Descrizioni VHDL Dataflow

- In questo capitolo vedremo come la struttura di un sistema digitale è descritto in VHDL utilizzando una descrizione di tipo *data-flow*.
- **Outline:**
  - espressioni ed operatori
  - assegnazione concorrente
  - assegnazione condizionata
  - assegnazione concorrente
  - esempi vari

Convenzioni Tipografiche

- In tutte le slide si useranno le seguenti convenzioni tipografiche:
- **`ENTITY`**, **`ARCHITECTURE`**, **`FOR`**, ... sono comandi del linguaggio VHDL e sono parole riservate, ovvero non si possono usare come nomi assegnati dall'utente ad oggetti.
- **`Nome_oggetto`** è un oggetto del VHDL a cui l'utente ha assegna il nome nome_oggetto.
- *Entity*, *architecture*, etc. fa riferimento al concetto di entity in VHDL e non al comando entity che permette di instanziare una enity.
- Il VHDL non è case sensitive.
- Per introdurre dei commenti si usano due segni '-':

```
-- commento
```

Espressioni e operatori (1)

- Ad un **segnale,** ad una **variabile** o ad una **costante** può essere associato il risultato di una espressione che coinvolge diversi altri valori (non si possono mischiare segnali e variabili!)
- Le **espressioni** in VHDL sono del tutto simili alle espressioni in altri linguaggi di programmazione:
  - Una espressione combina degli operandi (valori di segnali, costanti, variabili), tramite degli operatori;
  - Di seguito sono riportati gli operatori predefiniti dal VHDL, in ordine di priorità descrescente (le parentesi permettono di alterare l'ordine di valutazione):

Highest precedence:	**	**abs**	**not**			
	*	/	**mod**	**rem**		
	+ (sign)	– (sign)				
	+	–	&			
	=	/=	<	<=	>	>=
Lowest precedence:	**and**	**or**	**nand**	**nor**	**xor**	

Espressioni e operatori (2)

- Gli **operatori relazionali** sono:

  =, /= (diverso), <, <=, > e >=

  e devono essere applicati ad elementi dello stesso tipo, producendo un boolean;
- Gli **operatori aritmetici**:

  ** (elevamento a potenza), abs, *, /, mod, rem (resto), +, -

  possono essere applicati solo ai tipi numerici (integer, floating point ed i tipi physical), dando un valore dello stesso tipo (oppure overflow e underflow) secondo il normale significato;
- Gli **operatori logici** (and, or, nand, nor, xor e not) operano su oggetti di tipo bit, std_logic e boolean, ma anche sugli oggetti di tipo bit_vector e std_logic_vector (word gate).
- L'**operatore &** permette di concatenare diversi oggetti ottenendo array di oggetti;

- In generale è possibile per l'utente anche definire nuovi operatori e fare overloading di altri operatori;

Assegnazione concorrente (1)

- Le assegnazioni di segnale **concorrenti** appaiono all'interno del corpo di una architecture, mentre le assegnazioni nel corpo di un process sono chiamate assegnazioni **sequenziali**;
- Le assegnazioni concorrenti sono attivate ogni volta che c'è una variazione in uno dei segnali da cui dipende il segnale assegnato;

```
S <= NOT ((A XOR B) AND C);
```

- In VHDL
  - tutte le assegnazioni di segnali (di qualsiasi tipo: incondizionate, condizionate, selezionate),
  - tutte le instanziazioni di blocchi,
  - tutti i process,

  presenti all'interno del corpo di una *architecture*, sono concorrenti fra loro;

Assegnazione concorrente (2)

- Il **segnale target** di una assegnazione di qualsiasi tipo può essere un segnale semplice, un segnale indicizzato, e.g. a(5), una slice a (5 downto 0), o un segnale aggregate a(5) & a(2);
- Il simbolo di assegnazione fra segnali è <=

```
S <= A XOR B;
S(0) <= A(0) XOR B(0);
S(3 downto 0) <= A(3 downto 0) XOR B(3 downto 0);
S(1) & S(0) <= A(1 downto 0) XOR B(1 downto 0);
```

- In VHDL ci sono **3 tipi di assegnazioni** concorrenti fra segnali:
  1. assegnazione incondizionata – la waveform che viene assegnata al segnale è fissa;
  2. assegnazione condizionata – la waveform assegnata è determinata sulla base di un costrutto *if-then-else*;
  3. assegnazione selezionata – la waveform assegnata è determinata sulla base di un costrutto di tipo *case*;

Assegnazione incondizionata

- Un'espressione che coinvolge diverse forme d'onda è semplicemente assegnata ad un segnale:

```
ARCHITECTURE dataflow OF HalfAdder IS
BEGIN
  Sum <= A XOR B;
  Carry <= A AND B;
END ARCHITECTURE dataflow;
```

Assegnazione condizionata (1)

- Una **assegnazione condizionata** è del tipo:

```
segnale <=      waveform_1 WHEN condizione_1 ELSE
                waveform_2 WHEN condizione_2 ELSE

                waveform_(n-1) WHEN condizione_(n-2) ELSE
                waveform_n;
```

- Una assegnazione condizionata permette di scegliere la waveform da assegnare (fra *waveform_1* … *waveform_n*) ad un segnale target (*segnale*), a seconda di quale condizione booleana (*condizione_1* … *condizione_n*) sia verificata;

- **Nota bene**:
  - In generale le condizioni *possono* sovrapporsi ed in questo caso l'ordine di valutazione delle condizioni (dal primo fino all'else finale) è decisivo;
  - Una assegnazione condizionata deve terminare con una ELSE senza condizione;

Assegnazione condizionata (2)

- Un esempio tipico è quello di un **buffer tristate**, con segnale di selezione *Enable*:

```
ARCHITECTURE dataflow OF TriStateBuffer IS
BEGIN
  BufOut <=       BufIn WHEN Enable = '1
                  ELSE 'Z';
END ARCHITECTURE dataflow;
```

  - Quando *Enable* è alto, al segnale *BufOut* viene assegnato *BufIn*, altrimenti viene assegnato il valore tristate ('Z');
- Un altro esempio è quella di un **multiplexer 2 a 1** su un bit, con abilitazione tristate:

```
  mux_out <=      'Z'  AFTER Tpd WHEN en = '0' ELSE
                  in_0 AFTER Tpd WHEN sel = '0' ELSE
                  in_1 AFTER Tpd;
```

  - In pratica è un mux il cui ingresso di selezione è *sel*, la cui uscita è l'ingresso primario di una tristate controllata dal segnale *en*;
  - In questo esempio sono presenti anche dei ritardi di propagazione;

Assegnazione condizionata (3)

- In pratica una assegnazione condizionata è un abbreviazione per un processo che contiene assegnazioni di segnali, sottoposti a delle condizioni di tipo IF...THEN;
- Ad esempio l'assegnazione condizionata:

```
segnale <=      waveform_1 WHEN condition_1 ELSE
                waveform_2 WHEN condition_2 ELSE

                waveform_n;
```

è equivalente al process riportato:

```
PROCESS
        IF condition_1 THEN
                segnale <= waveform_1;
        ELSIF condition_2 THEN
                segnale <= waveform_2;
        ELSIF
        ELSE
                segnale <= waveform_n;
        WAIT [sensitivity_list];
END PROCESS;
```

dove [sensitivity list] è l'insieme dei segnali da cui dipendono le clausole.

Assegnazione condizionata (4)

- Il caso degenere di una assegnazione condizionata è una assegnazione che non contenga alcuna parte condizionale ed è quindi equivalente ad un processo con una sola assegnazione, ovvero:

```
segnale <= waveform;
```

- è equivalente a:

```
PROCESS
        segnale <= waveform;
        WAIT [ sensitivity_clause ];
END PROCESS;
```

ovvero a:

```
PROCESS (sensitivity_clause)
        segnale <= waveform;
END PROCESS;
```

Assegnazione selezionata (1)

- Una **assegnazione selezionata** è del tipo:

```
WITH segnale_di_selezione SELECT
        segnale <=     waveform_1 WHEN caso_1,
                       waveform_2 WHEN caso_2,

                       waveform_n WHEN caso_n;
```

- Una assegnazione selezionata permette di scegliere la waveform da assegnare (fra *waveform_1* … *waveform_n*) ad un segnale target (*segnale*), a seconda del valore (caso_1 … caso_n) assunto da un segnale (*segnale_di_selezione*);

- **Nota bene**:
  - I casi *devono essere esaustivi* (a meno che non si usi un caso OTHERS);
  - I casi non si possono sovrapporre, ovvero devono essere mutuamente esclusivi;

Assegnazione selezionata (2)

- Un blocco che esegue diverse operazioni logiche a seconda del segnale di controllo è riportata in seguito (**Universal Gate**):

```
ARCHITECTURE dataflow OF UniversalGate IS
BEGIN
  WITH Command SELECT
    DataOut <= InA AND InB          WHEN "000",
               InA OR InB           WHEN "001",
               InA NAND InB         WHEN "010",
               InA NOR InB          WHEN "011",
               InA XOR InB          WHEN "100",
               InA XNOR InB         WHEN "101",
               'Z' WHEN OTHERS;
END ARCHITECTURE dataflow;
```

- Si possono anche usare range o valori alternativi:

```
WITH IntCommand SELECT
  MuxOut <= InA WHEN 0 | 1,
            InB WHEN 2 to 5,
            InC WHEN 6,
            InD WHEN 7,
            'Z' WHEN OTHERS;
```

Assegnazione selezionata (3)

- In pratica una assegnazione condizionata è un abbreviazione per un processo che contiene assegnazioni di segnali in un costrutto CASE;
- Il processo equivalente a

```
WITH segnale_di_selezione SELECT
        segnale <=      waveform_1 WHEN caso_1,
                        waveform_2 WHEN caso_2,

                        waveform_n WHEN caso_n;
```

è :

```
PROCESS
    CASE segnale_di_selezione IS
        WHEN caso_1 => s <= waveform_1;
        WHEN caso_2=> s <= waveform_2;

        WHEN caso_n=> s <= waveform_n;
    END CASE;
    WAIT [ sensitivity_list ];
END PROCESS;
```

- La sensitivity_list del wait contiene tutti i segnali riferiti nelle forme d'onda (waveform_1 … waveform_n) e nell'espressione del segnale di selezione.

Assegnazione selezionata (4)

- Un esempio di assegnazione selezionata si può trovare in una ALU:

```
WITH alu_function SELECT
        alu_result <=  op1 + op2 WHEN alu_add | alu_incr,
                       op1   op2 WHEN alu_subtract,
                       op1 AND op2 WHEN alu_and,
                       op1 OR op2 WHEN alu_or,
                       op1 AND NOT op2 WHEN alu_mask;
```

  - Il valore del segnale *alu_function* è usato per scegliere quale assegnazione associare al segnale *alu_result*;
  - La sensitivity list del processo equivalente a questa assegnazione è composta dai segnali *alu_function*, *op1* e *op2*: un evento su questi segnali provoca la valutazione dell'espressione;

Confronto fra assegnazioni (1)

- **Assegnazione condizionata**:

```
segnale <=      waveform_1 WHEN condizione_1 ELSE
                waveform_2 WHEN condizione_2 ELSE

                waveform_(n-1) WHEN condizione_(n-2) ELSE
                waveform_n;
```

  - i casi *non sono mutuamente esclusivi*;
  - vengono valutate sequenzialmente le condizioni e la prima che è vera determina l'assegnazione;

- **Assegnazione selezionata**:

```
WITH segnale_di_selezione SELECT
        segnale <=      waveform_1 WHEN caso_1,
                        waveform_2 WHEN caso_2,

                        waveform_n WHEN caso_n;
```

  - i casi *sono mutuamente esclusivi*;
  - non c'è un ordine di scansione dei confronti (i confronti sono in "parallelo");

- Se tutti i casi di una assegnazione condizionata sono mutuamente esclusivi, allora questo costrutto è equivalente ad una assegnazione selezionata.

Confronto fra assegnazioni (2)

- Qual è la differenza fra i 2 multiplexer riportati?

```
ENTITY mux_3to1 IS PORT (
          i0, i1, i2: IN BIT;
          s0, s1, s2 : IN BIT;
          y : OUT BIT );
END mux_3to1;

ARCHITECTURE dataflow1 OF mux_4to1 IS
   SIGNAL sel : BIT_VECTOR(2 DOWNTO 0);
BEGIN
   sel <= s2 & s1 & s0;

   WITH sel SELECT
          y <=      i0 WHEN  001 ,
                    i1 WHEN  010 ,
                    i2 WHEN OTHERS;
END dataflow1;
```

```
ENTITY mux_3to1 IS PORT (
          i0, i1, i2: IN BIT;
          s0, s1, s2 : IN BIT;
          y : OUT BIT );
END mux_3to1;

ARCHITECTURE dataflow2 OF mux_4to1 IS
BEGIN
   y <=   i0 WHEN s0 =  1  ELSE
          i1 WHEN s1 =  1  ELSE
          i2;
END dataflow2;
```

- Se s0 = 1, s1 = 1, s2 = 1 cosa succede nei due casi?

D latch con assegnazione condizionata

- Un **D latch** 1-attivo è un blocco con memoria che
  - memorizza il dato quando il livello del "clock" è alto (basso nel caso 0-attivo) → trasparente;
  - propaga in uscita l'ingresso quando il livello del clock è basso (alto nel caso 0-attivo) → in hold;
- Una descrizione di un flip-flop trasparente alto, con una **assegnazione condizionata**:

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;

ENTITY d_latch IS PORT (
   d, clk : IN std_logic;
   q : OUT std_logic );
END d_latch;

ARCHITECTURE dataflow of d_latch IS
   SIGNAL q_tmp : std_logic;
BEGIN
   q <= q_tmp;

   q_tmp <= d WHEN clk =  1  ELSE q_tmp;
END dataflow;
```

D latch con assegnazione condizionata

- Una descrizione di un D latch trasparente alto, con una **assegnazione selezionata** è la seguente:

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;

ENTITY d_latch IS PORT (
   d, clk : IN std_logic;
   q : OUT std_logic );
END d_latch;

ARCHITECTURE dataflow of d_latch IS
   SIGNAL q_tmp : std_logic;
BEGIN
   q <= q_tmp;

   WITH clk SELECT
       q_tmp <=  d WHEN  1 ;
                 q_tmp WHEN OTHERS;
END dataflow;
```

Flip-flop D con assegnazione condizionata

- Un **flip-flop D attivo sul fronte di salita** (risp. di discesa) è un blocco con memoria che:
  - memorizza ("campiona") il dato immagazzinandolo nel suo "stato" quando il livello del "clock" commuta da 0 → 1 (risp. 1 → 0);
  - mostra in uscita il suo stato;
- Una descrizione con una **assegnazione condizionata**:

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;

ENTITY d_ff IS PORT (
   d, clk : IN std_logic;
   q : OUT std_logic );
END d_ff;

ARCHITECTURE dataflow of d_ff IS
   SIGNAL q_tmp : std_logic;
BEGIN
   q <= q_tmp;

   q_tmp <= d WHEN (clk =  1  and clk event) ELSE q_tmp;
END dataflow;
```

Flip-flop D con assegnazione selezionata

- Una descrizione di un flip-flop D attivo sul fronte di salita, con una **assegnazione selezionata**:

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;

ENTITY d_ff IS PORT (
   d, clk : IN std_logic;
   q : OUT std_logic );
END d_ff;

ARCHITECTURE dataflow of d_ff IS
   SIGNAL q_tmp : std_logic;
BEGIN
   q <= q_tmp;

   WITH (clk =  1  and clk event)
         q_tmp <=  d WHEN true
                   q_tmp WHEN false;
END dataflow;
```

Flip-flop D con reset

- Un flip-flop D attivo sui fronti di discesa, con reset asincrono 1-attivo potrebbe essere descritto come:

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;

ENTITY d_ff IS PORT (
   d, clk : IN std_logic;
   q : OUT std_logic );
END d_ff;

ARCHITECTURE dataflow of d_ff IS
   SIGNAL q_tmp : std_logic;
BEGIN
   q <= q_tmp;

   q_tmp <=  0  WHEN rst =  1  ELSE
             d WHEN (clk =  0  and clk event) ELSE
             q_tmp;
END dataflow;
```