UNIVERSITA' DEGLI STUDI DI CATANIA

FACOLTA' DI INGEGNERIA

Corso di Laurea Specialistica in Ingegneria Informatica

TESI DI LAUREA

Un sistema gestione dati per il monitoraggio di depositi di scorie radioattive: front-end, storage e rete

Candidato:

Sergio Simone Scirè Scappuzzo

A63/000174

Relatore:

Chia.mo Prof. Ing. Michele Malgeri

Correlatori:

Dott. Paolo Finocchiaro

Dott. Luigi Cosentino

Anno Accademico 2010/11

Indice dei contenuti

1 Introduzione

L'obiettivo di questo lavoro di tesi, maturato all'interno di un progetto sviluppato in collaborazione tra INFN [1] e Ansaldo Nucleare [2] (programma RIACE [INFN06]), è la realizzazione di un prototipo dimostrativo finalizzato al monitoraggio on-line di scorie radioattive a breve-medio termine. A tale proposito, la proposta DMNR (Detector Mesh for Nuclear Repositories) è nata per offrire un sistema distribuito, robusto, affidabile, granulare e basato su dispositivi a basso costo.

In ambito internazionale si definisce rifiuto radioattivo qualsiasi sostanza (in forma solida, liquida o gassosa), derivante da pratiche o interventi umani per la quale non sia previsto ulteriore utilizzo, che contenga o sia contaminata da radionuclidi a concentrazioni o livelli di radioattività superiori alle quantità permesse, stabilite dalle autorità competenti. [IAEA10]
Per scorie nucleari s'intende indicare il combustibile esausto originatosi all'interno dei reattori nucleari nel corso del loro esercizio. Esse rappresentano un sottoinsieme dei rifiuti radioattivi, a loro volta suddivisibili in base al livello di attività in tre categorie: basso, intermedio e alto. [ANPA26]

La pericolosità dei rifiuti radioattivi decresce con il passare del tempo poiché ogni specie decade seguendo una legge esponenziale. Sfortunatamente il tempo necessario affinché una data specie risulti non essere più dannosa per l'uomo e per l'ambiente, può variare da poche ore ad anni, per i rifiuti industriali e medicali, per arrivare a decine di migliaia di anni per quelli derivanti dallo smantellamento di reattori a fissione e armi nucleari.

---

[1] L'Istituto Nazionale di Fisica Nucleare è l'ente italiano che promuove, coordina ed effettua la ricerca scientifica nel campo della fisica nucleare, subnucleare e astro particellare, nonché lo sviluppo tecnologico necessario alle attività in tali settori. [INFN]

[2] Ansaldo Nucleare s.p.a. è un'azienda italiana che opera nel settore nucleare, realizzando centrali nucleari di terza generazione. [ANN]

I rifiuti a medio e breve termine vengono attualmente concentrati e confinati in fusti per essere stoccati all'interno delle strutture che li hanno prodotti, oppure trasportati in depositi superficiali dedicati allo stoccaggio; per i rifiuti a lungo termine invece si prevedono dei magazzini geologici come cave sotterranee o l'interno delle montagne.
Lo smantellamento delle centrali nucleari e lo stoccaggio dei rifiuti radioattivi da esse prodotti, pone a livello mondiale un argomento di discussione attuale: il monitoraggio delle scorie a breve, medio e lungo termine.

Qualunque sia la disposizione topologica o la struttura fisica dei bidoni di stoccaggio, sarebbe un grosso vantaggio poter misurare il tasso di radioattività intorno a ciascuno di essi; ciò permetterebbe, oltre ad avere un monitoraggio continuo della situazione complessiva, di verificare prontamente un danno alla struttura del bidone dovuto ad un eventuale cedimento strutturale.
Questo obiettivo potrebbe già oggi essere raggiunto utilizzando i classici rivelatori di radioattività a tubo Geiger-Muller, ma il costo di tale soluzione sarebbe proibitivo. Le procedure odierne di monitoraggio prevedono controlli periodici, ad opera di operatori specializzati o robots dotati di strumenti di misura adeguati, cui si aggiungono misurazioni ambientali effettuate all'interno dei depositi.
Il progetto DMNR [DMNR] ha permesso lo studio di un nuovo tipo di rivelatore di radiazioni ionizzanti, che presenta caratteristiche simili ai contatori Geiger-Muller (ma a differenza di questi risulta essere notevolmete più economico e versatile) e che può essere replicato sottoforma di griglia intorno ad ogni singolo bidone.
Ciascun rivelatore consiste di una fibra ottica realizzata con scintillatore plastico, con accoppiati alle due estremità due fotosensori al silicio (SiPM) di ultima generazione, capaci di rivelare i deboli segnali luminosi prodotti dall'interazione con la radiazione incidente, e convertirli in impulsi elettrici veloci.
In questo modo è possibile posizionare griglie di sensori intorno a ciascun bidone e registrare costantemente l'attività emessa, al fine di misurare in tempo reale il tasso di radiazione istantaneo e così registrare lo storico in un apposito Data Base.
Nell'ambito di DMNR è stata sviluppata sia l'elettronica di front-end che il sistema di conteggio eventi su piattaforma FPGA, per gestire il flusso dati proveniente dai sensori sul campo. Tale apparato, inoltre, è integrato da un sistema di trasmissione dati ridondante e da una console con interfaccia grafica ed immagazzinamento dati. [ANIMMA09] [ICENES09] [IPRD08]

1.1 Natura della ricerca

La tecnologia delle fibre scintillanti e dei fotorivelatori, si basa su una consolidata esperienza maturata in ambito INFN per la rivelazione delle particelle. In tale esperienza rientra anche la realizzazione di sistemi elettronici e informatici per il trattamento, l'elaborazione e la registrazione dei dati in tempo reale, direttamente derivante dalla tradizionale consuetudine alla realizzazione di analoghi apparati nell'ambito di esperimenti di fisica nucleare e delle alte energie.

I rivelatori della radiazione emessa dai fusti contenenti i rifiuti radioattivi, sono fibre ottiche realizzate con scintillatore plastico (produce fotoni nella lunghezza d'onda del visibile al passaggio di radiazioni ionizzanti), ai cui capi sono fotoaccoppiati due SiPM, Silicon Photomultiplier, ossia fotomoltiplicatori al silicio di nuova generazione, con sensibilità di rivelazione al singolo fotone [3].

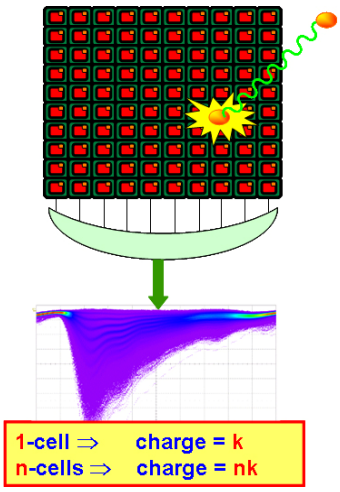


Figura 1

Generazione del segnale elettrico sul SiPM alla ricezione di un fotone

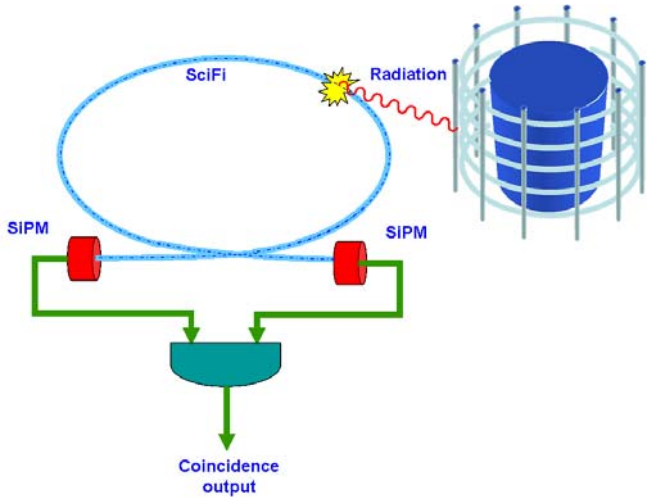


Figura 2

Disposizione delle fibre intorno il fusto e coincidenza dei segnali elettrici ai capi di ogni fibra scintillante

[3] Diversamente dai fotomoltiplicatori tradizionali (PMT o Photomultiplier Tubes), costruiti con tubi a vuoto, i SiPM sono prodotti direttamente da un wafer di silicio impiantando in esso matrici di microcelle lette in parallelo ciascuna delle quali è un diodo (Avalanche Photodiode o APD) che lavora in modalità Geiger.

L'impulso elettrico in uscita dal SiPM ha un'ampiezza proporzionale al numero di fotoni rivelati provenienti dalla fibra (Figura 1). Tali fotosensori sono sensibili al singolo fotone, in grado quindi di rivelare l'esigua quantità di luce rilasciata nella fibra della radiazione interagente con essa.

Con questa configurazione, quando si ha un evento di scintillazione indotto dal passaggio di radiazione ionizzante (principalmente radiazione γ), un certo numero di fotoni viene guidato verso le estremità della fibra, entro un intervallo di poche decine di nanosecondi.
La coincidenza dei due impulsi elettrici conseguenti, regolata su una finestra temporale legata al tempo massimo di percorrenza della luce, permette inoltre di ridurre drasticamente gli eventuali segnali spuri dovuti al fondo. Figura 2.
Ponendo una opportuna griglia di fibre intorno a ciascun fusto da monitorare, è possibile misurare in tempo reale la radiazione emessa dal fusto stesso in ogni sua parte, e registrare su files tutte le informazioni sull'attività emessa da ciascun fusto monitorato.
Qualora si dovessero verificare delle anomalie, consistenti ad esempio nell'apertura, anche di piccole crepe, nella superficie dei fusti, a seguito di reazioni chimico-fisiche indotte alla radiazione emessa dai rifiuti si riscontrerebbe un aumento nel tempo dei tassi di conteggio. Questo permetterebbe di identificare, con prontezza e decisione, la zona ove poter intervenire per eventuali operazioni di ripristino.

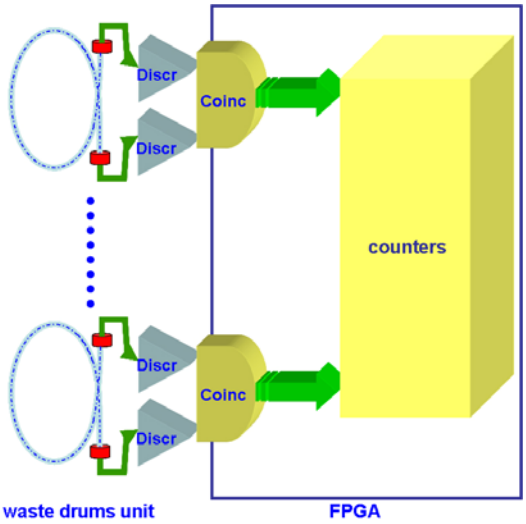


Figura 3

Coincidenza e conteggio eventi su tecnologia FPGA

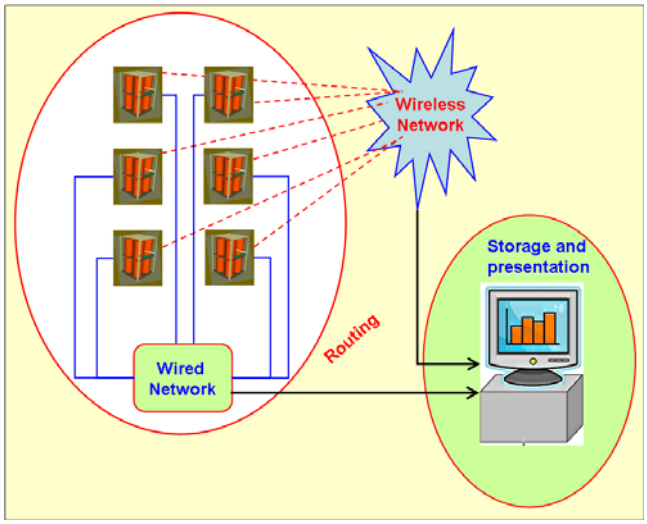


Figura 4

Reti di comunicazioni dati

La misura del tasso di conteggio della radiazione incidente su ciascuna fibra, è possibile grazie all'implementazione su tecnologia FPGA[4] di una batteria di contatori. Le informazioni su ogni bidone vengono poi impacchettate con dati aggiuntivi e inviate in ridondanza, tramite via seriale cablata e wireless, al sistema che si occuperà della supervisione del sito.

1.2 Rivelazione raggi γ

Come già accennato, il singolo rivelatore consiste in una fibra ottica scintillante di tipo plastico, accoppiata agli estremi a due SiPM.

L'efficienza luminosa della fibra, ossia la produzione di fotoni al passaggio di radiazione ionizzante, è pari a $\frac{10^4}{MeV}\left[\frac{fotoni}{energia}\right]$. L'efficienza di intrappolamento dei fotoni nella fibra varia dal 3.5% al 6%, e considerando anche un fattore di attenuazione esponenziale della luce che si propaga, la quantità di fotoni che arriva alle estremità è dell'orine di qualche decina.

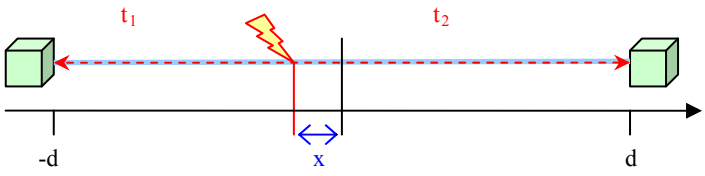


Figura 5

Evento produzione di luce

Il segnale luminoso, generato dalla radiazione all'interno della fibra, viene guidato dalla fibra ai suoi capi, ai quali arriverà negli istanti di tempo t_1 e t_2 rispettivamente (Figura 5).

Conoscendo la velocità di propagazione della luce sulla fibra v, parametro noto, è possibile determinare la posizione d'impatto x, in un sistema di riferimento che pone lo zero al centro della fibra stessa, tramite la relazione: $x = v \cdot \frac{t_1 - t_2}{2}$.

Questa semplice tecnica risulta però di difficile attuazione perché la produzione di fotoni da parte della fibra non è istantanea, segue una legge statistica esponenziale che introduce quindi

[4] Field Programmable Gate Array, dispositivi digitali la cui funzionalità è programmata via software con linguaggi di descrizione hardware (VHDL, Verilog).

indeterminazione, per questo tipo di fibra è circa 2 ns diviso la radice del numero di fotoni rilevati: un'indeterminazione pari a 2 ns porterebbe ad una risoluzione spaziale pari a 40 cm.

La soluzione proposta non intende effettuare misure di tempo per la determinazione della posizione d'impatto, si vuole invece implementare un sistema di monitoraggio statistico a breve-lungo termine, grazie al quale è possibile ricostruire una visione tridimensionale della radiazione emessa dai bidoni.
La struttura di rivelatori intorno ai bidoni sarà dunque composta da una griglia di fibre, disposte in una configurazione anulare-longitudinale. Il passo tra le fibre corrisponderà alla precisione spaziale richiesta per la localizzazione.

La struttura di questa tesi di laurea è stata suddivisa in tre sezioni, ciascuna delle quali mostrerà nel dettaglio i seguenti argomenti:

- DETECTOR: sviluppo del sistema elettronico di conteggio eventi e invio dei dati;
- MESH: sviluppo della logica di acquisizione, elaborazione e presentazione dati;
- NUCLEAR REPOSITORIES: realizzazione pratica dei rivelatori oggetto di studio, seguita da risultati sperimentali.

In conclusione una vasta appendice.

DETECTOR

I capitoli seguenti affronteranno nel dettaglio la realizzazione del sistema di acquisizione e trasmissione dati.

La scelta di utilizzare logiche programmabili FPGA è stata dettata dalla volontà di utilizzare sistemi elettronici economici e affidabili, ma allo stesso tempo performanti e versatili.

Oggi esistono al mondo diversi produttori di FPGA di ottima qualità, di sicuro la scelta non poteva che convergere su uno dei tre maggiori vendor[5].

Per lo sviluppo del sistema di acquisizione è stata utilizzata una development board Altera DE1 con FPGA della famiglia Cyclone II, di sicuro un prodotto con ottimo rapporto qualità-prezzo.

Tutti i moduli sono stati scritti in VHDL, linguaggio di descrizione hardware che permette di tradurre, in linguaggio di programmazione, il comportamento desiderato per i moduli elettronici.

La scelta del VHDL risulta essere vincente, nell'ottica di proporre un sistema indipendente da un particolare vendor di dispositivi: se in futuro si rendesse necessaria la migrazione su altra piattaforma ciò sarebbe possibile, ricompilando su nuovo target e rimappando i pins del dispositivo, senza alterare la struttura del programma.

Il sistema non è stato sovraccaricato di funzioni, al contrario si è scelto di far compiere all'FPGA una sola mansione: contare per ogni rivelatore, composto da una fibra scintillante accoppiata a due SiPM, il numero di eventi dovuti a radiazione. Tale quantità, integrata sul periodo di tempo in cui è stata registrata, permette a destinazione di ricavare il tasso di radiazione istantaneo registrato dai singoli rivelatori.

Il protocollo di trasmissione adottato è lo UART, standard de facto per la comunicazione tra dispositivi elettronici, che ha permesso un'immediata verifica della bontà delle trasmissioni, verso console remota.

---

[5] Altera [ALT], Xilinx [XIL], Actel [ACT]

Sono stati testate sia la trasmissione classica RS-232, utilizzata dai vecchi modem e stampanti seriali, che la sua versione industriale RS-422, che con segnalazione elettrica differenziale risulta più adatta per ambienti con rumore elettromagnetico elevato.
Sono attualmente sotto studio ulteriori modalità di trasmissione: RS-485 che permetterà il collegamento full-duplex di più dispositivi sullo stesso bus; Ethernet che con la sua versatilità renderà semplice l'indirizzamento dei dispositivi da remoto; ZigBee che permetterà di avere un canale di comunicazione via radio efficiente e con basso consumo energetico.
Queste modalità di trasmissione non saranno necessariemente in concorrenza tra loro poiché si prevede di fornire il sistema di un doppio canale di trasmissione dati, ridondanza che permette a destinazione di verificare l'integrità dei dati ricevuti.
La fase prototipale iniziale è stata caratterizzata da tanti piccoli traguardi nella realizzazione dei singoli moduli. Allo stato attuale il sistema è quasi totalmente parametrizzato per la scelta dei valori caratteristici di progetto (numero canali da monitorare, dimensione dei buffer di contenimento dati, risoluzione temporale, finestra coincidenza per i segnali in ingresso) da scegliere in fase di compilazione; altri sono modificabili a run time tramite switches presenti sulla board (intervallo di invio dati, baud rate di trasmissione, id dispositivo).
La presentazione dettagliata dei vari moduli, rispecchia l'approccio di tipo bottom-up che si è scelto di seguire nella progettazione completa del sistema.

Perché utilizzare gli FPGA nei sistemi embedded

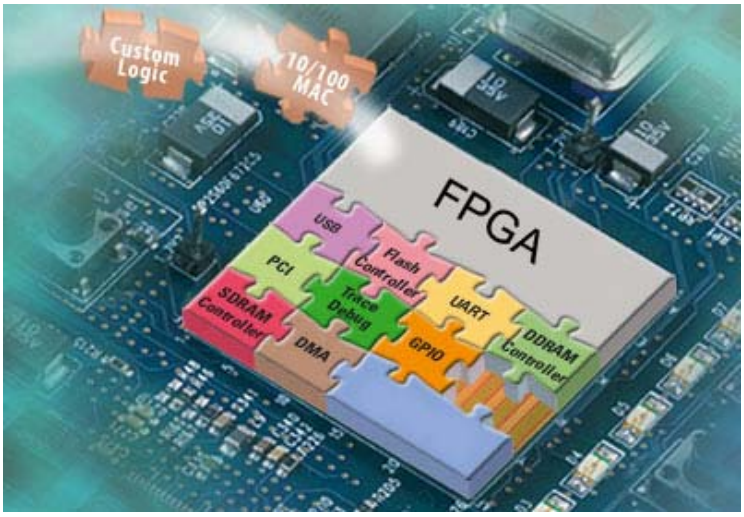


Figura 6

- **Strumento di sviluppo efficace per incrementare la produttività**

Con i tempi di sviluppo odierni tutto ciò che permette una riduzione dei tempi di produzione risulta essere un vantaggio competitivo nella corsa al mercato. I sistemi di sviluppo su piattaforma FPGA permettono una veloce fase di prototipazione, sviluppo e messa in opera di sistemi embedded, riducendo così il time-to-market.

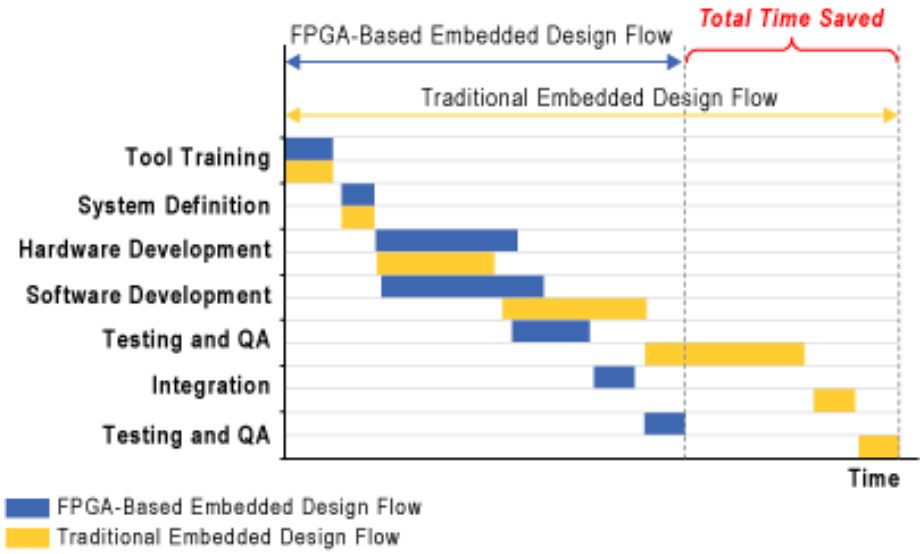


Figura 7

Il parallel design flow riduce il time-to-market

- **Protezione dell'investimento sul software contro l'obsolescenza dell'hardware**

Ogni campo dell'industria è in qualche modo soggetto all'obsolescenza dei dispositivi che utilizza, in particolar modo quelli che hanno un ciclo di vita molto lungo (automotive, militare, industriale, aerospaziale, medico).

In ogni sistema embedded il costo maggiore degli investimenti è dettato dallo sviluppo del software che, a meno di conoscere a priori il design definitivo, è sempre vulnerabile all'obsolescenza dell'hardware. [ALTa]

Questo fenomeno si paga inevitabilmente in termini di tempo, denaro e risorse di sviluppo.

Per lo sviluppo del modulo di rivelazione si è utilizzato la development board Altera DE1, su cui è montato un FPGA Altera EP2C20, appartenente alla famiglia Cyclone II.

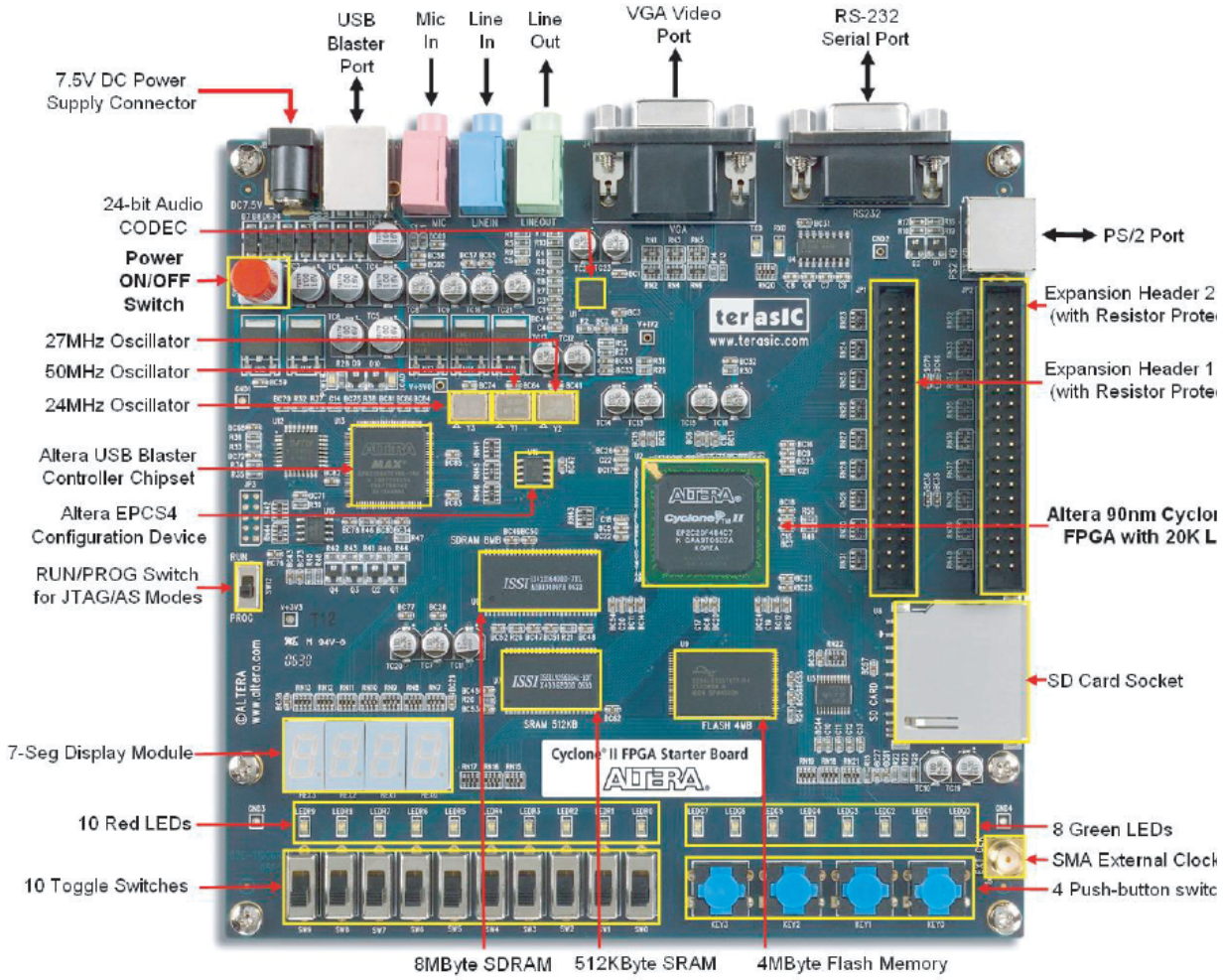


Figura 8

La development board Altera DE1

Sulla board è presente il seguente hardware :

- FPGA Altera Cyclone II 2C20
- Altera Serial Configuration device – EPCS4
- USB Blaster (on board) per la programmazione; sono supportati i modi JTAG e Active Serial (AS)
- 512 KB SRAM
- 8 MB SDRAM
- 4 MB Flash memory
- SD Card socket
- 4 pulsanti pushbutton
- 10 toggle switches
- 10 LEDs rossi

- 8 LEDs verdi
- 3 oscillatori al quarzo: 50 MHz, 27 MHz, 24 MHz
- CODEC audio 24 bit (qualità CD) con jacks line-in, line-out e microfono
- DAC VGA con connettore VGA 25 poli
- RS-232 transreceiver con connettore 9 poli
- Due slot di espansione a 40 pin

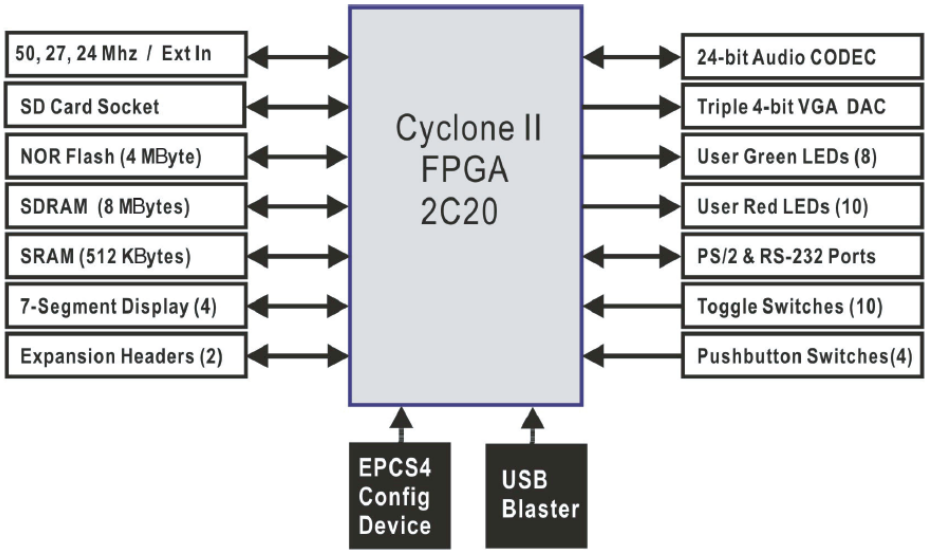


Figura 9

Block diagram

Configurare il Cyclone II FPGA

La board DE1 contiene una chip seriale EEPROM che immagazzina i dati di configurazione per il Cyclone II FPGA. Questa configurazione viene automaticamente caricata dal chip sull'FPGA ogni volta che la board viene accesa. Utilizzando il software di sviluppo (Altera Quartus II) è possibile programmare l'FPGA in due modi [ALTb]:

- JTAG Programming: In questo modo, così chiamato seguendo lo standard IEEE Joint Test Action Group, il bit stream di configurazione è caricato direttamente sull'FPGA. Il dispositivo manterrà la configurazione per tutto il tempo che la board sarà alimentata ; la configurazione andrà persa allo spegnimento.

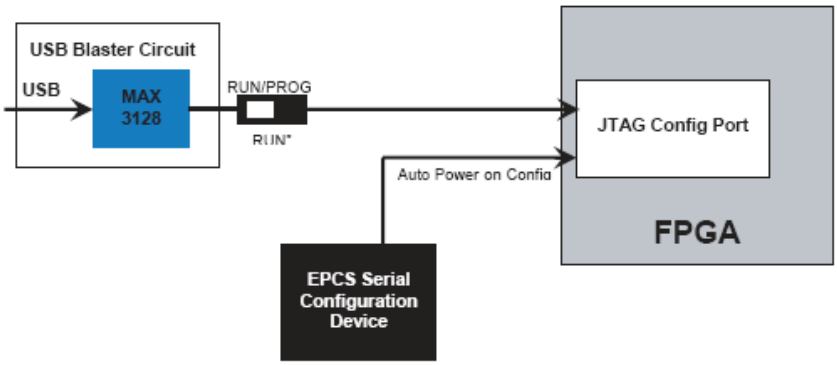


Figura 10

Schema JTAG configuration

- AS Programming: In questo modo, chiamato Active Serial programming, il bit stream di configurazione è caricato sul chip EEPROM Altera EPCS4 che assicura il salvataggio non volatile. L'informazione è mantenuta anche in mancanza di alimentazione esterna, così da essere automaticamente ricaricata sull'FPGA all'accensione successiva.

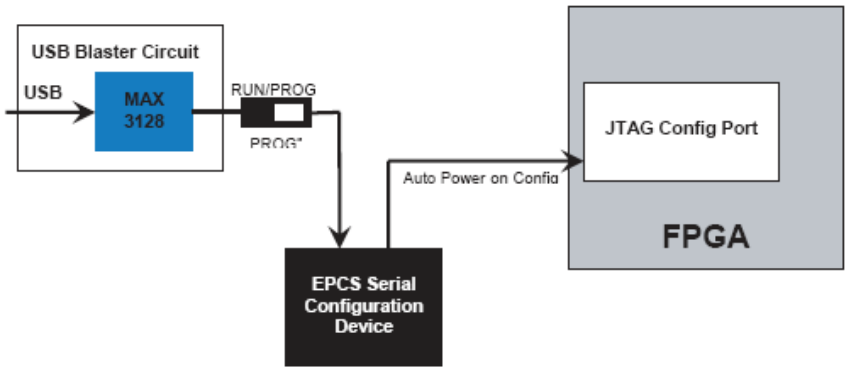
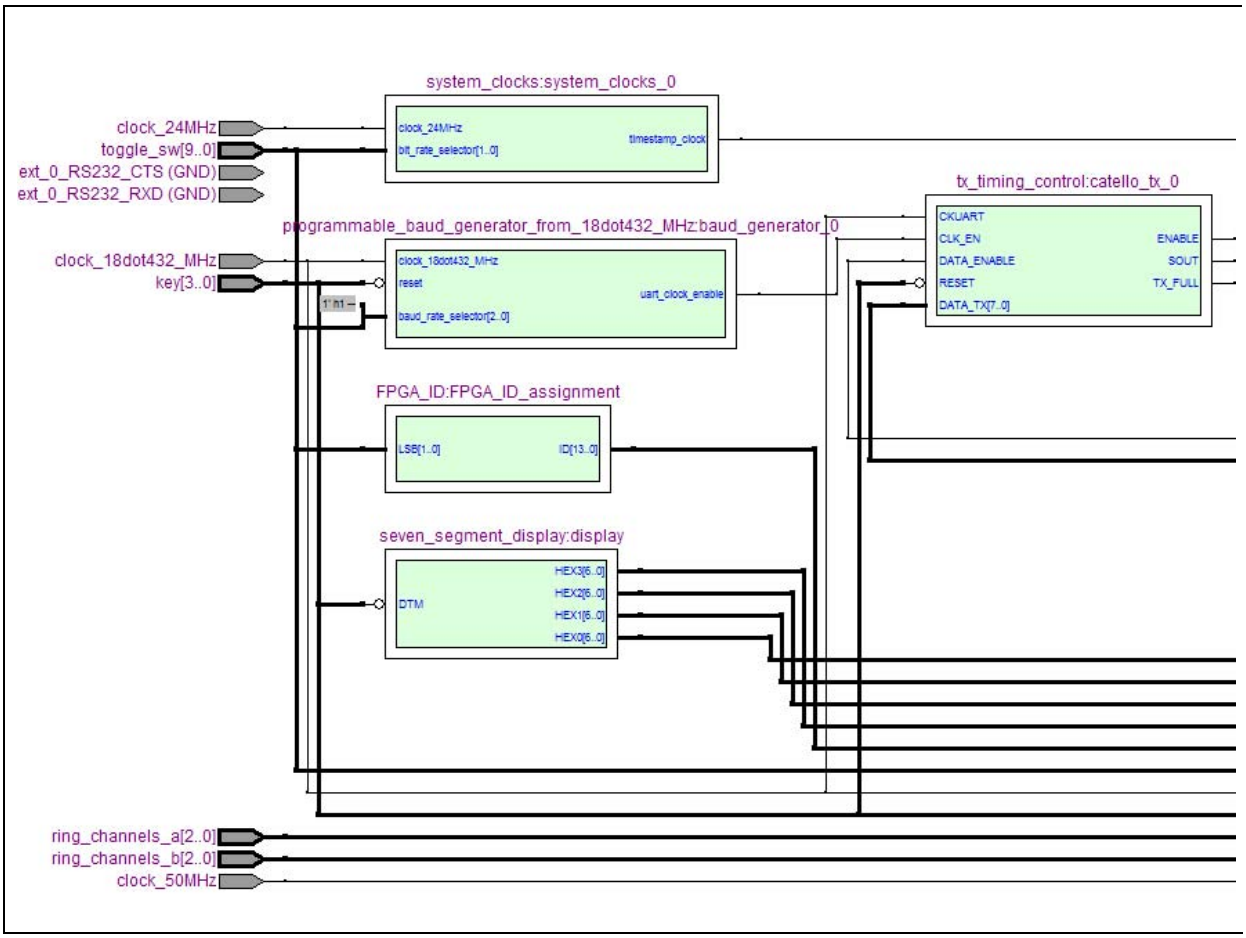


Figura 11

Schema AS configuration

2 Architettura moduli



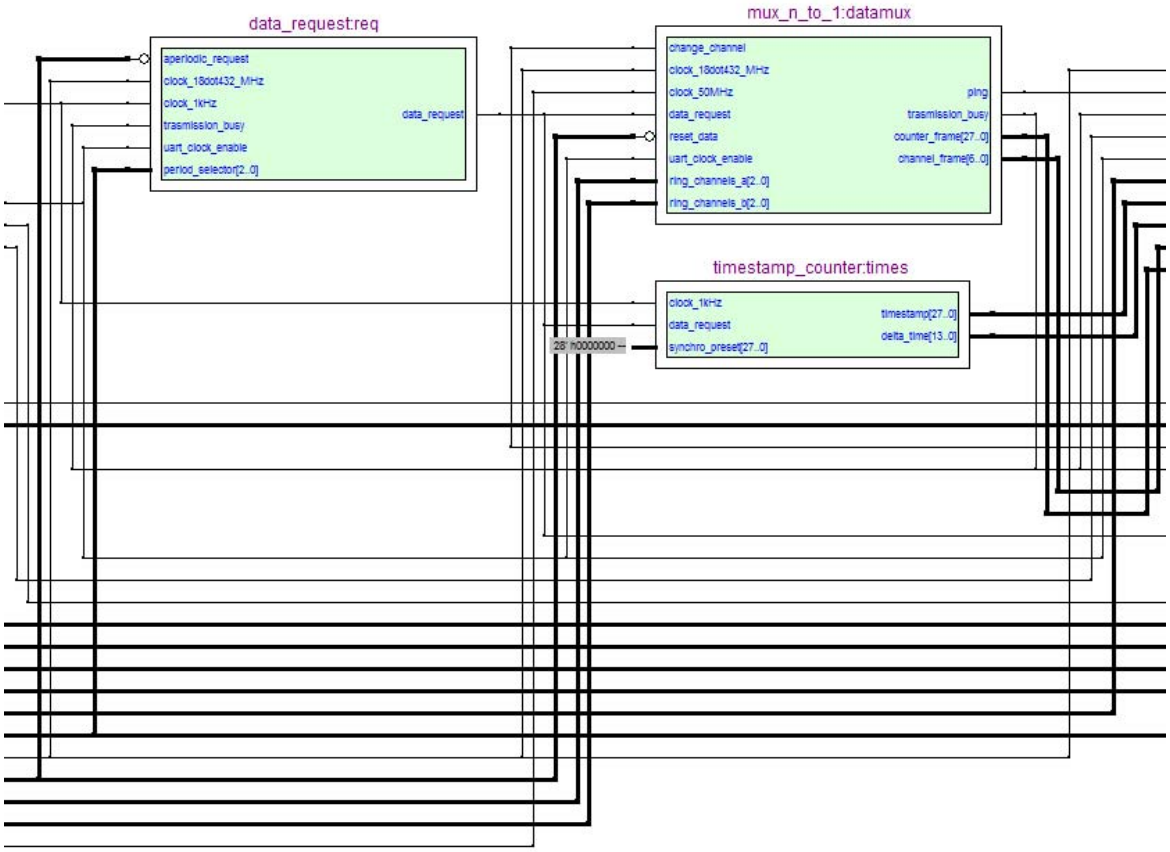
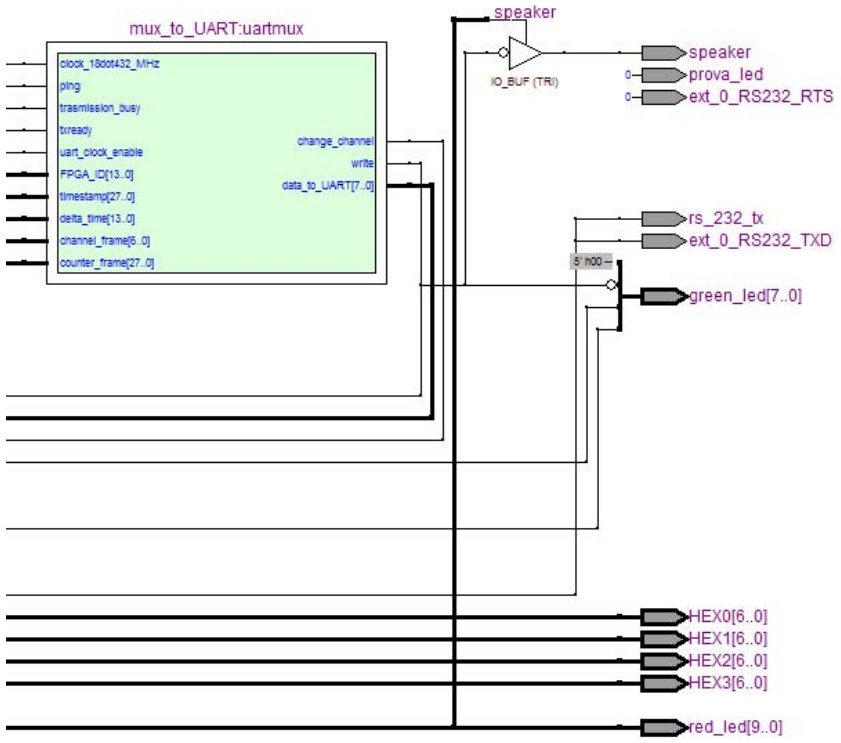


Figura 12

RTL Completo del programma



2.1 Data request

Il modulo si occupa di produrre il segnale di avvio della trasmissione, seguendo la richieste nel tempo sia periodiche, stabilite in fase di programmazione, sia aperiodiche, sotto richiesta esplicita dell'utente.

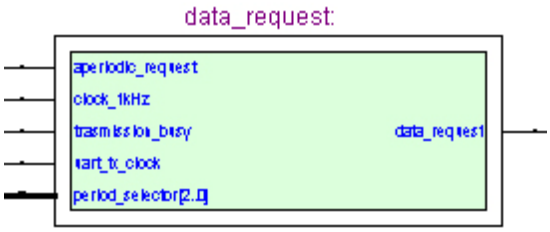


Figura 13

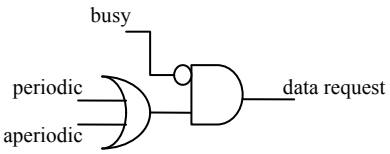


Figura 14

Riceve in ingresso un clock a frequenza 1 kHz, che serve come base temporale per calcolare l'intervallo di invio periodico dei dati acquisiti. La scelta di questo intervallo si ha tramite 3 bit (period_selector[2..0]) comandati da 3 toggle switch sulla board, parametro impostable manualmente a processo già inviato ma facilmente rimpiazzabile da un registro programabbile da remoto.

La tabella seguente mostra le combinazioni implementate:

Tabella 1

Combinazione	Δt [s]
000	10
001	30
011	60
010	90
110	120
100	240
101	300
111	2

La condizione di richiesta aperiodica è comandata da un push button switch sulla board, anche questa riprogrammabile con un bit di sistema comandato da remoto.

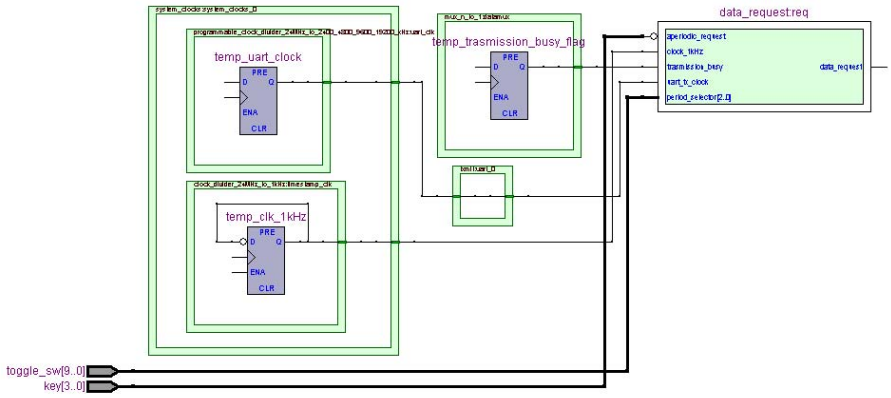


Figura 15

La condizione in uscita è il risultato di un OR logico tra la richiesta periodica e aperiodica, a condizione che la comunicazione seriale non sia già in corso, quindi non ancora conclusa. Questo segnale deriva direttamente dal blocco che si occupa di impacchettare i dati e darli in pasto al blocco UART per la serializzazione sul mezzo fisico.

```
PROCESS(uart_tx_clock, periodic_req, aperiodic_req, trasmission_busy)
  BEGIN
    IF(uart_tx_clock'EVENT AND uart_tx_clock = '1') THEN
      data_req_sig <= '0';
      IF( (periodic_req = '1' OR aperiodic_req = '1')
                                        AND (trasmission_busy = '0') ) THEN
        data_req_sig <= '1';
      END IF;
    END IF;
END PROCESS;

PROCESS(uart_tx_clock, data_req_sig)
  BEGIN
    -- FLIP FLOP M/S (SLAVE)
    IF(uart_tx_clock'EVENT AND uart_tx_clock = '0') THEN
      data_request <= data_req_sig;
    END IF;
END PROCESS;
```

Listato 1

Il process dentro cui viene analizzata la somma (OR) della richiesta periodica e aperiodica è sincrono con il clock che determina il baud rate della trasmissione seriale, in particolare il segnale in uscita (data_req_sig) sarà un impulso con larghezza pari al periodo di clock dello

stesso. Tale segnale passerà inoltre per un altro registro, clockato sul fronte opposto del clock della UART, per la sincronizzazione di data_request al nuovo dominio temporale.

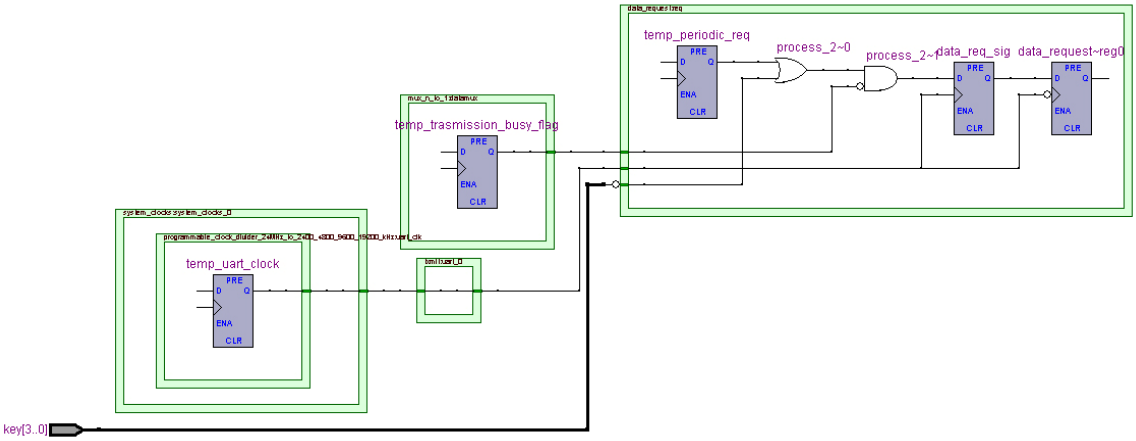


Figura 16

Questa implementazione si è ritenuta necessaria in quanto i domini temporali in ingresso, comandati dalla richiesta periodica e da quella aperiodica, sono scorrelati con il dominio temporale che governa la trasmissione.

2.2 Modulo timestamp

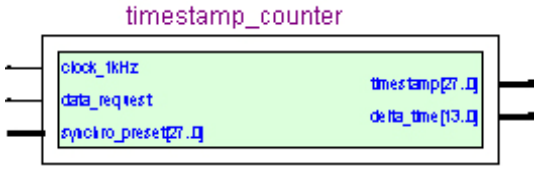


Figura 17

Il modulo riceve in ingresso il clock a 1 kHz, come base del tempo, e produce in uscita due dati che saranno comuni a tutti i canali durante la trasmissione seriale.
Sono stati implementati un timestamp assoluto, 28 bit, ed uno relativo, 14 bit; il funzionamento del modulo è il seguente:

- Il clock a 1 kHz, scalato a 1 Hz grazie ad una divisione per 1000, fornisce il dato in input ad un registro contatore a 32 bit, che si occupa di incrementare a dogni ciclo il valore registrato;

- Nell'istante in cui giunge in ingresso un segnale `data_request` il valore attuale viene copiato in un ulteriore registro a 32 bit, si passa dunque nella zona elettronica appartenente al dominio temporale della comunicazione UART;
- Il valore viene proposto in uscita, troncato dei 4 bit più significativi come timestamp assoluto a 28 bit;
- Per il calcolo dell'intervallo temporale tra una richiesta e quella precedente viene effettuata una sottrazione, tra il valore assoluto attuale e quello precedente, registrato in un ulteriore registro ad ogni invio di dati, e proposto in uscita troncato a 14 bit.

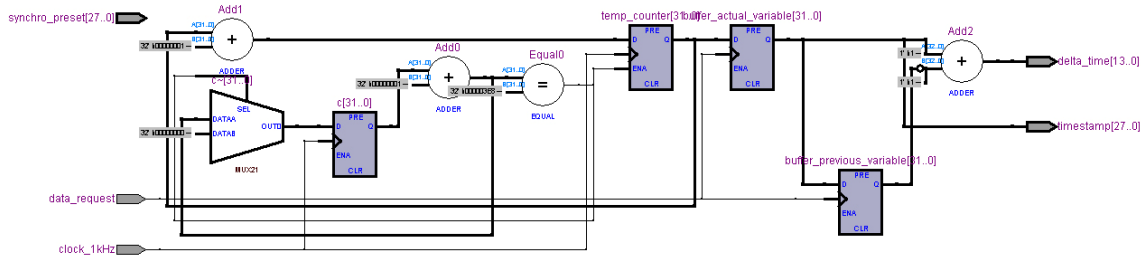


Figura 18

Un registro a 32 bit, alimentato a 1 kHz può contenere dati per

$$2^{32} \cdot \frac{[\text{sec}]}{1000} \cdot \frac{[\text{min}]}{60} \cdot \frac{[ore]}{60} \cdot \frac{[giorni]}{24} \cong 50 \text{ giorni}$$

La sua versione troncata a 28 bit si riempirà in un periodo pari a

$$2^{28} \cdot \frac{[\text{sec}]}{1000} \cdot \frac{[\text{min}]}{60} \cdot \frac{[ore]}{60} \cdot \frac{[giorni]}{24} \cong 3 \text{ giorni}$$

La differenza tra *actual* e *previous* timestamp fornisce un valore valido se i bit più significativi contengono lo stesso valore, e a 1 kHz la capacità è pari a

$2^{14} \cdot \frac{[\text{sec}]}{1000} \cong 16$ secondi, valore troppo piccolo per una realizzazione pratica.

Se il clock da 1 kHz viene però rallentato le capacità aumentano, come mostrato dalla seguente tabella.

Tabella 2

	1 kHz	100 Hz	10 Hz	1 Hz
32 bit	≈ 50 giorni	≈ 1.4 anni	≈ 14 anni	≈ 136 anni
28 bit	≈ 3 giorni	≈ 30 giorni	≈ 1 anno	≈ 8 anni
14 bit	≈ 16 sec	≈ 2.5 min	≈ 26 min	≈ 4.5 ore

Il rapporto di rallentamento non è un parametro configurabile da remoto, viene stabilito una volta per tutte in fase di compilazione. Esso corrisponde alla risoluzione temporale che si vuole avere per il timestamp.
A destinazione si renderà necessario, per leggere correttamente i dati ricevuti, conoscere il tempo fissato per il modulo timestamp, in modo da riuscire a risalire all'informazione corretta sul valore letto.

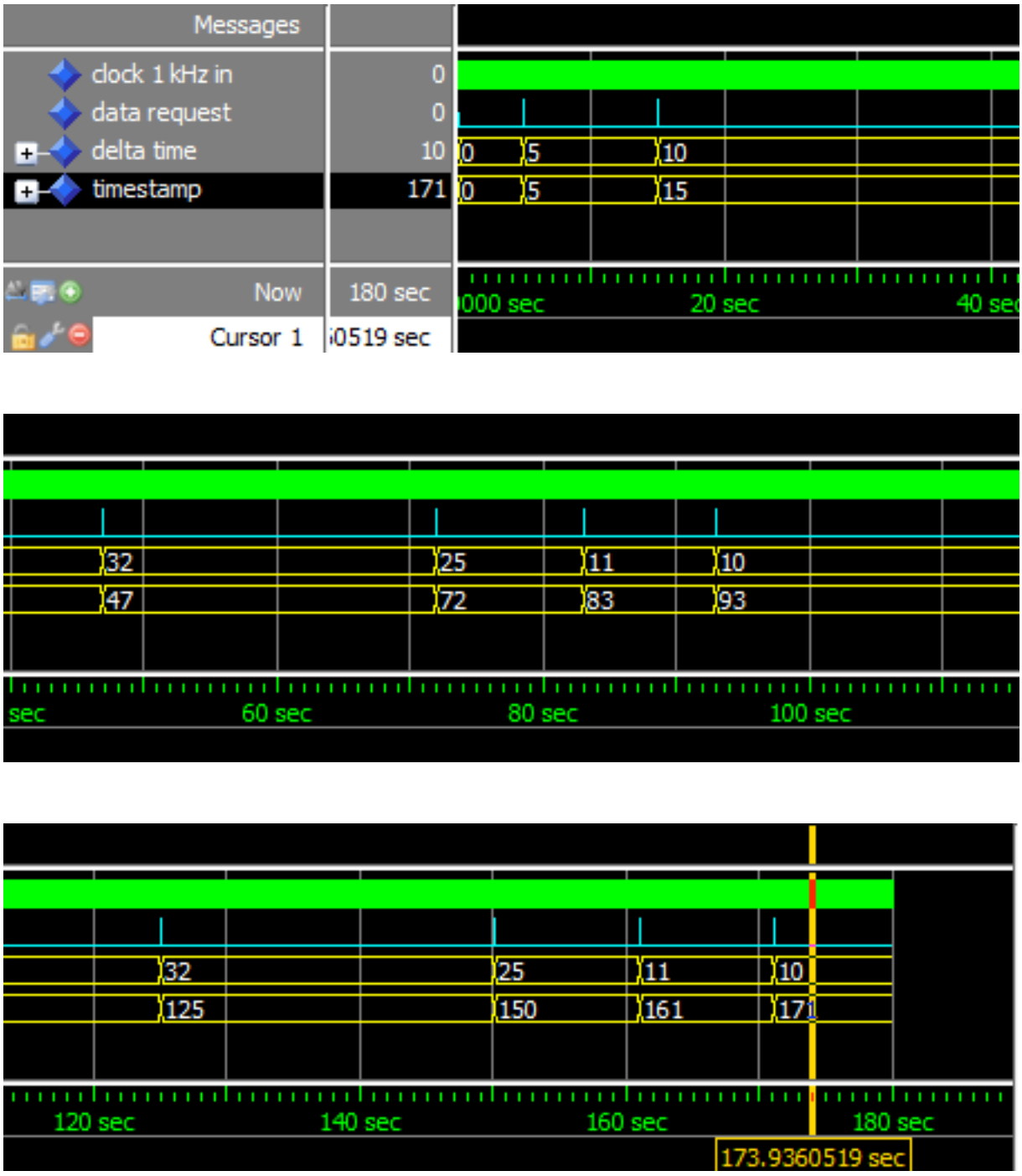


Figura 19

Risultato della simulazione

2.3 *Modulo coincidenza*

Il modulo di coincidenza è stato progettato per svolgere i seguenti compiti:

- attivarsi alla rilevazione di un segnale su un canale qualsiasi
- rimanere in attesa, per un time-slot fissato, di un segnale sull'altro canale in ingresso
- se il segnale arriva verificare la coincidenza e attivare l'uscita, in caso contrario azzerarsi

il comportamento è dunque analogo ad una porta AND temporizzata.

Il time-slot è funzione del clock in ingresso e di un registro a 3 bit che contiene la dimensione desiderata (da 0 a 7 cicli di clock).

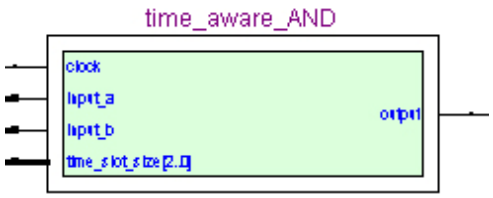


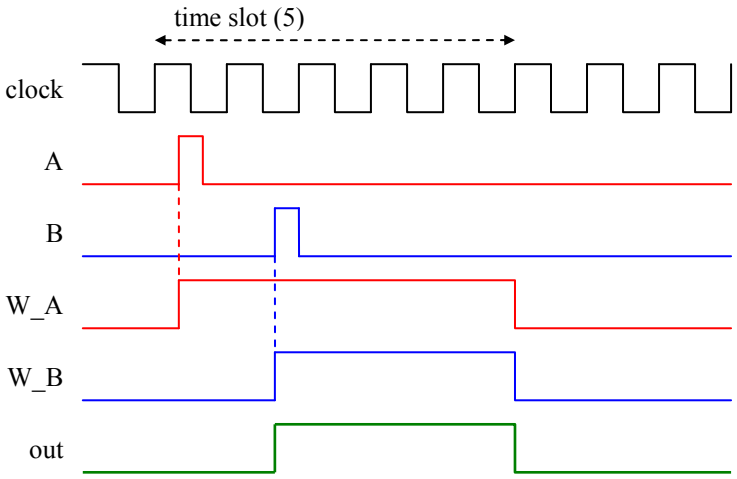
Figura 20

Interfaccia modulo coincidenza

2.3.1 *Simulazione del comportamento*

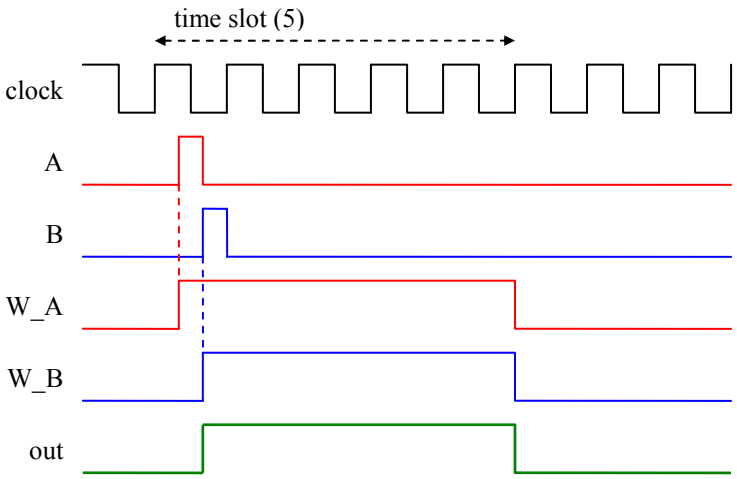
- **Desiderato:**

  quando su un canale arriva un impulso si avvia la finestra temporale di durata pari agli n slot fissati; se durante questo periodo giunge un impulso anche sull'altro canale allora la coincidenza è verificata



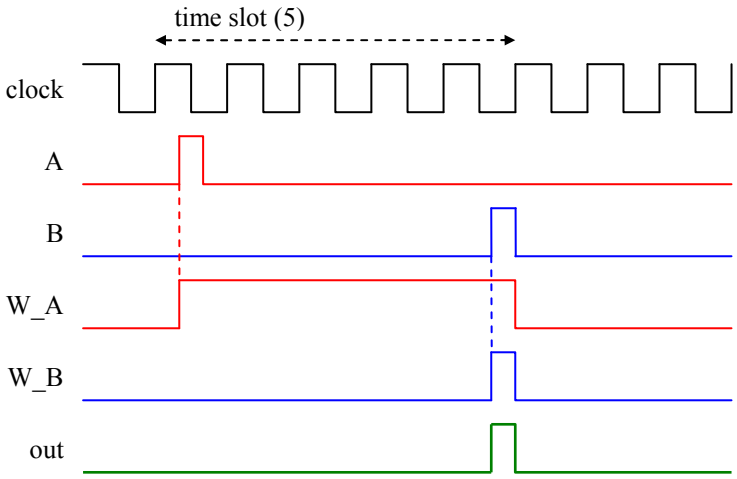
- **Caso limite:**

impulsi vicini



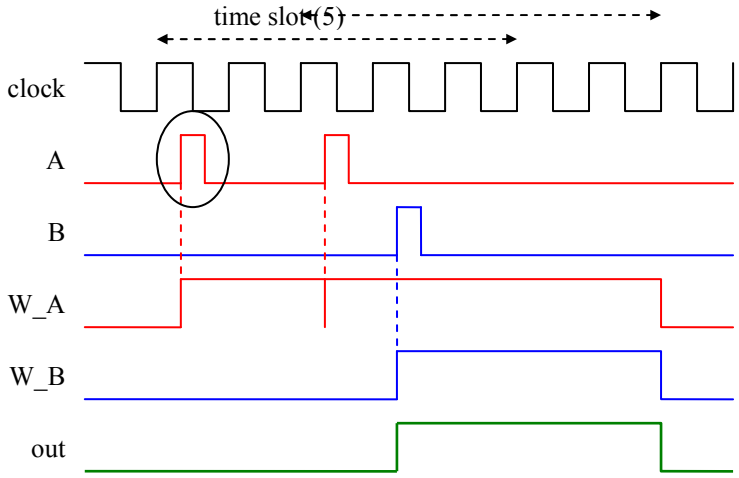
- **Caso limite:**

impulsi lontani


time slot (5)
clock
A
B
W_A
W_B
out

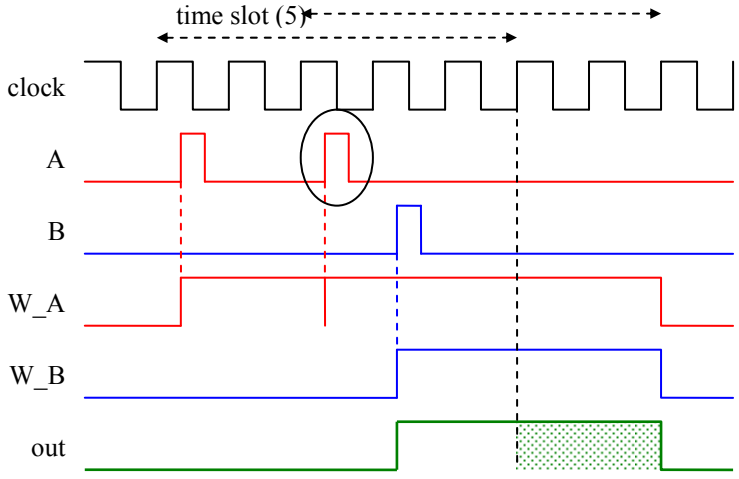
- **Caso eventi falso, vero, vero:**

il modulo si attiva con un impulso causato da rumore (cerchiato nel diagramma) ma l'arrivo di un impulso vero azzera il time slot e la coincidenza è verificata regolarmente



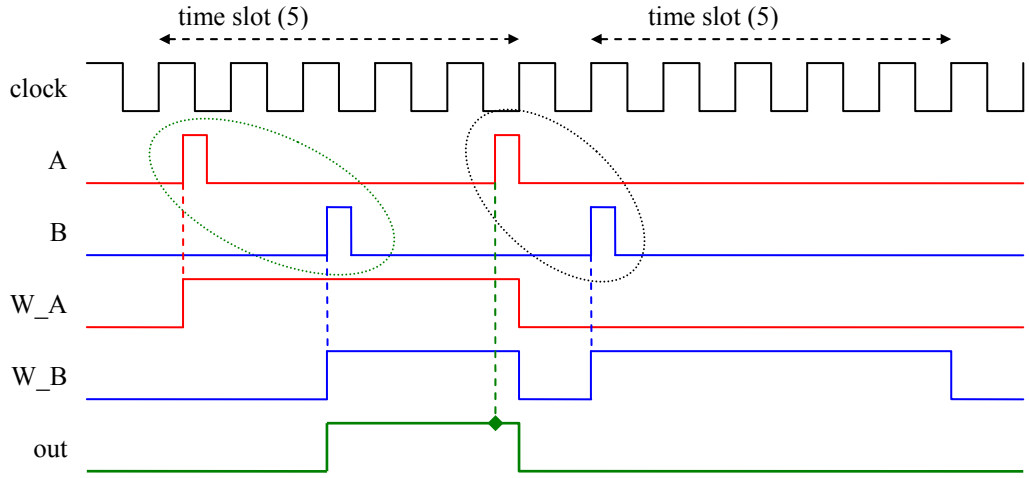
- **Caso eventi vero, falso, vero:**

comportamento analogo al precedente e non distinguibile da esso. L'effetto che produce un segnale di rumore (cerchiato nel diagramma) a conteggio avviato è prolungare il time slot, quindi il segnale di coincidenza; quando la coincidenza in uscita è verificata il modulo è insensibile ad altri segnali in ingresso. La zona ombreggiata indica gli istanti in cui si perderebbero segnali utili a causa del rumore precedente.



- **Caso coppie di eventi vicini:**

se una coppia di impulsi giunge durante la verifica di una coppia precedente la loro coincidenza andrà persa



La finestra temporale impone dunque il limite alla frequenza di campionamento massima raggiungibile: $f_{\max} = \frac{1}{timeslot}$.

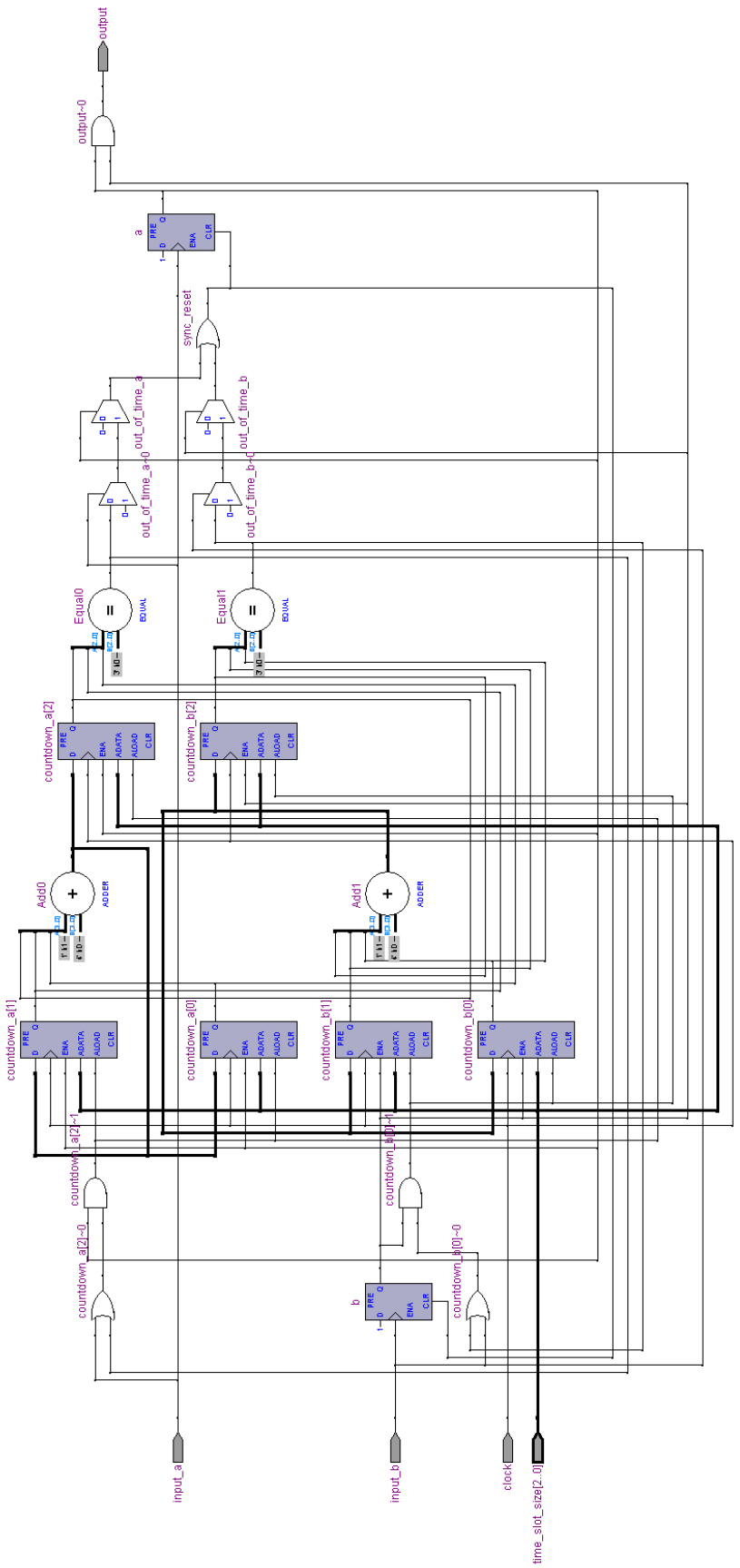


Figura 21

Visualizzazione RTL della logica combinatoria

```
ENTITY time_aware_AND IS

  PORT(

    input_a:          IN STD_LOGIC;  --SIPM A
    input_b:          IN STD_LOGIC;  --SIPM B
    clock:            IN STD_LOGIC;  --clock
    --moltiplicatore di periodo per finestra temporale, da 0 a 7 periodi
    time_slot_size:  IN STD_LOGIC_VECTOR(2 DOWNTO 0);
    probe_a, probe_b:  OUT STD_LOGIC;  --led di controllo
    output:                 OUT  STD_LOGIC    --coincidenza dei segnali in
ingresso
  );

END time_aware_AND;

ARCHITECTURE rtl OF time_aware_AND IS

  SIGNAL a:  STD_LOGIC := '0';  --attivo quando giunge un segnale sul SIPM
A
  SIGNAL b:  STD_LOGIC := '0';  --attivo quando giunge un segnale sul SIPM
B

  --valore iniziale per il conteggio al rovescio sul canale A
  SIGNAL countdown_a: STD_LOGIC_VECTOR (2 DOWNTO 0);
  --valore iniziale per il conteggio al rovescio sul canale B
  SIGNAL countdown_b:   STD_LOGIC_VECTOR (2 DOWNTO 0);

  --conteggio concluso, tempo scaduto sul canale A
  SIGNAL out_of_time_a: STD_LOGIC := '0';
  --conteggio concluso, tempo scaduto sul canale B
  SIGNAL out_of_time_b: STD_LOGIC := '0';

  --reset per i canali, sincrono non il clock dei registri
  SIGNAL sync_reset:  STD_LOGIC := '0';

  BEGIN

    --DETECT: se arriva un segnale sul canale lo imposta a '1',
    --il reset lo riporta a '0'
    channel_a_detect: PROCESS(input_a, sync_reset)
      BEGIN

        IF(sync_reset = '1') THEN
          a <= '0';
```

```
      ELSIF(input_a'EVENT AND input_a = '1') THEN
        a <= '1';
      END IF;

  END PROCESS channel_a_detect;

  channel_b_detect: PROCESS(input_b, sync_reset)
    BEGIN

      IF(sync_reset = '1') THEN
        b <= '0';
      ELSIF(input_b'EVENT AND input_b = '1') THEN
        b <= '1';
      END IF;

  END PROCESS channel_b_detect;

--CONSTRAIN:
channel_a_constrain: PROCESS(clock, a, input_a, input_b)
  BEGIN

    out_of_time_a <= '0';  --tempo scaduto impostato a falso
      IF(a = '1') THEN
         IF(input_a = '1') THEN
         --se arriva un altro segnale sullo stesso canale ricomincia
         --a contare
           countdown_a <= time_slot_size;
         ELSIF(countdown_a = "000") THEN
         --se termina il conteggio ripristina il valore predefinito
         --e notifica che il tempo è scaduto
           countdown_a <= time_slot_size;
           out_of_time_a <= '1';
         ELSIF(clock'EVENT AND clock = '1') THEN
         --decrementa di 1 il conteggio
           countdown_a <= countdown_a - 1;
         END IF;
       END IF;

  END PROCESS channel_a_constrain;

channel_b_constrain: PROCESS(clock, b, input_a, input_b)
  BEGIN

    out_of_time_b <= '0';
    --tempo scaduto impostato a falso
    IF(b = '1') THEN
```

```
        IF(input_b = '1') THEN
        --se arriva un altro segnale sullo stesso canale ricomincia a
contare
          countdown_b <= time_slot_size;
        ELSIF(countdown_b = "000") THEN
        --se termina il conteggio ripristina il valore predefinito
        --e notifica che il tempo è scaduto
          countdown_b <= time_slot_size;
          out_of_time_b <= '1';
        ELSIF(clock'EVENT AND clock = '1') THEN
        --decrementa di 1 il conteggio
          countdown_b <= countdown_b - 1;
        END IF;
      END IF;

  END PROCESS channel_b_constrain;

  --ripropone in uscita il valore logico dei canali
  probe_a <= a;
  probe_b <= b;

  --reset se un canale ha esaurito il conteggio
  sync_reset <= out_of_time_a OR out_of_time_b;

  --coincidenza dei canali A e B in uscita
  output <= a AND b;

END rtl;
```

Listato 2

Codice modulo di coincidenza

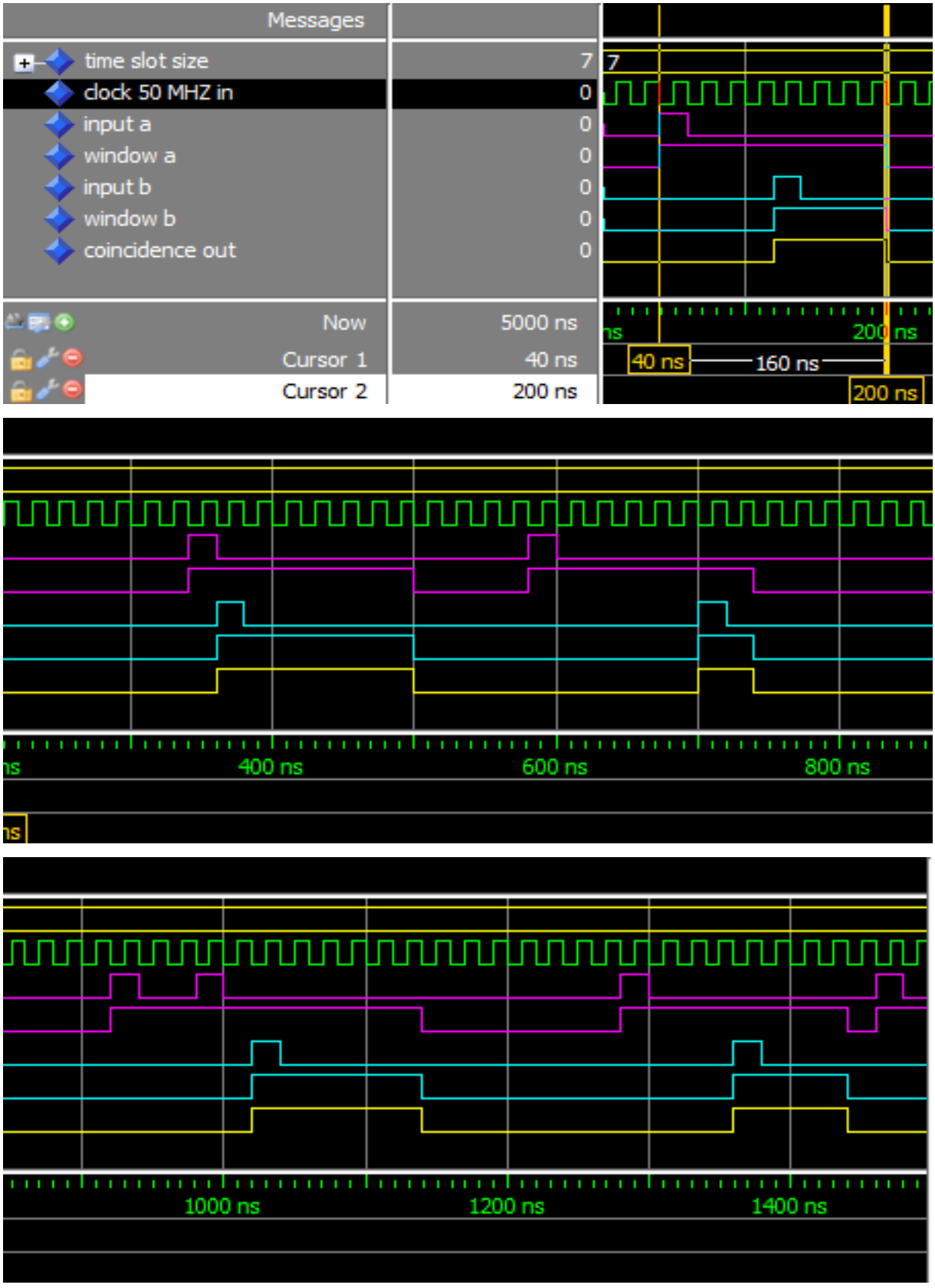


Figura 22

Risultato della simulazione

2.4 *Modulo contatore*

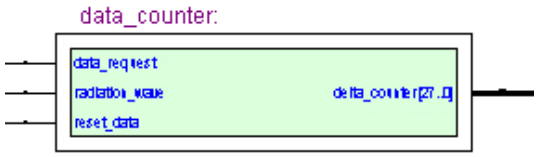


Figura 23

Il modulo contatore riceve in ingresso il segnale in uscita dal modulo di coincidenza e incrementa un contatore a 32 bit per gli eventi registrati. Quando arriva un segnale di richiesta invio dati trasferisce il proprio valore interno ad un altro registro a 32 bit appartenente al dominio temporale del `data_request`. Viene quindi effettuata una sottrazione tra il valore attuale e quello precedentemente registrato e il risultato, troncato a 28 bit viene trasferito all'esterno. Come per il modulo timestamp viene effettuata una troncatura per ridurre il numero di bit da trasmettere.

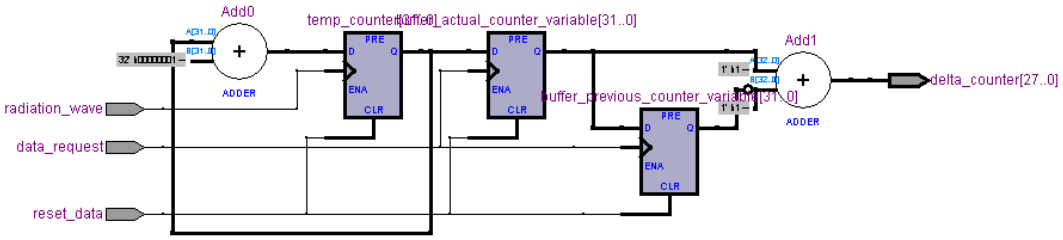


Figura 24

A differenza del modulo timestamp il tempo di riempimento dei registri non è deterministico, dipende infatti dal tasso di radiazione in ingresso, parametro da misurare, è però possibile calcolare dei tempi indicativi, come riportati in tabella

Tabella 3

	100 kHz	10 kHz	1 kHz	100 Hz	10 Hz
32 bit	≈ 12 ore	≈ 5 giorni	≈ 50 giorni	≈ 1.4 anni	≈ 13 anni
28 bit	≈ 45 min	≈ 7.5 0re	≈ 3 giorni	≈ 30 giorni	≈ 1 anno

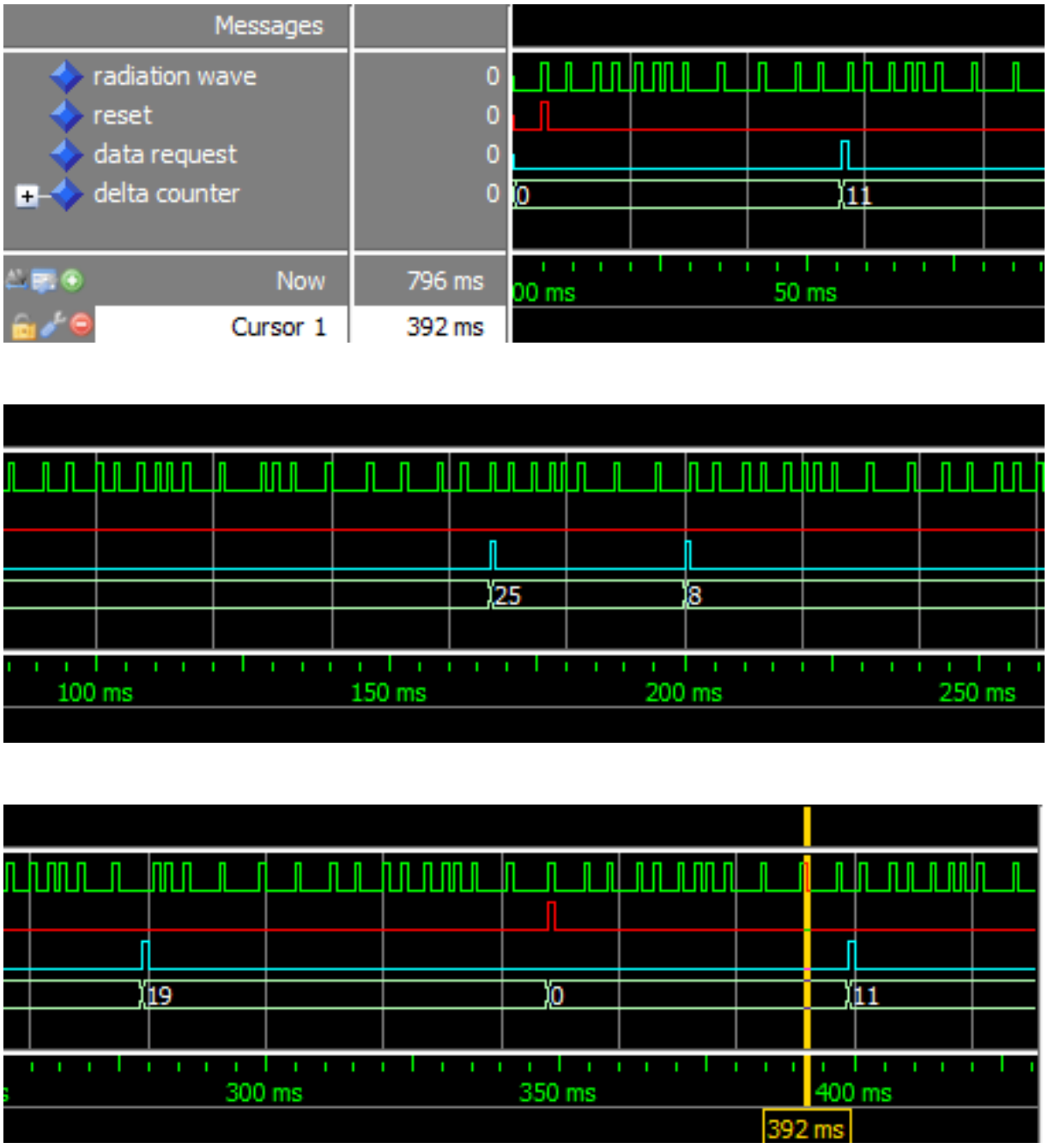


Figura 25

Risultato della simulazione

2.5 modulo n to 1

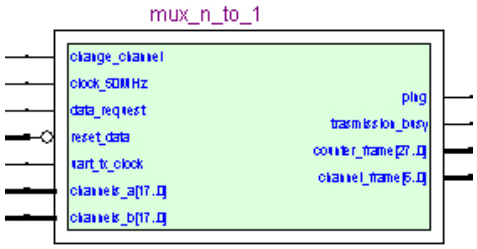


Figura 26

E' il modulo che che si occupa di instanziare, per ogni fibra da monitorare, la catena dei moduli necessaria. In esso vengono infatti instanziati i moduli di coincidenza per ogni coppia di canali associata alle fibre, e la loro uscita viene collegata ad un contatore indipendente.

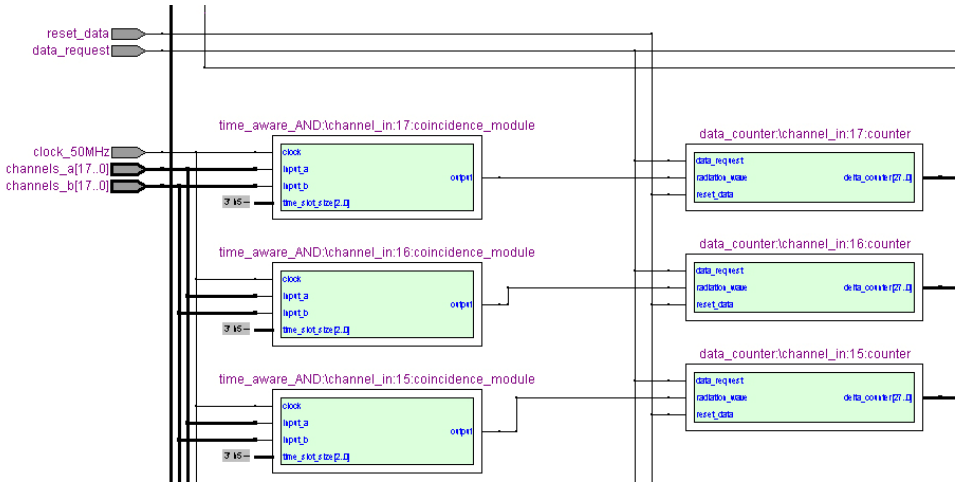


Figura 27

La procedura di creazione avviene in fase di compilazione del programma, il numero dei canali creati non può essere più modificabile, ma l'utilizzo del costrutto for permette in maniera automatica la dichiarazione, l'instanziazione e il mapaggio dei moduli in maniera automatica, ciò che occorre è solamente modificare il valore di `number_of_channels`, definito come `GENERICS`.

```
channel_in: FOR i IN 0 TO number_of_channels - 1 GENERATE
```

```
  coincidence_module:
        time_aware_AND PORT MAP(
                          input_a        => channels_a(i),
                          input_b        => channels_b(i),
                          clock          => clock_50MHz,
                          time_slot_size => coincidence_time_slot_size,
                          output         => coincidence_signal(i)
                        );

  counter: data_counter PORT MAP(
                          clock_1MHz      => clock_1MHz,
                          radiation_wave  => coincidence_signal(i),
                          reset_data      => reset_data,
                          data_request    => data_request,
                          delta_counter   => buffer_counter_array(i)
                        );

END GENERATE;
```

Listato 3

Si è fatto largo uso dei valori `GENERICS`, per quanto possibile, all'interno dell'intero progetto, ciò permette di cambiare i parametri di sistema agendo direttamente sui nomi simbolici, non è necessario intervenire sul codice.

Avendo disponibili al suo interno i dati relativi ad ogni contatore, quello che il modulo fa è semplicemente presentare in uscita, canale per canale, i valori delta_counter e channel.

La procedura si avvia quando arriva un segnale data_request:

- Viene attivata l'uscita transmission_busy, che notifica al modulo UART la presenza di una trasmissione e blocca il modulo data_request dal concederne un'altra durante la durata di quella attuale
- Vengono presentati in output i valori di delta_counter e channel dei contatori presenti all'interno, grazie ad un multiplexer che regola il canale attuale
- I dati in uscita vengono poi inviati al multiplexer in ingresso alla UART che dopo averli trasmessi richiede, tramite un segnale change_channel, di scorrere il numero del canale.

Questa procedura continua fin quando il numero di canali da trasmettere è concluso, viene quindi rilasciata l'uscita `transmission_busy`.

Il modulo fornisce in uscita un segnale, detto ping, che notifica al mux to UART la presenza in uscita di dati da processare. Questo segnale sarà un pettine il cui primo impulso è generato dal data_request, i successivi dal change_channel in uscita dal mux to UART stesso.

Questo blocco appartiene al dominio temporale imposto dalla comunicazione UART, ed è sincrono ad esso.

2.6 Mux to UART

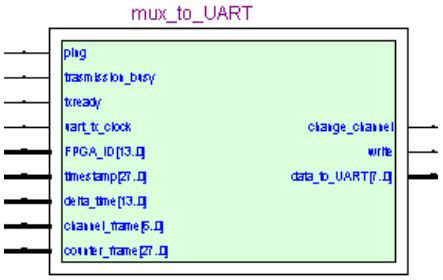


Figura 28

Raccoglie tutti i dati in ingresso e costruisce quelli che saranno i bytes in ingresso al modulo UART.

Ogni frame da trasmetteresarà composto dai seguenti bytes:

- Start Byte: 1 byte convenzionale che delimita l'inizio trasmissione di un nuovo canale, contiene la codifica ASCII a 8 bit del carattere #;
- Delta Counter: 4 bytes che rappresentano il conteggio registrato sul canale dall'invio dati precedente al momento della richiesta attuale;
- Channel: 1 byte che identifica il canale, quindi la fibra corrispondente;
- Delta Time: 2 bytes che rappresentano il tempo tra l'ultimo invio dati e quello attuale;
- Timestamp: 4 bytes che rappresentano un contatore assoluto;
- FPGA ID: 2 bytes, ID univoco dell'FPGA nell'impianto.

Il modulo si occupa di ricevere i dati, in un numero di bit multiplo di 7, e dopo averli impacchettati con un bit di controllo li propone in uscita al serializzatore UART.

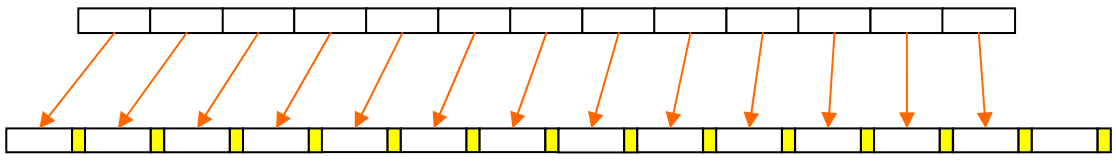


Figura 29

```
-- array 14 righe 8 colonne su cui vengo scritti i bytes da inviare
TYPE byte_n IS ARRAY(0 TO (num_byte_tot - 1)) OF STD_LOGIC_VECTOR(7 DOWNTO
```

```
0);

VARIABLE byte_vector: byte_n;

-- Frame costituito dall'aggregazione dei frame contenenti i dati,
-- viene diviso in 13 blocchi da 7 bit
VARIABLE frame_7bit_x_n:                                        --(90 DOWNTO
0)
                      STD_LOGIC_VECTOR((7 * (num_byte_tot - 1)) - 1 DOWNTO
0);

-- aggrega i frames in entrata
frame_7bit_x_n:= counter_frame & channel_frame & delta_time &
                 timestamp & FPGA_ID;

-- compone le righe dell'array temporaneo, aggiungendo il bit di controllo
byte_vector(13):= frame_7bit_x_n( 6 DOWNTO  0) & '0';  -- ******* + 0
byte_vector(12):= frame_7bit_x_n(13 DOWNTO  7) & '0';  -- ******* + 0
byte_vector(11):= frame_7bit_x_n(20 DOWNTO 14) & '0';  -- ******* + 0
byte_vector(10):= frame_7bit_x_n(27 DOWNTO 21) & '0';  -- ******* + 0
byte_vector(9) := frame_7bit_x_n(34 DOWNTO 28) & '0';  -- ******* + 0
byte_vector(8) := frame_7bit_x_n(41 DOWNTO 35) & '0';  -- ******* + 0
byte_vector(7) := frame_7bit_x_n(48 DOWNTO 42) & '0';  -- ******* + 0
byte_vector(6) := frame_7bit_x_n(55 DOWNTO 49) & '0';  -- ******* + 0
byte_vector(5) := frame_7bit_x_n(62 DOWNTO 56) & '0';  -- ******* + 0
byte_vector(4) := frame_7bit_x_n(69 DOWNTO 63) & '0';  -- ******* + 0
byte_vector(3) := frame_7bit_x_n(76 DOWNTO 70) & '0';  -- ******* + 0
byte_vector(2) := frame_7bit_x_n(83 DOWNTO 77) & '0';  -- ******* + 0
byte_vector(1) := frame_7bit_x_n(90 DOWNTO 84) & '0';  -- ******* + 0
byte_vector(0) := start_frame;                         -- 00100011 -- #
```

Listato 4

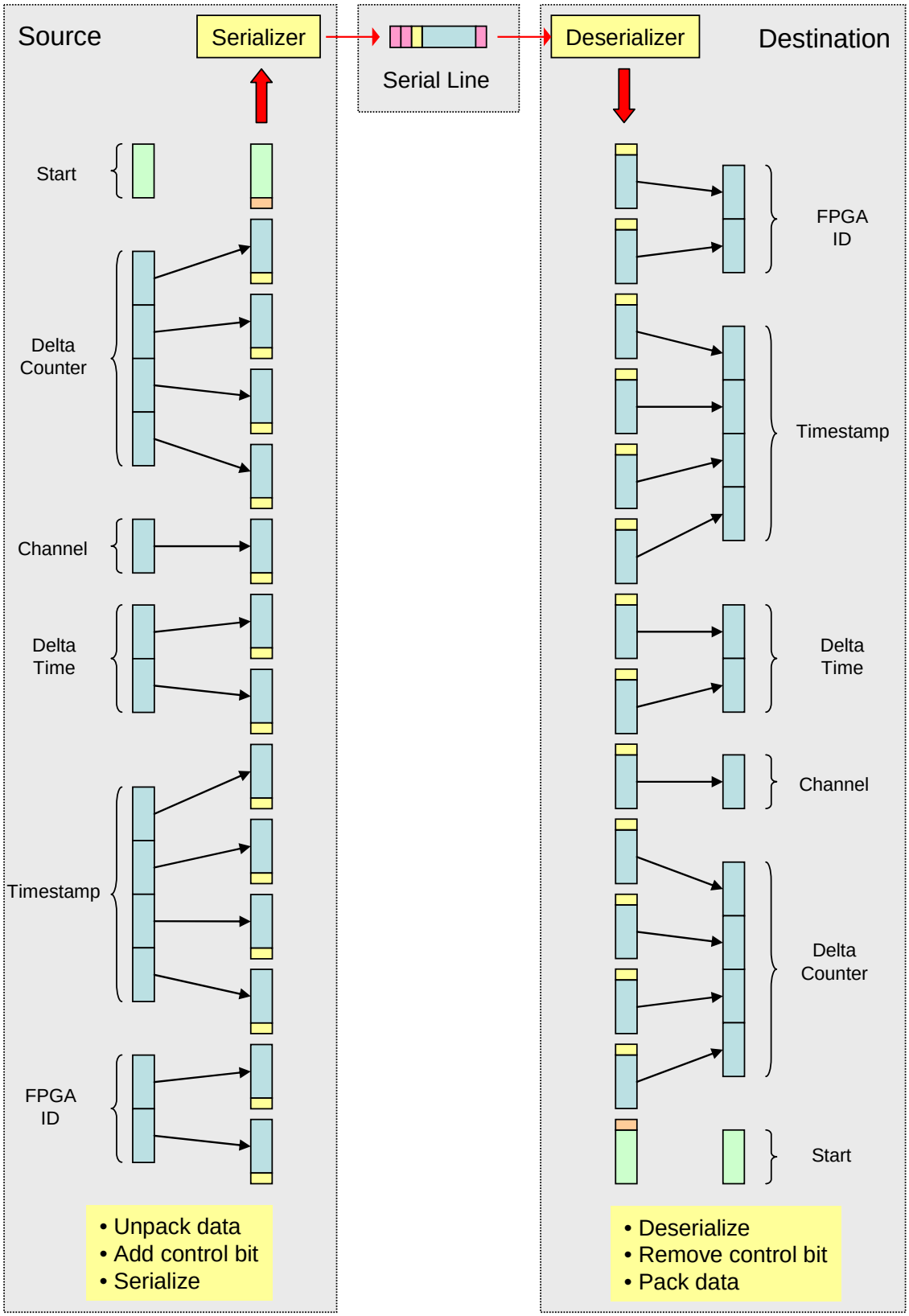


Figura 30

Serializzazione/Deserializzazione dati

3 Simulazioni

3.1 clock_divider_24MHz_to_1kHz

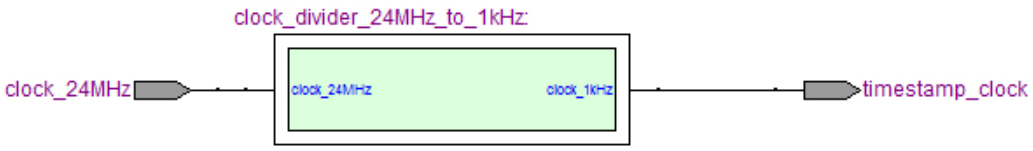


Figura 31

Il modulo opera da divisore di frequenza: dato un clock in entrata, a frequenza conosciuta, genera in uscita un clock a frequenza più bassa, in questo caso particolare si scala da una frequenza pari a 24 MHz ad una pari a 1 kHz.
Ciò viene reso possibile dal funzionamento del modulo che, ogni 24000 eventi di clock in input, genera un evento di clock in uscita: nel caso particolare, il clock in uscita avrà duty cycle pari a 50% quindi verrà imposto un aggiornamento dell'uscita ogni metà periodo, pari a 12000 eventi in ingresso.

Il testbench file presenta un solo `PROCESS` in cui viene simulato lo stimolo ad opera del clock a 24 MHz in entrata: il periodo di clock è pari a 4.16 ns, quindi viene simulato un fronte di salita ed uno di discesa ogni metà periodo.

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;
USE ieee.std_logic_arith.all;
USE ieee.std_logic_unsigned.all;

ENTITY clock_divider_24MHz_to_1kHz_tb IS
END    clock_divider_24MHz_to_1kHz_tb;

ARCHITECTURE sim OF clock_divider_24MHz_to_1kHz_tb IS
```

```
  COMPONENT clock_divider_24MHz_to_1kHz IS

    PORT(
      clock_24MHz : IN  STD_LOGIC;
        clock_1kHz  : OUT STD_LOGIC
    );

  END COMPONENT;

  SIGNAL clock_24MHz : STD_LOGIC ;
  SIGNAL clock_1kHz  : STD_LOGIC;

  BEGIN

    clock_sim : clock_divider_24MHz_to_1kHz
                PORT MAP(clock_24MHz,
                         clock_1kHz);

    clock:
    PROCESS
        BEGIN

        clock_24MHz <= '1';
        wait for 20 ns;
        wait for 833 ps;

        clock_24MHz <= '0';
        wait for 20 ns;
        wait for 833 ps;

    END PROCESS clock;

END sim;
```

Listato 5

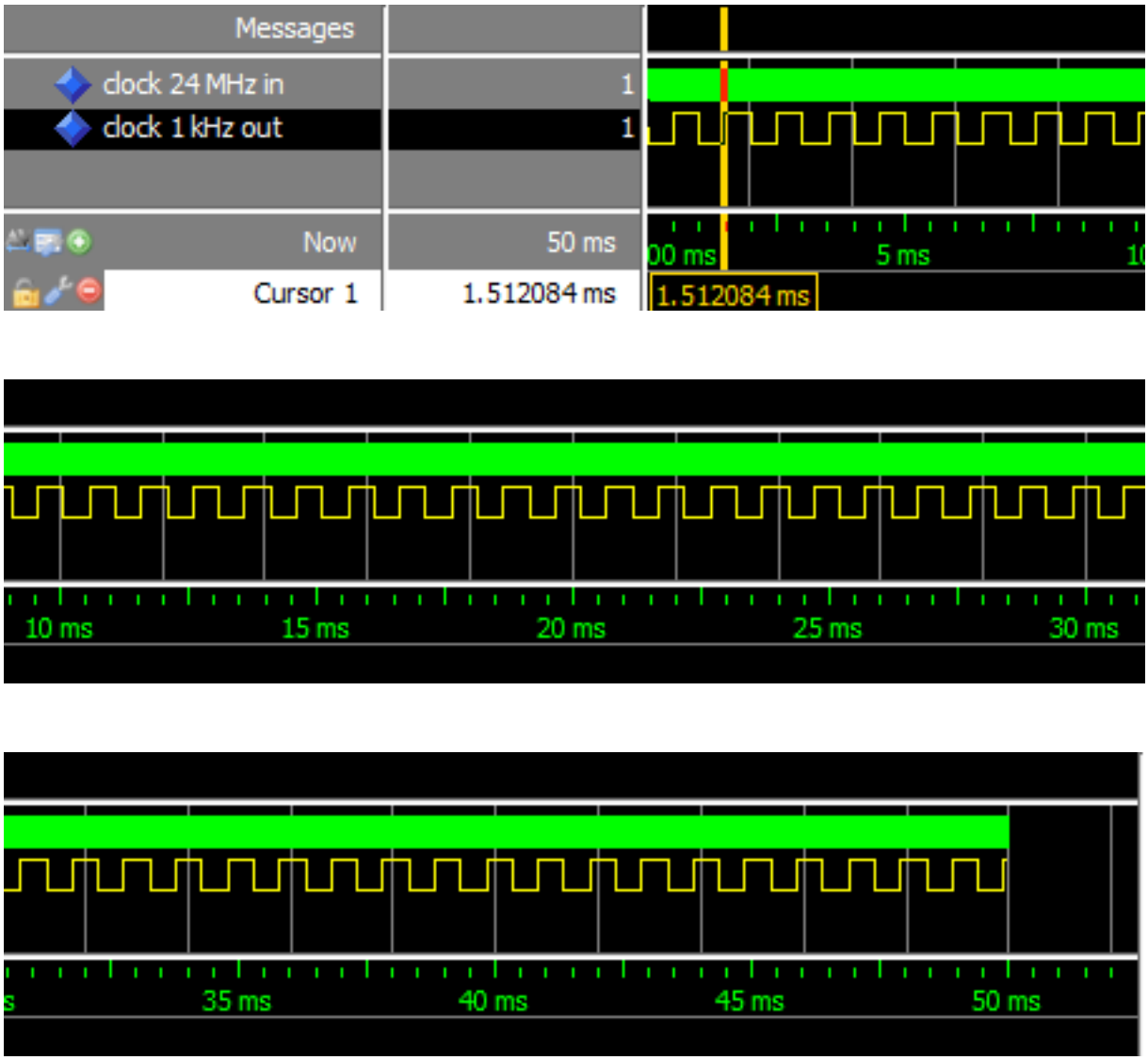


Figura 32

Risultato della simulazione

3.2 *programmable_baud_generator_from_18dot432_MHz*

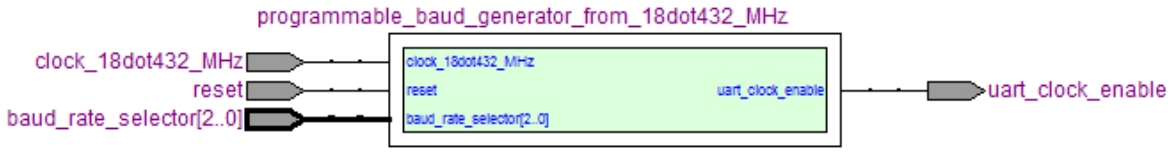


Figura 33

Il modulo accetta in ingresso un clock a 18.432 MHz, un segnale di reset e un vettore, che indica la velocità di baud rate desiderata in uscita; l'uscita sarà un impulso di enable per i moduli che operano nella regione temporale della segnalazione UART.

Il testbench file presenta un `PROCESS` che simula il clock in input a 18.432 MHz, un secondo `PROCESS` simula gli stimoli al modulo: un segnale di reset e la variazione della baudrate richiesta in uscita, raddoppiando ogni 150 µs dal valore 1200 bps fino al valore 115200 bps.

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;
USE ieee.std_logic_arith.all;
USE ieee.std_logic_unsigned.all;

ENTITY programmable_baud_generator_from_18dot432_MHz_tb IS
END programmable_baud_generator_from_18dot432_MHz_tb;

ARCHITECTURE sim OF programmable_baud_generator_from_18dot432_MHz_tb IS

    COMPONENT programmable_baud_generator_from_18dot432_MHz IS

        PORT(
          clock_18dot432_MHz : IN  STD_LOGIC;
          baud_rate_selector : IN  STD_LOGIC_VECTOR(2 DOWNTO 0);
          reset              : IN  STD_LOGIC;
          uart_clock_enable  : OUT STD_LOGIC
        );

    END COMPONENT;
```

```
SIGNAL clock_18dot432_MHz : STD_LOGIC ;
SIGNAL baud_rate_selector : STD_LOGIC_VECTOR(2 DOWNTO 0);
SIGNAL reset              : STD_LOGIC;
SIGNAL uart_clock_enable  : STD_LOGIC;

BEGIN

  clock_sim : programmable_baud_generator_from_18dot432_MHz
              PORT MAP(clock_18dot432_MHz,
                       baud_rate_selector,
                       reset,
                       uart_clock_enable);

clock:
PROCESS
  BEGIN

      clock_18dot432_MHz <= '1';
      wait for 27 ns;
      wait for 253 ps;

      clock_18dot432_MHz <= '0';
      wait for 27 ns;

END PROCESS clock;

stimulus:
PROCESS
  BEGIN

      reset <= '0';
      wait for 5 ns;

      reset <= '1';
      wait for 10 ns;

      reset <= '0';
      baud_rate_selector<="010"; --   1200 bps
      wait for 150 us;

      baud_rate_selector<="011"; --   2400 bps
```

```
          wait for 150 us;

          baud_rate_selector<="001"; --   4800 bps
          wait for 150 us;

          baud_rate_selector<="000"; --   9600 bps
          wait for 150 us;

          baud_rate_selector<="100"; --  19200 bps
          wait for 150 us;

          baud_rate_selector<="101"; --  38400 bps
          wait for 150 us;

          baud_rate_selector<="111"; --  57600 bps
          wait for 150 us;

          baud_rate_selector<="110"; -- 115200 bps
          wait for 150 us;

      END PROCESS stimulus;

END sim;
```

Listato 6

Figura 34

Risultato della simulazione

4 Formato dati trasmessi

Per ogni canale presente, che corrisponde alla coincidenza dei segnali dei SiPM posti alle estremità di una fibra, l'FPGA compone un frame seriale RS-232 contenente le seguenti informazioni:

- Start Byte: 1 byte convenzionale che delimita l'inizio della trasmissione di un nuovo canale e contiene la codifica ASCII a 8 bit del carattere `#`;
- Delta Counter: 4 bytes che rappresentano il conteggio registrato sul canale dall'invio dati precedente al momento della richiesta attuale;
- Channel: 1 byte che identifica il canale, quindi la fibra corrispondente;
- Delta Time: 2 bytes che rappresentano il tempo tra l'ultimo invio dati e quello attuale;
- Timestamp: 4 bytes che rappresentano un contatore assoluto;
- FPGA ID: 2 bytes, ID univoco dell'FPGA.

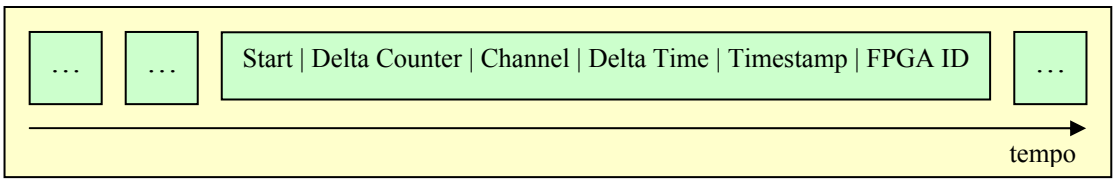


Figura 35

Composizione frame seriale

I dati sono inviati seguendo una tecnica di rilevazione errori sul singolo Byte, poichè si segue la seguente convenzione per il bit meno significativo:

- posto a 1 per lo Start Byte (`#`, `0x23`, `00100011`),
- posto a 0 per tutti gli altri bytes (`11110110000000101101000001000110`…),

questa tecnica permette a destinazione di verificare, al momento della lettura dello stream seriale in ingresso dalla porta di comunicazione, che le informazioni da analizzare sono state ricevute nel modo corretto, secondo la convenzione.

Nel caso di perdita di un byte sarà possibile dunque scartare i successivi (che potrebbero alterare l'analisi dei dati) fino al successivo byte pari a `#` (che è univocamente il byte di inizio frame perché tutti gli altri non possono terminare per `1`).
Questa tecnica permette anche di rilevare se il trasmettitore e il ricevitore sono effettivamente concordi sulla velocità di segnalazione (bit rate), in quanto un disallineamento provoca errori nei successivi bit di controllo che devono essere ricevuti tutti come `0`.

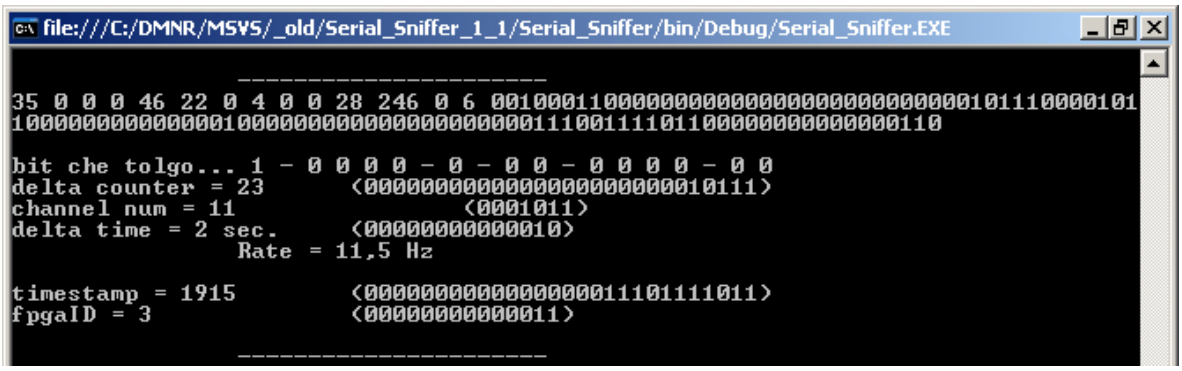


Figura 36

Lettura semplice di un frame seriale.

In alto, il frame ricevuto rappresentato sia in forma decimale che binaria.

A seguire l'estrazione dei bit di controllo e rappresentazione dei dati

4.1 Controllo integrità dati

E' possibile inoltre inserire un ulteriore byte per il controllo di integrità sui dati inviati.
Il **cyclic redundancy check** [6] è un metodo per il calcolo di somme di controllo (*checksum*).
Il nome deriva dal fatto che i dati d'uscita sono ottenuti elaborando i dati di ingresso i quali vengono fatti scorrere ciclicamente in una rete logica.
Il controllo CRC è molto diffuso perché la sua implementazione binaria è semplice da realizzare, richiede conoscenze matematiche modeste per la stima degli errori e si presta bene a rilevare errori di trasmissione su linee affette da elevato rumore di fondo.
Utile per l'individuazione degli errori casuali, il CRC non è invece affidabile per verificare la completa correttezza dei dati contro tentativi intenzionali di manomissione.

[6] Controllo a ridondanza ciclica, il cui acronimo CRC è ben più diffuso.

Il CRC prevede la generazione di una stringa di bit di controllo, che viene normalmente trasmessa assieme ai dati, e il calcolo è basato sull'aritmetica modulare. Un codice CRC è definito dal suo polinomio generatore, volendo privilegiare la velocità di calcolo e la semplicità di trasmissione e controllo, si può optare per una soluzione semplice, generando un byte di controllo (utilizzando l'operatore dell'algebra booleana XOR) da inviare in coda ad ogni frame trasmesso.

L'operatore XOR, detto anche OR esclusivo o somma modulo 2, restituisce 1 se e solo se la somma degli operandi uguali ad 1 è dispari, mentre restituisce 0 in tutti gli altri casi. Di seguito la tabella delle verità:

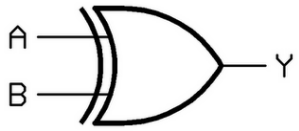


Figura 37

Tabella 4

A	B	Y
0	0	0
0	1	1
1	0	1
1	1	0

Supponiamo di voler trasmettere su un canale non affidabile tre bytes e un byte di controllo, ad esempio i caratteri ASCII #, H e T: il carattere risultante dal prodotto XOR dei tre bytes sarà ? e verrà accodato alla trasmissione.

Tabella 5

ASCII	Hex	bin
#	Ox23	00100011
H	Ox48	01001000
T	Ox54	01010100
# **XOR** H **XOR** T **=** ?	Ox3F	00111111

Il ricevitore, a destinazione, potrà eseguire localmente il calcolo del byte di controllo sui tre bytes ricevuti, confrontandolo con quello appena ricevuto: se corrispondono la probabilità che la i dati trasmessi siano stati ricevuti senza subire alterazioni durante il percorso è alta.

Non è tuttavia possibile con questo semplice sistema giudicare l'effettiva bontà dei dati trasmessi, a causa del funzionamento intrinseco dell'operatore XOR.

Supponiamo ad esempio che durante il tragitto si sia avuto un deterioramento sul secondo bit meno significativo del secondo byte: il carattere H trasmesso verrebbe letto come J. A destinazione verrebbe subito rilevato l'errore perché il byte di controllo calcolato localmente non sarebbe più uguale a quello ricevuto.

Tabella 6

Trasmissione			Ricezione		
ASCII	Hex	bin	ASCII	Hex	bin
#	Ox23	00100011	#	Ox23	00100011
H	Ox48	01001000	**J**	**Ox4A**	0100101**0**
T	Ox54	01010100	T	Ox54	01010100
# **XOR** H **XOR** T **=** ?	Ox3F	00111111	# **XOR** H **XOR** T **=** ?	Ox3F	00111111
			# **XOR J XOR** T **= <**	**Ox3C**	0011110**1**

Se accadesse che l'errore di trasmissione fosse presente su due bytes, localizzato sullo stesso bit, allora i dati ricevuti non sarebbero giudicati errati, poiché il byte di controllo corrisponderebbe a quello che avrebbero generato i bytes esatti.

Tabella 7

Trasmissione			Ricezione		
ASCII	Hex	bin	ASCII	Hex	bin
#	Ox23	00100011	#	Ox23	00100011
H	Ox48	01001000	**J**	**Ox4A**	0100101**0**
T	Ox54	01010100	**V**	**Ox56**	0101011**0**
# **XOR** H **XOR** T **=** ?	Ox3F	00111111	# **XOR** H **XOR** T **=** ?	Ox3F	00111111
			# **XOR J XOR V =** ?	Ox3F	00111111

Affinchè si verifichi questa situazione è necessario che l'evento di corruzione di un determinato bit (ad esempio il bit meno significativo di un byte) si verifichi un numero pari di volte, la probabilità congiunta può essere calcolata con le seguenti assunzioni:

- Sia p la probabilità di errore sul canale trasmissivo di un bit
- La probabilità che il bit sia ricevuto correttamente sarà 1-p

- Siano m i bytes, inviati per ogni trasmissione, coinvolti nel calcolo del byte di controllo

La probabilità congiunta che un determinato bit sia ricevuto errato n volte sarà data dalla formula

$$P_{err(n)} = (p)^n * (1-p)^{m-n}$$

Il grafico seguente mostra, in scala logaritmica, l'andamento per due valori di p: 1% e 10% (posto m pari a 14)

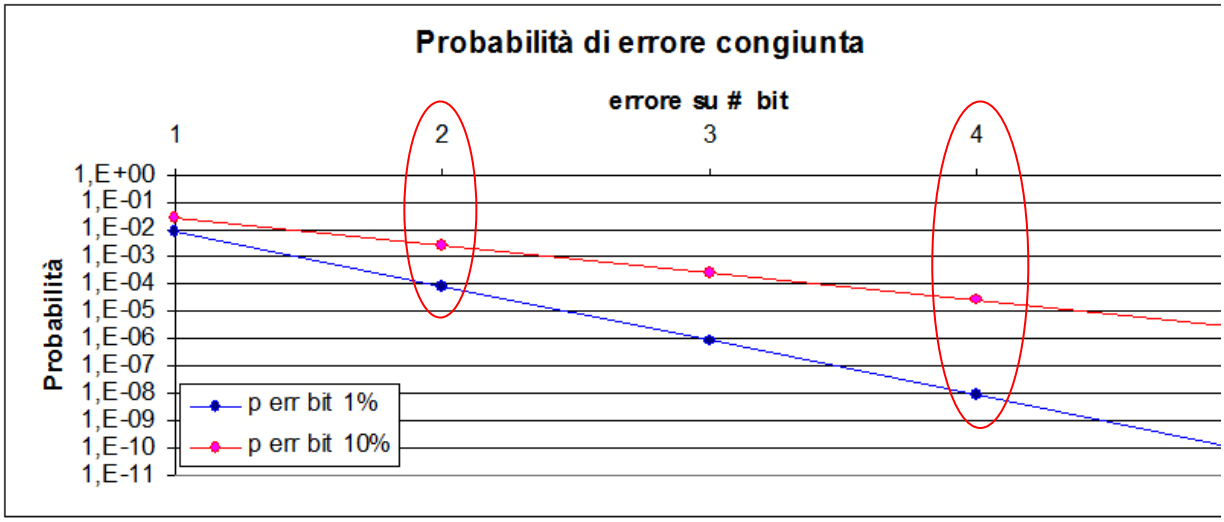


Figura 38

Dal grafico è possibile valutare come una sequenza di eventi, tale per cui si avrebbe a destinazione una ricezione dati errata ritenuta corretta, risulti statisticamente possibile, tuttavia poco probabile.

I valori significativi per il caso in esame sono localizzati sui numeri di bit in errore pari (2, 4).

Può inoltre anche accadere che l'errore sia localizzato su diversi bit dei bytes coinvolti, ad esempio la situazione seguente porterebbe ad un errore a ricezione:

Tabella 8

Trasmissione			Ricezione		
ASCII	Hex	bin	ASCII	Hex	bin
#	Ox23	00100011	+	**0x2B**	0010**1**011
H	Ox48	01001000	J	**0x4A**	0100101**0**
T	Ox54	01010100	^	**0x5E**	0101**11**10
# **XOR** H **XOR** T **=** ?	Ox3F	00111111	# **XOR** H **XOR** T **=** ?	Ox3F	00111111
			+ XOR J XOR ^ = ?	Ox3F	00111111

In questo caso la probabilità sarebbe pari a $P_{err(2,2)} = \left((p)^2 * (1-p)^{m-2}\right)^2$ poiché coinvolge 2 bit errati su due posizioni diverse, il risultato è un valore di probabilità molto piccolo.

Tabella 9

p	m	$P_{err(2)}$	$P_{err(2,2)}$
1%	14	8.78E-05	7.70E-09
10%	14	2.54E-03	6.46E-06

MESH

I capitoli seguenti mostreranno le scelte progettuali che hanno portato alla creazione dell'interfaccia grafica di presentazione dati e monitoraggio.

L'ambiente di sviluppo scelto è stata la piattaforma Microsoft .NET 3.5, linguaggio di programmazione utilizzato il C#, giovane e potente linguaggio che ha permesso, nella piena filosofia della programmazione ad oggetti, di creare strutture dati astratte, riusabili e personalizzate.

Poiché l'ambiente che si vuole monitorare risulta essere fortemente gerarchico, la struttura degli oggetti che rappresenta l'impianto rispecchia tale configurazione, in particolare si è scelto di considerare i singoli sensori di radiazione quali fonte puntuale di dati.

Le informazioni statistiche di interesse vengono aggregate crescendo di livello gerarchico, in modo da avere una visione globale e al contempo dettagliata dello stato attuale.

La ricezione dei dati, trasmessi dai dispositivi dislocati nell'impianto da monitorare, avviene tramite linea di collegamento seriale punto-punto; tale scelta ha portato a semplificare notevolmente lo sviluppo prototipale, della console di acquisizione dati, piuttosto che concentrare l'attenzione sulla modalità di trasmissione.

Tuttavia, il sistema, essendo stato progettato con la volontà di poter cambiare il protocollo di trasmissione, si presta facilmente a tale modifica; sarà sufficiente rimpiazzare il modulo di ricezione dati con uno conforme ad un altro protocollo, secondo le esigenze dettate dalla realizzazione pratica di un sistema distribuito.

Definite la struttura dati del programma e la modalità di ricezione delle informazioni, si è giunti alla creazione dell'interfaccia grafica, con tecnologia WPF.

Il prodotto finale ha attraversato molti stati intermedi, partendo da un semplice sniffer seriale in modalità testuale, passando poi per numerose produzioni di test, delle modalità ottimali per la presentazione dati in forma semplice ma dettagliata e delle tecniche di propagazione degli eventi, per giungere, infine, ad un prodotto maturo, che presenta attraverso un'interfaccia minimale e di semplice utilizzo tutti i dati e i comandi di interesse all'operatore.

I parametri di configurazione del programma sono letti da diversi files esterni in formato XML, tale soluzione permette di modificare il comportamento del programma stesso (definizione allarmi e azioni da eseguire, impostazioni sulla modalità di trasferimento dati, struttura dati di riferimento) senza che sia necessaria la ricompilazione.

Lo storico dei dati ricevuti è stato implementato in modalità differita, rispetto alla visualizzazione delle informazioni, poiché le esigenze dettate da sicurezza, integrità e non corruzione di dati sensibili, come il monitoraggio di depositi di rifiuti radioattivi, impongono sicuramente grande attenzione e cura in fase progettuale.

Tutti i dati ricevuti vengono dunque registrati su un file di log al quale si accede per eseguire la post-elaborazione statistica e l'aggiornamento del Data Base che contiene lo storico delle informazioni.

E' attualmente in fase di sviluppo avanzato un sistema di accesso a tali informazioni da postazioni remote, anche dispositivi mobili (smartphones), grazie all'utilizzo di tecnologie orientate alla rete internet, i web services.

5 *Gerarchia degli elementi*

La struttura di riferimento dell'impianto di stoccaggio segue la configurazione come illustrata nella Figura 39, nella quale quattro bidoni per volta vengono raggruppati in gabbie ed impilati gli uni sugli altri.

L'elemento base per la rivelazione delle radiazioni è composto dall'unione di una fibra scintillante e dai SiPM posti alle sue estremità; un elemento bidone sarà composto da una serie di fibre poste attorno ad esso e un box sarà una struttura che contiene quattro bidoni. E' possibile impilare i box in matrici tridimensionali, detti gruppi e considerare l'intero impianto come una collezione di gruppi.

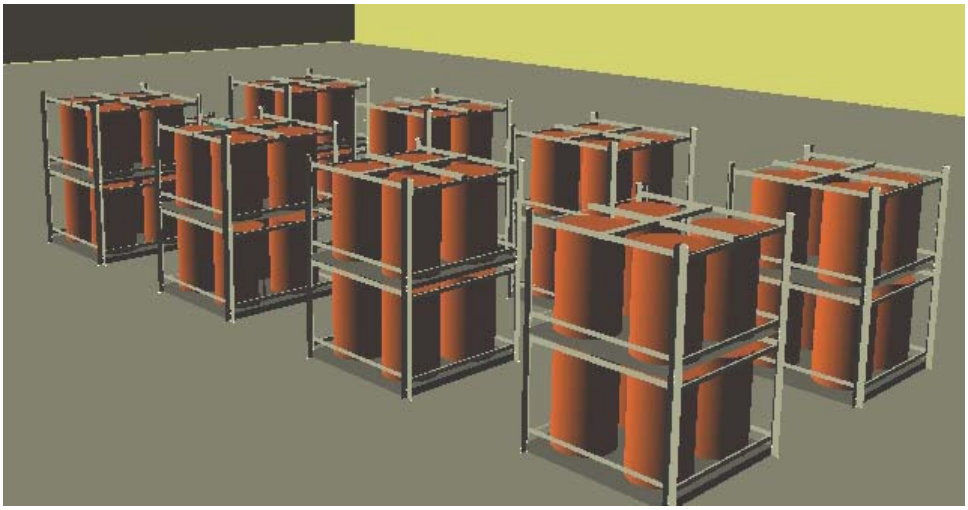


Figura 39
Simulazione 3D dell'impianto di stoccaggio

Poiché ogni singolo elemento, dal più semplice (la singola fibra) al più articolato (il gruppo è un contenitore di box che contiene quattro bidoni i quali, a loro volta, possiedono un numero definito di fibre), presenta delle caratteristiche comuni che vogliono essere visualizzate e registrate, si è scelto di sfruttare al meglio il polimorfismo[7] per definire la gerarchia delle classi.

[7] Il polimorfismo è un'importante proprietà dei linguaggi ad oggetti: attesta la possibilità di utilizzare un oggetto al posto di un altro, laddove esista parentela tra i due.

Come mostrato in Figura 40 tutti gli elementi sono una specializzazione della classe astratta `WasteItem`, in particolare ogni oggetto eredita le proprietà comuni a tutti i `WasteItem` e contiene un riferimento alle istanze degli oggetti figli. Questa configurazione permette, in maniera immediata, di organizzare una struttura gerarchica in cui un oggetto è contenitore di altri oggetti ad esso simili, ma qualificati in maniera differente.

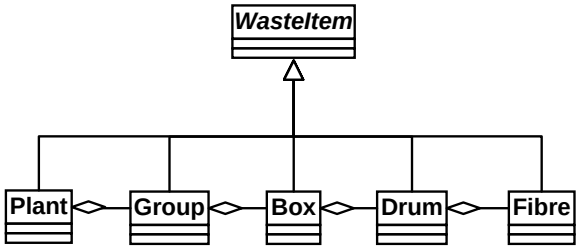


Figura 40

Struttura classi

La struttura ad albero che si viene a creare gode della proprietà che ogni elemento possiede sia il riferimento ai figli in esso contenuti, sia il riferimento al padre che lo contiene. Questa struttura permette la propagazione di un evento dal basso (fibra) a tutti i contenitori di gerarchia superiore, fino al nodo radice (deposito), oltre che il raggiungimento di qualsiasi elemento dall'alto verso il basso.

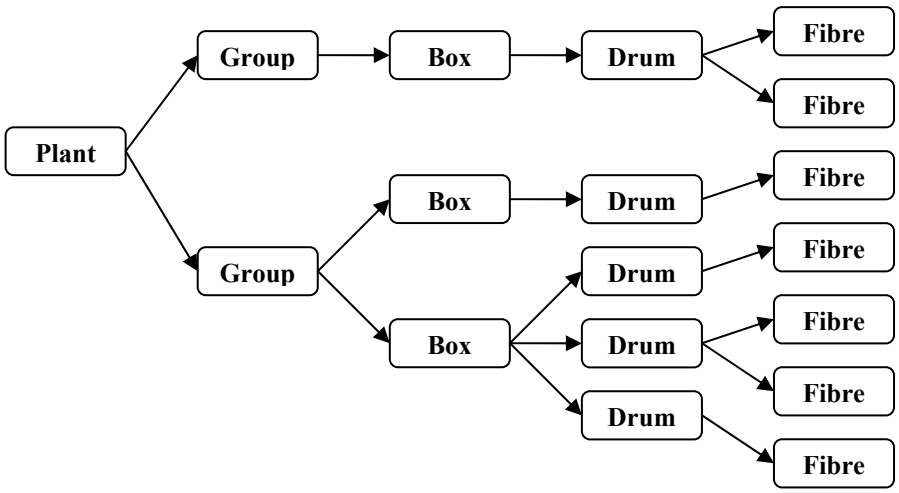


Figura 41

Struttura gerarchica ad albero degli elementi

5.1 *Classe WasteItem*

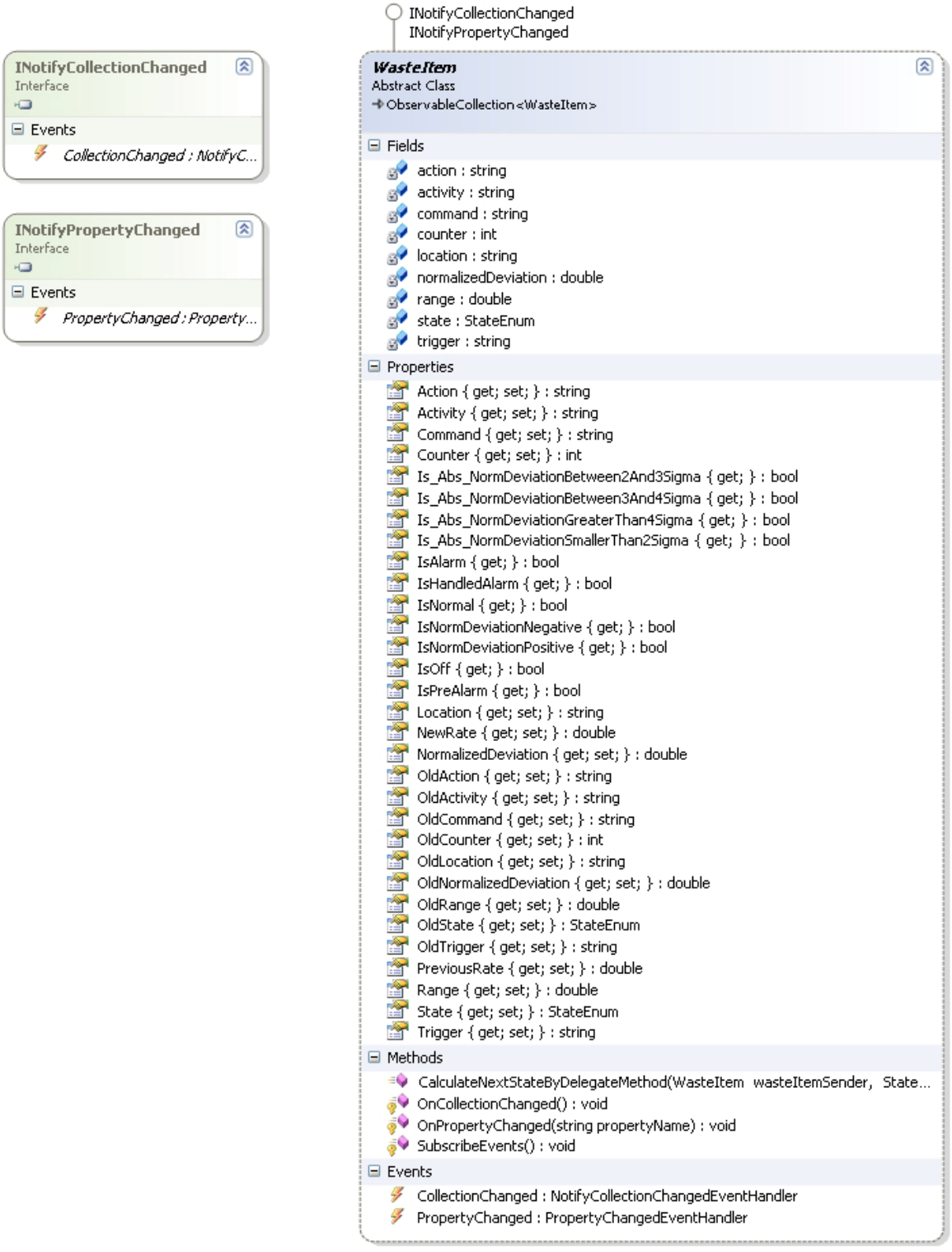


Figura 42

Classe WasteItem

Deriva dalla classe generica `ObservableCollection<T>`, che a sua volta è una derivazione di `Collection<T>`, e implementa le interfacce di sistema per la notifica eventi `INotifyPropertyChanged` e `INotifyCollectionChanged`.
Queste interfacce definiscono i due eventi che verranno "ascoltati" dal programma di monitoraggio:

- `PropertyChanged`, invocato quando una proprietà cambia valore;
- `CollectionChanged`, invocato quando cambia la struttura degli elementi, ad esempio a seguito di una creazione o rimozione di un oggetto.

```
#region INotifyPropertyChanged Members

    public new event PropertyChangedEventHandler PropertyChanged;

    protected virtual void OnPropertyChanged(string propertyName) {
        if (this.PropertyChanged != null) {
            this.PropertyChanged(this,
                                 new PropertyChangedEventArgs(propertyName));
        }
    }
#endregion

#region INotifyCollectionChanged Members
    public new event NotifyCollectionChangedEventHandler

CollectionChanged;
        protected void OnCollectionChanged() {
            if (this.CollectionChanged != null) {
                this.CollectionChanged(this,
                                 new NotifyCollectionChangedEventArgs
  (System.Collections.Specialized.NotifyCollectionChangedAction.Reset));
            }
    }
#endregion
```

Listato 7

Le informazioni elaborate di alto livello sono salvate sia nel valore attuale e che in quello precedente e sono così definite:

- `Action`;
- `Activity`;
- `Command`;
- `Counter`;
- `Location`;
- `NormalizedDeviation`;
- `Range`;

- `State`;
- `Trigger`.

Il Listato 8 mostra la dichiarazione comune a tutte loro.

```
#region Normalized Deviation

    double normalizedDeviation;

    public double NormalizedDeviation {
        get { return this.normalizedDeviation; }
        set {
            if (value != this.normalizedDeviation) {
                this.OldNormalizedDeviation = this.normalizedDeviation;
                this.normalizedDeviation = value;
                this.OnPropertyChanged("NormalizedDeviation");
                this.OnPropertyChanged("OldNormalizedDeviation");
            }
        }
    }

    public double OldNormalizedDeviation { private set; get; }

#endregion
```

Listato 8

`WasteItem` contiene inoltre le proprietà di tipo booleano alle quali si accede per conoscere lo stato dell'oggetto e la fascia di valore a cui appartiene il valore attuale della deviazione standard normalizzata.

```
#region Bool Properties

    public bool IsNormal       { ... }
    public bool IsPreAlarm     { ... }
    public bool IsAlarm        { ... }
    public bool IsHandledAlarm { ... }
    public bool IsOff          { ... }

    public bool IsNormDeviationPositive               { ... }
    public bool IsNormDeviationNegative               { ... }
    public bool Is_Abs_NormDeviationSmallerThan2Sigma { ... }
    public bool Is_Abs_NormDeviationBetween2And3Sigma { ... }
    public bool Is_Abs_NormDeviationBetween3And4Sigma { ... }
    public bool Is_Abs_NormDeviationGreaterThan4Sigma { ... }

#endregion
```

Listato 9

Infine il metodo

```
public void
    CalculateNextStateByDelegateMethod(WasteItem wasteItemSender,
                                       StateTransitionHandler delegateMethod)

    { wasteItemSender.State = delegateMethod(wasteItemSender); }
```

Listato 10

per determinare la transizione di stato, la cui chiamata sarà mostrata dopo aver chiarito la procedura per la creazione dei metodi delegati (Listato 49).

5.2 Classe Fibre

Ciascuna fibra viene instanziata dal bidone che la conterrà, il quale si occuperà di passare il proprio riferimento per il campo `ParentDrum`.

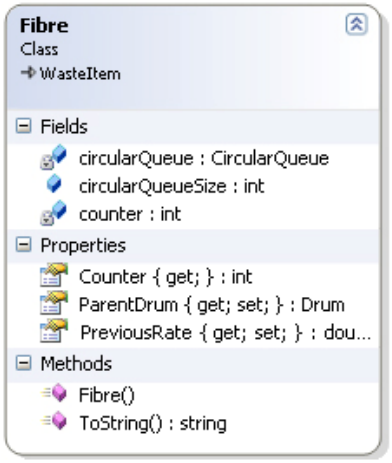


Figura 43

Classe Fibre

L'oggetto contiene un contatore statico di istanze, utile per risalire al numero totale di fibre create e un buffer circolare, `circularQueue` (in Appendice, Listato 72), che viene sostituito al membro `PreviousRate` della classe astratta madre.

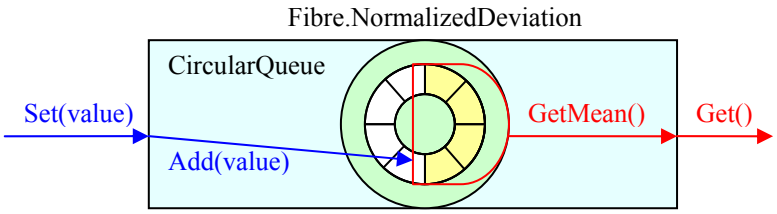


Figura 44

Incapsulamento del fuffer CircularQueue

Il buffer, di dimensione impostabile permette di registrare le ultime occorrenze del dato `PreviousRate` e di prendere la media dei valori inseriti, ciò è utile nel calcolo della deviazione standard normalizzata perché vengono smussate le statistiche in occorrenza degli eventi rari.

```
private static int counter;
public new int Counter { get { return counter; } }

//Bidone a cui appartiene la fibra
public Drum ParentDrum { set; get; }

public int circularQueueSize = 5;
CircularQueue circularQueue;

//Costruttore: incrementa un contatore di istanze attive
public Fibre() {
    counter++;
    circularQueue = new CircularQueue(circularQueueSize);
}

public override double PreviousRate {
    get { return this.circularQueue.GetMean(); }
    set { this.circularQueue.Add(value); }
}
```

Listato 11

5.3 Classe Drum

L'oggetto bidone è un contenitore di fibre che, poste attorno ad esso in senso verticale e longitudinale, formano una griglia (Figura 2). Combinando opportunamente i dati provenienti da ciascuna fibra è possibile ottenere una visione tridimensionale della radiazione emessa dai rifiuti posti dentro esso; ciò permetterebbe di osservare eventuali gradienti di emissione non regolari o rilevare una concentrazione anomale in un punto della superficie esterna, con

precisione pari a metà del passo della griglia, dovuta ad esempio da una lesione della struttura del bidone.

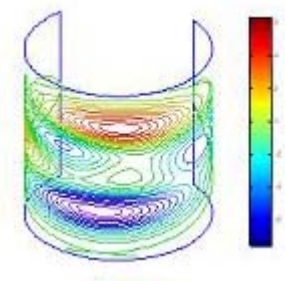


Figura 45

Visione 3D della radiazione emessa da un bidone

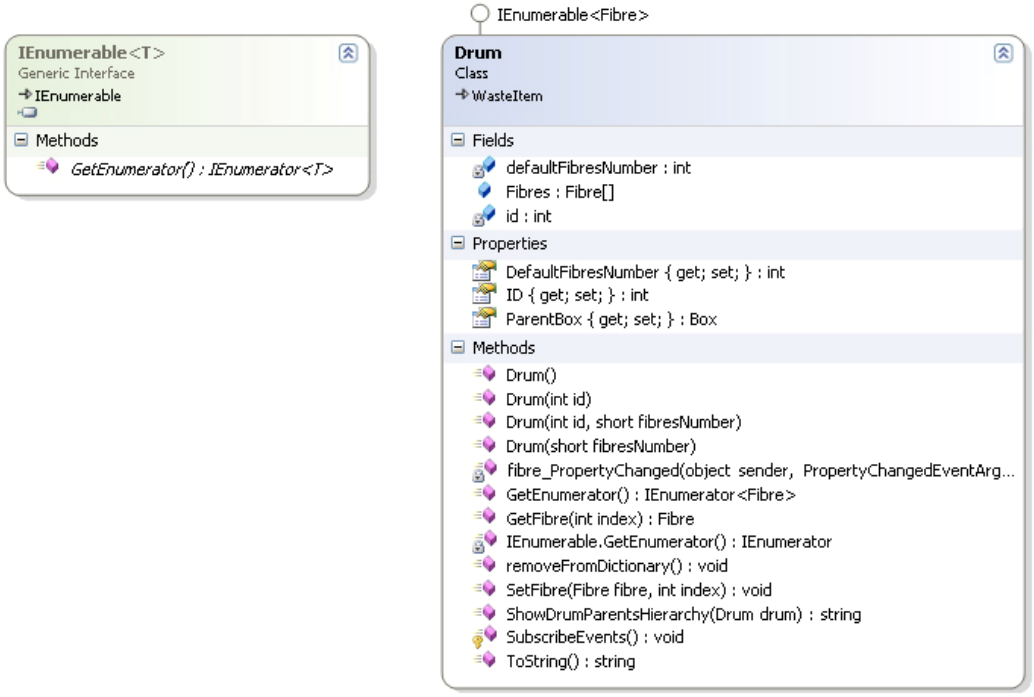


Figura 46

Classe Drum

Gli oggetti `Fibre`, `Drum` e `Box` vengono individuati, in vari punti del programma, utilizzando il comando `foreach` (sull'elemento che li contiene), che permette di iterare gli elementi di una collection senza conoscerne a priori il loro numero.

Il comando `foreach` utilizza a questo scopo un enumeratore; la Figura 47 mostra la relazione tra un client che invoca il metodo `foreach` e la collection. La collection implementa l'interfaccia `IEnumerable` con il metodo `GetEnumerator()`.

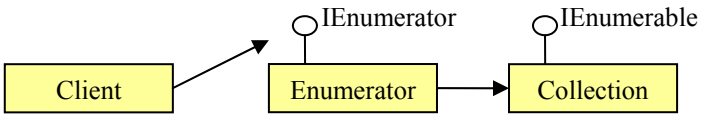


Figura 47

Richiesta di un enumeratore su una collection

La classe `Drum` implementa l'interfaccia `IEnumerable <Fibre>` perchè contiene un array, `Fibres[]`, di oggetti `Fibre`; deve quindi definire il proprio metodo `GetEnumerator()`.

```
#region IEnumerable Members

    IEnumerator IEnumerable.GetEnumerator() {
        return this.GetEnumerator();
    }

    public new IEnumerator<Fibre> GetEnumerator() {
        for (int index = 0; index < this.Fibres.GetLength(0); index++) {
            yield return this.Fibres[index];
        }
    }

#endregion
```

Listato 12

La direttiva `yield return` permette di implementare enumeratori in modo semplice, restituendo un elemento di una collezione e posizionandosi sul successivo quando si itera una collezione attraverso il comando `foreach` [NEGWS08].

Il costruttore (e gli overload) `Drum()`:

- crea un'istanza di un oggetto bidone, caratterizzato da un ID univoco (definito dall'operatore o se non specificato impostato a NONE), che corrisponde all'ID dell'FPGA ad esso corrispondente ;
- crea un array (di dimensione prefissata o a piacere) di `Fibre`, istanziando per ogni elemento un nuovo oggetto `Fibre` e passando il riferimento a se stesso come padre;
- Inserisce il bidone nel `AssignedDrumsDictionary` di `Plant`, una hash table chiave-valore che permette di accedere ad un elemento riferendosi ad una chiave univoca, l'ID del bidone;
- Nel caso in cui il bidone non abbia un ID assegnato viene invece inserito in `NotYetAssignedDrumsList` di `Plant`;

- Sottoscrive, al bidone appena creato, gli eventi di tipo cambiamento di stato che vengono generati dalle fibre in esso contenute.

```
protected override void SubscribeEvents() {
    foreach (Fibre fibre in this.Fibres) {
        fibre.PropertyChanged +=
                new PropertyChangedEventHandler(fibre_PropertyChanged);
    }
}

void fibre_PropertyChanged(object sender, PropertyChangedEventArgs e) {
    if (e.PropertyName.Equals("State")) {
        this.CalculateNextStateByDelegateMethod (this,
     Plant.UniqueInstance.Methods.SimpleParentPolicy_IfAtLeastOneChild);
    }
}
```

Listato 13

L'inserimento dei bidoni nel `AssignedDrumsDictionary` di `Plant` ha una funzione strategica, in quanto permette di individuare direttamente il bidone, e di conseguenza le fibre in esso contenute, semplicemente tenendo in considerazione l'ID trasmesso dall'FPGA.
Ciò facilita notevolmente il raggiungimento dell'oggetto cercato, perché il punto di accesso ai dati è unico e univoco[8]; la gerarchia superiore `Box` - `Group` - `Plant` assume solo utilità di tipo logico e gerarchico, rispecchia la disposizione fisica e geometrica dell'impianto.

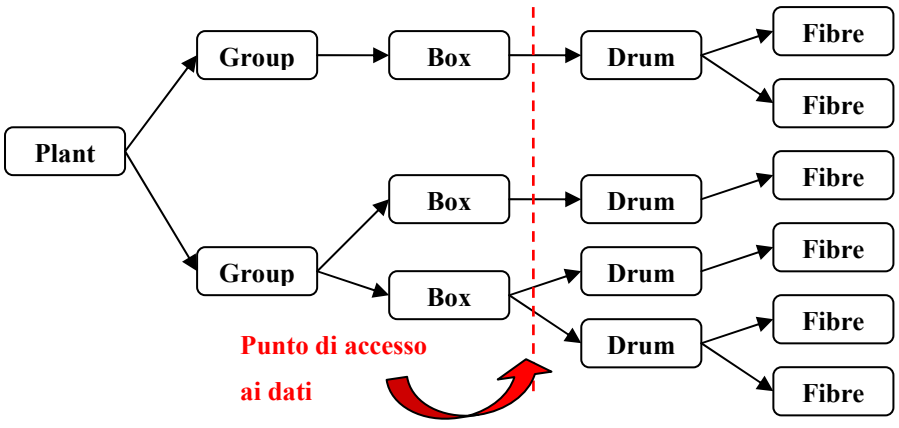


Figura 48

Grazie alla hash table chiave-valore il bidone viene identificato dal suo ID

Di seguito la procedura che si occupa di assegnare l'ID al bidone.

[8] le collections di tipo Dictionary non permettono di duplicare il valore key

```
//ID univoco del bidone, corrisponde all'ID FPGA
private int id;

public int ID {
    get { return this.id; }

    //aggiunge il bidone al dictionary, valore proibito: 0
    set {
      try {
        //Se non esiste già nel dictionary un ID uguale a quello
        //che si vuole impostare...
        if (! Plant.UniqueInstance.AssignedDrumsDictionary.
                                                ContainsKey(value)){
            //Se il bidone aveva già assegnato un altro ID...
            //quindi voglio sovrascrivere l'ID
            if (Plant.UniqueInstance.AssignedDrumsDictionary.
                                                ContainsKey(this.ID)){
                //... rimuovi il record
                Plant.UniqueInstance.AssignedDrumsDictionary.
                                                    Remove(this.ID);
            }
            //assegna quindi il nuovo ID al bidone
            this.id = value;
            //e lo aggiunge al Dictionary
            Plant.UniqueInstance.AssignedDrumsDictionary.
                                                Add(value, this);
        }
        //Controlla se il bidone è ancora presente
        //nella lista dei non assegnati, in tal caso lo rimuove
        if(Plant.UniqueInstance.NotYetAssignedDrumsList.Contains(this))
            Plant.UniqueInstance.NotYetAssignedDrumsList.Remove(this);
      }
      catch (ArgumentNullException) { }
      catch (ArgumentException) { }
    }
}
```

Listato 14

5.4 Classe Box

Ogni box è fisicamente costituito da una gabbia metallica che raggruppa quattro bidoni, viene movimentato da carroponti e posizionato in quella che sarà la sua postazione definitiva nel deposito.

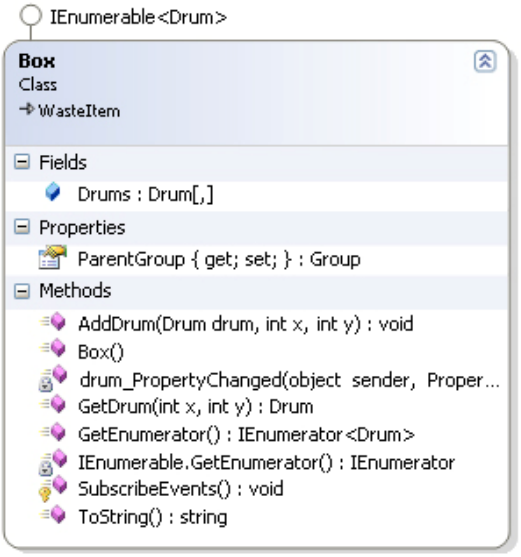


Figura 49

Classe Box

Il costruttore `Box()`:

- crea un array 2 x 2 di bidoni con ID non inizializzato, `Drums[,]`, e passa loro il riferimento a se stesso come padre;
- registra i bidoni appena creati nella lista dei bidoni senza ID, la `NotYetAssignedDrumsList` di `Plant`;
- sottoscrive gli eventi da essi generati.

```
public Box() {
    this.Drums = new Drum[2, 2];
    for (int x = 0; x < Drums.GetLength(0); x++) {
        for (int y = 0; y < Drums.GetLength(1); y++) {
            this.Drums[x, y] = new Drum();
            this.Add(this.Drums[x, y]);
            this.Drums[x, y].ParentBox = this;
            this.Drums[x, y].Location = x + "," + y;
                if(!Plant.UniqueInstance.NotYetAssignedDrumsList.
                                                    Contains(Drums[x,y]))
                    Plant.UniqueInstance.
                                NotYetAssignedDrumsList.Add(Drums[x, y]);
        }
    }
    this.SubscribeEvents();
}
```

Listato 15

Il comportamento di `Box` è analogo a quello di `Drum`, esso sottoscrive gli eventi dei figli che contiene.

```
protected override void SubscribeEvents() {
    foreach (Drum drum in this.Drums) {
        drum.PropertyChanged +=
                  new PropertyChangedEventHandler(drum_PropertyChanged);
    }
}

void drum_PropertyChanged(object sender, PropertyChangedEventArgs e) {
    if (e.PropertyName.Equals("State")) {
        this.CalculateNextStateByDelegateMethod(this,
     Plant.UniqueInstance.Methods.SimpleParentPolicy_IfAtLeastOneChild);
    }
}
```

Listato 16

L'enumerazione di `Box` restituisce i `Drum` in esso contenuti scorrendo l'array una riga per volta.

```
public new IEnumerator<Drum> GetEnumerator() {
    for (int x = 0; x < this.Drums.GetLength(0); x++) {
        for (int y = 0; y < this.Drums.GetLength(1); y++) {
            if (this.Drums[x, y] == null)
                break;
            yield return this.Drums[x, y];
        }
    }
}
```

Listato 17

5.5 Classe Group

Un gruppo è composto da tanti box accatastati, Figura 39, e può contenere al massimo 5x5x5 elementi.

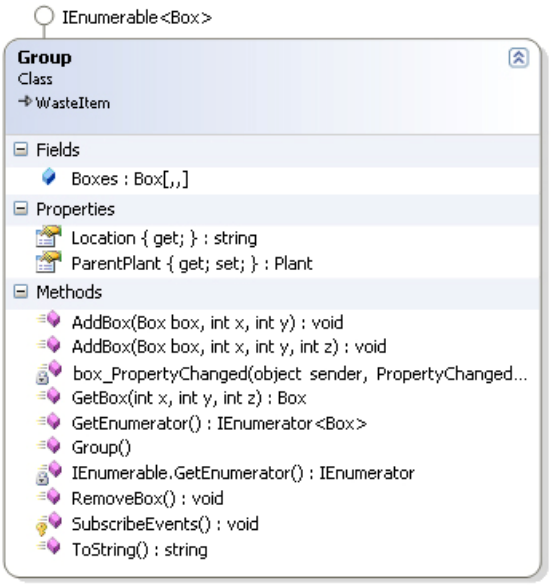


Figura 50

Classe Group

Il costruttore si occupa esclusivamente di creare l'array `Boxes[,,]` e registrare la presenza del gruppo nel deposito, `Plant`.

```
public Group() {
    //riferimento all'impianto
    this.ParentPlant = Plant.UniqueInstance;
    //inserimento nella lista dei gruppi
    Plant.UniqueInstance.GroupsList.Add(this);
    Plant.UniqueInstance.Add(this);
    //creazione della matrice dei box
    this.Boxes = new Box[5, 5, 5];
}
```

Listato 18

Il motivo per cui non vengano creati i box in questa fase si giustifica perché la creazione di un gruppo si limita esclusivamente a riservare in memoria lo spazio necessario ad ospitare 125 box; la loro allocazione reale e l'assegnamento in una determinata coordinata verrà eseguita solamente dopo che un box è stato inizializzato a dovere, quindi dopo aver assegnato ad ogni bidone un ID.

L'assegnazione al gruppo, che equivale allo spostamento meccanico del box sul sito definitivo, viene effettuata attraverso il seguente metodo.

```
public void AddBox(Box box, int x, int y, int z) {
    this.Boxes[x, y, z] = new Box();
    this.Boxes[x, y, z] = box;
```

```
    this.Add(box);
    this.Boxes[x, y, z].ParentGroup = this;
    this.Boxes[x, y, z].Location = x + "," + y + "," + z;
    this.SubscribeEvents();
}
```

Listato 19

Come di consueto viene passato il riferimento dell'oggetto padre al figlio e si effettua la sottoscrizione agli eventi che esso solleva.

L'enumeratore di `Group` itera `Boxes[,,]` riga per riga partendo dal piano a livello 0, per poi passare ai livelli superiori.

```
public new IEnumerator<Box> GetEnumerator() {
    for (int z = 0; z < this.Boxes.GetLength(2); z++) {
        for (int y = 0; y < this.Boxes.GetLength(1); y++) {
            for (int x = 0; x < this.Boxes.GetLength(0); x++) {
                if(this.Boxes[x,y,z] == null)
                    break;
                yield return this.Boxes[x, y, z];
            }
        }
    }
}
```

Listato 20

5.6 Classe Plant

Per fornire un punto di accesso globale all'oggetto `Plant` (che rappresenta l'impianto di stoccaggio dei rifiuti) e assicurare che tale classe abbia una sola istanza, la classe `Plant` è stata implementata come un "Singleton" [GHJV95].

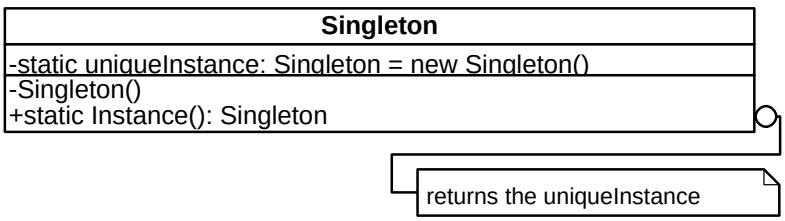


Figura 51

Pattern Singleton

La classe incapsula e regola l'accesso alla sua unica istanza, esponendosi ai client che la richiedono esclusivamente tramite una funzione membro (che si occupa di creare e inizializzare correttamente l'oggetto prima che venga utilizzato per la prima volta).

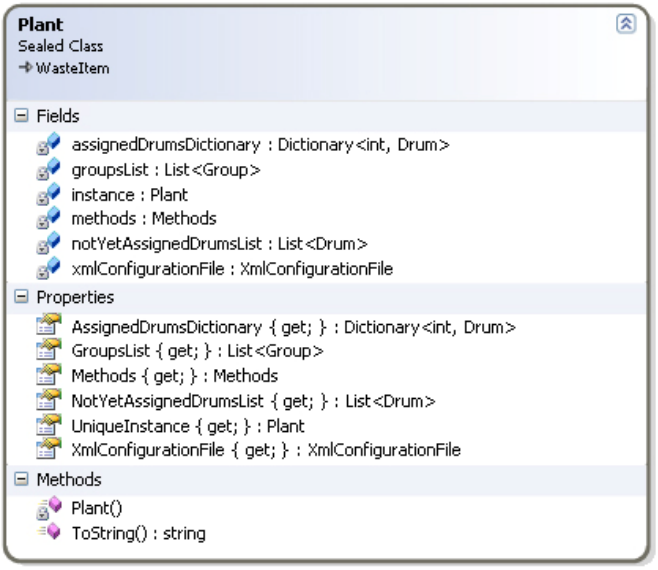


Figura 52

Classe Plant

`Plant` diventa pure unico punto di accesso al file di configurazione XML e alla classe che contiene i metodi per il cambiamento di stato, oltre che al dictionary dei bidoni con ID assegnato e alla lista dei bidoni non ancora assegnati.

```
//Costruttore privato
private Plant() {
    xmlConfigurationFile = new XmlConfigurationFile();
    groupsList = new List<Group>();
    methods = new Methods();
}

//Istanza privata di Plant: è un Singleton
static readonly Plant instance = new Plant();
public static Plant UniqueInstance
    { get { return instance; } }

//Hash table chiave-valore in cui vengono registrati i bidoni
Dictionary<int, Drum> assignedDrumsDictionary =
                                        new Dictionary<int, Drum>();
public Dictionary<int, Drum> AssignedDrumsDictionary
    { get { return assignedDrumsDictionary; } }

//Lista dei bidoni con ID non ancora assegnato
List<Drum> notYetAssignedDrumsList = new List<Drum>();
public List<Drum> NotYetAssignedDrumsList
    { get { return notYetAssignedDrumsList; } }
```

```
//Lista dei gruppi presenti nell'impianto
List<Group> groupsList;
public List<Group> GroupsList
    { get { return groupsList; } }

//Istanza dei metodi per il calcolo transizione di stato
Methods methods;
public Methods Methods
    { get { return methods; } }

//Istanza del file XML di configurazione
XmlConfigurationFile xmlConfigurationFile;
public XmlConfigurationFile XmlConfigurationFile
    { get { return xmlConfigurationFile; } }
```

Listato 21

La gerarchia degli elementi `WasteItem` è ora completa, è possibile percorrerla in entrambe le direzioni.

6 Macchina a Stati Finiti

Il sistema di supervisione dell'impianto è stato studiato in modo da poter conoscere, in ogni momento e per ogni elemento, la condizione attuale. Ogni oggetto `WasteItem` possiede un campo `State` che individua l'appartenenza ad uno stato logico ben determinato.

La Macchina a Stati Finiti (FSM) è una astrazione logica ben conosciuta nel campo informatico, esiste una vasta letteratura sull'argomento, specializzata sul tipo di linguaggio adottato. Naturalmente esiste anche un pattern correlato per la programmazione ad oggetti, il pattern "State", che rende possibile cambiare il comportamento di un oggetto come conseguenza del cambiamento del suo stato interno.

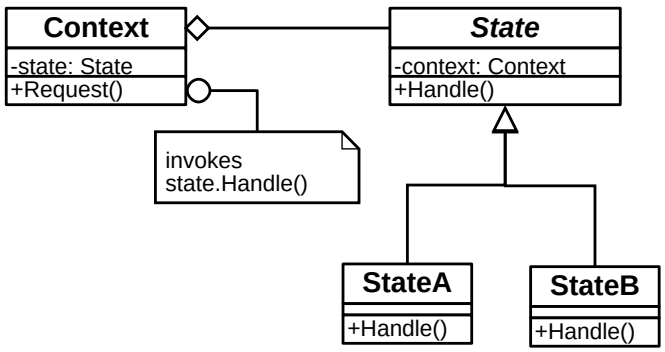


Figura 53

Pattern State

Il pattern State offre un modo elegante per organizzare il codice che dipende dallo stato di un oggetto; la logica che determina le transizioni di stato non è inserita all'interno di blocchi monolitici di codice ma è suddivisa fra le sottoclassi di *`State`*. Incapsulare ogni transizione di stato all'interno di una classe eleva il concetto di stato computazionale al rango di oggetto [GHJV95].

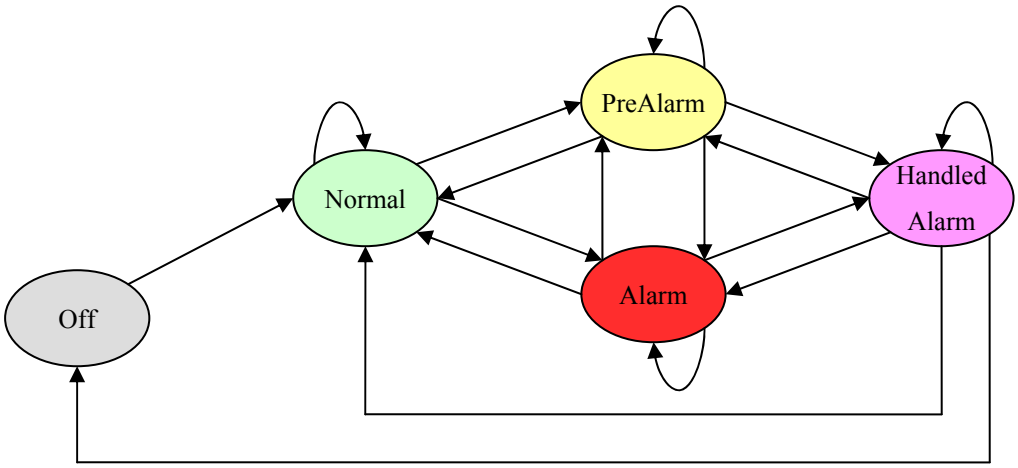


Figura 54

Macchina a Stati Finiti e transizioni di stato

Gli stati sono stati definiti per mezzo di un oggetto di tipo enumeration, questa soluzione offre il vantaggio di assegnare, contestualmente al nome dello stato, un ordine prioritario.

```
public enum StateEnum { Off, Normal, PreAlarm, Alarm, HandledAlarm }
```

Listato 22

Per ottenere il disaccoppiamento tra codice e funzionalità del programma, si è fatto largo uso delle potenzialità che il linguaggio C# mette a disposizione del programmatore; in particolar modo si è tratto notevole beneficio dall'utilizzo di metodi delegati[9].

```
public
   delegate StateEnum StateTransitionHandler(WasteItem wasteItemSender);
```

Listato 23

Il delegato si comporta a tutti gli effetti come "segnaposto" per i metodi che si occuperanno di valutare quale sarà lo stato successivo dell'elemento in questione.
Il Listato 10 mostra come il metodo `CalculateNextStateByDelegateMethod` sia totalmente generico: non viene fatto alcun riferimento all'oggetto per cui valutare la transizione di stato (può essere sia una fibra che un gruppo, la definizione non cambia perché grazie al polimorfismo sono entrambi `WasteItem`), né una particolare strategia decisionale.

[9] Un metodo Delegato è un tipo che definisce una firma di metodo e può essere associato a qualsiasi metodo con una firma compatibile. Tramite il delegato è possibile invocare il metodo. I delegati vengono utilizzati per passare metodi come argomenti ad altri metodi

`CalculateNextStateByDelegateMethod` è di tipo void, prende in ingresso un oggetto generico `WasteItem` e un delegato `StateTransitionHandler` (che sostituisce il metodo vero e proprio); come viene quindi valutata la transizione di stato?

6.1 Metodi Delegati.

La classe `Methods` mette a disposizione i metodi necessari a valutare quale sarà lo stato successivo dell'oggetto.

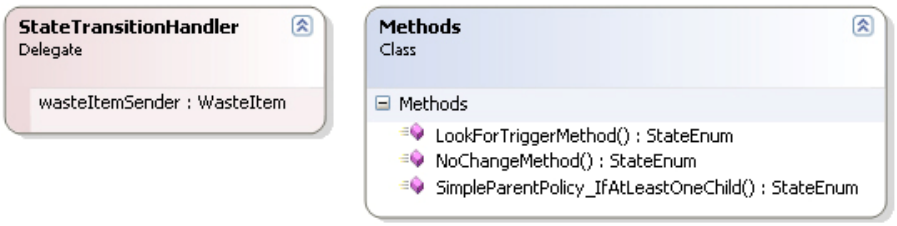


Figura 55

Metodo delegato e classe Methods, contenente i metodi effettivi

Naturalmente la signature dei metodi sarà concorde con quella del delegato ad essi associato (Listato 23): il valore in uscita sarà uno tra quelli definiti nella struttura `StateEnum` (`Normal`, `Alarm`, …), in entrata ci sarà uno qualsiasi degli oggetti `WasteItem`.
Ad esempio il metodo seguente non cambia stato all'oggetto, lo legge e lo ripropone in uscita; questa funzione è utile più ai fini di debug che ad altro.

```
public StateEnum NoChangeMethod(WasteItem wasteItemSender)
    { return wasteItemSender.State; }
```

Listato 24

Quest'altro metodo, invece, legge lo stato dell'oggetto e in funzione di esso entra in un blocco decisionale. Ad esempio se lo stato corrente è `"Normal"` e il valore di deviazione standard normalizzata è, in valore assoluto, compreso tra 2 e 3 ci sarà una transizione allo stato `"PreAlarm"`.

```
public StateEnum LookForTriggerMethod(WasteItem wasteItemSender) {
    //registra lo stato attuale
    StateEnum nextState = wasteItemSender.State;
```

```
    if (wasteItemSender.IsOff) {...}

    if (wasteItemSender.IsNormal) {

        if (wasteItemSender.Is_Abs_NormDeviationSmallerThan2Sigma)
            { nextState = StateEnum.Normal; }

        if (wasteItemSender.Is_Abs_NormDeviationBetween2And3Sigma)
            { nextState = StateEnum.PreAlarm; }

        if (wasteItemSender.Is_Abs_NormDeviationBetween3And4Sigma
            ||
            wasteItemSender.Is_Abs_NormDeviationGreaterThan4Sigma)
            { nextState = StateEnum.Alarm; }

    if (wasteItemSender.IsPreAlarm) {...}
    if (wasteItemSender.IsAlarm) {...}

    return nextState;
}
```

Listato 25

Tramite questi metodi è possibile, quindi, definire le transizioni di stato (gli archetti schematizzati in Figura 54) secondo strategie personalizzate, una sorta di "ricette" che vengono definite in funzione delle esigenze specifiche.
Si lascia quindi piena disponibilità, ad esempio, a differenziare il comportamento che deve essere adottato dal sistema di monitoraggio per un tipo di rifiuto piuttosto che per un altro.

```
public StateEnum BitterPlutonium(WasteItem wasteItemSender)    {...}
public StateEnum SweetPoloniumPie(WasteItem wasteItemSender)   {...}
public StateEnum SpicyUraniumSauce(WasteItem wasteItemSender) {...}
```

Listato 26

Analizzando la modalità per cui si effettua una transizione si può quindi notare che il pattern "State" non è stato implementato nel modo tradizionale suggerito dalla letteratura. Il `Context` diventa il metodo delegato scelto e lo stato rimane una variabile interna dell'oggetto; volendo seguire le motivazioni che originano tale pattern si dovrebbe creare un Context per ogni condizione logica che permette di decidere quale sarà la transizione futura (`trigger`, `normalizedDeviation`, `range`).

6.2 File di configurazione

La scelta del metodo per la transizione di stato più adatto allo scopo viene effettuata tramite un file di configurazione esterno, scritto nella sintassi XML.

```
<states>

    <state name="Off"          color="Gray"
           method="LookForTriggerMethod" >
        <transition next="Normal"       action="NoAction"           />
        <transition next="PreAlarm"     action="PreAlarmAction"     />
    </state>

    <state name="Normal"       color="Green"
           method="LookForTriggerMethod" >
        <transition next="Off"          action="NoAction"           />
        <transition next="Normal"       action="NoAction"           />
        <transition next="PreAlarm"     action="PreAlarmAction"     />
        <transition next="Alarm"        action="AlarmAction"        />
    </state>

    <state name="PreAlarm"     color="Gold"
           method="LookForTriggerMethod" >
        <transition next="Normal"       action="NoAction"           />
        <transition next="PreAlarm"     action="PreAlarmAction"     />
        <transition next="Alarm"        action="AlarmAction"        />
        <transition next="HandledAlarm" action="HandledAlarmAction" />
    </state>

    <state name="Alarm"        color="Red"
           method="LookForTriggerMethod" >
        <transition next="Normal"       action="NoAction"           />
        <transition next="PreAlarm"     action="PreAlarmAction"     />
        <transition next="Alarm"        action="AlarmAction"        />
        <transition next="HandledAlarm" action="HandledAlarmAction" />
    </state>

    <state name="HandledAlarm" color="Magenta"
           method="NoChangeMethod"       >
        <transition next="Off"          action="NoAction"           />
        <transition next="Normal"       action="NoAction"           />
        <transition next="PreAlarm"     action="PreAlarmAction"     />
        <transition next="Alarm"        action="AlarmAction"        />
        <transition next="HandledAlarm" action="HandledAlarmAction" />
    </state>

</states>
```

Listato 27

Dalla lettura del codice è possibile vedere come ogni stato (ad esempio `"Normal"`) è caratterizzato da un colore (`"Green"`), un metodo delegato per valutare la transizione di stato (`"LookForTriggerMethod"`), una action per ogni switch a stato differente, che identifica il nome di un altro metodo delegato che si vuole chiamare quando avviene una transizione (ad

esempio il metodo `alarmAction`, relativo alla transizione allo stato `"Alarm"`, potrebbe far attivare un lampeggiante e azionare una sirena).
Se si giunge, invece, nello stato `"HandledAlarm"`, lo stato rimane bloccato perché il metodo `NoChangeMethod` mantiene lo stato attuale (Listato 24).

```
<actions>

    <action name="NoAction">
        <notes>
            <![CDATA[OK! Do nothing... ]]>
        </notes>
    </action>

    <action name="PreAlarmAction">
        <notes>
            <![CDATA[Blinking lights, Bleep]]>
        </notes>
    </action>

    <action name="AlarmAction">
        <notes>
            <![CDATA[Constant lights, horns and warnings]]>
        </notes>
    </action>

    <action name="HandledAlarmAction">
        <notes>
            <![CDATA[Blinking lights, no sounds]]>
        </notes>
    </action>

</actions>
```

Listato 28

Sono stati previsti pure dei metodi `command`, intesi come comandi che l'operatore impartisce al sistema (accendi, spegni, gestisci …).

```
<commands>

    <command name="TurnON">
        <notes>
            <![CDATA[Operator: "Turn On the channel"]]>
        </notes>
    </command>

    <command name="TurnOff">
        <notes>
            <![CDATA[Operator: "Turn Off the channel"]]>
        </notes>
    </command>

    <command name="Manage">
        <notes>
```

```
            <![CDATA[Operator: "Alarm received"]]>
        </notes>
    </command>

    <command name="Wait">
        <notes>
            <![CDATA[Operator: "Trying to fix the problem"]]>
        </notes>
    </command>

    <command name="RunAway">
        <notes>
            <![CDATA[Operator: " DANGER! I can't solve the problem!"]]>
        </notes>
    </command>

</commands>
```

Listato 29

La classe `Methods` dovrebbe quindi implementare i seguenti metodi.

```
#region Actions
    public void NoAction()            { ... }
    public void PreAlarmAction()      { ... }
    public void AlarmAction()         { ... }
    public void HandledAlarmAction()  { ... }
#endregion

#region Commands
    public void TurnOn()   { ... }
    public void TurnOff()  { ... }
    public void Manage()   { ... }
    public void Wait()     { ... }
    public void RunAway()  { ... }
#endregion
```

Listato 30

La sezione `ranges` definisce degli intervalli di frequenze, identificati dall'esponente della potenze in base 10 della radiazione attualmente registrata, ed associa ad essi un colore.
Ad esempio un tasso misurato di radiazione pari a 10.5 kHz sarebbe identificato con

$$range = \lfloor \log(10500) \rfloor = 4$$

```
<ranges>

    <range name="0"  color="Blue">
        <notes>  <![CDATA[0 - 10 Hz]]>       </notes>
    </range>
```

```
    <range name="1"  color="Blue">
        <notes>  <![CDATA[10 - 100 Hz]]>     </notes>
    </range>

    <range name="2"  color="DarkCyan">
        <notes>  <![CDATA[100 - 1000 Hz]]>   </notes>
    </range>

    <range name="3"  color="Green">
        <notes>  <![CDATA[1 - 10 kHz]]>      </notes>
    </range>

    <range name="4"  color="Green">
        <notes>  <![CDATA[10 - 100 kHz]]>    </notes>
    </range>

    <range name="5"  color="Gold">
        <notes>  <![CDATA[100 - 1000 kHz]]>  </notes>
    </range>

    <range name="6"  color="OrangeRed">
        <notes>  <![CDATA[1 - 10 MHz]]>      </notes>
    </range>

    <range name="7"  color="Red">
        <notes>  <![CDATA[10 - 100 MHz]]>    </notes>
    </range>

</ranges>
```

Listato 31

```
<activities>

    <activity name="none"   downLimit="0"     upLimit="0"
              color="White">
        <notes>
            <![CDATA[No activity]]>
        </notes>
    </activity>

    <activity name="low"    downLimit="1"     upLimit="1000"
              color="Green">
        <notes>
            <![CDATA[This is Low Activity state; Data capture OK]]>
        </notes>
    </activity>

    <activity name="medium" downLimit="1000"  upLimit="10000"
              color="Orange">
        <notes>
            <![CDATA[This is Medium Activity state; Data capture OK]]>
        </notes>
    </activity>

    <activity name="high"   downLimit="10000" upLimit="50000000"
              color="Red">
        <notes>
            <![CDATA[This is High Activity state; Data capture OK]]>
        </notes>
    </activity>
```

```
</activities>
```

Listato 32

Il settore `activities` identifica degli intervalli per cui può essere associata una valutazione quantitativa al tasso di radiazione: basso (1 Hz – 1 kHz), medio (1 kHz – 10 kHz), alto (da 10 kHz in poi).

```
<triggers>

    <trigger name ="---"  downSigmaLimit="-2"    upSigmaLimit="2"    />
    <trigger name ="--/"  downSigmaLimit="2"     upSigmaLimit="3"    />
    <trigger name ="--\"  downSigmaLimit="-3"    upSigmaLimit="-2"   />
    <trigger name ="-//"  downSigmaLimit="3"     upSigmaLimit="4"    />
    <trigger name ="-\\"  downSigmaLimit="-4"    upSigmaLimit="-3"   />
    <trigger name ="///"  downSigmaLimit="4"     upSigmaLimit="1000" />
    <trigger name ="\\\"  downSigmaLimit="-1000" upSigmaLimit="-4"   />

</triggers>
```

Listato 33

Infine il settore `triggers`, nel quale sono definiti degli intervalli (la cui unità di misura è adimensionale perché misurati in termini di deviazione standard), entro cui si definisce l'andamento statistico dell'oggetto: stazionario (`"---"`), in lieve (`"--/"`) o repentino (`"///"`) incremento o decremento(`"--\"`, `"\\\"`); il significato di questi numeri verrà spiegato nel paragrafo successivo.

Tutte queste informazioni vengono caricate, all'avvio del programma, grazie alla classe `XmlConfigurationFile`, che viene resa disponibile da un unico punto di accesso: `Plant`.

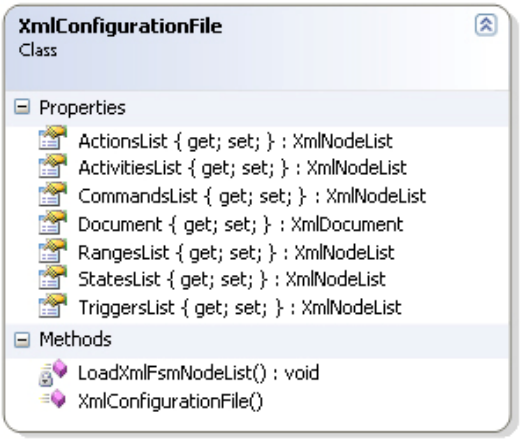


Figura 56

Classe XmlConfigurationFile

```
public XmlConfigurationFile() {
    Document = new XmlDocument();
        try {
            string path = Application.StartupPath.Remove
                              (Application.StartupPath.IndexOf("\\bin"));
            Document.Load(path + "/XmlConfig.xml");
        }
        catch (System.IO.FileNotFoundException)
            { MessageBox.Show("XML File not found!"); }
        LoadXmlFsmNodeList();
}

#region Presentation Members
    public XmlDocument Document              { private set; get; }
    public static XmlNodeList TriggersList   { private set; get; }
    public static XmlNodeList ActivitiesList { private set; get; }
    public static XmlNodeList RangesList     { private set; get; }
    public static XmlNodeList CommandsList   { private set; get; }
    public static XmlNodeList ActionsList    { private set; get; }
    public static XmlNodeList StatesList     { private set; get; }
#endregion

private void LoadXmlFsmNodeList() {
   try {
     TriggersList =   Document.SelectNodes(string.Format("//trigger"));
     ActivitiesList = Document.SelectNodes(string.Format("//activity"));
     RangesList =     Document.SelectNodes(string.Format("//range"));
     CommandsList =   Document.SelectNodes(string.Format("//command"));
     ActionsList =    Document.SelectNodes(string.Format("//action"));
     StatesList =     Document.SelectNodes(string.Format("//state"));
   }
   catch (System.NullReferenceException)
       { MessageBox.Show("Node List not found in XML file! :("); }
}
```

Listato 34

6.3 Statistiche sul tasso di radiazione

Gli eventi di radiazione, emessa dai rifiuti nucleari, seguono una distribuzione di probabilità di tipo poissoniana[10], definita dalla funzione

$$P(x) = \frac{e^{-\lambda}\lambda^{x}}{x!}$$

in cui:

- λ è un qualsiasi valore positivo equivalente al numero di successi che ci si aspetta si verifichino in un dato intervallo di tempo

[10] Una variabile casuale poissoniana esprime la probabilità che un numero di eventi si verifichino nell'unità di tempo, se questi eventi hanno una media conosciuta e ciascuno accade indipendentemente dagli altri.

- x è il numero di occorrenze per cui si vuole prevedere la probabilità

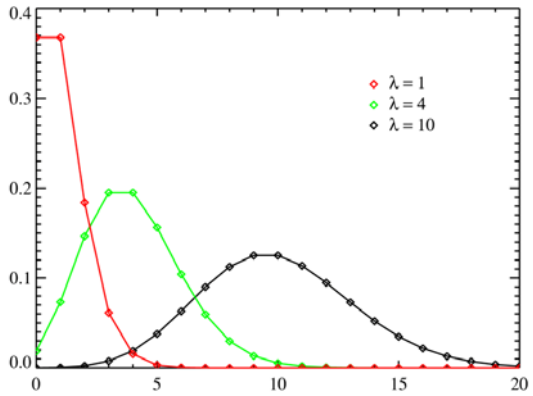


Figura 57

Distribuzione di Poisson

La distribuzione poissoniana gode della proprietà che il valore atteso μ e la varianza σ^2 coincidono e sono pari a λ. Poichè il valore di λ che si registra è molto grande la distribuzione può essere approssimata con una variabile casuale normale (gaussiana), con valore atteso e varianza pari a λ, che verrà associato al tasso di Radiazione R.

La deviazione standard σ, ossia la dispersione dei dati intorno al valore atteso, può essere determinata come $\sigma = \sqrt{R}$.

Calcolare l'integrale in un intervallo di una funzione densità di probabilità equivale a trovare la probabilità che un evento che segue quella distribuzione cada all'interno dell'intervallo stesso, $\int_a^b f(x) = P(a \le x \le b)$; in particolare se la distribuzione densità di probabilità è una gaussiana valgono le fasce di probabilità seguenti.

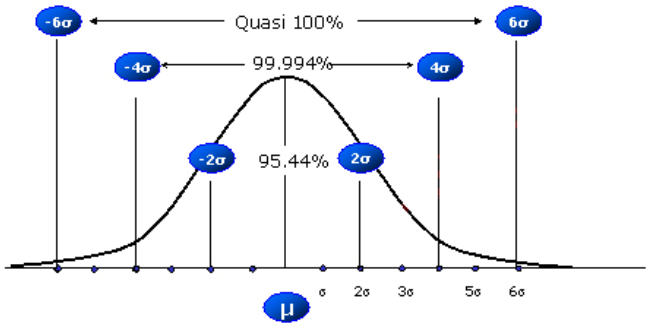


Figura 58

Distribuzione Normale e fasce di probabilità valutate in intervalli σ

Poiché il tempo di invio dati dall'FPGA non è un parametro fisso, ma configurabile da remoto come periodico o su richiesta, a destinazione il calcolo del tasso di radiazione deve essere ogni volta calcolato con la seguente formula:

$$R = RateRadiazione = \frac{DeltaCounter}{DeltaTime}$$

per ogni canale; il tasso di radiazione è la prima informazione di alto livello che viene estratta dal frame seriale.

Ogni lettura prevede la verifica che il nuovo dato sia compatibile con la fascia di valori entro una certa percentuale, valutata rispetto la statistica delle precedenti, per determinare se il tasso di Radiazione registrato si possa definire stazionario oppure no.

La deviazione standard normalizzata σ_{norm}, tra una misurazione e quella precedente, sarà calcolabile come $\sigma_{norm} = \left| \frac{R_{previous} - R_{new}}{\sigma_{previous}} \right|$

E' possibile dunque misurare direttamente la variazione statistica tenendo conto della storia precedente, poiché data la vecchia lettura PreviousRate la nuova lettura NewRate sarà distante da essa di una grandezza esprimibile in unità di σ.

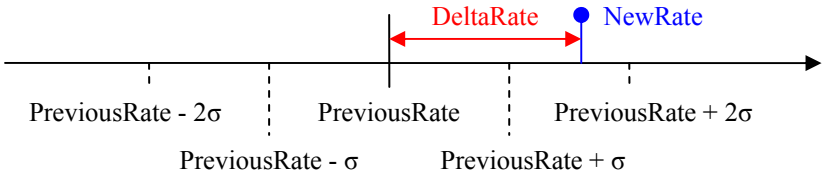


Figura 59

Misura della variazione

Ogni lettura sarà presumibilmente sempre diversa da quella precedente, perché si riferisce ad un processo aleatorio, si farà uso quindi di una percentuale di probabilità desiderata (ad esempio < 96%) entro la quale valutare l'andamento statistico.

```
public void CalculateSigmaAndNormalizedDeviation(int deltaCounter,
                                                  double deltaTime){
    double deltaRate, sigma;
    try{
        targetFibre.NewRate = deltaCounter / deltaTime;
        deltaRate = targetFibre.NewRate - targetFibre.PreviousRate;
        sigma = Math.Sqrt(targetFibre.PreviousRate);
        targetFibre.PreviousRate = targetFibre.NewRate;
        targetFibre.NormalizedDeviation = Math.Round(deltaRate/sigma,2);
    }
    catch (System.ArgumentOutOfRangeException){}
}
```

Listato 35

Funzione per il calcolo della Deviazione Standard Normalizzata

7 Gestione degli Eventi

Il pattern "Observer" (noto anche con il nome Publish-Subscribe) permette di definire una dipendenza uno a molti fra oggetti, in modo tale che se un oggetto cambia il suo stato interno, ciascuno degli oggetti dipendenti da esso viene notificato e aggiornato automaticamente. L'Observer nasce dall'esigenza di mantenere un alto livello di consistenza fra classi correlate, senza peraltro produrre situazioni di forte dipendenza e accoppiamento elevato [GHJV95].
Una situazione tipica di utilizzo si ha quando la modifica dello stato di un oggetto (nel caso in esame una fibra) implica un cambiamento dello stato di altri oggetti correlati, a prescindere dal loro numero (ad esempio il bidone a cui la fibra appartiene, l'interfaccia grafica, una routine che effettua il log dei cambiamenti di stati, ecc…). In questo caso la modifica dello stato dell'oggetto (Publisher) si deve propagare agli oggetti correlati (Subscriber) in modo tale che essi possano aggiornare il loro stato di conseguenza.

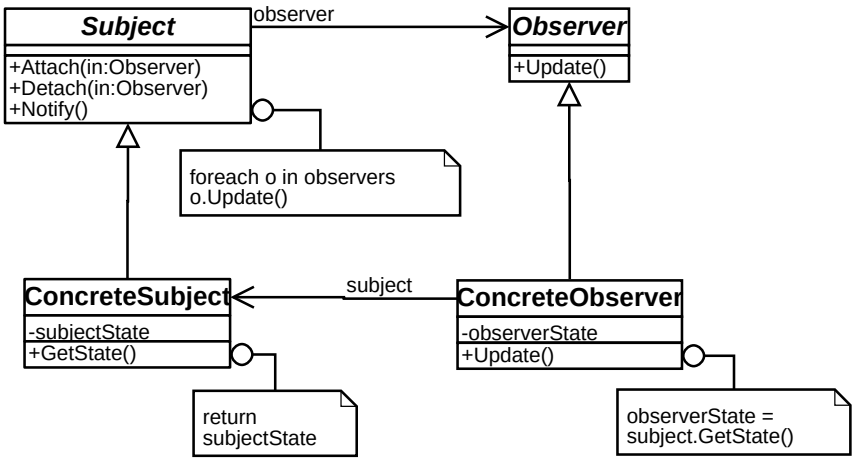


Figura 60

Pattern Observer

Il meccanismo per inviare modifiche nell'ambito del .NET Framework è fornito in modo nativo dai tipi delegate e dagli eventi.
Una classe che funge da Publisher (Subject) espone in generale sulla sua interfaccia una serie di eventi corrispondenti ad un tipo particolare di delegate. Le classi Subscriber (Observer)

sottoscrivono l'evento e ad esso associano un metodo interno (event handler) che deve rispettare la firma definita dal tipo delegate associato all'evento. L'event handler viene chiamato nel momento in cui il Publisher inoltra ai suoi Subscriber la notifica, rendendo possibile in questo modo l'esecuzione di codice in ciascun Subscriber al variare dello stato interno del Publisher.

La procedura per cui gli eventi, generati da una fibra, vengono notificati agli ascoltatori è la seguente:

- Ciascuna fibra eredita da `WasteItem` la definizione di un evento, `PropertyChanged` (dello stesso tipo del delegato contenuto nelle librerie di sistema .NET, `PropertyChangedEventHandler`), Listato 7.
- Quando una proprietà della fibra (ad esempio `State`) cambia valore viene chiamato il metodo gestore, `OnPropertyChanged`, con parametro la stringa corrispondente al nome della proprietà (`"State"`), Listato 8.
- Il gestore quindi, se ha almeno un ascoltatore che ha richiesto la notifica dell'evento, istanzia una classe contenitore dati evento (passando la stringa relativa al nome della proprietà cambiata e il riferimento all'oggetto dentro cui si è verificato l'evento), Listato 7.

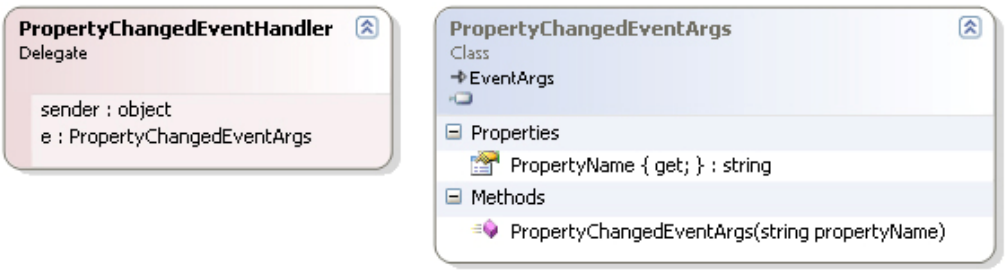


Figura 61

Evento e classe contenitore dati

- Tutti gli oggetti che vogliono ricevere notifica da quella particolare fibra devono sottoscrivere gli eventi che essa genera (ad esempio ogni bidone sottoscrive gli eventi generati da ogni fibra che contiene all'atto della loro creazione), Listato 13.
- Ogni oggetto Subscriber gestirà l'evento nella maniera più opportuna, ad esempio il bidone è interessato al cambiamento della proprietà `"State"` delle proprie fibre;

quando questa condizione accade rivaluta il proprio stato in funzione degli stati di tutte le fibre che contiene, Listato 13.

```
public StateEnum
       SimpleParentPolicy_IfAtLeastOneChild(WasteItem wasteItemSender) {

    // registra lo stato attuale
    StateEnum nextState = StateEnum.Off;
    // Bidone: prendi lo stato più alto di tutte le fibre contenute
    if (wasteItemSender.GetType().Equals(typeof(Drum))) {
        Drum targetDrum = (Drum)wasteItemSender;
        StateEnum maxState =
            (from fibre in targetDrum.Fibres select fibre.State).Max();
        nextState = maxState;
    }
    // Box: prendi lo stato più alto di tutti i bidoni contenuti
    if (wasteItemSender.GetType().Equals(typeof(Box))) {
        Box targetBox = (Box)wasteItemSender;
        foreach (Drum drum in targetBox.Drums) {
            if (drum.State > nextState)
                nextState = drum.State;
        }
    }
    // Gruppo: prendi lo stato più alto di tutti i box contenuti
    if (wasteItemSender.GetType().Equals(typeof(Group))) {
        Group targetGroup = (Group)wasteItemSender;
        foreach (Box box in targetGroup.Boxes) {
            if (box.State > nextState)
                nextState = box.State;
        }
    }
    return nextState;
}
```

Listato 36

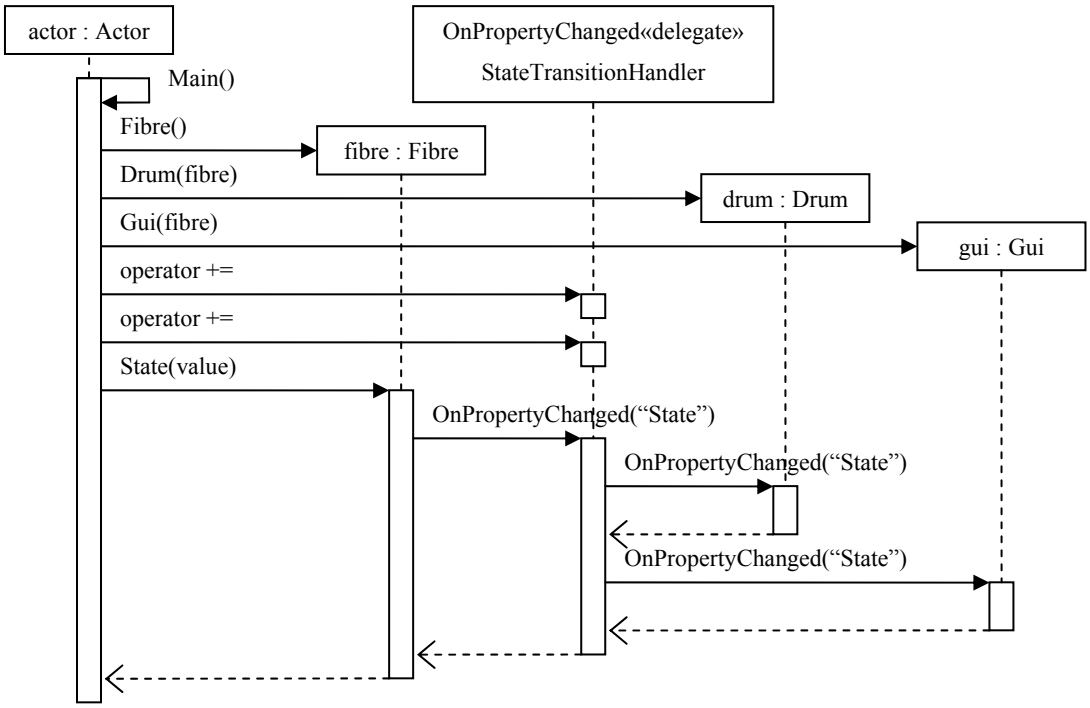


Figura 62

Delegate Invocation Sequence Diagram

Come conseguenza di ciò ogni bidone sarà "sensibile" al cambiamento di stato di ogni fibra che contiene e aggiornerà il proprio in funzione di esse; allo stesso modo i box sottoscriveranno gli eventi generati dai bidoni (Listato 16) e i gruppi faranno lo stesso con i box (Listato 19). La profondità di propagazione dell'evento generato dalla fibra, su per la gerarchia, viene determinata dalle policy di gestione di ogni elemento.

Il Listato 36 mostra un metodo generico, che fa uso del polimorfismo per identificare l'oggetto chiamante, per la determinazione dello stato successivo del contenitore in funzione della regola «se almeno uno dei figli è in uno stato a priorità maggiore acquisisci quello stato»; è possibile definire delle strategie differenziate per ogni oggetto.

```
public StateEnum DrumPolicy(Drum drumSender)     { ... }
public StateEnum BoxPolicy(Box boxSender)        { ... }
public StateEnum GroupPolicy(Group groupSender) { ... }
```

Listato 37

Rimane da implementare la gestione dell'evento `CollectionChanged`, analoga a quella fatta per `PropertyChanged`.

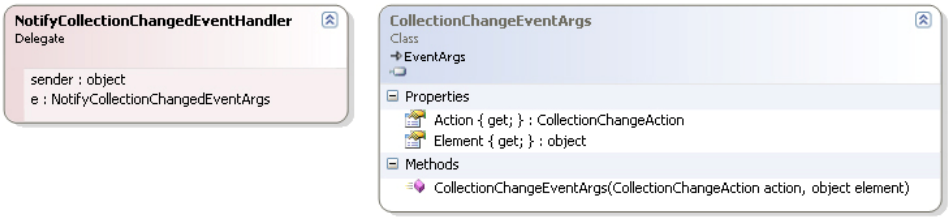


Figura 63

Evento e classe contenitore dati

8 Struttura del programma

All'avvio del programma viene inizializzata e aperta la comunicazione con la porta seriale del PC, collegata direttamente all'FPGA, tramite l'intervento della classe `SerialCommunication`.

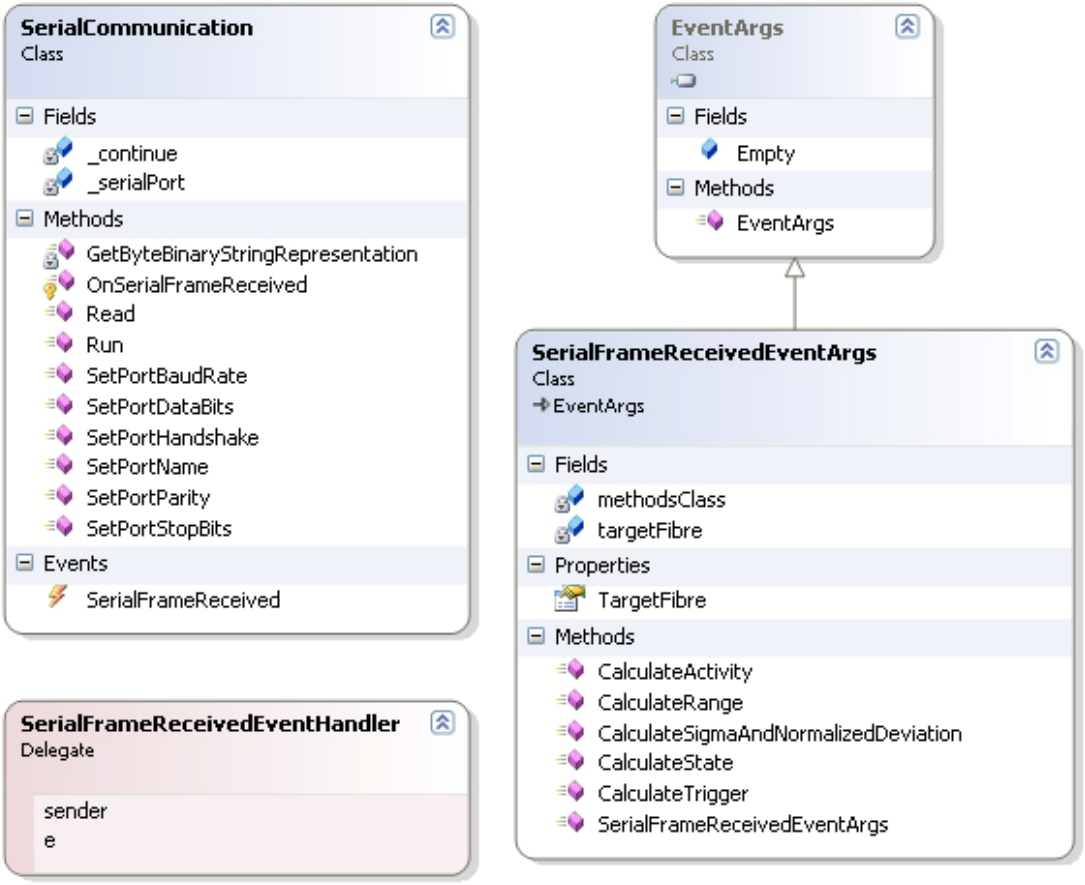


Figura 64

Classi per la comunicazione e l'interpretazione dei dati da porta seriale

Nella Classe `SerialCommunication` sono dichiarati il membro evento, `SerialFrameReceived`, dello stesso tipo del delegato (`SerialFrameReceivedEventHandler`), e il metodo `OnSerialFrameReceived`, che si

occupa di instanziare una nuova variabile di tipo event [11] alla quale assegna il membro event già dichiarato nella classe stessa (sempre `SerialFrameReceived`). Successivamente chiama il costruttore della variabile precedentemente definita con `sender=this` (istanza della classe corrente) e passa come argomento (la classe contenitore `e`) quella ricevuta in ingresso.

```
public event SerialFrameReceivedEventHandler SerialFrameReceived;

protected
virtual void OnSerialFrameReceived(SerialFrameReceivedEventArgs e){
    SerialFrameReceivedEventHandler evSerialFrameReceived =
                                                    SerialFrameReceived;
    if (evSerialFrameReceived != null){
        evSerialFrameReceived(this, e);
    }
}
```

Listato 38

```
//Delegato che invia la classe contenitore dati a chi la gestirà
public delegate void SerialFrameReceivedEventHandler(object sender,
                                        SerialFrameReceivedEventArgs e);
```

Listato 39

Viene quindi avviato, dal metodo `Run()`, un thread in ascolto sullo stream di byte in arrivo sul canale seriale e si avvia un ciclo continuo di lettura dati ed esecuzione delle procedure.

```
Thread readThread = new Thread(Read);
try { readThread.Start(); }
catch (System.UnauthorizedAccessException) { }
```

Listato 40

Lo stream in entrata dal canale seriale viene accolto in un vettore di bytes e successivamente convertito in un'unica stringa, contenente la rappresentazione binaria [12] dei bytes concatenati.

```
byte[] byteBuffer = new byte[14];

    for (int i = 0; i < 14; i++){
        byteBuffer[i] = Convert.ToByte(_serialPort.ReadByte());
    }

    string s = (GetByteBinaryStringRepresentation(byteBuffer[0])  +
```

[11] Secondo la procedura conforme al modello standard di .NET Framework

[12] Grazie alla mediazione della funzione "GetByteBinaryStringRepresentation", in Appendice (Listato 73), non vengono persi, nel passaggio da byte a string, i bit più significativi posti a 0

```
                GetByteBinaryStringRepresentation(byteBuffer[1])  +
                GetByteBinaryStringRepresentation(byteBuffer[2])  +
                GetByteBinaryStringRepresentation(byteBuffer[3])  +
                GetByteBinaryStringRepresentation(byteBuffer[4])  +
                GetByteBinaryStringRepresentation(byteBuffer[5])  +
                GetByteBinaryStringRepresentation(byteBuffer[6])  +
                GetByteBinaryStringRepresentation(byteBuffer[7])  +
                GetByteBinaryStringRepresentation(byteBuffer[8])  +
                GetByteBinaryStringRepresentation(byteBuffer[9])  +
                GetByteBinaryStringRepresentation(byteBuffer[10]) +
                GetByteBinaryStringRepresentation(byteBuffer[11]) +
                GetByteBinaryStringRepresentation(byteBuffer[12]) +
                GetByteBinaryStringRepresentation(byteBuffer[13]) );
```

Listato 41

Successivamente vengono estratti (la procedura di controllo sullo Start byte e sul riallineamento non è stata ancora implementata) i bit di controllo posti alla fine di ogni byte, e vengono ricompattati i dati.

```
// Bytes   1        4            1          2          4           2
// Tot: 14
//  +--------+--------------+---------+-----------+-----------+---------+
//s=| Start  | DeltaCounter | Channel | DeltaTime | Timestamp | FPGA ID |
//  +--------+--------------+---------+-----------+-----------+---------+
//  |0       |8             |40       |48         |64         |96       |
//   bit di inizio                                   Tot: 112 (0 -> 111)

// DeltaCounter
// 8       16      24      32
// |       |       |       |
// *******X*******X*******X*******X

string deltaCounterString = s.Substring( 8, 7) +
                            s.Substring(16, 7) +
                            s.Substring(24, 7) +
                            s.Substring(32, 7) ;

// Conversione da stringa binaria a numero decimale
int deltaCounter = Convert.ToInt32(deltaCounterString, 2);
```

Listato 42

Dopo aver recuperato, in modo analogo a come fatto per `deltaCounter`, tutti gli altri dati dal frame seriale viene instanziata la classe contenitore dati evento, a cui vengono passate tali informazioni.

```
int channel =      Convert.ToInt32(channelString,2);
double deltaTime = Convert.ToDouble(Convert.ToInt32(deltaTimeString,2));
int timestamp =    Convert.ToInt32(timestampString, 2);
int fpgaId =       Convert.ToInt32(fpgaIdString,2);
```

Listato 43

```
SerialFrameReceivedEventArgs arSerialFrameReceived =
    new SerialFrameReceivedEventArgs(fpgaId, timestamp, deltaTime,
                                     channel, deltaCounter);

//instanzio la classe perchè il metodo dentro cui mi trovo è statico...
SerialCommunication anInstanceOf = new SerialCommunication();
anInstanceOf.OnSerialFrameReceived(arSerialFrameReceived);
```

Listato 44

La classe `SerialFrameReceivedEventArgs`:

- acquisisce il riferimento alla classe Methods (che contiene le funzioni per la transizione di stato) e crea un oggetto fibra;
- quando istanziata assegna alla fibra creata (`targetFibre`) il riferimento all'elemento in uscita dalla query LINQ[13] sul dictionary dei bidoni assegnati (accedendo al bidone come schematizzato in Figura 48);
- esegue i metodi per ricavare, dai dati grezzi in entrata, le informazioni di alto livello per la fibra interessata.

```
Methods methodsClass = Plant.UniqueInstance.Methods;

Fibre targetFibre = new Fibre();
public Fibre TargetFibre
    { get { return targetFibre; } }

//costruttore
public SerialFrameReceivedEventArgs(int fpgaId, int timestamp,
                  double deltaTime, int channelNumber,int deltaCounter){

    //query LINQ sul Dictionary<Drums> statico, contenente la coppia
    //chiave: fpga ID
    //valore: bidone corrispondente
    var query = from drum in Plant.UniqueInstance.AssignedDrumsDictionary
                where drum.Key == fpgaId
                select drum.Value.Fibres[channelNumber];

    //assegna il risultato (una collezione) ad una fibra
    foreach (Fibre fibre in query)
        { targetFibre = fibre; }

    CalculateSigmaAndNormalizedDeviation(deltaCounter, deltaTime);
    CalculateActivity();
    CalculateRange();
    CalculateTrigger();
```

[13] Language Integrated Query: è un componente del .NET Framework che aggiunge ai linguaggi .NET la possibilità di effettuare interrogazioni su oggetti, utilizzando una sintassi simile a SQL.

```
    CalculateState();
}
```

Listato 45

Individuata la fibra desiderata si calcolano le informazioni di alto livello attraverso le funzioni:

- `CalculateSigmaAndNormalizedDeviation`, già vista nel Listato 35, che calcola la deviazione standard normalizzata;
- `CalculateActivity`, che legge dal file XML (Listato 32) la stringa corrispondente all'intervallo a cui appartiene il tasso di radiazione attuale e la assegna alla fibra;

```
public void CalculateActivity() {
    int upActivityLimit, downActivityLimit;
    try {
        foreach (XmlNode activityNode in
              Plant.UniqueInstance.XmlConfigurationFile.ActivitiesList){

            downActivityLimit =
            Convert.ToInt32(activityNode.Attributes["downLimit"].Value);
            upActivityLimit =
            Convert.ToInt32(activityNode.Attributes["upLimit"].Value);

            if ((downActivityLimit <= targetFibre.PreviousRate)
                &&
                (targetFibre.PreviousRate < upActivityLimit)) {

                targetFibre.Activity =
                                activityNode.Attributes["name"].Value;
            }
        }
    }

    catch (System.NullReferenceException) { }
}
```

Listato 46

- `CalculateRange`, che calcola la base intera dell'esponente in base 10 del tasso di radiazione attuale;

```
public void CalculateRange() {
    try {
        targetFibre.Range =
                        Math.Round(Math.Log10(targetFibre.NewRate),2);
    }

    catch (System.OverflowException) {}
}
```

Listato 47

- `CalculateTrigger`, che legge dal file XML (Listato 33) la stringa che rappresenta la variazione statistica del tasso di radiazione, in funzione delle fasce valutate in termini di σ;

```
public void CalculateTrigger() {
    int downTriggerLimit, upTriggerLimit;
    try {
        foreach (XmlNode triggerNode in
                Plant.UniqueInstance.XmlConfigurationFile.TriggersList){

            downTriggerLimit =
        Convert.ToInt32(triggerNode.Attributes["downSigmaLimit"].Value);

            upTriggerLimit =
          Convert.ToInt32(triggerNode.Attributes["upSigmaLimit"].Value);

            if ((downTriggerLimit <= targetFibre.NormalizedDeviation)
                &&
                (targetFibre.NormalizedDeviation < upTriggerLimit)) {

                targetFibre.Trigger =
                                triggerNode.Attributes["name"].Value;
            }
        }
    }

    catch (System.NullReferenceException) { }
}
```

Listato 48

- `CalculateState`, che si occupa di recuperare dal file XML (Listato 27) il nome del metodo che si vuole invocare per valutare la transizione di stato; se corrisponde ad una funzione presente nella classe Methods crea quindi il delegato e lo passa alla funzione `CalculateNextStateByDelegateMethod` della fibra (Listato 10).

```
Public void CalculateState() {
  try {
  foreach (XmlNode stateNode in
           Plant.UniqueInstance.XmlConfigurationFile.StatesList){

    //se nella lista mi trovo nel nodo che corrisponde allo stato
    //attuale... viene eseguito solo una volta nel foreach
    if (stateNode.Attributes["name"].Value.
                                  Equals(targetFibre.State.ToString())){

      //prendo dal file XML la stringa con il nome del metodo desiderato
      string methodString = stateNode.Attributes["method"].Value;
      //e la converto in un metodo presente nella classe
      MethodInfo method =
                          methodsClass.GetType().GetMethod(methodString);

      if (method != null) {
        try {
          //creo il delegato del metodo
          StateTransitionHandler theDelegate =
                          (StateTransitionHandler)Delegate.
                           CreateDelegate(typeof(StateTransitionHandler),
                                          methodsClass, method, false);
          //e lo passo alla funzione CalculateState(theDelegate);
```

```
            //invoco la chiamata al metodo che calcolerà lo stato
            //successivo passando il riferimento all'oggetto corrente,
            //in questo caso la fibra selezionata, il delegato con lo
            //stesso nome del metodo da invocare
            targetFibre.CalculateNextStateByDelegateMethod(targetFibre,
                                                           theDelegate);
          }

          catch (System.InvalidCastException) {
            MessageBox.Show("error loading delegate method\n" +
                            methodString +
                            "() !!! \nInvalidCastException :(");
          }
          catch (System.ArgumentException err) {
            MessageBox.Show(err.TargetSite.ToString() +
                            "\nerror loading delegate method\n" +
                            methodString +
                            "() !!! \nArgumentException :(");
          }
        }

        else { MessageBox.Show(methodString + "() not found!!!"); }
      }
    }
    }

  catch (System.NullReferenceException) {
    MessageBox.Show("Null Reference\n in CalculateState method!!!");
  }
}
```

Listato 49

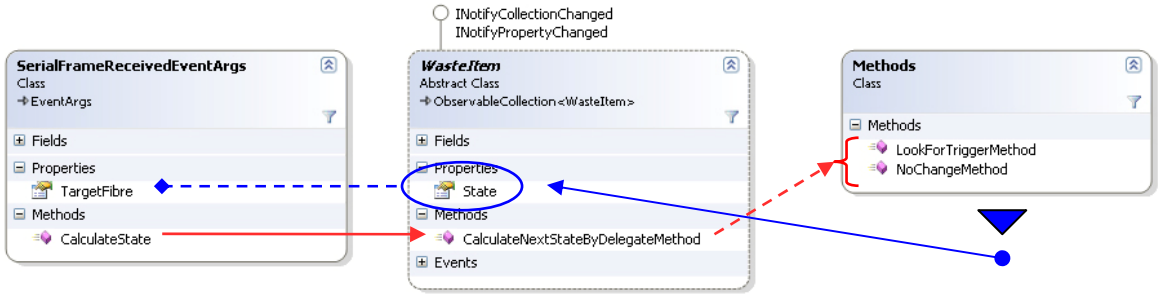


Figura 65

Schema di interazione tra classi per la valutazione dello stato

9 *Interfaccia Grafica*

Classicamente la tecnologia per lo sviluppo di applicazioni grafiche in ambiente MS Windows è basata su Graphics Device Interface (GDI), un set di API progettata oltre 20 anni fa per la renderizzazione grafica 2D incapace di sfruttare nativamente tutte le potenzialità messe a disposizione dall'hardware grafico di ultima generazione.

Windows Presentation Foundation (WPF) [14] è un motore grafico rivoluzionario, introdotto nella versione 3.0 di .NET Framework, che si propone di unire le varie tecnologie (grafica 3D, componenti multimediali, documenti) fornendo una piattaforma unica per la realizzazione di applicazioni "di impatto", eliminando la necessità di utilizzo di tecnologie e/o componenti esterni [MSDNg].

XAML (eXtensible Application Markup Language) è un linguaggio di descrizione utilizzato per istanziare oggetti .NET, prevalentemente per costruire le interfacce grafiche in WPF. Un'interfaccia grafica scritta con WPF, a differenza delle classiche finestre Windows Forms, non è tradotta in codice binario ma serializzata in un set di tags XAML.

Quando si esegue l'applicazione questi tags vengono utilizzati per generare gli oggetti che compongono l'interfaccia grafica.

WPF supporta nativamente il data binding[15], anzi incrementa il set di operazioni possibili rispetto Windows Forms.

Per l'implementazione dell'interfaccia grafica si è tratta liberamente ispirazione da un articolo presente sull'archivio The Code Project [CPJS], seguendo il pattern Model-View-ViewModel [MSDNe] [MSDNf].

---

[14] Precedentemente conosciuto con il nome Avalon

[15] Relazione che permette di accedere ad informazioni contenute in un oggetto sorgente e utilizzarle per impostare delle proprietà in un oggetto destinazione.

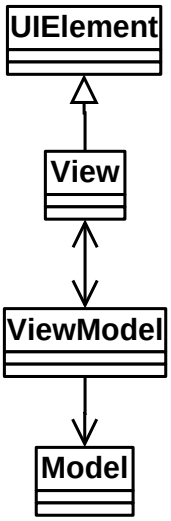


Figura 66

Pattern Model-View-ViewModel

La classe `ViewModel` contiene tutte le proprietà necessarie e le interfacce specifiche per la grafica, necessarie a sviluppare una GUI. La classe `View` [16] si lega (data binding) alla `ViewModel`, ed esegue i comandi per richiedere un'azione da essa. `ViewModel` comunica con `Model` e richiede l'aggiornamento in seguito ad un'azione dell'utente.

La gerarchia di classi ViewModel è stata definita in modo da legarsi, con una relazione uno a uno, con la gerarchia degli elementi WasteItem (Figura 40).

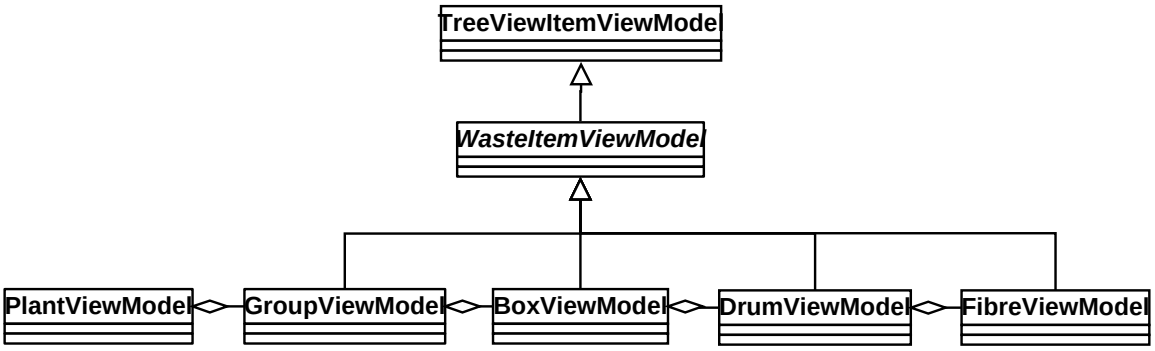


Figura 67

Gerarchia delle classi ViewModel

[16] La classe View è un elemento grafico descritto nel linguaggio XAML

9.1 Classe TreeViewItemViewModel

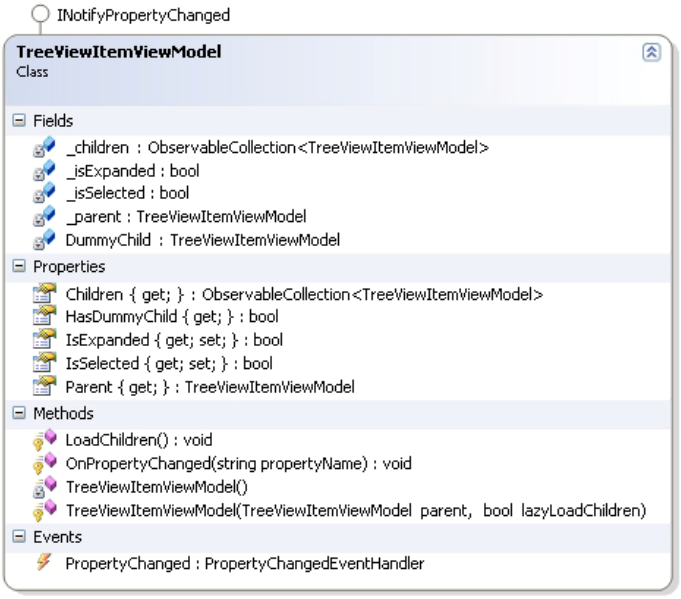


Figura 68

Classe TreeViewItemViewModel

Fornisce tutti i presentation members utili per costruite l'oggetto `TreeView` che verrà visualizzato nell'interfaccia finale. Implementa l'interfaccia `INotifyPropertyChanged`, quindi possiede l'evento `PropertyChanged` e il metodo gestore [17].

```
bool _isSelected;

//Notifica se il TreeViewItem associato
//con l'oggetto è selezionato
public bool IsSelected {

    get { return _isSelected; }
    set {
        if (value != _isSelected) {
            _isSelected = value;
            this.OnPropertyChanged("IsSelected");
        }
    }
}
```

Listato 50

[17] Gli oggetti chiameranno il metodo OnPropertyChanged passando la stringa corrispondente al nome della proprietà che è cambiata; se viene passata una stringa vuota saranno rivalutate tutte le proprietà della classe che sono state sottoscritte.

Il membro `IsExpanded`, che notifica la richiesta di espansione di un `TreeViewItem` (come conseguenza della selezione dell'operatore), rende disponibile la funzionalità del lazy-load dei figli, passando al costruttore l'informazione riguardante l'effettiva presenza di figli sul nodo selezionato.

```
bool _isExpanded;

public bool IsExpanded {
    get { return _isExpanded; }
    set {

        if (value != _isExpanded) {
            _isExpanded = value;
            this.OnPropertyChanged("IsExpanded");
        }

        // Expand all the way up to the root.
        if (_isExpanded && _parent != null)
            _parent.IsExpanded = true;

        // Lazy load the child items, if necessary.
        if (this.HasDummyChild) {
            this.Children.Remove(DummyChild);
            this.LoadChildren();
        }
    }
}
```

Listato 51

costruttore e dummy child

```
static readonly
         TreeViewItemViewModel DummyChild = new TreeViewItemViewModel();
readonly TreeViewItemViewModel _parent;
readonly ObservableCollection<TreeViewItemViewModel> _children;

public ObservableCollection<TreeViewItemViewModel> Children
    { get { return _children; } }

public bool HasDummyChild {
    get {
        return this.Children.Count == 1
               &&
               this.Children[0] == DummyChild;
    }
}

#region Constructors
    protected TreeViewItemViewModel(TreeViewItemViewModel parent,
                                    bool lazyLoadChildren) {
```

```
        _parent = parent;
        _children = new ObservableCollection<TreeViewItemViewModel>();

        if (lazyLoadChildren)
            _children.Add(DummyChild);
    }

    // This is used to create the DummyChild instance.
    private TreeViewItemViewModel() { }
#endregion
```

Listato 52

Il caricamento dei figli viene effettuato on demand, le sottoclassi effettueranno l'override di questo metodo per popolare le loro collection `Children`.

```
protected virtual void LoadChildren() { }
```

Listato 53

9.2 *Classe WasteItemViewModel*

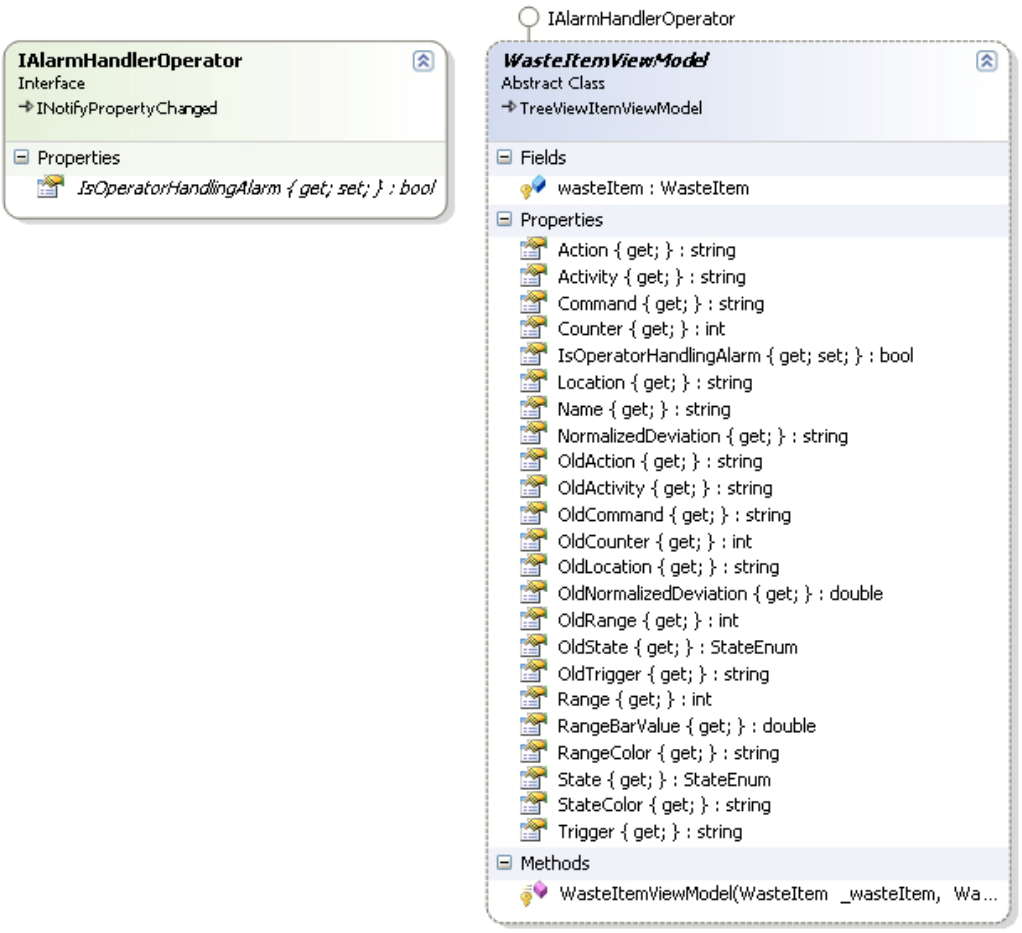


Figura 69

Classe WasteItemViewModel e interfaccia IAlarmHandlerOperator

Costruttore: richiama il costruttore della classe padre e tiene il riferimento dell'oggetto `WasteItem`, passato come parametro, a cui si collegherà.

```
protected WasteItem wasteItem;

protected WasteItemViewModel(WasteItem _wasteItem,
                             WasteItemViewModel parentWasteItem,
                             bool lazyLoadChildren)

                            : base(parentWasteItem, lazyLoadChildren) {

    wasteItem = _wasteItem;
}
```

Listato 54

Presentation Members: restituiscono i dati relativi all'oggetto `WasteItem` a cui il `ViewModel` è collegato.

```
public string NormalizedDeviation {
    get {
        string format = "0.00 [+];0.00 [--]";
        return wasteItem.NormalizedDeviation.ToString(format);
    }
}

public StateEnum State
    { get { return wasteItem.State; } }

public StateEnum OldState
    { get { return wasteItem.OldState; } }

    public int Range
        { get { return Convert.ToInt32(Math.Floor(wasteItem.Range)); } }

public int OldRange
        { get { return Convert.ToInt32(wasteItem.OldRange); } }

public double RangeBarValue {
    get {
        return Math.Pow(10, wasteItem.Range -
                        Math.Floor(wasteItem.Range));
    }
}

public string Name
    { get { return wasteItem.ToString(); } }

public string StateColor {
  get {
    string stateColor = "Black";

    foreach (XmlNode x in XmlConfigurationFile.StatesList) {
      if (x.Attributes["name"].Value.Equals(wasteItem.State.ToString()))
        stateColor = x.Attributes["color"].Value;
    }
    return stateColor;
  }
}

public string RangeColor {
  get {
    string rangeColor = "Black";

    foreach (XmlNode x in XmlConfigurationFile.RangesList) {
      if (x.Attributes["name"].Value.Equals(this.Range.ToString()))
        rangeColor = x.Attributes["color"].Value;
    }
    return rangeColor;
  }
}
```

Listato 55

La classe implementa inoltre la seguente interfaccia:

```
interface IAlarmHandlerOperator : INotifyPropertyChanged
```

```
    { bool IsOperatorHandlingAlarm { get; set; } }
```

Listato 56

che fornisce un punto di accesso per notificare che l'operatore ha ricevuto e sta gestendo un allarme, funzionalità non ancora implementata.

9.3 Classe PlantViewModel

Da qui in poi la gerarchia `ViewModel` effettua un riferimento uno a uno con il corrispettivo `Model` (oggetto `WasteItem`).

```
public class PlantViewModel {

    List<GroupViewModel> groups;

    public PlantViewModel(List<Group> _groups) {
        groups = new List<GroupViewModel>
                 (from g in _groups select new GroupViewModel(g));
    }

    public List<GroupViewModel> Groups
        { get { return groups; } }
}
```

Listato 57

`PlantViewModel` contiene una lista di oggetti `GroupViewModel` associata alla corrispettiva lista di `Group` contenuta in `Plant` (inviata come parametro al costruttore).

9.4 Classe GroupViewModel

```
public class GroupViewModel : WasteItemViewModel {

    Group group;

    public GroupViewModel(Group _group)
                          : base(_group, null, true) {
        group = (Group)base.wasteItem;
        group.PropertyChanged += this.SubscriveToEvent;
    }

    protected override void LoadChildren() {
        foreach (Box box in group)
            base.Children.Add(new BoxViewModel(box, this));
    }

    protected void SubscriveToEvent(object sender,
                                    PropertyChangedEventArgs e) {
```

```
        group = (Group)sender;
        base.OnPropertyChanged(e.PropertyName);
    }
}
```

Listato 58

`GroupViewModel` associa la collection di `Children "BoxViewModel"` ai `Box` contenuti in ciascun `Group`, sottoscrivendo gli eventi `PropertyChanged`.

9.5 Classe BoxViewModel

```
public class BoxViewModel : WasteItemViewModel {

    Box box;

    public BoxViewModel(Box _box, GroupViewModel parentGroup)
                        : base(_box, parentGroup, true) {
        box = (Box)base.wasteItem;
        box.PropertyChanged += this.SubscriveToEvent;
    }

    protected override void LoadChildren() {
         foreach (Drum drum in box.Drums)
             base.Children.Add(new DrumViewModel(drum, this));
    }

    protected void SubscriveToEvent(object sender,
                                    PropertyChangedEventArgs e) {
        box = (Box)sender;
        base.OnPropertyChanged(e.PropertyName);
    }
}
```

Listato 59

Comportamento analogo a `GroupViewModel`.

9.6 Classe DrumViewModel

```
public class BoxViewModel : WasteItemViewModel {

    Drum drum;

    public DrumViewModel(Drum _drum, BoxViewModel parentBox)
                         : base(_drum, parentBox, true) {
        drum = (Drum)base.wasteItem;
        drum.PropertyChanged += this.SubscriveToEvent;
```

```
    }

    protected override void LoadChildren() {
        foreach (Fibre fibre in drum.Fibres)
            base.Children.Add(new FibreViewModel(fibre, this));
    }

    protected void SubscriveToEvent(object sender,
                                    PropertyChangedEventArgs e) {
        drum = (Drum)sender;
        base.OnPropertyChanged(e.PropertyName);
    }
```

Listato 60

Comportamento analogo a `GroupViewModel`.

9.7 Classe FibreViewModel

```
public class FibreViewModel : WasteItemViewModel {
    Fibre fibre;

    public FibreViewModel(Fibre _fibre, DrumViewModel parentDrum)
                          : base(_fibre, parentDrum, false) {

        fibre = (Fibre)base.wasteItem;
        fibre.PropertyChanged += this.SubscribeToEvent;
    }

    protected void SubscribeToEvent(object sender,
                                    PropertyChangedEventArgs e) {
        fibre = (Fibre)sender;
        base.OnPropertyChanged(e.PropertyName);
    }
}
```

Listato 61

Comportamento analogo a `GroupViewModel`, con la differenza che `FibreViewModel` non ha figli.

9.8 Data Binding e Classe View

Le classi `View` incapsulano l'interfaccia grafica e la logica per l'interazione, Figura 66, tipicamente sono costituite da controlli progettati per interagire con gli oggetti `ViewModel`.
Il loro ruolo specifico consiste nel fornire all'operatore un'interfaccia grafica che rappresenti l'applicazione specifica, rendendo disponibili dati e logica nel modo più appropriato, oltre che interpretare i comandi impartiti dall'operatore e inoltrarli alle `ViewModel` associate.

Il data binding in WPF può essere dichiarato direttamente in XAML. Il `DataContext` della `View` viene associato alla `ViewModel` e in modo automatico si crea un legame con le sue proprietà ed i suoi eventi.

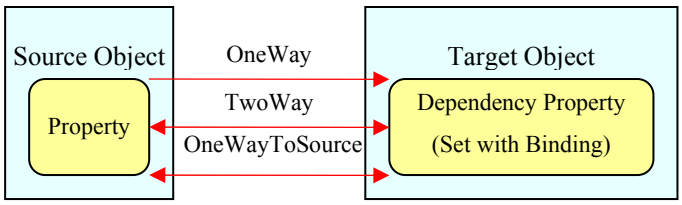


Figura 70

Differenti tipi di data binding

La modalità `"Two way"` si occupa di leggere lo stato della `ViewModel` per mostrarlo nella `View` ed aggiornare la `ViewModel` in seguito ad un'operazione dell'operatore sulla UI. Le notifiche di aggiornamento, generate dalla `ViewModel` permettono alla finestra grafica di aggiornare i dati automaticamente, quando cambia lo stato di una proprietà nella `ViewModel` [MSDNh].

```
public partial class FlatTree2 : UserControl {
    PlantViewModel viewModel;

    public FlatTree2() {
        InitializeComponent();
        viewModel = new PlantViewModel(Plant.UniqueInstance.GroupsList);
        base.DataContext = viewModel;
    }

    private void UserControl_Loaded(object sender, RoutedEventArgs e)
        {}

    private void TreeView_SelectedItemChanged(object sender,
                                RoutedPropertyChangedEventArgs<object> e)
        {}
}
```

Listato 62

L'oggetto `TreeView` di WPF è uno strumento estremamente flessibile, supporta la virtualizzazione (ad esempio la creazione on-demand di `TreeViewItem`s), permette una completa personalizzazione grafica, oltre ad avere il supporto nativo per il data binding.

Poiché la gerarchia degli elementi `WasteItem` è puramente gerarchica è stato creato un `TreeView` radice, che corrisponde a `Plant`.

```
<TreeView Name="treeView"
          ItemsSource="{Binding Groups}"
          SelectedItemChanged="TreeView_SelectedItemChanged">

    <TreeView.ItemContainerStyle>
        <Style TargetType="{x:Type TreeViewItem}">
            <Setter Property="IsExpanded"
                    Value="{Binding IsExpanded, Mode=TwoWay}" />
            <Setter Property="IsSelected"
                    Value="{Binding IsSelected, Mode=TwoWay}" />
            <Setter Property="FontWeight"
                    Value="Normal" />
            <Style.Triggers>
                <Trigger Property="IsSelected" Value="True">
                    <Setter Property="FontWeight" Value="Bold" />
                </Trigger>
            </Style.Triggers>
        </Style>
    </TreeView.ItemContainerStyle>
```

Listato 63

I nodi (livello 1) del `TreeView` sono associati a `Groups`, la lista di `GroupViewModel` che ha il riferimento alla lista `Groups` (`Model`) di `Plant`, Listato 57. Le proprietà `"IsExpanded"` e `"IsSelected"` vengono impostate in modalità `"TwoWay"` in modo da permettere l'inoltro degli eventi, che vengono generati nell'interfaccia grafica, ad opera dell'utente.

- Livello 1

```
<TreeView.Resources>
    <HierarchicalDataTemplate DataType="{x:Type DMNR:GroupViewModel}"
                              ItemsSource="{Binding Children}" >
        <StackPanel Orientation="Horizontal">
            <Image Width="30" Height="30" Margin="3,0"
                   Source="View\Images\Group.png" />
            <TextBlock Text="{Binding Name}"
                       VerticalAlignment="Center" />
            <TextBlock  Text="    "
                        Background="{Binding StateColor,
                        Mode=OneWay,
                        UpdateSourceTrigger=PropertyChanged}"
                        Width="20" Height="20" />
        </StackPanel>
    </HierarchicalDataTemplate>
```

Listato 64

E' la vista corrispondente ai gruppi, i nodi figlio vengono popolati con i box corrispondenti; ogni gruppo viene presentato con una stringa che lo identifica e da un'immagine. Le proprietà a cui si effettua il binding sono scelte tra quelle messe a disposizione da `WasteItemViewModel`, Listato 55.

Per ogni elemento della grarchia viene mostrato il colore corrispondente allo stato attuale (come specificato nel file XML, Listato 27): i gruppi, i box e i bidoni lo espongono attraverso un riquadro colorato.

- Livello 2

```
<HierarchicalDataTemplate DataType="{x:Type DMNR:BoxViewModel}"
                          ItemsSource="{Binding Children}">
    <StackPanel Orientation="Horizontal">
        <Image Width="35" Height="35" Margin="3,0"
               Source="View\Images\Box.png" />
        <TextBlock Text="{Binding Name}"
                   VerticalAlignment="Center"/>
        <TextBlock  Text="    "
                    Background="{Binding StateColor,
                    Mode=OneWay,
                    UpdateSourceTrigger=PropertyChanged}"
                    Width="20" Height="20" />
    </StackPanel>
</HierarchicalDataTemplate>
```

Listato 65

E' la vista corrispondente ai box, i nodi figlio vengono popolati con i bidoni corrispondenti. Valgono le stesse osservazioni fatte per il livello superiore.

- Livello 3

```
<HierarchicalDataTemplate DataType="{x:Type DMNR:DrumViewModel}"
                          ItemsSource="{Binding Children}">
    <StackPanel Orientation="Horizontal">
        <Image Width="30" Height="30" Margin="3,0"
               Source="View\Images\Drum.png" />
        <TextBlock VerticalAlignment="Center"
                   Text="{Binding Name}" />
        <TextBlock  Text="    "
                    Background="{Binding StateColor,
                    Mode=OneWay,
                    UpdateSourceTrigger=PropertyChanged}"
                    Width="20" Height="20" />
    </StackPanel>
</HierarchicalDataTemplate>
```

Listato 66

E' la vista corrispondente ai bidoni, come sopra.

- Dentro il livello 3

```
<DMNR:Fibre x:Key="myFibre"  />
<DataTemplate DataType="{x:Type DMNR:FibreViewModel }">
    <StackPanel Orientation="Horizontal">
        <Image Width="25" Height="15" Margin="3,0"
               Source="View\Images\Fibre.png" />
        <Grid>
            <Grid.RowDefinitions>
                <RowDefinition></RowDefinition>
            </Grid.RowDefinitions>
            <Grid.ColumnDefinitions>
                <ColumnDefinition Width="60"></ColumnDefinition>
                <ColumnDefinition Width="80"></ColumnDefinition>
            </Grid.ColumnDefinitions>

            <TextBlock Grid.Row="0" Grid.Column="0"
                       Text="{Binding Name}" />

            <TextBlock Grid.Row="0" Grid.Column="1"
                       HorizontalAlignment="Center"
                       Foreground="{Binding StateColor,
                       Mode=OneWay,
                       UpdateSourceTrigger=PropertyChanged}"
                       Text="{Binding State, Mode=OneWay,
                             UpdateSourceTrigger=PropertyChanged}"/>
        </Grid>

        <ProgressBar Height="10" Name="progressBar1" Width="80"
                     Foreground="{Binding RangeColor}"
                     Value="{Binding RangeBarValue,
                     Mode=OneWay,
                     UpdateSourceTrigger=PropertyChanged}"
                     Maximum="10" SmallChange="0.05" />

         <Grid>
            <Grid.RowDefinitions>
                <RowDefinition></RowDefinition>
            </Grid.RowDefinitions>
            <Grid.ColumnDefinitions>
                <ColumnDefinition Width="40"></ColumnDefinition>
                <ColumnDefinition Width="15"></ColumnDefinition>
                <ColumnDefinition Width="50"></ColumnDefinition>
                <ColumnDefinition Width="40"></ColumnDefinition>
                <ColumnDefinition Width="80"></ColumnDefinition>
                <ColumnDefinition Width="20"></ColumnDefinition>
                <ColumnDefinition Width="20"></ColumnDefinition>
            </Grid.ColumnDefinitions>

            <TextBlock Grid.Row="0" Grid.Column="0"
                       Text="exp" HorizontalAlignment="Center" />

            <TextBlock Grid.Row="0" Grid.Column="1"
                       Text="{Binding Range, Mode=OneWay,
                             UpdateSourceTrigger=PropertyChanged}"
                             TextAlignment="Center" />

            <TextBlock Grid.Row="0" Grid.Column="2"
                       TextAlignment="Right"
                       Text="{Binding NormalizedDeviation,
                             UpdateSourceTrigger=PropertyChanged},
                             Mode=OneWay "/>
```

```
                <TextBlock Grid.Row="0" Grid.Column="3"
                           Text="   σ" HorizontalAlignment="Left" />

                <TextBlock Grid.Row="0" Grid.Column="4"
                           HorizontalAlignment="Center"
                           Foreground="DarkOrange"
                           Text="{Binding OldState,
                                 UpdateSourceTrigger=PropertyChanged},
                                 Mode=OneWay" />
            </Grid>
        </StackPanel>
    </DataTemplate>
</TreeView.Resources>
```

Listato 67

Vengono visualizzate le informazioni dettagliate relative a ciascuna fibra:

- Immagine fibra e nome
- Stato attuale (nel colore corrispondente)
- Base ed esponente del tasso di radiazione attuale con la seguente vista: valore su una progress bar (base) ed un numero (esponente)
- Deviazione Standard Normalizzata, con segno
- Stato precedente a quello attuale

La finestra grafica risultante è la seguente

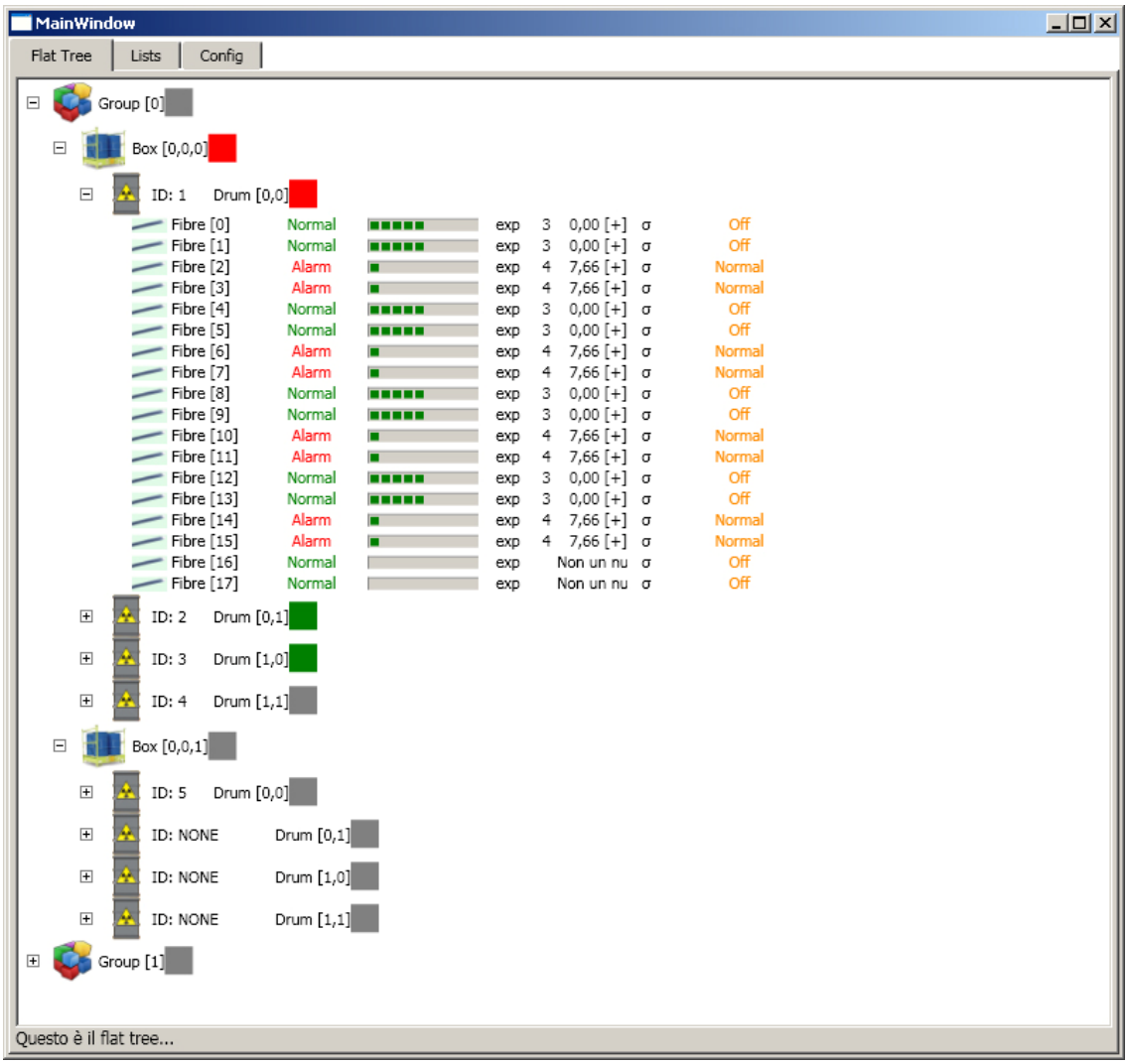


Figura 71

Interfaccia grafica sviluppata

NUCLEAR REPOSITORIES

La sezione seguente mostra i particolari di costruzione dei vari prototipi, sviluppati durante il presente lavoro di tesi presso i Laboratori Nazionali del Sud INFN di Catania. [LNS]
Il rivelatore di radiazione ionizzante è composto di un tratto di fibra ottica di materiale scintillante, ai cui capi sono accoppiati due SiPM.
Dopo aver testato diverse configurazioni meccaniche, di accoppiamento ottico fibra-SiPM, si è giunti ad un setup dalle caratteristiche interessanti, tra cui il basso costo e la facilità di installazione.
Una caratteristica nota delle fibre ottiche, siano esse composte da materiale plastico o vetroso, è la flessibilità del materiale che le compone; risulta indispensabile prevedere un sostegno meccanico intorno ad esse che al contempo fornisca rigidità e protezione.
E' necessario, inoltre, evitare che la fibra scintillante diventi guida di luce derivante dall'ambiente circostante, poiché si vuole far giungere ai suoi capi solo quell'esiguo numero di fotoni generato dal passaggio di radiazione ionizzante per essa.
La configurazione proposta prevede una schermatura dalla luce ambientale ad opera di guaina in poliolefina (un composto di polietilene e polipropilene) termorestringente nera, il sostegno meccanico è, invece, ad opera di tubi in PVC, rigidi lisci o corrugati flessibili, per impianti elettrici domestici.
La lunghezza e la geometria del tratto di fibra influiscono sulla capacità di rivelazione poiché fanno variare l'efficienza geometrica: un tratto più lungo esporrà alla radiazione una superficie più ampia, una fibra arrotolata su se stessa potrà essere colpita più volte; in entrambi i casi la probabilità di generare eventi di scintillazione aumenta.
L'accoppiamento ottico tra i capi della fibra e i SiPM è stato effettuato interponendo tra di essi una piccola quantità di grasso per applicazioni ottiche, con indice di rifrazione prossimo a quello della fibra. Questo accorgimento risulta necessario in quanto il passaggio, della lieve quantità di fotoni prodotti, tra i diversi materiali (fibra plastica, aria, resina epossidica sul SiPM) indurrebbe dispersione, in questo modo si cerca di concentrare tutta la luce sull'area attiva di rivelazione.

E’ stata inoltre realizzata una scatoletta in PVC nero avente funzioni di: allineamento e serraggio tra capo della fibra ottica e area attiva SiPM, contenimento a prova di luce di SiPM e circuito di polarizzazione, supporto meccanico rigido per il montaggio.
Verranno commentate le varie fasi del montaggio e saranno inoltre mostrate alcune configurazioni dimostrative prodotte.
Infine il risultato di diversi test di misura, l’ultimo effettuato in un ambiente in cui sono presenti radiazioni prodotte da rifiuti radioattivi, presso la Centrale Nucleare del Garigliano.

10 Realizzazione rivelatore

La fibra ottica scintillante utilizzata è di tipo monomodale e si presenta di colore giallo-verde. Prodotta da Saint-Gobain Crystals è composta da polistirene ed ha sezione circolare con diametro pari a 1 mm. [SGC]

A differenza di quanto si fa con le fibre composte di materiale vetroso, non è possibile effettuare la lappatura della superficie, dopo il taglio, perché l'operazione di sfregamento con i materiali abrasivi la renderebbe opaca. E' stata dunque utilizzata una ghigliottina, per troncature di fibre ottiche, che ha fornito un taglio netto accettabile per lo scopo.

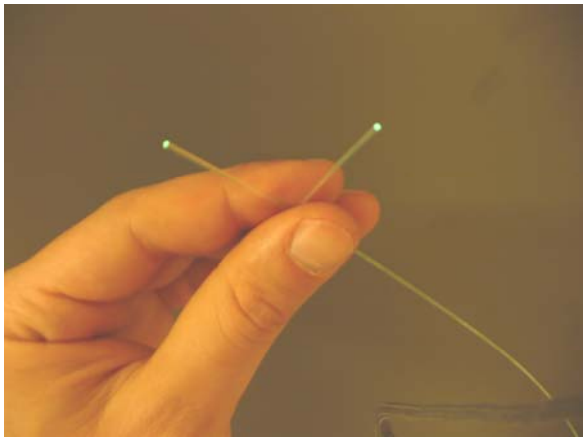


Figura 72
Fibra ottica scintillante

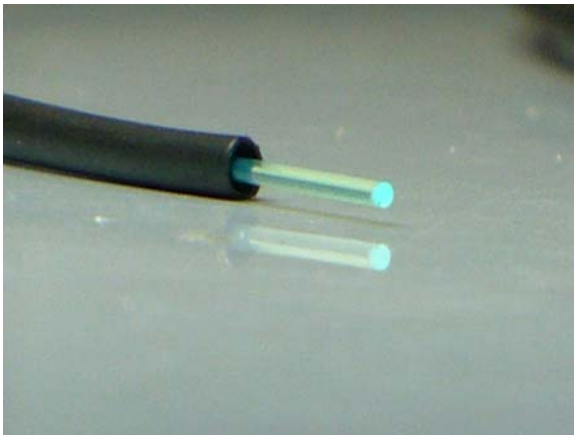


Figura 73
Guaina nera per schermare la luce ambientale

I SiPM, Silicon Photo Multipliers, sono dei rivelatori allo stato solido sensibili al singolo fotone. Essi consistono in una matrice di fotodiodi a valanga operanti in modalità Geiger (APDs), detti microcelle, ciascuno accoppiato individualmente ad un circuito elettronico di attenuazione.

I SiPM offrono guadagno ed efficienza quantica alti, confrontabili a quelli dei tubi fotomoltiplicatori (PMT), oltre ai vantaggi offerti dalle tecnologie su silicio come: dimensioni ridotte, insensibilità ai campi magnetici, basse tensioni di alimentazione e robustezza.

Sono stati utilizzati dei dispositivi da 400 microcelle, prodotti da sensL. [SL]

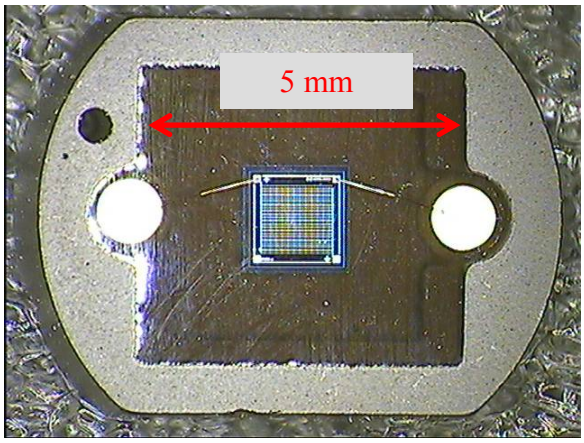


Figura 74

SiPM al microscopio (10x)

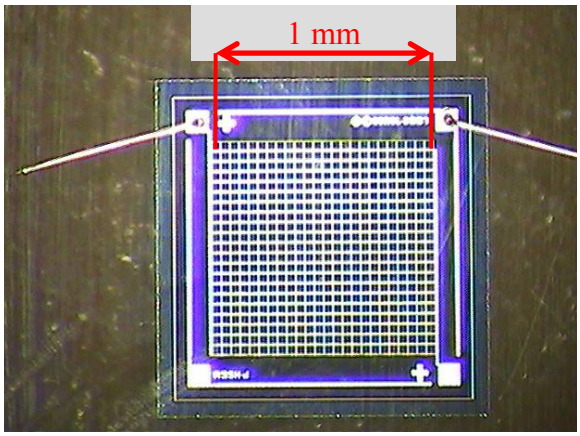


Figura 75

SiPM al microscopio (50x)

Entrambi i capi della fibra vengono posti a contatto dei SiPM per mezzo di una scatoletta in PVC nero, progettata per favorire l'allineamento ottico, il bloccaggio della fibra (effettuato tramite vite apposita) e l'isolamento ottico, realizzata presso le officine meccaniche dei Laboratori Nazionali del Sud.

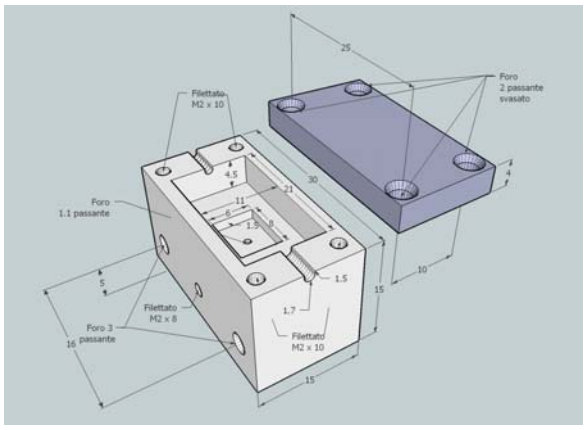


Figura 76

Progetto scatoletta

Figura 77

Scatoletta con fibra

La scatoletta offre un alloggio su misura per il SiPM, montato sul circuito di polarizzazione e prelievo segnale, riducendo al minimo gli spazi vuoti attraversabili da luce ambientale.
La chiusura a "prova di luce" è assicurata grazie ad una guaina di gomma nera elastica, compressa per mezzo viti di serraggio.

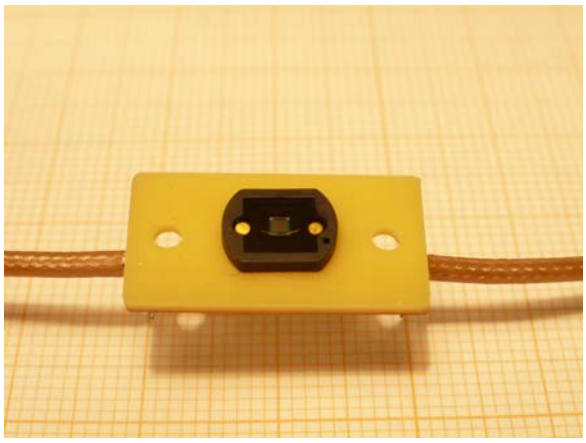


Figura 78
SiPM montato su circuito
di polarizzazione e prelievo segnale

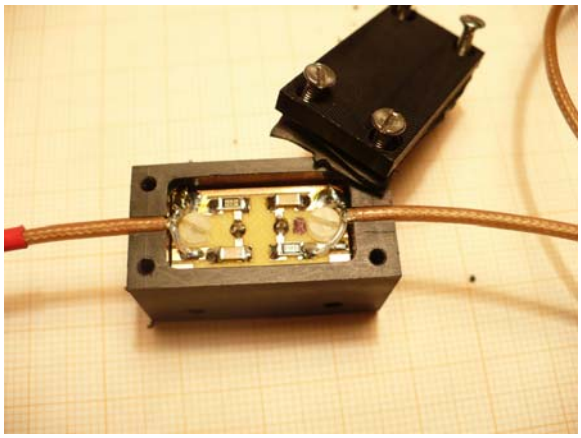


Figura 79
Circuito montato nella scatola
prima della chiusura

La figura seguente mostra lo schema di montaggio dei vari componenti, prima di mettere a contatto fibra e SiPM è necessario interporre una piccola quantità di grasso ottico.

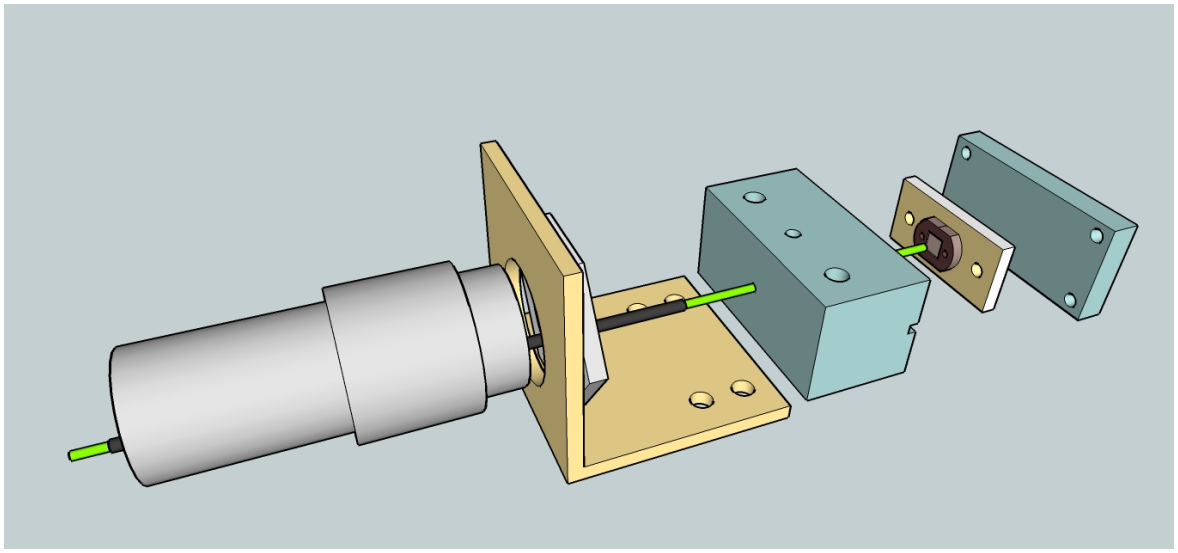


Figura 80
Schema di montaggio dei componenti

Seguendo questa tecnica sono stati realizzati sensori diversi in geometria e lunghezza. Come già detto questi fattori influenzano la sensibilità globale in quanto modificano in modo diretto l'efficienza geometrica della fibra.

Il rivelatore così costruito, a differenza di un contatore Geiger-Muller, non sarà dunque in grado di fornire una misura soggettiva della radiazione ionizzante che lo ha colpito, esistono tanti fattori da tenere in considerazione: lunghezza e diametro della fibra scintillante,

disposizione geometrica, transitori di temperatura dell'ambiente, tensione di alimentazione dei SiPM, tensione di riferimento per i circuiti di discriminazione del segnale.
Per ottenere la medesima lettura da ciascuno di essi, i rivelatori dovrebbero essere dunque caratterizzati in funzione di tutti questi fattori, il sistema di acquisizione dati quindi tarato in funzione di tutte queste variabili.
Il progetto DMNR tuttavia propone un sistema semplice di conteggio eventi di scintillazione, dovuti a radiazione, che "funziona" basandosi sul fatto che qualunque sia il tasso di conteggio registrato in un dato momento, in condizioni di equilibrio esso rimarrà pressoché costante (considerando le fluttuazioni statistiche). Analizzando simultaneamente i dati provenienti da più rivelatori e ponendoli in relazione tra loro, sarà possibile identificare degli incrementi anomali, su una o più fibra, non assimilabili a cause stocastiche.

11 Test di misura

I vari test di misura, effettuati in laboratorio, hanno richiesto l'utilizzo di una sorgente di radiazioni puntiforme che, posta in prossimità dei rivelatori, ha permesso di verificare la sensibilità delle fibre oscurate alla radiazione ionizzante incidente.

Figura 81
Sorgente di raggi γ (Europio, Cobalto e Cesio) con attività complessiva pari a 2.7 MBq

Figura 82
Movimentazione sorgente in prossimità delle fibre, per mezzo di un braccio robotico

Il grafico seguente mostra il risultato ottenuto movimentando la sorgente in prossimità di tre rivelatori, posti in configurazione anulare cilindrica.

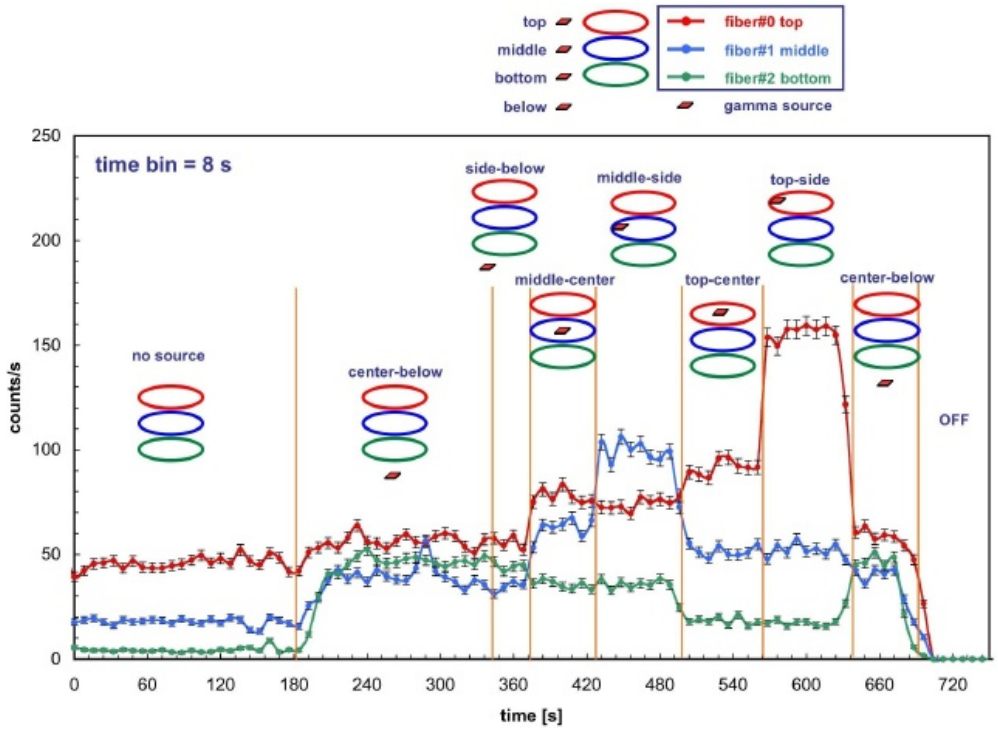


Figura 83

Grafico test di misura: i conteggi registrati dai rivelatori rispecchiano visibilmente l'intensità dei raggi incidenti, in funzione della distanza dalla sorgente di emissione

Sono stati costruiti rivelatori diversi in forma e lunghezza, sia rigidi che flessibili, in modo da poter esplorare varie modalità di utilizzo e geometrie di installazione.

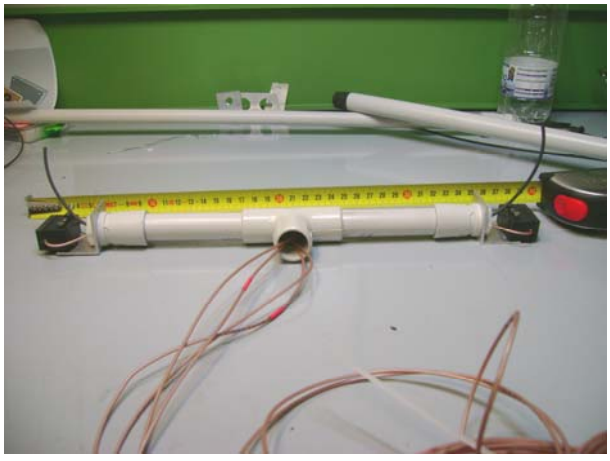


Figura 84

Bacchetta lunga 35 cm con predisposizione per manico telescopico

Figura 85

Configurazione composta da due rivelatori anulari e tre diritti, disposti in geometria cilindrica a griglia

Disponendo i rivelatori in configurazione a griglia, in un prototipo in scala che rappresenta un fusto contenente rifiuti radioattivi, è stata dunque verificata la capacità di determinare il punto sulla superficie del bidone in cui si potrebbe manifestare un incremento nell'emissione di radiazioni.

Mettendo in relazione, infatti, i conteggi registrati su ciascuna fibra è possibile determinare i punti in cui si registra maggiore attività, poiché si registrerà un incremento nelle letture provenienti dalle fibre che si incrociano su di essi.

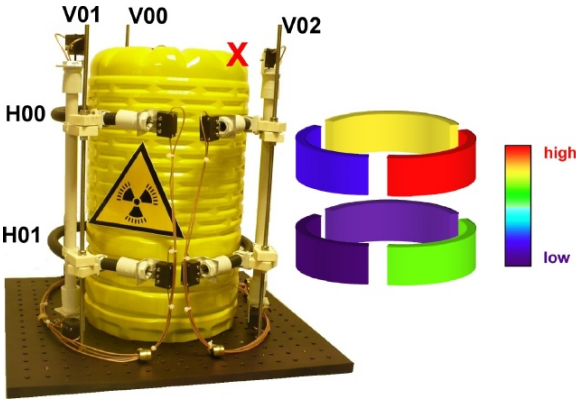


Figura 86

La configurazione a griglia permette, grazie ai punti di intersezione dei rivelatori, di identificare sulla superficie laterale del fusto la zona di incremento attività, situazione simulata posizionando la sorgente nel punto contrassegnato dalla lettera X

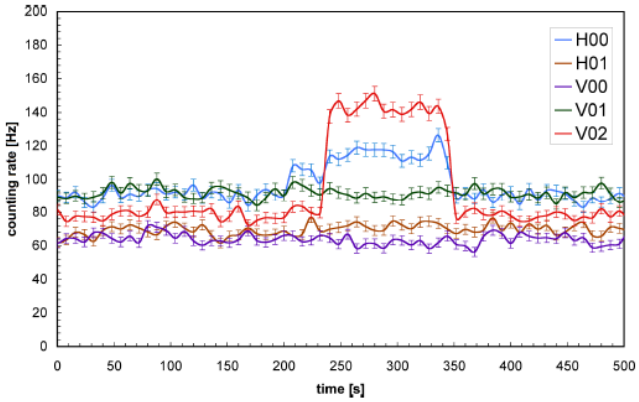


Figura 87

L'aumento dell'attività registrata viene reso evidente grazie alle letture registrati sui rivelatori contrassegnati H00 e V02

Nella volontà di proporre un'installazione semplice e fattibile, si è inoltre provveduto alla progettazione di una piattaforma in acciaio che, già accessoriata di rivelatori lungo gli spigoli e il fondo, possa essere luogo di contenimento per quattro fusti di rifiuti radioattivi.

La disposizione dei fusti da stoccare avverrebbe quindi grazie all'utilizzo di tali piattaforme che, progettate per essere movimentate per mezzo di carroponti, assicurano un ingombro spaziale ottimizzato, in quanto collocabili in posizione contigua o impilata.

Questa particolare configurazione permette inoltre ad un singolo rivelatore anulare, posizionato sul fondo della struttura, di poter monitorare contemporaneamente le basi superiori e inferiori dei fusti cui è interposto.

Si prevede inoltre di dotare, la struttura metallica, di connettori ad aggancio automatico che rendano possibile il collegamento elettrico all'impianto di alimentazione nonché alla rete di trasmissioni dati cablata.

Figura 88
Piattaforma realizzata in dimensioni reali con sagome bidoni

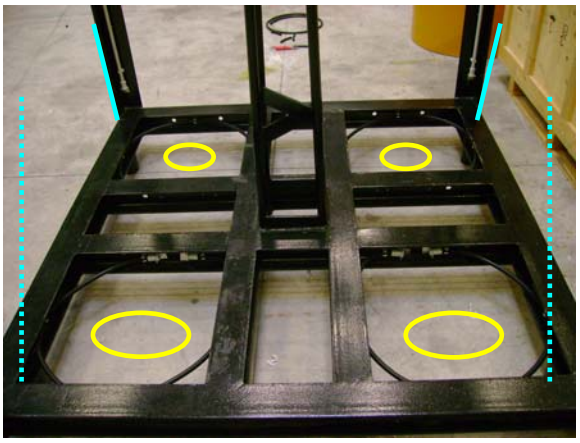


Figura 89
Piattaforma equipaggiata con rivelatori: 4 diritti perimetrali (in azzurro), 4 anulari nel fondo (in giallo)

I risultati incoraggianti ottenuti dal progetto DMNR, in accordo con la vasta produzione scientifica pubblicata su riviste e presentata a conferenze nell'ambito della fisica nucleare, ha fatto nascere una collaborazione tra INFN e Sogin[18].

Tale collaborazione ha permesso di effettuate un test di misura, di vari rivelatori prodotti, all'interno di un reale ambiente adibito allo stoccaggio dei rifiuti radioattivi, la Centrale Nucleare del Garigliano.

[18] Sogin è la di società di Stato incaricata della bonifica ambientale dei siti e della messa in sicurezza dei rifiuti radioattivi provenienti dalle attività nucleari industriali, mediche e di ricerca. [Sog]

Figura 90

Mappa dei depositi temporanei

di materiale radioattivo in Italia

Figura 91

Indicazioni stradali Centrale

Figura 92

Vista esterna della centrale:

camino e contenitore di sicurezza

Il test di misura effettuato è consistito nell'introdurre, all'interno di una sala di stoccaggio temporaneo dei fusti di rifiuti prodotti dalla centrale stessa, un carrello equipaggiato con quattro rivelatori, di varie dimensioni, e i moduli elettronici preposti all'alimentazione dei rivelatori stessi e all'amplificazione del segnale logici in uscita.

Sul carrello è stato pure posto un contatore Geiger-Muller, in modo da poter confrontare i tassi di conteggio registrati con una misurazione oggettiva.

Il grafico seguente mostra i tassi di conteggio registrati durante l'ingresso in sala stoccaggio, partendo dalla posizione P0 (fuori dalla sala, lettura Geiger 0.07 μSv/h) per giungere alla posizione P6 (quasi a contatto con i fusti, lettura Geiger 310 μSv/h). Le posizioni intermedie registrano i tassi rilevati in posizioni di sosta durante il tragitto.

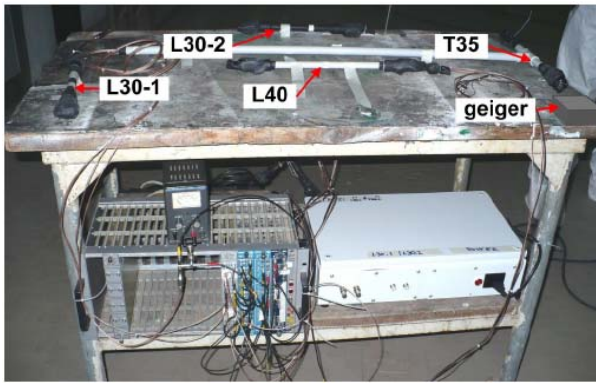


Figura 93

Carrello con dispositivi e moduli elettronici

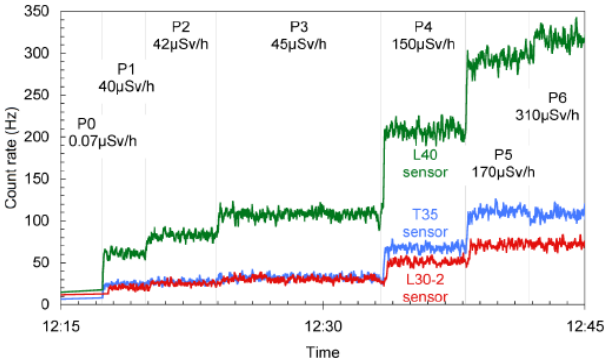


Figura 94

Conteggi registrati durante l'ingresso e il posizionamento in sala stoccaggio

Appendici

12 A - Tipi di comunicazione tra dispositivi elettronici

12.1 Comunicazione Parallela

Il livello fisico trasporta diversi bit di dati contemporaneamente, un bit per pista su PCB o per filo su cavo multipolare. Il throughput è alto per brevi distanze e generalmente viene utilizzata per connettere dispositivi nella stessa circuit board. La lunghezza del bus di solito va mantenuta il più breve possibile poiché il disallineamento dei valori logici aumenta all'aumentare della distanza.

I bus paralleli sono affetti da alti carichi capacitivi, ciò porta ad un aumento del tempo necessario per la carica e scarica elettrica che trasporta i valori logici. Il costo è elevato e generalmente il bus è voluminoso e necessita accorgimenti particolari per essere utilizzato.

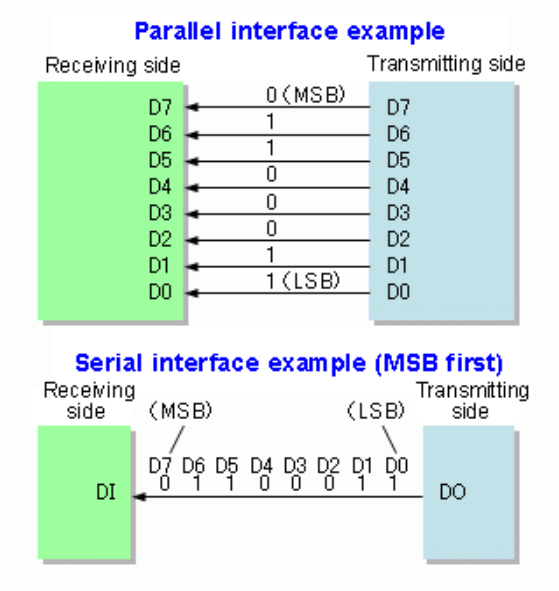


Figura 95

12.2 Comunicazione Seriale

Il livello fisico trasporta un solo bit di dati per unità di tempo, su un solo filo o per pista su PCB.
I blocchi di bit vengono trasmessi un bit per volta e si hanno throughput alti a lunghe distanze; il carico capacitivo sui cavi è inferiore rispetto alla comunicazione parallela, quindi è possibile commutare il valore logico più velocemente nella stessa unità di tempo.
Il costo è inferiore e l'utilizzo di un numero inferiore di fili conduttori la rende più economica della comunicazione parallela; tuttavia la logica d'interfaccia e comunicazione risulta essere più complicata.
Il trasmettitore deve decomporre i blocchi di dati in singoli bit prima di inviarli, a destinazione è necessaria una ricomposizione che opera in maniera inversa. L'utilizzo di segnali di controllo insieme ai dati, inoltre, accresce la complessità del protocollo.

Ci sono due forme principali di trasmissione seriale: trasmissione sincrona e trasmissione asincrona: [VG02]

- Trasmissione seriale sincrona: richiede che sia trasmettitore che ricevitore condividano un clock o che il trasmettitore invii un segnale di timing, in modo da permettere al ricevitore di conoscere l'istante di tempo corretto per leggere il bit di dato successivo. In molti casi di trasmissioni seriali sincrone, se in un dato istante non c'è alcun dato da trasmettere viene trasmessa ugualmente una serie di caratteri di riempimento. La trasmissione sincrona è generalmente più efficiente in termini di throughput poiché sono trasmessi solo bit che trasportano dati, di contro c'è da considerare il costo aggiuntivo dato dalla presenza di un filo dedicato alla condivisione del clock fra trasmettitore e ricevitore.

- Trasmissione seriale asincrona: permette la trasmissione di dati senza condivisione di clock tra sender e receiver. Ciò è reso possibile dal fatto che i partecipanti alla comunicazione concordano in anticipo sui parametri temporali da utilizzare e vengono utilizzati bit speciali di sincronizzazione a inizio e fine trasmissione di un blocco di bit.

12.3 Protocolli di trasmissione seriale

12.3.1 Protocollo I^2C

I^2C, acronimo di Inter Integrated Circuit, è un sistema di comunicazione seriale bifilare.
Sviluppato da Philips Semiconductors nel 1982, è diventato uno standard de facto per la comunicazione tra circuiti integrati.
Nei sistemi elettronici di consumo, industriali e per telecomunicazioni, molto spesso il design di architettura presenta molti elementi comuni come:

- Una logica di controllo, generalmente un microcontrollore;
- Circuiti general purpose, come driver per LED e LCD, RAM, EEPROM, convertitori A/D e D/A;
- Circuiti application oriented, come sensori, DSP, smart cards.

Questo semplice bus bidirezionale nacque quindi per sfruttare queste somiglianze al meglio e massimizzare l'efficienza hardware e la semplicità circuitistica.
Caratteristiche peculiari:

- Sono richiesti solo due fili: serial data line (SDA) e serial clock line (SCL);
- Ogni dispositivo connesso al bus è indirizzabile via software da un indirizzo unico;
- Esiste sempre una relazione master/slave tra dispositivi, i master possono operare sia come trasmettitori che ricevitori;
- Bus multimaster con collision detection e arbitraggio, per prevenire la corruzione dei dati se due o più master iniziano la trasmissione simultaneamente;
- Bus seriale sincrono bidirezionale, con velocità massima pari a: 100 kb/s (Standard mode), 400 kb/s (Fast mode), 1 Mb/s (Fast mode Plus), 3.4 Mb/s (High speed mode);
- Filtraggio on-chip dei picchi sul bus, per preservare l'integrità dei dati;
- Numero massimo dei dispositivi, collegabili sullo stesso bus, limitato solo dal massimo carico capacitivo ammesso dal bus;
- Sono necessari solo due fili, le interconnessioni tra IC sono minimizzate e si riduce la complessità di realizzazione di circuiti stampati.

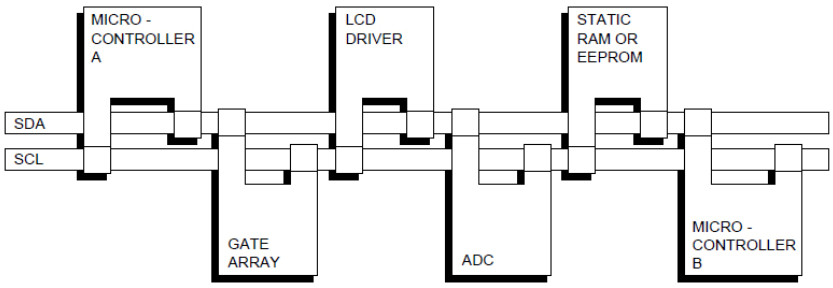
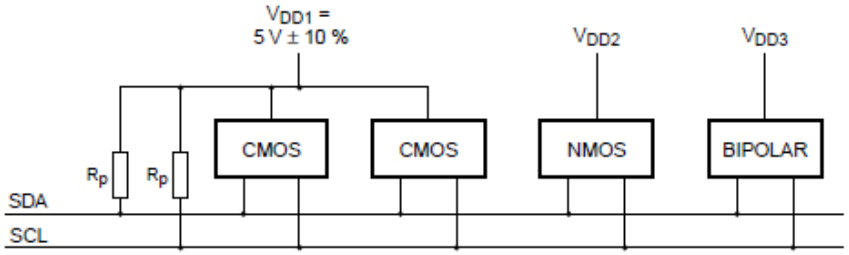


Figura 96

La generazione del clock sul bus è sempre a carico dei dispositivi master. Sia SDA che SCL sono linee bidirezionali connesse ad un generatore di tensione attraverso un resistore pull-up. Quando il bus è libero entrambe le linee sono al valore logico alto.

Poiché è possibile collegare dispositivi a diversa tecnologia sullo stesso bus (CMOS, NMOS, bipolar) i livelli dei valori logici "0" e "1" non sono fissi, ma dipendono dal valore associato di V_{DD}.



V_{DD2}, V_{DD3} are device dependent (e.g., 12 V).

Figura 97

Gli input reference sono fissati dunque in funzione di V_{DD}, nel dettaglio:

- V_{IL} = 30% V_{DD};
- V_{IH} = 70% V_{DD}.

Tutte le transazioni cominciano con uno START (S), una transizione da HIGH a LOW sulla linea SDA mentre SCL è HIGH; finiscono con uno STOP (P), una transizione da LOW a HIGH sulla linea SDA quando SCL è HIGH.

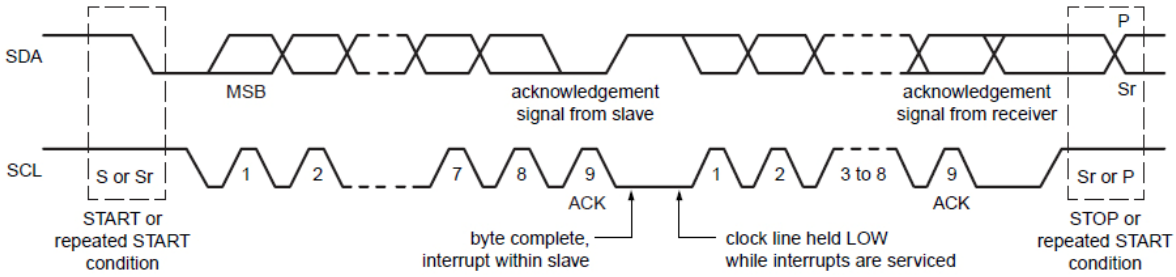


Figura 98

Le condizioni di START e STOP vengono sempre generate dal master, il bus viene considerato occupato dopo che si è verificato un evento START e considerato tale fino alla ricezione di STOP.

Ogni byte trasmesso su SDA deve essere lungo 8 bit, il numero di bytes che possono essere trasmessi in sequenza è senza limite.

Ogni byte deve essere seguito da un bit di Acknowledge, ad opera dello slave: lo slave ha la possibilità di mettere in pausa forzata il master tenendo la linea SCL a valore LOW.

L'arbitraggio entra in gioco quando più master tentano la trasmissione sul bus, inviando dunque uno START simultaneamente.

Poiché ogni master rimane in ascolto, bit per bit, dei dati inviati su SDA, giungerà un momento in cui un master invierà un bit HIGH e rileverà invece un bit LOW, dominante sulla linea, e capirà di aver perso la contesa. Il Master dunque interromperà la trasmissione e la ritenterà dopo che il bus si sarà liberato.

Esistono oggi sul mercato dispositivi che operano da interfaccia tra la porta UART standard di un microprocessore (o microcontrollore) e il bus seriale I^2C.

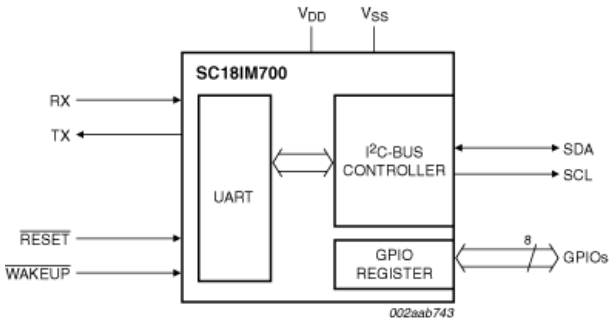


Figura 99

NXP SC18IM700

12.3.2 Protocollo SPI

SPI, acronimo di Serial Peripheral Interface, è un bus seriale sincrono full duplex progettato da Motorola e supportato da molti produttori di circuiti integrati.
I dispositivi comunicano utilizzando una relazione master/slave, la trasmissione viene sempre avviata dal master.
Quando il master genera il clock e seleziona uno slave i dati possono essere scambiati in entrambe le direzioni.

Il protocollo definisce quattro segnali:

- Clock: SCLK;
- Master Data Output, Slave Data Input: MOSI;
- Master Data Input, Slave Data Output: MISO;
- Slave Select: SS.

Se sono presenti più slaves il master deve generare un segnale SS per ogni slave.

Figura 100

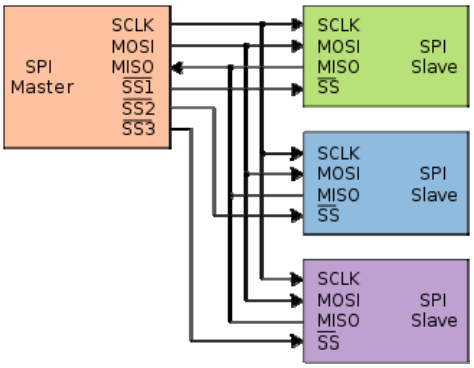


Figura 101

Dasy chain SPI configuration

Alcuni dispositivi SPI sono progettati in modo da essere capaci di operare connessi in cascata, con l'output di ogni slave connesso all'input del successivo.
La porta SPI di ogni slave è progettata in modo da riproporre al dispositivo seguente una copia esatta del blocco di dati ricevuto. Questa modalità richiede un solo segnale SS dal master.

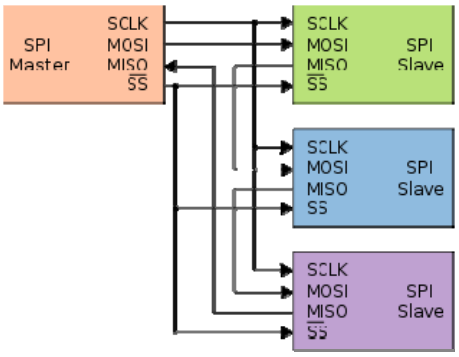


Figura 102

Per iniziare una comunicazione il master configura il clock, utilizzando una frequenza che lo slave supporta (generalmente tra 1 e 70 MHz), successivamente pone il segnale SS LOW per lo slave desiderato. Se necessario aspetta un ciclo di clock e inizia la trasmissione.
Per ogni ciclo di clock si ha una trasmissione bidirezionale lungo il bus: il master invia un bit sulla linea MOSI che verrà letto dallo slave, lo slave invia un bit sulla linea MISO che verrà letto dal master.

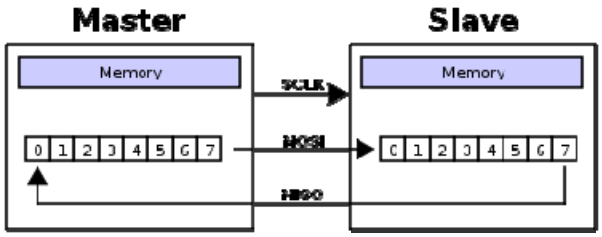


Figura 103

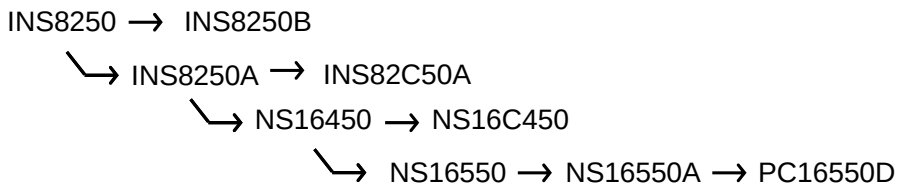
La trasmissione generalmente coinvolge due shift register di dimensioni adeguate, uno nel master ed uno nello slave connessi ad anello: di solito i bit sono inviati a partire dal più significativo.
A trasmissione avvenuta, sia master che slave processano il dato ricevuto: se la trasmissione non si è conclusa riprende senza vincoli temporali.
Il master decide di chiudere la comunicazione fermando il clock e deselezionando lo slave.

SPI non ha un meccanismo di acknowledgment che confermi la ricezione del dato né uno per il controllo di flusso, tuttavia la capacità di comunicazione full duplex ad alta velocità lo rende in molti casi la soluzione più semplice ed efficiente per applicazioni singolo-master/singolo-slave.

Di contro, può essere complicato implementare un sistema multi slave, a causa della mancanza di un sistema di indirizzamento integrato che fa aumentare la complessità del sistema all'aumentare degli slaves da comandare.

12.3.3 Protocollo UART

L'UART, acronimo di Universal Asynchronous Receiver-Transmitter, è un dispositivo hardware che consente una comunicazione seriale asincrona e che viene utilizzato per interfacciare computer, terminali e modem. Le porte seriali del computer sono pilotate da UART, così come le schede di rete e i modem.
Inizialmente IBM scelse come adattatore parallelo/seriale, per il Personal Computer, il modulo UART INS8250, prodotto da National Semiconductor. Le generazioni successive dei computers compatibili IBM continuarono ad utilizzare lo stesso modulo o versioni migliorate della stessa famiglia, fino ad arrivare al modulo NS16450, simile, ma in grado di supportare comunicazioni dati ad una velocità maggiore.
In questi dispositivi la trasmissione o la ricezione di un carattere per volta rappresentava un limite (ad ogni carattere trasmesso o ricevuto veniva generato un interrupt). A tale limite si è posto rimedio con il modulo UART NS16550 dotato di buffer a 16 byte, sia lato trasmissione che lato ricezione. Questo ha consentito di incrementare le velocità di trasferimento seriali e di ridurre i segnali d'interrupt generati verso il Microprocessore e quindi il carico di lavoro.



12.4 Comunicazione seriale asincrona

Il termine asincrono ha il significato di "indipendente dal tempo". In questo caso i dati sono trasmessi in successioni irregolari, anziché con un flusso regolare e continuo. Ciascuna serie di dati è composta dai bit necessari a formare uno o più caratteri ASCII.
La velocità, cui i dati sono trasmessi lungo una linea di comunicazione seriale, è detta *baud rate* ed è espressa in unità di bit al secondo.

La comunicazione seriale mediante UART, che avviene tra un terminale e un modem o tra un modem e un computer, è conforme allo standard elettrico RS-232; il modem è definito DCE, Data Communications Equipment, il terminale o il computer sono definiti DTE, Data Terminal Equipment.

Sulla linea RS-232 il livello logico 1 è definito livello di *mark*, quello 0 invece livello di *space*.

A tali valori logici in genere corrispondono dei livelli di tensione: +12 V per lo space (con possibili valori di utilizzo da +3 a +25 V) e -12 V per il mark (con possibili valori di utilizzo da -3 a -25 V).

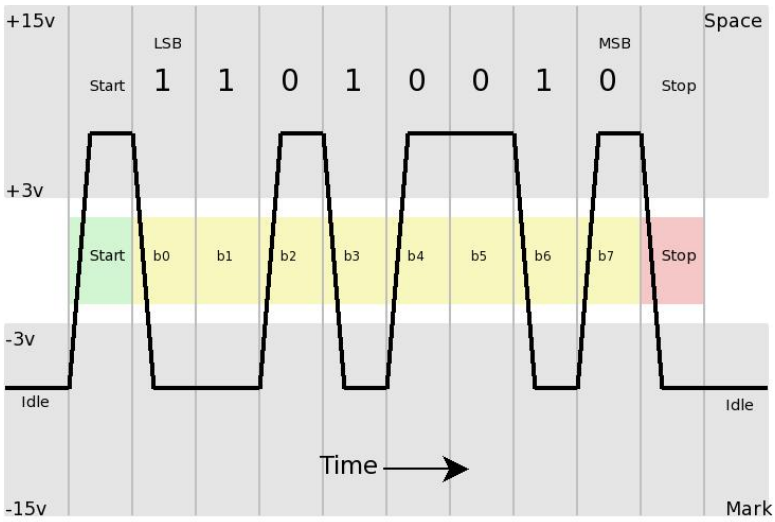


Figura 104

Il carattere inviato sulla linea seriale è delimitato da uno space (all'inizio con il bit di start) e da un mark (alla fine con un bit di stop). Tra questi delimitatori, detti anche dati d'inquadramento, sono inseriti i dati utili ed il relativo bit di parità.

12.4.1 Descrizione funzionale

L'UART realizza una conversione seriale/parallelo sui dati ricevuti dalla linea e una conversione parallelo/seriale su quelli ricevuti dal Microprocessore.

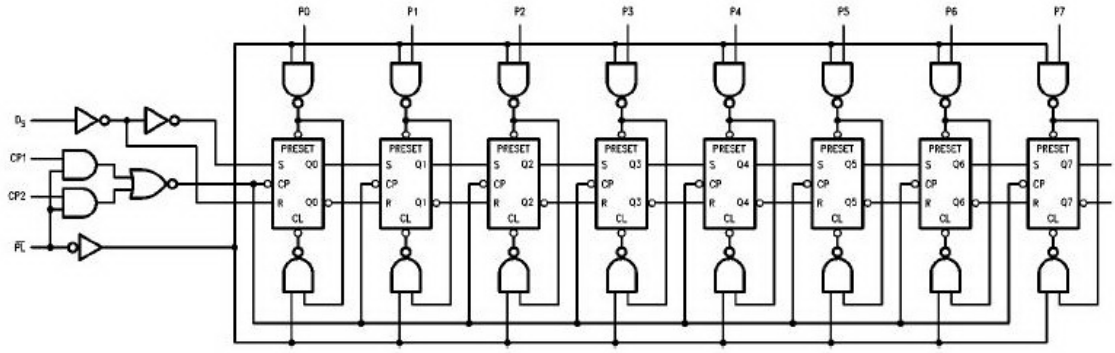


Figura 105

In ogni momento, il Microprocessore può leggere lo stato completo del dispositivo.
Tali informazioni riguardano lo stato delle operazioni di trasferimento dati e le possibili condizioni di errore.

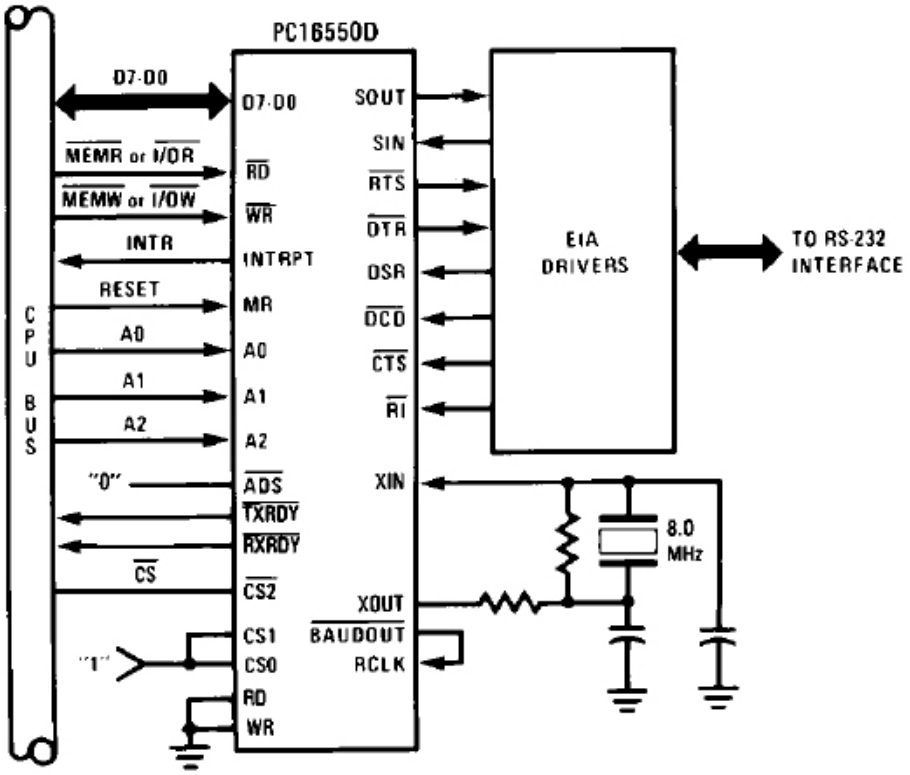


Figura 106

L'UART è costituita essenzialmente da quattro sezioni:

- Sezione di trasmissione: si occupa della costruzione del carattere da trasmettere serialmente e del formato della trasmissione;

- Sezione di ricezione: campiona opportunamente la linea dati e forma il byte che sarà trasferito verso il Microprocessore;
- Sezione di controllo: sovrintende alle operazioni ed elabora i segnali d'interrupt;
- Un insieme di registri di lettura e scrittura: comunica con il Microprocessore e configura le operazioni del dispositivo.

Per ridurre il sovraccarico di comunicazioni verso il Microprocessore sono presenti due memorie FIFO per la memorizzazione fino a 16 byte ricevuti o da trasmettere.

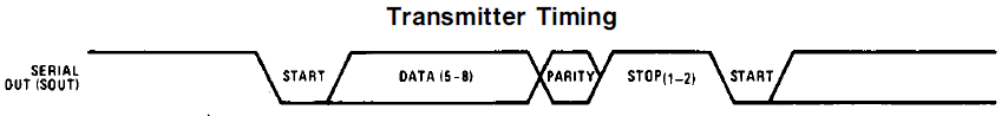


Figura 107

La ricezione del flusso seriale avviene in maniera asincrona, senza l'ausilio di un segnale di clock, pertanto è necessaria una sincronizzazione locale al clock che ha originato i dati. A tale scopo deve essere prevista una tolleranza sulla variazione di frequenza degli orologi, del trasmettitore e del ricevitore, entro il 10% per evitare variazioni di fase disastrose durante la trasmissione di un carattere. Tali variazioni di fase, se superano la durata di un bit, provocano acquisizione sbagliata del dato.

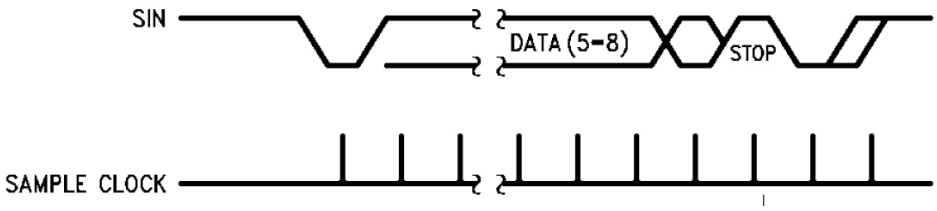


Figura 108

Per risolvere tali problemi di sincronizzazione viene utilizzato il bit di start.

In ricezione i circuiti dell'UART campionano il fronte di discesa del segnale che, con una certa probabilità, rappresenta il bit di start. Un'unità di temporizzazione locale (un contatore a 16 bit) predispone l'istante di campionamento a centro dato e, durante la ricezione dell'intero carattere, sfrutta le transizioni dei segnali per tenere l'istante di campionamento il più possibile al centro del dato.

E’ inoltre prevista una contromisura per gli eventi di rumore, sulla linea seriale, che provocano false rivelazioni del bit di start (false start bit detection).

12.4.2 Registri

I registri di configurazione sono divisi in registri di controllo, registri di stato e registri dati.
Dai registri di controllo è possibile controllare le operazioni dell’UART (compresa la trasmissione e la ricezione dei dati).

- La sezione di trasmissione utilizza due buffer di appoggio chiamati Transmitter Holding Register (THR, buffer cui il Microprocessore accede solo in scrittura) e Transmitter Shift Register (TSR) o PISO (Parallel In/Serial Out).
- La sezione di ricezione utilizza due buffer di appoggio chiamati Receiver Buffer Register (RBR, buffer cui il Microprocessore accede solo in lettura) e Receiver Shift Register (RSR) o SIPO (Serial-In/Parallel Out).

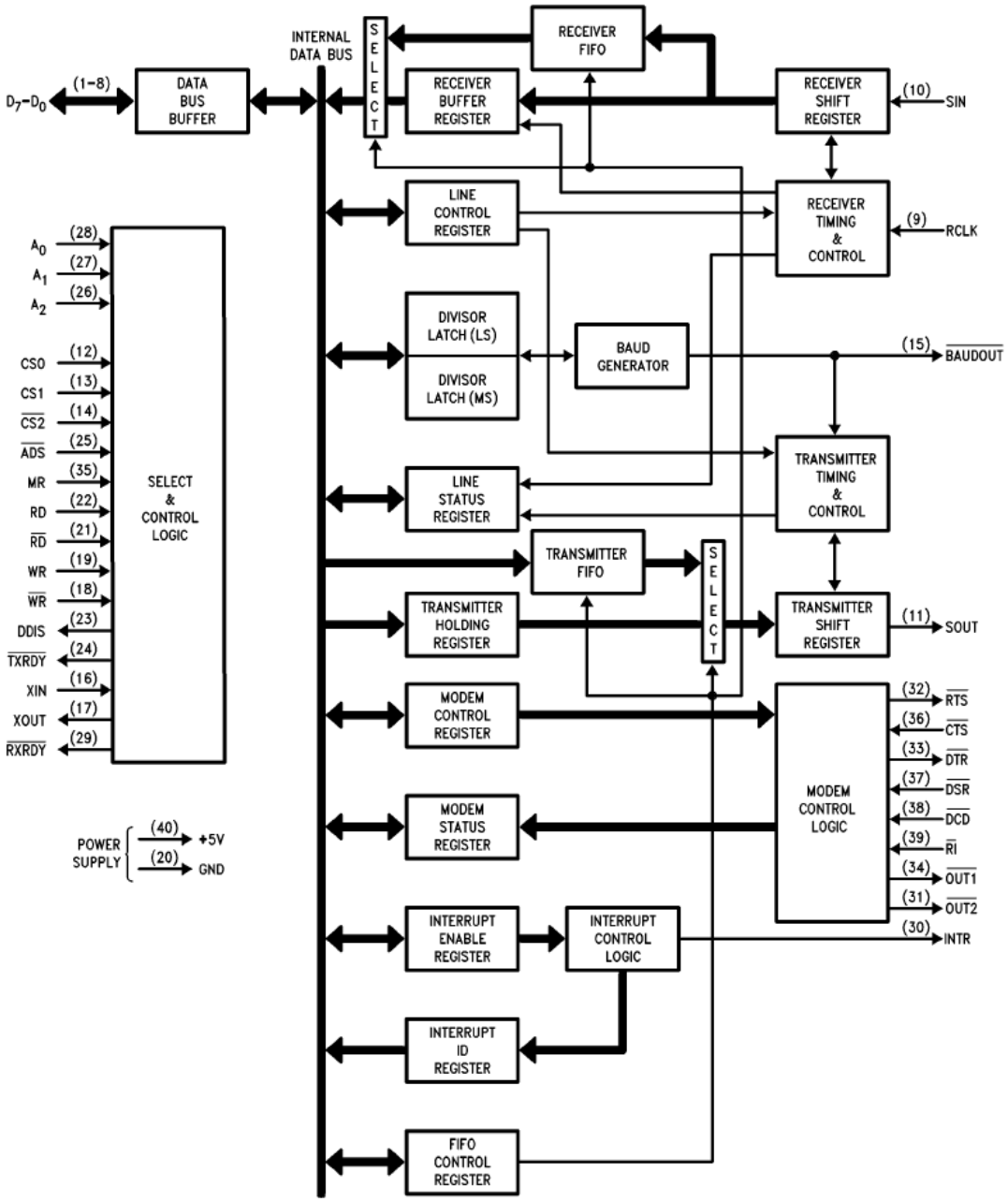


Figura 109

- Line Control Register (LCR): questo registro specifica il formato del dato asincrono, il numero di bit di dati previsti in un carattere trasmesso (5, 6, 7, 8 bit per carattere), il numero di stop bit (1 o 2), l'abilitazione alla funzione di parità (generazione in trasmissione e verifica in ricezione) e l'inserimento del break sulla linea dati (livello logico 0).

- Divisor Latch (DL): questi due registri servono a programmare il rapporto di divisione del clock di sistema (da cui dipende il baud rate). Si accede ad un registro o all'altro attraverso il bit numero 7 (DLAB) del precedente registro.

- Modem Control e Status Register: sono registri per controllare le funzionalità previste per i sistemi di trasmissione con modem. Sono previste le funzioni DTR (Data Terminal Ready), RTS (Request To Send), CTS (Clear To Send), DSR (Data Set Ready), RI (Ring Indicator), DCD (Data Carrier Detect).

- Scratchpad Register: registro generale di lettura/scrittura che serve solo al programmatore per memorizzare dati temporanei, non all'UART.

- FIFO Control Register: questo registro di sola scrittura è usato per controllare le FIFO.
  - Il bit 0 abilita le FIFO in trasmissione e in ricezione (in caso contrario il funzionamento avverrà con un carattere alla volta).
  - Il bit 1, se alto, resetta la FIFO di trasmissione e la relativa logica di gestione, ma non il registro di scorrimento di trasmissione.
  - Il bit 2, se alto, resetta la FIFO di ricezione e la relativa logica di gestione, ma non il registro a scorrimento di ricezione.

  Sia il bit 1 che il bit 2, quando attivati, azzerano il loro stato automaticamente da soli.
  - I bit 6 e 7 costituiscono la programmazione della soglia dei caratteri ricevuti in FIFO ("00"= 1 byte, "01"= 4 byte, "10"= 8 byte, "11"=14 byte).

- Line Status Register (LSR): registro di sola lettura che fornisce informazioni al Microprocessore, sullo stato del trasferimento dei caratteri. Ciascun'informazione di stato è memorizzata e trattenuta con il proprio valore fino a che il registro viene letto dal Microprocessore.
  - Il bit 0 si attiva quando è arrivato un carattere ed è stato trasferito nel Receiver Buffer Register (DR – Data Ready).
  - Il bit 1 si attiva quando il nuovo carattere giunto sovrascrive quello non ancora letto (OE – Overrun Error).
  - Il bit 2 indica errore di parità sui caratteri ricevuti (PE – Parity Error).

- Il bit 3 indica che il carattere ricevuto non ha uno stop bit valido (errore d'inquadramento), cioè c'è uno spazio al posto di un mark (FE – Framing Error).
- Il bit 4 si attiva quando i dati ricevuti in ingresso nello stato logico 0 (spazio) perdurano per un tempo maggiore o pari al tempo di un carattere, ovvero bit di start + dati + parità + bit di stop (BI – Break Indication).
- Il bit 5 si attiva quando l'UART è in grado di accettare un nuovo carattere da trasmettere, ovvero quando la FIFO di trasmissione è vuota (THRE – Transmitter Holding Register Empty).
- Il bit 6 indica che sia il Transmitter Shift Register che la FIFO sono vuoti (TEMT – Transmitter Empty).
- Il bit 7 si attiva quando nella FIFO è stato memorizzato un carattere con qualche errore di parità, framing o break indicator (RCVR FIFO error).

- Interrupt Identification Register: registro di sola lettura che ha lo scopo di ridurre l'impegno del Microprocessore verso il dispositivo UART. Segnala in ordine di priorità decrescente, e con opportuna codifica binaria dei bit da 3 a 0, gli eventi che riguardano la trasmissione o la ricezione dei caratteri. Le quattro condizioni d'interrupt sono denominate in ordine d'importanza:
  - Receiver Line Status (0110),
  - Receiver Data Available (0100) o Character Timeout Indication (1100),
  - Transmitter Holding Register Empty (0010),
  - Modem Status (0000).

  In assenza d'interrupt la codifica scelta è 0001. Quando il Microprocessore accede a questo registro, l'UART mostra la condizione prioritaria, pur registrando nuove condizioni d'interrupt. I bit 4 e 5 del registro non sono usati. I bit 6 e 7 sono al valore logico 1 se le FIFO sono abilitate.
  - L'interrupt di priorità maggiore (Receiver Line Status) accade per Overrun Error, Parity Error, Framing Error o Break Indication, pertanto occorre che il Microprocessore acceda al Line Status Register per individuarne la causa.
  - L'interrupt di tipo Receiver Data Available segnala che c'è un carattere (se la modalità FIFO è disabilitata) o è stato raggiunto il livello di riempimento, o trigger, preimpostato nella FIFO di ricezione; pertanto andranno letti i dati disponibili. La stessa priorità è data al timeout che segnala lentezza in lettura dei dati ricevuti da parte del Microprocessore.

- La condizione di terzo livello riguarda la disponibilità a trasmettere i dati giacché la FIFO di trasmissione è vuota.
- L'ultima riguarda il controllo del modem.

- Interrupt Enable Register: ciascun livello d'interrupt può essere abilitato con un bit del registro di abilitazione dell'interrupt (registro di sola scrittura) chiamato Interrupt Enable Register. L'abilitazione avviene ponendo il bit d'interesse a '1'.
  - Il bit 0 abilita il Received Data Available Interrupt (e il timeout interrupt nel FIFO mode).
  - Il bit 1 abilita il Transmitter Holding Register Empty Interrupt.
  - Il bit 2 abilita il Receiver Line Status Interrupt.
  - Il bit 3, che abilita il Modem Status Interrupt, non è utilizzato.

Tabella 10

Pinout

Signal	Signal Direction	Description	Activity
RS-232 Interface			
SIN	Input	Dato seriale in ingresso	
SOUT	Output	Dato seriale in uscita	
Micro Interface			
ADDR(2:0)	Input	Indirizzi stabili del Micro	
DATA(7:0)	Bidirectional	Dati da/verso il Micro	
INTR	Output	UART interrupt	High
RD	Input	Segnale di lettura	Low
WR	Input	Segnale di scrittura	Low
CS	Input	UART Chip Select	Low
General Interface			
RESET	Input	Reset asincrono	High
CKUART	Input	Clock di sistema (18.432 MHz)	

Tabella 11

Registri di configurazione

OFFSET	REG. NAME	DESCRIPTION	TYPE
0	RBR	Receiver Buffer Register	R
0	THR	Transmitter Holding Register	W
1	IER	Interrupt Enable Register	R/W
2	IIR	Interrupt Identification Register	R

2	FCR	FIFO Control Register	W
3	LCR	Line Control Register	R/W
4	MCR	Modem Control Register	R/W
5	LSR	Line Status Register	R
6	MSR	Modem Status Register	R
7	SCR	Scratch Register	R/W

13 B - Logiche e Standard per le trasmissioni elettroniche

13.1 Logica TTL

Un segnale TTL (Transistor-Transistor Logic) deve osservare specifiche determinate di tensione e corrente, sia come input che come output, e un tempo di rise/fall specifico.

Tabella 12

	Output	Input
V_L	0 to 0.4 V	0 to 0.8 V
V_H	2.4 to 5 V	2 to 5 V
I_L	16 mA	1.6 mA
I_H	400 mA	40 µA

Il massimo rise/fall time tra V_{IL} e V_{IH} è 50 ns.

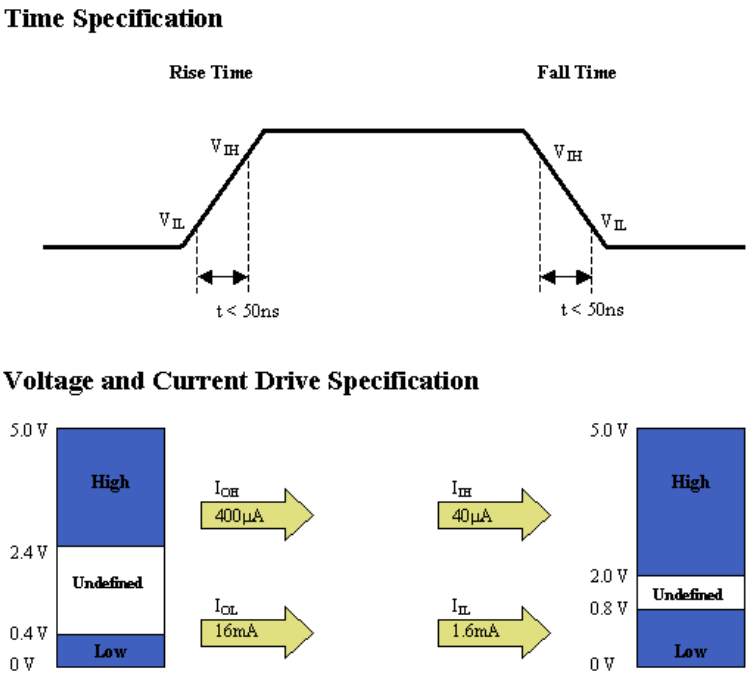


Figura 110

Il rapporto tra I_{OL} e I_{IL}, così come quello tra I_{OH} e I_{IH}, è pari a 10, come conseguenza dello standart Fan-out per la logica TTL.

13.2 Logica LVDS

Low-Voltage Differential Signaling, definita in ANSI/TIA/EIA-644-A, è una tecnologia di trasmissione dati bilanciata.

Proprietà:

- Basso consumo energetico;
- Alta velocità di segnalazione;
- Alta immunità al rumore;
- Basso rumore di commutazione interno.

Un tipico driver LVDS è composto da un generatore di corrente (3.5 mA). Poiché l'impedenza in ingresso al ricevitore è grande tutta la corrente scorre attraverso la terminazione in ingresso a 100 Ω, producendo dunque una tensione nominale pari a 350 mV ai capi del ricevitore.

La soglia del ricevitore è garantita intorno ai 100 mV, per tensioni di modo comune che vanno da 0 a 2.4 V.

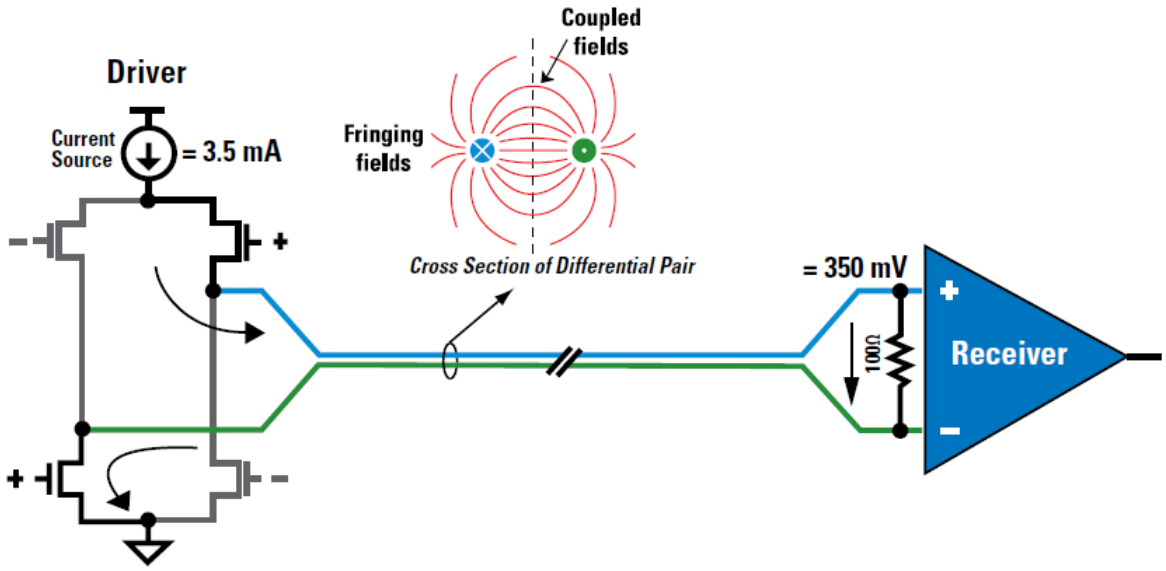


Figura 111

Cambiando la direzione della corrente, ai capi del ricevitore si ottiene la stessa tensione ma a polarità inversa: i valori logici 0 e 1 sono generati in questo modo.

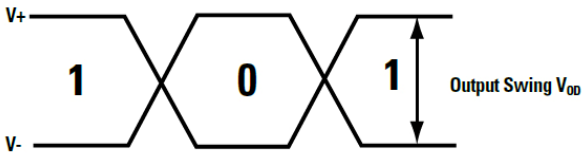


Figura 112

Lo swing tipico dei segnali LVDS, pari a 350 mV: fa consumare poca potenza ai dispositivi elettronici, rendendo LVDS una tecnologia molto efficiente, con capacità di trasmissione fino a 3.125 Gb/s.

Esistono due topologie fondamentali di interconnessione per dispositivi a tecnologia LVDS:

- Point-to-Point: coinvolge solo un trasmettitore e un ricevitore

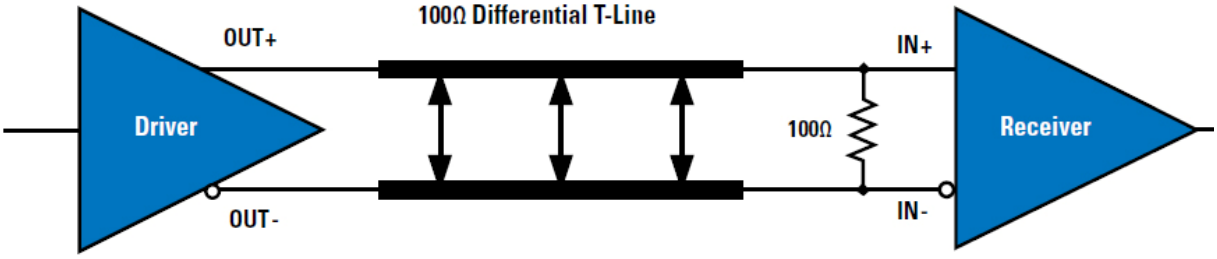


Figura 113

- Multipoint/Multidrop: sono presenti diversi trasmettitori e ricevitori, tutti sullo stesso bus.

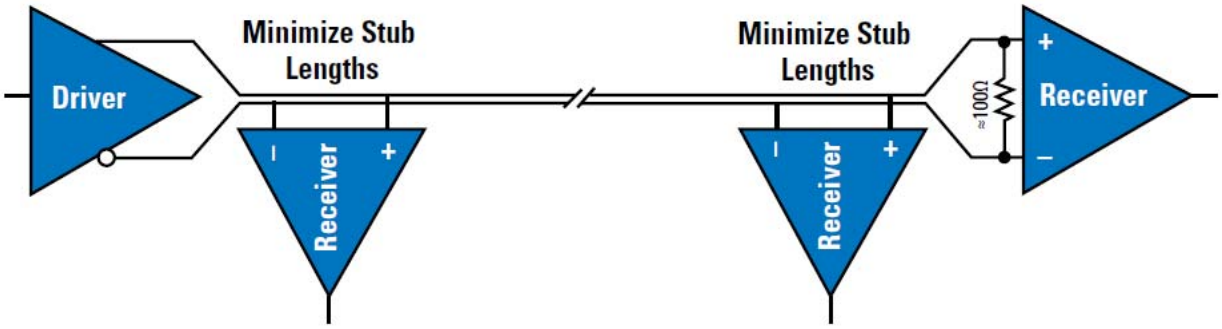


Figura 114

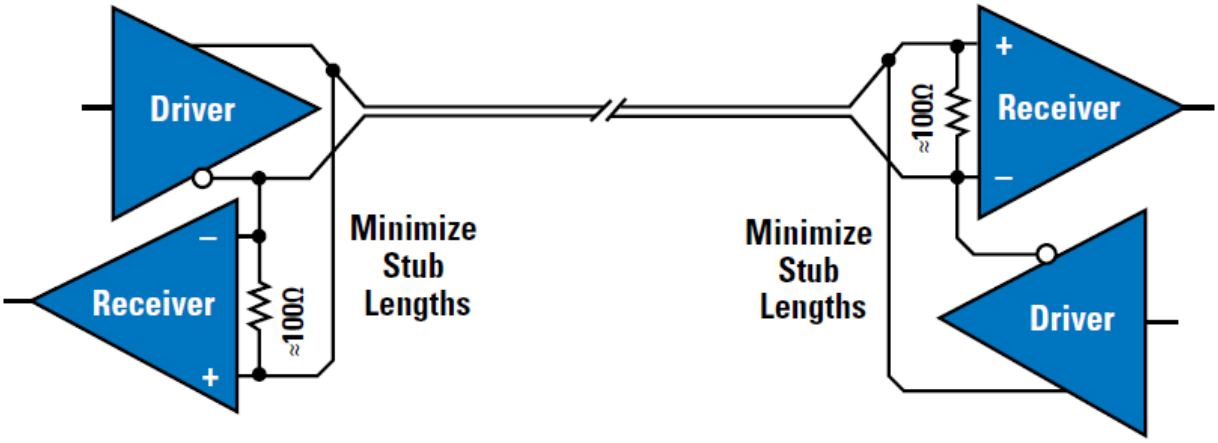


Figura 115

13.3 Logica NIM

NIM era inteso come l'acronimo per Nuclear Instrument Modules; successivamente, a causa del suo largo utilizzo anche in altri campi applicativi oltre che alla fisica nucleare, fu ribattezzato in National Instrumentation Methods.
Oggi NIM identifica sia il sistema di apparecchiature modulari standardizzate, definite nel documento, che la commissione responsabile al mantenimento dello standard.

I segnali logici ed analogici sono definiti da due specifiche diverse:

- logica positiva, molto simile ai livelli di tensione della logica TTL;
- logica negativa, di seguito riportata, che viene preferita nel campo delle applicazioni nucleari per segnali logici molto veloci.

Tabella 13

	Output Driver Current [mA] into 50 Ω	Receiver Input Voltage [V] Response
Logic Low	-1 to +1	-0.2 (min) to +1 (max)
Logic High	-14 to -18	-0.6 (max) to -1.8 (min)

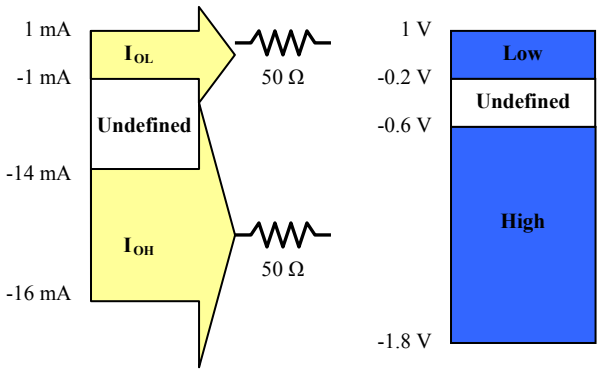


Figura 116

13.4 Trasmissione RS 232

Lo standard EIA RS-232 prevede, per la trasmissione di un singolo byte di informazioni, la trasmissione sul mezzo di comunicazione di un frame composto da 11 bit:

- Start bit: valore logico 0;
- 8 bit di dati;
- Parity bit: (pari, dispari, indifferente, nessuno) funzione dei primi 9 bit inviati;
- Stop bit: valore logico 1.

Tabella 14

	Voltage [V]
Logic Low	-15 (min) to -3 (max)
Logic High	+3 (min) to +15 (max)

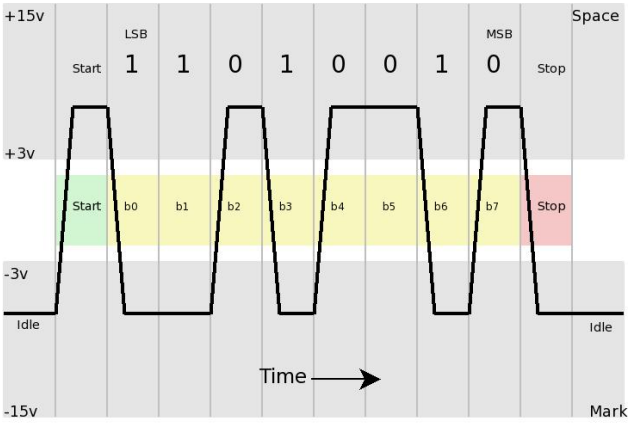


Figura 117

I dispositivi che utilizzano la trasmissione seriale vengono divisi in due categorie: Data Communications Equipment (DCE) e Data Terminal Equipment (DTE).
I DCE sono dispositivi come modem, plotter o terminal adapter (connettore femmina), mentre con DTE ci si riferisce a computer o terminali (connettore maschio).
RS-232 utilizza una trasmissione dati di tipo non bilanciato: il valore logico trasmesso è codificato dal valore della tensione della linea riferita alla massa; la trasmissione è monodirezionale.

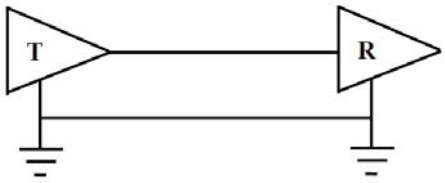


Figura 118

I limiti di questo standard sono:

- Lunghezza massima cavi: 15 metri;
- Bit Rate massimo: 20 Kb/s;
- Numero Trasmettitori: 1;
- Numero Ricevitori: 1.

Generalmente i dispositivi elettronici utilizzano una fonte di alimentazione che non fornisce i valori elettrici necessari alla comunicazione secondo lo standard (± 12 V), sono dunque necessari dei convertitori di livello aggiuntivi che vengono posti tra il modulo UART e i cavi di comunicazione.

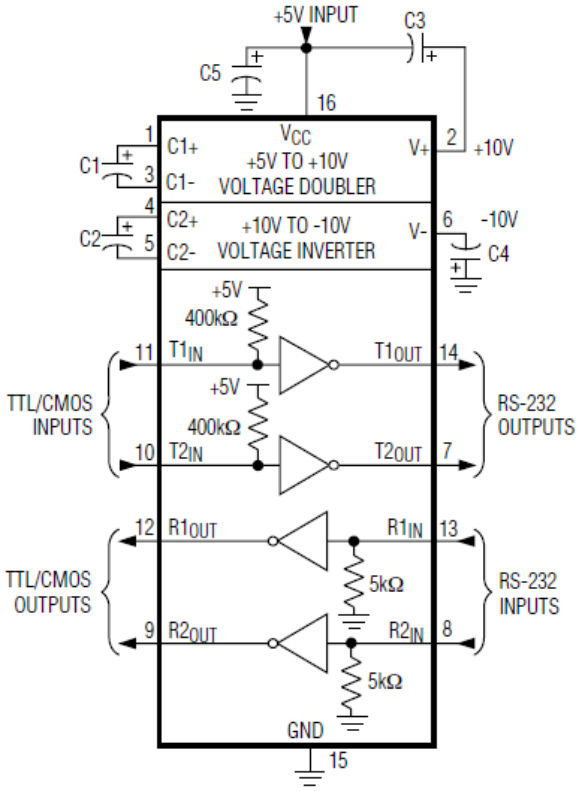


Figura 119

MAXIM MAX232

I cavi seriali RS-232 hanno due tipi di connettori: connettore a 25 pin e connettore a 9 pin, entrambi utilizzano gli stessi segnali di seguito definiti:

- TD: Transmit Data. Output seriale (TXD);
- RD: Receive Data. Input seriale (RXD);
- CTS: Clear To Send. Indica che il modem è pronto a scambiare dati;
- DCD: Data Carrier Detect. Il modem attiva il segnale quando rileva una portante dal modem remoto;
- DSR: Data Set Ready. Indica alla UART che il modem è pronto a stabilire una connessione;
- DTR: Data Terminal Ready. Indica al modem che la UART è pronta a stabilire una connessione;
- RTS: Request To Send. Informa il modem che la UART è pronta a scambiare dati;
- RI: Ring Indicator. Attivo quando il modem rileva un segnale di squillo dal PSTN;
- SG: Signal Ground.

Tabella 15

Signal	D type 9 pin #	D type 25 pin #
TD	3	2
RD	2	3
RTS	7	4
CTS	8	5
DSR	6	6
SG	5	7
CD	1	8
DTR	4	20
RI	9	22

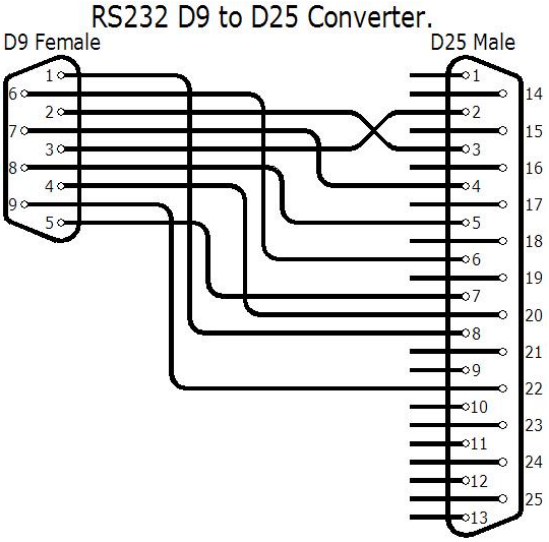


Figura 120

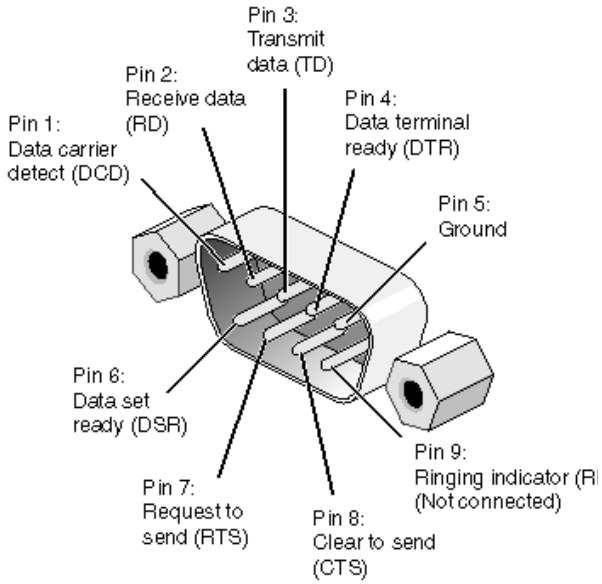


Figura 121

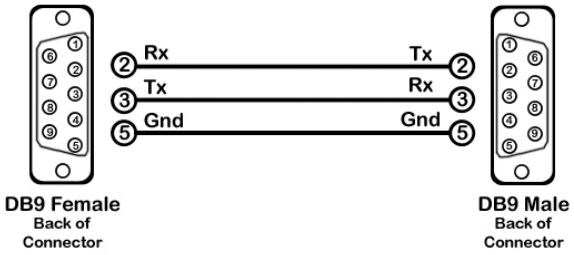


Figura 122

Il collegamento dei fili conduttori, per un cavo seriale standard tra DTE (connettore maschio) e DCE (connettore femmina), si effettua semplicemente collegando punto punto i rispettivi pin.

Quando si devono collegare due dispositivi DTE o due DCE si utilizzano dei cavi, detti Null Modem, in cui i segnali sono collegati in modo da far corrispondere trasmissione e ricezione.

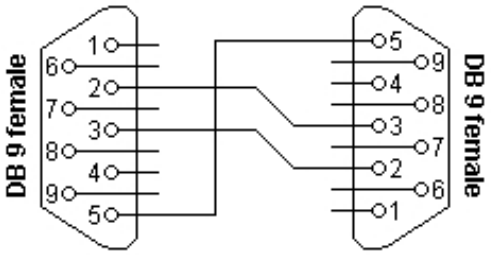


Figura 123

Nella versione più semplice di cavo seriale, sia standard che null modem, vengono usati solo tre fili conduttori per i seguenti segnali: TXD, RXD, GND.

13.5 Trasmissione RS 422

Definita nello standard EIA RS-422-A, utilizza una tipologia di trasmissione differenziale ed è stata ideata per permettere la connessione di più dispositivi riceventi allo stesso trasmettitore.

Vengono utilizzati due conduttori per la trasmissione di un segnale, un doppino intrecciato: il valore logico trasmesso viene codificato dalla differenza delle tensioni sulle linee.

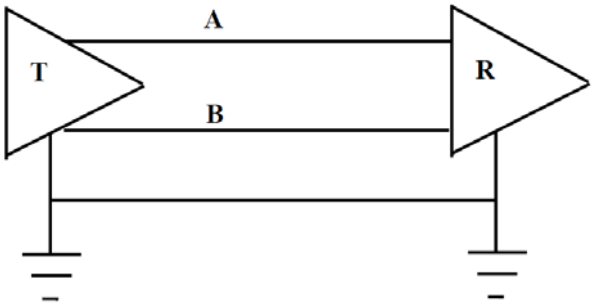


Figura 124

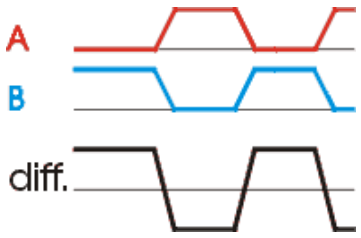


Figura 125

Tabella 16

	V_{AB} [V]
Logic Low	0.2 (min) to 6 (max)
Logic High	-0.2 (min) to -6 (max)

Poichè la variazione di stato del dato è determinata dalla differenza delle tensioni sui due fili, che sono intrecciati e seguono lo stesso percorso, un rumore elettrico o disturbo, ripercuotendosi su entrambi i conduttori, non altera la tensione relativa fra questi. Si ottiene così un'alta immunità ai disturbi.

I vantaggi offerti dalla codifica differenziale sono:

- Maggiore immunità ai disturbi rispetto la trasmissione non bilanciata;
- Lunghezza massima cavi: 1200 metri;
- Bit Rate Massimo: 10 Mb/s;
- Numero Trasmettitori: 1;
- Numero Ricevitori massimo: 10.

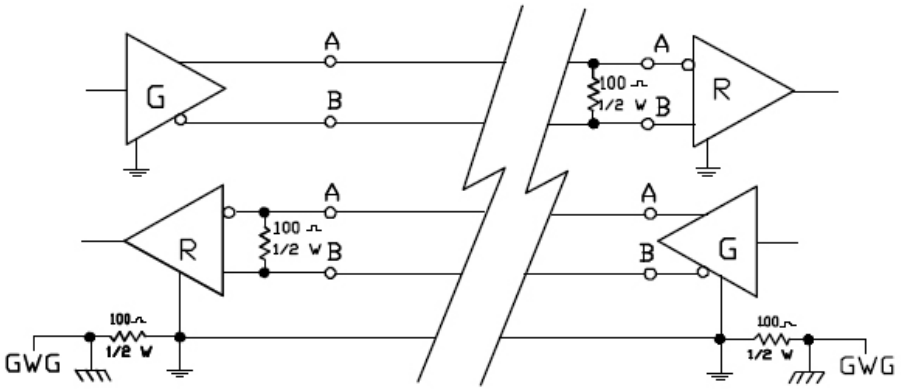


Figura 126

Per adattare i livelli di tensione, da TTL/CMOS allo standard RS-422, è necessario dotare il circuito di comunicazione di dispositivi che effettuino la generazione dei segnali elettrici adeguati.

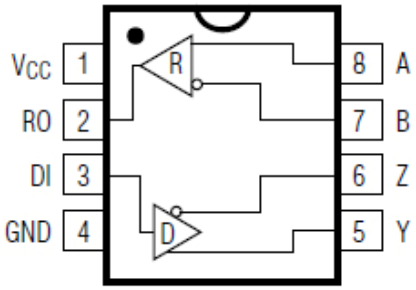


Figura 127

MAXIM MAX488

13.6 Trasmissione RS485

Definita dallo standard EIA RS-485, utilizza una modalità di trasmissione dati differenziale (come in RS-422) ma a differenza di quest'ultima, che ha singolo circuito di pilotaggio che non può essere spento, il trasmettitore viene messo in modalità di trasmissione in modo esplicito, applicando un segnale di enable.

Questa peculiarità permette il collegamento di più dispositivi trasmettitori sullo stesso bus, fino ad un massimo di 32, che dovranno solamente essere coordinati nell'accesso alla trasmissione.

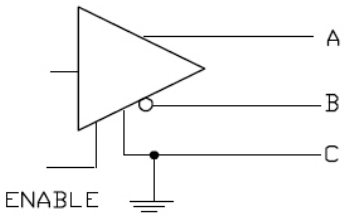


Figura 128

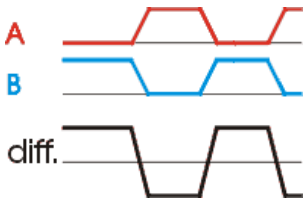


Figura 129

Un collegamento RS-485 può essere reso full duplex, come su RS-422, utilizzando due coppie di conduttori intrecciati, una per la trasmissione e una per la ricezione; in questo caso si parla di collegamento RS-485 a 4 fili.

In alcuni casi è possibile la compatibilità tra i due protocolli, ad esempio nel caso in cui si utilizzi un protocollo di tipo polling/selecting e il master sia un trasmettitore RS-422 e gli slave siano tutti ricevitori RS-485.

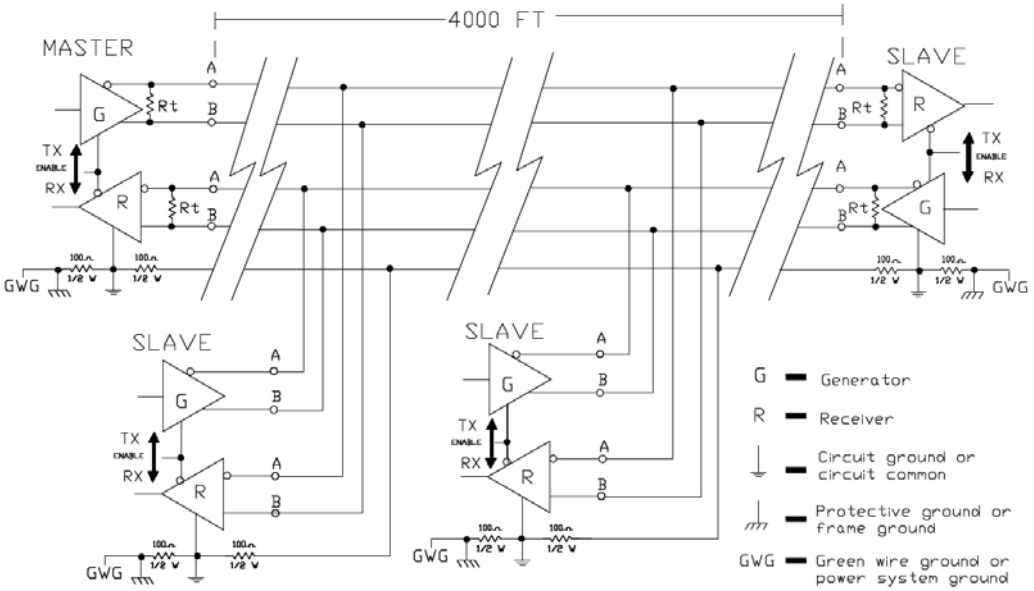


Figura 130

E' possibile inoltre risparmiare una coppia di conduttori intrecciati poiché, grazie alla disabilitazione del driver di trasmissione, si può imporre la trasmissione di un dispositivo per volta sul bus condiviso; in questo caso si avrà una rete RS-485 half-duplex a 2 fili.

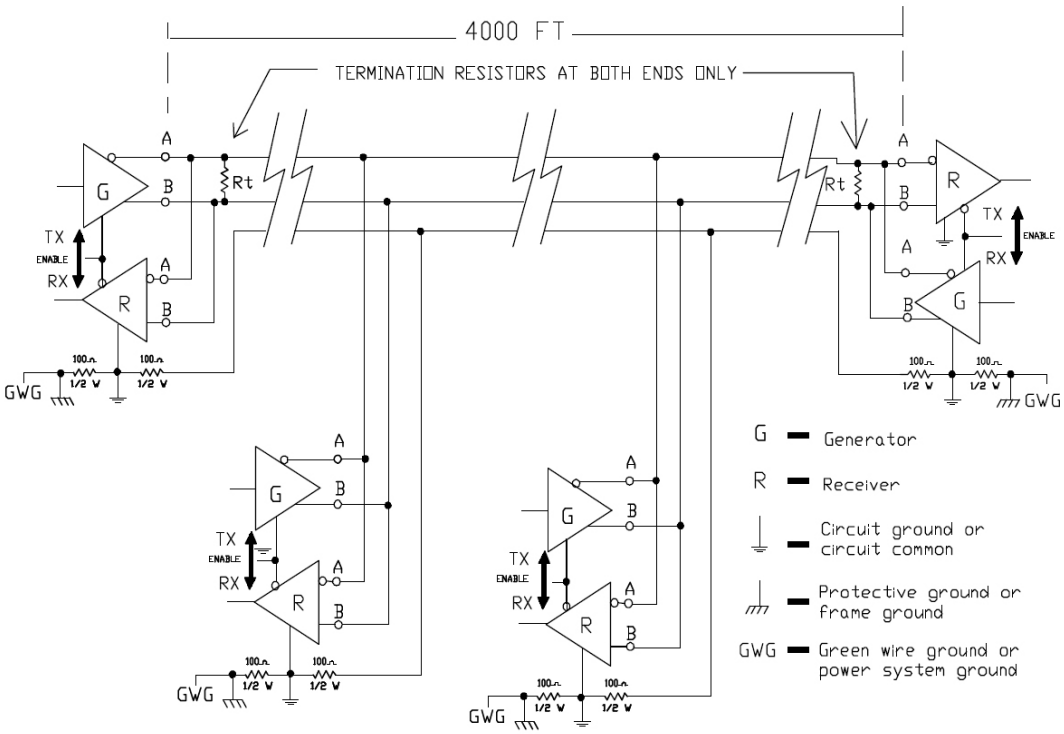


Figura 131

Per adattare i livelli di tensione, da TTL/CMOS allo standard RS-485, è necessario dotare il circuito di comunicazione di dispositivi che effettuino la generazione dei segnali elettrici adeguati.

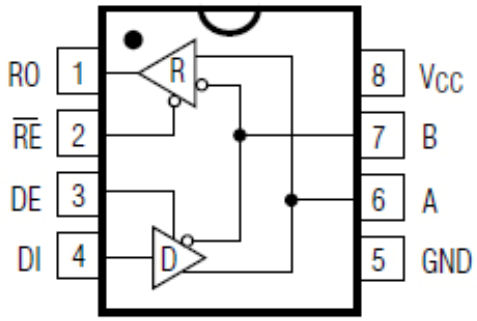


Figura 132

MAXIM MAX485

14 C - Domini temporali, divisori di frequenza, logica combinatoria

14.1 Clock

14.1.1 Clock Asincroni

Il modo più semplice per generare un clock si ottiene utilizzando un oscillatore, che genera una singola frequenza di output per un singolo componente. Gli oscillatori sono spesso utilizzati per applicazioni asincrone: ogni oscillatore provvede a fornire una risorsa locale e i domini temporali sono indipendenti. Le operazioni tra dispositivi in questo caso necessitano di avere frequenze di riferimento molto prossime, non necessariamente identiche. Questa architettura è ideale per tipi di traffico discontinuo (burst-mode), comunicazioni continue richiedono in questo caso lo stuffing di bit o pacchetti e l'utilizzo di code FIFO per prevenire le condizioni di overflow/underflow.

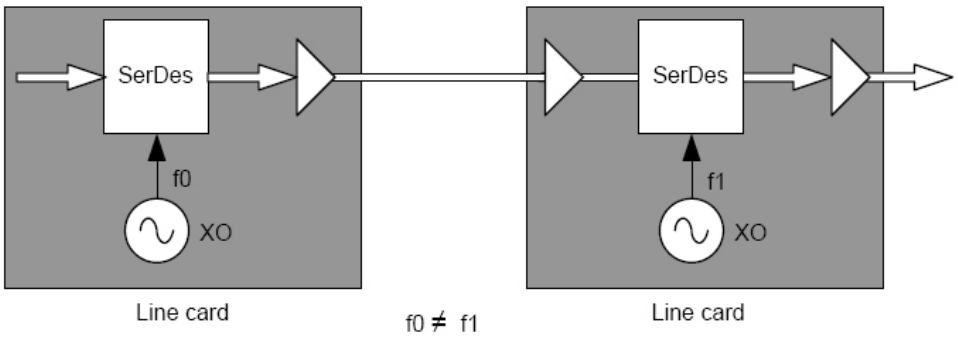


Figura 133

14.1.2 Clock sincroni

Il sistema sincrono è più utilizzato in applicazioni che richiedono un tipo di comunicazione continuata nel tempo. La latenza della rete di comunicazione e la sua variazione sono dei parametri da tenere sotto controllo e da minimizzare quanto più possibile. Per venire incontro a queste esigenze è necessario che sorgente e destinazione lavorino alla medesima frequenza.

Dal lato trasmettitore i clocks che definiscono il dominio temporale per la trasmissione sono agganciati a dei clock di riferimento (primario e secondario) molto accurati. Vengono utilizzati dei PLL per l'aggancio a questo riferimento sul backplane, attenuare il jitter sul segnale per ridurre il rumore e quindi fornire un clock in uscita, con basso jitter, al livello fisico.
Dal lato ricevitore viene utilizzato un Clock and Data Recovery (CDR) per recuperare il segnale di clock dalla trasmissione ricevuta. Il clock estratto passa attraverso un altro PLL che divide la frequenza ad un rate più basso. Il clock locale può essere sincronizzato a questo appena ottenuto o ad un altro clock locale sincronizzato con quello centrale, condizione che assicura la sincronizzazione di tutti i nodi presenti sulla rete. [TECHa]

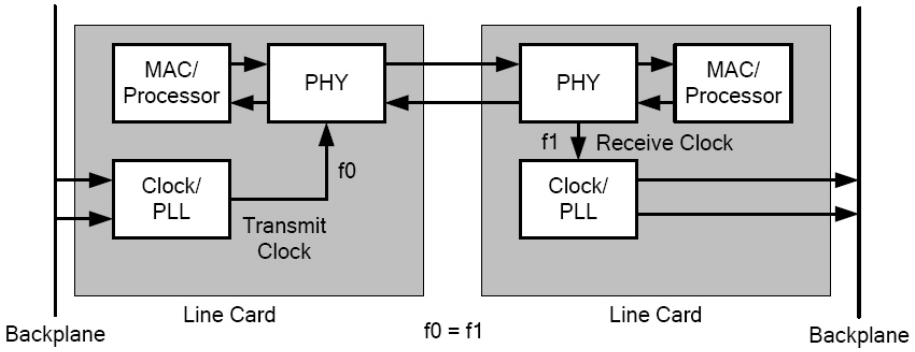


Figura 134

Il sistema sviluppato è di tipo asincrono, i dispositivi lavoreranno con un riferimento di clock locale, fornito dagli oscillatori presenti sulla board, e comunicheranno con l'esterno con una trasmissione seriale asincrona.
Sulla board sono presenti 3 oscillatori con frequenze pari a 50, 27 e 24 MHz. E' pure possibile introdurre un segnale di clock dall'esterno ma non è stato fatto.

14.2 Flip-flop

I flip-flop sono circuiti elettronici sequenziali, utilizzati in elettronica digitale come dispositivi di memoria elementare.

14.2.1 Flip-flop SR

Ha due ingressi (Set, Reset) e due uscite complementari (Q, –Q).
E' una rete sequenziale asincrona che si evolve in accordo alle seguenti specifiche:

- quando lo stato d'ingresso è S=1 e R=0, il flip-flop si setta, cioè porta a 1 il valore della variabile di uscita Q e 0 la variabile –Q;
- quando lo stato d'ingresso è S=0 e R=1, il flip-flop si resetta, cioè porta a 0 il valore della variabile di uscita Q e a 1 il valore di –Q;
- quando lo stato d'ingresso è S=0 e R=0, il flip-flop conserva i valori di entrambe le variabili di uscita.

Quando entrambi i valori logici S e R sono bassi, agisce come elemento di memoria e mantiene in uscita i valori precedentemente memorizzati. Esiste il caso in cui entrambi gli ingressi hanno valore logico alto (S=1, R=1) e l'uscita non è definita.
La realizzazione circuitale è molto semplice, può essere fatta con due porte NAND o NOR.

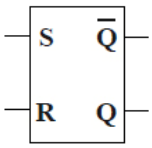


Figura 135

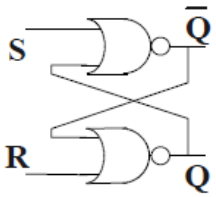


Figura 136

Tabella 17

S	R	Q	-Q
1	0	1	0
0	1	0	1
0	0	Q	-Q
1	1	0	0

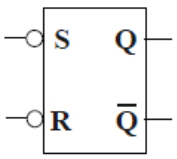


Figura 137

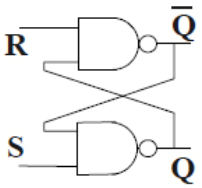


Figura 138

Tabella 18

S	R	Q	-Q
0	1	1	0
1	0	0	1
1	1	Q	-Q
0	0	1	1

14.2.2 Flip-flop JK

Ha due ingressi (Set, Reset), un ingresso di sincronizzazione e due uscite complementari (Q, –Q).

A differenza del flip-flop SR non ha lo stato indefinito, rimpiazzato dallo stato *toggle mode*.

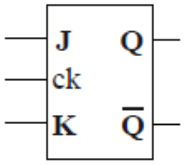


Figura 139

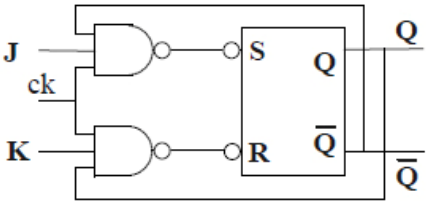


Figura 140

Tabella 19

S	R	Q_{n+1}
1	0	Q_n
0	1	0
0	0	1
1	1	$-Q_n$

E’ possibile aggiungere Preset e Clear asincroni

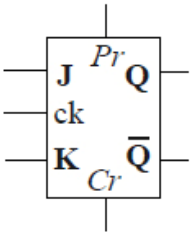


Figura 141

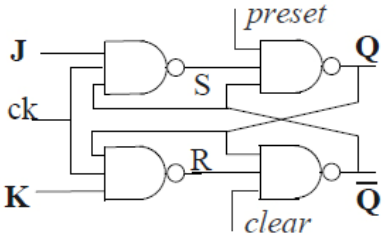


Figura 142

14.3 Implementazione di Latch e Flip-Flop in VHDL

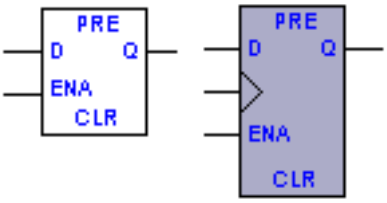


Figura 143

Entrambi i registri ripropongono in uscita il valore presente sull'input, se la condizione di enable è attiva. La differenza fondamentale tra i due registri sta nel fatto che i latches non sono regolati da alcuno stimolo di clock (appartengono ad un dominio temporale a se stante), i flip-flop controllano il valore in input ogni ciclo di clock (sono sincroni a uno specifico dominio temporale) e non presentano lo stato non definito che caratterizza i latches.
Non è consigliabile l'utilizzo indiscriminato di latches nell'architettura di un sistema elettronico, ma è opportuno che ogni registro sia arbitrato da un clock.
Codice VHDL per Latch:

```
ENTITY D_latch IS
  PORT(
    data_in  : IN  STD_LOGIC;
    enable   : IN  STD_LOGIC;
    data_out : OUT STD_LOGIC
  );
END D_latch;

ARCHITECTURE behv OF D_latch IS
  BEGIN
    PROCESS(data_in, enable)
      BEGIN
        IF (enable='1') THEN
           data_out <= data_in;
        END IF;
    END PROCESS;
END behv;
```

Listato 68

Codice VHDL per Flip Flop:

```
ENTITY dff IS
  PORT(
    data_in  : IN  STD_LOGIC;
    clock    : IN  STD_LOGIC;
    data_out : OUT STD_LOGIC
  );
END dff;

ARCHITECTURE behv OF dff IS
  BEGIN
    PROCESS(data_in, clock)
      BEGIN
        IF (clock'EVENT AND clock='1') THEN
          data_out <= data_in;
        END IF;
    END PROCESS;
END behv;
```

Listato 69

14.4 Domini temporali

Un segnale trasferito tra circuiti appartenenti a domini temporali differenti rende necessaria la sincronizzazione al nuovo dominio temporale, prima di essere utilizzato con successo.
Per minimizzare i fallimenti dovuti alla metastabilità, nel trasferimento di segnali asincroni, è pratica comune utilizzare una sequenza di registri al fine di risincronizzare il segnale in input al dominio temporale di destinazione. Tali registri permettono un tempo addizionale, ad un eventuale segnale metastabile, per raggiungere uno stato logico definito, prima di essere utilizzato nel resto del percorso circuitale.

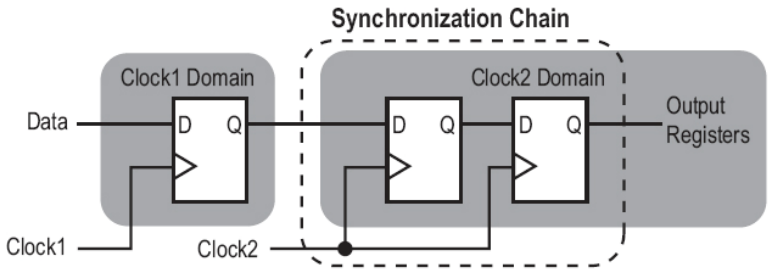


Figura 144

Un *synchronization register chain* (o *synchronizer*) è definito come una sequenza di registri che possiedono i seguenti requisiti:

- i registri della catena sono tutti comandati dallo stesso clock o da clock in fase tra loro;
- il primo registro della catena è comandato da un altro clock non correlato, o da un input asincrono;
- ogni registro pilota solamente un altro registro, eccetto l'ultimo della catena.

Il sistema progettato comprende tre differenti domini temporali, ciascuno dei quali racchiude la logica combinatoria necessaria alla registrazione dei dati in ingresso, che su richiesta saranno impacchettati e trasferiti all'esterno. Possiamo definire tali domini come segue:

- Dominio esterno: comprende la logica dei contatori che sono direttamente connessi ai segnali da registrare. Il sistema deve essere in grado di registrare sia eventi periodici che aperiodici, ciò impone una capacità di campionamento adeguata alla frequenza

Il primo registro è un contatore assoluto che incrementa il proprio valore di una unità per ogni segnale di input esterno.

Il secondo è un registro su cui viene riversato il contenuto del contatore precedente, nel momento in cui viene richiesto il dato attuale registrato. Questo passaggio permette ai due registri di poter lavorare indipendentemente poichè il secondo registro appartiene ad un dominio temporale differente, concorde alla richiesta dati che può essere periodica o aperiodica.

Il terzo registro conserva il penultimo dato registrato sul contatore: la differenza tra i valori contenuti nel secondo e terzo registro fornirà in uscita il numero di conteggi tra la richiesta dati attuale e quella precedente.

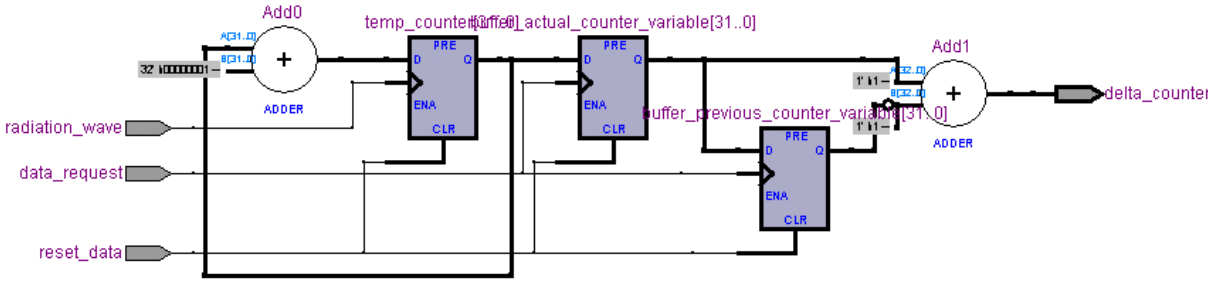


Figura 145

- Dominio del Timestamp: il funzionamento è analogo a quello del contatore ma con un passaggio in più. Il registro in ingresso viene incrementato di una unità per ogni evento di clock, che questa volta è di frequenza costante e definita (pari a 1 kHz).

  Ogni 1000 eventi di clock, dunque ogni secondo, verrà generato un segnale di enable per il registro successivo, che incrementerà il conteggio di una unità.

  Nell'istante in cui verrà richiesto il dato attuale, la logica rimanente fornirà in uscita sia il timestamp assoluto che la differenza di tempo, tra la richiesta attuale e quella precedente, entrambi espressi in secondi.

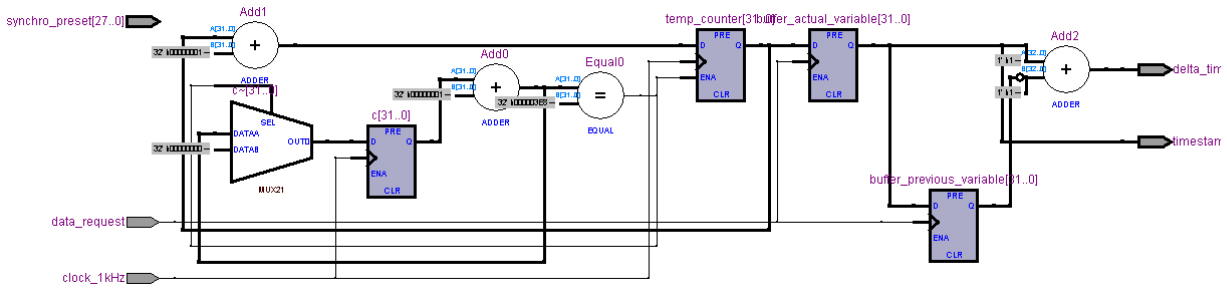


Figura 146

- Dominio della UART: tutta la logica combinatoria rimanente appartiene al dominio temporale della trasmissione UART, perchè i tempi di trasmissione su bus seriale dirigono tutti i passaggi di dati tra registri.
  Ogni registro accetta come clock lo stesso clock che pilota la trasmissione UART, in modo da permettere lo scambio dei dati sincronizzato ad essa.

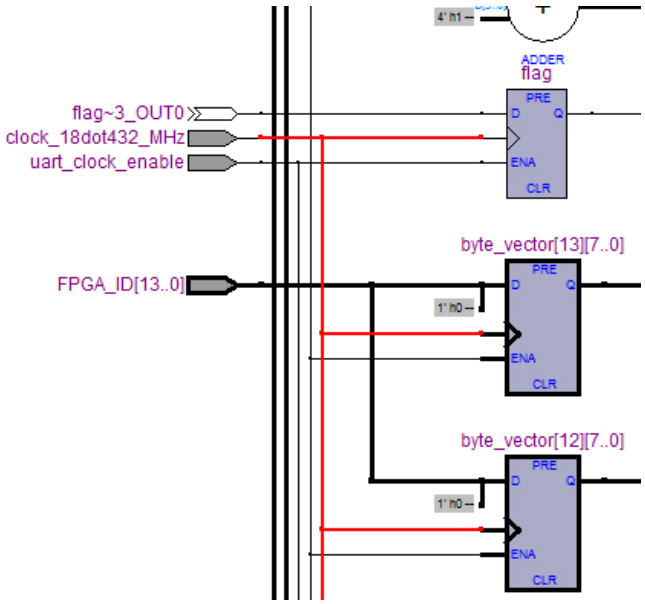


Figura 147

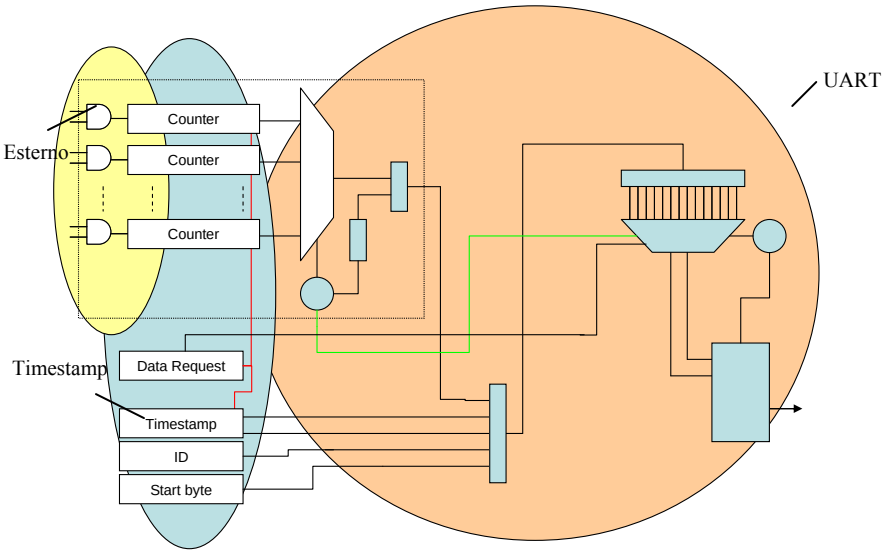


Figura 148

14.5 Divisori di frequenza

Dai circuiti oscillatori, presenti sulle boards elettroniche, è possibile ricavare le frequenze più opportune per il sistema in questione, realizzando dei divisori di frequenza che danno in uscita il valore desiderato partendo da quello imposto dalle specifiche hardware.

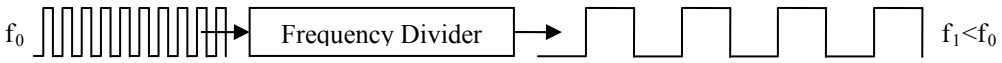


Figura 149

In VHDL è molto semplice scalare una frequenza.

Ad esempio se vogliamo generare un clock a 1 kHz partendo da uno a 24 MHz, è necessario imporre un "rallentamento" pari a 24000 volte.

Per fare questo si utilizza un flip flop che ogni 41.6 ns (ciclo di clock di 24 MHz) incrementa un contatore. Tale contatore, giunto a 12000 (la metà di 24000, se si vuole duty cycle 50%), viene azzerato e ricomincia a contare, mandando un segnale ad un altro flip flop che nega la logica dell'uscita attualmente attiva.

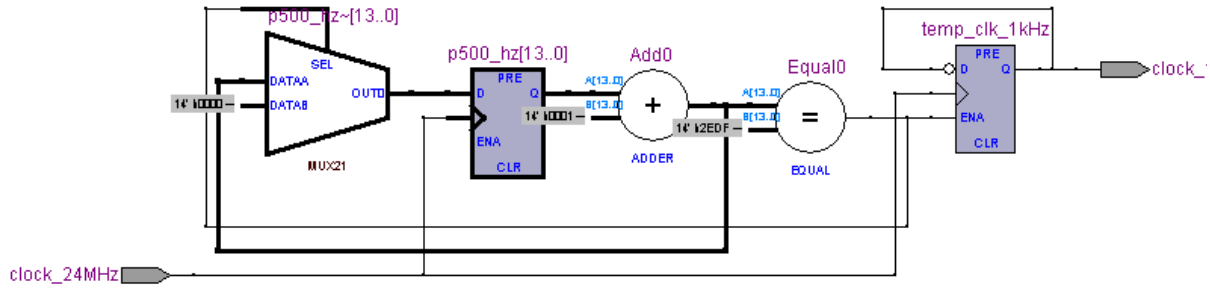


Figura 150

Codice VHDL per divisore di frequenza:

```
ENTITY clock_divider_24MHz_to_1kHz IS
  PORT(
    clock_24MHz: IN STD_LOGIC;
    clock_1kHz:  OUT STD_LOGIC
  );
END clock_divider_24MHz_to_1kHz;
---------------------------------------
ARCHITECTURE arch OF clock_divider_24MHz_to_1kHz IS
  SIGNAL temp_clk_1kHz:       STD_LOGIC;

  BEGIN
  PROCESS(clock_24MHz)
    -- 24 MHz / 24 000 = 1 kHz, devo prendere la metà
    VARIABLE p500_hz:   INTEGER RANGE 0 TO 11999 := 0;

    BEGIN
      IF(clock_24MHz'EVENT AND clock_24MHz = '1') THEN
        p500_hz := p500_hz + 1;

          -- quando completa il conteggio
          IF(p500_hz = 11999) THEN

            -- cambia lo stato logico dell'uscita
            temp_clk_1kHz <= NOT temp_clk_1kHz;
            -- azzera il contatore
            p500_hz := 0;

          END IF;
        END IF;
  END PROCESS;

  clock_1kHz <= temp_clk_1kHz;
```

```
END arch;
```

Listato 70

Per il generatore di frequenza adatto a dirigere la comunicazione UART bisogna fare una premessa.

Secondo le specifiche del modulo UART è necessario avere un ingresso un clock con frequenza pari a 18.432 MHz dal quale poi scalare la frequenza per il baud rate desiderato. Questo valore di divisione è contenuto in due registri dedicati: Divisor Latch MSB e LSB. Il clock scalato sarà ulteriormente rallentato di un fattore 16 e da quest'ultima divisione si avrà il valore effettivo di frequenza desiderata. [NS05]

Dal punto di vista hardware basta connettere l'oscillatore e impostare i byte al valore scelto.

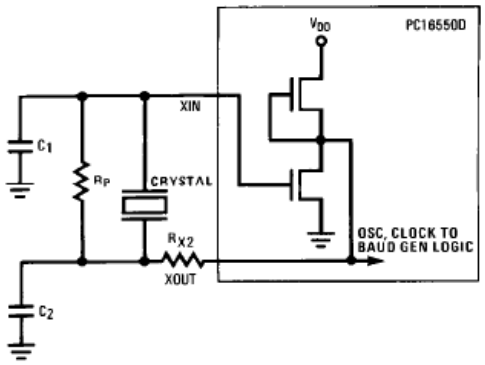


Figura 151

Il valore 18.432 MHz non è scelto a caso ma rappresenta una frequenza divisibile per valori interi che produce in uscita un basso errore percentuale. Mettendo a confronto i valori di divisione necessari al raggiungimento della frequenza desiderata tra il clock a 18.432 MHz con quello a 24 MHz si ottiene la seguente tabella

Tabella 20

	18.432 MHz Crystal			24 MHz Crystal		
Baud Rate bit/s	Divisor for 16 x Clock		Error %	Divisor for 1 x Clock		Error %
	Dec	Hex		Dec	Hex	
50	23040	5A 00-h	-	480000	> FF FF-h	-

150	7680	1E 00-h	-	160000	> FF FF-h	-
300	3840	0F 00-h	-	80000	> FF FF-h	-
600	1920	07 80-h	-	40000	9C 40-h	-
1200	960	03 98-h	-	20000	4E 20-h	-
2400	480	01 E0-h	-	10000	27 10-h	-
4800	240	00 F0-h	-	5000	13 88-h	-
9600	120	00 78-h	-	2500	09 C4-h	-
19200	60	00 3C-h	-	1250	04 E2-h	-
38400	30	00 1E-h	-	625	02 71-h	-
57600	20	00 14-h	-	416	01 A0-h	?
115200	10	00 0A-h	-	208	00 D0-h	?
230400	5	00 05-H	-	104	00 68-h	?

Com'è possibile verificare dalla tabella, la frequenza 18.432 MHz permette la divisione intera del suo ciclo di clock per tutte le velocità tipiche definite dallo standard UART, il rapporto di divisione è contenuto in esadecimale nei due registri DIVISOR LATCH (MSB & LSB). Dall'analisi della tabella è comprensibile il perché della divisione della frequenza di clock ad un valore ancora 16 volte superiore alla velocità desiderata. I registri possono contenere al massimo il valore 65535 (FF FF-h) che non permetterebbe un rallentamento abbastanza grande per i bit rate inferiori ai 300 bit/s, quindi si è scelto di imporre una prima divisione e poi un'ulteriore di valore fisso 16, ad opera del blocco BAUD GENERATOR.

Nel caso in esame dividere il clock ad una frequenza 16 volte quella dovuta avrebbe imposto un limite di velocità di trasmissione pari a 2400 bit/s (10000 è l'ultimo valore divisibile per 16 che da risultato intero), troppo basso, quindi operando sul blocco VHDL equivalente al BAUD GENERATOR si è eliminata quest'ulteriore divisione e si è inviato un clock direttamente scalato dai 24 MHz pari esattamente alla bit rate desiderata. [Pea05] [NS05]

Contrariamente a quanto raccomandi il datasheet, duty cycle > 50 % per i divisori > 3, per semplificare il design il clock è generato direttamente dai 24 MHz, con duty cycle 50 % e viene scelto tra quatto valori utilizzando la combinazione di due switch sulla board:

Tabella 21

Combinazione	Divisore per 24 MHz	Bit rate [bit/s]
00	10000	2400
01	5000	4800
11	2500	9600
10	1250	19200

```
ENTITY programmable_clock_divider_24MHz_to_2400_4800_9600_19200_kHz IS
  PORT(
    clock_24MHz:         IN STD_LOGIC;
    bit_rate_selector:   IN STD_LOGIC_VECTOR(1 DOWNTO 0);
    uart_clock:          OUT STD_LOGIC
  );
END programmable_clock_divider_24MHz_to_2400_4800_9600_19200_kHz;
-------------------------------------------------------------------
ARCHITECTURE arch OF
             programmable_clock_divider_24MHz_to_2400_4800_9600_19200_kHz
IS

  SIGNAL temp_uart_clock:  STD_LOGIC;

  BEGIN

  PROCESS(clock_24MHz)
    -- 24 MHz / 10 000 =  2.4 kHz
    VARIABLE  count_2400_bps: INTEGER RANGE 0 TO 4999;
    -- 24 MHz /  5 000 =  4.8 kHz
    VARIABLE  count_4800_bps: INTEGER RANGE 0 TO 2499;
    -- 24 MHZ /  2 500 =  9.6 kHz
    VARIABLE  count_9600_bps: INTEGER RANGE 0 TO 1249;
    -- 24 MHz /  1 250 = 19.2 kHz
    VARIABLE count_19200_bps: INTEGER RANGE 0 TO  624;

    BEGIN
    IF(clock_24MHz'EVENT AND clock_24MHz = '1') THEN
      CASE bit_rate_selector IS

        WHEN "00" =>
          count_2400_bps := count_2400_bps +1;
          IF(count_2400_bps = 4999) THEN
            temp_uart_clock <= NOT temp_uart_clock;
            count_2400_bps := 0;
          END IF;
```

```
        WHEN "01" =>
          count_4800_bps := count_4800_bps +1;
          IF(count_4800_bps = 2499) THEN
            temp_uart_clock <= NOT temp_uart_clock;
            count_4800_bps := 0;
          END IF;

        WHEN "11" =>
          count_9600_bps := count_9600_bps +1;
          IF(count_9600_bps = 1249) THEN
            temp_uart_clock <= NOT temp_uart_clock;
            count_9600_bps := 0;
          END IF;

        WHEN OTHERS =>   --(10)
          count_19200_bps := count_19200_bps +1;
          IF(count_19200_bps = 624) THEN
            temp_uart_clock <= NOT temp_uart_clock;
            count_19200_bps := 0;
          END IF;

      END CASE;
  END PROCESS;

  uart_clock <= temp_uart_clock;

END arch;
```

Listato 71

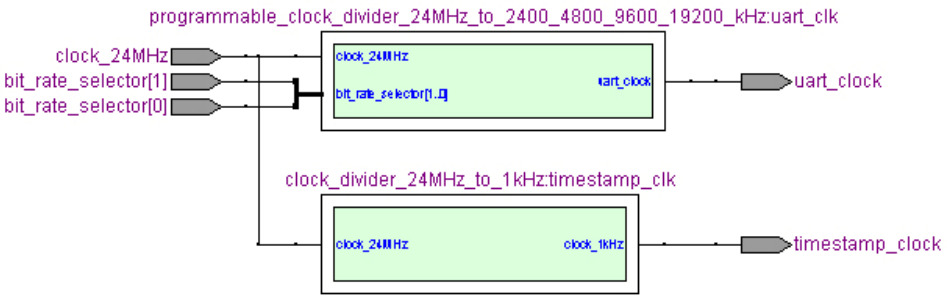


Figura 152

14.6 Metastabilità

La metastabilità è una condizione, che può causare esiti non predicibili nei sistemi elettronici, che si verifica quando un segnale viene trasferito tra circuiti che appartengono a domini temporali non correlati (o asincroni).

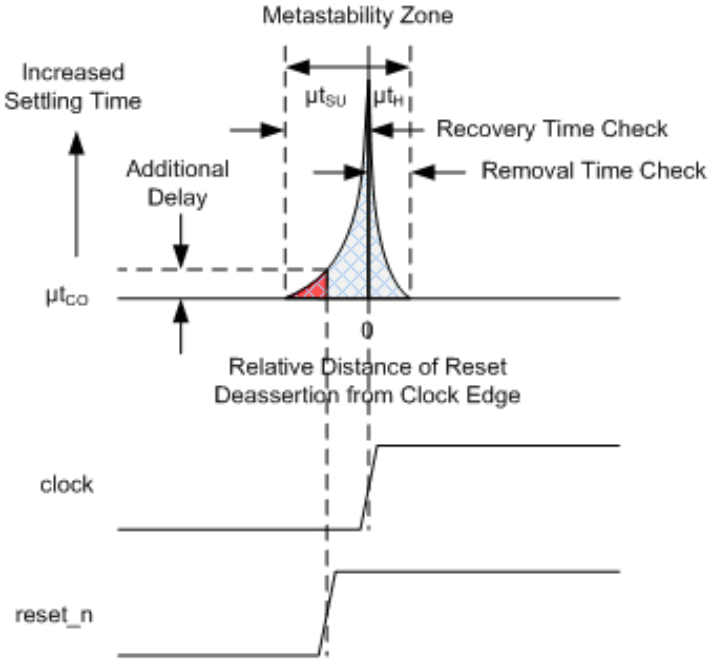


Figura 153

Nei dispositivi elettronici tutti i registri devono rispettare delle specifiche temporali ben definite, condizione che permette ad ogni registro di catturare correttamente ogni dato in ingresso e produrre il segnale in uscita.

Per assicurare operazioni affidabili, il segnale in ingresso ad un registro deve essere stabile nel suo valore logico per un minimo intervallo temporale, sia prima che dopo il fronte del clock (tempi detti rispettivamente *register setup time* t_{SU} e *register hold time* t_H). Il valore in output del registro diventa disponibile dopo uno specifico *clock-to-output delay* (t_{CO}).

Se la transizione su un dato in input viola le specifiche t_{SU} e t_H di un registro, può accadere che il dato in output si presenti in uno stato metastabile. Se ciò avviene lo stato logico in uscita sarà indefinito, tra alto e basso, per un periodo di tempo (condizione che provoca l'allungamento del tempo di uscita dal registro oltre il tempo t_{CO} specificato).

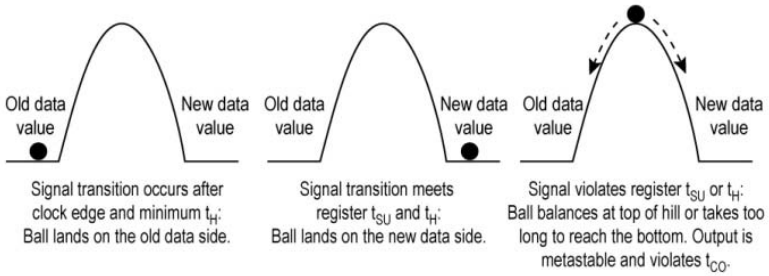


Figura 154

Un registro, che campiona un dato in ingresso ogni fronte di clock, può essere schematizzato come una sfera su una collina. Le basi ai lati della collina rappresentano gli stati logici stabili (lo stato precedente e successivo del segnale intorno alla sua transizione), la cima della collina rappresenta lo stato metastabile.

Se la sfera viene lasciata cadere sulla cima della collina può rimanere sospesa su di essa per un tempo non definito, in una condizione di equilibrio instabile, rotolerà lungo un lato di essa. Una posizione di arrivo più lontana dalla cima della collina sarà caratterizzata da un tempo di raggiungimento, della condizione stabile, più breve e il rotolamento sarà minimo.

Se la transizione di un dato sarà successiva al fronte del clock e al tempo t_H, si verificherà una condizione analoga a lasciare cadere la sfera sul lato *old data value* della collina, quindi il segnale in output si manterrà allo stesso valore logico assunto prima del fronte del clock.

Se invece la transizione in input ad un registro avviene prima del fronte del clock e rimane stabile per il tempo minimo t_{SU} e oltre t_H, si verificherà una condizione analoga a far cadere la sfera lungo il lato *new data value* della collina, di conseguenza il segnale in output raggiungerà il nuovo stato logico abbastanza velocemente da rispettare il tempo t_{CO} definito.

Se il dato in input viola uno dei tempi t_{SU} o t_H, sarà una situazione analoga alla caduta della sfera sulla collina. Più il punto di caduta è vicino alla cima, più tempo impiegherà la sfera a raggiungere la base, aumentando così il ritardo dal fronte del clock, per un segnale in output stabile oltre il tempo definito t_{CO}.

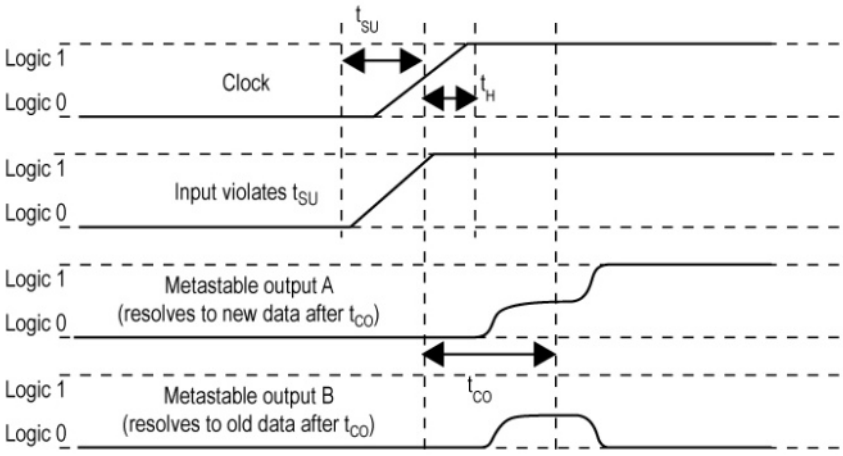


Figura 155

Se il segnale in uscita riesce comunque a raggiungere lo stato logico desiderato, prima di essere letto da un altro registro, non si riscontreranno effetti negativi per il segnale, prima metastabile, sul sistema.

Quando invece un segnale metastabile viene letto da un registro prima di raggiungere lo stato definitivo si può verificare il fallimento della logica del sistema. Lo stato logico inconsistente può essere recepito dai registri successivi come un valore differente da quello reale. [ALTd] [CM02] [CMG03]

15 D - Realizzazione circuiti di interfaccia digitale

15.1 Segnali in input al sistema

I segnali in ingresso al sistema giungono con logica TTL, dopo essere stati convertiti da NIM attraverso moduli dedicati. E' necessario, prima dell'inserzione sui GPIO dell'FPGA ad alta impedenza, inserire in parallelo un resistore da 50 Ω per ogni canale.

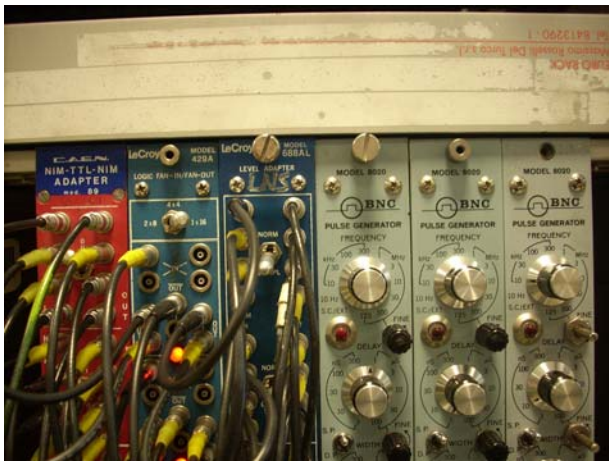


Figura 156

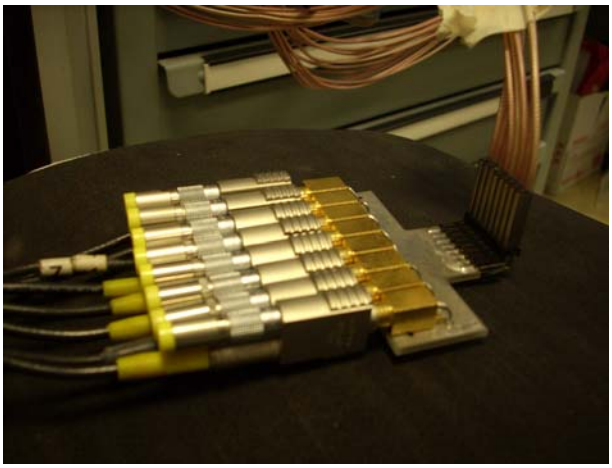


Figura 157

Il modulo contatore è stato sottoposto a verifica di prestazioni, per la risposta in funzione della frequenza di arrivo dei segnali in ingresso, grazie ad un generatore di frequenza.
Come visibile nel grafico seguente i conteggi ottenuti hanno risposta lineare, in funzione dell'ingresso (fino 50 MHz), e l'errore percentuale è molto basso, anche ad alte frequenze.

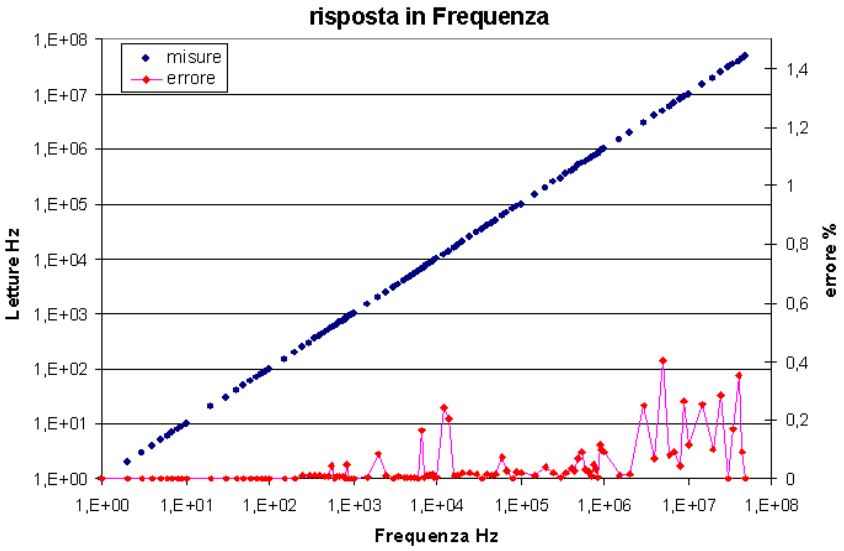


Figura 158

Per verificare la risposta del modulo coincidenza temporale, al variare del ritardo tra i due segnali accoppiati, è stato effettuato un test di misura, i cui risultati hanno permesso di confrontare i risultati ottenuti con quelli desiderati.

Sono state effettuate misure di conteggio, in diverse situazioni:

- time-slot: 60, 80, 100 ns (rispettivamente 3, 4, 5 cicli di clock per frequenza di ingresso pari a 50 MHz)
- pulse widht: 15, 23, 36 ns (rispettivamente poco meno, poco più e quasi il doppio di 20 ns, un ciclo di clock)

quindi 9 scenari differenti per i quali, fissato il tempo di delay, è stato esplorato tutto il range di frequenze alle quali il dispositivo potrebbe trovarsi a lavorare.

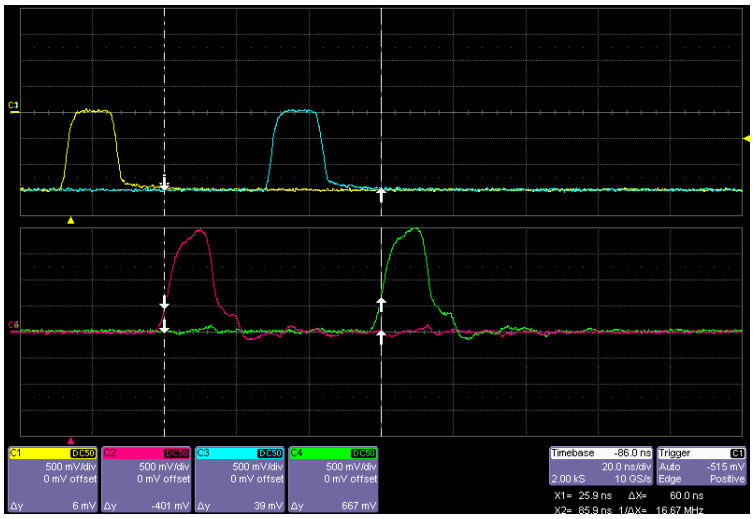


Figura 159

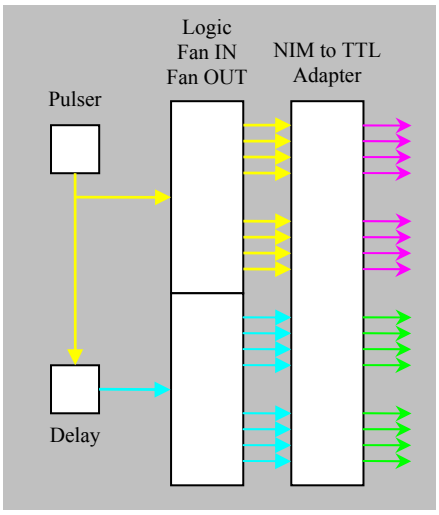


Figura 160

Il risultato complessivo di questo set di misure è descritto dal grafico seguente: le tre curve rappresentano i tre widht temporali, in ascissa è rappresentato l'asse dei tempi centrato sul time slot fissato (60, 80, 100 ns), in ordinata il tasso di coincidenza normalizzato $\left(\frac{frequenza\_ingresso}{tasso\_coincidenza}\right)$.

.

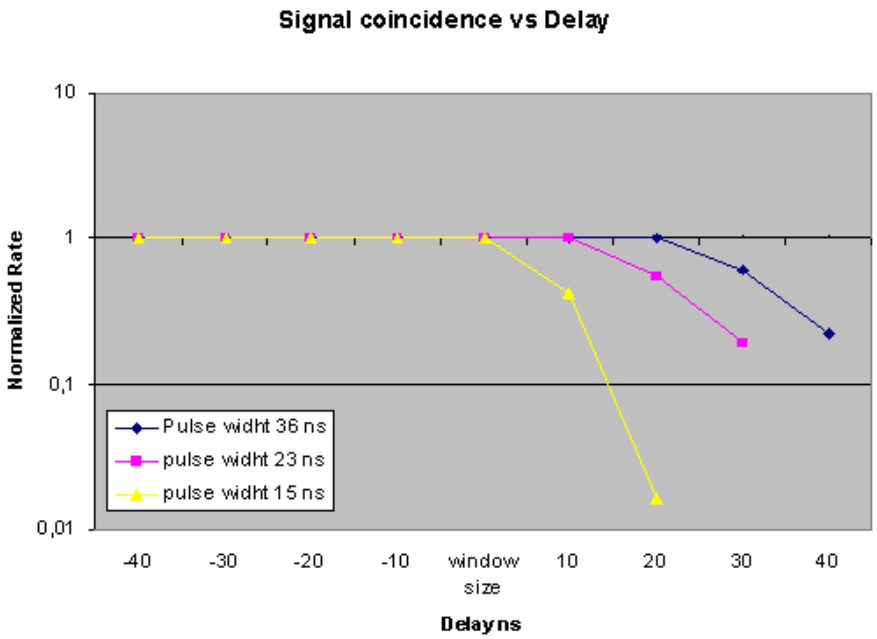


Figura 161

Il grafico di caratterizzazione mostra che il comportamento desiderato è stato raggiunto:

- per ritardi inferiori al time-slot la coincidenza è verificata normalmente
- per ritardi superiori al time-slot si nota una riduzione repentina del tasso di coincidenza in funzione della larghezza degli impulsi: 1,5 ordini di grandezza per 15 ns e 0,5 ordini di grandezza per 23 e 36 ns
- superata la "zona critica" la coincidenza viene definitivamente annullata.

15.2 Segnali in output al sistema

La comunicazione tra FPGA e terminale di acquisizione dati viene effettuata tramite protocollo seriale UART. Per adattare i livelli di tensione in ingresso/uscita all'FPGA agli standard di comunicazione seriale su cavo è necessario l'utilizzo di transreceivers, moduli che effettuano la conversione da TTL ai valori definiti dai vari standard.

E' stata realizzata una board di connessione multiprotocollo, da TTL a RS-232, RS-422 e RS-485, utilizzando i seguenti moduli:

- RS-232: MAXIM MAX232;
- RS-422: MAXIM MAX488;
- RS-485: MAXIM MAX485.

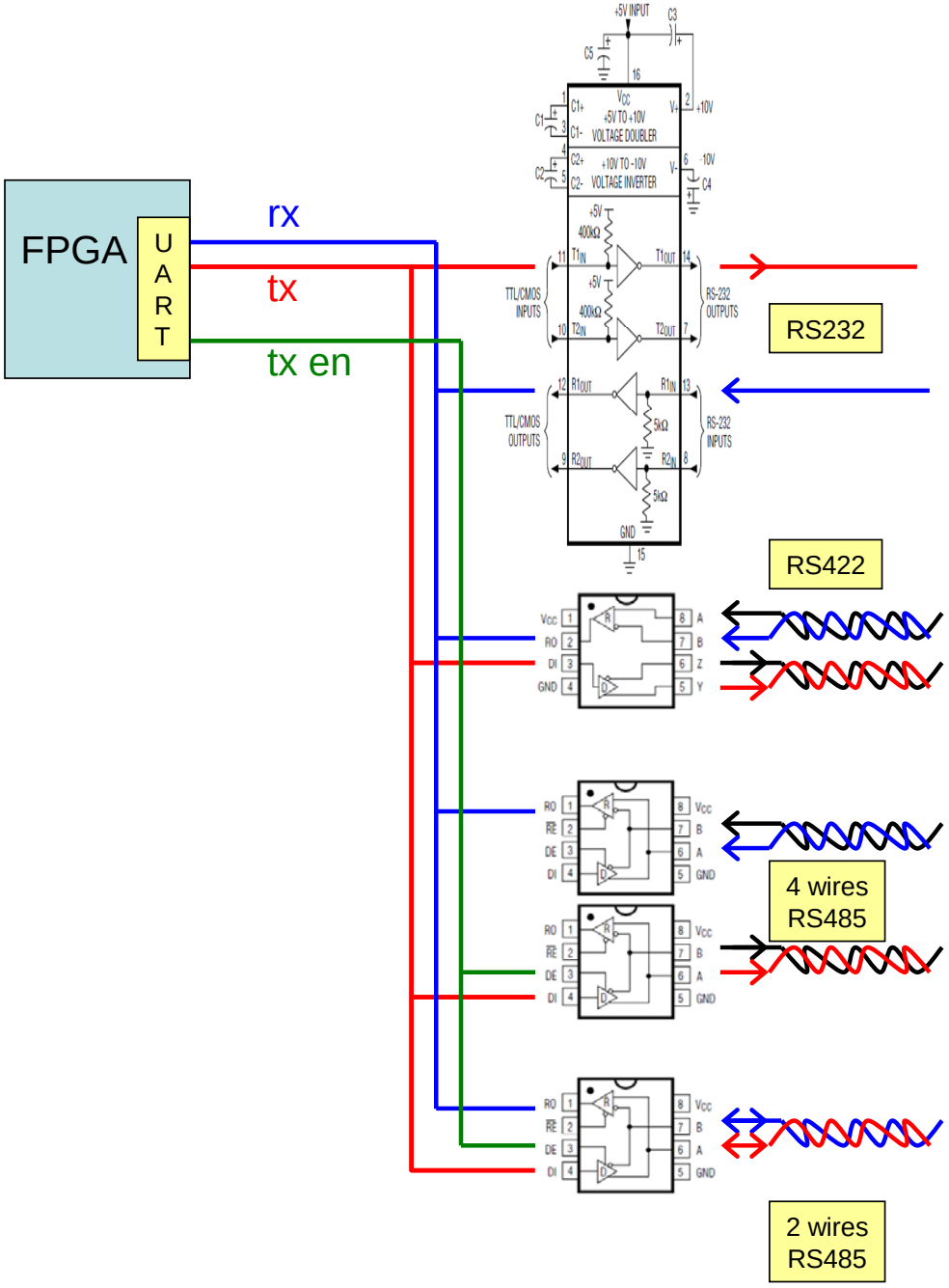


Figura 162

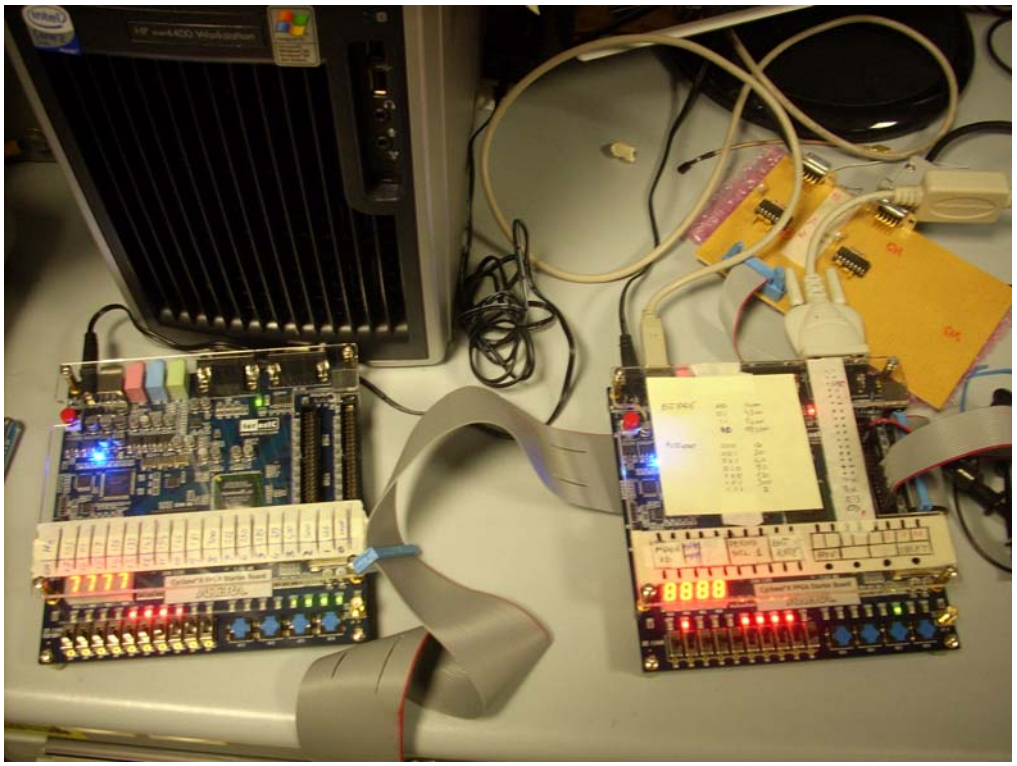


Figura 163

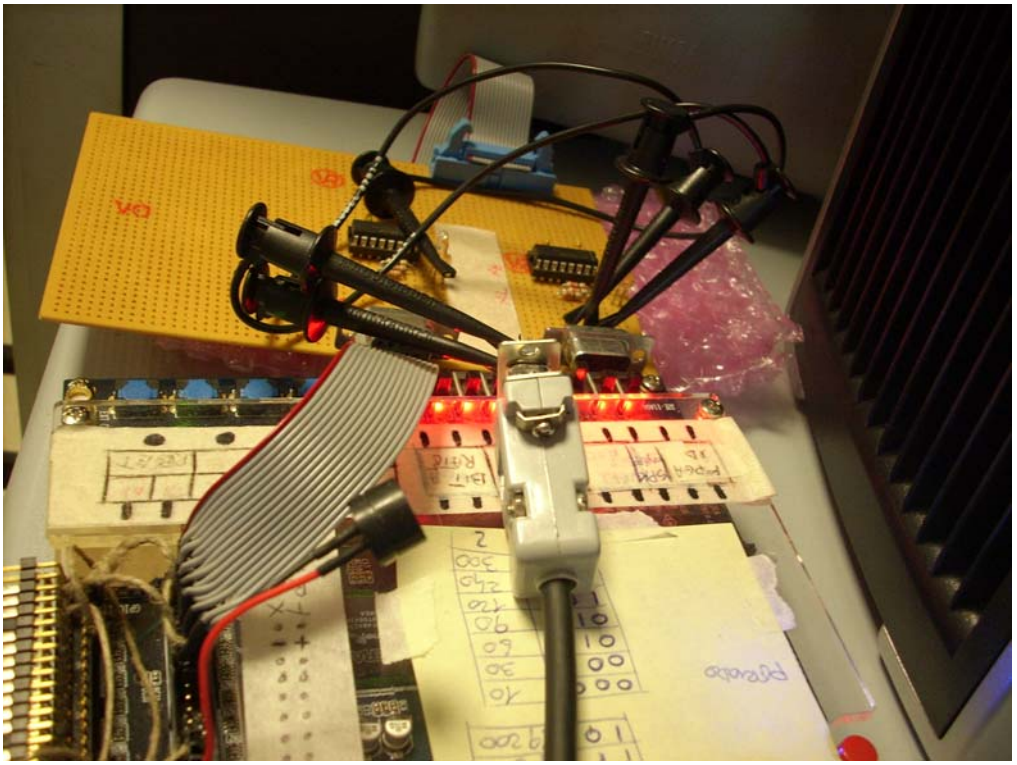


Figura 164

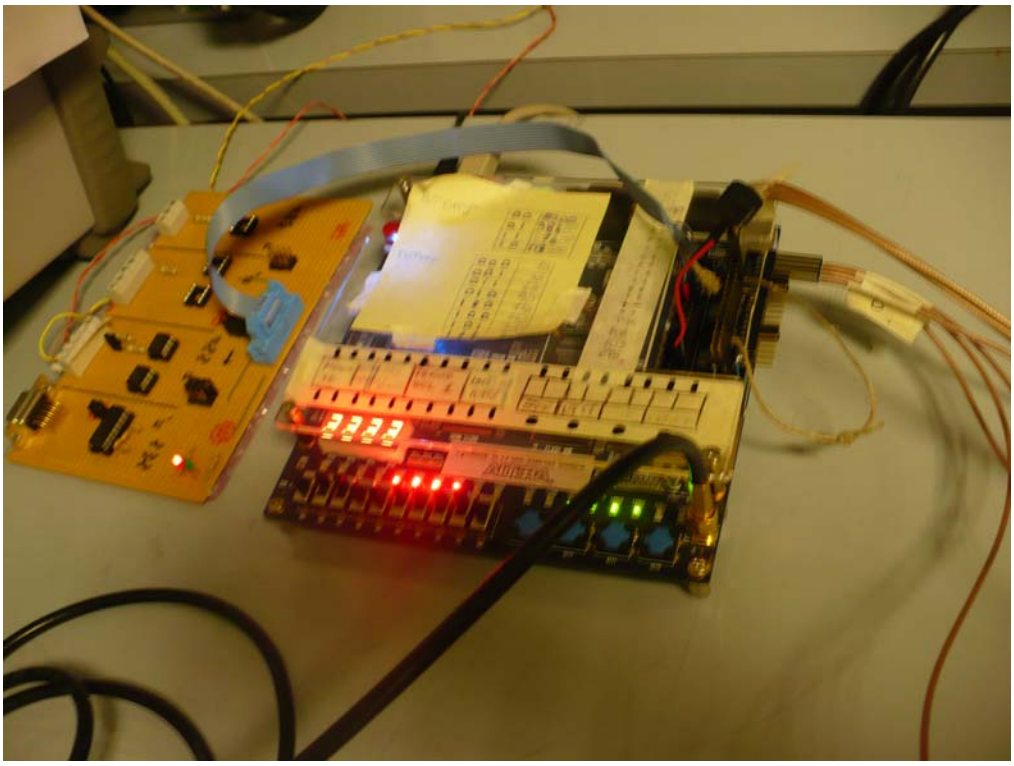


Figura 165

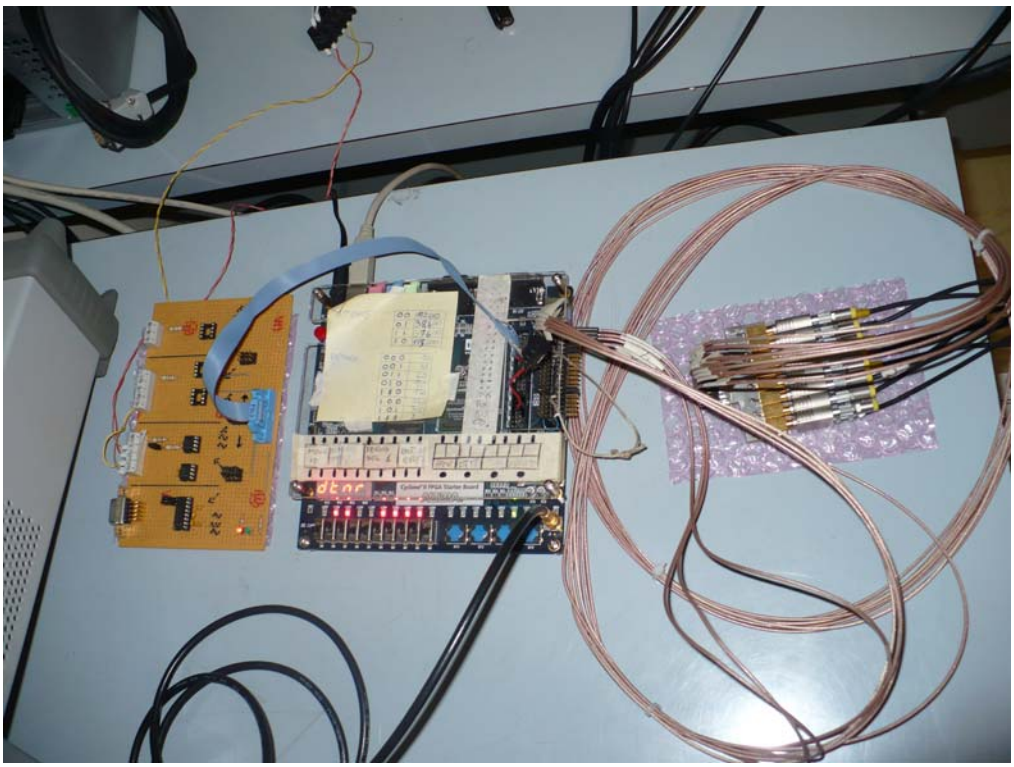


Figura 166

15.3 Class Diagram completo del programma

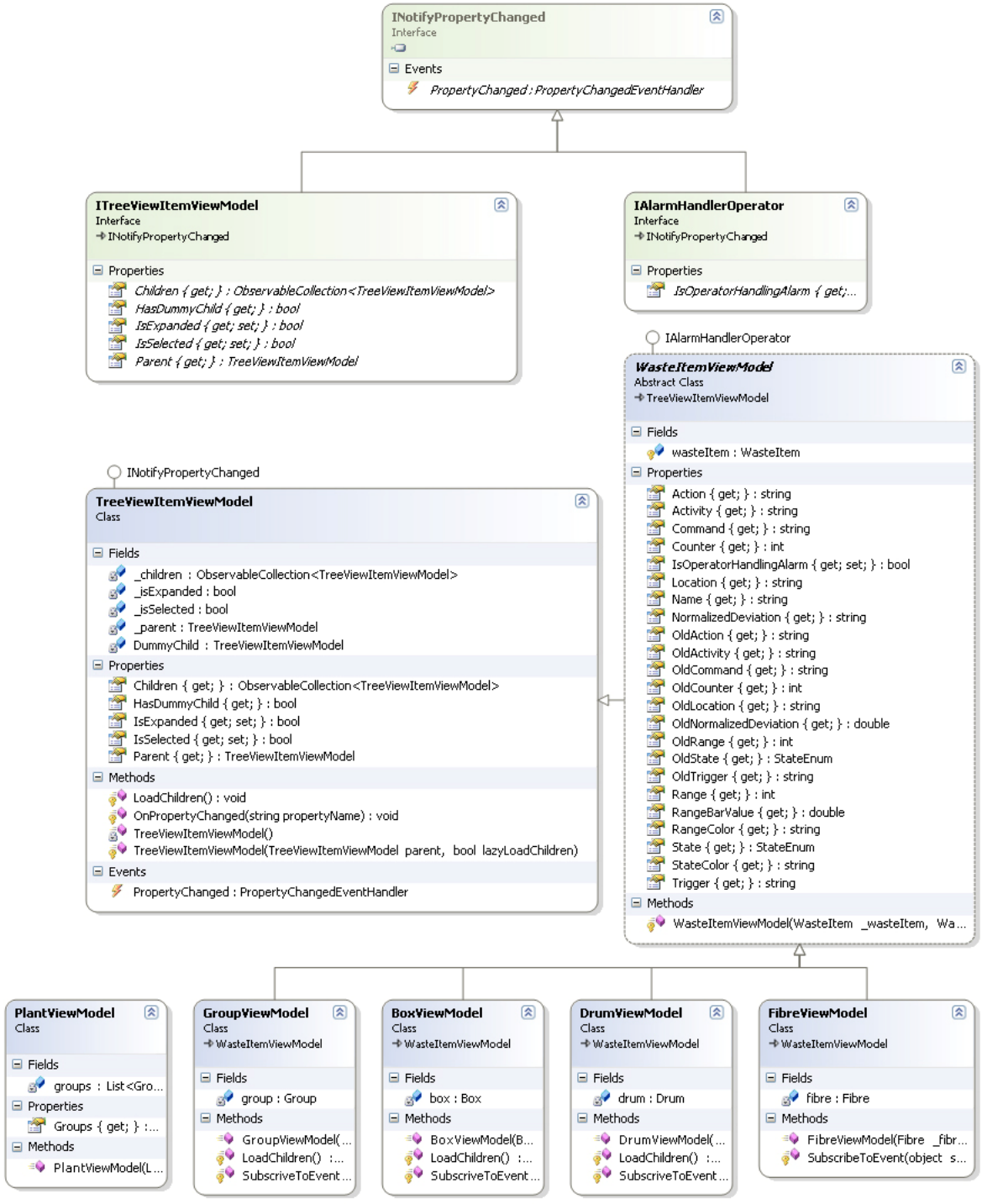


Figura 167

Gerarchia ViewModel

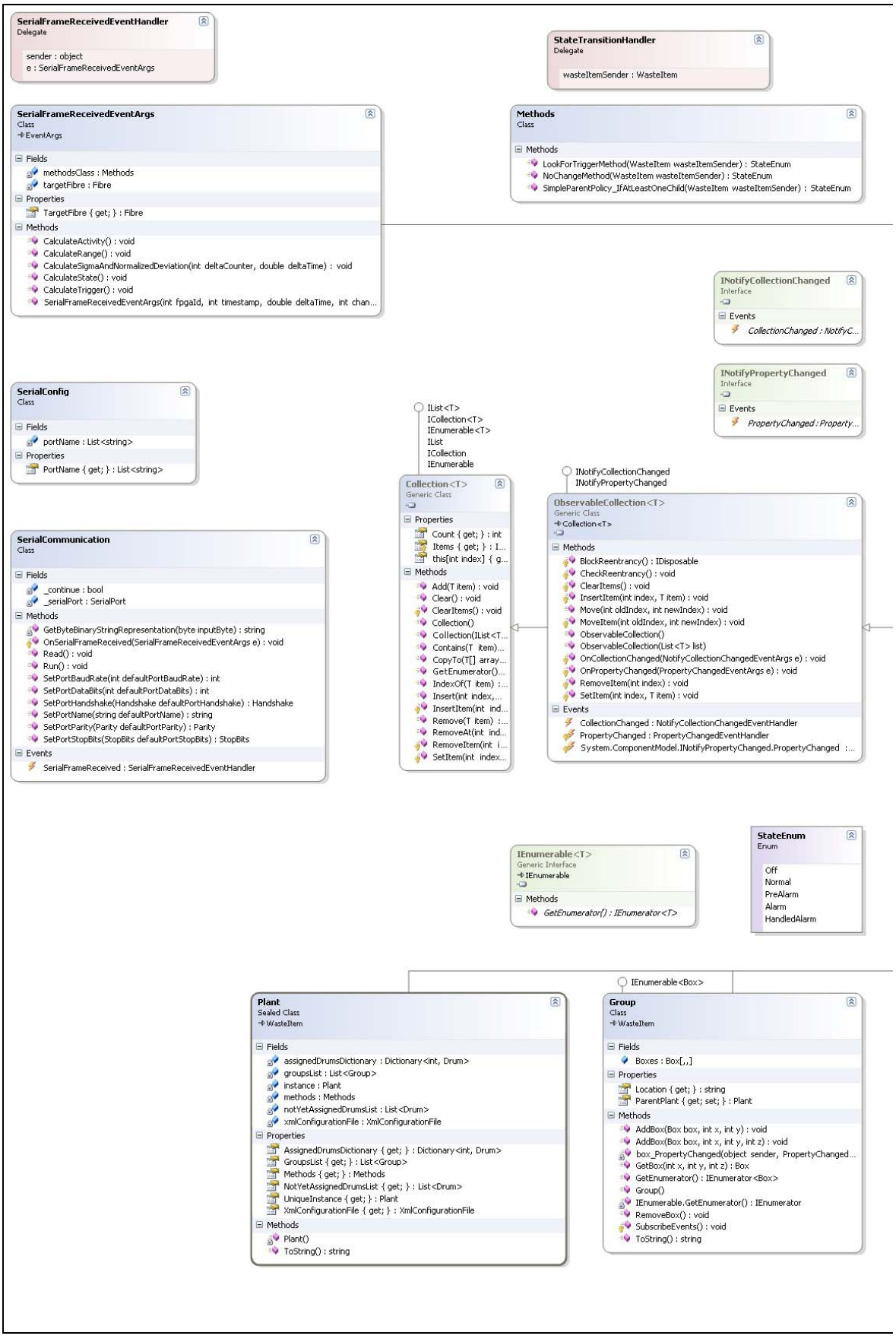
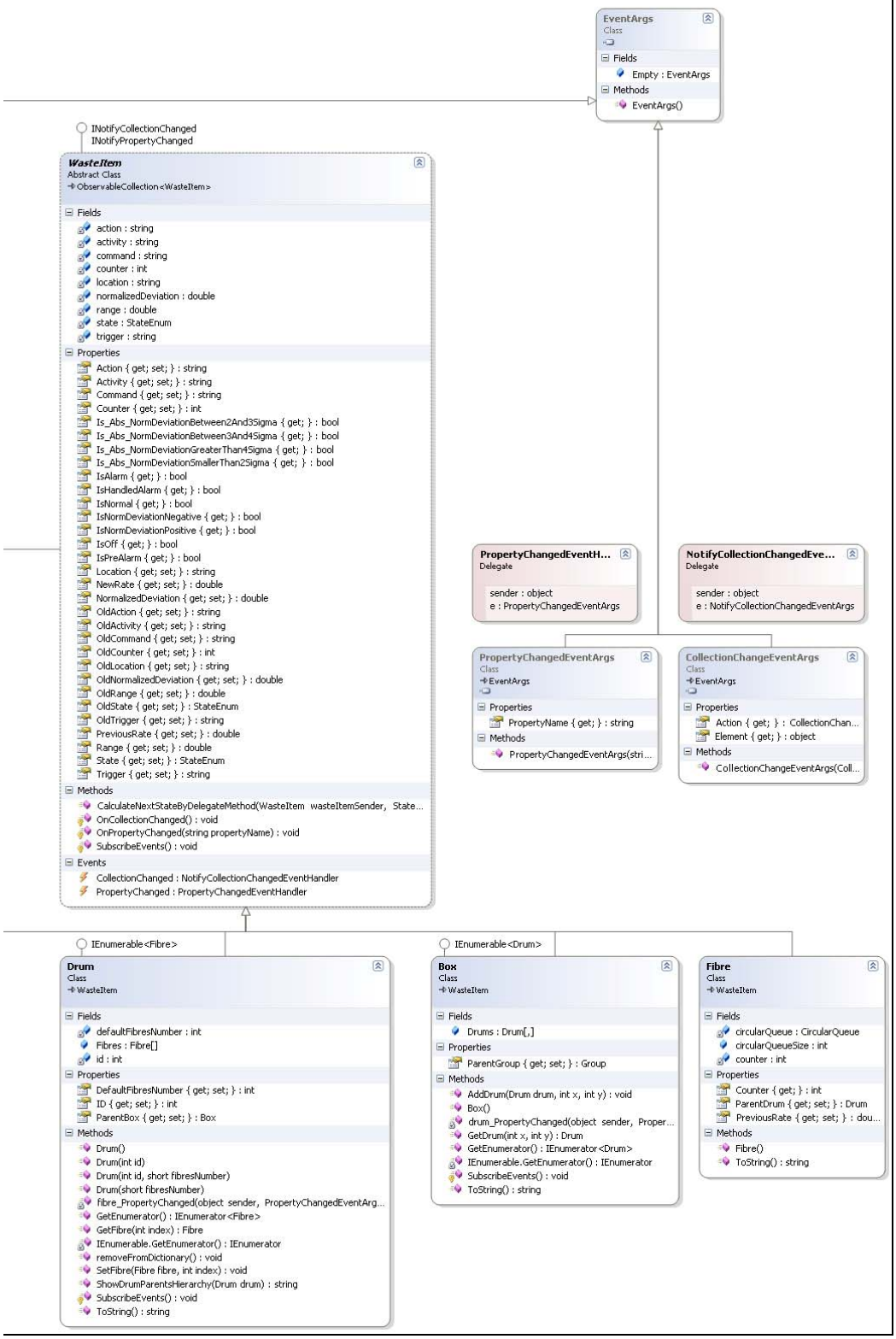


Figura 168

Class diagram completo


EventArgs
Class
Fields
Empty : EventArgs
Methods
EventArgs()
INotifyCollectionChanged
INotifyPropertyChanged
WasteItem
Abstract Class
ObservableCollection<WasteItem>
Fields
action : string
activity : string
command : string
counter : int
location : string
normalizedDeviation : double
range : double
state : StateEnum
trigger : string
Properties
Action { get; set; } : string
Activity { get; set; } : string
Command { get; set; } : string
Counter { get; set; } : int
Is_Abs_NormDeviationBetween2And3Sigma { get; } : bool
Is_Abs_NormDeviationBetween3And4Sigma { get; } : bool
Is_Abs_NormDeviationGreaterThan4Sigma { get; } : bool
Is_Abs_NormDeviationSmallerThan2Sigma { get; } : bool
IsAlarm { get; } : bool
IsHandledAlarm { get; } : bool
IsNormal { get; } : bool
IsNormDeviationNegative { get; } : bool
IsNormDeviationPositive { get; } : bool
IsOff { get; } : bool
IsPreAlarm { get; } : bool
Location { get; set; } : string
NewRate { get; set; } : double
NormalizedDeviation { get; set; } : double
OldAction { get; set; } : string
OldActivity { get; set; } : string
OldCommand { get; set; } : string
OldCounter { get; set; } : int
OldLocation { get; set; } : string
OldNormalizedDeviation { get; set; } : double
OldRange { get; set; } : double
OldState { get; set; } : StateEnum
OldTrigger { get; set; } : string
PreviousRate { get; set; } : double
Range { get; set; } : double
State { get; set; } : StateEnum
Trigger { get; set; } : string
Methods
CalculateNextStateByDelegateMethod(WasteItem wasteItemSender, State...
OnCollectionChanged() : void
OnPropertyChanged(string propertyName) : void
SubscribeEvents() : void
Events
CollectionChanged : NotifyCollectionChangedEventHandler
PropertyChanged : PropertyChangedEventHandler
PropertyChangedEventH...
Delegate
sender : object
e : PropertyChangedEventArgs
NotifyCollectionChangedEve...
Delegate
sender : object
e : NotifyCollectionChangedEventArgs
PropertyChangedEventArgs
Class
EventArgs
Properties
PropertyName { get; } : string
Methods
PropertyChangedEventArgs(stri...
CollectionChangeEventArgs
Class
EventArgs
Properties
Action { get; } : CollectionChan...
Element { get; } : object
Methods
CollectionChangeEventArgs(Coll...
IEnumerable<Fibre>
IEnumerable<Drum>
Drum
Class
WasteItem
Fields
defaultFibresNumber : int
Fibres : Fibre[]
id : int
Properties
DefaultFibresNumber { get; set; } : int
ID { get; set; } : int
ParentBox { get; set; } : Box
Methods
Drum()
Drum(int id)
Drum(int id, short fibresNumber)
Drum(short fibresNumber)
fibre_PropertyChanged(object sender, PropertyChangedEventArg...
GetEnumerator() : IEnumerator<Fibre>
GetFibre(int index) : Fibre
IEnumerable.GetEnumerator() : IEnumerator
removeFromDictionary() : void
SetFibre(Fibre fibre, int index) : void
ShowDrumParentsHierarchy(Drum drum) : string
SubscribeEvents() : void
ToString() : string
Box
Class
WasteItem
Fields
Drums : Drum[,]
Properties
ParentGroup { get; set; } : Group
Methods
AddDrum(Drum drum, int x, int y) : void
Box()
drum_PropertyChanged(object sender, Proper...
GetDrum(int x, int y) : Drum
GetEnumerator() : IEnumerator<Drum>
IEnumerable.GetEnumerator() : IEnumerator
SubscribeEvents() : void
ToString() : string
Fibre
Class
WasteItem
Fields
circularQueue : CircularQueue
circularQueueSize : int
counter : int
Properties
Counter { get; } : int
ParentDrum { get; set; } : Drum
PreviousRate { get; set; } : dou...
Methods
Fibre()
ToString() : string

15.4 Classe CircularQueue

```
public class CircularQueue {
    double[] queue;
    int index, size;
    double valuesSum, mean;
    bool firstLapConcluded;

    //Costruttore: crea una coda circolare di dimensione n
    public CircularQueue(int size) {
        this.size = size;
        this.queue = new double[this.size];
        this.index = 0;
        this.firstLapConcluded = false;
    }

    //Aggiunge il valore alla coda; sovrascrive il
    //valore più vecchio con quello passato
    public void Add(double value) {
        //se l'indice della coda è giunto alla fine di un giro
        if (this.index == this.size) {
            //notifica se è stato completato il primo giro
            if (!this.firstLapConcluded)
                this.firstLapConcluded = true;

            //azzera l'indice
            this.index = 0;
        }

        //se almeno un giro è stato completato sottrai
        //il valore da sovrascrivere alla somma parziale
        if (this.firstLapConcluded)
            this.valuesSum -= this.queue[index];

        //inserisci il nuovo valore
        this.queue[index] = value;
        //aggiorna la somma parziale
        this.valuesSum += value;
        //e l'indice per il prossimo inserimento
        this.index++;
    }

    //Media sui valori registrati
    public double GetMean() {
        //se non è stato completato almeno un giro dividi
        //per il numero di elementi presenti
        if (!this.firstLapConcluded)
            this.mean = this.valuesSum / this.index;

        //In caso contrario dividi per la dimensione della coda
        else
            this.mean = this.valuesSum / this.size;

        return this.mean;
    }
}
```

Listato 72

15.5 Funzione GetByteBinaryStringRepresentation

```
//Dato un 8-bit unsigned integer (Byte) restituisce la stringa
//a 8 cifre binaria che lo rappresenta, preservando i bit più
//significativi posti a 0 che si perderebbero
static string GetByteBinaryStringRepresentation(byte inputByte){

  //array che ospiterà la rappresentazione char dei bit
  char[] byteBynaryRepresentation = new char[8];
  //indice per scorrere il byte dal bit meno significativo
  int index = 0;
  //indice per scorrere l'array in senso inverso
  int position = 7;

  //viene analizzato il byte bit per bit a partire dal meno
  //significativo (uso index che parte da 0);
  //viene ricopiato il contenuto bit per bit in aun array di char
  //per la successiva conversione a stringa di 8 caratteri
  //di tipo "00000000", in modo da non perdere i bit più
  //significativi posti a zero
  while (index < 8) {
    //conrollo bit a bit coun un byte del tipo 00000001 in cui
    //l'1 finale viene shiftato a sinistra per ogni iterazione
    //(quindi 00000010, poi 00000100, ..., 10000000)

    //se il risultato dell' AND sul bit ritorna 1 ricopio 1
    if ((inputByte & (1 << index)) != 0)
      byteBynaryRepresentation[position] = '1';

    //in caso contrario 0
    else
      byteBynaryRepresentation[position] = '0';

    //indice per il byte incrementato per il giro successivo
    index++;

    //l'indice per l'array viene decrementato, perchè viene
    //percorso in ordine inverso,
    position--;
  }

  // Esempio: dato il byte in entrata 21(dec) = 00010101 (bin)
  //
  //            1 shiftato
  //            di index:           position:
  //            76543210            01234567
  //
  // 00010101 & 00000001 = 1 => out[_______1]
  // 00010101 & 00000010 = 0 => out[______01]
  // 00010101 & 00000100 = 1 => out[_____101]
  // 00010101 & 00001000 = 0 => out[____0101]
  // 00010101 & 00010000 = 1 => out[___10101]
  // 00010101 & 00100000 = 0 => out[__010101]
  // 00010101 & 01000000 = 0 => out[_0010101]
  // 00010101 & 10000000 = 0 => out[00010101]

  return new string(byteBynaryRepresentation);
}
```

Listato 73

16 Riferimenti Bibliografici

[ACT] Actel FPGAs
http://www.actel.com/products/devices.aspx

[ALT] Altera FPGAs
http://www.altera.com/devices/fpga/fpga-index.html

[ALTa] Why Use FPGAs in Embedded Designs?
http://www.altera.com/technology/embedded/fpgas/emb-why-use.html

[ALTb] Cyclone II Device Handbook
http://www.altera.com/literature/lit-cyc2.jsp

[ALTc] Understanding Metastability in FPGAs
ALTERA White Paper, 2009
http://www.altera.com/literature/wp/wp-01082-quartus-ii-metastability.pdf

[ALTd] Implementation and Timing of Reset Circuits in Altera FPGAs
http://www.alteraforum.com/forum/showthread.php?t=4281

[ANIMMA09] Pappalardo, Barbagallo, Bellini, Capogni, Cosentino, Febbraro, Finocchiaro, Greco, Scirè, Scirè
An Online Monitoring System for Nuclear Waste Storage
1st International Conference on Advancements in Nuclear Instrumentation, Measurement Methods and their Applications, Marseille (France), June 2009
http://www.animma.com/

[ANN] Ansaldo Nucleare s.p.a.
http://www.ansaldonucleare.it

[ANPA26] La gestione dei rifiuti radioattivi
Guida tecnica n° 26
Agenzia Nazionale Protezione Ambiente

[Bis07] Judith Bishop
C# 3.0 Design Patterns
O'Reilly, 2007

[CM02] Clifford E. Cummings and Don Mills;
"Synchronous Resets? Asynchronous Resets? I am so confused! How will I ever know which to use?";
SNUG San Jose, 2002 User Papers;
http://www.sunburst-design.com/papers/CummingsSNUG2002SJ_Resets.pdf

[CMG03] Clifford E. Cummings, Don Mills, and Steve Golson;
"Asynchronous & Synchronous Reset Design Techniques – Part Deux";
SNUG Boston, 2003 User Papers;
http://ens.ewi.tudelft.nl/Education/courses/et4351/CummingsSNUG2003Boston_Resets.pdf

[Coo02] James W. Cooper
Introduction to Design Patterns in C#
IBM T J Watson Research Center, 2002

[CPJS] Josh Smith
Simplifying the WPF TreeView by Using the ViewModel Pattern
The Code Project community, 2008
http://www.codeproject.com/KB/WPF/TreeViewWithViewModel.aspx

[DMNR] Strategic Project INFN Energy
Detector Mesh for Nuclear Repositories
http://www.lns.infn.it/link/dmnr

[GHJV95] Gamma, Helm, Johnson, Vlissides
Design Patterns: Elements of Reusable Object-Oriented Software
Addison Wesley, 1995

[IAEA10] Handbook on Nuclear Law – Implementing Legislation
International Atomic Energy Agency, 2010

[ICENES09] Cosentino, Barbagallo, Capogni, Febbraro, Finocchiaro, Greco, Pappalardo, Scirè, Scirè
A detector mesh for online monitoring of nuclear waste storage
14th International Conference on Emerging Nuclear Energy Systems, Ericeira (Portugal), July 2009
http://www.icenes2009.itn.pt/

[INFN] INFN - Istituto Nazionale di Fisica Nucleare
http://www.infn.it

[INFN06] Programma RIACE
Rivelatori e Acceleratori per l'Energia – Volume 1
http://www.infn.it/energia/Riace.pdf

[IPRD08] Cosentino, Pappalardo, Bellini, Febbraro, Scirè, Finocchiaro
DMNR (Detector Mesh for Nuclear Repositories): On-line monitoring of short/medium term radioactive waste storage
11th Topical Seminar on Innovative Particle and Radiation Detectors, Pisa (Italy), July 2008
http://www.bo.infn.it/sminiato/sm08/presentations/10-01/cosentino.pdf

[Jos08] Bipin Joshi
Beginning XML with C# 2008: From Novice to Professional
Apress, 2008

[LNS] Istituto Nazionale di Fisica Nucleare, Laboratori Nazionali del Sud
http://www.lns.infn.it

[Mac08] Matthew MacDonald
Pro WPF in C#2008: Windows Presentation Foundation with .NET 3.5
Apress, 2008

[Mic08] Mark Michaelis
Essential C# 3.0: For .NET Framework 3.5
Addison-Wesley, 2008

[MSDNa] Delegati (Guida per programmatori C#)
Microsoft Developer Network
http://msdn.microsoft.com/it-it/library/ms173171.aspx

[MSDNb] Eventi (Guida per programmatori C#)
Microsoft Developer Network
http://msdn.microsoft.com/it-it/library/awbftdfh.aspx

[MSDNc] Procedura: pubblicare eventi conformi alle indicazioni di .NET Framework (Guida per programmatori C#)
Microsoft Developer Network
http://msdn.microsoft.com/it-it/library/w369ty8x.aspx

[MSDNd] Introduzione ai design pattern
Microsoft Developer Network
http://msdn.microsoft.com/it-it/library/cc185081.aspx

[MSDNe] Dan Crevier
DataModel-View-ViewModel pattern series
Microsoft Developer Network Blogs, 2006
https://blogs.msdn.com/dancre/archive/2006/10/11/datamodel-view-viewmodel-pattern-series.aspx

[MSDNf] Josh Smith
WPF Apps with the Model-View-ViewModel Design Pattern
Microsoft Developer Network
http://msdn.microsoft.com/it-it/library/cc185038.aspx

[MSDNg] Corrado Cavalli
Introduzione a Windows Presentation Foundation
Microsoft Developer Network Magazine, 2009
http://msdn.microsoft.com/en-us/magazine/dd419663.aspx

[MSDNh] David Hill
The ViewModel Pattern
Microsoft Developer Network Blogs, 2009
http://blogs.msdn.com/dphill/archive/2009/01/31/the-viewmodel-pattern.aspx

[NEGWS08] Nagel, Evjen, Glynn, Watson, Skinner
Professional C# 2008
Wrox Press, 2008

[NIa] What Is the Definition of a TTL Compatible Signal?
National Instruments, 2002
http://digital.ni.com/public.nsf/allkb/ACB4BD7550C4374C86256BFB0067A4BD

[NIM] Standard NIM Instrumentation System,
U.S. NIM Committee, 1990

[NS05] PC16550D Universal Asynchronous Receiver/Trasmitter with FIFOs
National Semiconductor, June 1995
http://www.national.com/mpf/PC/PC16550D.html#Overview

[Pea05] Craig Peacock
Interfacing the Serial / RS232 Port
http://www.beyondlogic.org/serial/serial.htm

[PR08] Pialorsi, Russo
Programming Microsoft LINQ
Microsoft Press, 2008

[SGC] Scintillating Optical Fibers
Saint-Gobain Crystals
http://www.detectors.saint-gobain.com/fibers.aspx

[Sha08] John Sharp
Microsoft Visual C# 2008
Microsoft Press, 2008

[SL] MicroSL Family
sensL
http://sensl.com/products/silicon-photomultipliers/microsl/

[Sog] Sogin – Società Gestione Impianti Nucleari
http://www.sogin.it

[TECHa] James Wilson
When to Use a Clock vs. an Oscillator
Silicon Laboratories, 2009
http://www.techonline.com/showArticle.jhtml?articleID=217200702

[VG02] Frank Vahid, Tony Givargis
Embedded System Design: A Unified Hardware/Software Introduction
John Wiley & Sons, 2002
http://esd.cs.ucr.edu/

[XIL] XILINX FPGAs
http://www.xilinx.com/products/silicon-devices/fpga/index.htm

17 Ringraziamenti

Finalmente qualche parola di ringraziamento.
Aspettavo di scrivere questa pagina da quattro lunghi anni…
Devo, indubbiamente, riconfermare la mia profonda gratitudine nei confronti del prof. Michele Malgeri, dall'inizio alla fine, sempre disponibile e pronto ad indirizzarmi nello sviluppo di questo lungo lavoro.
Ai Laboratori Nazionali del Sud ho avuto due guide preziose, il Dott. Paolo Finocchiaro e il Dott. Luigi Cosentino, che sono riuscite a ispirarmi nel mio lavoro giorno per giorno.
Grazie, Paolo e Gigi, per la stima e l'affetto che avete dimostrato nei miei confronti.
Dott. Alfio Pappalardo: Alfio, amico sincero e generoso, non so come avrei fatto, senza il tuo appoggio, a superare i momenti di difficoltà.
Un ringraziamento speciale va al Dipartimento di Elettronica e Rivelatori: Claudio Calì, Pietro Litrico, Giuseppe Passaro e Salvatore Marino. Sono stato ospite nei vostri uffici per più di un anno e abbiamo lavorato insieme su tanti frammenti di questo progetto.
Voglio, inoltre, ricordare tutti voi che avete dato il vostro contributo da "dietro le quinte", dall'officina meccanica, dal centro di calcolo e dal magazzino: avete reso possibile la realizzazione di tutti i miei progetti.
Ringrazio Ansaldo Nucleare e INFN, per avermi dato l'opportunità di svolgere questo lavoro ed inoltre tutto lo staff di Microsensor srl.
Grazie pure a tutti voi che siete stati parte integrante del progetto DMNR: Vincenzo Febbraro, Giuseppe Greco, Massimo Barbagallo, Carlotta Scirè, Giovanni Guardo, Gianfranco Vecchio.

Grazie, papà e mamma, per tutto.

Infine grazie a te, Rosanna, per essere stata sempre al mio fianco, soprattutto nei momenti difficili. Sono giunto fin qui anche grazie a te. Ti amo.