ESSENTIAL VERILOG COMMANDS

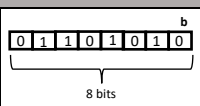
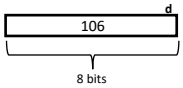
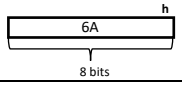
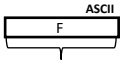
Corso di Architettura dei Calcolatori – Università di Siena

Roberto Giorgi, Marco Procaccini

COMMENTI

//commento su singola linea
/* commento multi-linea */

COSTANTI NUMERICHE

8'b01101010 //8-bit binario	
8'd106 //8-bit decimale	
8'h6A //8-bit esadecimale	
"F" //ASCII – ogni carattere asci occupa 8 bit	

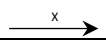
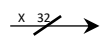
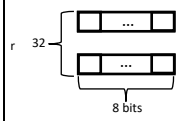
RAGGRUPPAMENTO DI COMANDI

begin //inizio gruppo comandi **<comando>;** **<comando>;** .... **end** //fine gruppo comandi	definisce un **<gruppo_comandi>**

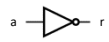
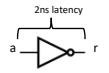
NOTE

- Il time diagram comprende le variabili del primo modulo nell'ordine in cui compaiono

TIPI DI DATO

wire x	un filo	
wire[31:0] x	filo multi-bit (bus)	
reg r	registro a 1 bit	
reg[31:0] r	registro a 32 bit	
reg[7:0] r[0:31]	vettore di 32 registri a 8 bit	

RETI COMBINATORIE

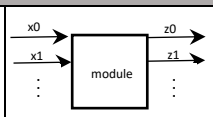
assign r =~a	assegnamento **continuo** di una espressione logica a una variabile r (**che deve essere un registro**)	
assign #2 r=~a	assegnamento **continuo** con ritardo di due unità di tempo (es. 2ns).	

DIRETTIVE E PROCESSI

`timescale <unita_tempo> / <precisione>	definisce l'unità di tempo della simulazione (es. `timescale 1ns/1ns) • **unita_tempo**: rappresenta l'unità con cui sono espressi i tempi • **precisione:** precisione con cui vengono approssimati i tempi
$display <output>	stampa i valori immediati di <output> in console
$monitor <output>	ogni volta che i valori di <output> cambiano, vengono stampati alla fine di ogni unità di tempo
$strobe <output>	stampa a video i valori di <output> alla fine di ogni unità di tempo
$readmemh	legge da un file
$stop	sospende la simulazione e attiva la modalità interattiva del simulatore
$finish	esce dalla simulazione

DEFINIZIONE DI UN MODULO

module (x0,x1,…, z0,z1,…);
input x0,x1,…;
output z0,z1,…;
…. //codice del modulo
endmodule;



COMANDI

case (<espressione>) **<costante1>:<coman.1>;** **<costante2>:<coman.2>;** **…..** **default:<comandoN>;** **endcase**	Scelta multipla: viene selezionata l'istruzione corrispondente al caso in cui l'<espressione> è uguale alla costante (**casex** considera i valori X e Z come non specificati nelle costanti nel case)
function z; **input x;** **<gruppo_comandi>;** **endfunction**	Definisce una funzione • <gruppo_comandi> deve assegnare z • x e z possono essere multi-bit
z(x)	Indica l'uso di una funzione con parametro di input e restituisce z
initial <gruppo_comandi>;	Definisce uno o più comandi che vengono valutati al primo step di simulazione
always@(<lista di sensitivita'>) **<gruppo_comandi>;**	Definisce uno o più comandi che vengono valutati ad ogni unità di tempo in cui varia la lista di sensibilità di "x". • <lista di sensitività>:= <variabile1> [or variabile2 or variabile3…] • Una variabile della <lista di sensitività> se preceduta dalla keyword **posedge** diventa sensibile solo sul fronte in salita (con **negedge** sul fronte in discesa)
wait <expressione>	ritarda l'esecuzione del successivo gruppo di comandi finché l'<espressione> non è vera
parameter <identificatore>=<costante>	esempio: parameter A = 1.
if (<espressione>) <gruppo_comandi>;	se l'espressione è vera, viene eseguito il blocco di comandi
<espressione>:= <combinazione di variabili e operatori>	gli operatori sono descritti nelle tabelle sottostanti
<=	assegnamento non bloccante
=	assegnamento bloccante

Relational Operators

Operator	Application
<	`a < b // is a less than b?` `// return 1-bit true/false`
>	`a > b // is a greater than b?`
>=	`a >= b // is a greater than or` `// equal to b`
<=	`a <= b // is a less than or` `// equal to b`

Logical Operators

Operator	Application

&&	`a && b ; // is a and b true?`
	`// returns 1-bit true/false`

||	`a || b ; // is a or b true?`
	`// returns 1-bit true/false`

!	`if (!a) ; // if a is not true`
	`c = b ; // assign b to c`

Arithmetic Operators

Operator	Application
*	`c = a * b ; // multiply a with b`
/	`c = a / b ; // int divide a by b`
+	`sum = a + b ; // add a and b`

–	`diff = a - b ; // subtract b`
	`// from a`

%	`amodb = a % b ; // a mod(b)`

Unary, Bitwise and Reduction Operators

Operator	Application
+	Unary plus & arithmetic(binary) addition
–	Unary negation & arithmetic (binary) sub-traction
=&	`b = &a ;` // AND all bits of a
=|	`b = |a ;` // OR all bits
=^	`b = ^a ;` // Exclusive or all bits of a
~&, ~|, ~^	NAND, NOR, EX-NOR all bits to-gether `c = ~& b ; d = ~| a; e = ^c ;`

~, &, |, ^	bit-wise NOT, AND, OR, EX-OR
	`b = ~a ; // invert a`
	`c = b & a ; // bitwise AND a,b`
	`e = b | a ; // bitwise OR`
	`f = b ^ a ; // bitwise EX-OR`

~&, ~|, ~^	bit-wise NAND, NOR, EX-NOR
	`c = a ~& b ; d = a ~| b ;`
	`e = a ~^ b ;`

Equality and Identity Operators

Operator	Application
=	`c = a ; // assign a to c`

==	`a == b; //equal ?`
	
	`//0==0→1,1==1→1,0==1→0,`
	`// 0==X→X,1==X→X`
	
	`//e.g., `hx == `h5 returns X`

!=	`c != a ; // is c not equal to`
	`// a, retruns 1-bit true/`
	`// false logic equality`

===	`a === b; //identical? (returns 1 bit)`
	`//also 0,1,X,Z must be identical to get 1`
	`//e.g., `hx === `h5 returns 0`

!==	`a !== b; //not identical? (returns 1 bit)`
	`//also 0,1,X,Z must be identical to get 1`
	`//e.g., `hx !== `h5 returns 1`

Shift Operators and other Operators

Operator	Application

<<	`a << 1 ; // shift left a by`
	`// 1-bit`

>>	`a >> 1 ; // shift right a by 1`

?:	`c = sel ? a : b ; /* if sel`
	`is true c = a, else c = b ,`
	`?: ternary operator */`

{}	`{co, sum } = a + b + ci ;`
	`/* add a, b, ci assign the`
	`overflow to co and the re-`
	`sult to sum: operator is`
	`called concatenation */`

{{}}	`b = {3{a}} /* replicate a 3`
	`times, equivalent to {a, a,`
	`a} */`

V220220