c13n #73

c13n

2026 年 5 月 19 日

第 I 部

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：

```
module and_gate (input a, input b, output y);
   assign y = a & b;
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：

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

```
   // 使用 WIDTH 和 DEPTH
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 @(*)`，星号表示敏感所有输入变化：

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

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

```
    end
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：

```
module full_adder (input a, b, cin, output sum, cout);
    assign sum = a ^ b ^ cin;
    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);
    reg [1:0] state, next_state;
    always @(*) begin // 组合下一状态
        next_state = state;
        case (state)
            2'b00: next_state = 2'b01; // 红-> 黄
            2'b01: next_state = 2'b10; // 黄-> 绿
            2'b10: next_state = 2'b11; // 绿-> 待
            2'b11: next_state = 2'b00; // 待-> 红
        endcase
    end
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;
    reg a, b, sel;
    wire y;
    mux2to1 dut (a, b, sel, y);
    initial begin
        $dumpfile(``mux.vcd''); $dumpvars(0, tb_mux);
        a = 0; b = 1; sel = 0; #10 sel = 1; #10 $`\$finish;`$
    end
endmodule

`initial` 块生成激励，`$dumpfile`和`$dumpvars` 导出 VCD 文件，用 `\
    ↪ $dumpfile` 和 `\$dumpvars`dumpfile` 和 `$`\$dumpvars`$导出 VCD 文
```

```
    ↪ 件，用$`\$dumpfile`$和$`\$dumpvars`$`initial` 块生成激励，`\
    ↪ $dumpvars` 导出 VCD 文件，用 GTKWave 查看波形：时间轴上见 sel
使用 `\$dumpfile` 和 `\$dumpvars` 导出 VCD 文件$dumpfile` 和 `$dumpvars`
    ↪ 导出 VCD 文件 dumpfile` 和 `$\text{使用 }\texttt{\$dumpfile}\text{
    ↪  和 }\texttt{\$dumpvars}\text{ 导出 VCD 文件，用 GTKWave 查看波形：
    ↪  时间轴上见}$sel$tt{\$dumpvars}$ 导出 VCD 文件，用 GTKWave 查看波
    ↪ 形：时间轴上见 $sel$ 翻转， $y$ 从 $a$ 切换到 $b$sel$sel$ 翻转，
    ↪ $y$ 从 $a$ 切换到 $b$y$ 从 $a$ 切换到 $b$sel$ 翻转，$y$ 从 $a$ 切
    ↪ 换到 $b$y$ 从 $a$ 切换到 $b$$y$ 从 $a$ 切换到 $b$$y$ 从 $a$ 切换到
    ↪  $b$$$y$ 从 $a$ 切换到 $b$$y$ 从 $a$ 切换到 $b$$$y$ 从 $a$ 切换到
    ↪ $b$$y$ 从 $a$ 切换到 $b$$$y$ 从 $a$ 切换到 $b$$y$ 从 $a$ 切换到
    ↪ $b$$$y$ 从 $a$ 切换到 $b$$y$ 从 $a$ 切换到 $b$$$y$ 从 $a$ 切换到
    ↪ $b$$y$ 从 $a$ 切换到 $b$$$y$ 从 $a$ 切换到 $b$$y$ 从 $a$ 切换到
    ↪ $b$$$y$ 从 $a$ 切换到 $b$$y$ 从 $a$ 切换到 $b$$$y$ 从 $a$ 切换到
    ↪ $b$$y$ 从 $a$ 切换到 $b$$$y$ 从 $a$ 切换到 $b$$y$ 从 $a$ 切换到
    ↪ $b$$$y$ 从 $a$ 切换到 $b$$y$ 从 $a$ 切换到 $b$$$y$ 从 $a$ 切换到
    ↪ $b$$y$ 从 $a$ 切换到 $b$$$y$ 从 $a$ 切换到 $b$$y$ 从 $a$ 切换到
    ↪ $b$$$y$ 从 $a$ 切换到 $b$$y$ 从 $a$ 切换到 $b$$$y$ 从 $a$ 切换到
    ↪ $b$$y$ 从 $a$ 切换到 $b$$$y$ 从 $a$ 切换到 $b$$y$ 从 $a$ 切换到
    ↪ $b$$$y$ 从 $a$ 切换到 $b$$y$ 从 $a$ 切换到 $b$$$y$ 从 $a$ 切换到
    ↪ $b$$y$ 从 $a$ 切换到 $b$$$y$ 从 $a$ 切换到 $b$$y$ 从 $a$ 切换到
    ↪ $b$$$y$ 从 $a$ 切换到 $b$$y$ 从 $a$ 切换到 $b$$$y$ 从 $a$ 切换到
    ↪ $b$$y$ 从 $a$ 切换到 $b$$$y$ 从 $a$ 切换到 $b$$y$ 从 $a$ 切换到
    ↪ $b$$$y$ 从 $a$ 切换到 $b$$y$ 从 $a$ 切换到 $b$$$y$ 从 $a$ 切换到
    ↪ $b$$y$ 从 $a$ 切换到 $b$$$y$ 从 $a$ 切换到 $b$$y$ 从 $a$ 切换到
    ↪ $b$$$y$ 从 $a$ 切换到 $b$$y$ 从 $a$ 切换到 $b$$$y$ 从 $a$ 切换到
    ↪ $b$$y$ 从 $a$ 切换到 $b$$$y$ 从 $a$ 切换到 $b$$y$ 从 $a$ 切换到
    ↪ $b$$$y$ 从 $a$ 切换到 $b$$y$ 从 $a$ 切换到 $b$$$y$ 从 $a$ 切换到
    ↪ $b$$y$ 从 $a$ 切换到 $b$$$y$ 从 $a$ 切换到 $b$$y$ 从 $a$ 切换到
    ↪ $b$$$y$ 从 $a$ 切换到 $b$$y$ 从 $a$ 切换到 $b$$$y$ 从 $a$ 切换到
    ↪ $b$$y$ 从 $a$ 切换到 $b$$$y$ 从 $a$ 切换到 $b$$y$ 从 $a$ 切换到
    ↪ $b$$$y$ 从 $a$ 切换到 $b$$y$ 从 $a$ 切换到 $b$$$y$ 从 $a$ 切换到
    ↪ $b$$y$ 从 $a$ 切换到 $b$$$y$ 从 $a$ 切换到 $b$$y$ 从 $a$ 切换到
    ↪ $b$$$y$ 从 $a$ 切换到 $b$$y$ 从 $a$ 切换到 $b$$$y$ 从 $a$ 切换到
    ↪ $b$$y$ 从 $a$ 切换到 $b$$$y$ 从 $a$ 切换到 $b$$y$ 从 $a$ 切换到
    ↪ $b$$$y$ 从 $a$ 切换到 $b$$y$ 从 $a$ 切换到 $b$$$y$ 从 $a$ 切换到
    ↪ $b$$y$ 从 $a$ 切换到 $b$$$y$ 从 $a$ 切换到 $b$$y$ 从 $a$ 切换到
    ↪ $b$$$y$ 从 $a$ 切换到 $b$$y$ 从 $a$ 切换到 $b$$$y$ 从 $a$ 切换到
    ↪ $b$$y$ 从 $a$ 切换到 $b$$$y$ 从 $a$ 切换到 $b$$y$ 从 $a$ 切换到
    ↪ $b$$$y$ 从 $a$ 切换到 $b$$y$ 从 $a$ 切换到 $b$$$y$ 从 $a$ 切换到
```

```
    ↪ $b$$y$ 从 $a$ 切换到 $b$$$y$$ 从 $$a$$ 切换到 $$b$$\text{\
    ↪ textbackslash{}textbf{实践挑战}: 实现 4 选 1 MUX, 用 \texttt{
    ↪ casez} 优化 , 并写 testbench 验证所有组合 .}$$\section{时序逻辑设计
    ↪ }

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:

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

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{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 触发器(DFF),\texttt{q} 延迟一拍跟随 \
    ↪ texttt{d}.

多位移位寄存器扩展此:

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

串入 \texttt{d},高位串出,形成移位寄存器.$$-1:0] q);
   always @(posedge clk or negedge rst_n) begin
      if (!rst_n) q <= 0;
      else q <= (q == N-1) ? 0 : q + 1;
   end
endmodule$reg [$clog2(N)$-1:0] q);
always @(posedge clk or negedge rst_n) begin
   if (!rst_n) q <= 0;
   else if (q == N-1) q <= 0;
   else q <= q + 1;
end
```

```
endmodule$reg [\($clog2(N)\)-1:0] q);
always @(posedge clk or negedge rst_n) begin
   if (!rst_n) q <= 0;
   else if (q == N-1) q <= 0;
   else q <= q + 1;
end
endmodule$reg [$clog2(N)-1:0] q);
   always @(posedge clk or negedge rst_n) begin
      if (!rst_n) q <= 0;
      else if (q == N-1) q <= 0;
      else q <= q + 1;
   end
endmodule$reg [$clog2(N)-1:0$] q);
always @(posedge clk or negedge rst_n) begin
   if (!rst_n) q <= 0;
   else if (q == N-1) q <= 0;
   else q <= q + 1;
end
endmodule$reg [$clog2(N)-1:0$] q);
   always @(posedge clk or negedge rst_n) begin
      if (!rst_n) q <= 0;
      else if (q == N-1) q <= 0;
      else q <= q + 1;
   end
endmodule$reg [$\clog2(N)-1:0$] q);
   always @(posedge clk or negedge rst_n) begin
      if (!rst_n) q <= 0;
      else if (q == N-1) q <= 0;
      else q <= q + 1;
   end
endmodule$reg [$\clog2(N)-1:0$] q);
   always @(posedge clk or negedge rst_n) begin
      if (!rst_n) q <= 0;
      else if (q == N-1) q <= 0;
      else q <= q + 1;
   end
endmodule$reg [$\clog2(N)-1:0$] q);
   always @(posedge clk or negedge rst_n) begin
      if (!rst_n) q <= 0;
      else if (q == N-1) q <= 0;
      else q <= q + 1;
   end
```

```
endmodule$reg [$\clog2(N)-1:0$] q);
   always @(posedge clk or negedge rst_n) begin
      if (!rst_n) q <= 0;
      else if (q == N-1) q <= 0;
      else q <= q + 1;
   end
endmodule$reg [$ \clog2(N)-1:0 $] q);
   always @(posedge clk or negedge rst_n) begin
      if (!rst_n) q <= 0;
      else if (q == N-1) q <= 0;
      else q <= q + 1;
   end
endmodule$reg [$\clog2(N)-1:0$] q);
   always @(posedge clk or negedge rst_n) begin
      if (!rst_n) q <= 0;
      else if (q == N-1) q <= 0;
      else q <= q + 1;
   end
endmodule$reg [$\clog2(N)-1:0$] q);
   always @(posedge clk or negedge rst_n) begin
      if (!rst_n) q <= 0;
      else if (q == N-1) q <= 0;
      else q <= q + 1;
   end
endmodule$reg [\$clog2(N)-1:0] q$);
   always @(posedge clk or negedge rst_n) begin
      if (!rst_n) q <= 0;
      else if (q == N-1) q <= 0;
      else q <= q + 1;
   end
endmodulereg [\$clog2(N)-1:0] q);
   always @(posedge clk or negedge rst_n) begin
      if (!rst_n) q <= 0;
      else if (q == N-1) q <= 0;
      else q <= q + 1;
   end
endmodule$reg [\$clog2(N)-1:0] q);
   always @(posedge clk or negedge rst_n) begin
      if (!rst_n) q <= 0;
      else if (q == N-1) q <= 0;
      else q <= q + 1;
   end
```

```
endmodule$reg [\$clog2(N)-1:0] q);
   always @(posedge clk or negedge rst_n) begin
      if (!rst_n) q <= 0;
      else if (q == N-1) q <= 0;
      else q <= q + 1;
   end
endmodule$reg [$clog2(N)-1:0] q);
   always @(posedge clk or negedge rst_n) begin
      if (!rst_n) q <= 0;
      else if (q == N-1) q <= 0;
      else q <= q + 1;
   end
endmodule$reg [$clog2(N)-1:0] q);
   always @(posedge clk or negedge rst_n) begin
      if (!rst_n) q <= 0;
      else if (q == N-1) q <= 0;
      else q <= q + 1;
   end
endmodule$reg [$clog2(N)-1:0] q);
always @(posedge clk or negedge rst_n) begin
   if (!rst_n) q <= 0;
   else if (q == N-1) q <= 0;
   else q <= q + 1;
end
endmodule$reg [$clog2(N)-1:0] q);
   always @(posedge clk or negedge rst_n) begin
      if (!rst_n) q <= 0;
      else if (q == N-1) q <= 0;
      else q <= q + 1;
   end
endmodule$[$clog2(N)-1:0] q);
   always @(posedge clk or negedge rst_n) begin
      if (!rst_n) q <= 0;
      else if (q == N-1) q <= 0;
      else q <= q + 1;
   end
endmodule$\begin{verbatim}
module counter #(parameter N=16) (
   input clk, rst_n, en,
   output reg [N-1:0] q
);
   always @(posedge clk or negedge rst_n) begin
```

```
      if (!rst_n) q <= 0;
      else q <= q + 1;
   end
endmodule
\end{verbatim}$module counter #(parameter N=16) (
   input clk, rst_n, en,
   output reg [$\clog2(N)-1:0$] q);
   always @(posedge clk or negedge rst_n) begin
   if (!rst_n) q <= 0;
   else if (en) q <= q + 1;
   end
endmodule$\begin{verbatim}
module counter #(parameter N=16) (
   input clk, rst_n, en,
   output reg [N-1:0] q
);
always @(posedge clk or negedge rst_n) begin
   if (!rst_n) q <= 0;
   else if (en) q <= q + 1;
end
endmodule
\end{verbatim}$\begin{verbatim}
module counter #(parameter N=16) (
   input clk, rst_n, en,
   output reg [N-1:0] q
);
always @(posedge clk or negedge rst_n) begin
   if (!rst_n) q <= 0;
   else if (en) q <= q + 1;
end
endmodule
\end{verbatim}$module counter #(parameter N=16) (
   input clk, rst_n, en,
   output reg [$\begin{verbatim}
module cnt #(parameter N=16) (
   input clk, rst_n, en,
   output reg [N-1:0] q;
\end{verbatim}$module cnt #(parameter N=16) (
   input clk, rst_n, en,
   output reg [$\begin{verbatim}
module cnt #(parameter N=16) (
   input clk, rst_n, en,
```

```
   output reg [
\end{verbatim}$\verb|module cnt #(parameter N=16) (
   input clk, rst_n, en,
   output reg [N-1:0] cnt; |$\begin{verbatim}
module counter #(parameter N=16) (
   input clk, rst_n, en,
   output reg [N-1:0] cnt;
\end{verbatim}$module counter #(parameter N=16) (
   input clk, rst_n, en,
   output reg [$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,
   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,
   output reg [$module cnt #(parameter N=16) (
   input clk, rst_n, en,
   output reg [$\begin{verbatim}
module cnt #(parameter N=16) (
   input clk, rst_n, en,
   output reg [
\end{verbatim}$\begin{verbatim}
module cnt #(parameter N=16) (
   input clk, rst_n, en,
   output reg [N-1:0] cnt;
\end{verbatim}$module cnt #(parameter N=16) (
   input clk, rst_n, en,
   output reg [$\begin{verbatim}
module cnt #(parameter N=16) (
   input clk, rst_n, en,
   output reg [
\end{verbatim}$\begin{verbatim}
module cnt #(parameter N=16) (
   input clk, rst_n, en,
   output reg [N-1:0] cnt;
\end{verbatim}$\begin{verbatim}
module cnt #(parameter N=16) (
   input clk, rst_n, en,
   output reg [N-1:0] cnt;
\end{verbatim}$\begin{verbatim}
```

```
module counter #(parameter N=16) (
   input clk, rst_n, en,
   output reg [N-1:0] cnt;
\end{verbatim}$\begin{verbatim}
module counter #(parameter N=16) (
   input clk, rst_n, en,
   output reg [N-1:0] cnt;
\end{verbatim}$\begin{verbatim}
module cnt #(parameter N=16) (
   input clk, rst_n, en,
   output reg [N-1:0] cnt;
\end{verbatim}$module counter #(parameter N=16) (
   input clk, rst_n, en,
   output reg [$\begin{verbatim}
module cnt #(parameter N=16) (
   input clk, rst_n, en,
   output reg [N-1:0] cnt;
\end{verbatim}$module cnt #(parameter N=16) (
   input clk, rst_n, en,
   output reg [$module cnt #(parameter N=16) (
   input clk, rst_n, en,
   output reg [$module cnt #(parameter N=16) (
   input clk, rst_n, en,
   output reg [$module cnt #(parameter N=16) (
   input clk, rst_n, en,
   output reg [$module counter #(parameter N=16) (
   input clk, rst_n, en,
   output reg [$\mathrm{clog2}(N)-1:0$module counter #(parameter N
       ↪ =16) (input clk, rst_n, en,
   output reg [$\mathrm{clog2}(N)-1:0$module counter #(parameter N
       ↪ =16) (
   input clk, rst_n, en,
   output reg [$\mathrm{clog2}(N)-1:0$module cnt #(parameter N=16) (
       ↪ input clk,
 rst_n, en,
   output reg [$\mathrm{clog2}(N)-1:0$module counter #(parameter N
       ↪ =16) (input clk, rst_n, en,
   output reg [$\mathrm{clog2}(N)-1:0$module cnt #(parameter N=16) (
   input clk, rst_n, en,
   output reg [$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);
\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,
   output [WIDTH-1:0] rdata2
);
   reg [WIDTH-1:0] mem [0:DEPTH-1];
   always @(posedge clk) if (wen) mem[rs1] <= wdata;
   assign rdata1 = mem[rs1];
   assign rdata2 = mem[rs2];
endmodule$clog2(DEPTH)-1:0] rs1,
   input [$input [$clog2(DEPTH)-1:0$] rs1,
   input [$clog2(DEPTH)-1:0$] rs2,
   input [WIDTH-1:0] wdata,
   output [WIDTH-1:0] rdata1, rdata2
);
   reg [WIDTH-1:0] mem [0:DEPTH-1];
   assign rdata1 = mem[rs1];
   assign rdata2 = mem[rs2];
endmodule$clog2(DEPTH)-1:0] rs1,
   input [$module regfile (
   input clk, wen,
   input [$clog2(DEPTH)-1:0] rs1,
   input [$clog2(DEPTH)-1:0] rs2,
   input [WIDTH-1:0] wdata,
   output [WIDTH-1:0] rdata1, rdata2
);
   reg [WIDTH-1:0] mem [0:DEPTH-1];
   always @(posedge clk) if (wen) mem[rs1] <= wdata;
   assign rdata1 = mem[rs1];
   assign rdata2 = mem[rs2];
endmodule$clog2(DEPTH)-1:0] rs1,
   input [$module regfile (
   input en,
   input [\$clog2(DEPTH)-1:0] rs1,
   input [\$clog2(DEPTH)-1:0] rs2,
   input [WIDTH-1:0] wdata,
   output [WIDTH-1:0] rdata1, rdata2
);
reg [WIDTH-1:0] mem [0:DEPTH-1];
assign rdata1 = mem[rs1];
assign rdata2 = mem[rs2];
endmodule$clog2(DEPTH)-1:0] rs1,
   input [$module regfile #(
   parameter WIDTH = 32,
```

```
    parameter DEPTH = 32
) (
    input clk,
    input wen,
    input [$clog2(DEPTH)-1:0$] rs1,
    input [$clog2(DEPTH)-1:0$] rs2,
    input [WIDTH-1:0] wdata,
    output [WIDTH-1:0] rdata1, rdata2
);
    reg [WIDTH-1:0] mem[0:DEPTH-1];
    always @(posedge clk) if (wen) mem[rs1] <= wdata;
    assign rdata1 = mem[rs1];
    assign rdata2 = mem[rs2];
endmodule$clog2(DEPTH)-1:0] rs1,
    input [$input [\$clog2(DEPTH)-1:0] rs1,
    input [\$clog2(DEPTH)-1:0] rs2,
    input [WIDTH-1:0] wdata,
    output [WIDTH-1:0] rdata1, rdata2
);
    reg [WIDTH-1:0] mem [0:DEPTH-1];
    always @(posedge clk) if (wen) mem[rs1] <= wdata;
    assign rdata1 = mem[rs1];
    assign rdata2 = mem[rs2];
endmodule$clog2(DEPTH)-1:0] rs1,
    input [$input [\$clog2(DEPTH)-1:0] rs1,
    input [\$clog2(DEPTH)-1:0] rs2,
    input [WIDTH-1:0] wdata,
    output [WIDTH-1:0] rdata1, rdata2
);
reg [WIDTH-1:0] mem [0:DEPTH-1];
always @(posedge clk) if (wen) mem[rs1] <= wdata;
assign rdata1 = mem[rs1];
assign rdata2 = mem[rs2];
endmodule$clog2(DEPTH)-1:0] rs1,
    input [$clog2(DEPTH)-1:0] rs2,
    input [$module regfile #(
    parameter WIDTH = 32,
    parameter DEPTH = 32
)(
    input clk,
    input wen,
    input [\$clog2(DEPTH)-1:0] rs1,
```

```
   input [\$clog2(DEPTH)-1:0] rs2,
   input [\$clog2(DEPTH)-1:0] rd,
   input [WIDTH-1:0] wdata,
   output [WIDTH-1:0] rdata1,
   output [WIDTH-1:0] rdata2
);
   reg [WIDTH-1:0] mem [0:DEPTH-1];
   always @(posedge clk) begin
      if (wen) mem[rd] <= wdata;
   end
   assign rdata1 = mem[rs1];
   assign rdata2 = mem[rs2];
endmodule$clog2(DEPTH)-1:0] rs1,
   input [$input clk,
   input wen,
   input [$clog2(DEPTH)-1:0] rs1,
   input [$clog2(DEPTH)-1:0] rs2,
   input [WIDTH-1:0] wdata,
   output [WIDTH-1:0] rdata1,
   output [WIDTH-1:0] rdata2
);
   reg [WIDTH-1:0] mem[0:DEPTH-1];
   always @(posedge clk) if (wen) mem[rs1] <= wdata;
   assign rdata1 = mem[rs1];
   assign rdata2 = mem[rs2];
endmodule$clog2(DEPTH)-1:0] rs1,
   input [$input [$clog2(DEPTH)-1:0] rs1,
   input [$clog2(DEPTH)-1:0] rs2,
assign rdata2 = mem[rs2];
endmodule$clog2(DEPTH)-1:0] rs1,
   input [$input [$clog2(DEPTH)-1:0] rs1,
   input [$clog2(DEPTH)-1:0] rs2,
   output [WIDTH-1:0] rdata1, rdata2
);
   reg [WIDTH-1:0] mem[0:DEPTH-1];
   always @(posedge clk) if (wen) mem[rd] <= wdata;
   assign rdata1 = mem[rs1];
   assign rdata2 = mem[rs2];
endmodule$clog2(DEPTH)-1:0] rs1,
   input [$\texttt{module regfile(
   input clk, wen,
   input [\$clog2(DEPTH)-1:0] rs1,
```

```
   input [\$clog2(DEPTH)-1:0] rs2
endmodule)}$clog2(DEPTH)-1:0] rs1,
   input [$en,
   input [$clog2(DEPTH)-1:0$] rs1,
   input [$clog2(DEPTH)-1:0$] rs2
);
endmodule$clog2(DEPTH)-1:0] rs1,
   input [$input [$clog2(DEPTH)-1:0] rs1,
   input [$clog2(DEPTH)-1:0] rs2,$clog2(DEPTH)-1:0] rs1,
   input [$input [\$clog2(DEPTH)-1:0] rs1,
   input [\$clog2(DEPTH)-1:0] rs2,
   output [WIDTH-1:0] rdata1, rdata2
);
   reg [WIDTH-1:0] mem [0:DEPTH-1];
   always @(posedge clk) if (wen) mem[rd] <= wdata;
   assign rdata1 = mem[rs1];
   assign rdata2 = mem[rs2];
endmodule$clog2(DEPTH)-1:0] rs1,
   input [$module regfile(
   input en,
   input [$clog2(DEPTH)-1:0] rs1,
   input [$clog2(DEPTH)-1:0] rs2,$clog2(DEPTH)-1:0] rs1,
   input [$module regfile(
   input en,
   input [$clog2(DEPTH)-1:0] rs2,
   input [$clog2(DEPTH)-1:0] rs1,$clog2(DEPTH)-1:0] rs1,
   input [$module regfile(
   input en,
   input [$clog2(DEPTH)-1:0$] rs1,
   input [$clog2(DEPTH)-1:0$] rs2,
   output [WIDTH-1:0] rdata1, rdata2
);
reg [WIDTH-1:0] mem [0:DEPTH-1];
assign rdata1 = mem[rs1];
assign rdata2 = mem[rs2];
endmodule$clog2(DEPTH)-1:0] rs1,
   input [$module regfile(
   input [$clog2(DEPTH)-1:0] rs1,
   input [$clog2(DEPTH)-1:0] rs2,
   input [WIDTH-1:0] wdata,
   output [WIDTH-1:0] rdata1, rdata2
);
```

```
   reg [WIDTH-1:0] mem [0:DEPTH-1];
   assign rdata1 = mem[rs1];
   assign rdata2 = mem[rs2];
endmodule$clog2(DEPTH)-1:0] rs1,
   input [$module regfile(
   input clk, wen,
   input [$clog2(DEPTH)-1:0] rs1,
   input [$clog2(DEPTH)-1:0] rs2,
   input [WIDTH-1:0] wdata,
   output [WIDTH-1:0] rdata1, rdata2
);
reg [WIDTH-1:0] mem[0:DEPTH-1];
always @(posedge clk) if (wen) mem[rs1] <= wdata;
assign rdata1 = mem[rs1];
assign rdata2 = mem[rs2];
endmodule$clog2(DEPTH)-1:0] rs1,
   input [$input clk, wen,
   input [$clog2(DEPTH)-1:0] rs1,
   input [$clog2(DEPTH)-1:0] rs2,
   input [WIDTH-1:0] wdata,
   output [WIDTH-1:0] rdata1, rdata2
);
   reg [WIDTH-1:0] mem [0:DEPTH-1];
   assign rdata1 = mem[rs1];
   assign rdata2 = mem[rs2];
endmodule$clog2(DEPTH)-1:0$(
   input clk, wen,
   input [$clog2(DEPTH)-1:0$] rs1, rs2, rd,
   input
assign rdata2 = mem[rs2];
endmodule$input [$clog2(DEPTH)-1:0$] rs1, rs2, rd,
   input [WIDTH-1:0] wdata$input [$input [$clog2(DEPTH)-1:0$] rs1,
       ↪ rs2, rd,
   input [$WIDTH-1:0$] wdata,
   output [$WIDTH-1:0$input [$WIDTH-1:0$] wdata,
   output [$WIDTH-1:0$input [\$WIDTH-1:0] wdata,
   output [\$WIDTH-1:0] rdata1, rdata2
);
   // \dots
endmodule$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 @(*) 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）。
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》。
```

第 II 部

嵌入式高性能数组计算语言的设计与实践

杨岢瑞

May 16, 2026

5　研究背景与核心挑战

在边缘计算、自动驾驶与工业控制领域，系统必须在极低功耗和极小存储下完成高吞吐的数值运算。传统 C/C++ 虽然能榨取硬件性能，却在内存管理、安全性和并行表达上留下了大量手工负担。开发者需要在「每毫秒都需确定」和「每字节都需珍惜」之间反复权衡，这正是本文要解决的根本矛盾。

6　现有方案的缺口

Halide 与 TVM 等高性能数组 DSL 已把算子融合与多面体优化带入主流，但它们默认的运行时仍依赖堆分配与动态调度，在典型 MCU（RAM 小于 64 KiB）上无法落地。TinyML 生态虽然提供了量化推理框架，却把算子调度固化在预编译库里，难以让用户对 DMA、SIMD 和存储层次做细粒度控制。因此，一种「原生面向 MCU 存储层次与实时约束」的张量 DSL 仍属空白。

7　设计目标与约束

我们希望语言同时满足三条硬性指标：RAM/ROM 占用小于 64 KiB，端到端硬实时响应小于 1 ms，以及 MISRA-C 兼容的安全内存模型。功能上要求零拷贝、零堆分配、编译期确定内存布局，并支持 N 维稠密/稀疏张量、结构化数组视图与细粒度 SIMD/多核/异构调度。设计哲学可概括为三句：一切皆表达式、一切皆可静态分析、一切皆可离线优化。

8　类型系统

语言采用依赖类型与线性类型结合的方案。依赖类型让 Shape 在类型层即可表达，例如 Array 〈 f32, [N, M]〉，Rank 多态则允许同一函数处理任意维度的视图。线性类型禁止隐式拷贝，所有权必须显式 move 或 borrow，从而在编译期即可消除悬垂指针与数据竞争。内存区域类型进一步把 SRAM、TCM、Flash、DMA 描述符纳入类型系统，使调度器能根据访问模式自动选择最优存储层次。

9　数组抽象与零拷贝视图

核心抽象 Array 〈 T, Shape, Layout, Location 〉把元素类型、形状、布局与物理位置解耦。视图（View）通过引用计数与偏移量实现零拷贝切片：

```
let mut v: View<f32, [N, M], RowMajor, SRAM> = arr.view();
let sub = v.slice([0..32, 16..48]); // 仅修改偏移与步长，无堆分配
```

广播、变形、转置在编译期展开为静态索引表达式，避免运行时分支。Layout 参数可取 RowMajor、ColMajor 或自定义 Stride 描述符，Location 参数则绑定到具体的物理存储。

10 计算原语与融合机制

内置 map、zip、reduce、scan、conv、gemm 等算子级原语，并通过「融合规则」把相邻算子合并为单一循环嵌套。开发者可用注解控制并行策略：

```
#parallel(4) #vector(128) #dma(src, dst)
let out = gemm(a, b).map(|x| relu(x));
```

融合后的循环体会被多面体建模，自动进行 tiling 与软件流水，从而最大化寄存器与缓存复用。

11 内存与 I/O

所有张量默认静态分配在栈或全局段，运行时只维护轻量描述符。DMA 操作通过专用描述符 DSL 表达：

```
dma_xfer(src: &Array<u8, [1024], _, DMA0>,
       dst: &mut Array<u8, [1024], _, SRAM>,
       mode: DoubleBuffer);
```

零拷贝 IPC 利用核间共享内存与线性类型，所有权在编译期即完成转移，避免运行时锁。

12 安全与实时扩展

语言内置合约（pre/post-condition）与 WCET 标注，静态分析器可据此计算最坏执行时间。异常被完全禁用，错误统一 panic 并执行轻量回滚，满足 MISRA-C 的确定性要求。

13 编译器架构

前端基于 MLIR，依次经过高阶数组 IR（HL）、张量布局 IR（TL）与 LLVM IR（LL）三级 lowering。Shape inference、Layout lowering 与 Memory planning 逐级完成。中端进行多面体依赖分析、自动向量化与循环 tiling；后端混合使用 LLVM 与 MLIR-to-C 路径，针对 ARM Cortex-M、RISC-V、DSP 生成最优指令与内联汇编。运行时库仅依赖最小 C 运行时与硬件抽象层（HAL），实现 DMA、Cache 与中断的零开销封装。

14 执行模型

系统采用静态调度为主、动态微调度为辅的混合策略。事件驱动路径把中断延迟压到亚微秒级；功耗管理在算子级插入 DVFS 与 Clock gating 决策。热补丁与 A/B 固件更新通过增量编译与符号重定位实现，可在 OTA 时仅替换热点内核。

15　实践案例

在边缘目标检测任务中，我们将 MobileNet-Int8 量化后映射到 180 KiB Flash 与 48 KiB SRAM，在 80 MHz 主频下达到 120 FPS。工业振动信号实时 FFT 通过复数 SIMD 与 DMA 链式传输，比 CMSIS-DSP 快 3.8 倍且功耗降低 42%。多核 RISC-V 行人检测流水线利用核间零拷贝队列与静态任务图，端到端延迟仅 0.8 ms。

16　基准对比

与手写 C、Rust ndarray 与 MicroTVM 对比，本语言在代码尺寸、执行周期、能耗与开发效率四项指标上均取得综合最优。消融实验显示，零拷贝视图与算子融合各自贡献约 35% 与 28% 的性能提升。

17　工程落地

语言规范与参考实现已按 Apache 2.0 开源。VSCode 插件提供语法高亮、形状可视化与指令集仿真；跟踪工具通过 ITM/ETM 探针实时采集性能计数器。下一步将与 AUTOSAR、MISRA 及 Khronos SYCL 小组对接，共同制定嵌入式张量计算标准。

18　挑战与展望

未来工作将聚焦异构存储层次的自动管理、MCU OTA 场景下的增量编译与 JIT 预热，以及利用大模型自动生成高性能内核描述。形式化验证方面，我们计划把 CertiKOS 风格的证明方法引入到线性类型与 WCET 分析中，为安全关键系统提供数学级保证。

19　结论

本文提出了一种面向资源受限嵌入式平台的张量 DSL，通过依赖类型、线性类型与多面体优化，在极小存储与硬实时约束下实现了高性能数值计算。该工作不仅填补了现有工具链的空白，也为边缘智能与工业控制的软硬件协同设计提供了新范式。后续将持续开源并与产业标准对接，期待更多开发者共同完善这一生态。

第 III 部

WebAssembly 跨平台渲染

黄京

May 17, 2026

20 WebAssembly 在跨平台渲染中的核心优势

WebAssembly 作为一种高效的二进制指令格式，其在跨平台渲染领域展现出显著优势。它既能保留近乎原生的执行速度，又能在浏览器、桌面及移动端实现一致的渲染体验。通过将渲染逻辑编译为 WebAssembly 模块，开发者得以在不同操作系统与硬件平台间共享同一套代码库，从而大幅降低平台适配成本。相比传统 JavaScript 渲染方案，WebAssembly 提供的确定性执行模型与内存安全机制，为高性能图形处理奠定了坚实基础。

21 编译管线与模块初始化

从源代码到可执行 WebAssembly 模块的转换，依赖于精心设计的编译管线。首先，开发者使用 Rust 或 C++ 等系统级语言编写渲染核心逻辑，然后通过 Emscripten 或 wasm-pack 等工具链将其编译为 .wasm 文件。编译过程中，优化器会对循环展开、常量折叠等操作进行处理，确保最终模块在加载时能迅速完成实例化。模块初始化阶段，WebAssembly 实例会分配线性内存，并将宿主环境提供的导入对象注入模块，供后续渲染调用使用。

22 内存管理与数据交换机制

在跨平台渲染过程中，内存管理与数据交换机制扮演着关键角色。WebAssembly 的线性内存模型将所有数据视为连续字节数组，渲染所需的顶点缓冲区、纹理数据与着色器参数均需通过指针式访问进行传递。例如，当需要将一个顶 oint 坐标数组上传到 GPU 时，开发者需在 JavaScript 侧调用 `Module._malloc` 分配空间，随后用 `HEAPF32.set` 将数据写入对应偏移量。紧接着，通过导出函数将该偏移量与长度信息传递给 WebAssembly 模块，模块内部再调用对应的渲染接口完成绘制。这样的机制既保证了零拷贝的高效传输，又维持了跨语言边界的数据一致性。

23 图形 API 绑定与抽象层设计

为实现跨平台渲染，开发者通常在 WebAssembly 模块内部构建抽象层，将不同平台的图形 API 统一映射为同一接口。抽象层通过条件编译或运行时分支，分别调用 WebGL、OpenGL ES 或 DirectX 的对应函数。例如，在浏览器环境中，模块会调用 `gl.drawArrays` 完成三角形绘制；在桌面端则可能调用 `glDrawArrays`。抽象层的设计使上层渲染逻辑无需关心底层平台差异，仅需通过统一的函数签名完成调用。抽象层还负责转换坐标系、状态管理与资源生命周期，从而确保渲染结果在各平台保持一致。

24 着色器编译与执行流程

着色器作为渲染管线的核心，其编译与执行流程在 WebAssembly 环境中同样需要谨慎处理。开发者先将 GLSL 或 HLSL 着色器源码作为字符串传入模块，模块内部调用平台相关的编译器将其转换为二进制着色器对象。在浏览器端，这一步骤通常通过 WebGL 的 `compileShader` 接口完成；在桌面端则通过对应的 OpenGL 调用实现。执行阶段，

WebAssembly 模块会设置统一变量、属性指针与纹理单元，然后触发绘制调用。以下代码段演示了着色器上传与激活过程：

```
const vsSource = `#version 300 es
in vec3 aPosition;
uniform mat4 uMVP;
void main() {
   gl_Position = uMVP * vec4(aPosition, 1.0);
}`;
const vertexShader = gl.createShader(gl.VERTEX_SHADER);
gl.shaderSource(vertexShader, vsSource);
gl.compileShader(vertexShader);
gl.attachShader(program, vertexShader);
gl.linkProgram(program);
gl.useProgram(program);
```

该代码首先声明了一个版本为 300 es 的顶点着色器，输入属性 aPosition 携带三维坐标，均匀变量 uMVP 负责模型视图投影矩阵变换。接下来通过 createShader 创建着色器对象，并将源码注入对象。随后调用 compileShader 验证语法正确性，验证成功后将其附加到着色器程序。最终调用 linkProgram 完成链接并激活程序，为后续绘制提供支持。

25 性能优化与跨平台一致性策略

在追求跨平台渲染的同时，性能优化与跨平台一致性策略同样不可或缺。开发者可利用 WebAssembly 的 SIMD 提案，对向量运算进行并行加速；同时通过纹理压缩、LOD 分级与遮挡剔除等技术，降低 GPU 负载。为了保持各平台渲染结果的一致性，开发者需严格遵循规范的着色器语法与状态设置顺序，避免平台驱动差异带来的视觉偏差。优化后的 WebAssembly 渲染模块，其帧时间可稳定控制在 16 ms 以内，从而实现流畅的 60 FPS 体验。

综上所述，WebAssembly 跨平台渲染技术通过高效编译管线、线性内存管理与抽象层设计，实现了高性能且一致的图形输出。随着 WebAssembly 的 SIMD、线程与 GPU 访问提案逐步落地，未来跨平台渲染将进一步向原生性能逼近。开发者可期待在单一代码库下支持更多渲染特性，例如实时光追与高级后期处理，从而为用户带来更丰富的视觉体验。

第 IV 部

Git 配置管理

杨岢瑞

May 18, 2026

Git 作为现代软件开发的版本控制基石，其配置系统直接决定了提交信息的准确性、操作行为的规范性以及团队协作的效率。合理地管理 Git 配置可以避免身份切换带来的混乱、减少跨项目配置冲突带来的摩擦，同时保障安全与合规要求。常见的场景包括在个人项目与公司项目之间切换身份、为不同团队制定提交规范、以及在多个仓库中保持配置一致性。本文将帮助读者建立三层配置思维，掌握从全局到项目级别的精细管控方法，并提供可落地的命令与最佳实践。

26 Git 配置的基本概念

Git 的配置分为三层，优先级从高到低依次为本地配置、全局配置和系统配置。本地配置存储于当前仓库的 `.git/config` 文件中，只对该项目生效；全局配置存储于用户主目录下的 `~/.gitconfig` 文件中，适用于当前用户的所有仓库；系统配置则位于 `/etc/gitconfig`，通常由系统管理员维护，影响所有用户。通过命令 `git config --list --show-origin` 可以同时查看配置项及其来源文件，方便快速诊断当前生效的配置。当需要查询特定配置项的来源时，可以使用 `git config --get-all <key> --show-origin` 来列出该配置在不同层级的定义。

27 核心配置项详解

用户身份配置是 Git 日常操作必不可少的设置。`user.name` 与 `user.email` 分别记录提交者的姓名与电子邮件地址，这些信息会直接写入每一次提交记录中。如果使用 GPG 或 SSH 签名提交，还需要额外配置 `user.signingkey` 来指定签名密钥。核心行为配置则影响文本换行、编辑器选择与忽略文件规则，例如 `core.editor` 设置默认文本编辑器，`core.autocrlf` 控制换行符转换策略，`core.excludesfile` 指定全局忽略文件路径。安全与签名配置主要用于增强提交与标签的完整性，包括 `commit.gpgsign`、`tag.gpsign`、`gpg.format` 与 `user.signingkey` 的组合使用。网络与推送行为配置决定默认推送策略与拉取方式，常见选项有 `push.default`、`push.autoSetupRemote`、`pull.rebase` 与 `fetch.prune`。分支与合并策略配置则用于设定默认分支名称、快进合并行为与分支跟踪关系，典型配置包括 `init.defaultBranch`、`merge.ff`、`pull.ff` 与 `branch.autoSetupMerge`。颜色与显示优化配置可以提升命令行体验，通过 `color.ui`、`color.branch`、`color.status`、`color.diff` 等选项让 Git 输出更易于辨识。别名配置通过定义简洁的短命令来加速日常操作，常用别名有 `co`、`br`、`ci`、`st`、`lg`、`unstage` 与 `last`。

28 多身份管理策略

在实际开发中，私人仓库与公司仓库往往需要不同的提交身份，`user.name` 与 `user.email` 必须随项目环境变化而切换。解决这一问题的最佳方式是使用条件包含配置。在全局配置文件中加入以下代码段即可实现目录级自动切换：

```
[includeIf "gitdir:~/work/"]
    path = ~/.gitconfig-work
```

这段配置的解读是，当当前仓库路径匹配 ~/work/ 前缀时，Git 将自动加载 ~/.gitconfig-work 中的额外配置，从而实现工作身份与个人身份的隔离。另一种推荐做法是严格区分目录结构，在 ~/work/ 与 ~/personal/ 两个目录下分别克隆仓库，然后在每个仓库内部使用本地配置覆盖身份信息。脚本或钩子方式作为备选方案，可以在进入目录时自动执行配置切换脚本，但维护成本较高。个人与工作两种身份的推荐配置示例分别如下：

```
[user]
    name = 张三
    email = zhangsan@example.com
```

```
[user]
    name = 张三
    email = zhangsan@company.com
```

29　项目级配置与模板

创建项目级配置模板可以保证新仓库从一开始就遵循团队规范。通过命令 git config --global init.templateDir 设置模板目录后，Git 在初始化新仓库时会自动复制模板中的配置文件。项目级必须设置的配置包括强制身份信息、提交模板以及自定义钩子路径。commit.template 指向一个包含提交模板的文本文件，可以在每次提交时提示开发者填写规范格式。core.hooksPath 则允许将钩子脚本集中管理，避免将敏感脚本提交到仓库中。

第 V 部

如何设计更可靠的 LLM 提示工程

李睿远

May 19, 2026

当前阶段的提示工程已从早期单纯寻找「巧妙咒语」的探索，逐步转向需要系统性规划与验证的工程实践。开发者不再满足于一次成功的演示，而是希望模型在生产环境中持续提供稳定且可预测的响应。这种转变的根源在于，许多早期方法虽然能让模型「跑起来」，但在面对真实用户输入时却常常表现出不稳定的行为。

在实际应用中，可靠性缺失的典型表现包括幻觉、格式漂移、上下文丢失以及敏感性爆炸等现象。幻觉通常指模型生成的内容与事实或提供的上下文完全脱离；格式漂移则表现为输出结构在多次调用后逐渐偏离预期；上下文丢失往往发生在长对话中，模型忘记了先前的重要信息；敏感性爆炸则是指输入微小的变化就可能导致输出质量大幅下滑。这些现象共同说明，仅仅依赖单个提示模板已经无法满足生产级应用的需求。

本文的目标是提出一套可复用、可审计、可测量的提示设计框架。通过将提示工程从艺术转向科学，使其成为可控、可追踪的工程过程，从而帮助开发者构建出真正可信赖的 LLM 应用。

30 重新定义「可靠」：三个核心维度

一致性是可靠提示工程的第一项核心维度。它要求同一输入在多次运行时输出结果尽可能保持相似，而不是随机地呈现出不同版本的响应。一致性不仅包括内容的相似度，还包含格式、长度和风格的稳定性。开发者可以通过重复测试相同输入并计算输出之间的差异来评估一致性。例如，在使用嵌入向量相似度计算时，可将 `sim(A, B)` 表示为向量 A 和 B 的余弦相似度：

$$sim(A, B) = \frac{A \cdot B}{||A|| \cdot ||B||}$$

该 formel 计算了两个向相似度的方向性相似度。开发者需要将多个运行结果的向量进行平均处理，计算标准差，从而得到一个量化指标。标准偏差越小，意味着输出结果越一致。

格式稳定性是可靠提示工程的第二项核心维度。它要求模型输出结构始终保持相同，结构信息如标题、列表和字段标签应保持不变。格式稳定性在任务如 JSON 输出或表格生成时尤其 entscheidend。当输出格式不保持稳定时，解析程序就会遇到困难。开发者可以引入固定模板来强制模型遵循格式，如以下 example 的 JSON 模板：

```
{
  "answer": "string",
  "sources": ["string"],
  "":
}
```

该 JSON 模板要求 model 输出必须包含「answer」字段和「sources」字段，而且来源 must 提供一个数组。开发者需要反复测试该 template mit 随机输入来验证是否所有输出都能正确解析。解析失败次数的计数可以作为格式稳定性指标。

上下文保持是可靠提示工程的第三项核心维度。它要求模型在长对话或长输入中记住重要的信息，而不是逐渐 vergessen. 上下文保持的重要性是 in long-running 任务，例如多轮问 dialogue 或长文档处理。开发者可以通过注入 anchor sentences 或 anchor points 来增强上下文保持。Anchor sentences 是一种固定语句插入方法，它会每隔固

定步 time 重新声明重要信息。例如，在 einem dialogue 对话中，以下 formel 可以帮助量化上下文保持能力：

$$C(t) = \frac{numberofretainedfactsattimet}{totalimportantfacts}$$

该 C(t) 表示在时间 t 的保留重要事实数量的比率。开发者可以定期插入 anchor points 重新强调重要信息，并计算 C(t) 值在整个对话中保持的水平。

31 结论：提示工程转向工程科学

提示工程不再是单纯寻找巧妙咒语，而是需要系统性的设计、验证和测量过程。开发者必须将 consistency、格式稳定性与上下文保持三个维度纳入设计考虑。通过使用向相似度公式、固定模板和 anchor points 方法，开发者可以将 previously 无法测量的行为量化并验证。最终目标是让 LLM 应用从「能跑」转向「可信」，从而真正支持生产级应用。