VHDL Behavioral

Obbiettivi

- **Comprensione dei costrutti VHDL behavioral**
- **Espandere la conoscenza della sintassi del VHDL**
- **Fornire un esempio reale di VHDL**

Sommario

- Introduzione
- Modelli behavioral
  - Processi
  - Istruzioni sequenziali
  - Packages
  - Problemi
- Esempi
- Conclusioni

Introduzione al Behavioral Modeling in VHDL

- Livelli di astrazione nei modelli VHDL
  - Strutturale
  - Behavioral/strutturale mixed (i.e., data flow)
  - Behavioral
- Behavioral Modeling
  - Interessano quasi eclusivamente gli aspetti funzionali
  - Timing opzionale
  - Per sviluppare "buoni modelli", bisogna utilizzare tecniche ben note di software engineering
    - Progetto strutturato
    - Rifinimento iterattivo
    - Tipi di dato astratti

Esempio di modello Behavioral in VHDL

```
USE TEXTIO.all, mypackage.all;
ENTITY module IS
PORT (X, Y: IN BIT; Z: out BIT_VECTOR(3 DOWNTO 0);
END ENTITY module;
ARCHITECTURE behavior OF module IS
SIGNAL A, B: BIT_VECTOR(3 DOWNTO 0);
BEGIN
A(0) <= X AFTER 20 ns; A(1) <= Y AFTER 40 ns;
PROCESS (A)
VARIABLE P, Q: BIT_VECTOR(3 DOWNTO 0);
BEGIN
P := fft(A);
B <= P AFTER 10 ns;
END PROCESS;
Z <= B;
END behavior;
```


Processi VHDL

- **L'istruzione VHDL process e' usata per tutti i costrutti comportamentali**
- **Esempio:**

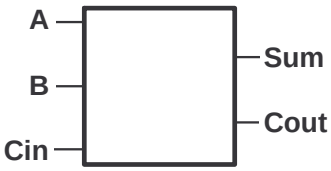
```
ARCHITECTURE behavioral OF clock_component IS
BEGIN
  PROCESS
    VARIABLE periodic: BIT := '1';
        BEGIN
          IF en = '1' THEN
            periodic := not periodic;
          END IF;
          ck <= periodic;
          WAIT FOR 1 us;
  END PROCESS;
END behavioral;
```

Sintassi


```
[ process_label : ] PROCESS
[( sensitivity_list )]

       process_declarations

BEGIN

       process_statements

END PROCESS [ process_label ] ;
```

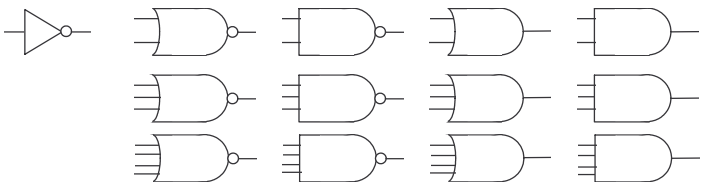
NON DICHIARARE SEGNALI

Esempio di modello VHDL




```
ENTITY full_adder IS
   PORT ( A, B, Cin : IN BIT;
          Sum, Cout : OUT BIT );
END ENTITY full_adder;
```

Librerie



Specifiche funzionali

A	B	Cin	Sum	Cout
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

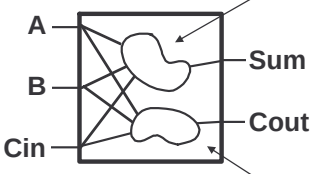
for Cout (I.e. Carry Out):

A B	Cin (I.e. Carry In) 0	1
0 0	0	0
0 1	0	1
1 1	1	1
1 0	0	1

for Sum:

A B	Cin (I.e. Carry In): 0	1
0 0	0	1
0 1	1	0
1 1	0	1
1 0	1	0

Esempio di descrizioni comportamentali del Full Adder




```
Summation:
PROCESS( A, B, Cin)
BEGIN
       Sum <= A XOR B XOR Cin;
END PROCESS Summation;
```

```
Carry:
PROCESS( A, B, Cin)
BEGIN
       Cout <= (A AND B) OR
               (A AND Cin) OR
               (B AND Cin);
END PROCESS Carry;
```

Architecture completa

VHDL Behavioral

```
ARCHITECTURE example OF full_adder IS
       -- Nothing needed in declarative block...
BEGIN

 Summation:  PROCESS( A, B, Cin)
       BEGIN
              Sum <= A XOR B XOR Cin;
 END PROCESS Summation;

 Carry:  PROCESS( A, B, Cin)
       BEGIN
              Cout <= (A AND B) OR
                     (A AND Cin) OR
                     (B AND Cin);
 END PROCESS Carry;

END example;
```

11

Alternativa per il carry

VHDL Behavioral

```
Carry:  PROCESS( A, B, Cin)
      BEGIN
             IF ( A = '1' AND B = '1' ) THEN
                    Cout <= '1';
             ELSIF ( A = '1' AND Cin = '1' ) THEN
                    Cout < = '1';
             ELSIF ( B = '1' AND Cin = '1' ) THEN
                    Cout <= '1';
             ELSE
                    Cout <= '0';
             END IF;
END PROCESS Carry;
```

12

Istruzioni VHDL Sequenziali

- All'interno dei processi el istruzioni sono eseguite sequenzialmente
  - Assegnamenti a {segnali, e variabili}
  - Controllo di flusso
    - IF <condition> THEN <statements> [ELSIF <statements>] [ELSE <statements>] END IF;
    - FOR <range> LOOP <statements> END LOOP;
    - WHILE <condition> LOOP <statements> END LOOP;
    - CASE <condition> IS WHEN <value> => <statements>
      {WHEN <value> => <statements>}
      [WHEN others => <statements>]
      END CASE;
  - WAIT [ON <signal>] [UNTIL <expression>] [FOR <time>] ;
  - ASSERT <condition> [REPORT <string>] [SEVERITY <level>] ;

Esempio 2-bit Counter


```
ENTITY count2 IS
       GENERIC(prop_delay : TIME := 10 ns);
       PORT (clock : IN BIT;
             q1, q0: OUT BIT);
END count2;

ARCHITECTURE behavior OF count2 IS
BEGIN
       count_up: PROCESS (clock)
        VARIABLE count_value: NATURAL := 0;
       BEGIN
         IF clock='1' THEN
           count_value := (count_value+1) MOD 4;
           q0 <= bit'val(count_value MOD 2) AFTER prop_delay;
           q1 <= bit'val(count_value/2) AFTER prop_delay;
         END IF;
       END PROCESS count_up;
END behavior;
```

L'istruzione Wait

- **L'istruzione *wait* provoca la sospensione di un processo o di una procedura**
- **wait [sensitivity_clause] [condition_clause] [timeout_clause] ;**
  - **sensitivity_clause ::= ON signal_name { , signal_name }**

```
WAIT ON clock;
```

  - **condition_clause ::= UNTIL boolean_expression**

```
WAIT UNTIL clock = '1';
```

  - **timeout_clause ::= FOR time_expression**

```
WAIT FOR 150 ns;
```


Processi equivalenti

- **"Sensitivity List" vs "wait on"**

```
Summation:
  PROCESS( A, B, Cin)
  BEGIN
        Sum <= A XOR B XOR
Cin;
END PROCESS Summation;
```

=

```
Summation:  PROCESS
  BEGIN
        Sum <= A XOR B XOR Ci
                  WAIT ON A,
END PROCESS Summation;
```

Se si mette una sensitivity list in un processo, non si puo' usare l'istruzione wait e viceversa

“wait until” e “wait for”

- **Tracciare le waveform in questi casi:**

```
Summation:  PROCESS
  BEGIN
         Sum <= A XOR B XOR Cin;
         WAIT UNTIL A = ‘1’;
END PROCESS Summation;
```

```
Summation:  PROCESS
  BEGIN
         Sum <= A XOR B XOR Cin;
       WAIT FOR 100 ns;
END PROCESS Summation;
```


Utilizzo di entrambi i costrutti

- **Si supponga di avere due segnali in un architettura**

```
DoSomething:  PROCESS
              BEGIN

      WAIT ON TheirSignal;

                    OurSignal <= ‘1’;
                    WAIT FOR 10 ns;

                    OurSignal <= ‘0’;
                    WAIT UNTIL (TheirSignal = ‘1’);

                    OurSignal <= ‘1’;

END PROCESS DoSomething;
```

Testbenches

- **Un testbench e' il componente top-level**
  - **La sua dichiarazione non contiene alcuna PORT**
  - **Instanzia tutti i componenti del sistema**
- **I testbenches servono anche per:**
  - **Generare stimoli per la simulazione:**
    - **Per questo si possono usare descrizioni behavioral**
  - **Generare stimoli per il collaudo**
    - **Segnali dichiarati localmente si possono connettere a PORT del sistema**
  - **Possono confrontare risposte di uscita con valori attesi**
    - **Verifica**

Testbenches (Cont.)

- **Esempio di testbench:**

```
ENTITY testbench IS
-- no PORT statement necessary
END testbench;

ARCHITECTURE example IS testbench
      COMPONENT entity_under_test
            PORT(...)
      END COMPONENT;
BEGIN
      Generate_waveforms_for_test;
      Instantiate_component;
      Monitoring_statements;
END example;
```

Si possono usare diversi costrutti

VHDL Behavioral

```
ARCHITECTURE example
       OF testbench IS
  .
       .
BEGIN
       MakeReset( ResetSignal,
                          100 ns );
       MakeClock( ClockSignal,
                          10 ns );
       .
END example;
```

```
PROCESS (ResetSignal)
BEGIN
       MakeReset( ResetSignal,
                          100
ns );
END PROCESS;
```

```
PROCESS
BEGIN
       MakeClock( ClockSignal,
                          10 ns );
       WAIT ON ClockSignal;
END PROCESS;
```

21

Ulteriore esempio

VHDL Behavioral

```
ARCHITECTURE example OF
              full_adder IS
BEGIN
       Summation:  PROCESS( A, B, Cin)
              BEGIN
                     Sum <= A XOR B XOR Cin;
       END PROCESS Summation;

       Carry:  PROCESS( A, B, Ci
              BEGIN
                     Cout <= (A
                                   (A A
                                   (B A
       END PROCESS Carry;
END example;
```

```
ARCHITECTURE example OF
              full_adder IS

BEGIN

       Sum <= A XOR B XOR Cin;

       Cout <= (A AND B) OR
                                   (A AND
                                   (B AND

END example;
```

22

Istruzioni di assegnamento dei segnali

```
ARCHITECTURE stuff OF my_entity IS
  SIGNAL ThisBit : BIT;
  SIGNAL ThisBitVector : BIT_VECTOR(1 TO 5);
  SIGNAL ThisInteger : INTEGER;
  SIGNAL ThisString : STRING(1 TO 4);
BEGIN
  ThisBit <= '1';
  ThisBitVector <= "10010";
  ThisInteger <= 567 AFTER 10 ns;
  ThisString <= "VHDL" AFTER 10 ns,
                " is " AFTER 20 ns,
                "fun!" AFTER 30 ns;
END stuff;
```


Ritardi inerziali e di trasporto

Trasporto

```
ENTITY nand2 IS
       PORT( A, B : IN BIT; C : OUT
BIT);
END nand2;

ARCHITECTURE behavior OF nand2 IS
BEGIN
  C <= TRANSPORT NOT(A AND B)
                              AFTER 25
ns;
END behavior;
```

Inerziale

```
ENTITY nand2 IS
       PORT( A, B : IN BIT; C : OUT
BIT);
END nand2;

ARCHITECTURE behavior OF nand2 IS
BEGIN
  C <= NOT(A AND B) AFTER 25 ns;
END behavior;
```

Sottoprogrammi

- **Simili a quelli tipici di altri linguaggi**
- **Permettono di utilizzare ripetutamente del codice senza bisogno di riscriverlo**
- **Divide grossi blocchi di software in parti piu' piccole e meglio maneggiabili**
- **Il VHDL mette a disposizione sia funzioni che procedure**

25

Funzioni

- **Producono un singolo valore di ritorno**
- **Chiamate all'interno di espressioni**
- **Non possono modificare i parametri che le sono passati**
- **RETURN statement**

```
FUNCTION add_bits (a, b : IN BIT) RETURN BIT IS
BEGIN  -- functions cannot return multiple values
       RETURN (a XOR b);
END add_bits;
```

```
FUNCTION add_bits2 (a, b : IN BIT) RETURN BIT IS
 VARIABLE result : BIT;  -- variable is local to function
      BEGIN
             result := (a XOR b);
             RETURN result;  -- the two functions are
equivalent
END add_bits2;
```

26

Funzioni


```
ARCHITECTURE behavior OF adder IS
BEGIN
       PROCESS (enable, x, y)
       BEGIN
       IF (enable = '1') THEN
              result <= add_bits(x,
y);
              carry  <= x AND y;
       ELSE
              carry, result <= '0';
       END PROCESS;
END behavior;
```

```
FUNCTION add_bits
(a, b : IN BIT)
```

- **Non possono essere chiamate autonomamente**
- **Associazione posizionale dei parametri**

Procedure

- **Valori di uscita multipli**
- **Chiamabili autonomamente**
- **Possono modificare i parametri**

```
PROCEDURE add_bits3 (SIGNAL a, b, en : IN BIT;
          SIGNAL temp_result, temp_carry : OUT BIT) IS

BEGIN -- procedures can return multiple values
             temp_result <= (a XOR b) AND en;
             temp_carry <= a AND b AND en;

END add_bits3;
```

- **Non richiedono un RETURN**

Procedure (Cont.)

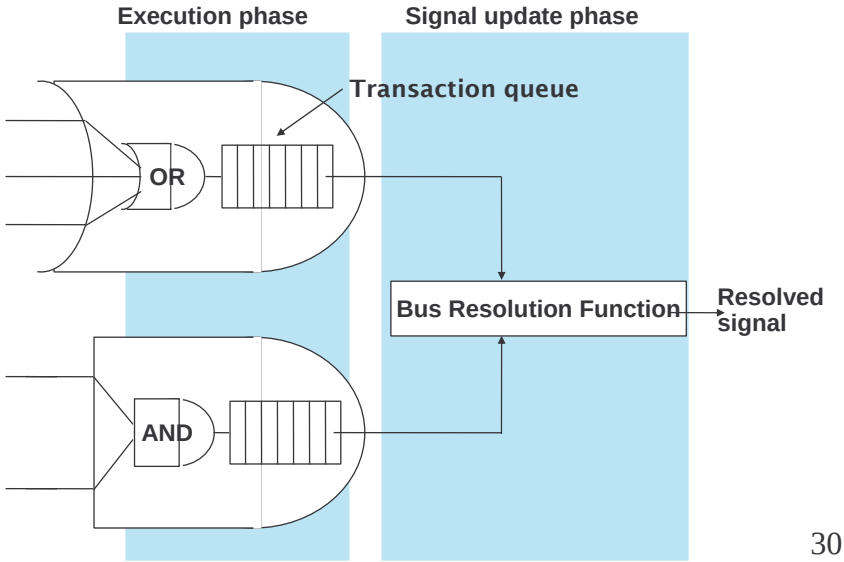
```
ARCHITECTURE behavior OF adder IS
BEGIN
       PROCESS (enable, x, y)
       BEGIN
       add_bits3(x, y, enable,

result, carry);
       END PROCESS;
END behavior;
```

- **Consentono di semplificare ulteriormente l'architettura**

- **Compatibilita' fra i parametri**

```
PROCEDURE add_bits3

(SIGNAL a, b, en : IN BIT;
 SIGNAL temp_result,
        temp_carry : OUT
BIT)
```


Risoluzione dei segnali e buses



Risoluzione dei Bus

- **Il VHDL non permette assegnamenti multipli concorrenti allo stesso segnale**
  - **Permette assegnamenti multipli sequenziali**

```
LIBRARY attlib; USE attlib.att_mvl.ALL;
-- this code will generate an error
ENTITY bus IS
       PORT (a, b, c : IN MVL; z : OUT MVL);
END bus;

ARCHITECTURE smoke_generator OF bus IS
       SIGNAL circuit_node : MVL;
BEGIN
       circuit_node <= a;
       circuit_node <= b;
       circuit_node <= c;
       z <= circuit_node;
END smoke_generator;
```

31

Funzioni di risoluzione dei bus

- **Utilizzate per calcolare i valori di un nodo connesso a piu' segnali**

```
FUNCTION wired_and (drivers : MVL_VECTOR) RETURN MVL IS
       VARIABLE accumulate : MVL := '1';
BEGIN
       FOR i IN drivers'RANGE LOOP
              accumulate := accumulate AND drivers(i);
       END LOOP;
       RETURN accumulate;
END wired_and;
```

- **Possono essere definite dall'utilizzatore o definite in un pacchetto**

32

Bus Resolution

- **Un segnale che ha una funzione di risoluzione associata puo' avere piu' drivers**

```
LIBRARY attlib; USE attlib.att_mvl.ALL;
USE WORK.bus_resolution.ALL;

ENTITY bus IS
       PORT (a, b, c : IN MVL; z : OUT MVL);
END bus;

ARCHITECTURE fixed OF bus IS
       SIGNAL circuit_node : wired_and MVL;
BEGIN
       circuit_node <= a;
       circuit_node <= b;
       circuit_node <= c;
       z <= circuit_node;
END fixed;
```


Null Transactions

- **Utilizzate per fare in modo che un segnale non influenzi il bus**
  - **null waveform element**
- **Esempio**

```
bus_out <= NULL AFTER 17 ns;
```

- **Cosa succede se tutti i drivers sono disconnessi (Z) ?**
  - **Utilizza il tipo di dato *register* nella dichiarazione dei segnali per mantenere l'ultimo valore**
  - **Il tipo *bus* si usa se invece e' la funzione di risoluzione a determinare l'ultimo valore**
- **Esempio**

```
signal t :  wired_bus BUS;
signal u :  BIT REGISTER;
```

Istruzioni nelle entity

- **Le entities possono contenere solo istruzioni di questo tipo :**
  - **Concurrent assertion statements**
  - ***Passive* concurrent procedure calls**
  - ***Passive* process statements**
- **Esempio :**

```
ENTITY multiplexor IS
PORT (a, b: IN BIT; select: IN BIT;
   output: OUT BIT);
BEGIN
check: PROCESS(a, b)
BEGIN
ASSERT NOT(a=b) REPORT “a equals b”
   SEVERITY NOTE;
END PROCESS;
 END multiplexor;
```


Blocks e Guards

- **I blocks sono istruzioni concorrenti e provvedono un meccanismo per partizionare la descrizione di una architettura**
  - **Gli items dichiarati nella regione dichiarativa di un block sono visibili solo dentro al block, e.g. :**
    - **segnali, sottoprogrammi**
- **I blocchi possono essere innestati per avere un partizionamento gerarchico della descrzione di architettura**

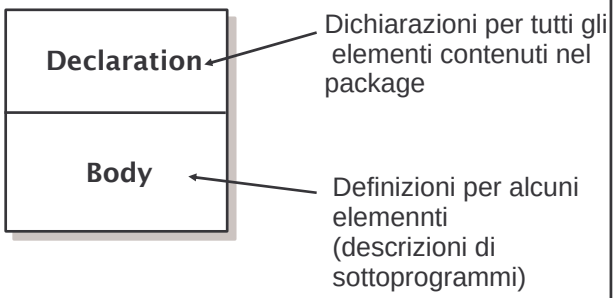
Blocks e Guards

- **Il costrutto GUARD compare solo nei blcoks**
  - **Un assegnamento a un segnale *guarded programma un assegnamento al driver solo se l'espressione* GUARD e' vera. Se GUARD e' falsa, i driver dei segnali sono disconnessi**
  - **Esempio**

```
ARCHITECTURE guarded_assignments OF n_1_mux IS
BEGIN
  bi: FOR j IN i'RANGE GENERATE
    bj: BLOCK (s(j)='1' OR s(j)='Z')
BEGIN
  x <= GUARDED i(j);
END BLOCK;
  END GENERATE;
END guarded_assignments
```


VHDL Packages

- **I packages incapsulano elementi che possono essere condivisi da piu' unita' di progetto**
- **Un package consiste di due parti**



Packages

- Esempio di contenuti:
  - Sottoprogrammi (i.e. funzioni e procedure)
  - Dichiarazioni di dati e tipi di dato
    - User record
    - User types e enumerated types
    - Costanti
    - File
    - Aliases
    - Attributi
  - Dichiarazioni di Component
- Entities e architectures non possono essere dichiarati o definiti in un
- Devono essere resi visibili tramite il costrutto use

File

- Il VHDL mette a disposizione il package `textio`
- Esempio di dichiarazione:

```
USE std.textio.all;
FILE data : text OPEN READ_MODE is “data.txt”;
```

- Funzioni associate ai file:

```
file_open();
file_close();
read(); write();
readline(); writeline();
endfile();
```

- Le operazioni di `read(); e write();` sono applicabili anche a stringhe

Esempio di operazioni sui file

- **Una lista di vettori logici si trova in un file nel seguente formato**

```
0100 10 ns
1010 40 ns
....
1111 200 ns
```

- **Si vuole realizzare un architettura che legga i vettori di test e li applichi in maniera ordinata ai segnali**

Esempio di operazioni sui file


```
LIBRARY ieee;
USE ieee.std_logic_1164.all,
   std.textio.all;

ENTITY prova IS
END ENTITY prova;

ARCHITECTURE leggi_file OF prova
   IS
FILE test_vectors: TEXT;
CONSTANT n: NATURAL := 4;
SIGNAL x: std_logic_vector((N-1)
   DOWNTO 0);

BEGIN
p: PROCESS IS
VARIABLE inline : LINE;
VARIABLE ch : CHARACTER;
VARIABLE del : time;
VARIABLE y: bit_vector((N-1)
   downto 0);
```

stringa

```
BEGIN
file_open(test_vectors,"test.vec",re
   ad_mode);
WHILE NOT endfile(test_vectors) LOOP
  readline(test_vectors,inline);
  FOR i IN N-1 DOWNTO 0 LOOP
    read(inline,y(i));
  END LOOP;
  read(inline,ch);
  read(inline,del);
  x <= to_stdlogicvector(y);
  WAIT FOR del;
END LOOP;
file_close(test_vectors);
WAIT;
END PROCESS p;
END ARCHITECTURE leggi_file;
```

utility di conversione

la read non legge il tipo std_logic

Esempi

- **Creare una funzione di risoluzione di un bus tri-state per una logica a quattro-valori**
- **Costruire la descrizione della macchina a stati dell'unita' di controllo di un moltiplicatore a 8 bit**
- **Realizzare la procedura Quicksort in VHDL sequenziale**

Package con funzione di risoluzione del bus (Package Declaration)

```
PACKAGE resources IS

 -- user defined enumerated type
   TYPE level IS ('X', '0', '1', 'Z');

 -- type for vectors (buses)
   TYPE level_vector IS ARRAY (NATURAL RANGE <>) OF level;

 -- subtype used for delays
   SUBTYPE delay IS time;

 -- resolution function for level
   FUNCTION wired_x (input : level_vector) RETURN level;

 -- subtype of resolved values
   SUBTYPE level_resolved_x IS wired_x level;

 -- type for vectors of resolved values
   TYPE level_resolved_x_vector IS
      ARRAY (NATURAL RANGE <>) OF level_resolved_x;

 END resources;
```

Package con funzione di risoluzione del bus (Package Body)

```
PACKAGE BODY resources IS
  -- resolution function
  FUNCTION wired_x (input : level_vector) RETURN level IS

    VARIABLE has_driver : BOOLEAN := FALSE;
    VARIABLE result     : level   := 'Z';
    BEGIN
      L1 : FOR i IN input'RANGE LOOP

        IF(input(i) /= 'Z') THEN
          IF(NOT has_driver) THEN
            has_driver := TRUE;
            result := input(i);
          ELSE                    -- has more than one driver
            result := 'X';
            EXIT L1;
          END IF;
        END IF;

      END LOOP L1;

      RETURN result;
  END wired_x;
END resources;
```


Bus Resolution Function Risultati della simulazione

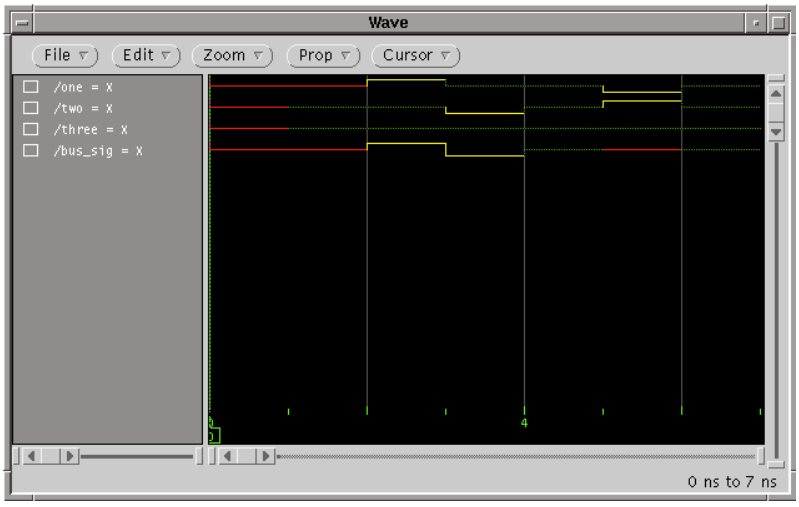


Diagramma di flusso per il controllore di un moltiplicatore a 8 Bit

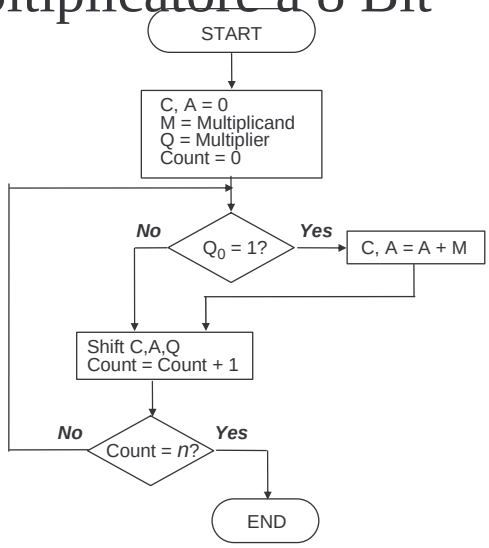
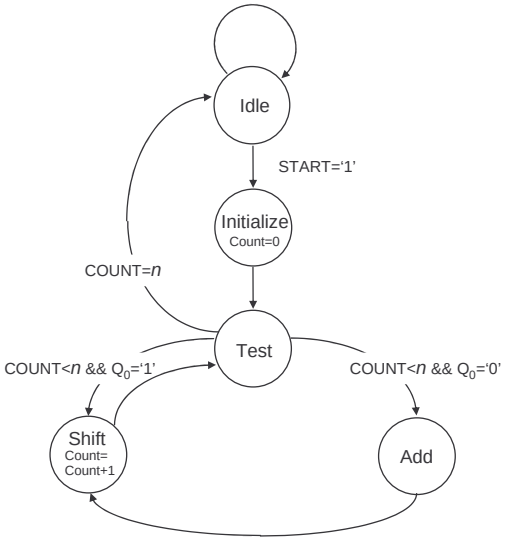


Diagramma degli stati



VHDL

Descrizione comportamentale

Behavioral

- **Descrizione delle macchine a stati sintetizzabile VHDL**
- **Due variabili di stato interne**
  - **present_state**
  - **present_count**
- **Tre processi VHDL che interagiscono**
  - **Clock process**
  - **State Transition process**
  - **Output process**
- **Macchina di Moore**
- **Reset asincrono e start sincrono**

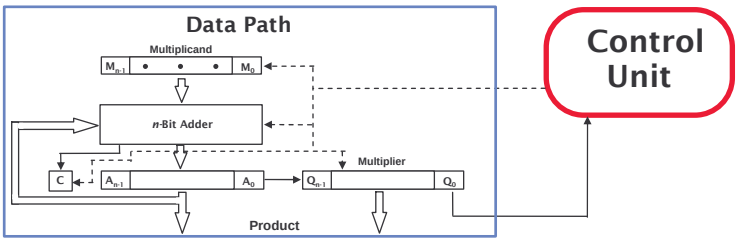
49

VHDL

Moltiplicatore a 8 bit control unit (Entity)

Behavioral

```
LIBRARY gate_lib;
USE gate_lib.resources.all;
ENTITY mult_controller_behav IS
  PORT(reset    : IN level;   -- global reset signal
       start    : IN level;   -- input to indicate start of process
       q0       : IN level;   -- q0 ,input from data path
       clk      : IN level;   -- clock signal
       a_enable : OUT level;  -- clock enable for A register
       a_reset  : OUT level;  -- Reset control for A register
       a_mode   : OUT level;  -- Shift or load mode for A
       c_enable : OUT level;  -- clock enable for c register
       m_enable : OUT level;  -- clock enable for M register
       q_enable : OUT level;  -- clock enable for Q register
       q_mode   : OUT level); -- Shift or load mode for Q
END mult_controller_behav;
```



50

VHDL

processo di controllo (clock)

Behavioral

(Architecture - Clock Process)

```
ARCHITECTURE state_machine OF mult_controller_behav IS

  SUBTYPE count_integer IS INTEGER RANGE 0 TO 8;
  TYPE states IS (idle,initialize,test,shift,add);
  SIGNAL present_state : states := idle;
  SIGNAL next_state    : states := idle;
  SIGNAL present_count : count_integer := 0;
  SIGNAL next_count    : count_integer := 0;

  BEGIN

    CLKD : PROCESS(clk,reset)
      BEGIN
        IF(reset = '1') THEN
          present_state <= idle;
          present_count <= 0;
        ELSIF(clk'EVENT AND clk = '1' AND clk'LAST_VALUE = '0') THEN
          present_state <= next_state;
          present_count <= next_count;
        END IF;
    END PROCESS CLKD;
```

51

VHDL

processo di controllo

Behavioral

(Architecture - State Transition Process)

```
STATE_TRANS : PROCESS(present_state,present_count,start,q0)
     BEGIN
       next_state <= present_state;       -- default case
       next_count <= present_count;       -- default case
       CASE present_state IS
         WHEN idle =>
           IF(start = '1') THEN
             next_state <= initialize;
           ELSE
             next_state <= idle;
           END IF;
           next_count <= present_count;
         WHEN initialize =>
           next_state <= test;
           next_count <= present_count;
         WHEN test =>
           IF(present_count < 8) THEN
             IF(q0 = '0') THEN
               next_state <= shift;
             ELSE
               next_state <= add;
             END IF;
           ELSE
             next_state <= idle;
           END IF;
           next_count <= present_count;
```

```
         WHEN add =>
           next_state <= shift;
           next_count <= present_count;
         WHEN shift =>
           next_state <= test;
           next_count <= present_count +
           1;
         WHEN OTHERS =>
           next_state <= idle;
           next_count <= present_count;
       END CASE;
END PROCESS STATE_TRANS;
```

52

Moltiplicatore a 8 bit uscite (Architecture - Output Process)

VHDL Behavioral

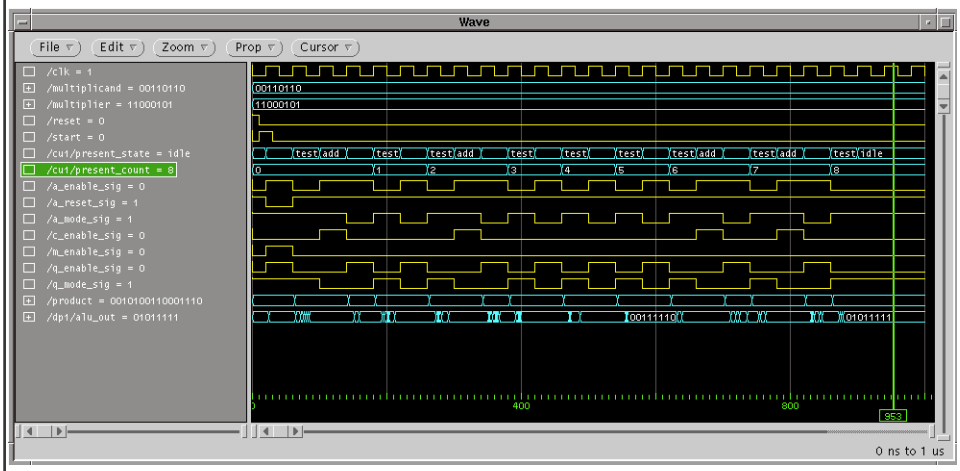
```
OUTPUT : PROCESS(present_state)
  BEGIN
    CASE present_state IS
      WHEN idle =>
        a_enable <= '0';
        a_reset  <= '1';
        a_mode   <= '1';
        c_enable <= '0';
        m_enable <= '0';
        q_enable <= '0';
        q_mode   <= '1';
      WHEN initialize =>
        a_enable <= '1';
        a_reset  <= '0';
        a_mode   <= '1';
        c_enable <= '0';
        m_enable <= '1';
        q_enable <= '1';
        q_mode   <= '1';
      WHEN test =>
        a_enable <= '0';
        a_reset  <= '1';
        a_mode   <= '1';
        c_enable <= '0';
        m_enable <= '0';
        q_enable <= '0';
        q_mode   <= '1';
```

```
WHEN add =>
            a_enable <= '1';
            a_reset  <= '1';
            a_mode   <= '1';
            c_enable <= '1';
            m_enable <= '0';
            q_enable <= '0';
            q_mode   <= '0';
          WHEN shift =>
            a_enable <= '1';
            a_reset  <= '1';
            a_mode   <= '0';
            c_enable <= '0';
            m_enable <= '0';
            q_enable <= '1';
            q_mode   <= '0';
          WHEN OTHERS =>
            a_enable <= '0';
            a_reset  <= '1';
            a_mode   <= '1';
            c_enable <= '0';
            m_enable <= '0';
            q_enable <= '0';
            q_mode   <= '1';
        END CASE;
    END PROCESS OUTPUT;

END state_machine;
```

53

Risultati della simulazione (Control Unit & Data Path)

VHDL Behavioral



54

Package per la procedura di Quicksort

VHDL Behavioral

```
PACKAGE qsort_resources IS
  CONSTANT maxarray : INTEGER := 100;
  TYPE integer_array IS ARRAY (NATURAL RANGE 0 to maxarray) OF integer;
  PROCEDURE quicksort(VARIABLE a : INOUT integer_array;
                              l : INTEGER;
                              r : INTEGER);
END qsort_resources;
```

55

Quicksort (Package Body - Quicksort Procedure)

VHDL Behavioral

```
PACKAGE BODY qsort_resources IS
  PROCEDURE quicksort(VARIABLE a : INOUT integer_array;
                              l : INTEGER; r : INTEGER) IS
    VARIABLE v, t : INTEGER;
    VARIABLE i, j : INTEGER;
    BEGIN
      IF(r > l) THEN
        v := a(r);
        i := l - 1;
        j := r;
        LOOP
          LOOP
            i := i + 1;
            EXIT WHEN(a(i) >= v);
          END LOOP;
          LOOP
            j := j - 1;
            EXIT WHEN(a(j) <= v);
          END LOOP;
          t := a(i);
          a(i) := a(j);
          a(j) := t;
          EXIT WHEN(j <= i);
        END LOOP;
        a(j) := a(i);
        a(i) := a(r);
        a(r) := t;
        quicksort(a, l, i - 1);
        quicksort(a, i + 1, r);
      END IF;
  END quicksort;
END qsort_resources;
```

56

Procedura di quicksort (Entity & Architecture)

VHDL Behavioral

```
LIBRARY STD;
USE STD.TEXTIO.all;
LIBRARY work;
USE work.qsort_resources.all;

ENTITY qsort IS
  GENERIC(infile  : STRING := "default";
          outfile : STRING := "default");
END qsort;

ARCHITECTURE test OF qsort IS

  BEGIN

    P1 : PROCESS

      VARIABLE nelements, i, tempint, temppointer :
   integer;
      VARIABLE iarray : integer_array;
      VARIABLE fresult : FILE_OPEN_STATUS := STATUS_ERROR;
      VARIABLE l : LINE;
      FILE in_fd  : TEXT;
      FILE out_fd : TEXT;
```

57

Procedura di quicksort (Architecture Cont.)

VHDL Behavioral

```
BEGIN

  file_open(fresult,in_fd,infile,READ_MODE);
  IF(fresult /= OPEN_OK) THEN
    ASSERT FALSE
      REPORT "Usage: qvsim qsort
                -ginfile=<infile>
                -goutfile=<outfile>"
    SEVERITY FAILURE;
  END IF;

  FILE_OPEN(fresult,out_fd,outfile,WRITE_MODE);
  IF(fresult /= OPEN_OK) THEN
    ASSERT FALSE
      REPORT "Usage: qvsim qsort
                -ginfile=<infile>
                -goutfile=<outfile>"
    SEVERITY FAILURE;
  END IF;

-- read in file and set number of elements
nelements := 0;
WHILE(NOT ENDFILE(in_fd)) LOOP
  READLINE(in_fd,l);
  READ(l,iarray(nelements));
  nelements := nelements + 1;
END LOOP;
```

```
    -- find minimum element and place in
    -- element zero for sentinel
    tempint := iarray(0);
    temppointer := 0;
    FOR i IN 1 TO nelements - 1 LOOP
      IF(iarray(i) < tempint) THEN
        tempint := iarray(i);
        temppointer := i;
      END IF;
    END LOOP;
    IF(temppointer /= 0) THEN
      iarray(temppointer) := iarray(0);
      iarray(0) := tempint;
    END IF;

    -- do the quicksort!
    quicksort(iarray,0,nelements-1);

    -- write out results
    FOR i IN 0 TO nelements - 1 LOOP
      WRITE(l,iarray(i));
      WRITELINE(out_fd,l);
    END LOOP;

    WAIT;
  END PROCESS P1;

END test;
```

58

Sommario

- **Il VHDL behavioral si utilizza per focalizzare sul compartamento e non sulla struttura del dispositivo**
- **Costrutti tipici dei linguaggi di programmazione, e.g CASE, IF-THEN-ELSE**
- **I sottoprogrammi permettono l'uso della programmazione strutturata**
- **La funzione di risoluzione del bus consente di descrivere componenti connessi mediante bus**

Bibliografia

[Ashenden], Peter Ashenden, "The VHDL Cookbook," Available via ftp from thor.ece.uc.edu.

[IEEE93], "The VHDL Language Reference Manual," *IEEE Standard* 1076-93, 1993.

[Jain91], Ravi Jain, The Art of Computer Systems Performance Analysis, John Wiley & Sons, 1991.

[Navabi93], Zain Navabi, VHDL: Analysis and Modeling of Digital Systems McGraw Hill, 1993.

[Mohanty95], Sidhatha Mohanty, V. Krishnaswamy, P. Wilsey, "Systems Modeling, Performance Analysis, and Evolutionary Prototyping with Hardware Description Languages," *Proceedings of the 1995 Multiconference on Simulation*, pp 312-318.

61

62

VHDL Behavioral

63