Università degli Studi di Verona

DIPARTIMENTO DI INFORMATICA
Corso di Laurea Magistrale in Ingegneria Informatica

Progettazione di sistemi embedded

Autori:
Matteo Iervasi
Linda Sacchetto
Leonardo Testolin

Anno Accademico 2018–2019

Indice

Prefazione

Questa dispensa si basa sugli appunti presi da Linda Sacchetto e Leonardo Testolin durante il corso di *Progettazione di Sistemi Embedded* dell'anno accademico 2018/2019. Nonostante sia stata revisionata in corso di scrittura, potrebbe contenere errori di vario tipo. In tal caso potete segnalarli inviando una mail all'indirizzo matteoiervasi@gmail.com.

1 Introduzione

2 SystemC Design Flow

3 SystemC TLM

4 SystemC AMS

5 VHDL

Lo script di sintesi dipende dal tool di sintesi, quindi se uso, per esempio con la DC_SHELL lo script può essere composto anche da 10/100 operazioni. DC_SHELL è l'ingegnerizzazione di sys, ed è nata perché è stato ceduto sys a synopsys la quale ha permesso di avere come interfaccia non più sys ma verilog o VHDL.

SLIDE 23

Devo dichiarare un componente tutte le volte che voglio usare una design entity che ho già compilato in libreria, tutte le volte che la voglio usare come componente di un oggetto devo dichiarare un componente e dichiarare un componente è un modo per dire che voglio usare quella particolare entity e quella particolare architecture come componente di un altro dispositivo. Quindi quando voglio utilizzare una design entity dentro un altro dispositivo devo dichiararlo come componente. La dichiarazione di un componente si fa tra architecture e begin. La dichiarazione di un componente si basa su tre cose: prima dichiaro il componente (cioè faccio il copia e incolla della entity e cambio da entity a component), poi devo dire che per quel componente uso quella particolare entity con quella particolare architecture (costrutto for) e infine instanzio il componente dentro la mia architecture, nel momento in cui instanzio un componente faccio due cose: il generic map e il port map. Con il generic map posso andare a modificare il generico, cioè cerco il generico e cambio il suo valore. Il port map devo sempre farlo, dice che le porte di quel componente le vado a collegare a quei segnali. Ho due sintassi diverse per fare i port map: posso riportare semplicemente i nomi dei segnali e questi vengono associati in ordine alle porte oppure metto il nome della porta =¿ nome del segnale per tutte le associazioni porta/segnale. Quando si usa questa seconda modalità si divide su più righe per avere tutte le porte una sotto l'altra e non sbagliare.

SLIDE 31

L'assegnamento dei segnali si fa con ¡=, assegno ad un segnale una certa espressione. Attenzione che è molto semplice ruotare un vettore: assegno al mio vettore dal bit 0 a 7 un vettore da 7 a 0. Posso sempre decidere quali sono le parti e l'ordine del vettore che voglio assegnare.

SLIDE 32

Bit per bit assegnare quello che voglio non viene mai usato.

È possibile decidere che un insieme di bit ad esempio 3 down to 2 valga 1. Si usa moltissimo la keyword others che va ad indicare tutto quello che non ho già assegnato. Others permette di riempire un vettore senza sapere quanto è lungo.

SLIDE 33

Si può assegnare un valore oppure si può fare un conditional assignment, il quale mi permette di assegnare un'espressione quando c'è una condizione altrimenti assegno un'altra espressione. Devono sempre finire con un ultima espressione perché non è possibile non dire al tool di sintesi e al tool di simulazione cosa cosa avviene in certe condizioni. Corrisponde all'if...then...else che c'è dentro ai processi dove scriviamo in maniera sequenziale. Negli assegnamenti condizionali scriviamo le stesse cose con gli assegnamenti condizionali. When...else si usa molto, invece si preferisce un processo al case.

SLIDE 35

Si può usare with...select...when, attenzione che non ci possono essere sovrapposizioni di when.

SLIDE 39

Non servono per produrre hw, ma servono per scrivere codice che sia facile da valutare/verificare durante la simulazione. La condizione deve essere vera, per poter scrivere sul simulatore la stringa con un certo livello di rischio, questo livello è importante perché il simulatore in base al livello può decidere se andare avanti nella simulazione oppure bloccarsi. Se io dico che le assertion sono di tipo note o warning il simulatore scrive sullo schermo e va avanti a simulare, se invece sono error o failure si blocca. È importante perché se lancio una simulazione che dura un giorno è estremamente seccante si fermi per una assertion e quindi così mi permette di decidere se voglio bloccare la simulazione perché l'errore è irrecuperabile oppure se posso andare avanti. Permette a noi di controllare ciò che sta avvenendo.

ULTIMA LEZIONE

SLIDE 41

Quando uno scrive in VHDL si deve adattare ad uno dei tre stili: data flow, behavioural o strutturale, poi in realtà ogni progetto è un mix degli stili. Se scriviamo qualcosa in stile behavioural significa che usiamo dei processi. Ci sono una serie di keyword che possono essere messe dentro un processo, alcune come l'assegnamento dei segnale, le chiamate a procedure o le asserzioni erano anche keyword data flow, quindi possono essere usate sia all'interno di un processo che all'esterno di un processo in un'architettura data flow, di conseguenza uno stesso costrutto assume comportamenti diversi in base al fatto che sia richiamato dentro un processo o fuori da un processo. Le asserzioni sono sempre asserzioni sia dentro che fuori da un processo, le chiamate a procedura non vengono quasi mai usate perché posso scrivere il codice di una procedura e se lo richiamo da dentro un processo ciò che c'è scritto nella procedura deve essere compatibile con il processo, se invece lo richiamo da dentro un'istruzione data flow deve essere compatibile con quell'istruzione data flow perciò una stessa procedura rischia di andare in errore durante la simulazione perché se sono chiamata da un processo potrei fare cose che in un processo non ci possono stare e lo stesso può succedere se vengo chiamato fuori da un processo. Nel momento in cui progettiamo del SW e scriviamo delle procedure lo facciamo per partizionare il nostro progetto, ma anche per riusare lo stesso codice (abbiamo un pezzo di codice comune che viene richiamato da più punti di un programma), nell'HW, invece, non è possibile, infatti se io utilizzo una procedura ho utilizzato una struttura di programmazione e se questa procedura viene richiamata in 3 punti diversi durante la sintesi verrà copiata nei tre punti diversi (ci saranno 3 istanze della procedura). Quindi se abbiamo qualcosa da mettere in comune non possiamo farlo con le procedure perché l'HW non implementa quei meccanismi di caricamento sullo stack dei vari

parametri della procedura, dunque devo scrivere un componente, devo gestire come fanno vari punti del mio circuito a interagire con quel componente (magari con un bus), la gestione devo farla io, non è la procedura che la risolve.

Assegnamento dei segnali I segnali sono tipici della parte parallela di un'architecture però quando un processo fa un calcolo deve comunicare il risultato all'esterno e lo fa scrivendo su un segnale. Un processo (sequenziale) che scrive su un segnale (concorrente) può creare qualche problema.

SLIDE 42

Dentro un processo posso dichiarare una variabile. Fra PROCESS e BEGIN dichiaro tutte le variabili che mi servono, vengono utilizzate per semplificare i calcoli, spezzarli in passi e per avere dei valori intermedi che servono a rendere il codice più leggibile. La variabile non è quella del linguaggio di programmazione che corrisponde ad una cella di memoria in cui metto un valore, ma è utilizzata essenzialmente per descrivere il nostro algoritmo. Cosa avverrà alla variabile dipenderà dai template della sintesi che ci diranno se la variabile scompare, se diventa un filo o un registro.

SLIDE 43

Differenza tra segnale e variabile Innanzitutto si dichiarano in posti diversi, infatti il segnale si dichiara tra ARCHITECTURE e BEGIN, mentre la variabile si dichiara tra PROCESS e BEGIN. Hanno anche visibilità diverse: il segnale ha visibilità nell'intera architettura, mentre la variabile ha visibilità nel processo in cui è dichiarata. Esiste una variante del VHDL 92, che però non viene utilizzata da nessuno, cioè le shared variable, le variabili condivise sono fatte per semplificare come ci sono nei linguaggi di programmazione concorrenti anche qui si può dichiarare una variabile shared condivisa tra due processi e i due processi si scambiano dati tramite questa variabile, dal punto di vista simulativo si può fare, poi però quando si fa la sintesi il meccanismo di una variabile shared non è semplice, infatti questo "contenitore" comune deve essere gestito con delle regole per esempio con dei semafori o dei segnali di sincronismo, quindi non posso pretendere che il tool di sintesi si occupi anche di questo. Se ho bisogno di un contenitore comune tra due processi me lo creo con un segnale che va in un registro, a cui accedono due processi diversi che si accordano per l'accesso con delle regole magari utilizzando un altro segnale di sincronismo. Quando dichiariamo un segnale o una variabile noi vogliamo inizializzarli e usiamo la stessa sintassi :=, po quando si fanno gli assegnamenti per una variabile si usa ancora :=, mentre per un segnale si utilizza ¡=.

SLIDE 45

Ma la differenza fondamentale è che le variabili non implicano un concetto di tempo mentre il segnale implica un concetto di tempo, quindi nel momento in cui assegno un valore ad una variabile quell'assegnamento viene fatto esattamente in quel momento, mentre se assegno un valore ad un segnale posso esplicitare un tempo (con after specifico tra quanto tempo farò l'assegnamento dal tempo attuale di simulazione). In ogni caso anche se non metto esplicitamente il tempo il valore che viene assegnato al segnale crea il prossimo tempo di simulazione, viene assegnato per creare un evento che manda avanti il tempo di simulazione. Se noi usiamo i segnali fuori da un processo se qualcosa a destra del segnale cambia il segnale viene ricalcolato, se il segnale è assegnato dentro un processo e lo assegno in un punto in cui ci sono istruzioni prima e istruzioni dopo siamo in qualcosa di sequenziale che deve essere sincronizzato con il resto che è fuori dal processo che è concorrente.

SLIDE 46

ESEMPIO:

Sinistra: si basa su 2 variabili il processo non ha una sensitivity list, ma utilizza un wait

Un processo che ha una sensitivity list è come se fosse un $\mathrm{SC}_{METHOD}, nelmomentoincuiunsegnaledellase$

Destra: si basa su due segnali

Al tempo 0 i due segnali sono inizializzati e quindi a t=0 valgono 0, dopo 20 ns c'è un assegnamento a segnale, è come se il mio processo venisse compresso in unici assegnamenti a segnali cioè tutte le istruzioni che non fanno riferimento a segnali non fanno passare il tempo ma non avvengono in un ordine diverso da quello sequenziale, invece l'assegnamento a segnale, dato che deve sincronizzarsi con il mondo esterno, è come se avvenisse sempre all'ultima istruzione del processo, quindi prende valore solamente all'ultima istruzione del mio processo. Di conseguenza num diventa 1 solo alla fine del mio processo, dunque per tutto il processo num vale 0, allora nell'assegnamento di sum il valore di num è ancora 0 e sum rimane 0. Rientrando nel processo succede nuovamente la stessa cosa. L'assegnamento a segnale non viene fatto nella posizione in cui lo scriviamo, perché se venisse fatto in quella posizione staremmo dando un comportamento sequenziale all'assegnamento al segnale, che non è possibile dato che il segnale è un costrutto concorrente, perciò il suo assegnamento viene sempre fatto alla fine.

Per ogni segnale viene costruito dal simulatore un driver, cioè un vettore di coppie dove si associano il tempo e il valore che il segnale dovrà avere a quell'istante di tempo. Mi dice per degli istanti di tempo i valori che assegnerò, in un driver ci sono più posizione perché può essere che prometta degli assegnamenti, faccio degli assegnamenti con dei costrutti after che ipotecano in futuro cosa avverrà. Il driver, raggiunto un istante di tempo, decide seguendo delle regole se cambiare valore al segnale oppure no. Invece la variabile nel momento che scrivo qualcosa assume quel valore e basta.

I segnali quindi vengono assegnati o alla fine del processo, oppure quando si arriva al wait successivo, infatti se dentro un processo ci sono più wait, il wait successivo corrisponde alla fine del processo (un momento in cui bisogna risincronizzarsi).

SLIDE 47

Un processo può quindi avere o no una sensitivity list, ma nel momento in cui non ce l'ha deve avere almeno un wait. Ci sono diversi tipi di wait. WAIT FOR fa aspettare del tempo, di solito, però, viene utilizzato WAIT ON (un certo segnale), cioè corrisponde esattamente alla sensitivity list perché aspetto finché su quel segnale o su almeno uno di quei segnali avviene un evento (il segnale riceve un assegnamento che gli fa cambiare valore).

SLIDE 49

Sensitivity list: nel momento in cui avviene un evento su un segnale eseguo il processo, wait on: quando un segnale cambia vado avanti. Una stessa cosa posso esprimerla con wait on oppure con una sensitivity list, attenzione però che la sensitivity list va all'inizio, mentre il wait on va alla fine. Il simulatore è un simulatore di HW, di conseguenza deve simulare ciò che da corrente al sistema, all'inizio il mio sistema deve raggiungere una condizione iniziale nota e per simulare l'accensione fa un passaggio di simulazione su tutte le istruzioni cioè esegue una volta tutte le istruzioni quindi il tempo 0 ha una sorta di tempo precedente in cui simula tutto una volta soltanto (parte dai valori di default e considera i valori di default come una sorta di evento). Inizia poi la vera simulazione e i processi con la

sensitivity list vengono percorsi fino in fondo, mentre i processi con il wait vengono eseguiti fino al primo wait quindi perché i due processi siano identici anche dal punto di vista della simulazione, anche nella fase di inizializzazione della simulazione è necessario mettere il wait in fondo. In questo caso in base all'uguaglianza tra alarm$_t$ecurrent$_t$mettosounda0oa1quindialtempo0avròsounda1oa0asecondadelvaloreinizialechehodatoaalar

SLIDE 52

I tipi di wait sono 4:

- wait: suicidio, nel senso che se scrivo "wait;" il mio processo non può più evolvere. Se noi abbiamo un processo come il testbench che genera i segnali e i valori per gli altri processi, un tipico testbench alterna una serie di assegnamenti a wait di 20 ns e se voglio interrompere uso wait, quel processo si ferma, non genera più segnali, tutti gli altri processi non ricevono più segnali, non ci sono più prossimi eventi e il simulatore interrompe la simulazione.

- wait for: scrivo quanto tempo.

-wait on con riferimento ad un segnale: aspetto fino a che c'è un cambiamento in almeno uno dei segnali nella lista (ne basta uno).

-wait util: è un'istruzione un po' più complessa perché dopo until c'è una condizione, quindi non solo i segnali della condizione devono avere un evento (subire un cambiamento), ma anche la condizione deve essere vera. Ad esempio wait until(sig=0), ci deve essere un evento sul segnale sig e alla fine di questo evento il segnale deve valere 0, se alla fine dell'evento non vale 0 non vado avanti perché avere solamente l'evento non mi basta.

SLIDE 53

Un programma di questo tipo con un testbench così posso caricarlo nel simulatore e farlo partire (run) senza problemi tanto siamo sicuri che prima o poi si fermerà. Se invece non ho un testbench scritto così con un wait finale ma ho un testbench che è dentro ad un processo e che continua ad assegnare dei valori se io do il comando run la simulazione parte e non si ferma più allora da simulatore evo dire run per un certo tempo facendo attenzione all'unità di misura a cui fa riferimento il tempo e che dunque il tempo che diamo sia coerente con il simulatore.

Wait until utilizza gli attributi (alcuni valori che tutti i segnali hanno, durante la simulazione il simulatore interroga questo attributo e ottengo il suo valore), per esempio clk'event è un attributo (booleano) che diventa vero quando c'è un evento sul segnale di clock. Questo wait until dice di aspettare finché c'è un evento sul segnale di clock, questa condizione potrebbe sembrare inutile dato che wait until funziona quando c'è un cambiamento di un segnale e di conseguenza scrivere solamente clk='1'. "clk'event and clk='1' " e "clk='1' " sono semanticamente la stessa cosa perché wait until ipotizza ci sia un evento sui segnali della condizione, ma vengono scritte entrambe perché quel template è utilizzato per indicare al tool di sintesi che quello è il segnale di clock su cui mi devo sincronizzare e verifico sia che ci sia un evento sia che sia sul fronte di salita (clk='1').

SLIDE 54

Non è else if, ma elseif tutto attaccato. È bene completare sempre le condizioni con un else finale nel caso tutti i casi precedenti non si verifichino, se non metto l'ultimo else (perché non lo ritengo necessario) vuol dire che tutte le variabili o i segnali che non sono coinvolte in quel calcolo mantengono i valori precedenti perché non li ho cambiati, ma questo dal punto di vista della sintesi significa che ci vuole un elemento di memoria altrimenti non possono mantenere i valori

precedenti. Attenzione dunque che se non vogliamo avere un comportamento sequenziale inserendo un registro dobbiamo completare tutti i rami con l'else finale.

SLIDE 55

WITH/SELECT: parallelo, CASE/WHEN: sequenziale. Non c'è il break alla fine di ogni caso ma è come se ci fosse, infatti alla fine di ogni when è come se ci fosse il break, attenzione che non possiamo sovrapporre delle condizioni. In questo caso è sbagliato perché il 2 appartiene sia alla seconda che alla terza condizione. I case/when richiede che gli intervalli siano espliciti (ci siano i numeri e non per esempio tra la variabile a e la variabile b: a to b), quindi il compilatore vede che ci sono delle sovrapposizioni.

Mettiamo che la variabile a (dell'esempio) sia in un range tra 0 e 8, potrei con il when esplicitare tutti i casi quindi when others dovrebbe essere inutile, il tool di sintesi, però, non essendo sicuro che tutti i casi sono considerati allora mette un elemento di memoria perché nei casi che non ho considerato l'assegnamento deve valere il valore precedente. Perciò è meglio mettere sempre when others tanto dal punto di vista della simulazione può tranquillamente essere inutile e dal punto della sintesi evitiamo la creazione di registri non voluti. SLIDE 56

for...loop serve per descrivere un ciclo su un indice e utilizzare quell'indice per scrivere le istruzioni una volta sola invece di srotolare le istruzioni rispetto all'indice. È un costrutto che ci semplifica la scrittura, infatti se per esempio devo fare molti assegnamenti ad una variabile o agli elementi di un vettore e questi li posso fare sfruttando un indice invece di scrivere assegnamento per assegnamento li posso mettere in un ciclo for poi il compilatore è come se srotolasse il ciclo for e scrivesse tutti gli assegnamenti con i valori degli indici. Per questo l'indice non è una variabile che dobbiamo dichiarare, ma è solamente funzionale a quel ciclo.

Se abbiamo bisogno di un contatore non usiamo il for, ma usiamo il segnale di clock e con la variabile che si incrementa.

while...loop: la condizione potrebbe riferirsi ad una variabile, il simulatore non ha problemi, ma il sintetizzatore si trova davanti delle istruzioni che devono essere ripetute fino a che una condizione non è falsa, quindi deve costruire una macchina a stati che controlla questo pezzo di codice e quando la condizione è falsa deve passare a eseguire la parte successiva, cioè fa una high level synthesis. Usando il while non facciamo la sintesi a livello RT dicendo ad ogni ciclo di clock cosa succede perché non sappiamo quanti cicli di clock ci serviranno, dipende dalla condizione. Per il while dunque dobbiamo utilizzare un tool di sintesi che faccia la sintesi ad alto livello (a livello comportamentale / algoritmico).

SLIDE 62

Le procedure rispetto alle funzioni non ritornano nulla, ma vanno solamente a modificare qualcosa dell'ambiente chiamante e per farlo usano una variabile in ingresso di tipo INOUT, perciò fornisco questa variabile alla procedura la quale la modifica. Il fatto che usiamo una porta INOUT ci rende impossibile la sintesi, infatti non vengono utilizzate.

SLIDE 65

La funzione serve per scrivere in maniera "comoda" un pezzo di codice che deve fare una determinata elaborazione. Le funzioni mi rendono più facile l'interpretazione e l'interpretazione di un codice, per esempio nel mio processo spezzo i calcoli tramite la chiamata ad una funzione. Nel momento in cui ho un calcolo che devo mettere in comune tra più parti del circuito che sto realizzando se io faccio una funzione e la richiamo tante volte alla fine della sintesi avrò

altrettante istanze della mia funzione. Le funzioni vengono processate dal tool di sintesi come il compilatore di c fa con le funzioni inline, cioè le copia nel codice. Le funzioni servono anche per riuscire a cambiare i tipi alle variabili perché è un linguaggio fortemente tipizzato, altrimenti non riusciremmo a fare le operazioni.

La funzione si spezza in due parti: dichiarazione e corpo le quali si possono scrivere in posti diversi. In particolare abbiamo i package, una sorta di header file, in cui però non abbiamo solo l'intestazione (come le dichiarazioni dell'header file), ma anche il corpo in cui mettiamo i corpi delle funzioni, quindi l'intestazione la metto nell'intestazione del package, mentre il corpo lo inserisci nel package body. Questa divisione è finalizzata alla diminuzione delle ricompilazioni, attenzione che se metto sia intestazione che corpo del package nello stesso file quando ricompilo vado a ricompilare tutto.

Nelle funzioni non metto nulla che abbia un comportamento sequenziale, perciò lavoro solo sulle variabili. Con il costrutto return ritorno il risultato. Nel corpo di una procedura, invece, posso mettere anche assegnamenti a segnali e wait che non posso mettere nelle funzioni. Se metto assegnamenti a segnali posso chiamarla sia da contesto concorrente che da un processo, con anche il wait può essere chiamata solo da un processo.

SLIDE 67

VHDL è un linguaggio ad oggetti e implementa l'overloading. È possibile chiamare con lo stesso nome metodi che lavorano su tipi diversi, questo può significare avere valori in ingresso di tipo diverso oppure produrre un risultato di tipo diverso. I valori in ingresso e quelli di ritorno hanno un tipo e in base a questo tipo il compilatore va a scegliere la funzione giusta, anche in base al punto in cui chiamo quella funzione. Non scelgo la funzione solo in base al nome, ma tra tutte le funzioni con uno stesso nome sceglie quella corretta rispetto al contesto della chiamata. Il contesto della chiamata viene risolto con una serie di regole. Ad esempio l'operatore + può essere ridefinito, ovviamente, però, non per gli interi che ritornano un intero.

SLIDE 68

Le regole che segue il compilatore sono: 1) il numero di parametri, quindi se ho una funzione che lavora su due parametri e una che lavora su tre, se nella chiamata passo tre parametri, viene scelta la funzione con tre parametri 2) a parità di numero di parametri viene considerato il tipo dei parametri passati, quindi se passo due variabili di tipo intero e due real nell'altro viene presa una o l'altra in base al fatto che ci sia una funzione che riceve due interi o due real, se passo un intero e un real il compilatore dice di non avere una funzione per quel caso 3) nome dei parametri, cioè l'uguaglianza nel nome tra parametri attuali e parametri formali 4) tipo di ritorno, se ho due funzioni una che ritorna un real e una un float e io ho come valore di ritorno uno dei due scelgo quella corretta

SLIDE 69

Il simulatore nel momento in cui ha un segnale da gestire associa al segnale un driver, il quale tiene il tempo e il valore. Il driver viene pian piano riempito durante la simulazione, man mano che la simulazione crea degli eventi su quel segnale questi vengono messi nel driver con il tempo corrispondente (quello esatto), di conseguenza man mano che la simulazione procede gli eventi vengono eliminati dal driver. Ad esempio al tempo 0 c'è scritto sul driver che il valore è 0, a tempo 0 il tool di simulazione crea un evento sul segnale perché gli assegno 0, al tempo 5 assegna 1 e così via. Se ho un driver per ogni segnale, non è ammesso che ci siano due driver per un segnale, cioè un segnale non può essere

assegnato in due posti diversi del nostro codice (da due processi). Se ho due processi non possono entrambi assegnare uno stesso segnale perché avrei due driver, nemmeno se uno assegna il segnale sul fronte di salita e l'altro sul fronte di discesa e perciò non c'è mai un conflitto, non è possibile a prescindere. Se abbiamo una situazione con due processi che vorrebbero fare un assegnamento ad uno stesso segnale o facciamo la resolution function, oppure si crea un terzo processo o un pezzo di codice concorrente a cui mandiamo questo segnale e in base ad una politica da noi definita e a ciò che noi mandiamo decide cosa assegnare al segnale. Il problema passa ora alla scelta della politica.

SLIDE 70 È una funzione fatta apposta per risolvere questa condizione. È una funzione che quando noi definiamo un segnale prima del tipo mettiamo un altro nome (rispetto alla dichiarazione di segnale c'è un nome in più), cioè il nome di una funzione che verrà chiamata tutte le volte che si fa un assegnamento a quel segnale. La funzione riceve in ingresso tutti i valori che noi vogliamo assegnare al segnale in quel momento e decide quale assegnare effettivamente. Molto utili queste funzioni, ma il sintetizzatore non le fa automaticamente. Per la simulazione si può fare questa funzione che ha in ingresso un vettore di valori e ritornerà un valore tra questi secondo la sua politica. Non posso sapere a priori quanto è grande il vettore di valori, lo so grazie agli attributi, infatti posso estrarre delle informazioni dai segnali in un qualsiasi momento del codice che scrivo e questi valori non sono solo statici (non sono risolti al momento della compilazione), ma possono essere risolti dinamicamente durante la simulazione.

SLIDE 72

'high e 'low viene risolto con il valore più grande che una variabile o un segnale possono assumere.

'left e 'right vengono usati se ho un'enumerazione.

Se ho un array posso avere 'range (non la lunghezza). Gli array se hanno una dimensione fissa si risolvono al momento della compilazione, invece nel caso precedente avevo un vettore che cambia dinamicamente e con 'length so dinamicamente quale sia la sua lunghezza.

È possibile anche creare propri attributi, potrebbero servire per annotare una variabile o un segnale. Un tempo serviva per fornire delle informazioni al tool di sintesi, se per esempio dovevo dire che una variabile doveva diventare un registro dicevo 'register, ma poi iniziò ad essere usata una strada diversa per ricevere informazioni. Le informazioni erano ricevute e fornite tramite delle direttive al tool di sintesi, ma dato che queste direttive non fanno parte del linguaggio originale allora hanno deciso che le direttive vengono date nei commenti. Ci sono quindi dei commenti in cui utilizzo delle parole chiave: pragma, se un commento utilizza questa parola chiave non è più un commento ma viene preso dal sintetizzatore come direttiva.

Per esempio pragma synthesis on o pragma synthesis off sono due commenti fondamentali che passo se non voglio sintetizzare completamente un componente molto grande. O anche il testbench non lo voglio sintetizzare tutto perché magari ha parti che non si possono sintetizzare, allora all'inizio del testbench metto una direttiva pragma synthesis off e alla fine del testbench metto pragma synthesis on e ricomincio a fare la sintesi. Così può saltare tutte le parti che hanno senso dal punto di vista della simulazione e non della sentesi.

SLIDE 73

Questi attributi si sovrappongono uno con l'altro, sono dinamici e vengono valorizzati durante la simulazione, cioè non possono essere calcolati al tempo della compilazione.

'event : c'è un evento, se è binario passo da 0 a 1 o da 1 a 0 e se non è binario c'è stato comunque un cambiamento. Possiamo chiedere se un segnale non ha avuto eventi da un certo quantitativo di tempo con 'quiet(t). 'active significa che ho fatto un assegnamento al segnale, ma non ha cambiato il valore, cioè valeva 0 e gli ho riassegnato 0. C'è stata un'attività ma questa attività non ha prodotto un evento. Acrive è vera sempre quando c'è un evento mentre event è vera quando cambia valore.

SLIDE 74

In VHDL è possibile anche gestire dei file, ma la sintassi è davvero molto pesante, infatti è più semplice scrivere in C la gestione di un file, quindi nel momento in cui vogliamo costruire un testbench esterno, un file esterno con dei dati e vogliamo che durante la simulazione i dati vengano presi da quel file facciamo un modulo (una entity) che fa questo, ma il corpo è conveniente scriverlo in C sfruttando tutte le classi del C: file open, leggere i dati, assegnarli alle porte così entrano nella simulazione VHDL. Farlo direttamente in VHDL è davvero molto pesante.

SLIDE 4-5

Adesso che conosciamo la sintassi dobbiamo vedere come si può simulare. Ci sono ancora alcuni aspetti del linguaggio che aiutano ad approcciarsi e ad organizzare meglio un progetto complesso.

DESIGN UNIT esistono quelle principali e quelle secondarie. Le primarie sono la dichiarazione del package, l'entity e la configuration, mentre le secondarie sono il corpo del package e l'architecture, ciò significa che quando compilo una design unit primaria la compilo, quando però compilo una design unit secondaria devo aver prima compilato la design unit primaria corrispondente.

Se noi all'inizio di un file non scriviamo niente, cioè iniziamo direttamente con entity perché dobbiamo descrivere un modulo in realtà è come se avessimo già scritto delle cose, infatti il compilatore ipotizza che io abbia scritto library work library std use std.standard.old Il compilatore sottintende che usiamo la libreria work dove andranno i risultati della compilazione, che usiamo la libreria standard da cui carico il package standard, il quale contiene i tipi di base. Perciò queste 3 righe ci sono indipendentemente dal fatto che noi le scriviamo o meno. Attenzione che la visibilità (lo scope) della dichiarazione di libreria è relativa esclusivamente alla design entity successiva, ciò significa che se io decido che ho bisogno di una costante e nel package definisco una costante questa non sarà visibile in tutto il file, ma ???????????? NON CI STO CAPENDO