Università degli Studi di Messina
Dipartimento di Ingegneria

Introduzione al progetto di sistemi digitali mediante VHDL

Ultima modifica 4 ottobre 2021

Author
Carmine Ciofi

4 ottobre 2021

Indice

1 Linguaggi di descrizione dell'hardware (HDL) **1**
1.1 Introduzione 1
1.2 Fondamenti di VHDL : il simulatore VHDL 2
1.3 Fondamenti di VHDL : entità e architettura 4

Capitolo 1

Linguaggi di descrizione dell'hardware (HDL)

1.1 Introduzione

I linguaggi di descrizione dell'hardware o HDL (Hardware Description Language) nascono dall'esigenza di disporre di linguaggi formali per descrivere in modo completo e non ambiguo il comportamento di sistemi digitali. I linguaggi HDL più comunemente usati sono il Verilog e il VHDL. In queste note faremo esclusivo riferimento al VHDL. Ci sono diverse ragioni per le quali può essere utile, o necessario, ricorrere a un linguaggio formale per la descrizione dell'hardware.

Nel caso di un hardware digitale già realizzato e disponibile, il ricorso a un linguaggio formale di descrizione dell'hardware rappresenta un modo efficace e non ambiguo di documentare il comportamento dal circuito.

Nel caso in cui si stia progettando un nuovo sistema digitale, il ricorso a un linguaggio formale di descrizione dell'hardware rappresenta un modo efficace per definire in modo chiaro e non equivoco le specifiche del sistema, ovvero il comportamento che si vuole ottenere dal sistema che si intende realizzare.

Nel caso in cui il sistema da realizzare sia composto da più sottosistemi che devono operare in maniera coordinata, disporre di una descrizione formale dei sottosistemi e del modo in cui sono interconnessi consente di eseguire delle simulazioni per verificare che il comportamento del sistema, per come risulta dalla struttura impiegata, sia consistente con le specifiche desiderate.

Nel caso in cui si disponga di una descrizione formale sufficientemente dettagliata, è generalmente possibile arrivare a una sintesi automatica, mediante l'interconnessione di porte elementari standardizzate, del sistema digitale descritto.

Da un altro punto di vista è evidente che se si procede al progetto di un sistema digitale facendo uso di un linguaggio di descrizione dell'hardware, questo fatto si traduce, di per sé, nella disponibilità di una descrizione non equivoca del suo comportamento, nella possibilità di eseguire simulazioni per verificare che il comportamento ottenuto sia conforme a quanto desiderato, nella disponibilità di una base per procedere, ove possibile, alla

sintesi automatica del circuito per una sua implementazione mediante ASIC o mediante FPGA.

Come si può facilmente comprendere, non tutto ciò che può essere immaginato, ancorché corretto da un punto di vista logico, può trovare realizzazione mediante blocchi logici standard. Si supponga per esempio che i blocchi logici standardizzati di cui disponiamo siano costituito da blocchi logici combinatoriali con la possibilità della presenza di un flip flop non trasparente (un registro) che può operare esclusivamente o sul fronte di salita o sul fronte di discesa di un segnale di clock. In una situazione del genere è evidente che se è da un lato possibile immaginare un sistema logico sequenziale sincronizzato che operi sia sul fronte di salita sia su quello di discesa, in nessun caso questo sistema potrà essere sintetizzato (leggi "realizzazione mediante l'interconnessione dei blocchi logici disponibili") perché non si dispone di registri che possano operare sia sul frnte di salita sia sul fronte di discesa del clock.

Anche se il fine ultimo di questo corso è quello di arrivare ad essere in grado di progettare sistemi complessi per la sintesi automatica su dispositivi FPGA, nella prima parte di queste note non ci occuperemo dei problemi relativi alla possibilità di sintetizzare i sistemi che verranno descritti in VHDL: ci occuperemo principalmente di come il VHDL possa essere usato per descrivere un sistema logico e di come si possa procedere alla simulazione del comportamento del sistema stesso a partire dalla sua descrizione in VHDL.

1.2 Fondamenti di VHDL : il simulatore VHDL

Anche se da ora in poi faremo esplicito riferimento al VHDL, molti dei concetti fondamentali che verranno discussi in questa sezione si applicano a tutti i linguaggi di descrizione dell'hardware.

Per capire come sia possibile descrivere il comportamento di un circuito digitale mediante VHDL bisogna avere ben presente il modello di evoluzione dei circuiti logici che viene assunto dal linguaggio. Siccome il modello di evoluzione assunto dal linguaggio coincide con il modello usato per eseguire la simulazione del circuito stesso a partire dalla descrizione in VHDL, per capire i fondamenti del VHDL è utile partire dal comportamento del simulatore VHDL.

In una descrizione VHDL un sistema digitale è chiamato "entità" (parola chiave "entity"). Un sistema digitale può essere ottenuto dalla interconnessione di altri sistemi digitali, ovvero essere il risultato della interconnessione di altre entità. Una entità è caratterizzata da un certo numero di ingressi e un certo numero di uscite. Per semplicità in questa sede assumeremo che tutti gli ingressi e tutte le uscite corrispondano a variabili booleane (in linguaggio VHDL si direbbe che appartengono al tipo predefinito "bit").

Con riferimento alla Fig.1.1 l'entità E_3 rappresenta il circuito digitale descritto in VHDL che ha come ingressi x_1 ed x_2 e come uscite y_1 e y_2. Come indicato dalle aree tratteggiate che vogliono suggerire una vista della "struttura interna", l'entità E_3 si suppone costituita dall'interconnesione di moltre alte entità. Gli elementi (i "fili", semplificando al massimo) di collegamento fra ingressi e uscite di altre entità si chiamano "segnali" (parola chiave "signal") in VHDL. Sono i segnali a stabilire come ogni entità è collegata ad ogni altra. In un certo senso, per chi ha familiarità con la sintassi del simulatore circuitale SPICE, i

segnali in VHDL svolgono lo stesso ruolo dei nomi di nodo nella descrizione di un circuito in accordo con la sintassi SPICE.

Come si vede dalla Fig.1.1, il sistema che siamo interessati a simulare è parte di un'altra entità (E_0) all'interno della quale sono comprese due ulteriori entità (E_1 e E_2) che hanno il compito di generare i segnali di ingresso per l'entità E_3. Il motivo per il quale è necessario ricorrere all'entità "contenitore" E_0 che, come si nota dalla figura, non ha né ingressi né uscite, è che la simulazione VHDL non può tenere conto di variazioni su ingressi indotte da sistemi esterni al simulatore stesso. Le sollecitazioni sul sistema, che dal nostro punto di vista sono esterne al sistema E_3, con il ricorso al sistema "contenitore" E_0 sono, a tutti gli effetti, parte del sistema stesso.

L'elemento fondamentale alla base del meccanismo di simulazione del VHDL è il concetto di evento (parola chiave "event"). Si ha un evento all'istante di tempo t_i se un segnale cambia valore rispetto agli istanti di tempo precedenti. In generale, compito ci ciascuna entità è quella di programmare nuovi eventi (eventi in tempi futuri) ogni volta che si verifica un evento su un segnale collegato a un suo ingresso (vedremo più avanti il caso delle entità che non hanno ingressi). Gli eventi programmati per tempi futuri possono interessare segnali interni all'entità o porte di uscita dell'entità stessa (più propriamente gli eventi interesseranno i segnali collegati alle porte di uscita).

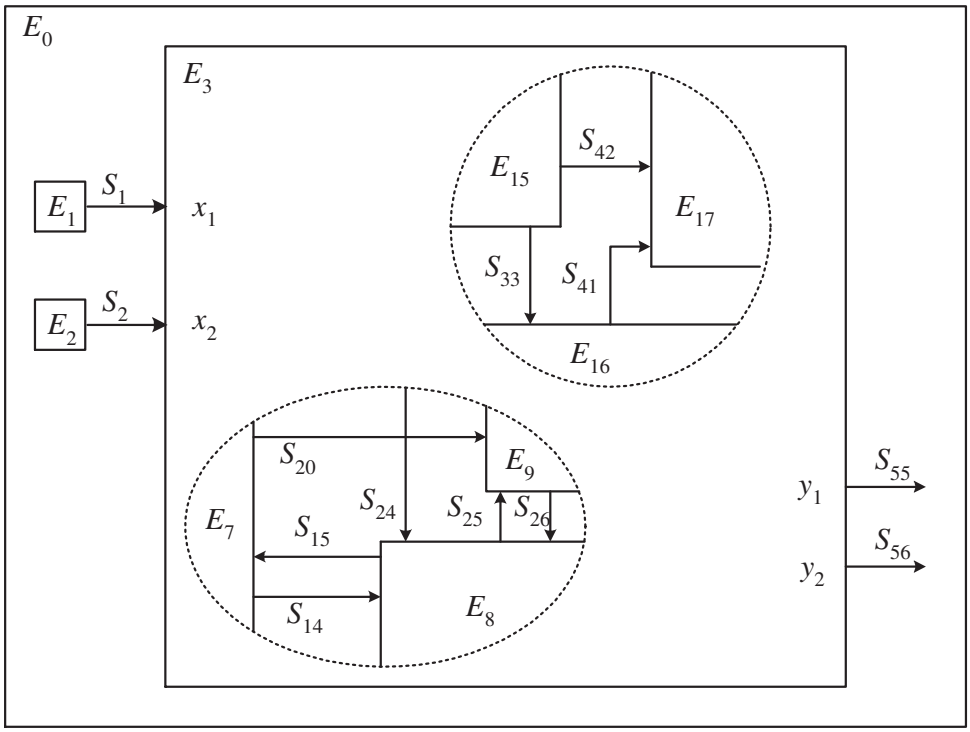

Figura 1.1: Sistema digitale generico.

Il simulatore VHDL mantiene costantemente aggiornata una lista degli eventi, con gli eventi ordinati temporalmente. Il simulatore VHDL tiene nota del tempo trascorso (il tempo "simulato" di evoluzione del sistema). Si rifletta però sul fatto che qualunque cambiamento dello stato del sistema digitale non può che manifestarsi nella forma di un evento, ovvero di cambiamento di valore di un segnale. Pertanto, gli unici istanti di tempo che sono rilevanti sono quelli in cui avvengono eventi. Il simulatore VHDL non segue pertanto l'evoluzione temporale del sistema "istante per istante" che, oltre

che impossibile (gli istanti di tempo in un qualunque intervallo finito di tempo sono infiniti), sarebbe inutile, ma scorre la lista degli eventi, passando da un evento a quello immediatamente successivo. E' bene tenere presente che la lista degli eventi si modifica continuamente, giacché in corrispondenza di ogni evento le diverse entità che compongono il sistema possono programmare nuovi eventi (ovviamente solo nel futuro) e ciascuno dei nuovi eventi programmati entra a far parte della lista degli eventi. La simulazione, in VHDL, procede nel modo seguente: si cerca nella lista il primo evento (primo in ordine temporale); si fa avvenire l'evento (per esempio si fa passare un certo segnale da '0' a '1' o viceversa); si "congela" lo stato del sistema e si esaminano tutte le entità per verificare se, come parte della loro definizione, è previsto che esse programmino nuovi eventi in risposta all'evento appena avvenuto. Tutti i nuovi eventi (programmati per il futuro) vengono inseriti nella lista degli eventi nella posizione specificata da ciascuna entità; completato l'inserimento dei nuovi eventi nella lista degli eventi, il simulatore esamina nuovamente la lista degli eventi e passa all'evento futuro più vicino e si ripete quanto fatto per l'evento precedente. La simulazione si completa quando si esaurisce la lista degli eventi oppure quando viene raggiunto il tempo massimo di simulazione predeterminato.

Sulla base di quanto sopra esposto, nuovi eventi nascono come conseguenza di altri eventi e i segnali assumono specifici valori in corrispondenza di eventi. Come è evidente rimane da specificare come, all'inizio dei tempi, sono stabiliti i valori iniziali di tutti i segnali e come possano essere inseriti i primi eventi nella lista degli eventi, in assenza dei quali il meccanismo di simulazione, per come descritto, non potrebbe procedere.

Proprio perché il VHDL nasce dall'esigenza di descrivere in maniera univoca e non ambigua il comportamento di un sistema digitale, questi due problemi sono affrontati e risolti in modo assolutamente univoco e definito. Per quanto riguarda il valore iniziale assunto dai segnali, visto che per il momento ci siamo limitati a segnali di tipo "bit" che possono assumere solo i valori '0' e '1', all'inizio dei tempi tutti i segnali assumono il valore '0'[1]. Inoltre, l'inizio dei tempi viene trattato come un evento "attiva" tutte le entità, le quali, sulla base di questa attivazione, anche in assenza di specifici eventi sui segnali (siamo all'inizio dei tempi) possono, secondo meccanismi che impareremo a conoscere, programmare eventi.

1.3 Fondamenti di VHDL : entità e architettura

L'entità, per qunato abbiamo visto fino a questo punto, è il blocco funzionale fondamentale attraverso il quale si può descrivere un sistema digitale complesso in VHDL. Alla definizione di una entità si arriva mediante due blocchi di codice VHDL distinti. Il primo blocco è relativo alla dichiarazione dell'entità (costrutto *entity*) il secondo blocco è relativo alla specifiche di funzionamento dell'entità stessa, ovvero al modo in cui l'entità risponde a eventi ai suoi ingressi (costrutto *architecture*).

Nel caso più semplice, la dichiarazione di entità contiene soltanto il nome dell'entità stessa e la dichiarazione delle porte di ingresso e di uscita. Nel dichiarare una nuova entità non

[1] il tipo bit è un tipo definito per enumerazione con due soli elementi che sono '0' e '1'. I segnali possono in realtà appartenere a molteplici tipi. Tutte le volte che i segnali appartengono a un tipo definito per enumerazione (più avanti avremo a che fare con il tipo definito per enumerazione "std_logic" che contiene 9 elementi), all'inizio dei tempi il valore iniziale dei segnali è quello "più a sinistra" nell'elenco che definisce i valori che possono essere assunti.

si vincola in alcun modo il suo comportamento, se non per il fatto di definire il numero e il tipo dei suoi ingressi e delle sue uscite.

Una volta definita l'entità, si può usare il costrutto "architecture" per specificare in dettaglio il suo comportamento. Per semplicità nel seguito faremo riferimento al costrutto "architecture" con il termine italiano di "architettura". E' bene chiarire sin da ora che ad una entità dichiarata possono corrispondere più architetture. Il VHDL mette a disposizioni opportuni meccanismi per associare, volta per volta,l'architettura desiderate fra quelle disponibili per una stessa entità. Può essere utile, a questo punto, procedere alla descrizione e alla simulazione di una entità estremamente semplice, ovvero di un inverter, cosa che ci consentitrà da un lato di familiarizzare con la sintassi VHDL, dall'altro di puntualizzare e chiarire alcune delle conseguenze insite nel processo di simulazione descritto nel paragrafo precedente. Iniziamo quindi a dichiarare l'entita che svolgerà la funzione di inverster e decidiamo di darle il nome di "invertitore". Questa entità avrà un solo ingresso, che chiameremo "a" e una sola uscita che chiameremo "y". Sia all'ingresso sia all'uscita supporremo di collegare segnali di tipo bit e pertanto le porte di ingresso e di uscita saranno anch'esse di tipo bit. Per la dichiarazione dell'entità è sufficiente predisporre un file di testo con il contenuto seguente:

```
entity invertitore is
        port(a:in bit; y:out bit);
end entity invertitore;
```

Listing 1: Dichiarazione dell'entità invertitore.

Le parole evidenziate in verde sono parole chiave del linguaggio e non possono essere usate come nomi per oggetti definiti dall'utente. Le parola "bit" identifica il tipo delle porte di ingresso e di uscita. Si presti attenzione al fatto che il VHDL non distingue fra lettere maiuscole e minuscole, sia nel caso di identificatori, sia nel caso di parole chiave del linguaggio (più avanti vederemo alcune eccezioni).

Un file che contenga la definizione di una entità costituisce un blocco di codice che può essere esaminato dal compilatore VHDL per verificarne le correttezza formale. Se la definizione di entità risulta corretta, la sua esistenza, il suo nome e le sue caratteristiche vengono memorizzate un un data-base locale. Questa operazione, da parte del compilatore VHDL è necessaria perché quando verrà eseminato, in un momento successivo, il file che contiene la definizione del comportamento (blocco "architecture") il nome dell'entità e le sue caratteristiche devono essere già note.

Supponiamo ora di volere fornire una descrizione del comportamento dell'entità appena dichiarata. Per fare ciò dobbiamo predisporre una "architettura". Possiamo predisporre l'architettura in un file separato da quello della definizione dell'entità. La descrizione del comportamento si ottiene avendo ben presente il modo di operare del VHDL. Una possibie architettura per l'inverter è riportata nel listing 2.

Nel seguito faremo riferimento al simulatori ghdl. Parttiamo da una directory vuota nella quale creeremo tutti i file necessari a descrivere completamente il sistema al quale siamo interessati. Supponiamo allora di aver preparato un file di testo dal nome `first_entity_inv.vhd` che contiene il codice nel listing 2.

```
architecture mia_arc of invertitore is
-- sezione dichiarativa dell'architettura
-- che per il momento è vuota
begin
   process (a) is
   begin
       if (a='1') then
           y<='0' after 10 ns;
       else
           y<='1' after 29 ns;
       end if;
   end process;
end architecture mia_arc;
```

Listing 2: Architettura per l'entità invertitore.

Nella prima riga di codice dichiariamo di voler definire una architettura (parola chiave `architecture`) dal nome `mia_arc` per l'entità `invertitore`. Come già detto, possono essere definite più architetture per una stessa entità. Il VHDL prevede meccanismi per selezionare quale architettura deve essere usata in ogni specifica simulazione o nella fase di sintesi automatica del sistema. Fra la prima riga e la riga in cui compare la parola chiave `begin` per la prima volta è presente la sezione dichiarativa dell'architettura, dove, fra le altre cose, si possono dichiarare segnali che possono essere usati all'interno dell'architettura. In questo esempio non facciamo uso della parte dichiarativa. Si noti che in VHDL tutto quanto segue un "doppio meno" (`-`) viene considerato un commento e viene ignorato. Più precisamente, nell'esaminare una riga, se viene incontrato un "doppio meno" vengono ignorati i caratteri "`-`" e tutti i caratteri che seguono fino a fine riga. La descrizione del comportamento dell'architettura inizia dopo la parola chiave `begin` e può essere ottenuta mediante diversi costrutti. In questo primo esempio facciamo uso del costrutto `process`. Prima di procedere oltre, si noti che non sono presenti nell'architettura dichiarazioni che fanno riferimento alla natura di "a" e di "y": la natura (ovvero il ruolo di porte di ingresso o di uscita e il tipo) di "a" e "y" sono noti perché dichiarati nell'entità per la quale stiamo scrivendo l'architettura. Per comprendere quale sia il senso di quanto contenuto all'interno di un blocco process dobbiamo ricordare come opera il simulatore VHDL: tutte le volte che si verifca un evento, vengono prese in esame tutte le entità e si controlla se l'evento appena avenuto produce altri eventi da inserire nella lista degli eventi. Con questo in mente, lo statement "`process(a)`" può essere interpretato come segue: se, a un certo passo di simulazione VHDL (ovvero quando stiamo prendendo in considerazione un evento nella lista degli eventi) capita che questo evento sia avvenuto su "a", allora il comportamento dell'entità deve essere quello che viene descritto di seguito (fra le parole chiave `begin` e `end process;`). Si ricordi sempre che per "comportamento dell'entità si intende sempre e solo l'eventuale inserimento di eventi nella lista degli eventi generale. Nel nostro sempice esempio c'e' il solo "`a`" in ingresso all'entità e non ci sono altri segnali interni all'architettura che possono subire eventi. In generale, però, possono verificarsi eventi su più ingressi e segnali e la descrizione del comportamento dell'entità può avvenire mediante più processi, che operano in parallelo, ciascuno dei quali è sensibile a un diverso gruppo di eventi. Il gruppo di eventi al quale un process è sensibile si specifica elencando fra parentesi le porte di ingresso o i segnali i cui eventi devono attivare il

process. Questo elenco prende il nome di "sensitivity list". Nel nostro esempio semplice facciamo uso di un solo process nella cui sensitivity list compare esclusivamente la porta di ingresso "a". Ricordiamo che se è avvenuto un evento su "a", nel momento in cui prendiamo in esame il corpo del process, "a" ha il valore assunto per effetto dell'evento: se l'evento è consistito nel passaggio dal valore '0' al valore '1' di un segnale collegato alla porta a dell'entità, all'interno del process "a" ha il valore '1'. Come si può facilmente intuire, anche se non conosciamo ancora i dettagli del linguaggio VHDL, all'interno del process si compiono azioni distinte a seconda l'unico comportamento possibile dell'entità quando l'evento su "a" è consistito in un suo passaggio al valore '1' o meno. Si noti che visto che "a" appartiene al tipo bit, l'unica altra possibilità di avere un evento su "a" senza che questo assuma il valore '1' è che "a" abbia subito un evento che lo ha fatto passare al valore '0' a partire dal valore '1'. La coppia di caratteri "<=" viene utilizzata in VHDL come un unico simbolo che indica che si sta programmando un evento sulla porta (o, se si vuole, sul segnale che sarà collegato alla porta) alla sinistra del simbolo stesso. A destra del simbolo viene indicato il valore che dovrà assumere la porta ("y" in questo caso) e dopo quanto tempo questo valore verrà assunto rispetto al tempo in cui è evento l'evento che sta portando alla programmazione del nuovo evento. Supponiamo per esempio che al tempo $t = 500$ ns "a" sia passato da '0' a '1'. Quando il simulatore VHDL arriva a esaminare gli eventi che avvengono a $t = 500$ ns, siccome fra questi c'e' un evento su "a", e a seguito dell'evento "a" ha assunto il valore '1', il process dell'architettura porterà alla programmazione dell'evento "y diventa '0' al tempo $t = 500 + 10 = 510$ ns. A questo punto il compito del process è concluso e il process, per così dire, resterà in attesa del prossimo evento su "a" per procedere, eventualmente, all'inserimento di nuovi eventi nella lista degli eventi. Resta da esaminare cosa accade all'inizio dei tempi. Nella sezione precedente avevamo detto che tutti i segnali di tipo bit assumono il valore '0'. Pertanto anche "a" al tempo $t = 0$ vale '0' perché varrà zero il segnale ad esso collegato nel sistema complessivo. Varrà inoltre '0' anche il segnale collegato alla porta di uscita "y". Si noti che non possiamo propriamente dire che "y" vale '0' perché all'interno di una architettura (e quindi di un process) no c'e' modo di accedere ("di leggere") il valore di una porta di uscita (ovvero dichiarata out al momento della creazione dell'entità). Inoltre, avevamo affermato genericamente che all'inizio dei tempi tutte le entità erano chiamate a intervenire. Ora che abbiamo visto che a determinare la programmazione di eventi sono i process, possiamo più propriamente dire che all'inizio dei tempi tutti i process di tutte le entità vengono attivati, indipendentemente dalla sensitivity list di ciascuno di essi. Questo significa, nel nostro caso, che all'inizio dei tempi, e indipendentemente dal comportamento futuro dell'ingresso "a", verà programmato l'evento "y diventa '1' a $t = 0 + 29$ ns.

Ora che abbiano creato una entità e abbiamo definito un suo possibile comportamento, supponiamo di volere arrivare a verificarne il funzionamento mediante il simulatore VHDL. Come abbiamo detto nella sezione precedente, dobbiamo riuscire a create una entità che sia in grado di generare un segnale di test da mandare in ingresso all'inverter, dopo di che dobbiamo costruire una entità sneza ingressi e senza uscite, che ontiene al suo interno il generatore e l'inverter per poter procedere alla simulazione.

Cominciamo a vedere come si può creare una entità con 0 ingressi e una sola uscita capace di generare un segnale di test. Supponiamo di chiamare questa entità "sorgente". Un esempio di sorgente (dichiarazione di entità e relativa architettura è riportata nel listing 3

```vhdl
entity sorgente is
        port (y:out bit);
end entity sorgente;

architecture prova of sorgente is
begin
        process is
        begin
                wait for 100 ns;
                y<='1';
                wait for 50 ns;
                y<='0';
                wait for 100 ns;
                y<='1';
                wait for 50 ns;
                y<='0';
                wait for 100 ns;
                wait;
        end process;
end architecture prova;
```

Listing 3: Dichiarazione di entità e architettura per il generatore di stimolo.

Nell'architettura dell'entità `sorgente` si sfrutta il fatto che tutti i process, inclusi quelli senza alcuna sensitivity list, vengono attivati all'inizio dei tempi. Senza per ora entrare nei dettagli della sintassi usata, tutto quanto scritto nel corpo del process nel listing 3 corrisponde, al tempo zero, a inserire l'evento `y<='1'` al tempo $t = 0 + 100$ ns, l'evento `y<='0'` al tempo $t = 0 + 100 + 50$ ns e così via. L'ultimo statemant `wait;` senza alcuna specifica di tempo corrisponde a escludere ongli ulteriore possibile attivazione del process. In sostanza, il process svolge il compito, al tempo $t = 0$, di riempire la lista degli eventi secondo le specifiche date.

A questo punto dobbiamo realizzare un sistema (una entità) senza ingressi e senza uscite che consiste nell'opportuno collegamento delle due entità sviluppate fino a questo punto. Chiameremo questa entità `testbench` e per la sua architettura sfrutteremo la possibilità, offerta dal VHDL, di definire il comportamento di una entità mediali il collegamento di altre entità.

La dichiarazione di entità e l'architettura (che chiameremo `struct`) dell'entità `testbench` è riportata nel listing 4.

La dichiarazione dell'entità `tetsbench`, in assenza di ingressi e uscite, si riduce alle linee 1 e 2 nel listing 4. Nella parte dichiarativa dell'architettura sono introdotti due segnali di tipo `bit`: `sin` e `sout` (linee 6 e 7). Il primo segnale svolge la funzione di collegamento fra l'uscita dell'entità che genera il seganle di ingresso e l'ingresso dell'invertitore; il secondo segnale è semplicemente collegato all'uscita dell'invertore, in modo che sia possibile conoscere lo stato dell'uscita in fase di verifica della simulazione. Il comportamento dell'entità (corpo dell'architettura) è in questo caso il risultato del collegamento di due "copie" (in inglese si direbbe "instances") delle entità definite in precedenza. Per poter usare altre entità come parte di una architettura, il compilatore/simulatore VHDL deve essere a

```
1  entity testbench is
2  end entity testbench;
3
4  architecture struct of testbench is
5
6  signal sin:bit;
7  signal sout:bit;
8  begin
9
10 the_inverter:entity work.invertitore(mia_arc)
11                 port map(a=>sin, y=>sout);
12
13 the_source:entity work.sorgente(prova)
14         port map (y=>sin);
15 end architecture struct;
```

Listing 4: Testbench per la simulazione dell'invertitore. I numeri di riga non fanno parte del file. Sono stati aggiunti per semplificare la discussione

conoscenza della loro esistenza e della loro architettura. Ogni volta che si definisce una entità o una architettura, questa può essere farra analizzare dal compilatore VHDL. Una volta analizzate, se sintatticamentye corrette, queste entrano a far parte della libreria predefinita `work`, accessibile dalla directory in cui si sta lavorando. Per usare una copia di una entità occorre darle un nome (nella forma di una label, vedi per esempio `the_inverter` alla linea 10 e `the_source` alla linea 13) e specificare di quale entità si tratti (nella forma `work.<nome_entità>` come nelle linee 10 e 13), e anche quale delle possibili architetture, fra quelle disponibili, deve essere usata. [2]

Il collegamento fra le entità si ottiene facendo ricorso ai segnali definiti nel corpo dell'architettura, indicando quale segnale è collegato a ciascuna porta. Per esempio, nella linea 11 si specifica che la porta `"a"` dell'entità `the_inverter` è collagata (`"=>"`) al segnale `sin`, mentre la porta `"y"` della stessa entità viene collegata al segnale `sout`. ALlo stesso modo, nella linea 14 si specifica che la porta `"y"` dell'entità `the_source` deve essere collegata al segnale `sin`. Come è intuibile, porte collegate allo stesso segnale sono a tutti gli effetti collegate fra di loro.

Ora che abbiamo a disposizione tutti gli elementi e file necessari, possiamo procedere alla simulazione del sistema. A questo scopo faremo uso del compilatore/simulatore VHDL chiamato GHDL. Il software GHDL è un software di pubblico dominio ed è incluso nella macchina virtuale linux a disposizione degli studenti.

I passaggi necessari per la simzulazione sono illustrati nel video prima_simulazione_vhdl.mp4 reperibile dalla pagian web `celec.org`.

[2] Il fatto che si possano usare nel corpo di architetture altre entità solo se queste sono già state completamente definite consente una agevole progettazione di tipo bottom-to (dal dettaglio al generale), ma limiterebbe fortemente un approccio di progettazione di tipo top-bottom. Per risolvere questo problema si ricorre ad altri modi di descrivere la struttura di un sistema che vedremo in seguito.