計算機組織與 RISC-V CPU 設計 · 第 22 章

Verilog HDL 基礎

硬體思維 / 模組與埠 / 組合與時序邏輯 / 測試平台 / 常見陷阱

摘要

本章為第三部的工具課: 以「做 CPU 需要什麼」為範圍, 精講 Verilog 硬體描述語言。核心觀念只有一個——**Verilog 描述的是電路, 不是程序**: 每一行都對應真實的閘、導線與正反器, 所有 always 區塊永遠並行存在。我們將依序建立: 模組與埠、wire/reg 與四值邏輯、組合邏輯 (assign 與 always @(*))、時序邏輯 (always @(posedge clk) 與非阻塞指定)、參數化、記憶體陣列與 $readmemh、測試平台寫法, 以及新手最常犯的錯誤清單。本章所有語法都將在第 23–26 章的 CPU 程式碼中反覆出現。

目錄

1 硬體思維: 與軟體語言的根本差異

	C(程序)	Verilog(電路)
執行模型	逐行循序執行	所有模組/區塊永遠同時運作
變數	記憶體位置	導線 (wire) 或儲存元件 (暫存器)
if/case	控制流程	多工器 (選擇電路)
for 迴圈	重複執行	複製硬體 (generate) 或僅限模擬
函式呼叫	堆疊跳轉	模組實例化 (實體電路一份)

注意

寫 Verilog 時, 問自己的問題永遠是:「這段描述會合成出什麼電路?」寫出漂亮但不可合成 (或合成出非預期電路) 的程式碼, 是初學者的第一大坑。

2 模組、埠與實例化

```
module mux2 #(
    parameter WIDTH = 32          // 參數化位寬
)(
    input  wire [WIDTH-1:0] a,
    input  wire [WIDTH-1:0] b,
    input  wire             sel,
    output wire [WIDTH-1:0] y
);
    assign y = sel ? b : a;      // 條件運算子 = 多工器
endmodule

// 實例化(named port connection, 建議寫法)
mux2 #(.WIDTH(32)) u_alusrc_mux (
    .a(rs2_data), .b(imm), .sel(alu_src), .y(alu_b)
);
```

程式 1 模組樣板:2 對 1 多工器

模組是硬體的基本封裝單位; 實例化一次就是放一份實體電路。第 23 章的 CPU 頂層就是把 ALU、暫存器檔、控制器等模組「接線」起來。

3 資料型態與常數

- **wire**: 導線, 必須被連續驅動 (assign 或模組輸出);
- **reg**:「在 always 區塊中被指定的訊號」——不一定是正反器**!** 組合 always 中的 reg 仍是純組合邏輯;
- 四值邏輯:0、1、**x**(未知)、**z**(高阻抗); 模擬中看到 x 通常代表未初始化或多重驅動;
- 常數寫法:`32'h00000013`(32 位元十六進位)、`5'd10`、`4'b0010`;
- 向量切片:`instr[19:15]`; 拼接:`{a, b[3:0], 2'b00}`; 複製:`{{20{instr[31]}}, instr[31:20]}` ——這正是第 23 章立即值產生器的符號延伸寫法。

4 組合邏輯

兩種等價寫法:

```
// 寫法一: 連續指定 (簡單邏輯)
assign zero = (y == 32'b0);

// 寫法二: always @(*) + case (複雜選擇, 如 ALU)
always @(*) begin
    case (alu_op)
        4'b0000: y = a + b;
        4'b0001: y = a - b;
        // ...
        default: y = 32'b0;      // 必須有 default!
    endcase
end
```

程式 2 組合邏輯:assign 與 always @(*)

組合 always 的兩條鐵律

(1) 敏感列表用 @(*), 讓工具自動推導;(2) 所有分支都要指定所有輸出 (或在區塊開頭給預設值)——漏掉任何一種情況, 工具會推斷「保持原值」, 合成出意外的鎖存器 **(latch)**。第 23 章 control.v 在 always 開頭統一給預設值, 就是為了這條。

5 時序邏輯

```
reg [31:0] pc;
always @(posedge clk or negedge rst_n) begin
    if (!rst_n)
        pc <= 32'b0;             // 非同步重置
    else
        pc <= pc_next;           // 非阻塞指定 <=
end
```

程式 3 時序邏輯: 程式計數器

5.1 阻塞 (=) 與非阻塞 (<=)

- =(阻塞): 立即生效, 後面的敍述看得到新值——用於組合 always;
- <=(非阻塞): 右式在時脈邊緣先「取樣」, 所有左式同時更新——用於時序 always, 正確模擬所有正反器同時打拍的物理事實。

為什麼時序邏輯必須用 <

交換兩個暫存器:

```
always @(posedge clk) begin a <= b; b <= a; end  // 正確: 交換
always @(posedge clk) begin a = b;  b = a;  end  // 錯誤: b 得到 b
```

管線暫存器 (第 25 章) 是一整排這樣的正反器:IF/ID、ID/EX、EX/MEM、MEM/WB 同一邊緣同時更新, 全靠非阻塞語意。

6 記憶體陣列與初始化

```
reg [31:0] regs [0:31];          // 32 個 32 位元字
reg [31:0] mem  [0:1023];        // 4 KiB 指令記憶體

initial $readmemh("program.hex", mem);   // 從十六進位檔載入

assign rd1 = (ra1 == 5'd0) ? 32'b0 : regs[ra1];  // 讀埠 (組合)
always @(posedge clk)
    if (we && wa != 5'd0) regs[wa] <= wd;        // 寫埠 (時序)
```

程式 4 記憶體: 暫存器檔與 $readmemh

這就是第 23 章 regfile.v 與 imem.v 的骨架: 非同步讀、同步寫、x0 恆零。

7 測試平台 (Testbench)

測試平台是不可合成的模擬程式: 產生時脈與重置、施加激勵、檢查結果:

```
`timescale 1ns/1ps
module tb;
    reg clk = 0, rst_n = 0;
    always #5 clk = ~clk;            // 100 MHz 時脈

    top_single_cycle dut (.clk(clk), .rst_n(rst_n), ...);

    initial begin
        $dumpfile("wave.vcd");       // 波形輸出
        $dumpvars(0, tb);
        rst_n = 0; repeat (2) @(posedge clk);
        rst_n = 1;                   // 釋放重置
        // ... 等待/檢查 ...
        if (dut.u_dmem.mem[0] == 32'hC0DE600D) $display("PASS");
        $finish;
    end
endmodule
```

程式 5 測試平台骨架

常用系統任務:`$display`(印出)、`$monitor`、`$finish`、`$fatal`、`$dumpfile`/`$dumpvars`(VCD 波形,GTKWave 檢視)。層級參考 `dut.u_dmem.mem[0]` 可直接窺探內部訊號——驗證的利器 (第 26 章)。

8 工具流程

```
iverilog -g2012 -o sim.vvp alu.v regfile.v ... tb.v   # 編譯
vvp sim.vvp                                           # 模擬
gtkwave wave.vcd                                      # 看波形
```

程式 6 Icarus Verilog 模擬流程

本教材使用開源的 Icarus Verilog; 業界常用 Verilator(開源、極快)、VCS、Questa。FPGA 合成用 Vivado/Quartus 或開源的 Yosys。

9 常見陷阱清單

1. 意外鎖存器: 組合 always 分支不完整 (前述鐵律)。
2. 混用 = 與 <=: 時序用 <=、組合用 =, 絕不混用。
3. 多重驅動: 同一 wire 被兩處 assign, 或同一 reg 在兩個 always 中指定——模擬出 x、合成報錯。
4. 組合迴圈: 組合訊號繞一圈回到自己 (我們在本書開發中就真實踩過: 暫存器檔的寫穿旁路在單週期 CPU 形成迴圈——輸出 wb_data 經旁路回到 ALU 輸入再回到 wb_data, 模擬器直接卡死。解法: 旁路做成參數, 只在管線版開啟)。
5. 位寬不符:Verilog 靜默截斷/延伸; 比較帶號數要用 `$signed()`(見 alu.v 的 slt/sra)。
6. 用 **initial** 初始化可合成邏輯:FPGA 可以、ASIC 不行; 正式設計一律用重置訊號。

10 本章重點回顧

本章要點

- Verilog 描述電路:always 區塊全部並行;if/case 是多工器。
- 組合:always @(*) + =、分支必須完整; 時序:posedge clk + <=。
- reg 不等於正反器;wire 需連續驅動;x 代表未初始化或衝突。
- 記憶體 =reg 陣列;$readmemh 載入程式;testbench 以層級參考檢查內部狀態。

11 練習

1. 寫出 4 對 1 多工器的三種寫法: 巢狀條件運算子、case、if-else。
2. 下列程式碼會合成出什麼問題?`always @(*) if (en) y = a;` 如何修正?
3. 用 Verilog 寫一個 0–9 循環計數器 (同步重置), 並寫 testbench 驗證。
4. 解釋為什麼管線暫存器必須用非阻塞指定; 若用阻塞會發生什麼?
5. 寫一個 8 位元移位暫存器, 並以 $monitor 觀察它十個週期。

參考資料

[1] IEEE Std 1364-2005, *IEEE Standard for Verilog Hardware Description Language*.

[2] S. Harris and D. Harris, *Digital Design and Computer Architecture, RISC-V Edition*, Morgan Kaufmann, 2021.

[3] C. Cummings, “Nonblocking Assignments in Verilog Synthesis, Coding Styles That Kill!”, SNUG 2000.