Assert statement

All'interno di un process si può impiegare lo statement assert per individuare il verificarsi di situazioni di errore (errore funzionale) o situazioni critiche dal punto di vista del comportamento del sistema. Quando viene rilevata una di tali situazioni, si può predisporre un messaggio di errore che viene visualizzato in fase di simulazione. Alle situazioni di errore può essere assegnato un diverso livello di "criticità". In fase di simulazione si può istruire il simulatore perché la simulazione stessa venga interrotta al verificarsi di una condizione di errore con livello di criticità superiore o uguale a un livello dato.

La sintassi per lo statement assert è la seguente:

assertion_statement<= **assert** boolean_expression [**report** message_string] [**severity** severity_value]

Se l'espressione booleana boolean_expression è "false" allora il simulatore segnala questo fatto indicanto il tempo di simulazione corrente accompagnandolo con la stampa della stringa message_string (se la clausola **report** è presente) e col il livello di severity. Il livello di severity (severety_value) può assumere solo i valori note, warning, error e failure (corrispondenti ai quattro elementi del tipo predefinito per enumerazione severety_level).
Se non viene inclusa la clausola **severity**, il valore di default è error.
Il simulatore VHDL può essere istruito circa il modo di comportarsi in presenza di una "assertion violation". Infatti è possibile specificare se continuare l'esecuzione o se interromperala in corrispondenza di una violazione con critictà pari o superiore a un livello prestabilito.

Nell'esempio seguente, si usa lo statement assert per individuare e segnalare il presentarsi di una situazione di errore in una entità che implementa un flif flop SR. Nel caso in cui entrambi gli ingressi (s e r) sono a '1', si ha una assert violation che viene segnalata con il messaggio "Uso improprio del flip flop".

```
entity sr is
      port (s,r: in bit; q, qn: out bit);
end entity sr;


architecture datapath of sr is
signal ua:bit:='1';
signal ub:bit:='0';
begin
      ua<=r nor ub;
      ub<=s nor ua;
      q<=ua;
      qn<=ub;

controllo: process (s,r)
      begin
      assert ((s/='1')or(r/='1')) report "Uso improprio del flip flop" severity
error;
      end process controllo;
end architecture datapath;
```

In appendice è riportato il file completo di test_bench per verificare il comportamento dello statement assert.

Lo statement assert può essere usato come parte dei “passive processes” che possono essere inclusi nella dichiarazione di entità. In questo modo non è necessario includere il controllo di violazione in tutte le architetture che si riferiscono a una certa entità.

L'esempio precedente può essere riscritto come segue:

```
entity sr is
      port (s,r: in bit; q, qn: out bit);
      begin
contr: process (s,r)
      begin
      assert ((s/='1')or(r/='1')) report "Uso improprio del flip flop" severity
error;
      end process contr;

end entity sr;


architecture datapath of sr is
signal ua:bit:='1';
signal ub:bit:='0';
begin
      ua<=r nor ub;
      ub<=s nor ua;
      q<=ua;
      qn<=ub;
end architecture datapath;
```

Il file completo di test_bench è riportato in appendice.

L'istruzione assert è particolarmente utile per individuare la violazione del rispetto delle temporizzazioni in circuiti sequenziali sincronizzati. In VHDL esiste una funzione predefinta che restituisce il tempo di simulazione corrente (funzione predefinita now). In questo modo, grazie all'ausilio di variabili di appoggio di tipo time è possibile tener conto degli istanti di tempo in cui avvengono gli eventi e controllare la violazione di vincoli di temporizzazione.
Nell'esempio seguente definiamo un'entità che riproduce il comportamento di un flip flop D di tipo edge triggered. Nella definizione di entità è stato inserito tutto quanto necessario per verificare la presenza di violazione del rispetto dei tempi di setup e di hold del dato rispetto al fronte positivo del segnale di clock. Il codice completo di un opportuno test_bench per la simulazione è riportato in appendice. Dopo aver studiatio con attenzione le righe di codice seguente, si cerchi di dare risposta alla seguente domanda: se il dato cambia nello stesso istante in cui il clock va alto, che tipo di violazione viene segnalata? Si osservi infatti che nonostante che in una situazione del genere siamo certamente in presenza della violazione di entrambi i vincoli sui tempi, il codice è scritto in modo tale che viene rilevata una sola violazione. Si può modificare il codice in modo che in questa situazione vengano segnalate entrambe le violazioni?

```
entity ffd is
      port(reset,ck, d:in bit; uscita:out bit);

      constant T_setup:time:=5 ns;
      constant T_hold:time:=5 ns;
      begin
timing: process (reset,ck,d) is
      variable time_last_change,time_last_ck_rise:time:=0 ns;
      begin
      if (reset/='0') then
            if (ck'event and ck='1') then
                  time_last_ck_rise:=now;
```

```
                  assert ((time_last_ck_rise-time_last_change)>T_setup) report
"time setup violation" severity error;
            end if;
            if (d'event) then
                  time_last_change:=now;
                  assert ((time_last_change-time_last_ck_rise)>T_hold) report
"time hold violation" severity error;
            end if;
      end if;
end process timing;
end entity ffd;


architecture behav of ffd is

begin
flip_flop:process (reset,ck,d) is
      begin
      if (reset='0') then
            uscita<='0';
      else
            if (ck'event and ck='1') then
                  uscita <=d after 3 ns;
            end if;
      end if;
end process flip_flop;
end architecture behav;
```

Si può, nell'ipotesi semplificativa in cui si abbia che la durata del semi-periodo positivo del clock è sempre superiore al tempo di hold, riscrivere l'entità precedente facendo ricorso agli attributi dei segnali. In particolare ck'stable(T) produce true solo se il segnale ck è rimasto stabile nei T secondi precedenti al tempo di simulazione in cui viene valutato. Si noti inoltre che, se si scrive la dichiarazione di entità nel modo che segue, nel caso particolare in cui il dato cambia nello stesso istante in cui il clock va alto, si hanno entrambe le segnalazioni di violazione del tempo di setup e del tempo di hold.

```
entity ffd is
      port(reset,ck, d:in bit; uscita:out bit);

      constant T_setup:time:=5 ns;
      constant T_hold:time:=5 ns;
      begin
timing: process (reset,ck,d) is
      begin
      if (reset/='0') then
            if (ck'event and ck='1') then
                  assert d'stable(T_setup) report "time setup violation"
severity error;
            end if;
            if (d'event) then
                  assert (ck='0' or (ck='1' and ck'stable(T_hold))) report "time
hold violation" severity error;
            end if;
      end if;
end process timing;
end entity ffd;
```

L'argomento dell'assert che controlla il tempo di hold merita qualche attenzione. Quando il dato cambia, si possono distinguere 4 casi distinti:

1. Il dato cambia mentre il segnale di clock è a '1';

2. Il dato cambia mentre il segnale di clock è a '0';
3. il dato cambia nello stesso istante in cui il clock diventa '1';
4. il dato cambia nello stesso istante in cui il clock diventa 0;

Immaginiamo per semplicità che il clock sia periodico.
Se siamo nella condizione 2 o nella condizione 4, l'argomento dell'assert restituisce sempre true grazie al fatto che ck='0' sarebbe true. In ragione dell'ipotesi semplificativa assunta (semi-periodo positivo del clock di durata superiore al tempo di hold), in questo caso non si avrebbe certamente una violazione del tempo di hold. Se siamo nella condizione 1 o 3, allora ck='0' sarebbe false e si controlla effettivamente che quando il dato cambia, il clock sia rimasto alto per un tempo superiore al tempo di hold. In particolare, se siamo nel caso 3 si ha certamente una assert violation poiché ck'stable(T_hold) sarebbe false per qualunque valore positivo di T_hold.
Senza l'ipotesi semplificativa, nel caso in cui il dato cambi dopo (o in coincidenza con) il fronte di discesa del clock, non possiamo risalire, mediante il ricorso ai soli attributi dei segnali, al tempo trascorso dall'ultimo fronte di salita del clock.

File assert_sr.vhd

```
entity sr is
      port (s,r: in bit; q, qn: out bit);
end entity sr;

architecture datapath of sr is
signal ua:bit:='1';
signal ub:bit:='0';
begin
      ua<=r nor ub;
      ub<=s nor ua;
      q<=ua;
      qn<=ub;
controllo: process (s,r)
      begin
      assert ((s/='1')or(r/='1')) report "Errore: entrambi gli ingressi a 1!"
severity error;
      end process controllo;
end architecture datapath;

entity test_bench is
end test_bench;

architecture behav of test_bench is
      signal in1,in2:bit;
      signal u1:bit;
      signal u2:bit;
begin
      flip:entity work.sr(datapath)
            port map (in1,in2,u1,u2);
simula: process is
      begin
      in1<='0';
      in2<='0';
      wait for 10 ns;
      in1<='1';
      wait for 10 ns;
      in1<='0';
      wait for 10 ns;
      in2<='1';
      wait for 10 ns;
      in1<='1';
      wait for 10 ns;
      in2<='0';
      wait for 10 ns;
      in1<='0';
      wait for 20 ns;
      wait;
end process simula;
end behav;
```

Per eseguire la simulazione in ghdl (windows):

```
ghdl -a  aasert_sr.vhd
ghdl -e test_bench
ghdl -r test_bench -assert-level=error --vcd=out.vcd
winwave out.vcd
```

File assert_sr_pass.vhd

```
entity sr is
      port (s,r: in bit; q, qn: out bit);
      begin
contr: process (s,r)
      begin
      assert ((s/='1')or(r/='1')) report "Uso improprio del flip flop" severity
error;
      end process contr;

end entity sr;


architecture datapath of sr is
signal ua:bit:='1';
signal ub:bit:='0';
begin
      ua<=r nor ub;
      ub<=s nor ua;
      q<=ua;
      qn<=ub;
end architecture datapath;

entity test_bench is
end test_bench;

architecture behav of test_bench is
      signal in1,in2:bit;
      signal u1:bit;
      signal u2:bit;
begin
      flip:entity work.sr(datapath)
            port map (in1,in2,u1,u2);
simula: process is
      begin
      in1<='0';
      in2<='0';
      wait for 10 ns;
      in1<='1';
      wait for 10 ns;
      in1<='0';
      wait for 10 ns;
      in2<='1';
      wait for 10 ns;
      in1<='1';
      wait for 10 ns;
      in2<='0';
      wait for 10 ns;
      in1<='0';
      wait for 20 ns;
      wait;
end process simula;
end behav;
```

Per eseguire la simulazione in ghdl (windows):

```
ghdl -a  assert_sr_pass.vhd
ghdl -e test_bench
ghdl -r test_bench -assert-level=error --vcd=out.vcd
winwave out.vcd
```

File ffd_assert.vhd

```
entity ffd is
      port(reset,ck, d:in bit; uscita:out bit);

      constant T_setup:time:=5 ns;
      constant T_hold:time:=5 ns;
      begin
timing: process (reset,ck,d) is
      variable time_last_change,time_last_ck_rise:time:=0 ns;
      begin
      if (reset/='0') then
            if (ck'event and ck='1') then
                  time_last_ck_rise:=now;
                  assert ((time_last_ck_rise-time_last_change)>T_setup) report
"time setup violation" severity error;
            end if;
            if (d'event) then
                  time_last_change:=now;
                  assert ((time_last_change-time_last_ck_rise)>T_hold) report
"time hold violation" severity error;
            end if;
      end if;
end process timing;
end entity ffd;


architecture behav of ffd is

begin
flip_flop:process (reset,ck,d) is
      begin
      if (reset='0') then
            uscita<='0';
      else
            if (ck'event and ck='1') then
                  uscita <=d after 3 ns;
            end if;
      end if;
end process flip_flop;
end architecture behav;

entity test_bench is
end entity test_bench;

architecture behav of test_bench is

signal test_reset,test_ck,test_d:bit:='0';
signal test_out:bit;
begin

flip_ffd:entity work.ffd(behav)
      port map (test_reset,test_ck,test_d,test_out);
simula:process is
      begin
      test_reset<='0';
      wait for 20 ns;
      test_reset <='1';
      wait for 26 ns;
      test_d<='1';
      wait for 4 ns;
      test_ck<='1';
      wait for 5 ns;
```

```
        test_d<='0';
        wait for 5 ns;
        test_ck<='0';
        wait for 10 ns;
        test_ck<='1';
        wait for 2 ns;
        test_d<='1';
        wait for 10 ns;
        test_d<='0';
        test_ck<='0';
        wait for 20 ns;
        test_d<='1';
        test_ck<='1';
        wait for 20 ns;
        wait;

end process simula;
end architecture behav;
```

Per eseguire la simulazione in ghdl (windows):

```
ghdl -a  ffd_assert.vhd
ghdl -e test_bench
ghdl -r test_bench -assert-level=error --vcd=out.vcd
winwave out.vcd
```