FPGA: La suite

/tmp/lab

Samedi 28 mars 2009

Le workshop précédent

- Fonctions combinatoires simples
- Quelques registres (bascules D)
- Utilisation de l'éditeur de schémas (que vous avez tous aimé)

Comment réaliser les circuits d'aujourd'hui, beaucoup plus complexes ?

- Conception
- Débogage

Solutions...

- Langage de description du matériel (ex. Verilog, VHDL)
  - *Conception rapide de circuits complexes*
  - *Réalisation de “test benches” et simulations*
- Utilisation d'ISE en ligne de commande
  - *Pas obligatoire mais appréciable !*

Les modules Verilog

- Un “module” est une “boîte noire” ayant des signaux d'entrée et des signaux de sortie
- Il peut être
  - Un “top-level module” (module de niveau supérieur en Québecois) : les entrées et sorties correspondent aux pins du FPGA
  - Utilisé dans un autre module : ses entrées et ses sorties sont connectées à l'intérieur du module hôte

Les modules Verilog

```
module workshop2(
    input clock,
    output speaker,
    input [3:0] buttons,
    output [3:0] leds
);

endmodule
```

Logique combinatoire simple

- A l'aide de l'opérateur “assign”
- Syntaxe des expressions proche de celle du C
- Constantes
  - 1'b0
  - 5'b01011
  - 16'h39F3
  - 32'd394982

Logique combinatoire simple

```
module workshop2(
	input clock,
	output speaker,
	input [3:0] buttons,
	output [3:0] leds
);
assign speaker = 1'b0;
assign leds[0] = buttons[0] & buttons[1];
assign leds[1] = buttons[0] | buttons[1];
assign leds[2] = buttons[0] | (buttons[2] ^ buttons[3]);
assign leds[3] = 1'b1;
endmodule
```

Utilisation en ligne de commande

- Suite d'outils : xst, ngdbuild, map, par, bitgen
- Utilisation de script (Makefile)
- Fichiers d'entrée:
  - xst: sources HDL, paramètres (type FPGA...)
  - ngdbuild: fichier produit par “xst”, fichier UCF (positions des pins)
  - map: fichier produit par “ngdbuild”
  - par: fichier produit par “map”
  - bitgen: fichier produit par “par”

Hands-on #1

- Synthétiser le fichier Verilog donné en exemple, et le charger dans le kit

Hands-on #2

- Allumer la première diode si un nombre impair de touches est activé
- Les autres diodes doivent être éteintes

Verilog: signaux locaux

```
module workshop2(
	input clock,
	output speaker,
	input [3:0] buttons,
	output [3:0] leds
);
assign speaker = 1'b0;
wire light_all;
assign light_all = buttons[0] ^ buttons[1] ^ buttons[2] ^ buttons[3];
assign leds[0] = light_all;
assign leds[1] = light_all;
assign leds[2] = light_all;
assign leds[3] = light_all;
endmodule
```

Hands-on #3

- Tester le fichier précédent

Simulation

- La simulation se fait à l'aide d'un module Verilog générant des signaux de tests (appellé “test bench”)
- Ce module instantie le sous-module à tester
- Ce module ne possède pas d'entrée ni de sortie
- Pour vérifier le bon fonctionnement, on peut :
  - Enregistrer les formes d'onde dans un fichier
  - Afficher des informations au cours du fonctionnement
  - Appeler du code C (VPI)
  - ...

Instantier le module à tester

```
module testbench();

reg clock;
wire speaker;
reg [3:0] buttons;
wire [3:0] leds;
workshop2 dut(
        .clock(clock),
        .speaker(speaker),
        .buttons(buttons),
        .leds(leds)
);

endmodule
```

Générer les signaux de test

```
initial begin
        // nous allons écrire les formes d'onde dans le fichier “workshop.vcd”
        // lisible ensuite par GTKWave
        $dumpfile(“workshop.vcd”);
        // active l'enregistrement des signaux du sous-module “dut”
        $dumpvars(dut, 0);
        // génère quelques signaux de test
        clock = 1'b0;
        buttons = 4'b0000;
        #10 buttons = 4'b0010; // #10 sert à attendre 10 unités de temps
        #10 buttons = 4'b0110;
        #10 buttons = 4'b1000;
        #10 buttons = 4'b0001;
        // arrête le simulateur
        $finish;
end
```

Hands-on #4

- Lancer le simulateur
  - GPL Cver : cver testbench.v workshop.v
  - Icarus Verilog : iverilog -o test testbench.v workshop.v && ./test
- Afficher les formes d'onde
  - gtkwave workshop.vcd
- Vérifier
  - Dans ce workshop nous vérifions à la main, mais en pratique on utilise souvent des scripts

Verilog: bloc always

- Permet de générer automatiquement de la logique (combinatoire ou séquentielle) décrite par un “programme”
- Exemple:

```
always @(buttons[0], buttons[1], buttons[2]) begin
        if(buttons[0])
                leds[0] = buttons[1] & buttons[2];
        else
                leds[0] = buttons[1] | buttons[2];
end
```

- Le bloc always est exécuté à chaque changement des signaux de la liste de sensibilité
- Les sorties du circuit généré doivent correspondrent à celles décrites par le “programme”
- Les signaux modifiés par un bloc always doivent être déclarés comme “reg” (ce n'est qu'une question de syntaxe)

Hands-on #5

```
module workshop2(
	input clock,
	output speaker,
	input [3:0] buttons,
	output [3:0] leds
);
assign speaker = 1'b0;
reg light_all;
always @(buttons[0], buttons[1], buttons[2]) begin
	if(buttons[0])
		light_all = buttons[1] & buttons[2];
	else
		light_all = buttons[1] | buttons[2];
end

assign leds = {4{light_all}};
endmodule
```

Exercice

- Dessiner le circuit généré par le source Verilog précédent

Arithmétique

- Il est possible d'utiliser les opérateurs +, -, *, / sur les vecteurs de bits
- Exemple:

```
wire [12:0] a;
wire [12:0] b;
wire [13:0] c;
assign c = a + b;
```

- La division ne fonctionne pas ou fonctionne mal avec la plupart des FPGA – à éviter

Hands-on #6

- Ajouter 1 au nombre encodé en binaire à l'aide des touches, et afficher le résultat sur les LEDs

La liste de sensibilité

- Pour générer des bascules D
  - always @(posedge clock)
  - always @(negedge clock)
- Pour inclure automatiquement tous les signaux utilisés comme entrées
  - always @(*)

Hands-on #7

- Compter les appuis sur une des touches, et afficher le résultat en binaire sur les LEDs

Hands-on #8

- Faire clignoter les 4 LEDs en même temps
  - En utilisant l'horloge de la carte
  - Sur les kits Avnet: horloge à 16MHz sur C10
  - Indice: se baser sur le compteur précédent...

Hands-on #9

- Emettre un “bip” continu
  - C'est comme pour les LEDs clignotantes, mais avec une fréquence plus élevée...

Hands-on #10

- Emettre un “bip” à 440Hz
  - En divisant 16MHz par 36363 on obtient 440.007700134...Hz ce qui conviendra très bien pour ce workshop :)

Hands-on #11

- Générer 3 autres fréquences
  - 440Hz = note “La”
  - Pour ajouter un demi-ton, multiplier la fréquence par la racine douzième de 2
  - Exemple: fréquence du Si = 440*2^(1/12)
  - Pour enlever un demi-ton, diviser

Hands-on #12

- Choisir l'une des fréquences (notes) parmi les 4 en appuyant sur l'un des boutons

Ressources

- FPGA4fun: www.fpga4fun.com - tutoriels, projets
- ASIC World: www.asic-world.com - tutoriels
- Opencores: www.opencores.org - « bibliothèque » de designs
- Milkymist: www.milkymist.org
- GRLIB: www.gaisler.com
- OpenSparc : www.opensparc.org