VHDL Testbenches

La libreria IEEE
La libreria STD
La libreria MODELSIM_LIB
Il package TEXTIO
Struttura dei testbench

La libreria IEEE

- La libreria standard IEEE è composta 6 package
  - **std_logic_1164**
  - **std_logic_arith**
  - **std_logic_unsigned**
  - **std_logic_signed**
  - **std_logic_misc**
  - **std_logic_textio**
- Solitamente si usano nel modo seguente

```
library IEEE
use IEEE.<packagename>.all;
```

Le librerie IEEE

- Definiscono
  - Tipi
    - std_logic, std_logic_vector, std_ulogic, ...
  - Operatori aritmetici
    - +, -, *, shl, shr
  - Operatori relazionali
    - >, >=, <, <=, …
  - Funzioni di conversione
    - conv_integer, conv_std_logic_vector, …
    - to_stdlogic, to_stdlogicvector, …
- Per la scrittura di testbench hanno particolare rilevanza le funzioni di conversione

Le librerie IEEE

- Conversione in interi
  - `conv_integer(arg: unsigned)`
  - `conv_integer(arg: signed)`
- Conversione in std_logic_vector
  - `conv_std_logic_vector(arg: integer, size: integer)`
  - `conv_std_logic_vector(arg: unsigned, size: integer)`
  - `conv_std_logic_vector(arg: signed, size: integer)`
- Conversione in signed
  - `conv_signed(arg: integer, size: integer)`
  - `conv_signed(arg: unsigned, size: integer)`
  - `conv_signed(arg: signed, size: integer)`
- Conversione in unsigned
  - `conv_unsigned(arg: integer, size: integer)`
  - `conv_unsigned(arg: unsigned, size: integer)`
  - `conv_unsigned(arg: signed, size: integer)`

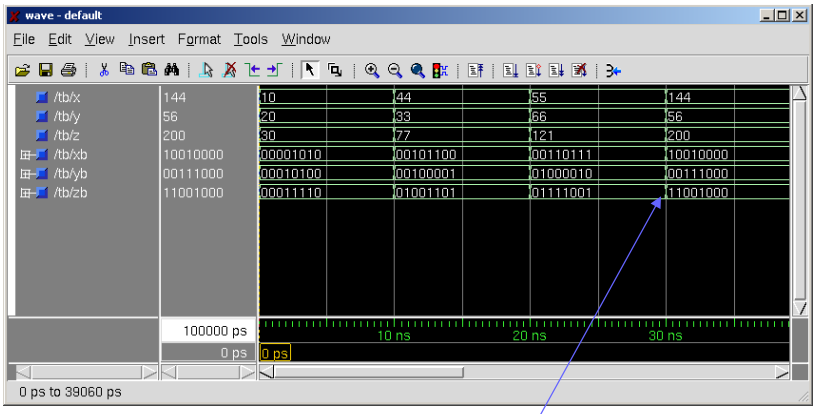
Le librerie IEEE

- Conversione in bit/bit_vector
  - `to_bit (s: std_ulogic)`
  - `to_bitvector(s: std_logic_vector)`
  - `to_bitvector(s: std_ulogic_vector)`
- Conversione in std_logic /std_logic_vector
  - `to_stdulogic (b: bit)`
  - `to_stdlogicvector(b: bit_vector)`
  - `to_stdlogicvector(s: std_ulogic_vector)`
  - `to_stdulogicvector(b: bit_vector)`
  - `to_stdulogicvector(s: std_logic_vector)`

Esempio: Conversione da/verso interi

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_arith.all;
use IEEE.std_logic_unsigned.all;

entity TB is
end TB;

architecture BEH of TB is
    signal x,  y,  z:  integer;
    signal xb, yb, zb: std_logic_vector(7 downto 0);
begin
    gen: process
    begin
        x <=  10; y <= 20;  wait for 10 ns;
        x <=  44; y <= 33;  wait for 10 ns;
        x <=  55; y <= 66;  wait for 10 ns;
        x <= 144; y <= 56;  wait;
    end process;

    xb <= conv_std_logic_vector( x, 8 );
    yb <= conv_std_logic_vector( y, 8 );
    z  <= conv_integer( zb );

    DUT: zb <= xb + yb;

end BEH;
```

Esempio: Conversione da/verso interi

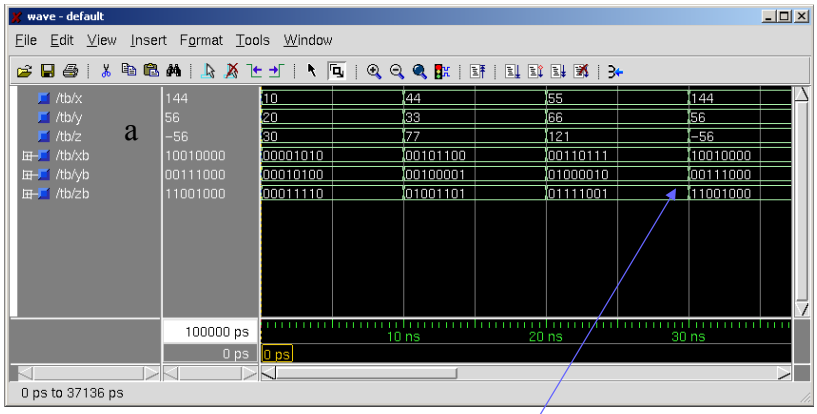


Interpretato senza segno in modo corretto

Esempio: Conversione da/verso interi

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_arith.all;
use IEEE.std_logic_signed.all;

entity TB is
end TB;

architecture BEH of TB is
    signal x,  y,  z:  integer;
    signal xb, yb, zb: std_logic_vector(7 downto 0);
begin
    gen: process
    begin
        x <=  10; y <= 20;  wait for 10 ns;
        x <=  44; y <= 33;  wait for 10 ns;
        x <=  55; y <= 66;  wait for 10 ns;
        x <= 144; y <= 56;  wait;
    end process;

    xb <= conv_std_logic_vector( x, 8 );
    yb <= conv_std_logic_vector( y, 8 );
    z  <= conv_integer( zb );

    DUT: zb <= xb + yb;

end BEH;
```

Esempio: Conversione da/verso interi



Interpretato con segno in modo corretto

La libreria STD

- La libreria STD contiene due packages
  - **`standard`**
  - **`textio`**
- Il package **`standard`** definisce
  - Tipi
  - Unità di misura del tempo
  - Files
- Il package **`textio`** definisce
  - Il tipo necessario alla gestione dei file di testo
  - Funzioni per la lettura e scrittura dei file

La libreria MODELSIM_LIB

- È specifica del simulatore Mentor Graphics ModelSim
- Contiene il solo package
  - util
- Il package util definisce alcune funzioni fra cui
  - **`get_resolution()`**
    - Ritorna la risoluzione temporale in uso
  - **`to_real( time: v )`**
    - Converte un valore di tipo **`time`** in **`real`**
  - **`to_time( real: v )`**
    - Converte un valore di tipo **`real`** in **`time`**
- Funzioni utili nella scrittura dei testbench

Il package TEXTIO

- Il package TEXTIO è definito in due librerie
  - Libreria STD

    ```
    use STD.textio.all;
    ```
  - Libreria IEEE

    ```
    use IEEE.std_logic_textio.all;
    ```
- Il package definisce
  - Due tipi di dati per file e linee di testo
  - Due funzioni per l'input
    - **`readline()`**, **`read()`**
  - Due funzioni per l'output
    - **`writeline()`**, **`write()`**

Il package TEXTIO: Apertura di un file

- L'apertura di un file avviene con la dichiarazione

  ```
  file <handle>: text is in "<file>";
  ```

- Il file è aperto in lettura/scrittura
  - Dipende solo dalle operazioni effettuate
- Da un file si leggono unicamente stringhe
- È necessario dichairare almeno una variabile per la lettura di linee di testo

  ```
  variable <var>: line;
  ```

- Il testo letto non è interpretato

Il package TEXTIO: Input

- **`readline(F,L)`**
  - Legge una linea dal file **`F`**
  - Lo memorizza nella variabile **`L`**
- **`read(L,V,B)`**
  - Estrae e converte un valore dalla linea **`L`**
  - Lo memorizza nella variablie **`V`**
  - Se la conversione ha successo **`B`** vale **`true`**
  - Il tipo della variabile **`V`** può essere
    - **`bit`**, **`bit_vector`**
    - **`integer`**, **`real`**
    - **`character`**, **`string`**
    - **`time`**

Il package TEXTIO: Output

- **`writeline(F,L)`**
  - Scrive sul file **`F`** la linea di testo **`L`**
- **`write(L,V,J,W)`**
  - Formatta il valore **`V`** e lo scrive nella variabile **`L`**
  - L'argomento **`J`** indica la giustificazione e può essere **`"left"`** o **`"right"`**
  - L'argomento **`W`** indica la dimensione del campo
  - Il tipo della variabile **`V`** può essere
    - **`bit`**, **`bit_vector`**
    - **`integer`**, **`real`**
    - **`character`**, **`string`**
    - **`time`**

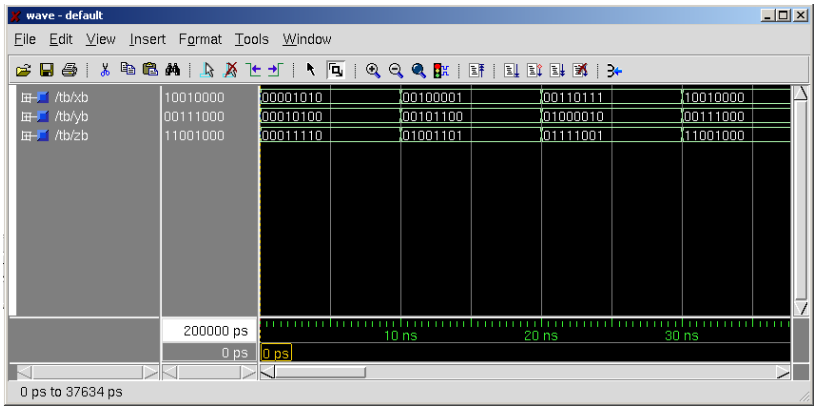
Esempio: Lettura da file

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_arith.all;
use IEEE.std_logic_unsigned.all;
library STD;
use STD.textio.all;

entity TB is
end TB;

architecture BEH of TB is
    signal   xb, yb, zb: std_logic_vector(7 downto 0);
begin
    gen: process
        file     fp:      text is in "in.dat";
        variable ln:      line;
        variable x, y, z: integer;
    begin
        readline( fp, ln ); read( ln, x ); read( ln, y );
        xb <= conv_std_logic_vector( x, 8 );
        yb <= conv_std_logic_vector( y, 8 );
        if endfile( fp ) = true then
            wait;
        else
            wait for 10 ns;
        end if;
    end process;

    DUT: zb <= xb + yb;
end BEH;
```

in.dat

```
10 20
33 44
55 66
144 56
```

Esempio: Lettura da file



Esempio: Lettura/scrittura da file

```
...

architecture BEH of TB is
    signal   xb, yb, zb: std_logic_vector(7 downto 0);
begin
   gen: process
        file     fpi: text open read_mode is "in.dat";
        variable lni: line;
        file     fpo: text open write_mode is "out.dat";
        variable lno: line;
        variable x, y, z: integer;
    begin
        readline( fpi, lni ); read( lni, x ); read( lni, y );
        xb <= conv_std_logic_vector( x, 8 );
        yb <= conv_std_logic_vector( y, 8 );
        wait for 9 ns;
        z := conv_integer( zb );
        write( lno, z );
        writeline( fpo, lno );
        if endfile( fpi ) = true then
            wait;
        else
            wait for 1 ns;
        end if;
    end process;

    ...

end BEH;
```

in.dat

```
10 20
33 44
55 66
144 56
```

out.dat

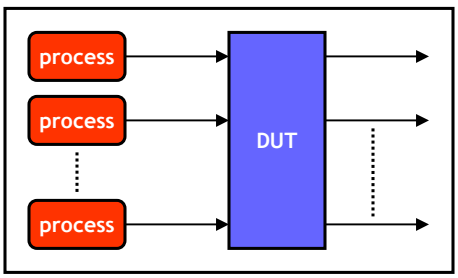
```
30
77
121
200
```

Testbench

- Ha lo scopo di verificare la funzionalità di un circuito
  - In altre parole, modella gli aspetti salienti dell'ambiente in cui il circuito si troverà ad operare
- Per ottenere questo
  - Deve generare ingressi significativi
    - Valori significativi
    - Temporizzazione
  - Deve mostrare i valori delle uscite
  - Eventualmente, eseguire controlli di correttezza
    - Per confronto con un gloden-model
    - Rispetto a valori noti

Testbench: Struttura di base

- Il testbench si limita a generare gli ingressi
- Il progettista controlla i valori di output analizzando le tracce ottenute in simulazione
  - Inapplicabile per problemi complessi



Testbench: Struttura di base

- I process per la generazione dei segnali di ingresso hanno la forma seguente

Segnali aperiodici

```
GEN_X: process
begin

    X <= value1 ;
    wait for time1 ns;

    X <= value2 ;
    wait for time2 ns;

    ...

    X <= valueN ;
    wait;

end process
```

Segnali periodici

```
GEN_X: process
begin

    X <= value1 ;
    wait for time1 ns;

    X <= value2 ;
    wait for time2 ns;

    ...

    X <= valueN ;
    wait for timeN ns;

end process
```

Testbench: Struttura di base

- Esempio
  - Segnali di clock e di reset

Reset

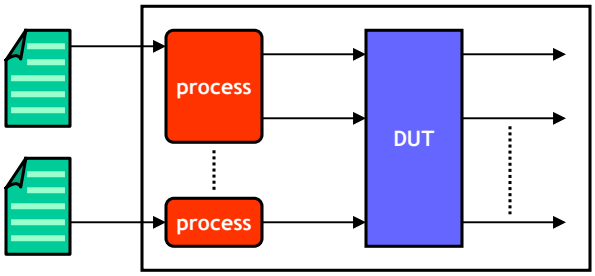
```
GEN_RESET: process
begin

    RESET <= '1';
    wait for 125 ns;

    RESET <= '0';
    wait;

end process
```

Clock

```
GEN_CLK: process
begin

    CLK <= '0';
    wait for 10 ns;

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

end process
```

Testbench: Lettura da file

- Il testbench si limita a generare gli ingressi
  - I valori degli ingressi sono letti da file
- Il progettista controlla i valori di output analizzando le tracce ottenute in simulazione
  - Inapplicabile per problemi complessi



Testbench: Lettura da file

- Un process
  - Legge il/i file di dati in variabili temporanee
  - Converte tali valori in un tipo adatto
    - Solitamente std_logic o std_logic_vector
  - Li assegna ai segnali d'ingresso del DUT
- Si hanno diverse possibilità
  - Un unico process genera clock e dati
  - Un process genera il clock, un altro i dati
  - Diversi process generano diversi dati
  - ...

Testbench: Lettura da file

- Esempio 1: temporizzazione esplicita
  - Lettura dei dati ogni 20 ns
  - **xb** e **yb** sono segnali **std_logic_vector** a 8 bit

```
READ_XY: process
    file     fpi:  text open read_mode is "in.dat";
    variable lni:  line;
    variable x, y: integer;
begin
    readline( fpi, lni );

    read( lni, x );
    xb <= conv_std_logic_vector( x, 8 );

    read( lni, y );
    yb <= conv_std_logic_vector( y, 8 );

    if endfile( fpi ) = true then
        wait;
    end if;

    wait for 20 ns;

end process;
```

in.dat

```
Xvalue1 Yvalue1
Xvalue2 Yvalue2
...
XvalueN YvalueN
```

Testbench: Lettura da file

- Esempio 2: temporizzazione implicita
  - Lettura dei dati in corrispondenza di un evento
  - **xb** e **yb** sono segnali **std_logic_vector** a 8 bit

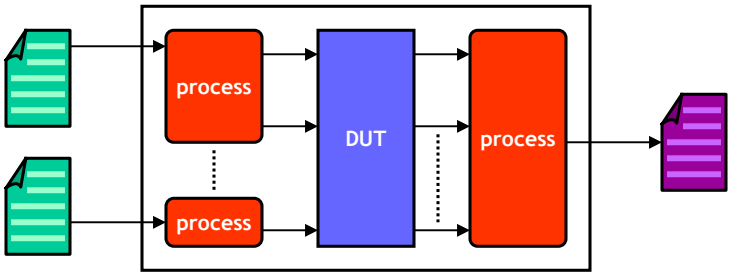
```
READ_XY: process( clk )
    file     fpi:  text open read_mode is "in.dat";
    variable lni:  line;
    variable x, y: integer;
begin
    if clk'event and clk = '1' then
        readline( fpi, lni );

        read( lni, x );
        xb <= conv_std_logic_vector( x, 8 );

        read( lni, y );
        yb <= conv_std_logic_vector( y, 8 );

        if endfile( fpi ) = true then
            wait;
        end if;
    end if;
end process;
```

in.dat

```
Xvalue1 Yvalue1
Xvalue2 Yvalue2
...
XvalueN YvalueN
```

Testbench: Lettura/scrittura di file

- Il testbench
  - Legge i valori degli ingressi da file
  - Scrive i valori delle uscite su file
- Il controllo è lasciato al progettista



Testbench: Lettura/scrittura di file

- Esempio
  - Lettura dei dati secondo uno degli schemi visti
  - Scrittura dei dati in corrispondenza di un evento

```
WRITE_XY: process( clk )
    file     fpo:  text open write_mode is "out.dat";
    variable lno:  line;
    variable x, y: bit_vector(7 downto 0);
begin
    if clk'event and clk = '1' then
        x <= to_bitvector( xb );
        write( lno, x, "left", 10 );

        y <= to_bitvector( yb );
        write( lno, y, "left", 10 );

        writeline( fpo, ln0 );
    end if;
end process;
```

Testbench: Foreign Language Interface

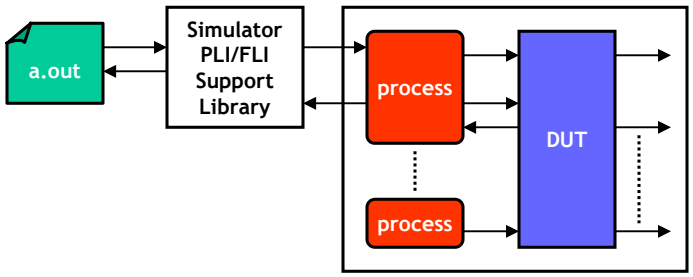
- I linguaggi di descrizione dell'hardware dispongono di un'interfaccia di programmazione per altri linguaggi
  - Tipicamente C/C++
  - Sono estensioni del linguaggio
  - Richiedono il supporto da parte del simulatore
- Per il linguaggio Verilog
  - PLI: Programming Language Interface
  - Schema semplice
- Per il linguaggio VHDL
  - FLI: Foreign Language Interface
  - Schema complesso, mutuato dal Verilog

Testbench: Foreign Language Interface

- Prescindendo dai dettagli specifici, il sistema PLI/FLI consente di
  - Dichiarare una interfaccia di funzione nel liguaggio HDL in uso (Verilog/VHDL)
  - Agganciare a tale interfaccia una funzione sviluppata in un altro linguaggio
    - Tipicamente C
  - Chiamare la funzione esterna
    - Il fatto che la funzione sia sviluppata in un linguaggio diverso è a queto punto indifferente
- Nella scrittura di test bench
  - Questo meccanismo è usato per generare dati

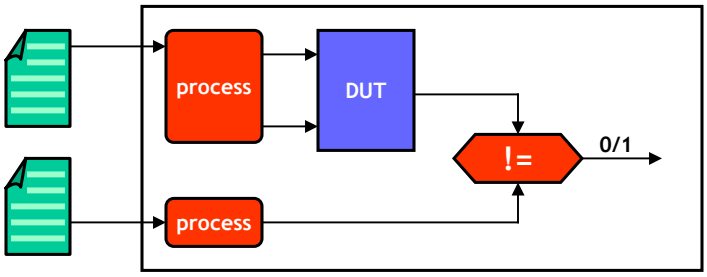
Testbench: Foreign Language Interface

- Può sostituire la lettura da file
  - Quando i dati sono di dimensioni eccessive
  - Quando i dati in ingresso dipendono dalle uscite
- Lo schema di base è il seguente



Testbench: Verifca automatica

- Il testbench
  - Legge i valori degli ingressi da file
  - Legge i valori delle uscite attese da file
- Il controllo è effettuato automaticamente



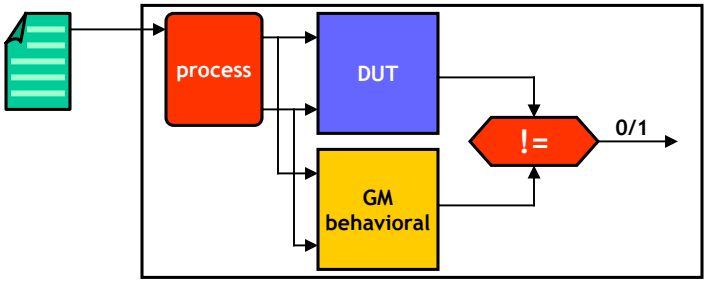
Testbench: Verifca automatica

- Esempio
  - Il controllo di correttezza avviene in un process sincronizzato da un opportuno evento
    - Tipicamente un clock
  - In caso contrario il testbench può essere inusabile per la verifica del design dopo mapping e layout
    - La simulazione timing introduce ritardi di cui bisogna tenere conto

```
CHECK_XY: process( clk )
begin
    if clk'event and clk = '1' then
        if X /= Xexp or Y /= Yexp then
            XYerr <= '1';
        else
            XYerr <= '0';
        end if;
    end if;
end process;
```

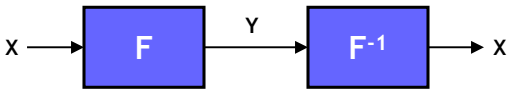
Testbench: Verifca automatica

- Il testbench
  - Legge i valori degli ingressi da file
  - Fornisce gli ingressi a DUT e ad un golden model
- Il controllo è effettuato automaticamente



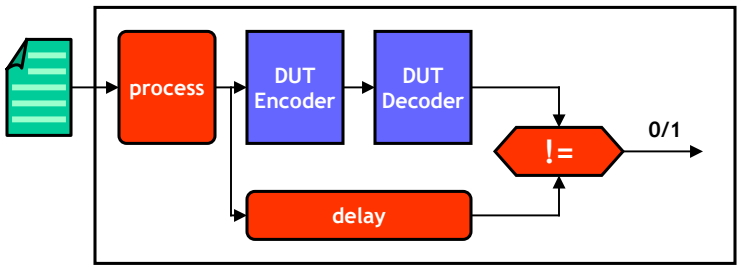
Testbench: Verifca intrinseca

- Alcuni sistemi sono scostituiti da due parti che svolgono funzionalità inverse
  - Codifica/decodifica
  - Compressione/decompressione
  - Modulazione/demodulazione
  - Trasformata/trasformata inversa
  - ...
- È possibile utilizzare una delle due sezioni per la verifca dell'altra e viceversa



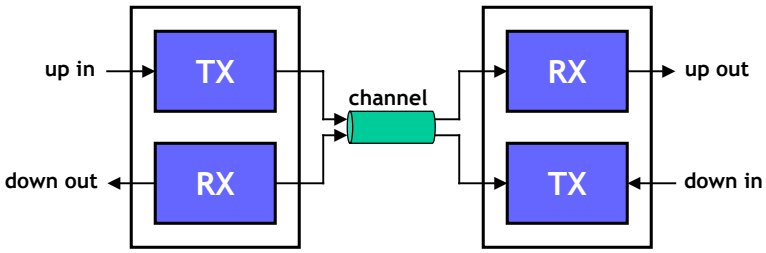
Testbench: Verifca intrinseca

- Esempio: encoder/decoder
  - Il testbench produce gli ingressi
  - Encoder e decoder sono connessi in cascata
- Il testbench confronta le uscite del decoder con gli ingressi, opportunamente ritardati



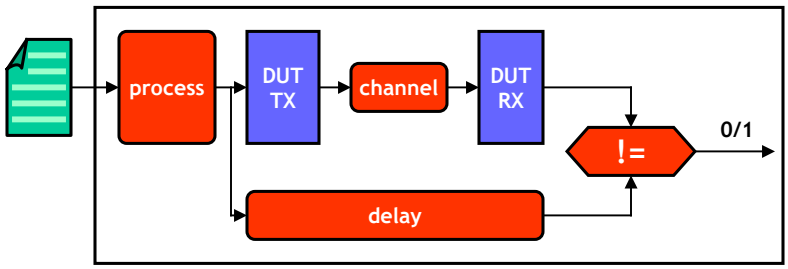
Testbench: Verifca intrinseca

- In alcuni casi è necessario tenere conto dell'ambiente nella verifica di tipo intrinseco
- Si consideri ad esempio un sistema di trasmissione
  - Volendo verificare la qualità del sistema è necessario modellizzare il canale
    - Tipicamente come sorgente di errori casuali



Testbench: Verifca intrinseca

- Esempio: trasmettitore/canale/ricevitore
  - Il testbench produce gli ingressi
  - Trasmettitore e ricevitore sono connessi in cascata mediante il modello del canale
- Il confronto avviene come già discusso



Testbench: Verifca intrinseca

- È possibile estendere il modello di testbench appena visto introducedo moduli (process) aggiuntivi
  - Calcolo del BER (Bit Error Rate)
    - Un process conta i bit ricevuti ed il numero di errori rilevati e calcola il BER ad ogni istante
  - Calcolo del MTBF (Mean Time Between Faults)
    - Un process conta i cicli di clock e memorizza i tempi a cui si verificano gli errori
- Questo tipo di misure non fa parte a rigore del processo di verifica
  - Più spesso si ricorre alla prototipazione e alla misurazione sperimentale