Laboratorio di Sistemi per la Progettazione Automatica
a.a. 2008/09

Giuseppe Di Guglielmo
Università degli Studi Di Verona
Dipartimento di Informatica

Revisione: mercoledì 18 marzo 2009 - 16.19

Lezione 2: Simulazione design VHDL

In questa lezione prosegue l'esplorazione delle funzionalità di ModelSim di Mentor Graphics, in particolare vengono analizzati ulteriori aspetti riguardanti la simulazione e l'analisi di descrizioni VHDL attraverso alcuni esercizi.

Questo tutorial è stato realizzato utilizzando la versione **6.4b di Modelsim** su piattaforma Linux. Di seguito i comandi riportati con sfondo grigio (`$`) sono da intendersi come comandi della console Linux, mentre i comandi riportati con sfondo rosa (`>`) sono da intedersi come comandi di ModelSim.

Gli esempi di questa lezione sono posizionati in:

```
/home/gdg/teaching/spa_lab/aa_2008_09/lesson_2
```

Generazione dei database delle forme d'onda

I risultati di una sessione di simulazione di ModelSim possono essere salvati in *file WLF* (Wave Log Format). Un file WLF è un archivio compresso contenente informazioni riguardanti sia le forme d'onda dei segnali in ingresso, interni e in uscita sia informazioni strutturali delle descrizione VHDL.

Ogniqualvolta si aggiungano segnali al pannello Wave, i relativi risultati vengono memorizzati in un file di default (`vsim.wlf`). Per ogni nuova simulazione il file vsim.wlf verrà sovrascritto.

Se si vuole salvare il file WLF per la simulazione corrente si può aggiungere l'argomento `-wlf <filename>` al comando `vsim`:

```
$ vsim -wlf waveform_examples/wave_1.wlf work.ex_1
```

La simulazione in questo caso andrà conclusa specificando il comando `quit -sim` al fine di produrre un file WLF valido:

```
VSIM > do stimuli_ex1_ex2/stimuli_0.do
VSIM > quit -sim
ModelSim >
```

Nota 1:
Il commando `vsim` può essere utilizzato sia dalla shell di Linux che dal prompt dei comandi di ModelSim (pannello *Transcript*):

```
ModelSim > vsim -wlf waveform_examples/wave_2.wlf work.ex_1
VSIM > do stimuli_ex1_ex2/stimuli_0diff.do
VSIM > quit -sim
ModelSim >
```

Nota 2:
Per visualizzare il contenuto di un file WLF da linea di comando è possibile utilizzare `dumplog64`. Tale comando estrae le informazioni rappresentate in formato binario e compresse del file WLF passatogli come argomento:

```
ModelSim > dumplog64 waveform_examples/wave_1.wlf
```

Confronto delle forme d'onda

ModelSim offre la possibilità di confrontare il comportamento delle simulazioni attraverso il confronto delle forme d'onda relative ai segnali coinvolti in ciascuna simulazione.

Dopo aver generato i file WLF relativi a simulazioni differenti (delle stessa descrizione), si può accedere a ***Tools >> Waveform Compare >> Comparison Wizard***.

A questo punto è necessario:

- caricare i due file WLF rispettivamente come ***Reference Database*** e ***Test Database***;
- specificare il ***Comparison Method***, ad esempio è possibile confrontare tutti i segnali coinvolti nella simulazione oppure scegliere un sott'insieme dei segnali;
- calcolare le differenze;
- a termine delle operazioni di confronto uscire dall'ambiente con ***Tools >> Waveform Compare >> End Comparison***

L'interfaccia grafica produrrà un rappresentazione delle forme d'onda evidenziando in rosso i punti in cui esse si differenziano.

Esercizio 1

Si consideri la macchina a stati finiti di *Mealy* rappresentata in Figura 1.

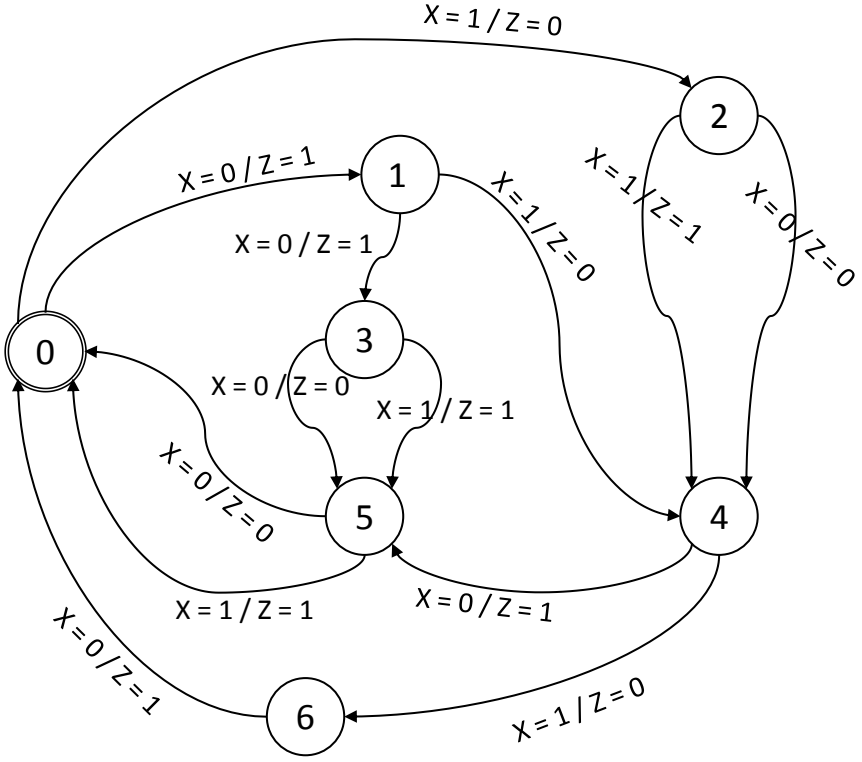

FIGURA 1. MEALY FINITE STATE MACHINE

Il comportamento di questa macchina a stati finiti può essere descritto mediante il seguente codice VHDL:

```
-- This is a behavioral model of a Mealy state machine
-- based on its state table. The output (z) and next state are
-- computed before the active edge of the clock.  The state change
-- occurs on the rising edge of the clock.

entity ex_1 is
   port( x, clk: in bit;
         z: out bit );
end ex_1;

architecture fsm of ex_1 is
   signal state, nextstate: integer := 0;
begin
   process( state, x ) --Combinational Network
   begin
     case state is
      when 0 =>
         if x='0' then z<='1'; nextstate<=1; end if;
         if x='1' then z<='0'; nextstate<=2; end if;
      when 1 =>
         if x='0' then z<='1'; nextstate<=3; end if;
         if x='1' then z<='0'; nextstate<=4; end if;
      when 2 =>
         if x='0' then z<='0'; nextstate<=4; end if;
         if x='1' then z<='1'; nextstate<=4; end if;
      when 3 =>
         if x='0' then z<='0'; nextstate<=5; end if;
         if x='1' then z<='1'; nextstate<=5; end if;
      when 4 =>
         if x='0' then z<='1'; nextstate<=5; end if;
         if x='1' then z<='0'; nextstate<=6; end if;
      when 5 =>
         if x='0' then z<='0'; nextstate<=0; end if;
         if x='1' then z<='1'; nextstate<=0; end if;
      when 6 =>
         if x='0' then z<='1'; nextstate<=0; end if;
      when others => null; -- should not occur
     end case;
  end process;

  process( clk ) -- state Register
    begin
     if clk='1' then -- rising edge of clock
       state <= nextstate;
     end if;
  end process;
end fsm;
```

Questo codice è contenuto nel file:
`/home/gdg/teaching/spa_lab/aa_2008_09/lesson_2/ex_1/ex_1.vhd`

Si simuli la descrizione a due processi `ex_1` di una macchina a stati finiti. Alcuni esempi di stimoli per i segnali sono contenuti nei file

`/home/gdg/teaching/spa_lab/aa_2008_09/lesson_2/stimuli_ex1_ex2`

In particolare si analizzino i seguenti casi:

1. `x` cambia contemporaneamente con il fronte in salita dell'orologio (file: `stimuli_1.do`);
2. `x` cambia dopo il fronte in salita dell'orologio (file: `stimuli_2.do`);
3. `x` cambia prima del fronte in salita dell'orologio (file: `stimuli_3.do`).

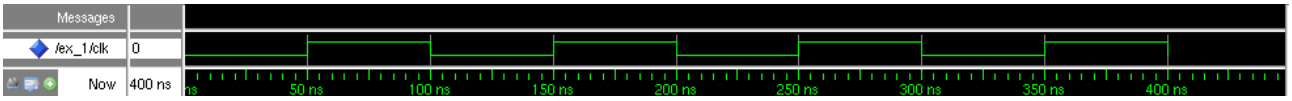

FIGURA 2: COMPORTAMENTO DEL SEGNALE DI CLOCK CON PERIODO 100NS

Esercizio 2

La macchina in Figura 1 può essere rappresentata anche mediante la seguente descrizione:

```
-- This is a behavioral model of a Mealy state machine based on
-- its state table.  The output (z) and next state are computed
-- on the rising edge of the clock OR when the input (x) changes.
-- The state change occurs on the rising edge of the clock.

entity ex_2 is
    port(x, clk: in bit;
         z: out bit);
end ex_2;

architecture fsm of ex_2 is
    signal state, nextstate: integer := 0;
begin
    process
    begin
      case state is
            when 0 =>
                if x='0' then z<='1'; nextstate<=1; end if;
                if x='1' then z<='0'; nextstate<=2; end if;
            when 1 =>
                if x='0' then z<='1'; nextstate<=3; end if;
                if x='1' then z<='0'; nextstate<=4; end if;
            when 2 =>
                if x='0' then z<='0'; nextstate<=4; end if;
                if x='1' then z<='1'; nextstate<=4; end if;
            when 3 =>
                if x='0' then z<='0'; nextstate<=5; end if;
                if x='1' then z<='1'; nextstate<=5; end if;
            when 4 =>
                if x='0' then z<='1'; nextstate<=5; end if;
                if x='1' then z<='0'; nextstate<=6; end if;
            when 5 =>
                if x='0' then z<='0'; nextstate<=0; end if;
                if x='1' then z<='1'; nextstate<=0; end if;
            when 6 =>
                if x='0' then z<='1'; nextstate<=0; end if;
            when others => null; -- should not occur
```

```
      end case;
      wait on clk, x;
      if (clk'event and clk='1') then
            state <= nextstate;
            wait for 0 ns; -- wait for state to be updated
      end if;
    end process;
end table;
```

Questo codice è contenuto nel file:
`/home/gdg/teaching/spa_lab/aa_2008_09/lesson_2/ex_2/ex_2.vhd`

Si confrontino le simulazioni della descrizione a due processi `ex_1` e quella ad un processo `ex_2` della medesima macchina a stati finiti. Si ottengono i medesimi risultati?

Come nel caso precedente, si analizzino i seguenti casi:

1. `x` cambia contemporaneamente con il fronte in salita dell'orologio;
2. `x` cambia dopo il fronte in salita dell'orologio;
3. `x` cambia prima del fronte in salita dell'orologio.

Nota 1:
Le simulazioni delle due descrizioni sono equivalenti. Infatti esse pur con stili di rappresentazione differenti modellano lo stesso comportamento per la macchina a stati finiti.

Ora si considerino le seguenti varianti del costrutto if in `ex_2`:

```
if (clk'event and clk='1') then
  state <= nextstate;
  wait for 0 ns;  -- wait for state to be updated
end if;
```

a. `-- ex_2_a.vhd`
```
if (clk'event and clk='1') then
  state <= nextstate after 5 ns;
  wait for 5 ns; -- wait for state to be updated
end if;
```

b. `-- ex_2_b.vhd`
```
if (clk'event and clk='1') then
  state <= nextstate;
  wait for 5 ns; -- wait for state to be updated
end if;
```

c. `-- ex_2_c.vhd`
```
if (clk'event and clk='1') then
  state <= nextstate after 5 ns;
  wait for 0 ns;       -- wait for state to be updated
end if;
```

d. `-- ex_2_d.vhd`
```
if (clk'event and clk='1') then
  state <= nextstate after 5 ns;
end if;
```

Per ognuna di esse si simuli la macchina ottenuta inserendo tale specifica versione del costrutto if nel codice di `ex_2`, si spieghino i risultati e le eventuali differenze tra i vari casi.
Ad esempio nel caso del costrutto a. come reagisce la macchina a cambiamenti in `x` che avvengono mentre è in attesa che trascorrano 5 ns a causa dell'istruzione
`wait for 5 ns; -- wait for state to be updated` ?

Nota 2:
Quando si parlerà di sintesi commentare il costrutto wait for 0 ns;

Nota 3:
E' possibile specificare come parametro del comando `vsim` non solo quale *architecture* simulare ma anche quale *entity* della descrizione VHDL.

Ad esempio assumendo di aver definito un nuovo file `ex_2_a.vhd` corrispondente al caso a. come segue:

```
entity ex_2 is
    port( x, clk: in bit;
          z: out bit );
end ex_2;

architecture fsm_a of ex_2 is
    signal state : integer := 0;
    signal nextstate: integer := 0;
begin
    process
    ...
    end process;
end fsm_a;
```

Il relativo comando per la compilazione (e la generazione del file WLF) è il seguente:

```
ModelSim > vsim –wlf waveform_examples/ex_2_a.wlf work.ex_2(fsm_a)
```

Sospensione dell'esecuzione (costrutto WAIT)

Un flusso di istruzioni sequenziali può essere sospeso in tre modi:

- specificando la *durata* della pausa;
- specificando una *condizione* che deve essere verificata per poter continuare;
- specificando una lista di segnali (*sensitivity list*) su cui si deve verificare un evento per poter continuare;

L'istruzione `wait` sospende l'esecuzione in un modo definito da una combinazione dei tre modi precedenti.

Un processo non deve avere la sensitivity list se contiene una wait.

Sintassi

```
wait [on sensitivity_list] [until condizione] [for tempo] ;
```

Assegnamenti di sequenziali e ritardo di propagazione (costrutto AFTER)

È possibile definire un ritardo nella propagazione della forma d'onda su un segnale, con gli stessi meccanismi degli assegnamenti concorrenti.

Sintassi

Assegnamento sequenziale di segnale:

```
nome_segnale <= espressione [after ritardo] ;
```

Esempio di simulazione e gestione della coda dei segnali (delta cycle)

Dato il seguente modulo VHDL (`example_1`) a due processi è possibile emulare il comportamento del simulatore mediante la rappresentazione della code dei segnali. La Figura 3 riporta una rappresentazione grafica di tali code nell'intervallo temporale *0 – 15 ns*.

```
entity example_1 is
end example_1;

architecture behav of example_1 is
   signal A,B: bit;
begin
P1: process(B)
begin
   A <= '1';
   A <= transport '0' after 5 ns;
end process P1;
P2: process(A)
begin
   if A = '1' then B <= not B after 10 ns; end if;
end process P2;
end behav;
```

La simulazione mediante ModelSim di questa descrizione rispetta il comportamento atteso, per verificarlo è possibile compilare il file:

```
/home/gdg/teaching/spa_lab/aa_2008_09/lesson_2/example_1/example_1.vhd
```

E quindi simulare il modulo `example_1`:

```
ModelSim > vsim work.example_1(behav)
VSIM > do example_1/stimuli_b.do
```

Terminata la simulazione sarà visibile un nuovo pannello: *List*. Nel pannello *List* viene riportato il comportamento dei segnali considerando gli istanti temporali di delta-cycle.

Nel caso del modulo appena simulato il contenuto di *List* è il seguente ed è esattamente equivalente a quanto ci si aspettava (si confronti con Figura 3). Si fa notare che è stata richiesta a

ModelSim una simulazione di durata *50 ns* e che i due processi coinvolti hanno un comportamento ciclico.

```
          ns        /example_1/a
           delta      /example_1/b
           0  +0                0 0
           0  +1                1 0
           5  +0                0 0
          10  +0                0 1
          10  +1                1 1
          15  +0                0 1
          20  +0                0 0
          20  +1                1 0
          25  +0                0 0
          30  +0                0 1
          30  +1                1 1
          35  +0                0 1
          40  +0                0 0
          40  +1                1 0
          45  +0                0 0
          50  +0                0 1
          50  +1                1 1
```

Nota:
I comandi `view list` e `add list *` permettono rispettivamente di visualizzare il pannello *List* e di osservare i comportamento di tutti i segnali.

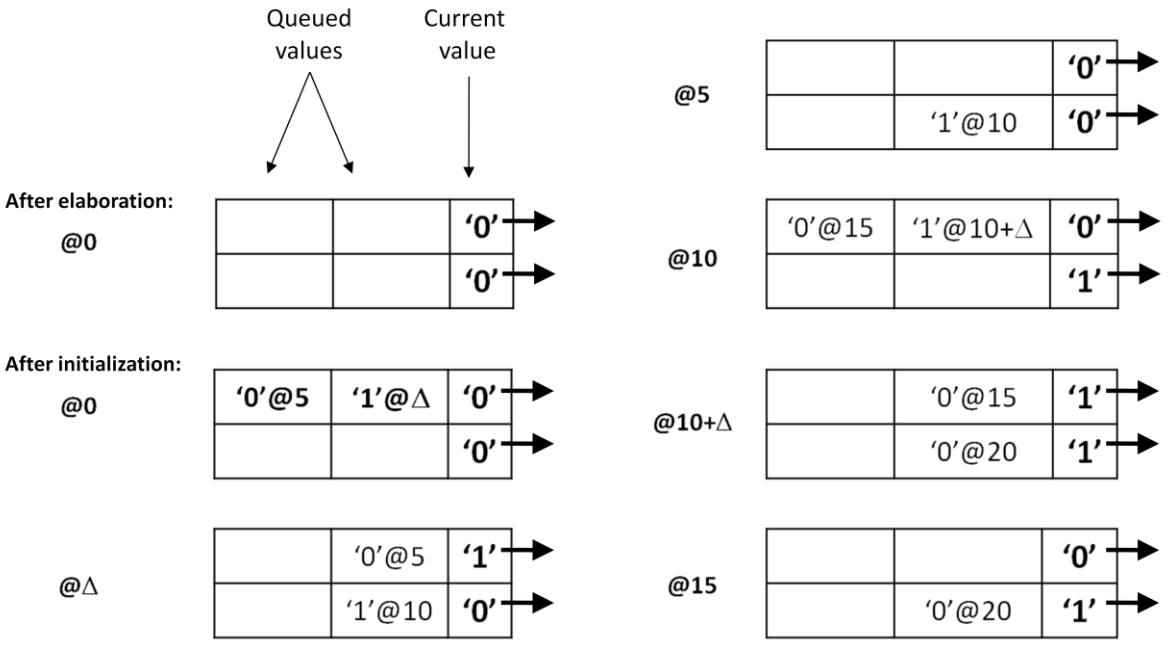

FIGURA 3: SIMULAZIONE DELLE CODE DEI SEGNALI SULL'INTERVALLO TEMPORALE 0 - 15NS

Esercizio 3

Nel seguente codice VHDL, si assuma che `D` cambi a *1* al tempo *t = 5 ns*. Si emuli il comportamento del simulatore indicando in un grafico (delle code dei segnali, "signal drivers") per ogni segnale i tempi in cui cambierà (es., *t = 5*, *5 + delta*, *5 + 2*delta* etc.) e i nuovi valori assunti. Si continui per almeno *20* unità di tempo, o fino a quando non ci siano cambiamenti di valore dei segnali o fino a quando emerga un comportamento ciclico.

```
entity ex_3 is
       port (D: inout bit);
end ex_3;

architecture behav of ex_3 is
       signal A : bit := '0';
       signal B : bit := '0';
       signal C : bit := '0';
       signal E : bit := '0';
       signal F : bit := '0';
begin
       C <= A;
       A <= B or D;

       P1: process (A)
       begin
              B <= A;
       end process P1;

       P2: process
       begin
              wait until A = '1';
              wait for 0 ns;
              E <= B;
              D <= '0';
              F <= E;
       end process P2;

end architecture behav;
```

Esercizio 4

Nel seguente processo VHDL i segnali `A`, `B`, `C` e `D` sono interi con valore *0* al tempo *t = 10 ns*. Se `E` cambia da *0* a *1* al tempo *t = 10 ns*, si emuli il comportamento del simulatore indicando in un grafico (delle code dei segnali, "signal drivers") per ogni segnale i tempi in cui cambierà (es., *t = 10*, *10 + delta*, *10 + 2*delta* etc.) e i nuovi valori assunti.

```
p1: process
begin
   wait on E;
   A <= 1 after 5 ns;
   B <= A +1;
   C <= B after 10 ns;
   wait for 0 ns;
   D <= B after 3 ns;
   A <= A + 5 after 15 ns;
   B <= B + 7;
```

```
end process p1;
```

Documentazione

La documentazione di Modelsim 6.4b è disponibile in formato PDF:

- `/opt/EDA_Software/mentor/modeltech/6.4b/modeltech/docs/pdfdocs`

Help in linea

Una guida testuale di ogni comando della console del simulatore può essere richiesta digitando `help` e il nome del comando

```
VSIM> help do
```

Sommario dei comandi di Modelsim

Questo è il sommario dei comadi base di Modelsim attinenti a questa lezione.

Comandi per la creazione della *design library* e per la compilazione.

`Vlib` - Crea una nuova *design library.*
`Vmap` - Mappa il nome della *design library* e il *path* alla stessa.
`vdir` - Lista il contenuto di una *design library*.
`vdel` - Rimuove un design da una *design library*.
`vmake` - Crea un Makefile per la design library specificata.
`vcom` - Compila il codice VHDL nella design library specificata.
`vsim` - Invoca il simulatore VSIM.
`verror` *i* - Stampa le informazioni riguardo messaggi di errore/warning, *i* = numero a 4-cifre

Comandi di simulazione.

`view` - Crea una finestra (wave, list, source etc).
`add wave` - Aggiunge un oggetto alla finestra Wave.
`force` - Forza uno stimolo per un segnale della descrizione VHDL.
`change` - Cambia il valore per una variabile, costante o generic VHDL.
`run` - Avvia la simulazione.
`restart` - Ricarica e riavvia la simulazione, l'opzione –f evita la richiesta di conferma.
`do` - Esegue un file di macro.

Comandi di debugging.

`examine` - Esamina il valore di una variabile o segnale.
`bp` - Aggiunge un breakpoint.
`step` - Esegue la prossima istruzione (dopo aver inserito un breakpoint).
`disablebp` - Disabilita un breakpoint.
`enablebp` - Abilita un breakpoint.
`checkpoint` - Salva uno stato del simulatore.
`restore` - Ripristino dello stato del simulatore.

Lezione 2: Soluzioni

Esercizio 2

Il comportamento di ciascuna delle quattro descrizioni, ottenute modificando `ex_2.vhd`, differisce dall'originale:

1. `ex_2_a.vhd`
   L'aggiornamento del segnale `state` è ritardato di *5 ns* rispetto al comportamento della descrizione originale. Inoltre la simulazione del processo ad ogni ciclo di clock è ferma per *5 ns* ignorando quindi eventuali aggiornamenti del segnale in ingresso `x` e ritardando anche l'aggiornamento del segnale `nextstate`.
2. `ex_2_b.vhd`
   La simulazione del processo ad ogni ciclo di clock è ferma per *5 ns* ignorando quindi eventuali aggiornamenti del segnale in ingresso `x` e ritardando anche l'aggiornamento del segnale `nextstate`.
3. `ex_2_c.vhd`
   L'aggiornamento del segnale `state` è ritardato di *5 ns* rispetto al comportamento della descrizione originale ma la simulazione prosegue in ogni caso, rimanendo nello stato precedente. La macchina a stati pertanto evolve in maniera scorretta: il valore `nextstate` è aggiornato a partire da uno stato non ancora aggiornato ("comportamento a singhiozzo").
4. `ex_2_d.vhd`
   Stesso comportamento del caso precedente.

Esercizio 3

Modelsim (Pannello *List*):

```
   ns       /ex_3/d   /ex_3/f
    delta     /ex_3/a
                /ex_3/b
                  /ex_3/c
                    /ex_3/e
    0  +0          0 0 0 0 0 0
    5  +0          1 0 0 0 0 0
    5  +1          1 1 0 0 0 0
    5  +2          1 1 1 1 0 0
    5  +3          0 1 1 1 1 0
```

Nota 1:

La porta `D` è di tipo `inout`. In questo caso è necessario forzare inizialmente il segnale D a 1 specificando l'opzione `-deposit`. Quest'opzione manterrà il valore passato come argomento al comando `force` fino a un eventuale nuovo cambiamento del segnale (anche dovuto ad un driver interno al modulo stesso). Se l'opzione `-deposit` non viene specificata il valore del segnale sarà costantemente quello forzato esternamente.

Viene di seguito riportato il file dei comandi (`stimuli.do`):

```
# Restart the simulation (and ignore confirmation message).
restart -f

# Open the Wave pane (undocked).
view wave
delete wave *
#-undock

# Select all primary inputs, primary output
# and internal signals and
# show them in the Wave pane.
add wave *

# E changes its value at t = 5 ns
force D 0 0
force -deposit D 1 5

# Run the simulation for 200 ns.
delete list *
run 200
add list *
```

Provare a sperimentare il comportamento della simulazione escludendo il flag `-deposit` dal file di comandi.

Code dei segnali:

Time						
					0	→A
					0	→B
@init					0	→C
					0	→D
					0	→E
					0	→F

Time						
					1	→A
					1	→B
@5+2Δ					1	→C
				0@5+3Δ	1	→D
				1@5+3Δ	0	→E
				0@5+3Δ	0	→F

Time						
				1@5+Δ	0	→A
					0	→B
@5					0	→C
					1	→D
					0	→E
					0	→F

Time						
				1@5+4Δ	1	→A
					1	→B
@5+3Δ					1	→C
					0	→D
					1	→E
					0	→F

Time						
					1	→A
				1@5+2Δ	0	→B
@5+Δ				1@5+2Δ	0	→C
					1	→D
					0	→E
					0	→F

Time						
					1	→A
					1	→B
@5+4Δ					1	→C
					0	→D
					1	→E
					0	→F

Nota 2:
Al tempo *(5ns+4Δ)* viene evidenziato uno stato della coda che ModelSim scarta. Al tempo *(5ns+3Δ)* si può osservare che il segnale A, avente valore *1*, verrà posto a *1* al passo successivo. Tale operazione risulta essere inutile e pertanto ignorata dal simulatore. Analogamente, anche l'evento *0@(5ns+3Δ)* sul segnale *F*, messo in coda al tempo *(5ns+2Δ)*, non viene eseguito.

Esercizio 4

L'esercizio forniva solamente la descrizione del processo, di seguito viene riportata la descrizione dell'intero modulo:

```
entity ex_4 is
  port( E: in bit );
end ex_4;

architecture behav of ex_4 is
  signal A : integer := 0;
  signal B : integer := 0;
  signal C : integer := 0;
  signal D : integer := 0;
begin
  p1: process
  begin
      wait on E;
      A <= 1 after 5 ns;
      B <= A + 1;
      C <= B after 10 ns;
      wait for 0 ns;
      D <= B after 3 ns;
      A <= A + 5 after 15 ns;
      B <= B + 7;
  end process;
end behav;
```

Modelsim (pannello *List*):

ns	delta	/ex_4/e	/ex_4/a	/ex_4/b	/ex_4/c	/ex_4/d
0	+0	0	0	0	0	0
10	+0	1	0	0	0	0
10	+1	1	0	1	0	0
10	+2	1	0	8	0	0
13	+0	1	0	8	0	1
25	+0	1	5	8	0	1

Code dei segnali:

Time				
			0	→A
			0	→B
@init			0	→C
			0	→D
			0	→E

Time				
	5@25	1@15	0	→A
			8	→B
@13		0@20	0	→C
			1	→D
			1	→E

Time				
		1@15	0	→A
		1@10+Δ	0	→B
@10		0@20	0	→C
			0	→D
			1	→E

Time				
		5@25	1	→A
			8	→B
@15		0@20	0	→C
			1	→D
			1	→E

Time				
	5@25	1@15	0	→A
		8@10+2Δ	1	→B
@10+Δ		0@20	0	→C
		1@13	0	→D
			1	→E

Time				
		5@25	1	→A
			8	→B
@20			0	→C
			1	→D
			1	→E

Time				
	5@25	1@15	0	→A
			8	→B
@10+2Δ		0@20	0	→C
		1@13	0	→D
			1	→E

Time				
			5	→A
			8	→B
@25			0	→C
			1	→D
			1	→E

Nota 1:
Al tempo *20ns* viene evidenziato uno stato della coda che ModelSim scarta. Al tempo *15ns* si può osservare che il segnale `C`, avente valore *0*, verrà posto a *0* al passo successivo. Tale operazione risulta essere inutile e pertanto ignorata dal simulatore.

Nota 2:
ModelSim sopprime l'evento *1@15ns* sul segnale *A*, messo in coda al tempo *10ns*: il listato del simulatore, infatti, non riporta nessun evento al tempo *(15ns)*. Questo comportamento dipende dal fatto che

1. al tempo *(10ns+Δ)* per il segnale *A* viene accodato l'evento 5@*25ns*, tale evento sovrascrive il valore del segnale
2. nessun altro segnale dipende da *A*

Un possibile esperimento consiste nel commentare nel processo *P1* l'istruzione

```
A <= A + 5 after 15 ns;
```

In questo caso il precedente punto 1. viene meno e il comportamento del simulatore è il seguente

```
  ns       /ex_4/e      /ex_4/a      /ex_4/b      /ex_4/c      /ex_4/d
    delta
  0   +0          0            0            0            0            0
 10   +0          1            0            0            0            0
 10   +1          1            0            1            0            0
 10   +2          1            0            8            0            0
 13   +0          1            0            8            0            1
 15   +0          1            1            8            0            1
```