Géneration en Verilog

Michel Billaud (michel.billaud@laposte.net)

20 juin 2022

Table des matières

1 Objectifs **1**

2 Cas général, boucle de génération **2**

3 Relier les retenues entrante et sortante du circuit **2**

4 Code du module **3**

1 Objectifs

Dans un épisode précédent, on a présenté une définition structurelle d'un additionneur 2 x 4 bits, dans laquelle on faisait apparaitre explicitement les 4 additionneurs complets qui traitent chacun un bit.

```
`include "full-adder.vl"

module four_bit_adder
  (
   output              co,
   output[NB_BITS-1:0] s,
   input [NB_BITS-1:0] a,
   input [NB_BITS-1:0] b,
   input               ci
   );

   parameter NB_BITS = 4;
```

```
    wire [NB_BITS : 1] c;

    full_adder fh0(c[1], s[0], a[0], b[0], ci);
    full_adder fh1(c[2], s[1], a[1], b[1], c[1]);
    full_adder fh2(c[3], s[2], a[2], b[2], c[2]);
    full_adder fh3(co,   s[3], a[3], b[3], c[3]);

endmodule // four_bit_adder
```

Avec l'instruction `generate` de Verilog, nous allons éviter de construire ce type de code grâce à une boucle de génération, au lieu de faire du copier-coller-modifier.

2 Cas général, boucle de génération

On remarque que les 4 instances de "full_adder" sont *à peu près* similaires, et de la forme

```
full_adder fh(c[i + 1], s[i], a[i], b[i], c[i]);
```

pour `i` variant de 0 à 3.

On remplace les 4 déclarations par une **boucle de génération**, avec une variable de contrôle `i` :

```
generate
    genvar i;
        for (i = 0; i < NB_BITS; i = i + 1) begin
            full_adder fh(c[i + 1], s[i], a[i], b[i], c[i]);
        end
endgenerate
```

3 Relier les retenues entrante et sortante du circuit

Comme nous faisons maintenant usage de `c[0]` et `c[4]`, il faut commencer par adapter la déclaration de `c` :

```
wire [NB_BITS : 0] c;
```

et s'occuper de relier

- la retenue `c[0]` avec l'entrée `ci`,
- la sortie `co` avec `c[4]`.

Ce qui s'écrit

```
    assign c[0] = ci;
    assign co = c[NB_BITS];
```

Ces deux instructions `assign`, qu'on appelle **affectations ou assignations continues**, décrivent des **connexions permanentes**, des fils qui font que

- la retenue entrante `c[0]` de l'additionneur de poids faible prend sa valeur sur `ci` (retenue entrante du circuit)
- la retenue sortant du circuit `co` tient sa valeur de la retenue sortant de l'additionneur de poids fort.

Important ne pas confondre les affectations continues (des fils de connexion entre circuits) et les affectations que l'on utilise dans les scénarios de simulation pour changer la valeur d'un registre à un moment donné.

4 Code du module

Avec ces modifications, le code du module devient :

```
// four-bit-adder.vl
//

`include "full-adder.vl"

module four_bit_adder
  (
   output              co,
   output[NB_BITS-1:0] s,
   input [NB_BITS-1:0] a,
   input [NB_BITS-1:0] b,
   input               ci
   );

   parameter NB_BITS = 4;

   wire [NB_BITS : 0] c;     // changement

   assign c[0] = ci;         // connexions à ci et co
   assign co = c[NB_BITS];

   generate                  // génération des 4 additionneurs complets
      genvar i;
      for (i = 0; i < NB_BITS; i = i + 1) begin
         full_adder fh(c[i + 1], s[i], a[i], b[i], c[i]);
      end
   endgenerate

endmodule // four_bit_adder
```