissione dati tramite il protocollo di trasporto TCP oppure UDP. La frequenza di trasmissione inferiore al mezzo secondo, richiesta per soddisfare le caratteristiche real-time dell'emulatore, e la necessitá di ottenere una comunicazione semplice e senza intoppi ha spostato la scelta all'impiego del protocollo UDP. La porta d'ascolto impiegata

per effettuare la trasmissione tramite protocollo UDP é la numero 23, tipica delle applicazioni Telnet.

Nella trasmissione dall'emulatore verso il PC dell'applicazione si vuole inviare un dato comprendente 10 variabili e per la trasmissione opposta inviare un dato di 12 variabili. Per entrambi i sensi di trasmissione tale dato contiene i bit che descrivono i parametri di simulazione. Questi parametri sono a tutti gli effetti variabili in virgola mobile (nel gergo informatico variabili di tipo *float*) grazie al microcontrollore TMS320F28377S della scheda Texas che dispone di un'unitá di calcolo floating-point. Nel linguaggio di programmazione C/C++ la variabile di tipo *float* viene descritta con lunghezza predefinita di 32 bit, perció il Payload nella trasmissione dall'emulatore verso il PC assume una lunghezza di 40 byte e nella trasmissione opposta prende lunghezza di 48 byte. Si ottiene quindi che il singolo datagramma utilizzato dall'UDP é composto da:

- 48 byte di lunghezza nella fase di trasmissione che parte dall'emulatore verso il PC del software ad interfaccia
- 56 byte di lunghezza per la trasmissione opposta dal PC verso l'emulatore

La scheda Texas é sprovvista della porta di comunicazione Ethernet integrata. Per l'invio degli output la scheda Texas dispone di interfacce di comunicazione seriali I2C, SPI e SCI. Si é deciso di impiegare la porta di comunicazione SCI per la grande facilitá di trovare dispositivi che permettono la conversione della comunicazione da SCI a Ethernet. In commercio si trovano sistemi dedicati alla conversione seriale/Ethernet nelle versioni piú comuni UART/Ethernet oppure RS-232/Ethernet. Grazie a questi dispositivi é possibile dotare di porta Ethernet tutti quei sistemi che nascono con solo comunicazioni seriali tradizionali, SCI o RS-232. In buona sostanza questi dispositivi fungono da vere e proprie "schede di rete" come nei piú comuni personal computer. Per la scheda Texas la scelta é caduta sul convertitore SB70LC-100IR prodotto da NetBurner, mostrato in fig. 35.

Figura 35: Scheda NetBurner SB70LC-100IR.

4.2 SPECIFICHE SCHEDA NETBURNER

Come detto in precedenza il dispositivo NetBurner (fig. 35) permette la conversione seriale/Ethernet grazie al microcontrollore integrato *Freescale ColdFire 5270* con architettura a 32 bit operante alla frequenza di 147,5 MHz avente a disposizione 8 MB di memoria SDRAM e 512 kB di memoria *FLASH*. Per la comunicazione seriale questa scheda offre due porte seriali *UART*, in grado di gestire anche la connessione RS-232. Riguardo l'interfaccia di rete il convertitore dispone di una porta Ethernet con connettore RJ-45 e frequenze di trasmissione di 10/100 Mbps (standard *Fast Ethernet*). Supporta i protocolli di comunicazione Telnet, UDP, TCP con cifratura SSL e SSH, HTTPS. Offre modalitá di indirizzamento in rete con IP statico oppure tramite IP dinamico DHCP.

A disposizione dell'utente é implementata, all'interno della memoria del microcontrollore, la pagina web di configurazione dei parametri di trasmissione intitolata *NetBurner Page Configuration* (fig. 36) e accessibile dall'interfaccia di rete mediante browser internet per PC.

Il prezzo d'acquisto si aggira attorno ai 53*Euro*, e per le funzioni messe a disposizione il costo é uno dei piú bassi rispetto a dispositivi analoghi prodotti da altre marche. Si fa notare che questo componente che copre una funzione accessoria del sistema principale Texas é piú costoso della scheda principale stessa.

Il dispositivo é subito pronto all'uso e non ha bisogno di essere programmato. É sufficiente settare i parametri di trasmissione desiderati, grazie alla pagina *NetBurner Page Configuration*, per ottenere la conversione seriale/Ethernet. Per instaurare la corretta comunicazione

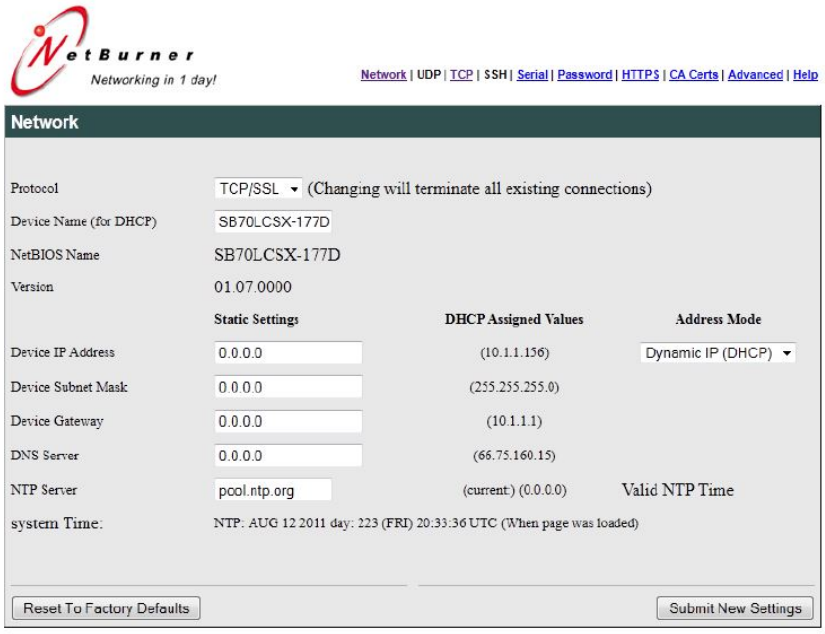


Figura 36: Istantanea di *NetBurner Page Configuration*.

seriale fra la scheda Texas e il dispositivo NetBurner é stata collegata la porta *SCI-B* della scheda Texas direttamente con la prima porta delle due disponibili nella scheda NetBurner, ovvero la *Port-0*, fissando a 115200 bps il valore della frequenza di trasmissione. Si é impostata la lunghezza del pacchetto dati elementare a 8 bit con 1 bit di STOP finale e nessun bit di controllo di paritá. Si é lasciato disabilitato anche il controllo di flusso *Xon/Xoff*. Anche se sono stati disabilitati tali controlli si é comunque implementata nel software a interfaccia una regola di controllo errori di trasmissione. Questa soluzione ha permesso di ottenere lo stesso un'alta efficienza della trasmissione dati pur avendo disabilitato le principali tecnologie di controllo.

Per il settaggio della modalitá di indirizzamento in rete NetBurner, oltre alla pagina di configurazione *NetBurner Page Configuration*, mette a disposizione per i propri prodotti l'applicazione desktop denominata *IPSetup*, illustrata in fig. 37. É sufficiente avviare il programma, trovare tramite la funzione *Search a Unit* il dispositivo collegato alla rete, o come nel caso in esame collegato direttamente alla porta Ethernet del PC e successivamente modificare i parametri d'interesse quali indirizzo IP, subnet mask e gateway. Utilizzando questo software é stato possibile assegnare l'indirizzo IP statico *192.168.1.3*, il valore della maschera di sottorete *subnet mask* pari a *255.255.255.0* e il valore del gateway predefinito di *192.168.1.1*. l'impostazione del gateway deve coincidere anche per la scheda di rete del PC. Quest'ultimo é basato sul sistema operativo *Microsoft Windows 7* e per la modifica del settaggio del collegamento Ethernet é sufficiente andare nelle impostazioni della scheda di rete presenti nella pagina *Centro Connessioni di Rete* accessibile dal *Pannello di Controllo*. Anche questa pagina permette la scelta della modalitá di indirizzamento statico oppure dinamico ed é stato fissato l'IP statico *192.168.1.10* con valore della maschera di sottorete sempre di *255.255.255.0* e lo stesso valore del gateway impostato nella scheda NetBurner di *192.168.1.1*.

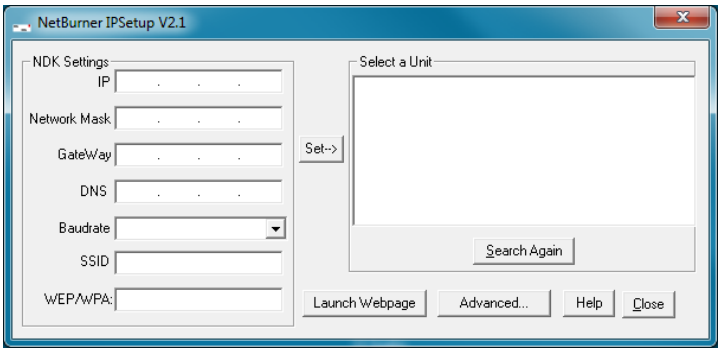


Figura 37: Istantanea di *IPSetup*.

Gazie a questeconfigurazioni il dispositivo NetBurner SB70LC-100IR ha reso possibile la comunicazione stabile sia per la parte seriale che per il lato Ethernet fra l'emulatore e il software PC a interfaccia.

4.3 SPECIFICHE DELL'AMBIENTE DI SVILUPPO

Il sistema operativo scelto per lo sviluppo del software a interfaccia é *Microsoft Windows* data l'altissima diffusione nel mondo professionale. Si é scelto di creare un'applicazione desktop classica *stand-alone* a interfaccia grafica dotata di una propria finestra di lavoro e completamente funzionante senza l'appoggio di qualsiasi altro software. Per evitare problemi di "trasportabilitá" del software si é optato per realizzare un'applicazione denominata, nel gergo informatico, di tipo portabile (dall'inglese *portable application*). L'applicazione portabile non necessita di alcuna installazione nel sistema operativo per poter essere eseguita. Essa é strutturata su di un unico file principale, detto file eseguibile, che permette l'esecuzione del software come se fosse stato installato. Da qui deriva il termine *portabile* dato dal fatto che l'applicazione puó indistintamente essere trasferita ed eseguita su qualsiasi altro computer in cui si dispone di un sistema operativo compatibile con l'applicazione stessa.

Per la programmazione del software é stato scelto l'ambiente di sviluppo integrato (dall'inglese *Integrated Development Environment*) *Qt Creator* (fig. 38) prodotto dalla societá Qt Software. Esso é un abiente di sviluppo multipiattaforma, sviluppato anche per altri sistemi operativi oltre a Windows (quali ad esempio Mac OS X e diverse distribuzioni Linux a partire da Ubuntu, una fra le piú note). Le principali caratteristiche che hanno indotto nella sua scelta sono:

- dispone di ampie librerie dedicate alla programmazione di interfacce grafiche
- usa il linguaggio C++ standard per la programmazione
- é dotato di licenza GNU GPL per software libero senza scopi commerciali.

La versione dell'ambiente *Qt* installata per la programmazione é la *5.5.1* basata sul compilatore *MinGW*. MinGW é la versione del famoso compilatore *GCC* per sistemi Linux convertita per l'ambiente Windows.

4.4 SOCKET

In informatica con il termine *socket* si indica uno strumento software che permette la trasmissione e la ricezione di dati attraverso una rete. Dal punto di vista di un programmatore un socket é un particolare oggetto software sul quale leggere e scrivere i dati da trasmettere o ricevere [5].

Esistono due tipi fondamentali di socket:

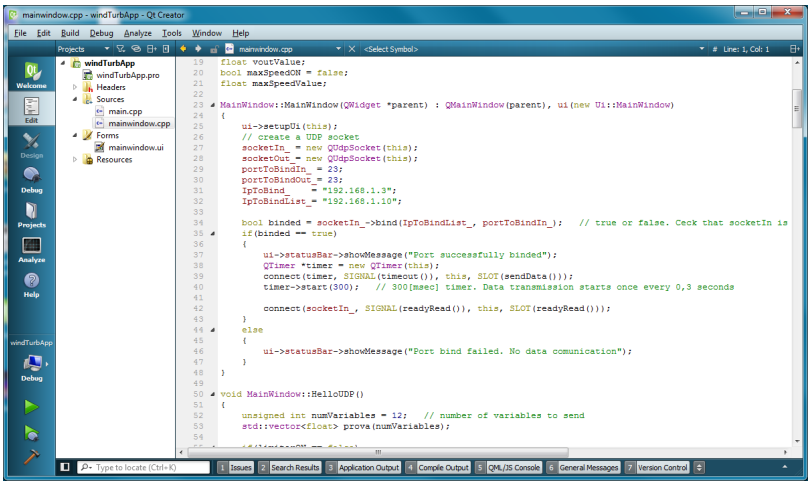


Figura 38: Istantanea dell'ambiente di sviluppo *Qt Creator*.

1. **socket tradizionali** su protocollo IP, usati in molti sistemi operativi per le comunicazioni attraverso un protocollo di trasporto di rete, quali TCP o UDP.

2. **socket locali**, usati nei sistemi operativi per le comunicazioni tra processi residenti sullo stesso computer.

A sua volta esistono altri due tipi di socket su protocollo IP:

1. **socket listen**, che rappresentano la possibilitá di ricevere nuove connessioni. Un socket di questo tipo é identificato dalla terna protocollo di trasporto, indirizzo IP del computer, numero di porta.

2. **socket established**, che rappresentano una particolare connessione attiva. Un socket di questo tipo é identificato dalla quintupla protocollo di trasporto, indirizzo IP sorgente, indirizzo IP destinazione, numero di porta sorgente, numero di porta destinazione.

In base alla modalitá di connessione si distinguono inoltre:

- **stream socket:** connection-oriented, basati su TCP.
- **datagram socket:** connectionless, basati su UDP.
- **raw socket:** utilizzati per lo sviluppo di protocolli.

5 SVILUPPO DEL SOFTWARE DI INTERFACCIA

5.1 INTRODUZIONE

Lo scopo del software é la visualizzazione e settaggio dei parametri di simulazione necessari al funzionamento dell'emulatore eolico. Sono presenti quindi due sottofunzioni principali:

- la ricezione e visualizzazione mediante display grafici dei dati provenienti dall'emulatore
- l'impostazione dei parametri tramite caselle di digitazione con il conseguente invio degli stessi all'emulatore

Il progetto é stato intitolato *windTurbApp*; un nome inglese mnemonico che ricorda che il software é a servizio di un emulatore per generatori eolici. Il titolo attribuito all'applicazione desktop in esecuzione é invece *Turbine Control*.

5.1.1 *Struttura del codice di programmazione*

Il codice di programmazione C++ é strutturato su sei pagine di segiuto elencate:

1. **windTurbApp.pro** (*.project*): é la pagina che racchiude le specifiche principali del progetto quali ad esempio la lista di tutte le pagine di programmazione, la presenza di eventuali interfacce grafiche e la presenza di impiego di risorse di rete.
2. **mainwindow.h** (*.header*): é la pagina di intestazione dove vengono specificate le variabili statiche, tutti i metodi pubblici o privati, e parte delle librerie incluse nel codice.
3. **main.cpp** (*.c++*): é la pagina principale del codice sorgente dove vengono stese parte delle istruzioni che permettono l'esecuzione del programma.
4. **mainwindow.cpp** (*.c++*): altra pagina dove viene steso codice sorgente necessario per il funzionamento del software.
5. **mainwindow.ui** (*.user-interface*): é la pagina dove viene descritto l'aspetto finale dell'interfaccia grafica, detta *GUI* (Graphical User Interface).
6. **resource.qrc** (*.Qt-resource-collection-file*): é la pagina dove vengono indicizzate tutte le varie risorse necessarie quali ad esempio immagini, icone grafiche, effetti sonori.

In fig. 39 é illustrata la struttura del codice all'interno dell'ambiente di sviluppo textitQt Creator.

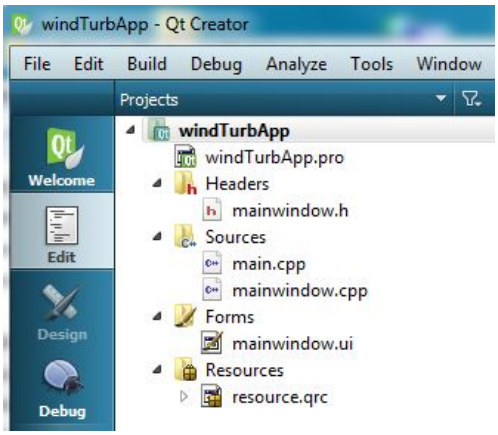


Figura 39: Struttura del codice nell'ambiente Qt.

5.1.2 *Parametri di simulazione*

I parametri di simulazione necessari si dividono in due categorie:

1. parametri che dal software di interfaccia vengono inviati all'emulatore
2. parametri che dall'emulatore vengono spediti al PC del software di interfaccia.

Parametri inviati all'emulatore

I parametri inviati all'emulatore rappresentano le variabili impostate per il corretto svolgimento della simulazione. Esse sono dichiarate all'interno della pagina di programmazione *mainwindow.cpp* da opportune variabili globali di tipo float. I parametri di simulazione con le relative variabili associate sono di seguito elencate:

- **Wind Speed:** rappresenta la velocitá del vento espressa in m/s. Essa é descritta dalla variabile *windSpeed*.
- **Max Speed:** specifica il valore massimo ammissibile della velocitá di rotazione della turbina simulata espresso in rpm. Essa é descritta dalla variabile *maxSpeed*.
- **Turbine Radius:** rappresenta la dimensione del raggio della turbina espressa in m. É contenuta nella variabile *turbRad*.
- **Rotor Inertia:** specifica il valore dell'inerzia del rotore della turbina espresso in $kg * m^2$. Essa viene memorizzata nella variabile *rotIn*.

- **Gear Ratio:** rappresenta il rapporto di riduzione del cambio della turbina. É descritta dalla variabile *gearRatio*.
- **Poles:** specifica il numero dei poli impiegato nel generatore sincrono a magneti permanenti. É contenuta nella variabile *poles*.
- **Resistance:** rappresenta il valore della resistenza elettrica del generatore espresso in ohm. É descritta dalla variabile *resistance*.
- **Magnetic Flux:** specifica il flusso magnetico del generatore elettrico espresso in Wb. É descritta dalla variabile *magnFlux*.
- **Generator Inertia:** rappresenta il valore dell'inerzia del generatore espresso in $\mathrm{kg} * \mathrm{m}^2$. Essa viene memorizzata nella variabile *genIn*.
- $\mathbf{V_{out}}$ **Limiter:** specifica il valore limite della tensione d'uscita del generatore simulato espresso in valore efficace RMS. Essa é descritta dalla variabile *voutLim*.

Parametri ricevuti dall'emulatore

I parametri ricevuti dall'emulatore rappresentano le variabili di simulazione visualizzate dal software di interfaccia. Esse sono di seguito elencate:

- **Wind Power:** rappresenta la potenza espressa in kW prodotta dal vento simulato che soffia sulle pale della turbina.
- **Turbine Speed:** specifica la velocitá della turbina espressa in rpm.
- **Turbine Torque:** descrive la coppia della turbina espressa in Nm.
- **Pitch Angle:** rappresenta il valore dell'angolo di pitch, espresso in gradi, assunto dalle pale della turbina.
- **Active Power:** specifica la potenza elettrica attiva espressa in kW prodotta dal generatore elettrico simulato.
- **Reactive Power:** descrive la potenza reattiva espressa in kvar prodotta dal generatore.
- **Power Factor:** specifica il fattore di potenza del generatore.
- $\mathbf{V_{out}}$**:** rappresenta il valore della tensione d'uscita del generatore espresso in valore efficace RMS.
- $\mathbf{I_{out}}$**:** specifica il valore della corrente d'uscita del generatore espresso in valore efficace RMS.

- **V_{out} Frequency:** descrive il valore della frequenza della tensione d'uscita dell'amplificatore espresso in Hz.

Il software per poter scambiare i dati di simulazione con l'emulatore necessita di interfacciarsi con la porta di comunicazione Ethernet presente nel personal computer in cui esso é in esecuzione. Per realizzare ció éstato sviluppato nel codice di programmazione un socket che ha il compito di collegare il software con la porta Ethernet.

5.2 DESCRIZIONE DEL CODICE DI PROGRAMMAZIONE

5.2.1 *mainwindow.h*

Per agevolare la programmazione del software l'ambiente di sviluppo Qt mette a disposizione dell'utente il file di intestazione *mainwindow.h*.

Sotto a seguire é riportato il codice relativo alle funzioni *#include* che collega nel file *mainwindow.h* le librerie *QMainWindow*, *QObject*, *QUDPSocket* e *vector*.

```
1 #include <QMainWindow> // offre gli strumenti per la creazione
      della finestra utente grafica del software
2 #include <QObject> // descrive le caratteristiche degli oggetti
3 #include <QUdpSocket> // offrei gli strumenti per la
      programmazione di socket di tipo datagram
4 #include <vector> // libreria necessaria per importare le
      proprieta' dei vettori
```

Nel file *mainwindow.h* sono specificate le varie funzioni di tipo pubblico e privato sviluppate nei file *.cpp*.

```
1  ...
2  public slots:
3          void readyRead();
4          void sendData();
5
6  private slots:
7          void on_StopButton_clicked();
8          void on_WindSpeedSpinBox_editingFinished();
9          void on_TurbRadSpinBox_editingFinished();
10         void on_MaxSpeedSpinBox_editingFinished();
11         void on_PolesSpinBox_editingFinished();
12         void on_ResistanceSpinBox_editingFinished();
13         void on_MagFluxSpinBox_editingFinished();
14         void on_RotInSpinBox_editingFinished();
15         void on_GearRatioSpinBox_editingFinished();
16         void on_LimiterSpinBox_editingFinished();
17         void on_GenInSpinBox_editingFinished();
18         void on_actionExit_triggered();
```

```
19          void on_actionAbout_Turbine_Control_triggered();
20          void on_StartButton_clicked();
21          void on_actionRun_triggered();
22          void on_actionStop_triggered();
23          void on_VoutLimCheckBox_clicked();
24          void on_actionVout_Limiter_triggered();
25          void on_MaxSpeedCheckBox_clicked();
26          void on_actionMax_Speed_triggered();
27          ...
```

La parte finale del file specifica le variabili globali private.

```
1 ...
2 private:
3         QHostAddress IpToBind_;  // memorizza l'indirizzo IP di
              destinazione
4         QHostAddress IpToBindList_; // contiene l'indirizzo IP
              della scheda di rete del PC
5         qint64 portToBindIn_; // variabile "qint64" (integer
              descritto da 64 bit). Memorizza il numero della porta
               d'ingresso
6         qint64 portToBindOut_; // memorizza il valore della porta
               d'uscita
7         std::vector<float> receivedData_; // vettore di tipo
              float necessario per la ricezione dati
8 ...
```

5.2.2 *main.cpp*

Questa pagina é la prima di codice sorgente C++. É caratterizzata da una breve stesura di istruzioni. Esse sono in assoluto le prime istruzioni eseguite durante la fase di avvio del programma. Il compito principale é quello di generare l'oggetto *w* della classe *MainWindow* che altro non é che la finestra grafica dell'interfaccia utente dell'applicazione. Il codice del file *main.cpp* é il seguente:

```
1 #include "mainwindow.h"  // inclusione del file di intestazione
2 #include <QCoreApplication> // offre gli strumenti per le
      funzioni software che non interagiscono con l'interfaccia
      grafica
3 #include <QApplication> // la classe QApplication gestisce il
      flusso di controllo dell'interfaccia grafica e le
      impostazioni principali.
4
5 int main(int argc, char *argv[])
6 {
7         QApplication a(argc, argv); // crea una nuova istanza di
              tipo QApplication e invoca il costruttore di tale
              classe
```

```
8
9           MainWindow w; // creazione dell'oggetto w di classe
                MainWindow
10          w.show(); // istruzione che inizializza la finestra
                utente a livello grafico
11
12          return a.exec();
13  }
```

5.2.3 *mainwindow.cpp*

Questo file di codice sorgente descrive tutte le funzioni e gli oggetti che completano il software inizializzato nel file *main.cpp*. Nella prima parte del file sono presenti le funzioni *#include* che includono il file *mainwindow.h,* la libreria *QTimer* e la libreria *QMessageBox*. Il codice é il seguente:

```
1  #include "mainwindow.h"  // include il file mainwindow.h
2  #include <QTimer>  // mette a disposizione funzioni per l'
       utilizzo dei timer
3  #include <QMessageBox>  // rende disponibili le funzioni per l'
       impiego delle finestre di dialogo per informare l'utente o
       per chiedere all'utente una domanda e ricevere una risposta.
4  ...
```

Successivamente vengono dichiarate le variabili globali. Fra di esse sono presenti anche le variabili che rappresentano i parametri di simulazione da spedire all'emulatore.

```
1  ...
2  float windSpeed = 0;
3  float turbRad   = 0;
4  float maxSpeed = 10000;
5  float poles = 0;
6  float resistance = 0;
7  float magnFlux   = 0;
8  float rotIn = 0;
9  float gearRatio  = 0;
10 float voutLim  = 10000;
11 float genIn = 0;
12 float isON  = 0;  // variabile di controlle che verifica se la
       simulazione e' in esecuzione
13 float voutValue;
14 float maxSpeedValue;
15 bool limiterON = false;   // variabile impiegata per l'
       attivazione/disattivazione del controllo della tensione Vout
16 bool maxSpeedON = false;  // variabile impiegata per l'
       attivazione/disattivazione del controllo dell'angolo di pitch
17 ...
```

La prima funzione della pagina é il metodo *MainWindow* appartenente all'omonima classe. Tale metodo descrive i passi svolti dal programma subito dopo aver creato la finestra dell'interfaccia utente. Il primo gruppo di istruzioni setta i parametri necessari per effettuare la comunicazione con l'emulatore mediante protocollo UDP. Vengono creati due socket; il primo per gestire la ricezione dati e il secondo per gestire la trasmissione. Questi socket sono del tipo *datagram established*.

```
1 ...
2 socketIn_ = new QUdpSocket(this);  // creazione del socket
      dedicato per latrasmissione dati
3 socketOut_= new QUdpSocket(this);  // creazione del socket
4 portToBindIn_ = 23;  // impostazzione della porta di ricezione
      della scheda di rete del PC
5 portToBindOut_= 23;  // impostazione della porta di destinazione
      (dell'emulatore)
6 IpToBind_    = "192.168.1.3";  // impostazione dell'indirizzo IP
      di destinazione
7 IpToBindList_= "192.168.1.10"; // impostazione dell'indirizzo IP
      della scheda di rete del PC
8 ...
```

Successivamente tramite l'istruzione *binded* si vuole verificare se il socket di lettura *socketIn_* riesce ad instaurare il collegamento con la scheda di rete del PC. L'istruzione di tipo *bool* restituisce un valore *true* o *false*. Questo valore restituito viene analizzato da una selezione *if*. Se il valore restituito dalla funzione *binded* vale true allora si instaura il collegamento tra la *socketIn_* e l'interfaccia di rete del PC. L'utente viene avvertito dall'avviso testuale "Port successfully binded" che compare nella barra di stato della finestra di lavoro del software. A questo punto viene creato un timer e poi settato a 300 ms. Tramite la funzione *connect* il segnale di *timeout()* del timer (indica che il tempo settato é scaduto) viene legato alla funzione *sendData()*. Questa funzione ha lo scopo di temporizzare l'invio dei dati ogni qualvolta sia scaduto il timer (in questo caso ogni 300 ms). La funzione *connect* viene impiegata anche per collegare il segnale *readyRead()* della *socketIn_* (dati ricevuti e pronti per la lettura) alla funzione *readyRead()* del file *mainwindow.cpp* che ha il compito di raccogliere i dati letti da *socketIn_*. Di seguito il relativo codice:

```
1 ...
2  bool binded = socketIn_->bind(IpToBindList_, portToBindIn_);
       //  verifica se e' possibile legare socketIn_ alla porta
       Ethernet del PC. Essa restituisce true o false
3  if(binded == true)  // socketIn_ riesce a stabilire il
       collegamento con la porta Ethernet
4  {
```

```
5         ui->statusBar->showMessage("Port successfully binded");
              // istruzione che crea nella barra di stato della
              finestra utente l'avviso testuale "Port successfully
              binded"
6         QTimer *timer = new QTimer(this);  // creazione del timer
7         connect(timer, SIGNAL(timeout()), this, SLOT(sendData()))
              ;  // collega il segnale timeout() alla funzione
              sendData()
8         timer->start(300);   // settaggio del timer a 300 ms
9
10        connect(socketIn_, SIGNAL(readyRead()), this, SLOT(
              readyRead()));  // collega il segnale readyRead() con
               la funzione readyRead()
11  }
12  else \\ socketIn_ non riesce a stabilire il collegamento con la
        porta Ethernet
13  {
14        ui->statusBar->showMessage("Port bind failed. No data
              comunication"); // avviso testuale di collegamento
              alla porta di rete fallito
15  }
16  }
```

Se il valore restituito dalla funzione *binded* vale false vuol dire che *socketIn_* non é riuscito a stabilire il collegamento con la porta di rete del PC e viene lanciato l'avviso testuale "Port bind failed, no data comunication".

La funzione *sendData()* verifica tramite una selezione *if* se l'emulatore sta eseguendo la simulazione. Questa verifica viene fatta analizzando il valore della variabile globale di tipo float *isON*. Se *isON* vale 1 significa che la simulazione e' in corso. A questo punto viene visualizzato sulla barra di stato l'avviso testuale "Sending data..." e il programma passa alla funzione *HelloUDP*.

Per tutti gli altri valori di *isON* diversi da 1 significa che l'emulatore non sta eseguendo la simulazione. L'utente viene avvisato, nella barra di stato, dall'avviso testuale "Simulation stopped" e il programma passa alla funzione *HelloUDP*. Di seguito il codice relativo:

```
1  void MainWindow::sendData()
2  {
3         if(isON == 1)   // simulazione in corso
4         {
5                 ui->statusBar->showMessage("Sending data..."); //
                       avviso testuale di avviso di invio dati
6                 HelloUDP();
7         }
8         else   // simulazione ferma
9         {
10                ui->statusBar->showMessage("Simulation stopped!")
                      ; // avviso di simulazione non in corso
11                HelloUDP();
```

```
12         }
13 }
```

La funzione *HelloUDP()* ha lo scopo di raccogliere i parametri di simulazione necessari e trasmetterli all'emulatore. La prima fase crea il vettore *prova* di 12 elementi di tipo float e alloca in esso i parametri di simulazione. Come sopra citato i parametri di simulazione necessari all'emulatore sono le 11 variabili globali di tipo float *windSpeed*, *magnFlux*, *turbRad*, *poles*, *resistance*, *gearRatio*, *rotIn*, *genIn*, *maxSpeed*, *voutLim* e *isON*. Il codice relativo é il seguente:

```
1  void MainWindow::HelloUDP()
2  {
3          unsigned int numVariables = 12;   // numero di variabili
               da trasmettere
4          std::vector<float> prova(numVariables);  // creazione del
               vettore prova di dimensione 12 variabili
5
6          // allocazione dei parametri nel vettore prova
7          prova.at(0) = windSpeed;
8          prova.at(1) = magnFlux;
9          prova.at(2) = turbRad;
10         prova.at(3) = poles;
11         prova.at(4) = resistance;
12         prova.at(5) = gearRatio;
13         prova.at(6) = rotIn;
14         prova.at(7) = genIn;
15         prova.at(8) = maxSpeed;
16         prova.at(9) = voutLim;
17         prova.at(10)= isON;
18         ...
```

La trasmissione dei dati viene svolta da *socketOut_* grazie all'istruzione *writeDatagram*. Tale istruzione accetta per l'invio solo variabili di tipo *char*. É quindi necessario riscrivere il vettore *prova* appena creato in un nuovo vettore, nominato "c", contenente le stesse variabili convertite da tipo *float* a *char*. Si ricorda che nel linguaggio C++ lo spazio di memoria occupato da una variabile di tipo float é di 4 byte, quello occupato da una variabile di tipo char é di 1 byte. In conclusione per descrivere una variabile di tipo float sono necessari 4 variabili di tipo char (fig. 40).

La porzione di codice che tratta questi passi é la seguente:

```
1 ...
2 unsigned char c[sizeof(float)*numVariables];   // "sizeof(float)"
       restituisce il valore 4. Viene creato il vettore "c" di
      dimensione 48 byte
3 float tmp;
4 for(int i=0; i<numVariables-1; i++)   // indice che va da 0 a 11.
       Il ciclo for svolge 12 iterazioni
```

```
{
        tmp = prova.at(i);  // memorizza la variabile i-esima del
             vettore "prova"
        memcpy(c+i*sizeof(tmp), (char*)&tmp, sizeof(tmp));   //
            spezza la variabile i-esima memorizzata in "tmp" in 4
             variabili char e le alloca nel vettore "c" a partire
             dall'indice i-esimo
}
...
```

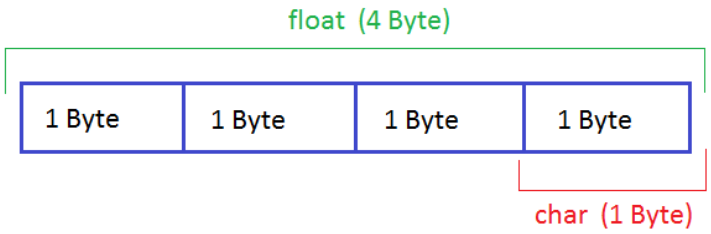


Figura 40: Confronto tra variabile *float* e *char*.

All'interno del ciclo *for* l'istruzione *memcpy()* (copy memory block) permette la trascrizione delle variabili float del vettore "prova" nel vettore "c" spezzando ciascuna variabile di tipo float in 4 variabili di tipo char. Una volta ultimato il ciclo for, il vettore "c" contiene 44 byte di caratteri char.

Gli ultimi 4 byte liberi del vettore "c" sono dedicati per l'allocazione della variabile *ckSm* necessaria ad implementare il controllo di somma per la verifica di eventuali errori di trasmissione. Tale controllo consiste nell'effettuare la somma algebrica dei 44 byte che descrivono gli 11 parametri di simulazione tramite ad un'iterazione di ciclo for. La somma poi viene memorizzata nella variabile *ckSm*, successivamente caricata nel vettore "c" per mano dell'istruzione *memcpy()* e inviata all'emulatore mediante il socket d'uscita *socketOut_*. I dati, una volta giunti a destinazione, vengono elaborati dal firmware dell'emulatore che ricalcola la somma algebrica degli 11 parametri di simulazione per poi confrontarne il valore con quello della variabile *ckSm* ricevuta. Se i due valori coincidono la trasmissione é avvenuta con successo, altrimenti il caso contrario indica che si é verificato qualche sorta di errore durante la fase di trasmissione. I dati appena ricevuti non possono essere considerati validi e vanno subito scartati.

A seguire la parte di codice interessata:

```
        ...
        qint32 ckSm = 0;   // ckSm salva la somma dei 44 byte
            allocati nel vettore "c"
        for(int i=0; i<(numVariables-1)*4; i++)  // indice che va
             da 0 a 43. Il ciclo for svolge 44 iterazioni
        {
```

```
5                 ckSm = ckSm+c[i];   // svolge la somma dei 44
                      byte,in sequenza  ciclo dopo ciclo, per il
                      controllo checksum
6         }
7         memcpy(c+(numVariables-1)*sizeof(ckSm), (char*)&ckSm,
              sizeof(ckSm));   // "sizeof(ckSm)" restituisce il
              numero 4. "memcpy" alloca "ckSm" negli ultimi 4 byte
              del vettore "c"
8
9         // trasmissione dei dati all'indirizzo IP di destinazione
               "IpToBind_"
10        socketOut_->writeDatagram(reinterpret_cast<const char*>(c
              ), sizeof(c), IpToBind_, portToBindOut_);
11 }
```

L'emulatore trasmette al software un pacchetto di 10 varibili che descrivono i seguenti parametri di simulazione: *Turbine Torque, Turbine Speed, Wind Power, Active Power, Reactive Power, Power Factor, Vout, Iout, Vout Frequency, Pitch Angle*.

La funzione dedicata alla ricezione dei dati é *readyRead()*. Le istruzioni della prima parte della funzione inizializzano i parametri necessari alla socket di ricezione *socketIn_* per la lettura dei dati. Per la corretta ricezione dei dati viene allocata una porzione di memoria di dimensione "messageSize" byte tramite l'istruzione *malloc()* (memory allocation) a cui fa riferimento il puntatore "data" di tipo char necessario alla socket di ricezione per la memorizzazione dei dati ricevuti dall'emulatore. Il codice relativo é il seguente:

```
1  void MainWindow::readyRead()
2  {
3      qint64 messageSize = socketIn_->pendingDatagramSize();   // "
           pendingDatagramSize()" restituisce la dimensione in byte
           del pacchetto dati ricevuto (in questo caso 40). Questo
           valore viene allocato nella variabile "messageSize" di
           tipo "qint64" ovvero descritta da 64bit
4      QHostAddress sender;  // contiene l'indirizzo IP del mittente
            del pacchetto dati
5      quint16 senderPort;  // contiene il num. di porta del
           mittente
6      receivedData_.resize(messageSize/sizeof(float));   //
           ridimensiona il vettore "receivedData_" a 10 variabili
           (40/4)
7      qint16 nInFloatsState_ = 10;   // numero delle variabili
           float da ricevere
8      char *data;  // puntatore di tipo char
9      data = (char *) malloc(messageSize);   // puntatore ad un
           area di memoria dedicata alla memorizzazione dei dati
           ricevuti
10     float tmp;
11  ...
```

Il pacchetto dati viene effettivamente ricevuto dal socket di ricezione tramite l'istruzione "*socketIn_->readDatagram(data, messageSize, &sender, &senderPort);*".

La ricezione dati é regolata da una selezione *if*. La ricezione dati avviene correttamente quando si verifica la condizione "messageSize == sizeof(float)*nInFloatsState" ovvero quando la variabile *messageSize* ha lo stesso valore dell'espressione *sizeof(float)*nInFloatsState*. "*sizeof(float)*nInFloatsState*" restituisce il valore 40 (4*10). "*messageSize*" contiene la dimensione in byte del pacchetto dati ricevuto. La ricezione avviene senza errori quando il pacchetto dati contiene 10 variabili di tipo float quindi una dimensione di 40 byte.

Nel caso di corretta ricezione dei dati il *socketIn_* scarica i dati ricevuti nella porzione di memoria precedentemente allocata dal puntatore "data". I dati appena scaricati sono di tipo char e si ha la necessitá di convertirli in variabili di tipo float. Per ottenere ció viene impiegata nuovamente l'istruzione "memcpy()" iterata tramite un ciclo for che trascrive le variabili di tipo char allocate nel vettore "data" nel nuovo vettore di tipo float "*receivedData_*". Giunto a termine il ciclo for, i parametri di simulazione del nuovo vettore vengono poi visualizzati in forma grafica nei rispettivi display dell'interfaccia utente tramite l'istruzione "*ui->"nome display"->display(receivedData_.at(o));*".

L'ultima istruzione del primo ramo della selezione *if* della funzione ha il compito di visualizzare nella barra di stato dell'interfaccia utente l'avviso testuale *"Receiving data..."* ad indicare la corretta ricezione dati. Il codice interessato é il seguente:

```
1  ...
2  if(messageSize == sizeof(float)*nInFloatsState_)   // se "
       messageSize" e' uguale a 40 allora la ricezione e' avvenuta
       con successo.
3  {
4          socketIn_->readDatagram(data, messageSize, &sender, &
               senderPort);   // ricezione dei dati dalla porta di
               rete del PC.
5
6          for(int i=0; i<messageSize/sizeof(float); i++)  // "
               sizeof(float)" restituisce il numero 4. indice che va
                da 0 a 9. Il ciclo for svolge 10 iterazioni
7          {
8                  memcpy(&tmp, (float*)(data+i*sizeof(float)),
                       sizeof(float));  // "memcpy" alloca nella
                       variabile "tmp" 4 caratteri char memorizzati
                       nel vettore "data" a partire dall'indice i-
                       esimo
9                  receivedData_.at(i) = tmp; // trasferisce il
                       valore appena memorizzato di "tmp" nella
                       cella di indice i-esimo del vettore "
                       receivedData_".
10         }
```

```
11          // trasferimento del valore delle variabili del vettore "
                receivedData_" nei relatici display grafici a 7
                segmenti
12          ui->TurbTorqLCD->display(receivedData_.at(0));  //
                display nominato "TurbTorqLCD" visualizza il
                parametro Turbine Torque.
13          ui->TurbSpeedLCD->display(receivedData_.at(1));
14          ui->WindPowLCD->display(receivedData_.at(2));
15          ui->ActPowerLCD->display(receivedData_.at(3));
16          ui->ReactPowerLCD->display(receivedData_.at(4));
17          if(receivedData_.at(5) < 0.00001) // se il numero
                ricevuto e' troppo piccolo visualizza nel display il
                valore 0.
18          {
19          int powData = 0;
20          ui->PowFactLCD->display(powData);
21          }
22          else
23          {
24          ui->PowFactLCD->display(receivedData_.at(5));
25          }
26          ui->VoutLCD->display(receivedData_.at(6));
27          ui->IoutLCD->display(receivedData_.at(7));
28          ui->VoutFreqLCD->display(receivedData_.at(8));
29          if(receivedData_.at(9) < 0.0001)  // // se il numero
                ricevuto e' troppo piccolo visualizza nel display il
                valore 0.
30          {
31          int pitchData = 0;
32          ui->PitchAnLCD->display(pitchData);
33  }
34  else ...
```

Nel caso in cui la condizione "messageSize == sizeof(float)*nInFloatsState" non sia verificata, ovvero che la variabile *messageSize* non contenga il valore 40, indica che la ricezione é stata interrotta. É necessario comunque scaricare i dati ricevuti fino al momento dell'improvvisa interruzione della trasmissione per poter liberare il socket di ricezione. Se il socket non viene liberato il software non é piú in grado di ricevere i dati qualora la trasmissione dati riprendesse. Anche in questo caso, per mezzo dell'istruzione *malloc()* viene allocata una porzione di memoria a cui fa riferimento il puntatore "*devNull*" di tipo char. Tale area di memoria deve avere la stessa dimensione del pacchetto dati ricevuto perció viene sfruttata la funzione "*socketIn_->pendingDatagramSize()*" che restituisce la dimensione esatta del pacchetto ricevuto. A questo punto é possibile liberare la socket di ricezione mediante l'istruzione: "*socketIn_->readDatagram(devNull, socketIn_->pendingDatagramSize(), &sender, &senderPort);*" che scarica nella porzione di memoria riferita da "devNull" quel poco di dati raccolti prima dell'interruzione accidentale della trasmissione. L'ultima

istruzione, che chiude la funzione *readyRead()*, ha il compito di visualizzare nella barra di stato dell'interfaccia utente l'avviso testuale *"Wrong data reception!"*. Il codice relativo é il seguente:

```
...
        else   // trasmissione interrotta
        {
                char *devNull;  // puntatore di tipo char
                devNull = (char *) malloc(socketIn_->
                    pendingDatagramSize()); // // puntatore ad un
                     area di memoria dedicata per scaricare il
                    socket dai dati ricevuti.
                socketIn_->readDatagram(devNull, socketIn_->
                    pendingDatagramSize(), &sender, &senderPort);
                      // ricezione dati dalla porta Ethernet del
                    PC.
                ui->statusBar->showMessage("Wrong data reception
                    !");  // visualizzazione sulla barra di stato
                     del messaggio "Wrong data reception!"
        }
}
```

Le istruzioni restanti del file *mainwindow.cpp* descrivono il funzionamento dei vari oggetti presenti nell'interfaccia utente quali ad esempio: campi di digitazione numerica, tasti, check box di selezione/deselezione, display grafici a 7 segmenti e menú opzioni a tendina.

Le funzioni che gestiscono l'inserimento dati dal campo di digitazione a livello grafico (chiamati *spinbox*) alla variabile relativa al parametro sono cosí strutturate:

```
void MainWindow::on_"nome-campo-di-digitazione"_editingFinished()
{
  "nome-variabile" = ui->"nome-campo-di-digitazione"->value();
}
```

A titolo di esempio si riporta la funzione che gestisce l'inserimento del parametro *Wind Speed*. Il campo di digitazione, nel caso in esempio, e' nominato *WindSpeedSpinBox*. Il relativo codice é il seguente:

```
void MainWindow::on_WindSpeedSpinBox_editingFinished()
{
  windSpeed = ui->WindSpeedSpinBox->value();   // assegna il
      valore digitato nella variabile windSpeed.
}
```

Per gestire l'avvio e l'arresto della simulazione vengono impiegati due tasti grafici nominati *RUN* E *STOP*. Il comportamento del tasto *RUN* é descritto dalla seguente funzione:

```
1 void MainWindow::on_StartButton_clicked()
2 {
3    isON = 1;   // quando il tasto viene "cliccato" fissa il
         valore della variabile "isON" pari a 1. Tale valore indica
          che l'emulatore \'e pronto per eseguire la simulazione.
4 }
```

Il comportamento del tasto *STOP* invece é descritto dalla seguente funzione:

```
1 void MainWindow::on_StopButton_clicked()
2 {
3    isON = 0;   // quando il tasto viene "cliccato" fissa il
         valore della variabile "isON" a 0. Tale valore indica che
         all'emulatore di fermare la simulazione.
4 }
```

I comandi di RUN e STOP sono replicati anche nel menú a tendina chiamato *Simulation*. Le funzioni che gestiscono questi due comandi sono le seguenti:

```
1 void MainWindow::on_actionRun_triggered()
2 {
3    isON = 1;   // se il comando RUN presente nel menu' viene
         cliccato esso fissa il valore della variabile "isON" a 1.
4 }
5
6 void MainWindow::on_actionStop_triggered()
7 {
8    isON = 0;   // se il comando STOP presente nel menu' viene
         cliccato esso fissa il valore della variabile "isON" a 0.
9 }
```

La simulazione prevede la possibilitá di poter attivare/disattivare il controllo dell'angolo di pitch. Nell'interfaccia utente l'inserimento/-disinserimento del pitch-controller é realizzato tramite una casella di spunta, detta anche *ceck-box*. La funzione che descrive l'attivazione del controllo dell'angolo di pitch é la seguente:

```
1 void MainWindow::on_MaxSpeedCheckBox_clicked()
2 {
3        if(ui->MaxSpeedCheckBox->isChecked())  // se il ceck-bok
             viene spuntato significa controllo inserito.
4        {
5                maxSpeedON = true; // fissa il valore della
                     variabile di tipo bool al valore "true".
6        }
7        else   // ceck-box non spuntato. Controllo non inserito.
8        {
```

```
9                 maxSpeedON = false;  // // fissa il valore della
                      variabile di tipo bool al valore "false".
10        }
11 }
```

All'interno della funzione *HelloUDP()* é presente una piccola porzione di codice che completa il funzionamento del pitch-controller. Il codice d'interesse é il seguente:

```
1  ...
2  if(maxSpeedON == false)
3  {
4      maxSpeed = 10000;  // controllo non inserito
5  }
6  else  // controllo inserito
7  {
8      maxSpeed = maxSpeedValue;  // il valore digitato nel campo "
           Max Speed" viene trasferito nella variabile maxSpeeed. In
            questo modo si trasmette all'emulatore un valore di
           velocita' limite.
9  }
10 ...
```

La simulazione prevede anche un controllo della tensione d'uscita del generatore. Anche questo controllo, nell'interfaccia utente, é realizzato da un ceck-box. La funzione che descrive l'inserimento/disinserimento del Vout Controller é la seguente:

```
1  void MainWindow::on_VoutLimCheckBox_clicked()
2  {
3          if(ui->VoutLimCheckBox->isChecked())  // ceck-box
               spuntato. Controllo inserito.
4          {
5                  limiterON = true;
6          }
7          else  // controllo non inserito.
8          {
9                  limiterON = false;
10         }
11 }
```

Anche in questo caso all'interno della funzione *HelloUDP()* é presente una piccola porzione di codice che completa il funzionamento del Vout Controller. Il codice é il seguente:

```
1  ...
2  if(limiterON == false)
3  {
4      voutLim = 10000;  // controllo non inserito
5  }
```

```
6  else   // limiterON == true
7  {
8      voutLim = voutValue;  // il valore digitato nel campo "Vout
           Limiter" viene trasferito nella variabile "voutLim". In
           questo modo si trasmette all'emulatore il valore della
           tensione limite.
9  }
10 ...
```

5.3 DESCRIZIONE DELL'INTERFACCIA GRAFICA

Per la realizzazione di interfacce grafiche QT Creator mette a disposizione dell'utente l'ambiente *Design* (fig. 41). In questo ambiente sono disponibili moltissimi oggetti grafici che permettono di creare interfacce anche molto complesse.

L'aspetto grafico del software é descritto nel file di programmazione *mainwindow.ui*, e a differenza delle altre pagine scritte in linguaggio C++, essa é autocompilata in linguaggio *XML* dall'ambiente *Design*.

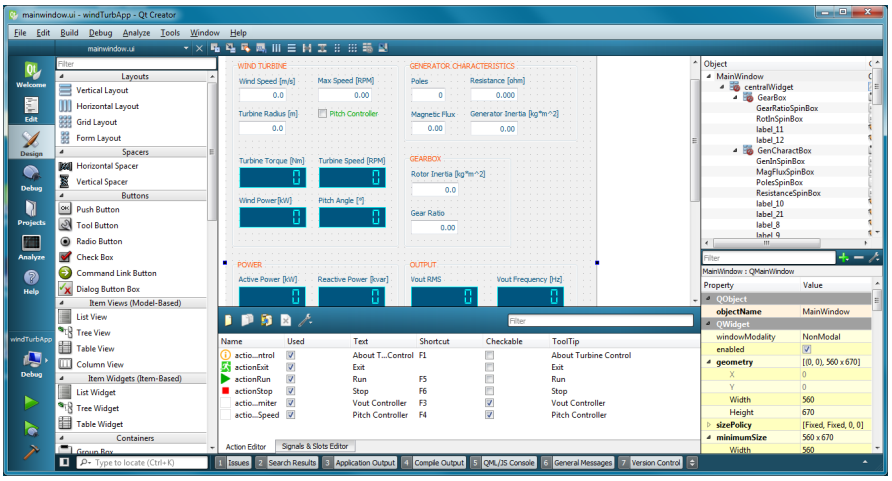


Figura 41: Istantanea dell'ambiente di sviluppo di interfacce grafiche *Qt Design*.

L'obbiettivo é stato quello di creare un'interfaccia utente sobria ma comunque completa di tutte le funzioni necessarie per il corretto svolgimento della simulazione. In fig. 42 é raffigurata la finestra utente del software.

Questa interfaccia é caratterizzata da una finestra principale strutturata secondo l'impostazione standard di un'applicazione desktop per l'ambiente Windows che richiede:

- Una finestra utente grafica dove visualizzare le informazioni principali (fig. 42).

Figura 42: Istantanea del software Turbine Control sviluppato.

- La presenza di menú opzioni a tendina presenti nella fascia superiore della finestra utente. In fig. 43 sono illustrati i menú: *File*, *Simulation* e *"?"*.

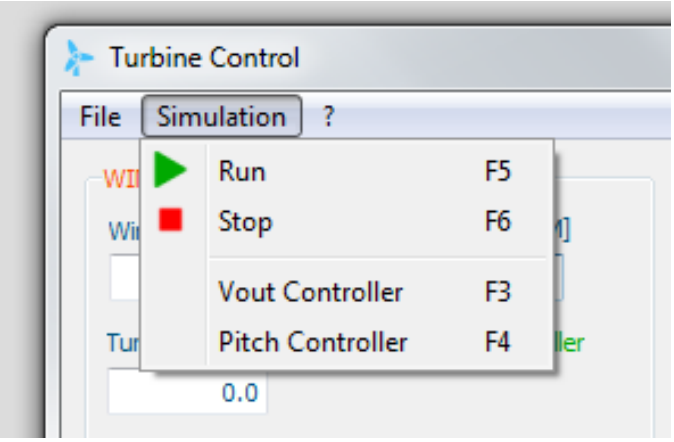


Figura 43: Menú a tendina.

- La presenza di un titolo del software con la relativa icona grafica presente nella cornice della finestra utente (fig. 44)

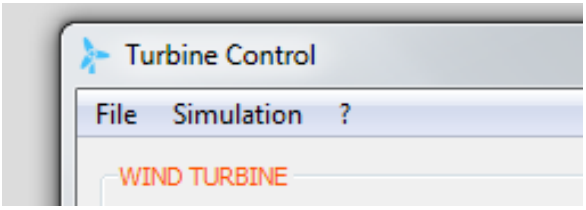


Figura 44: Icona e titolo software.

- La presenza di una barra di stato dell'applicazione presente nella fascia inferiore della finestra utente. Vedi fig. 45.

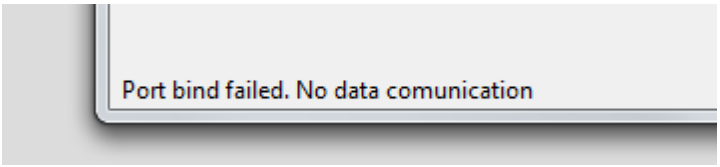


Figura 45: Barra di stato.

La finestra utente é stata pensata per raggruppare la stessa tipologia di dati tramite l'ausilio dell'elemento grafico *Group Box* disponibile fra gli oggetti messi a disposizione dall'ambiente *Design* di Qt. Ogni Group Box é stato fornito di un titolo testuale esplicativo della tipologia di parametri raggruppati. Essi sono:

- **Wind Turbine:** questo box, rappresentato in fig. 46 raggruppa i parametri relativi al vento e alla turbina. Tali parametri sono:

Wind Speed, Max Speed, Turbine Radius, Turbine Torque, Turbine Speed, Wind Power, Pitch Angle. Sono presenti sia i campi di digitazione numerica per l'inserimento dei dati che i display grafici a 7 segmenti che visualizzano i parametri ricevuti dall'emulatore. É anche presente il check box del controllo del pitch controller. Il comando di tale controllo é replicato anche nel menú opzioni *Simulation* (fig. 43). Ognuno degli elementi grafici all'interno del box é accompagnato dal titolo relativo al parametro di simulazione con indicata l'unitá di misura impiegata per l'elaborazione numerica.

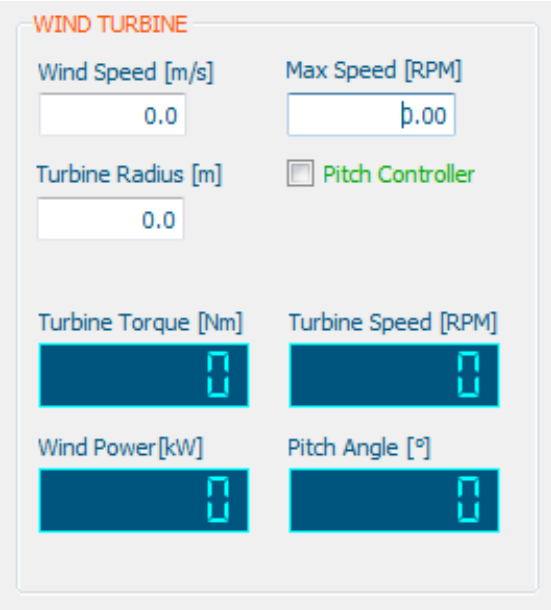


Figura 46: Group Box *Wind Turbine*.

- **Generator Characteristics:** tale box (fig. 47) rappresenta i parametri relativi alle caratteristiche del generatore elettrico. Sono presenti solo i campi di digitazione per l'inserimento delle variabili Poles, Resistance, Magnetic Flux, Generator Inertia. Ognuno di essi é accompagnato a livello grafico dal titolo e dall'unitá di misura del relativo parametro.

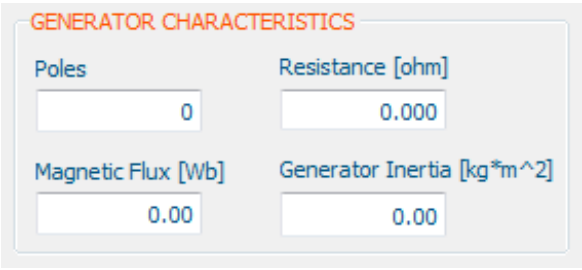


Figura 47: Group Box *Generator Characteristics*.

- **Gear Box:** é illustrato in fig. 48. Esso visualizza i due parametri relativi al riduttore meccanico della turbina. Sono presenti i due campi di digitazione per l'iserimento delle variabili Rotor Inertia e Gear Ratio.

Figura 48: Group Box *Gear Box*.

- **Power:** racchiude i parametri relativi alla potenza elettrica prodotta dal generatore elettrico della turbina. Sono presenti i tre display grafici a 7 segmenti che visualizzano i parametri: Active Power, Reactive Power e Power Factor. Il box é rappresentato in fig. 49.

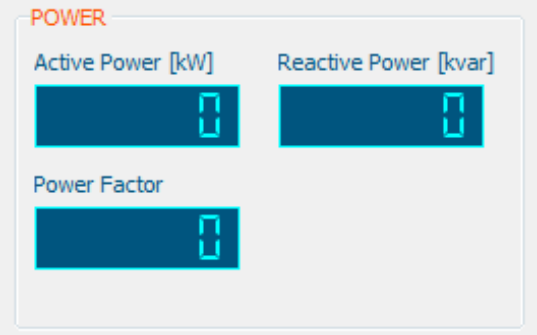


Figura 49: Group Box *Power*.

- **Output:** illustrato in fig. 50. Esso raggruppa i parametri relativi alle grandezze elettriche in uscita dal generatore della turbina. Sono presenti i tre display a 7 segmenti per la visualizzazione delle variabili Vout, Iout, Vout Frequency e il campo di digitazione per l'inserimento del parametro Vout Limiter. É anche presente il check box di selezione del Vout Controller ovvero del controllo della tensione d'uscita. Anche in questo caso l'azionamento del comando é replicato nel menú *Simulation* (fig. 43).

- **Simulation:** quest'ultimo box, raffigurato in fig. 51, racchiude i pulsanti dei due comandi principali RUN e STOP necessari per

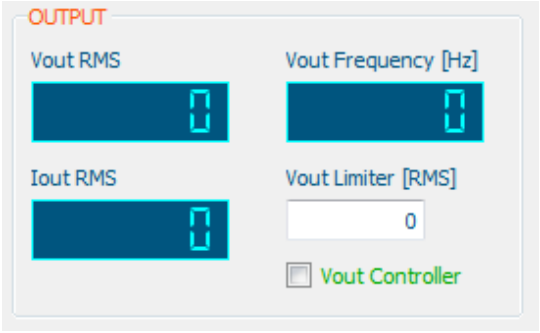


Figura 50: Group Box *Output*.

permettere l'avvio e l'arresto della simulazione. Anch'essi sono replicati nel menú a tendina *Simulation* rappresentato in fig. 43.

Figura 51: Group Box *Simulation*.

I comandi RUN, STOP, Vout Controller, Pitch Controller presenti nella finestra utente, oltre ad essere disponibili nel menú opzioni *Simulation* (fig. 43) possono anche essere attivati/disattivati tramite comandi rapidi da tastiera opportunamente programmati. A titolo di esempio, una volta avviato il software e inserito i parametri necessari all'esecuzione della simulazione, per avviare la simulazione é sufficiente premere il tasto *F5* della tastiera del PC.

Per ogni menú a tendina sono riportati, nel lato destro, i tasti corretti dei rispettivi comandi da abilitare o disabilitare (fig. 43).

Il codice di programmazione completo in tutte le sue parti é riportato in appendice.

6

RISULTATI SPERIMENTALI

L'implementazione finale dell'emulatore del generatore eolico per la verifica sperimentale al test *LVRT* é raffigurata in fig. 52.

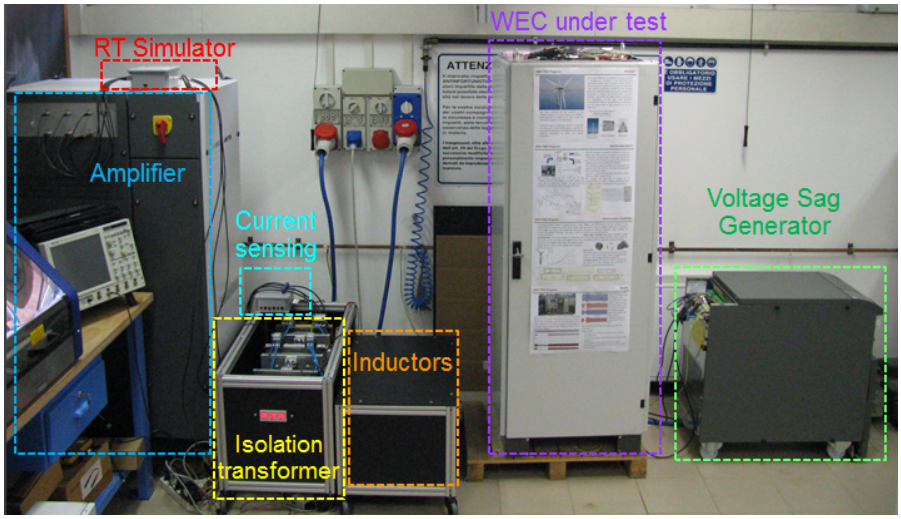


Figura 52: Realizzazione dell'emulatore sviluppato.

La simulazione si é svolta inserendo nel software *Turbine Control* i parametri riportati in tab. 1

Gli standard italiani relativi alla rete in bassa tensione specificano delle aree ben precise relative a buchi di tensione di rete secondo le quali gli aerogeneratori devono mantenere o meno la connessione alla rete, elencate in fig 53 ed evidenziata dalla parte tratteggiata. Il test seguente si riferisce alle condizioni di fig. 54 evidenziate dal cerchio di colore rosso. Queste condizioni si riferiscono ad un buco simmetrico avente durata di 390 ms e ampiezza del 45% di V_n. In queste condizioni l'inverter, per risultare conforme agli standard, non deve disconnettersi dalla rete.

Tabella 1: Riepilogo dei parametri utilizzati per l'emulazione

Raggio della turbina	5 [m]
Flusso di rotore	0.9 [*Wb*]
Numero di coppie polari	5
Rapporto di riduzione	3
Inerzia del rotore	90 [$kg \cdot m^2$]
Velocitá del vento	9 [m/s]

Case	Magnitude of voltage phase to phase (fraction of voltage immediately before the drop occurs)	Magnitude of positive sequence voltage (fraction of voltage immediately before the drop occurs)	Duration (s)	Shape
VD1 – symmetrical three-phase voltage drop	0,90 ± 0,05	0,90 ± 0,05	0,5 ± 0,02	
VD2 – symmetrical three-phase voltage drop	0,50 ± 0,05	0,50 ± 0,05	0,5 ± 0,02	
VD3 – symmetrical three-phase voltage drop	0,20 ± 0,05	0,20 ± 0,05	0,2 ± 0,02	
VD4 – two-phase voltage drop	0,90 ± 0,05	0,95 ± 0,05	0,5 ± 0,02	
VD5 – two-phase voltage drop	0,50 ± 0,05	0,75 ± 0,05	0,5 ± 0,02	
VD6 – two-phase voltage drop	0,20 ± 0,05	0,60 ± 0,05	0,2 ± 0,02	

Figura 53: Specifiche relative ai buchi di tensione dettate dalle normative.

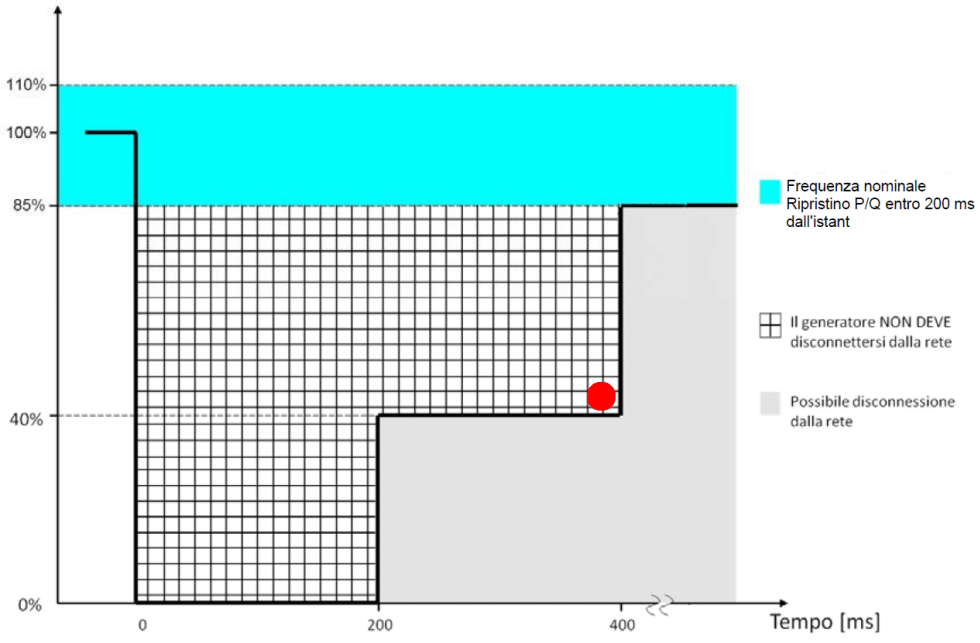


Figura 54: Area relativa alla disconnessione consentita del generatore durante un buco di tensione.

Tabella 2: Notazione delle forme d'onda illustrate

Tensione d'uscita del generatore [V]	V_{gen}
Corrente d'uscita del generatore [V]	I_{gen}
Potenza estratta dal generatore [kW]	P_{gen}
Tensione di rete [V]	V_{grid}
Corrente di rete [I]	I_{grid}

Il test in questione é raffigurato in fig. 55.

I risultati del test LVRT sono stati acquisiti tramite due oscilloscopi *Tektronix 7054* da 4 canali e 500 MHz di banda analogica.

Il controllore dell'angolo delle pale dell'emulatore della turbina eolica é disattivato. Si osserva in fig. 55 che il convertitore per turbine eoliche cessa l'assorbimento di corrente dal generatore a magneti permanenti simulato durante il buco di tensione. Non trovando una coppia elettrica che controbilancia quella meccanica del rotore, la velocitá di rotazione della turbina eolica aumenta, cosí come la tensione in uscita al generatore a magneti permanenti emulato. Nell'istante in cui la tensione di rete torna al suo valore nominale il convertitore ritorna ad assorbire corrente e dopo un breve transitorio l'emulatore di turbine eoliche torna al valore di equilibrio precedente al buco di tensione.

Il software Turbine Control ha sempre visualizzato i parametri in modo continuo e stabile per tutto il tempo della durata del test. Si é riscontrato che la frequenza di aggiornamento dei display dell'interfaccia grafica é abbastanza elevata da poter permettere una visualizzazione fluida dei dati di simulazione, assicurando un monitoraggio accurato da parte dell'operatore. Il delay tra l'aggiornamento dei dati inseriti nell'interfaccia grafica rispetto alla reazione sull'emulatore é impercettibile, questo permette di comandare l'emulatore in modo quasi istantaneo. Nel caso di qualche malfunzionamento il tasto *STOP* cessa l'emulazione e spegne il sistema immediatamente.

Il comportamento corretto della simulazione ha dato conferma della riuscita programmazione del software sviluppato.

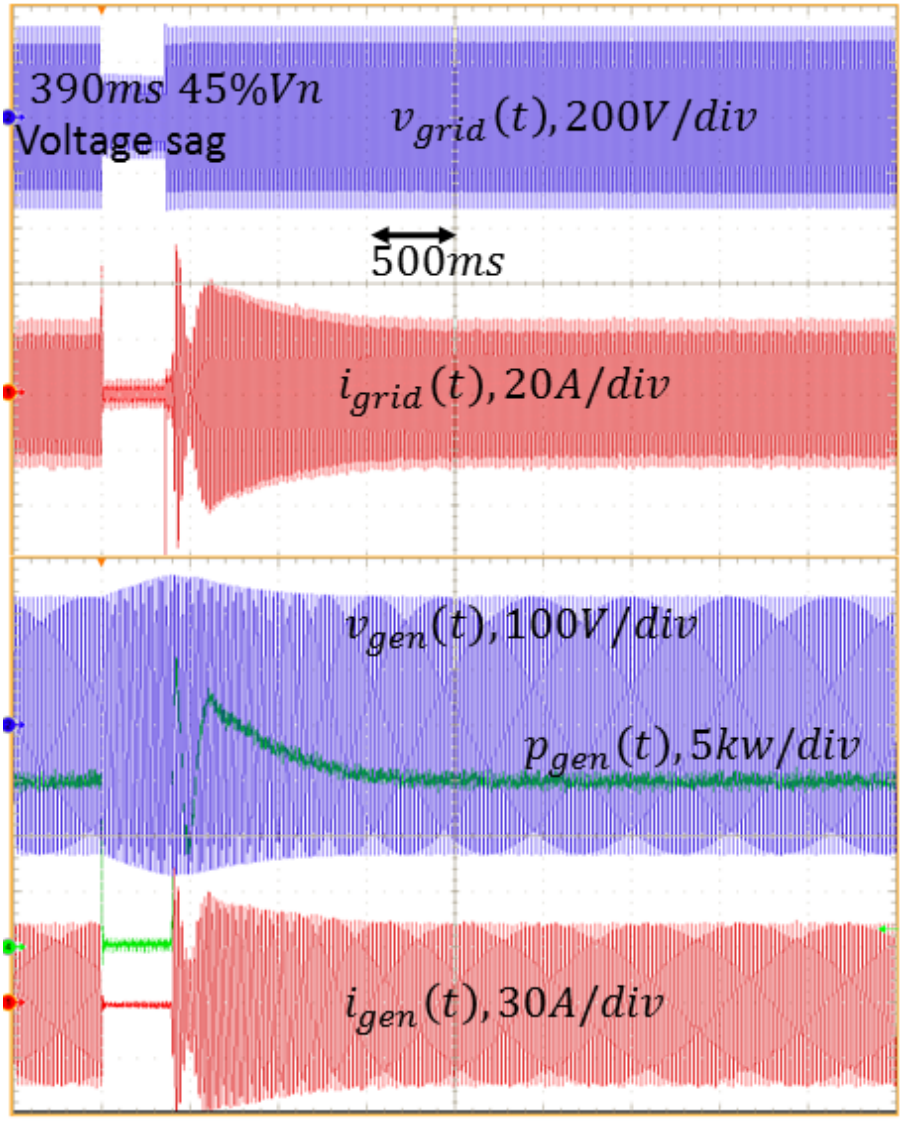


Figura 55: Forme d'onda acuisite dagli oscilloscopi.

CONCLUSIONI

Si ritiene che il software di interfaccia sviluppato abbia raggiunto gli obbiettivi prefissati.

L'interfaccia utente di *Turbine Control* é risultata semplice e intuitiva poiché si é riusciti ad organizzare in maniera sobria la grafica di visualizzazione delle informazioni.

La comunicazione via Ethernet ha pienamente soddisfatto i requisiti real-time dell'emulatore garantendo la trasmissione dei parametri di simulazione con la minima latenza.

APPENDIX

A

CODICE DI PROGRAMMAZIONE

windTurbApp.pro

```
#-------------------------------------------------------------
# Turbine Control
# Built on Febbruary 2016
# University of Padova
#-------------------------------------------------------------

QT       += core gui
QT       += core network

greaterThan(QT_MAJOR_VERSION, 4): QT += widgets

CONFIG += c++11

TARGET = windTurbApp
TEMPLATE = app


SOURCES += main.cpp\
mainwindow.cpp

HEADERS  += mainwindow.h

FORMS    += \
mainwindow.ui

RESOURCES += \
resource.qrc
```

mainwindow.h

```
#ifndef MAINWINDOW_H
#define MAINWINDOW_H

#include <QMainWindow>
#include <QObject>
#include <QUdpSocket>
#include <vector>

namespace Ui
{
        class MainWindow;
}

```

```
14 class MainWindow : public QMainWindow
15 {
16         Q_OBJECT
17 
18         public:
19         explicit MainWindow(QWidget *parent = 0);
20         ~MainWindow();
21 
22         public slots:
23         void readyRead();
24         void sendData();
25 
26         private slots:
27         void on_StopButton_clicked();
28         void on_WindSpeedSpinBox_editingFinished();
29         void on_TurbRadSpinBox_editingFinished();
30         void on_MaxSpeedSpinBox_editingFinished();
31         void on_PolesSpinBox_editingFinished();
32         void on_ResistanceSpinBox_editingFinished();
33         void on_MagFluxSpinBox_editingFinished();
34         void on_RotInSpinBox_editingFinished();
35         void on_GearRatioSpinBox_editingFinished();
36         void on_LimiterSpinBox_editingFinished();
37         void on_GenInSpinBox_editingFinished();
38         void on_actionExit_triggered();
39         void on_actionAbout_Turbine_Control_triggered();
40         void on_StartButton_clicked();
41         void on_actionRun_triggered();
42         void on_actionStop_triggered();
43         void on_VoutLimCheckBox_clicked();
44         void on_actionVout_Limiter_triggered();
45         void on_MaxSpeedCheckBox_clicked();
46         void on_actionMax_Speed_triggered();
47 
48         private:
49         Ui::MainWindow *ui;
50         void HelloUDP();
51         QUdpSocket *socketIn_;
52         QUdpSocket *socketOut_;
53         QHostAddress IpToBind_;
54         QHostAddress IpToBindList_;
55         qint64 portToBindIn_;
56         qint64 portToBindOut_;
57         std::vector<float> receivedData_;
58 };
59 
60 #endif // MAINWINDOW_H
```

main.cpp

```
1 #include "mainwindow.h"
2 #include <QCoreApplication>
```

```
#include <QApplication>

int main(int argc, char *argv[])
{
QApplication a(argc, argv);

MainWindow w;
w.show();

return a.exec();
}
```

mainwindow.cpp

```
#include "mainwindow.h"
#include "ui_mainwindow.h"
#include <QTimer>
#include <QTime>
#include <QMessageBox>

float windSpeed = 0;
float turbRad   = 0;
float maxSpeed = 10000;
float poles = 0;
float resistance = 0;
float magnFlux   = 0;
float rotIn = 0;
float gearRatio  = 0;
float voutLim  = 10000;
float genIn = 0;
float isON  = 0;   // while "isON" = 1 the simulation is runnning
bool limiterON = false;
float voutValue;
bool maxSpeedON = false;
float maxSpeedValue;

MainWindow::MainWindow(QWidget *parent) : QMainWindow(parent), ui
    (new Ui::MainWindow)
{
        ui->setupUi(this);
        // create a UDP socket
        socketIn_ = new QUdpSocket(this);
        socketOut_= new QUdpSocket(this);
        portToBindIn_ = 23;
        portToBindOut_= 23;
        IpToBind_    = "192.168.1.3";
        IpToBindList_= "192.168.1.10";

        bool binded = socketIn_->bind(IpToBindList_, portToBindIn
            _);   // true or false. Ceck that socketIn is in
            BoundState (the socket is bound to an address and
            port)
```

```
35          if(binded == true)
36          {
37                  ui->statusBar->showMessage("Port successfully
                        binded");
38                  QTimer *timer = new QTimer(this);
39                  connect(timer, SIGNAL(timeout()), this, SLOT(
                        sendData()));
40                  timer->start(300);   // 300[msec] timer. Data
                        transmission starts once every 0,3 seconds
41
42                  connect(socketIn_, SIGNAL(readyRead()), this,
                        SLOT(readyRead()));
43          }
44          else
45          {
46                  ui->statusBar->showMessage("Port bind failed. No
                        data comunication");
47          }
48  }
49
50  void MainWindow::HelloUDP()
51  {
52          unsigned int numVariables = 12;   // number of variables
                to send
53          std::vector<float> prova(numVariables);
54
55          if(limiterON == false)
56          {
57                  voutLim = 10000;
58          }
59          else   // limiterON == true
60          {
61                  voutLim = voutValue;
62          }
63          if(maxSpeedON == false)
64          {
65                  maxSpeed = 10000;
66          }
67          else
68          {
69                  maxSpeed = maxSpeedValue;
70          }
71          // setting vector "prova"
72          prova.at(0) = windSpeed;
73          prova.at(1) = magnFlux;
74          prova.at(2) = turbRad;
75          prova.at(3) = poles;
76          prova.at(4) = resistance;
77          prova.at(5) = gearRatio;
78          prova.at(6) = rotIn;
79          prova.at(7) = genIn;
80          prova.at(8) = maxSpeed;
```

```
81          prova.at(9) = voutLim;
82          prova.at(10)= isON;
83
84          unsigned char c[sizeof(float)*numVariables];   // "sizeof
                (float)" returns 4 bytes
85          float tmp;
86          for(int i=0; i<numVariables-1; i++)   // "numVariables-1"
                 leaves the last free space for store the ceck sum "
                ckSm" in vector c
87          {
88                  tmp = prova.at(i);
89                  memcpy(c+i*sizeof(tmp), (char*)&tmp, sizeof(tmp))
                        ;   // copies the 11 variables of vector
                        prova in vector c
90                  // "sizeof(tmp)" returns 4 bytes
91          }
92          qint32 ckSm = 0;   // ckSm saves the sum of the 44 bytes
                stored in vector c
93          for(int i=0; i<(numVariables-1)*4; i++)
94          {
95                  ckSm = ckSm+c[i];   // calculates the sum of the
                        44 bytes, cycle after cycle for the ceck sum
                        control
96          }
97          memcpy(c+(numVariables-1)*sizeof(ckSm), (char*)&ckSm,
                sizeof(ckSm));   // copies ckSm in the last 4 bytes
                of the vector c
98          // "sizeof(ckSm)" returns 4 bytes
99          // sending data to "IpToBind_"
100         socketOut_->writeDatagram(reinterpret_cast<const char*>(c
                ), sizeof(c), IpToBind_, portToBindOut_);
101 }
102
103 void MainWindow::sendData()
104 {
105         if(isON == 1)   // start simulation
106         {
107                 ui->statusBar->showMessage("Sending data...");
108                 HelloUDP();
109         }
110         else   // stop simulation
111         {
112                 ui->statusBar->showMessage("Simulation stopped!")
                        ;
113                 HelloUDP();
114         }
115 }
116
117 void MainWindow::readyRead()
118 {
119         qint64 messageSize = socketIn_->pendingDatagramSize();
                // size of data received from the socket
```

```
120         QHostAddress sender;
121         quint16 senderPort;
122         receivedData_.resize(messageSize/sizeof(float));   //
                resizes vector to 10 variables
123         qint16 nInFloatsState_ = 10;   // number of variables to
                receive
124         char *data;
125         data = (char *) malloc(messageSize);   // pointer to a
                dedicated memory area used to download the received
                data
126         float tmp;
127
128         if(messageSize == sizeof(float)*nInFloatsState_)   // if
                data reception is successful without interruption
129         {
130                 socketIn_->readDatagram(data, messageSize, &
                        sender, &senderPort);   // receiving data
131
132                 for(int i=0; i<messageSize/sizeof(float); i++)
133                 {
134                         memcpy(&tmp, (float*)(data+i*sizeof(float
                                )), sizeof(float));
135                         receivedData_.at(i) = tmp;
136                 }
137                 // display received data
138                 ui->TurbTorqLCD->display(receivedData_.at(0));
139                 ui->TurbSpeedLCD->display(receivedData_.at(1));
140                 ui->WindPowLCD->display(receivedData_.at(2));
141                 ui->ActPowerLCD->display(receivedData_.at(3));
142                 ui->ReactPowerLCD->display(receivedData_.at(4));
143                 if(receivedData_.at(5) < 0.00001)
144                 {
145                         int powData = 0;
146                         ui->PowFactLCD->display(powData);
147                 }
148                 else
149                 {
150                         ui->PowFactLCD->display(receivedData_.at
                                (5));
151                 }
152                 ui->VoutLCD->display(receivedData_.at(6));
153                 ui->IoutLCD->display(receivedData_.at(7));
154                 ui->VoutFreqLCD->display(receivedData_.at(8));
155                 if(receivedData_.at(9) < 0.0001)
156                 {
157                         int pitchData = 0;
158                         ui->PitchAnLCD->display(pitchData);
159                 }
160                 else
161                 {
162                         ui->PitchAnLCD->display(receivedData_.at
                                (9));
```

```
163                 }
164
165                 ui->statusBar->showMessage("Receiving data...");
166         }
167         else   // data reception isn't successful (transmission
                interruptions)
168         {
169                 char *devNull;   // pointer to a dedicated memory
                        area used to discard the received data
170                 devNull = (char *) malloc(socketIn_->
                        pendingDatagramSize());
171                 socketIn_->readDatagram(devNull, socketIn_->
                        pendingDatagramSize(), &sender, &senderPort);
172                 ui->statusBar->showMessage("Wrong data reception
                        !");
173         }
174 }
175
176 MainWindow::~MainWindow()
177 {
178         delete ui;
179 }
180
181 void MainWindow::on_StopButton_clicked()
182 {
183         isON = 0;   // if "isON" is = 0 the simulation is stopped
184 }
185
186 void MainWindow::on_WindSpeedSpinBox_editingFinished()
187 {
188         windSpeed = ui->WindSpeedSpinBox->value();   // assigns
                the writed value of the box in to "windSpeed"
189 }
190
191 void MainWindow::on_TurbRadSpinBox_editingFinished()
192 {
193         turbRad = ui->TurbRadSpinBox->value();
194 }
195
196 void MainWindow::on_MaxSpeedSpinBox_editingFinished()
197 {
198         maxSpeedValue = ui->MaxSpeedSpinBox->value();
199 }
200
201 void MainWindow::on_PolesSpinBox_editingFinished()
202 {
203         poles = ui->PolesSpinBox->value();
204 }
205
206 void MainWindow::on_ResistanceSpinBox_editingFinished()
207 {
208         resistance = ui->ResistanceSpinBox->value();
```

```
209 }
210
211 void MainWindow::on_MagFluxSpinBox_editingFinished()
212 {
213         magnFlux = ui->MagFluxSpinBox->value();
214 }
215
216 void MainWindow::on_RotInSpinBox_editingFinished()
217 {
218         rotIn = ui->RotInSpinBox->value();
219 }
220
221 void MainWindow::on_GearRatioSpinBox_editingFinished()
222 {
223         gearRatio = ui->GearRatioSpinBox->value();
224 }
225
226 void MainWindow::on_LimiterSpinBox_editingFinished()
227 {
228         voutValue = ui->LimiterSpinBox->value();
229 }
230
231 void MainWindow::on_GenInSpinBox_editingFinished()
232 {
233         genIn = ui->GenInSpinBox->value();
234 }
235
236 void MainWindow::on_actionExit_triggered()
237 {
238         close();   // close the application
239 }
240
241 void MainWindow::on_actionAbout_Turbine_Control_triggered()
242 {
243         QMessageBox::information(this, "About Turbine Control",
              "\nTurbine Control\n\nRelease 1.0\nBuilt on Febbruary
               2016\n\nDevelopers:  Andrea Petucco, Stefano
              Tonegato, Luca Tagliapietra\n\n\nUniversity of Padova
              \nDTG - Department of Engineering and Management of
              Industrial Systems", "Close");
244 }
245
246 void MainWindow::on_StartButton_clicked()
247 {
248         isON = 1;   // if "isON" is = 1 the simulation is running
249 }
250
251 void MainWindow::on_actionRun_triggered()
252 {
253         isON = 1;   // if "isON" is = 1 the simulation is running
254 }
255
```

```
256 void MainWindow::on_actionStop_triggered()
257 {
258         isON = 0;   // if "isON" is = 0 the simulation is stopped
259 }
260
261 void MainWindow::on_VoutLimCheckBox_clicked()
262 {
263         if(ui->VoutLimCheckBox->isChecked())
264         {
265                 limiterON = true;
266                 ui->actionVout_Limiter->setChecked(true);
267         }
268         else
269         {
270                 limiterON = false;
271                 ui->actionVout_Limiter->setChecked(false);
272         }
273 }
274
275 void MainWindow::on_actionVout_Limiter_triggered()
276 {
277         if(ui->actionVout_Limiter->isChecked())
278         {
279                 ui->VoutLimCheckBox->setChecked(true);
280                 limiterON = true;
281         }
282         else
283         {
284                 limiterON = false;
285                 ui->VoutLimCheckBox->setChecked(false);
286         }
287 }
288
289 void MainWindow::on_MaxSpeedCheckBox_clicked()
290 {
291         if(ui->MaxSpeedCheckBox->isChecked())
292         {
293                 maxSpeedON = true;
294                 ui->actionMax_Speed->setChecked(true);
295         }
296         else
297         {
298                 maxSpeedON = false;
299                 ui->actionMax_Speed->setChecked(false);
300         }
301 }
302
303 void MainWindow::on_actionMax_Speed_triggered()
304 {
305         if(ui->actionMax_Speed->isChecked())
306         {
307                 maxSpeedON = true;
```

```
308                 ui->MaxSpeedCheckBox->setChecked(true);
309         }
310         else
311         {
312                 maxSpeedON = false;
313                 ui->MaxSpeedCheckBox->setChecked(false);
314         }
315 }
```

BIBLIOGRAFIA

[1] User datagram protocol, 1980. URL http://www.tools.ietf.org/html/rfc768.

[2] Ethernet, 2007. URL http://www.it.wikipedia.org/wiki/Ethernet.

[3] Vento, curva di potenza, stima della producibilitá, 2009. URL http://www.nextville.it.

[4] Una prima classificazione delle turbine eoliche, 2011. URL http://www.consulente-energia.com.

[5] Socket: cosa sono e come funzionano, 2015. URL http://www.fortyzone.it/socket-cosa-sono.

[6] Juan Carlos Ausin, Daniel Navarro Gevers, and Bjorn Andresen. *Fault ride-through capability test unit for wind turbines*, volume 11. John Wiley & Sons, Ltd., 2008. doi: 10.1002/we.255.

[7] CEI Comitato Elettrotecnico Italiano. Standard-CEI 0-16, reference technical rules for the connection of active and passive consumers to the hv and mv electrical networks of distribution company. 2011-12. Standards.

[8] CEI Comitato Elettrotecnico Italiano. Standard-CEI 0-21, reference technical rules for the connection of active and passive users to the lc electrical utilities, 2011-12. Standards.

[9] Nicolas Espinoza. *Grid Code Testing of Wind Turbines by Voltage Source Converter Based Test Equipment*. Chalmers University of Technology, PHD Thesis, 2015.

[10] A. Helmedag, T. Isermann, U. Jassmann, D. Radner, D. Abel, G. Jacobs, and A. Monti. Testing nacelles of wind turbines with a hardware in the loop test bench. *Instrumentation Measurement Magazine, IEEE*, 17(5):26–33, Oct 2014. ISSN 1094-6969. doi: 10.1109/MIM.2014.6912198.

[11] A. Helmedag, T. Isermann, and A. Monti. Fault ride through certification of wind turbines based on a hardware in the loop setup. *Instrumentation and Measurement, IEEE Transactions on*, 63 (10):2312–2321, Oct 2014. ISSN 0018-9456. doi: 10.1109/TIM.2014.2315736.

[12] Texas Instruments. *TMS320F2837xS Delfino Microcontrollers - Technical Reference Manual*, 2015. URL http://www.ti.com.

RINGRAZIAMENTI

Desidero ricordare tutti coloro che mi hanno aiutato nella stesura della tesi con suggerimenti, critiche ed osservazioni: a loro va la mia gratitudine.

Ringrazio anzitutto il professor Paolo Mattavelli, Relatore, ed il dottorando Andrea Petucco, Correlatore: senza il loro supporto e la loro guida sapiente questa tesi non esisterebbe.

Ringrazio il dottorando Luca Tagliapietra per i suoi utili suggerimenti.

Un ringraziamento particolare va a Simone Menegatti per aver fornito l'hardware informatico necessario per lo sviluppo di questa tesi.

Un ringraziamento va ai vecchi compagni di studi Francesco e Roberto per i bei momenti passati assieme, sia di divertimento che di studio sfrenato.

Un ringraziamento anche alla compagnia di amici piú cari, compresi i colleghi dell'associazione tennis, che riescono ancora a sopportarmi.

Vorrei infine ringraziare i miei genitori, che grazie al loro sostegno mi hanno permesso di concludere questo lungo percorso.