

# Verilog 数字电路设计基础

王思成

May 16, 2026

在现代电子技术领域，数字电路设计扮演着核心角色，它支撑着从智能手机到人工智能芯片再到现场可编程门阵列（FPGA）等众多设备的运行。这些设备无处不在，推动着信息时代的快速发展。回顾历史，数字电路设计最初依赖原理图工具，手动连接逻辑门和触发器，但随着电路复杂度急剧上升，这种方法很快显露出局限性。硬件描述语言（HDL）的出现彻底改变了这一局面，它允许工程师以文本形式描述硬件行为，并通过综合工具自动生成网表和比特流。Verilog 作为最早的标准之一，于 1995 年以 IEEE 1364 标准发布，后来演变为更强大的 SystemVerilog，如今已成为集成电路（IC）设计的主流语言。

为什么选择学习 Verilog 呢？与竞争对手 VHDL 相比，Verilog 的语法更简洁，类似于 C 语言，这让软件背景的开发者更容易上手。它的应用场景广泛，包括 ASIC（专用集成电路）设计、FPGA 原型验证以及可编程逻辑实现。掌握 Verilog 不仅仅是学习一种工具，更是培养硬件思维方式——从并行执行的视角思考电路行为，这对进入 IC 设计行业至关重要。通过 Verilog，你能直接操控硅片上的逻辑，实现高效能、低功耗的硬件加速器。

本文针对初学者撰写，适合电子工程学生、嵌入式开发者以及自学者。读者需具备基本数字逻辑知识，如与门、或门和触发器的功能，但无需深入硬件经验。建议结合仿真工具实践，例如 ModelSim 或 Xilinx Vivado，这些工具能即时验证代码，提供波形查看功能，帮助你从仿真走向实际部署。本文的结构清晰：从 Verilog 基础语法入手，逐步深入行为建模、时序逻辑、结构化设计、高级主题，最后以工具链和实践项目收尾。每节末尾附实践挑战，鼓励动手编码。

实践挑战：安装 Icarus Verilog，编写一个简单与门模块并仿真，观察输入变化对输出的影响。

## 1 Verilog 基础语法

Verilog 代码的核心是模块，它封装了硬件单元的基本结构。以一个简单示例模块来说明：

```
1 module and_gate (input a, input b, output y);  
    assign y = a & b;  
3 endmodule
```

这里，`module` 关键字定义模块名称 `and_gate`，括号内列出端口：`input a` 和 `input b` 为输入信号，默认单比特宽；`output y` 为输出。模块体中，`assign` 语句实现连续赋值，`&` 是逻辑与运算符，将 `a` 和 `b` 的与结果赋给 `y`。`endmodule` 标记结束。这个结构模拟了一个与门，端口是模块与外部世界的接口。注意，`input` 默认是 `wire` 类型，`output` 可隐式为 `wire` 或 `reg`，具体取决于赋值方式。

Verilog 的数据类型多样，`wire` 是最基础的，用于组合逻辑连接，默认驱动强度为高阻抗。它常指定位宽，如 `wire [7:0] data;`，表示 8 位向量，高位（MSB）为 `bit7`，低位（LSB）为 `bit0`。访问时用方括号，例如 `data[3]` 取第 4 位（从 0 计数）。`reg` 类型则用于过程赋值，如 `always` 块中，可存储值，但综合后通常映射

为触发器，而非软件中的寄存器。integer 是 32 位有符号整数，适合循环索引，如 integer i;，real 为浮点数，如 real pi = 3.14;，但在 RTL 设计中少用，因 FPGA 不支持浮点硬件。

常量定义增强代码复用。parameter 用于模块参数化，例如：

```
1 module fifo #parameter WIDTH = 8, DEPTH = 16 (input clk, output [WIDTH-1:0] dout);
    // 使用 WIDTH 和 DEPTH
3 endmodule
```

实例化时可覆盖：fifo #(.WIDTH(32)) my\_fifo (...);。与之不同，`define 是宏定义，全局文本替换，如 `define CLK\_PERIOD 10，在代码中用 `CLK\_PERIOD 替换为 10，适合仿真延迟设置。

赋值和运算符是 Verilog 的精髓。连续赋值 assign out = a & b; 即时计算，适合组合逻辑。过程赋值出现在 always 块中，用 = (阻塞) 或 <= (非阻塞)。阻塞赋值按序执行，如软件顺序；非阻塞模拟并行硬件，推荐用于时序逻辑。运算符丰富：逻辑 & | ~ ^ (与、或、非、异或)；算术 + - \* / %；位移 << >> (左/右移)；条件 ? :，如 y = sel ? a : b; 相当于 if-else。

实践挑战：设计一个 8 位加法器，使用 assign 实现半加器逻辑，并用 parameter 参数化位宽。

## 2 行为建模：组合逻辑设计

行为建模以 always 块描述电路行为。对于组合逻辑，用 always a(\*)，星号表示敏感所有输入变化：

```
1 module mux2to1 (input a, input b, input sel, output y);
    always a(*) begin
3     case (sel)
        1'b0: y = a;
5         1'b1: y = b;
        default: y = 1'b0;
7     endcase
    end
9 endmodule
```

这个 2 选 1 多路选择器 (MUX) 用 case 语句实现，sel 为 0 选 a，为 1 选 b。always 块在任何输入变时触发，case 确保穷尽覆盖，避免锁存器。解读时，注意 y 是 reg 类型，因过程赋值；综合后生成多路器硬件。波形中，sel 跳变瞬间 y 切换，无时钟依赖。

优先级编码器处理多输入优先级，如 4 位输入找出最高有效位：

```
1 module prio_enc (input [3:0] in, output reg [1:0] code, output reg valid);
    always a(*) begin
3     code = 2'b00; valid = 1'b0;
    casez (in)
5         4'b1???: begin code = 2'd3; valid = 1'b1; end
        4'b01??: begin code = 2'd2; valid = 1'b1; end
7         4'b001?: begin code = 2'd1; valid = 1'b1; end
        4'b0001: begin code = 2'd0; valid = 1'b1; end
```

```

9     endcase
    end
11 endmodule

```

casez 视 ? 为无关位，从高位优先匹配，设置 code 和 valid。最高位 1 时 code=3，依次类推，确保单热输出。

全加器示例展示进位逻辑：

```

1 module full_adder (input a, b, cin, output sum, cout);
    assign sum = a ^ b ^ cin;
3    assign cout = (a & b) | (a & cin) | (b & cin);
endmodule

```

异或计算和，多数投票产生进位。级联多个形成多位加法器。

状态机是组合逻辑的高级形式。Moore 机输出仅依状态，Mealy 依状态和输入。以交通灯控制器为例，4 状态：红、黄、绿、待（二进制编码 00-11）：

```

module traffic_light (input clk, rst_n, output reg [1:0] light);
2   reg [1:0] state, next_state;
   always @(*) begin // 组合下一状态
4       next_state = state;
       case (state)
6           2'b00: next_state = 2'b01; // 红-> 黄
           2'b01: next_state = 2'b10; // 黄-> 绿
8           2'b10: next_state = 2'b11; // 绿-> 待
           2'b11: next_state = 2'b00; // 待-> 红
10        endcase
       end
12 endmodule

```

这是 Mealy 简化版，实际需时钟转移（后节详述）。一热编码用 4 位，每状态一 1，提高速度但耗资源。

仿真用 testbench 验证：

```

module tb_mux;
2   reg a, b, sel;
   wire y;
4   mux2to1 dut (a, b, sel, y);
   initial begin
6       $dumpfile(``mux.vcd``); $dumpvars(0, tb_mux);
       a = 0; b = 1; sel = 0; #10 sel = 1; #10 `$\`$finish;`$
8   end
endmodule
10

```



14 时序逻辑引入时钟和复位,确保同步行为.标准模板处理异步复位:

```
16 \begin{verbatim}
module dff (input clk, rst_n, d, output reg q);
18     always @ (posedge clk or negedge rst_n) begin
        if (!rst_n) q <= 1'b0;
20         else q <= d;
        end
22 \end{verbatim}
```

24 敏感列表 `\texttt{posedge clk or negedge rst_n}` 触发于时钟上升沿或复位下降沿.优先 `\texttt{}`  
 $\hookrightarrow$  `if` 检查 `\texttt{rst_n}`,低电平清零 `\texttt{q}`,否则非阻塞赋 `\texttt{d}`.综合为 D 触发  
 $\hookrightarrow$  器(DFF),`\texttt{q}` 延迟一拍跟随 `\texttt{d}`.

26 多位移位寄存器扩展此:

```
28 \begin{verbatim}
module shift_reg #(parameter WIDTH=8) (input clk, rst_n, d, output reg [WIDTH-1:0] q
 $\hookrightarrow$  );
30     always @ (posedge clk or negedge rst_n) begin
        if (!rst_n) q <= 0;
32         else q <= {d, q[WIDTH-2:0]};
        end
34 \end{verbatim}
```

36 串入 `\texttt{d}`,高位串出,形成移位寄存器.`q[WIDTH-1:0]`);

```
    always @ (posedge clk or negedge rst_n) begin
38         if (!rst_n) q <= 0;
        else q <= (q == N-1) ? 0 : q + 1;
40     end
endmodule$reg [clog2(N)-1:0] q);
42 always @ (posedge clk or negedge rst_n) begin
    if (!rst_n) q <= 0;
44     else if (q == N-1) q <= 0;
        else q <= q + 1;
46 end
endmodule$reg [clog2(N)-1:0] q);
48 always @ (posedge clk or negedge rst_n) begin
    if (!rst_n) q <= 0;
50     else if (q == N-1) q <= 0;
```

```
    else q <= q + 1;
52 end
endmodule$reg [%clog2(N)-1:0] q);
54 always @ (posedge clk or negedge rst_n) begin
    if (!rst_n) q <= 0;
56     else if (q == N-1) q <= 0;
    else q <= q + 1;
58 end
endmodule$reg [%clog2(N)-1:0$] q);
60 always @ (posedge clk or negedge rst_n) begin
    if (!rst_n) q <= 0;
62     else if (q == N-1) q <= 0;
    else q <= q + 1;
64 end
endmodule$reg [%clog2(N)-1:0$] q);
66 always @ (posedge clk or negedge rst_n) begin
    if (!rst_n) q <= 0;
68     else if (q == N-1) q <= 0;
    else q <= q + 1;
70 end
endmodule$reg [%\clog2(N)-1:0$] q);
72 always @ (posedge clk or negedge rst_n) begin
    if (!rst_n) q <= 0;
74     else if (q == N-1) q <= 0;
    else q <= q + 1;
76 end
endmodule$reg [%\clog2(N)-1:0$] q);
78 always @ (posedge clk or negedge rst_n) begin
    if (!rst_n) q <= 0;
80     else if (q == N-1) q <= 0;
    else q <= q + 1;
82 end
endmodule$reg [%\clog2(N)-1:0$] q);
84 always @ (posedge clk or negedge rst_n) begin
    if (!rst_n) q <= 0;
86     else if (q == N-1) q <= 0;
    else q <= q + 1;
88 end
endmodule$reg [%\clog2(N)-1:0$] q);
90 always @ (posedge clk or negedge rst_n) begin
```

```
    if (!rst_n) q <= 0;
92    else if (q == N-1) q <= 0;
    else q <= q + 1;
94    end
endmodule$reg [$ \clog2(N)-1:0 $] q);
96    always @(posedge clk or negedge rst_n) begin
    if (!rst_n) q <= 0;
98    else if (q == N-1) q <= 0;
    else q <= q + 1;
100    end
endmodule$reg [$\clog2(N)-1:0$] q);
102    always @(posedge clk or negedge rst_n) begin
    if (!rst_n) q <= 0;
104    else if (q == N-1) q <= 0;
    else q <= q + 1;
106    end
endmodule$reg [$\clog2(N)-1:0$] q);
108    always @(posedge clk or negedge rst_n) begin
    if (!rst_n) q <= 0;
110    else if (q == N-1) q <= 0;
    else q <= q + 1;
112    end
endmodule$reg [\clog2(N)-1:0] q$);
114    always @(posedge clk or negedge rst_n) begin
    if (!rst_n) q <= 0;
116    else if (q == N-1) q <= 0;
    else q <= q + 1;
118    end
endmodulereg [\clog2(N)-1:0] q);
120    always @(posedge clk or negedge rst_n) begin
    if (!rst_n) q <= 0;
122    else if (q == N-1) q <= 0;
    else q <= q + 1;
124    end
endmodule$reg [\clog2(N)-1:0] q);
126    always @(posedge clk or negedge rst_n) begin
    if (!rst_n) q <= 0;
128    else if (q == N-1) q <= 0;
    else q <= q + 1;
130    end
```

```
endmodule$reg [\$clog2(N)-1:0] q);
132   always @(posedge clk or negedge rst_n) begin
       if (!rst_n) q <= 0;
134       else if (q == N-1) q <= 0;
           else q <= q + 1;
136   end
endmodule$reg [$clog2(N)-1:0] q);
138   always @(posedge clk or negedge rst_n) begin
       if (!rst_n) q <= 0;
140       else if (q == N-1) q <= 0;
           else q <= q + 1;
142   end
endmodule$reg [$clog2(N)-1:0] q);
144   always @(posedge clk or negedge rst_n) begin
       if (!rst_n) q <= 0;
146       else if (q == N-1) q <= 0;
           else q <= q + 1;
148   end
endmodule$reg [$clog2(N)-1:0] q);
150 always @(posedge clk or negedge rst_n) begin
       if (!rst_n) q <= 0;
152       else if (q == N-1) q <= 0;
           else q <= q + 1;
154 end
endmodule$reg [$clog2(N)-1:0] q);
156   always @(posedge clk or negedge rst_n) begin
       if (!rst_n) q <= 0;
158       else if (q == N-1) q <= 0;
           else q <= q + 1;
160   end
endmodule$[$clog2(N)-1:0] q);
162   always @(posedge clk or negedge rst_n) begin
       if (!rst_n) q <= 0;
164       else if (q == N-1) q <= 0;
           else q <= q + 1;
166   end
endmodule$\begin{verbatim}
168 module counter #(parameter N=16) (
       input clk, rst_n, en,
170       output reg [N-1:0] q
```

```

);
172   always @(posedge clk or negedge rst_n) begin
        if (!rst_n) q <= 0;
174   else q <= q + 1;
        end
176 endmodule

\end{verbatim}$module counter #(parameter N=16) (
178   input clk, rst_n, en,
        output reg [ $\clog2(N)-1:0$ ] q);
180   always @(posedge clk or negedge rst_n) begin
        if (!rst_n) q <= 0;
182   else if (en) q <= q + 1;
        end
184 endmodule$\begin{verbatim}
module counter #(parameter N=16) (
186   input clk, rst_n, en,
        output reg [N-1:0] q
188 );
        always @(posedge clk or negedge rst_n) begin
190   if (!rst_n) q <= 0;
        else if (en) q <= q + 1;
192 end
        endmodule
194 \end{verbatim}$\begin{verbatim}
module counter #(parameter N=16) (
196   input clk, rst_n, en,
        output reg [N-1:0] q
198 );
        always @(posedge clk or negedge rst_n) begin
200   if (!rst_n) q <= 0;
        else if (en) q <= q + 1;
202 end
        endmodule
204 \end{verbatim}$module counter #(parameter N=16) (
        input clk, rst_n, en,
206   output reg [ $\begin{verbatim}
module cnt #(parameter N=16) (
208   input clk, rst_n, en,
        output reg [N-1:0] q;
210 \end{verbatim}$module cnt #(parameter N=16) (

```

```

    input clk, rst_n, en,
212   output reg [$\begin{verbatim}
module cnt #(parameter N=16) (
214   input clk, rst_n, en,
    output reg [
216 \end{verbatim}]\verb|module cnt #(parameter N=16) (
    input clk, rst_n, en,
218   output reg [N-1:0] cnt; |$\begin{verbatim}
module counter #(parameter N=16) (
220   input clk, rst_n, en,
    output reg [N-1:0] cnt;
222 \end{verbatim}$module counter #(parameter N=16) (
    input clk, rst_n, en,
224   output reg [$module counter #(parameter N=16) (
    input clk, rst_n, en,
226   output reg [$\clog2(N)-1:0$module counter #(parameter N=16) (input clk, rst_n, en,
    output reg [$\clog2(N)-1:0$module counter #(parameter N=16) (input clk, rst_n, en,
228   output reg [$\clog2(N)-1:0$module counter #(parameter N=16) (
    input clk, rst_n, en,
230   output reg [$module cnt #(parameter N=16) (
    input clk, rst_n, en,
232   output reg [$\begin{verbatim}
module cnt #(parameter N=16) (
234   input clk, rst_n, en,
    output reg [
236 \end{verbatim}]\begin{verbatim}
module cnt #(parameter N=16) (
238   input clk, rst_n, en,
    output reg [N-1:0] cnt;
240 \end{verbatim}$module cnt #(parameter N=16) (
    input clk, rst_n, en,
242   output reg [$\begin{verbatim}
module cnt #(parameter N=16) (
244   input clk, rst_n, en,
    output reg [
246 \end{verbatim}]\begin{verbatim}
module cnt #(parameter N=16) (
248   input clk, rst_n, en,
    output reg [N-1:0] cnt;
250 \end{verbatim}]\begin{verbatim}

```

```

module cnt #(parameter N=16) (
252   input clk, rst_n, en,
       output reg [N-1:0] cnt;
254 \end{verbatim}$\begin{verbatim}
module counter #(parameter N=16) (
256   input clk, rst_n, en,
       output reg [N-1:0] cnt;
258 \end{verbatim}$\begin{verbatim}
module counter #(parameter N=16) (
260   input clk, rst_n, en,
       output reg [N-1:0] cnt;
262 \end{verbatim}$\begin{verbatim}
module cnt #(parameter N=16) (
264   input clk, rst_n, en,
       output reg [N-1:0] cnt;
266 \end{verbatim}$module counter #(parameter N=16) (
       input clk, rst_n, en,
268   output reg [$\begin{verbatim}
module cnt #(parameter N=16) (
270   input clk, rst_n, en,
       output reg [N-1:0] cnt;
272 \end{verbatim}$module cnt #(parameter N=16) (
       input clk, rst_n, en,
274   output reg [$module cnt #(parameter N=16) (
       input clk, rst_n, en,
276   output reg [$module cnt #(parameter N=16) (
       input clk, rst_n, en,
278   output reg [$module cnt #(parameter N=16) (
       input clk, rst_n, en,
280   output reg [$module counter #(parameter N=16) (
       input clk, rst_n, en,
282   output reg [$\mathrm{clog2}(N)-1:0$module counter #(parameter N=16) (input clk,
           ↪ rst_n, en,
       output reg [$\mathrm{clog2}(N)-1:0$module counter #(parameter N=16) (
284   input clk, rst_n, en,
       output reg [$\mathrm{clog2}(N)-1:0$module cnt #(parameter N=16) (input clk,
286   rst_n, en,
       output reg [$\mathrm{clog2}(N)-1:0$module counter #(parameter N=16) (input clk,
           ↪ rst_n, en,
288   output reg [$\mathrm{clog2}(N)-1:0$module cnt #(parameter N=16) (

```

```

    input clk, rst_n, en,
290   output reg [$clog2(N)-1:0]$clog2(N)-1:0$input clk, rst_n, en,
    output reg [$clog2(N)-1:0$]output reg [$clog2(N)-1:0$] q);
292 \begin{verbatim}
module counter #(parameter N=16)
294 \end{verbatim}$\begin{verbatim}
module counter #(parameter N=16) (input clk, rst_n, en, output reg [N-1:0] count);
296 \end{verbatim}$\begin{verbatim}
module cnt #(parameter N=16) (input clk, rst_n, en, output reg [N-1:0] count);
298 \end{verbatim}$reg [$clog2(N)-1:0$] q);
    always @(posedge clk or negedge rst_n)$clog2(N)-1:0] q);
300   always @(posedge clk or negedge rst_n)$[clog2(N)-1:0]$ q);
    always @(posedge clk or negedge rst_n)[$clog2(N)-1:0]$ q);
302   always @(posedge clk or negedge rst_n)[$clog2(N)-1:0$] q);
    always @(posedge clk or negedge rst_n) begin
304       if (!rst_n) q <= 0;
        else if (en)
306           q <= (q == N-1) ? 0 : q+1;
    end
308 endmodule$\begin{verbatim}
module regfile #(parameter WIDTH=32, DEPTH=32) (
310   input clk, wen,
    input [clog2(DEPTH)-1:0] rs1,
312   input clk,
\end{verbatim}$input [$clog2(DEPTH)-1:0] rs1,
314   input clk,$clog2(DEPTH)-1:0] rs1,
    input [$input clk,
316   input wen,
    input [\clog2(DEPTH)-1:0] rs1,
318   input [\clog2(DEPTH)-1:0] rs2,
    input [WIDTH-1:0] wdata,
320   input [\clog2(DEPTH)-1:0] rd,
    output [WIDTH-1:0] rdata1, rdata2
322 );
    reg [WIDTH-1:0] mem [0:DEPTH-1];
324 assign rdata1 = mem[rs1];
    assign rdata2 = mem[rs2];
326 always @(posedge clk) if (wen) mem[rd] <= wdata;
endmodule$clog2(DEPTH)-1:0] rs1,
328   input [$input clk,
```

```
    input wen,
330    input [$clog2(DEPTH)-1:0] rs1,
    input [$clog2(DEPTH)-1:0] rs2,
332    input [WIDTH-1:0] wdata,
    output [WIDTH-1:0] rdata1,
334    output [WIDTH-1:0] rdata2
);
336    reg [WIDTH-1:0] mem [0:DEPTH-1];
    always @(posedge clk) if (wen) mem[rs1] <= wdata;
338    assign rdata1 = mem[rs1];
    assign rdata2 = mem[rs2];
340 endmodule $clog2(DEPTH)-1:0 rs1,
    input [$input [$clog2(DEPTH)-1:0$] rs1,
342    input [$clog2(DEPTH)-1:0$] rs2,
    input [WIDTH-1:0] wdata,
344    output [WIDTH-1:0] rdata1, rdata2
);
346    reg [WIDTH-1:0] mem [0:DEPTH-1];
    assign rdata1 = mem[rs1];
348    assign rdata2 = mem[rs2];
endmodule $clog2(DEPTH)-1:0 rs1,
350    input [$module regfile (
    input clk, wen,
352    input [$clog2(DEPTH)-1:0] rs1,
    input [$clog2(DEPTH)-1:0] rs2,
354    input [WIDTH-1:0] wdata,
    output [WIDTH-1:0] rdata1, rdata2
356 );
    reg [WIDTH-1:0] mem [0:DEPTH-1];
358    always @(posedge clk) if (wen) mem[rs1] <= wdata;
    assign rdata1 = mem[rs1];
360    assign rdata2 = mem[rs2];
endmodule $clog2(DEPTH)-1:0 rs1,
362    input [$module regfile (
    input en,
364    input [\ $clog2(DEPTH)-1:0] rs1,
    input [\ $clog2(DEPTH)-1:0] rs2,
366    input [WIDTH-1:0] wdata,
    output [WIDTH-1:0] rdata1, rdata2
368 );
```

```
reg [WIDTH-1:0] mem [0:DEPTH-1];
370 assign rdata1 = mem[rs1];
assign rdata2 = mem[rs2];
372 endmodule$clog2(DEPTH)-1:0] rs1,
input [$module regfile #(
374 parameter WIDTH = 32,
parameter DEPTH = 32
376 ) (
input clk,
378 input wen,
input [$clog2(DEPTH)-1:0$] rs1,
380 input [$clog2(DEPTH)-1:0$] rs2,
input [WIDTH-1:0] wdata,
382 output [WIDTH-1:0] rdata1, rdata2
);
384 reg [WIDTH-1:0] mem[0:DEPTH-1];
always @(posedge clk) if (wen) mem[rs1] <= wdata;
386 assign rdata1 = mem[rs1];
assign rdata2 = mem[rs2];
388 endmodule$clog2(DEPTH)-1:0] rs1,
input [$input [\clog2(DEPTH)-1:0] rs1,
390 input [\clog2(DEPTH)-1:0] rs2,
input [WIDTH-1:0] wdata,
392 output [WIDTH-1:0] rdata1, rdata2
);
394 reg [WIDTH-1:0] mem [0:DEPTH-1];
always @(posedge clk) if (wen) mem[rs1] <= wdata;
396 assign rdata1 = mem[rs1];
assign rdata2 = mem[rs2];
398 endmodule$clog2(DEPTH)-1:0] rs1,
input [$input [\clog2(DEPTH)-1:0] rs1,
400 input [\clog2(DEPTH)-1:0] rs2,
input [WIDTH-1:0] wdata,
402 output [WIDTH-1:0] rdata1, rdata2
);
404 reg [WIDTH-1:0] mem [0:DEPTH-1];
always @(posedge clk) if (wen) mem[rs1] <= wdata;
406 assign rdata1 = mem[rs1];
assign rdata2 = mem[rs2];
408 endmodule$clog2(DEPTH)-1:0] rs1,
```

```

    input [$clog2(DEPTH)-1:0] rs2,
410   input [$module regfile #(
    parameter WIDTH = 32,
412   parameter DEPTH = 32
  )(
414   input clk,
    input wen,
416   input [\clog2(DEPTH)-1:0] rs1,
    input [\clog2(DEPTH)-1:0] rs2,
418   input [\clog2(DEPTH)-1:0] rd,
    input [WIDTH-1:0] wdata,
420   output [WIDTH-1:0] rdata1,
    output [WIDTH-1:0] rdata2
422 );
    reg [WIDTH-1:0] mem [0:DEPTH-1];
424   always @(posedge clk) begin
        if (wen) mem[rd] <= wdata;
426   end
    assign rdata1 = mem[rs1];
428   assign rdata2 = mem[rs2];
endmodule$clog2(DEPTH)-1:0] rs1,
430   input [$input clk,
    input wen,
432   input [$clog2(DEPTH)-1:0] rs1,
    input [$clog2(DEPTH)-1:0] rs2,
434   input [WIDTH-1:0] wdata,
    output [WIDTH-1:0] rdata1,
436   output [WIDTH-1:0] rdata2
  );
438   reg [WIDTH-1:0] mem[0:DEPTH-1];
    always @(posedge clk) if (wen) mem[rs1] <= wdata;
440   assign rdata1 = mem[rs1];
    assign rdata2 = mem[rs2];
442 endmodule$clog2(DEPTH)-1:0] rs1,
    input [$input [$clog2(DEPTH)-1:0] rs1,
444   input [$clog2(DEPTH)-1:0] rs2,
    assign rdata2 = mem[rs2];
446 endmodule$clog2(DEPTH)-1:0] rs1,
    input [$input [$clog2(DEPTH)-1:0] rs1,
448   input [$clog2(DEPTH)-1:0] rs2,

```

```

    output [WIDTH-1:0] rdata1, rdata2
450 );
    reg [WIDTH-1:0] mem[0:DEPTH-1];
452 always @(posedge clk) if (wen) mem[rd] <= wdata;
    assign rdata1 = mem[rs1];
454 assign rdata2 = mem[rs2];
endmodule$clog2(DEPTH)-1:0 rs1,
456 input [$\texttt{module regfile(
    input clk, wen,
458 input [\clog2(DEPTH)-1:0] rs1,
    input [\clog2(DEPTH)-1:0] rs2
460 endmodule)}$clog2(DEPTH)-1:0 rs1,
    input [$en,
462 input [$clog2(DEPTH)-1:0$] rs1,
    input [$clog2(DEPTH)-1:0$] rs2
464 );
endmodule$clog2(DEPTH)-1:0 rs1,
466 input [$input [$clog2(DEPTH)-1:0] rs1,
    input [$clog2(DEPTH)-1:0] rs2,$clog2(DEPTH)-1:0] rs1,
468 input [$input [\clog2(DEPTH)-1:0] rs1,
    input [\clog2(DEPTH)-1:0] rs2,
470 output [WIDTH-1:0] rdata1, rdata2
);
472 reg [WIDTH-1:0] mem [0:DEPTH-1];
    always @(posedge clk) if (wen) mem[rd] <= wdata;
474 assign rdata1 = mem[rs1];
    assign rdata2 = mem[rs2];
476 endmodule$clog2(DEPTH)-1:0 rs1,
    input [$module regfile(
478 input en,
    input [$clog2(DEPTH)-1:0] rs1,
480 input [$clog2(DEPTH)-1:0] rs2,$clog2(DEPTH)-1:0] rs1,
    input [$module regfile(
482 input en,
    input [$clog2(DEPTH)-1:0] rs2,
484 input [$clog2(DEPTH)-1:0] rs1,$clog2(DEPTH)-1:0] rs1,
    input [$module regfile(
486 input en,
    input [$clog2(DEPTH)-1:0$] rs1,
488 input [$clog2(DEPTH)-1:0$] rs2,

```

```

    output [WIDTH-1:0] rdata1, rdata2
490 );
    reg [WIDTH-1:0] mem [0:DEPTH-1];
492 assign rdata1 = mem[rs1];
    assign rdata2 = mem[rs2];
494 endmodule$clog2(DEPTH)-1:0] rs1,
    input [$module regfile(
496     input [$clog2(DEPTH)-1:0] rs1,
    input [$clog2(DEPTH)-1:0] rs2,
498     input [WIDTH-1:0] wdata,
    output [WIDTH-1:0] rdata1, rdata2
500 );
    reg [WIDTH-1:0] mem [0:DEPTH-1];
502 assign rdata1 = mem[rs1];
    assign rdata2 = mem[rs2];
504 endmodule$clog2(DEPTH)-1:0] rs1,
    input [$module regfile(
506     input clk, wen,
    input [$clog2(DEPTH)-1:0] rs1,
508     input [$clog2(DEPTH)-1:0] rs2,
    input [WIDTH-1:0] wdata,
510     output [WIDTH-1:0] rdata1, rdata2
    );
512 reg [WIDTH-1:0] mem[0:DEPTH-1];
    always @(posedge clk) if (wen) mem[rs1] <= wdata;
514 assign rdata1 = mem[rs1];
    assign rdata2 = mem[rs2];
516 endmodule$clog2(DEPTH)-1:0] rs1,
    input [$input clk, wen,
518     input [$clog2(DEPTH)-1:0] rs1,
    input [$clog2(DEPTH)-1:0] rs2,
520     input [WIDTH-1:0] wdata,
    output [WIDTH-1:0] rdata1, rdata2
522 );
    reg [WIDTH-1:0] mem [0:DEPTH-1];
524 assign rdata1 = mem[rs1];
    assign rdata2 = mem[rs2];
526 endmodule$clog2(DEPTH)-1:0$(
    input clk, wen,
528     input [$clog2(DEPTH)-1:0$] rs1, rs2, rd,

```

```

    input
530 assign rdata2 = mem[rs2];
endmodule$input [$clog2(DEPTH)-1:0$] rs1, rs2, rd,
532   input [WIDTH-1:0] wdata$input [$input [$clog2(DEPTH)-1:0$] rs1, rs2, rd,
    input [$WIDTH-1:0$] wdata,
534   output [$WIDTH-1:0$]input [$WIDTH-1:0$] wdata,
    output [$WIDTH-1:0$]input [\'$WIDTH-1:0] wdata,
536   output [\'$WIDTH-1:0] rdata1, rdata2
);
538 // \dots
endmodule$使用 \texttt{\$display(``cnt=\%d'\'', cnt)} 打印、\texttt{\$monitor} 持续监视变
    ↪ 化。$monitor} 持续监视变化。monitor} 持续监视变化。monitor}\'\' 持续监视变化。monitor
    ↪ \'\' 持续监视变化。monitor\'\' 持续监视变化。\'\' 持续监视变化。
540
综合优化避免锁存器 monitor` 持续监视变化。 持续监视变化。monitor` 持续监视变化。
542
综合优化避免锁存器: if 必配 else, 如:
544
```verilog
546 always a(*) begin
    if (en) out = in;
548     else out = 0; // 防 latch
end

```

时序用 `<=`, 资源复用流水线寄存数据。

常见陷阱: 仿真-综合不匹配因 real, 用定点如 `[31:0] fixed = val << 16;`。竞争冒险加寄存器缓冲。多个驱动 wire 错误, 用 tri 或单 assign。

实践挑战: 写 task 生成正弦波激励, monitor 输出观察。

### 3 工具链与实践项目

开发环境首选 Vivado (Xilinx FPGA 免费版, 支持综合仿真)、Quartus (Intel)、ModelSim 学生版 (精确仿真)、Icarus Verilog (开源命令行)。

完整项目: 4 位 RISC 处理器。顶层集成 ALU、RegFile、Controller、PC、IM (指令存储)、DM (数据存储)。架构单周期, 每指令一拍。testbench 加载指令序列, 如 ADD R1,R2,R3; 仿真见寄存器更新。FPGA 部署: 写 XDC 约束引脚, 时序报告, 生成 bitstream 下载板卡。

资源推荐书籍《Verilog 数字系统设计》、在线 HDLBits 练习、GitHub RISC-V 项目。

实践挑战: 用 Vivado 实现处理器, 跑斐波那契程序。

## 4 结论与进阶路径

本文从语法到 FSM、模块化，勾勒 Verilog 学习曲线：先组合再时序，实践驱动。进阶 SystemVerilog UVM 验证、VLSI 后端 STA 布局。探索 RISC-V SoC 或 ASIC 流片。

行动号召：下载 Vivado，码第一个 MUX，加入社区交流。

附录 **A**：GitHub 仓库（虚构链接：[github.com/verilog-tutorial](https://github.com/verilog-tutorial)）。

附录 **B**：术语：wire（连线）、reg（过程变量）、RTL（寄存器传输级）、FSM（状态机）。

附录 **C**：IEEE 1364、Thomas & Moor 《数字设计》。