龙芯中科
LOONGSON TECHNOLOGY

数字电路与Verilog知识回顾

龙芯中科
LOONGSON TECHNOLOGY

目录

龙芯中科
LOONGSON TECHNOLOGY

目录

CPU实验中Verilog的使用

- CPU实验中有两个地方用到Verilog语言：一个地方是要求用Verilog语言进行电路设计，另一个地方是用Verilog语言搭建功能验证平台（testbench）
  - CPU的实现就属于电路设计的范畴。
  - 为了验证所实现的CPU是否正确，我们通常会搭建一个简单的系统，包括总线、内存、串口，把测试CPU的程序装载到内存中，同时给系统以复位和时钟。这些总线、内存、串口、时钟以及复位、程序的装载等都属于testbench的范畴。为了减少学生的学习负担，我们也用Verilog语言进行testbench的开发
- 用Verilog进行电路设计是重点，要求学生写的Verilog代码都属于这一类。这其中的关键是要求学生掌握Verilog中的可综合子集，并知晓常见电路的Verilog描述方法
- 用Verilog写的testbench在实验环境中都已提供，因此不要求学生动手写，能看得懂就可以。

面向硬件电路的Verilog设计思路

龙芯中科 LOONGSON TECHNOLOGY

- CPU本质上是一个数字逻辑电路，所以电路是设计的对象，Verilog只是描述电路的一个工具而已
- Verilog语言的很多语法要素与C语言很像，而且Verilog语言也支持行为级建模。这就使得很多同学习惯于用C程序开发的思路去使用Verilog语言，这种串行的过程化的思维对于Verilog语言设计来说是有很大弊端的。
- 要求学生必须采用这样的设计步骤：先进行电路结构设计，再进行Verilog代码编写
  - 当电路设计已经被清晰地分解为结构图中的各个模块和模块之间的连接、模块内部的数据通路和状态机、数据通路中的电路逻辑以及状态机中的状态转换图，那么接下来的Verilog代码设计就只是一个简单的“翻译”而已

可综合Verilog

龙芯中科 LOONGSON TECHNOLOGY

- Verilog和VHDL这类硬件描述语言设计最初是用于大型数字电路的建模、仿真，由基于HDL的设计转换为逻辑门相互连接的电路图的工作仍是由设计人员手工完成的，这个过程既费时费力又容易出错。后来逻辑综合工具的出现和发展改变了这一状况。设计者可以使用HDL在RTL级对电路进行描述，然后选定标准单元库并定义相关设计约束，那么逻辑综合工具就会自动的将HDL语言转换为门级网表，这个转换的过程称为逻辑综合。
- 在逻辑综合过程中，工具能够支持的语言要素是Verilog语言的一个子集，也就是说并不是所有的Verilog语言要素都可以进行逻辑综合
- 要求同学们采用RTL（Register Transfer Level，寄存器传输级）设计，且要求设计可以被EDA工具综合成最终的电路，那么同学们在进行电路设计时只能用到Verilog语言的可综合子集

龙芯中科
LOONGSON TECHNOLOGY

目录

模块声明和实例化

```
module bottom #(
    parameter A_WIDTH = 8,
    parameter B_WIDTH = 4,
    parameter Y_WIDTH = 2
)(
    input wire [A_WIDTH-1:0] a,
    input wire [B_WIDTH-1:0] b,
    input wire [ 3:0] c,
    output wire [Y_WIDTH-1:0] y,
    output reg z
);
......
endmodule
```

```
module top;
wire [15:0] btm_a;
wire [ 7:0] btm_b;
wire [ 3:0] btm_c;
wire [ 3:0] btm_y;
wire btm_z;

bottom #(
    .A_WIDTH (16),
    .B_WIDTH ( 8),
    .Y_WIDTH ( 4)
    )

inst_btm(
    .a (btm_a), // I
    .b (btm_b), // I
    .c (btm_c), // I
    .y (btm_y), // O
    .z (btm_z) // O
    );
endmodule
```

基础逻辑门

龙芯中科
LOONGSON TECHNOLOGY

```
wire [7:0] a;
wire [7:0] b;
assign y1 = ~a; // 反相器
assign y2 = a & b; // 与
assign y3 = a | b; // 或
assign y4 = a ^ b; // 异或
assign y5 = ~(a & b); // 与非
assign y6 = ~(a | b); // 或非
```

译码器

龙芯中科 LOONGSON TECHNOLOGY

```
module decoder_3_8(
    input [2:0] in,
    output [7:0] out
);
assign out[0] = (in == 3'd0);
assign out[1] = (in == 3'd1);
assign out[2] = (in == 3'd2);
assign out[3] = (in == 3'd3);
assign out[4] = (in == 3'd4);
assign out[5] = (in == 3'd5);
assign out[6] = (in == 3'd6);
assign out[7] = (in == 3'd7);
endmodule
```

编码器

```
module encoder_8_3(
    input [7:0] in,
    output [2:0] out
);
assign out =   in[0] ? 3'd0 :
               in[1] ? 3'd1 :
               in[2] ? 3'd2 :
               in[3] ? 3'd3 :
               in[4] ? 3'd4 :
               in[5] ? 3'd5 :
               in[6] ? 3'd6 :
               3'd7;
endmodule
```

```
module encoder_8_3(
    input [7:0] in,
    output [2:0] out
);
assign out = ({3{in[0]}} & 3'd0)
           | ({3{in[1]}} & 3'd1)
           | ({3{in[2]}} & 3'd2)
           | ({3{in[3]}} & 3'd3)
           | ({3{in[4]}} & 3'd4)
           | ({3{in[5]}} & 3'd5)
           | ({3{in[6]}} & 3'd6)
           | ({3{in[7]}} & 3'd7);
endmodule
```

多路选择器

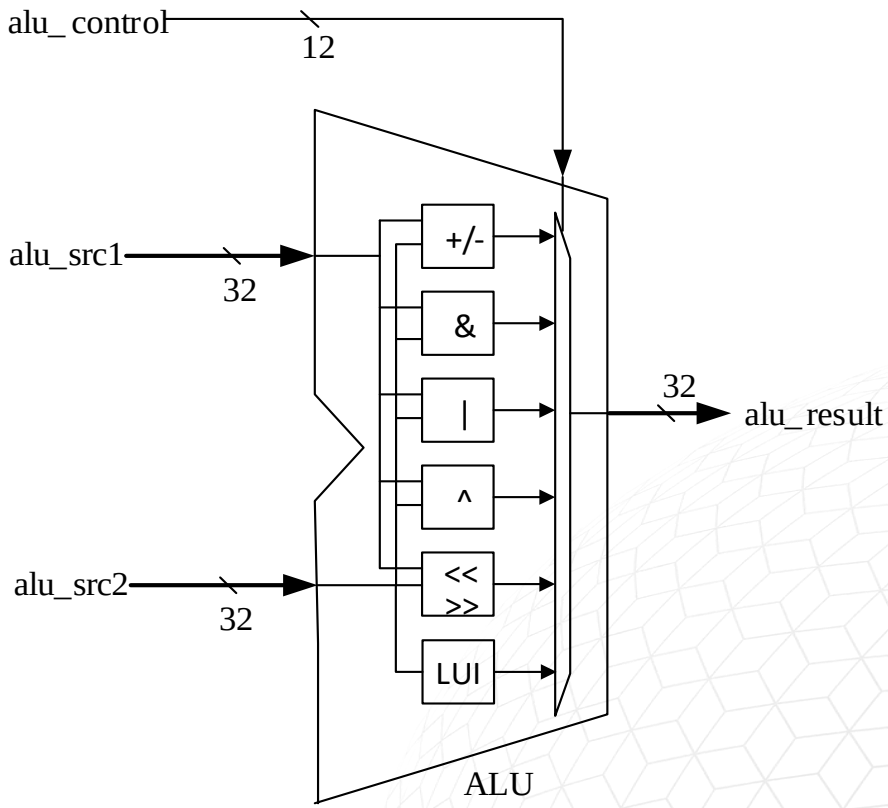
```
module mux5_8b(
    input [7:0] in0, in1, in2, in3, in4,
    input [2:0] sel,
    output [7:0] out
);
assign out =   (sel==3'd0) ? in0 :
               (sel==3'd1) ? in1 :
               (sel==3'd2) ? in2 :
               (sel==3'd3) ? in3 :
               (sel==3'd4) ? in4 :
                             8'b0;
endmodule
```

```
module mux5_8b(
    input [7:0] in0, in1, in2, in3, in4,
    input [2:0] sel,
    output [7:0] out
);
assign out = ({8{sel==3'd0}} & in0)
           | ({8{sel==3'd1}} & in1)
           | ({8{sel==3'd2}} & in2)
           | ({8{sel==3'd3}} & in3)
           | ({8{sel==3'd4}} & in4);
endmodule
```

```
module mux5_8b_onehot(
    input [7:0] in0, in1, in2, in3, in4,
    input [4:0] sel,
    output [7:0] out
);
assign out = ({8{sel[0]}} & in0)
           | ({8{sel[1]}} & in1)
           | ({8{sel[2]}} & in2)
           | ({8{sel[3]}} & in3)
           | ({8{sel[4]}} & in4);
endmodule
```

ALU模块

```
module simple_alu(
    input [11:0] alu_op,
    input [31:0] alu_src1,
    input [31:0] alu_src2,
    output [31:0] alu_result
);
......
assign op_add  = alu_op[ 0];
assign op_sub  = alu_op[ 1];
assign op_slt  = alu_op[ 2];
assign op_sltu = alu_op[ 3];
assign op_and  = alu_op[ 4];
assign op_nor  = alu_op[ 5];
assign op_or   = alu_op[ 6];
assign op_xor  = alu_op[ 7];
assign op_sll  = alu_op[ 8];
assign op_srl  = alu_op[ 9];
assign op_sra  = alu_op[10];
assign op_lui  = alu_op[11];
```



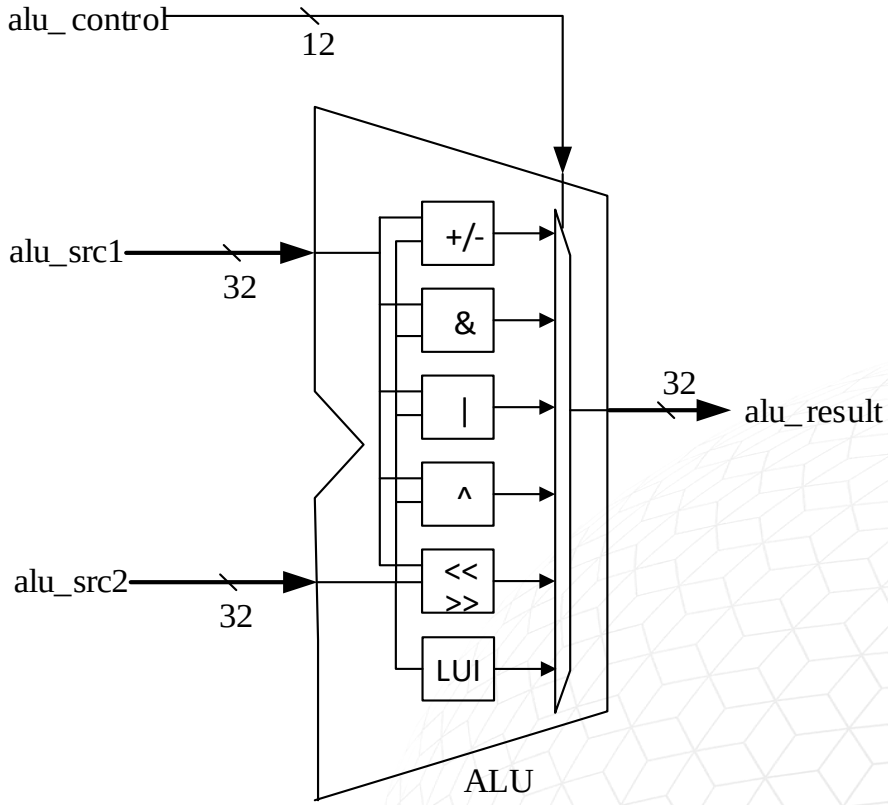
ALU模块

龙芯中科 LOONGSON TECHNOLOGY

```
assign and_result = alu_src1 & alu_src2;
assign or_result = alu_src1 | alu_src2;
assign nor_result = ~or_result;
assign xor_result = alu_src1 ^ alu_src2;
assign lui_result = {alu_src2[19:0], 12' b0};

assign adder_a = alu_src1;
assign adder_b = (op_sub | op_slt | op_sltu) ? ~alu_src2 : alu_src2;
assign adder_cin = (op_sub | op_slt | op_sltu) ? 1'b1 : 1'b0;
assign {adder_cout, adder_result} = adder_a + adder_b + adder_cin;

assign add_sub_result = adder_result;

assign slt_result[31:1] = 31' b0;
assign slt_result[0] = (alu_src1[31] & ~alu_src2[31])
                     | (~(alu_src1[31]^alu_src2[31]) & adder_result[31]);
assign sltu_result[31:1] = 31' b0;
assign sltu_result[0] = ~adder_cout;

assign sll_result = alu_src2 << alu_src1[4:0];
assign srl_result = alu_src2 >> alu_src1[4:0];
assign sra_result = ($signed(alu_src2)) >>> alu_src1[4:0];
```

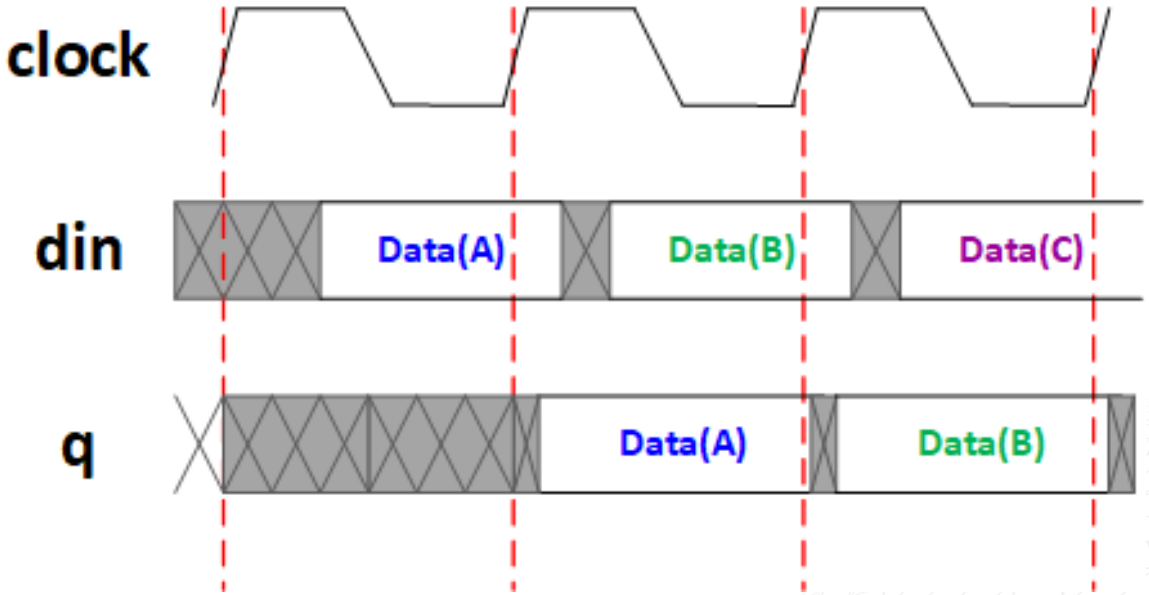
ALU模块

```
assign alu_result =
          ({32{op_add|op_sub   }} & add_sub_result)
        | ({32{op_slt          }} & slt_result)
        | ({32{op_sltu         }} & sltu_result)
        | ({32{op_and          }} & and_result)
        | ({32{op_nor          }} & nor_result)
        | ({32{op_or           }} & or_result)
        | ({32{op_xor          }} & xor_result)
        | ({32{op_sll          }} & sll_result)
        | ({32{op_srl          }} & srl_result)
        | ({32{op_sra          }} & sra_result)
        | ({32{op_lui          }} & lui_result);

endmodule
```



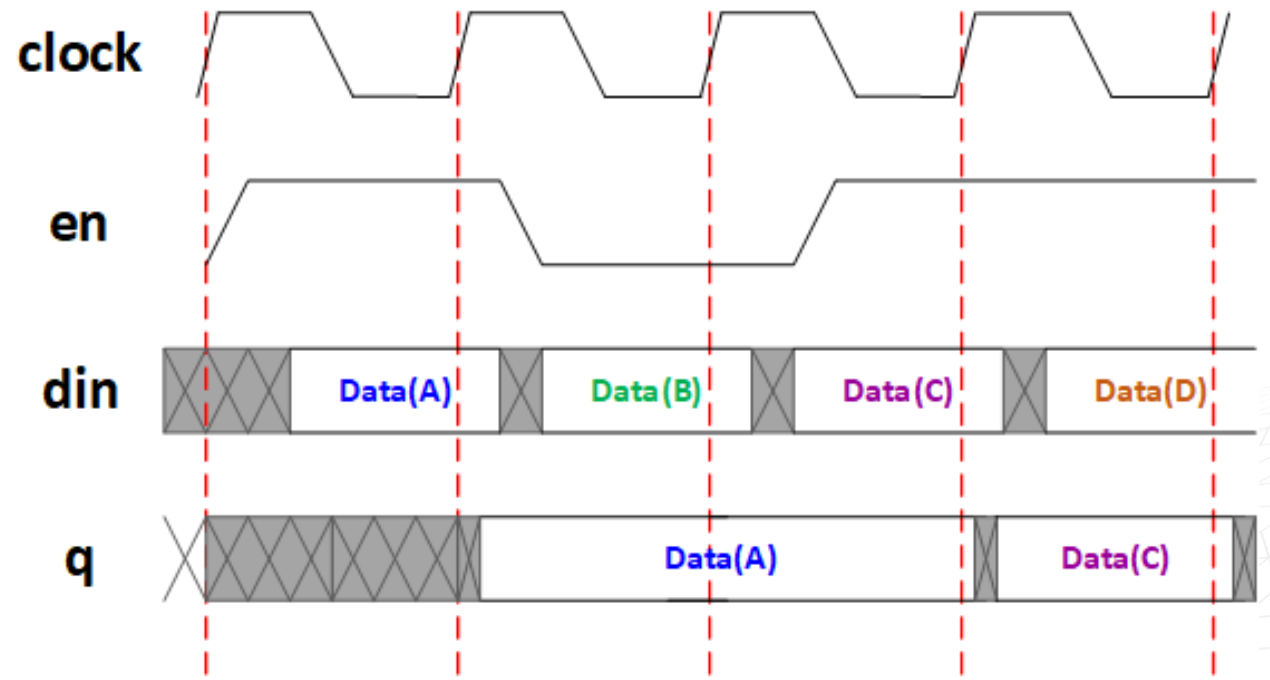
D触发器

```
module dff(
    input clk,
    input din,
    output reg q
);
always @(posedge clk) begin
    q <= din;
end
endmodule
```



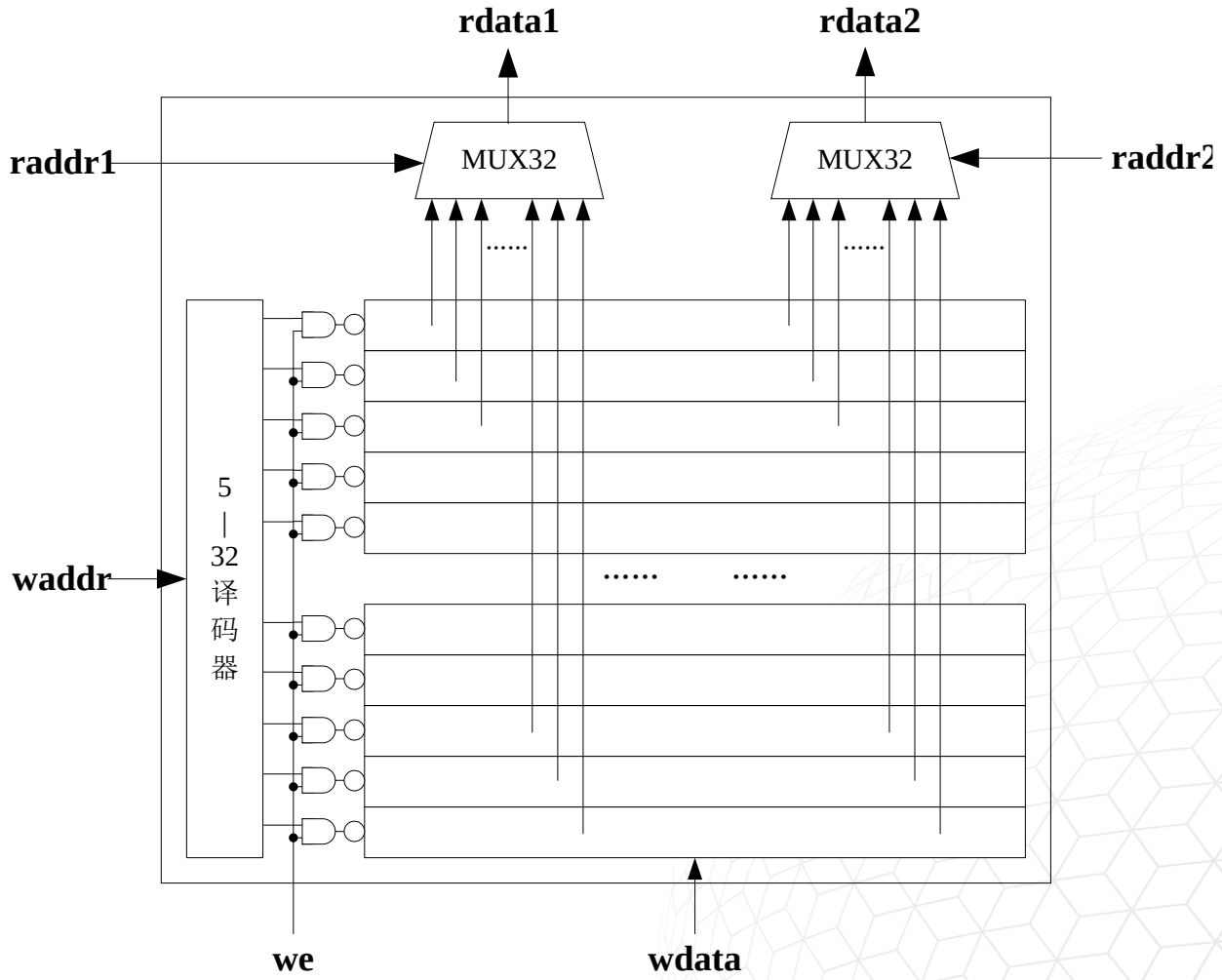
带使能端的D触发器

```
module dff_en(
    input clk,
    input en,
    input din,
    output reg q
);
always @(posedge clk) begin
    if (en) q <= din;
end
endmodule
```



实验中CPU所用的寄存器堆

```
module regfile(
    input clk,
    input [ 4:0] raddr1,
    output [31:0] rdata1,
    input [ 4:0] raddr2,
    output [31:0] rdata2,
    input we,
    input [ 4:0] waddr,
    input [31:0] wdata
);
reg [31:0] reg_array[31:0];
// WRITE
always @(posedge clk) begin
    if (we) reg_array[waddr]<= wdata;
end
// READ OUT 1
assign rdata1 = (raddr1==5'b0) ? 32'b0
                               : reg_array[raddr1];
// READ OUT 2
assign rdata2 = (raddr2==5'b0) ? 32'b0
                               : reg_array[raddr2];
endmodule
```



龙芯中科
LOONGSON TECHNOLOGY

为人民做我芯