Lez. 4 – Modellazione in Verilog

Laboratorio di Architettura degli Elaboratori

Michele Lora

23 Novembre 2023

Livelli di astrazione in Verilog

- **Comportamentale**: il comportamento del sistema hardware viene descritto specificando le azioni da comprendere, in maniera quasi algoritmica.
- **Strutturale**: il comportamento del sistema hardware viene descritto come aggregazione e interconnessione di componenti, i quali a loro volta specificheranno delle funzionalità in uno dei tre stili di modellazione.
- **Gate-level**: il comportamento del sistema hardware viene descritto come aggregazione di porte logiche di base, quali porte **and**, **or**, **xor**, **nand**, etc.

Tipi di dato

I segnali in Verilog possono avere solo i quattro valori corrispondenti a stati fisici di segnali hardware:

- `'0'`: zero logico, oppure una condizione false;
- `'1'`: uno logico, oppure una condizione vera;
- `'X'`: valore sconosciuto, oppure don't care;
- `'Z'`: alta impedenza del segnale.

È possibile esprimere valori numerici a più bit utilizzando i letterali numerici, la cui struttura base è: `D'Bvalore`

- `D`: dimensione del segnale (numero di bit) espresso come numero in base 10.
- `B`: base, può essere una delle seguenti: `d`: base 10, `b`: base 2, `o`: base 8, `h`: base 16 (esadecimale).
- Esempi: `32'd17` $= (17)_{10}$, `32'hABCD` $= (43981)_{10}$

Tipi di dato

Gli elementi di memorizzazione e le connessioni tra componenti vengono rappresentati mediante due costrutti principali:

- `wire`: rappresenta la connessione tra elementi, il cui valore è guidato solamente dall'uscita di una rete logica (oppure, mediante **assegnamenti continui**).
- `reg`: modella il comportamento di un elemento di memorizzazione.

Per specificare **modelli a livello comportamentale** è possibile utilizzare variabili, simili a quelle dei linguaggi di programmazione:

```
integer i; // variabile intera a 32 bit.
real r;    // variabile Reale (floating point).
time t;    // variabile di tipo tempo.
```

Le variabili non possono essere dichiarate dentro a blocchi sequenziali, ma solo nel preambolo del modulo.

Moduli e interfacce

Il costrutto `module` è l'unità fondamentale di un modello Verilog.

Un modulo rappresenta un componente del sistema, caratterizzato da un'**interfaccia**, ossia la definizione delle sue **porte di ingresso** e **porte di uscita**.

Esempio

```
module complemento(input [3:0] in, output [3:0] out);
begin
...
endmodule
```

Blocchi sequenziali

Blocco sequenziale: costrutto principale per la modellazione del comportamento di un sistema elettronico. È descritto da un insieme di istruzioni comprese tra le parole `begin` and `end`. Esistono due tipi di blocchi sequenziali:

- blocco `always`: blocco che viene eseguito ogniqualvolta si verifica un evento.
- blocco `initial`: blocco che descrive il comporamento del sistema a partire dall'istante iniziale della sua esecuzione.

Sintassi always

```
always @( evento )
begin : nome
    ...
end
```

Sintassi initial

```
initial begin : nome
    ...
end
```

Costrutti condizionali e cicli

Il linguaggio Verilog mette a disposizione vari costrutti condizionali ed iterativi:

- costrutto condizionale `if`;
- costrutto condizionale `case`;
- costrutto di ciclo `while`;
- costrutto di ciclo `for`;
- blocco sequenziale `repeat`.

Costrutto condizionale **if**

```
if(condizione) begin
    [istruzioni sequenziali]
end
else if(condizione) begin
    [istruzioni sequenziali]
end else
    [istruzioni sequenziali]
end
```

Costrutto condizionale **case**

```
case(espressione)
    valore_1: begin
        [blocco di istruzioni sequenziali]
    end
    valore_2: begin
        [blocco di istruzioni sequenziali]
    end
    ...
    default: begin
        [blocco di istruzioni sequenziali]
    end
endcase
```

Costrutti iterativi, cicli **while**, **for** e **repeat**

Sintassi del ciclo **while**

```
while(condizione) begin
    [istruzioni sequenziali]
end
```

Sintassi del ciclo **for**

```
for(inizializzazione; condizione; update) begin
    [istruzioni sequenziali]
end
```

Sintassi del blocco **repeat**

```
repeat(numero) @ (evento) begin
    [istruzioni sequenziali]
end
```

Assegnamenti

Assegnamento continuo dei segnali

Segnali: tipicamente oggetti di tipo `wire`. Vengono continuamente assegnati mediante il costrutto:

```
assign segnale = [ritardo] valore o espressione;
```

Assegnamento dei registri

Due tipi diversi di assegnamento ai registri:

- **assegnamenti non bloccanti**:
  `registro <= valore o espressione;`
- **assegnamenti bloccanti**:
  `registro = valore o espressione;`

Solitamente, entrambi utilizzati dentro a blocchi sequenziali.

Esempio 1: Calcolo del complemento a 2

```
module complemento(input [3:0] in, output [3:0] out);

    integer i;
    reg [3:0] negato;

    always @(in)
    begin
        for(i = 0; i < 4; i = i + 1) begin
            negato[i] <= !in[i];
        end
    end

    assign out = negato + 1;
endmodule
```

Esempio 2: Sommatore a 4 bit

```
module somma4bit(
    input [3:0] a,
    input [3:0] b,
    output reg [3:0] out);

    always @(a or b) begin
        out = a + b;
    end
endmodule
```

Stimolare il modello: testbench

Il **testbench** è il modulo che genera gli ingressi e legge le uscite del modello che si sta progettando. Dunque, il suo ruolo è quello di stimolare l'esecuzione del modello in esame.
Due alternative principali:

- **module** ad-hoc che istanzia il componente in esame;
- **script** di comandi che generano gli ingresso del componente in esame.

In questo corso, ci limitiamo a utilizzare la prima soluzione.
Comandi utili alla creazione di un testbench:

- `$dumpfile(nomefile.vcd)`: crea un file di tracce (waveform) in cui salvare l'andamento della simulazione;
- `$display(argomenti)`: stampa a video la stringa creata specificando gli argomenti (sintassi C-like).

Esempio: testbench per il complemento a 2.

```
‘timescale 1ns / 1ps

module tb();
    reg [3:0] in;
    wire [3:0] out;

    integer i;

    complemento c(.in(in), .out(out));

    ...
```

Esempio: testbench per il complemento a 2.

```
...
    initial begin
        $dumpfile("dump.vcd");
        $dumpvars(1);
        for(i = -7; i <= 7; i = i + 1) begin
        in <= i;
        #2;
        $display("in: %d -> out: %d",
            $signed(in), $signed(out));
        end
    end
    endtask
endmodule
```

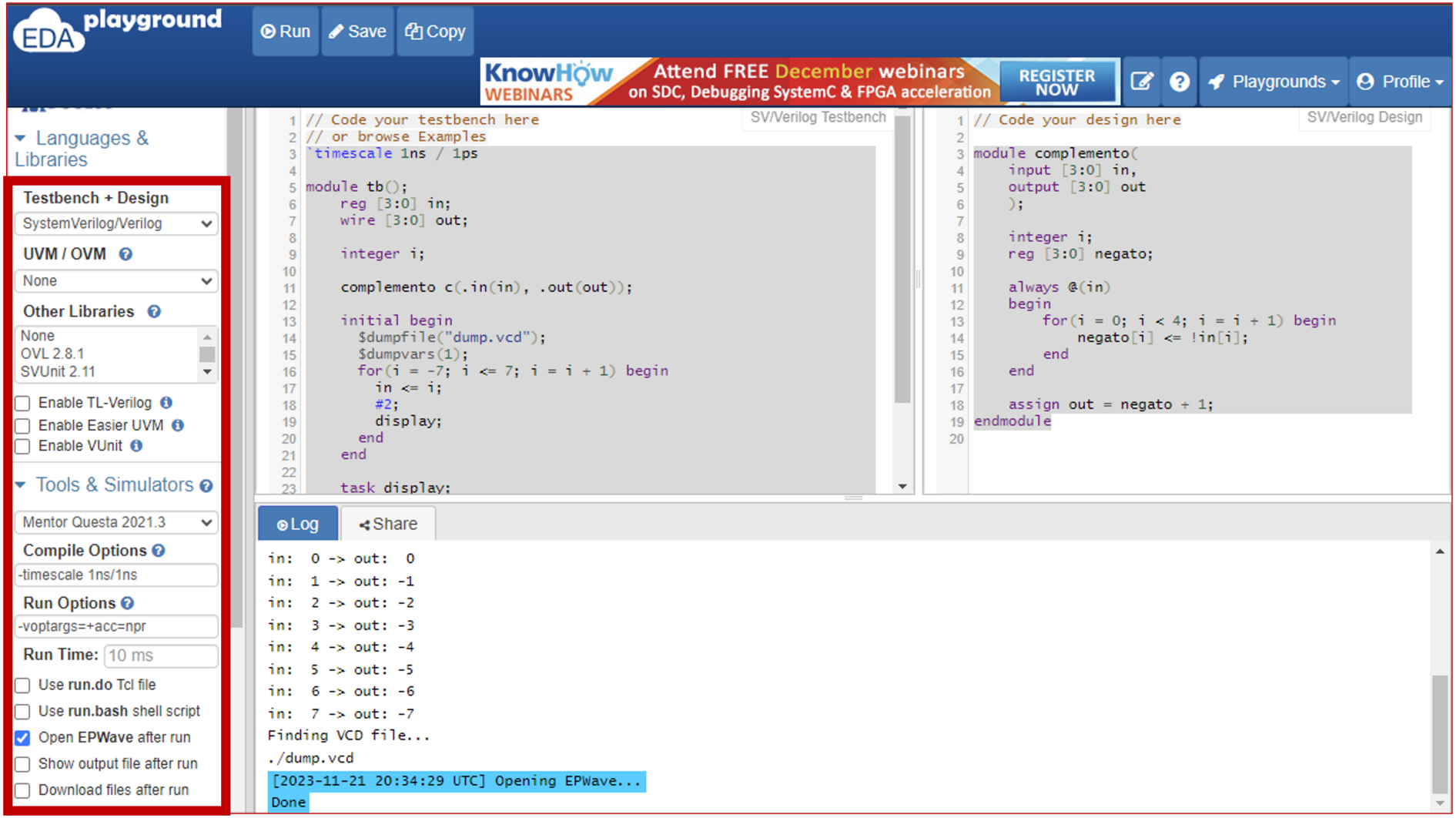
Simulazione di Verilog

Durante il corso utilizzeremo **EDAPlayground** un insieme di strumenti online: `https://www.edaplayground.com/`

EDAPlayground mette a disposizione un editor di testo, diversi simulatori (sia commerciali che open-source) e tool di sintesi. È necessario registrarsi, se la registrazione viene effettuata con l'indirizzo email **istituzionale** (**e.g.**, @studenti.univr.it) permette di utilizzare i simulatori commerciali.

In laboratorio useremo **Mentor Questa**, un tool commerciale. Dunque, è consigliato di effettuare la registrazione a EDAPlayground utilizzando la propria email istituzionale. In alternativa, è possibile usare Icarus Verilog (strumento libero).

Configurazione di EDAPlayground



Modellazione a Gate Level

Verilog permette di specificare modelli pi'u vicini a quella che sarà l'implementazione finale del circuito. Il linguaggio mette a disposizione una libreria di porte logiche standard che implementano le funzioni Booleane corrispondenti. Le porte nella libreria sono: `and`, `or`, `xor`, `nand`, `nor`, `xnor`, e `not`.

Ogni porta logica richiede di specificare i segnali di uscita seguiti da quelli di ingresso. Quindi, ad esempio, $O = A \vee B$ sarà implementata specificando:

```
or(O, A, B);
```

gate e continuous assignments: la rappresentazione a gate level può essere espressa anche mediante assegnamenti continui, conoscendo l'operatore bit-wise corrispondente ad ogni porta:
`https://www.asic-world.com/verilog/operators1.html`

Esempio: sommatore ad 1 bit - gate level

```
module sommatore( input A, B, CIN, output O, COUT);

    wire V, W, Z;

    xor(V, A, B);
    xor(O, V, CIN);
    and(W, A, B);
    and(Z, V, CIN);
    or(COUT, W, Z);
endmodule
```

Esercizio 1

Scrivere il testbench per il sommatore a quattro bit (esempio 2), descritto con lo stile comportamentale. Utilizzare i costrutti iterativi (`for` oppure `while`) per testare tutte le possibili combinazioni di ingresso.

Esercizio 2

Scrivere un componente che riceve in ingresso due numeri a 4 bit ed un segnale di controllo. Se il segnale di controllo vale '0' il componente effettuerà la somma dei due numeri in ingresso; altrimenti, eseguirá la sottrazione. Utilizzare lo stile comportamentale per descrivere il circuito.

Esercizio 3

Riscrivere il sommatore ad 1 bit descritto a gate level utilizzando gli assegnamenti continui.