

计算机组成原理

实验3 浮点运算器设计



HITSZ 实验与创新实践教育中心
Education Center of Experiments and Innovations, HITsZ

实验目的

- 掌握IEEE754单精度浮点数的**格式**
- 掌握IEEE754单精度浮点数加减法**运算的过程**
- 了解CPU**运算器**的设计方法



实验内容

◆ 设计浮点运算器

- 在模板工程上，设计支持IEEE754 float32加减法的运算器
- 要求：
 - ① 实现规格化数据的加减法运算
 - ② 必做题只需通过Testbench前5组(共10个)测试用例
 - ③ 实现方式不限（原码、补码均可；组合、时序均可）
 - ④ 不考虑 $\pm\infty$ 、NaN作为输入数据、运算结果的情况
 - ⑤ 不能使用Vivado库或任何第三方IP核

实验内容

- 附加题 (+0.5分):

- 修改必做题代码，使运算器支持非规格化数据的加减法运算
- 要求:
 - ① 不考虑 $\pm\infty$ 、NaN作为输入数据、运算结果的情况
 - ② 不能使用Vivado库或任何第三方IP核
 - ③ 需通过Testbench的所有11组 (共22个) 测试用例

实验原理

◆ IEEE754单精度浮点数

- 格式：

S	Exponent	Mantissa
---	----------	----------

1bit

8bit阶码

23bit尾数

- 表示的数据：

阶码	尾数	表示的数据	换算方法
8'h0	23'h0	± 0	-
8'h0	除23'h0外	非规格化数	$(-1)^S \cdot (Mantissa)_2 \cdot 2^{-126}$
8'h1 ~ 8'hFE	任意	规格化数	$(-1)^S \cdot (\{1, Mantissa\})_2 \cdot 2^{Exponent-127}$
8'hFF	23'h0	$\pm Inf$	$\pm \infty$
8'hFF	除23'h0外	NaN	Not a Number

- 运算基本步骤：求阶差、对阶、尾数运算、规格化
- 编码实现方式：原码、补码

实验原理 - 编码实现

◆ 基于原码的浮点加减步骤:

• $x = (-1)^{S_x} \cdot M_x \cdot 2^{E_x-127}$, $y = (-1)^{S_y} \cdot M_y \cdot 2^{E_y-127}$, 求 $z = x \pm y$

① 求阶差: $\Delta E = |E_x - E_y|$

② 小阶对大阶: 设 E_x 更大, 则令 y 的尾数 M_y 右移 ΔE 位, 得到 M_y'

③ 尾数运算: 计算 $(-1)^{S_x} \cdot M_x \pm (-1)^{S_y} \cdot M_y'$, 根据结果得出 S_z 和 M_z

④ 规格化: 规格化数的格式: **1位隐藏1** + **23位小数**

找出 M_z 最左侧的1作为隐藏1, 再截取其后的23bit作为尾数

M_z :

左规

0_0000_0101_1001_0101_1110_0011_0

23bit的定宽窗口

$E_z = E_x - 5$



实验原理 - 编码实现

◆ 基于原码的浮点加减步骤:

• $x = (-1)^{S_x} \cdot M_x \cdot 2^{E_x-127}$, $y = (-1)^{S_y} \cdot M_y \cdot 2^{E_y-127}$, 求 $z = x \pm y$

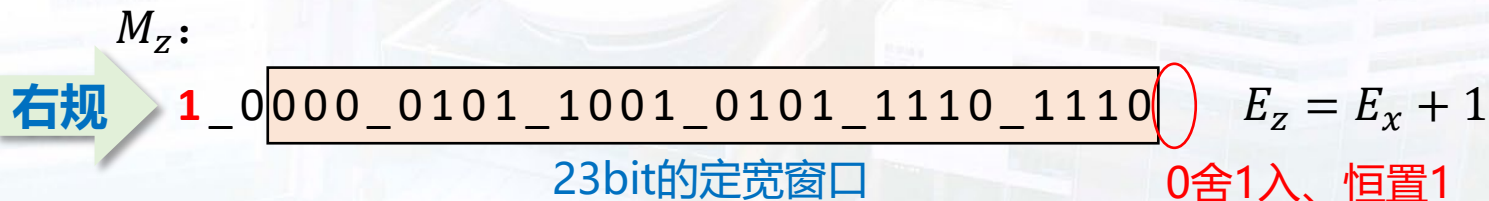
① 求阶差: $\Delta E = |E_x - E_y|$

② 小阶对大阶: 设 E_x 更大, 则令 y 的尾数 M_y 右移 ΔE 位, 得到 M_y'

③ 尾数运算: 计算 $(-1)^{S_x} \cdot M_x \pm (-1)^{S_y} \cdot M_y'$, 根据结果得出 S_z 和 M_z

④ 规格化: 规格化数的格式: **1位隐藏1** + **23位小数**

找出 M_z 最左侧的1作为隐藏1, 再截取其后的23bit作为尾数



实验原理 - 编码实现

◆ 基于原码的浮点加减步骤:

- 【步骤】求阶差 ➡ 对阶 ➡ 尾数运算 ➡ 规格化
- 【优点】容易理解、无编码转换
- 【缺点】需保证运算结果也是原码，故加减法需分开处理
 - 尾数运算： $(-1)^{S_x} \cdot M_x \pm (-1)^{S_y} \cdot M_y'$
 - 被减数 < 减数时，需变换被减数和减数
- 【注意】尾数运算时，增加1bit数据位（记录进位）

实验原理 - 编码实现

◆ 基于补码的浮点加减步骤:

- $x = \{S_x, E_x, M_x\}$, $y = \{S_y, E_y, M_y\}$, 求 $z = x \pm y$

① 求补码:

- ◆ 阶码的补码: 假定y阶码更小, 则求 $[-E_y]_{\text{补}}$;
- ◆ 尾数的补码: $[M_x]_{\text{补}}$ 、 $[M_y]_{\text{补}}$ (若是减法运算, 求 $[-M_y]_{\text{补}}$)
 - $[M_x]_{\text{补}} = \{S_x, [\{S_x, M_x\}]_{\text{补}}\}$ (尾数是无符号数)

② 求阶差: $\Delta E = [E_x]_{\text{补}} + [-E_y]_{\text{补}}$

③ 对阶: $[M'_y]_{\text{补}} = [M_y]_{\text{补}} \gg_s \Delta E$ (若是减法运算, $[M'_y]_{\text{补}} = [-M_y]_{\text{补}} \gg_s \Delta E$)

④ 尾数运算: 计算 $\text{Sum} = [M_x]_{\text{补}} + [M'_y]_{\text{补}}$, 得z的符号 $S_z = \text{Sum}[\text{MSB}]$

实验原理 - 编码实现

◆ 基于补码的浮点加减步骤:

⑤ 规格化:

- ◆ 右规: 双符号位不同, $Sum \gg_s 1$
 - ◇ 阶码 $E_z = E_x + 1$
 - ◇ 尾数 $[M_z]_{\text{补}} = Sum[\text{MSB}-1:\text{MSB}-25]$ (1位符号、1位隐藏1、23位尾数)
- ◆ 左规: 符号与最高的数据位相同, Sum 左移直到符号与最高数据位不同
 - ◇ 阶码 $E_z = E_x - \text{移位位数}$
 - ◇ 尾数 $[M_z]_{\text{补}} = Sum[\text{MSB}-1:\text{MSB}-25]$

⑥ 求尾数原码:

- 根据 $[M_z]_{\text{补}}$ 求出 M_z , 得 $z = \{ S_z, E_z, M_z[22:0] \}$

实验原理 - 编码实现

◆ 基于补码的浮点加减步骤:

- 【步骤】原转补 ➡ 求阶差 ➡ 对阶 ➡ 尾数运算 ➡ 规格化 ➡ 补转原
- 【优点】符号与数值一起运算、减法可当作加法处理
- 【缺点】不直观、编码转换
 - 在IEEE754标准中，阶码是移码，尾数是原码
 - 移码的求补方法与原码相同
- 【注意】尾数设置双符号位（记录进位）

实验原理 - 电路实现

◆ 运算器的**电路实现方案**：组合电路

- 依靠**assign**语句、**always @(*)**实现
 - 【难点】规格化的**位数**
 - ✓ Solution: 结合**位操作**和**多路选择器**
- 接口信号：

接口信号	位宽	属性	释义
op	1	输入	0-加法；1-减法
A	32	输入	被加/减数
B	32	输入	加/减数
C	32	输出	运算结果

实验原理 - 电路实现

◆ 运算器的**电路实现方案**：时序电路

- 运算过程有清晰的步骤，适合使用**状态机**实现
- 接口信号：

接口信号	位宽	属性	释义
rst	1	输入	高电平复位
clk	1	输入	时钟信号
start	1	输入	运算开始信号
op	1	输入	0-加法；1-减法
A	32	输入	被加/减数
B	32	输入	加/减数
ready	1	输出	就绪信号
C	32	输出	运算结果

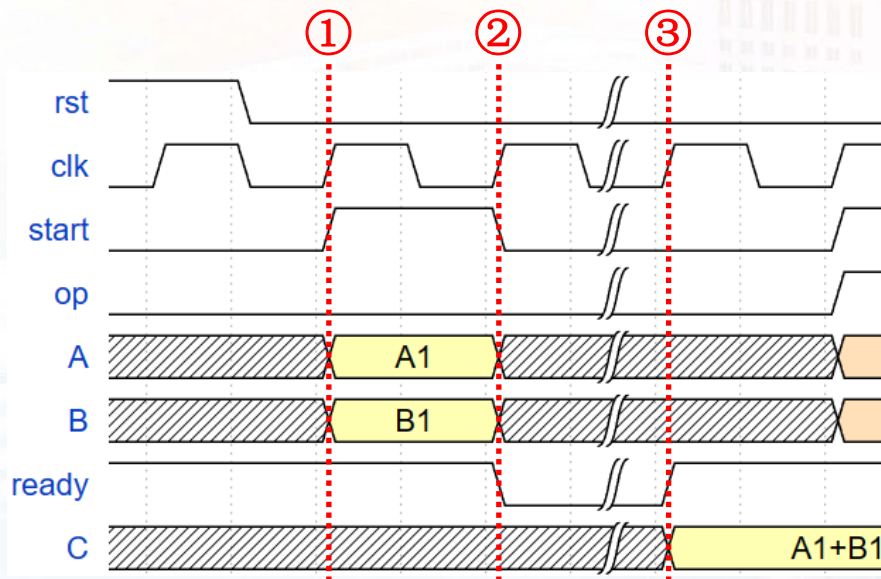
start有效代表
有新数据输入

ready有效代表
运算器就绪，
可进行下一次
运算

实验原理 - 电路实现

◆ 运算器的**电路实现方案**：时序电路

- 运算过程有清晰的步骤，适合使用**状态机**实现
- 遵循以下时序：

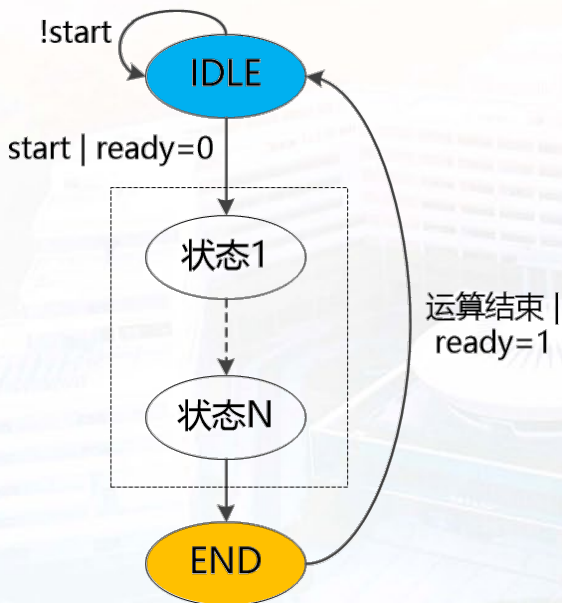


- ① **ready**有效，**start**才会有效
start有效时，**op**、**A**和**B**有效
- ② **start**只维持1个**clk**
- ③ 运算完成后，拉高**ready**，
输出运算结果

实验原理 - 电路实现

◆ 运算器的**电路实现方案**：时序电路

- 状态机示例：



- ◆ IDLE状态：等待运算开始
 - start有效时进入下一状态
 - 缓存输入的op、A和B
 - 拉低ready信号
- ◆ END状态：运算完成
 - 输出运算结果
 - 拉高ready信号

实验步骤

1. 理清IEEE754单精度浮点数格式及其运算过程
2. 打开模板工程，实现浮点运算器
 - 若采用组合电路方案，完成fpu.v
 - 若采用**时序**电路方案，完成fpu_**clk**.v
3. 运行功能仿真，根据波形完成调试
 - 若采用组合电路方案，用fpu_tb.sv作为Testbench
 - 若采用**时序**电路方案，用fpu_tb_**clk**.sv作为Testbench
4. 按模板撰写实验报告

仿真设置

1. 设置顶层模块（仿真的顶层模块也要改）

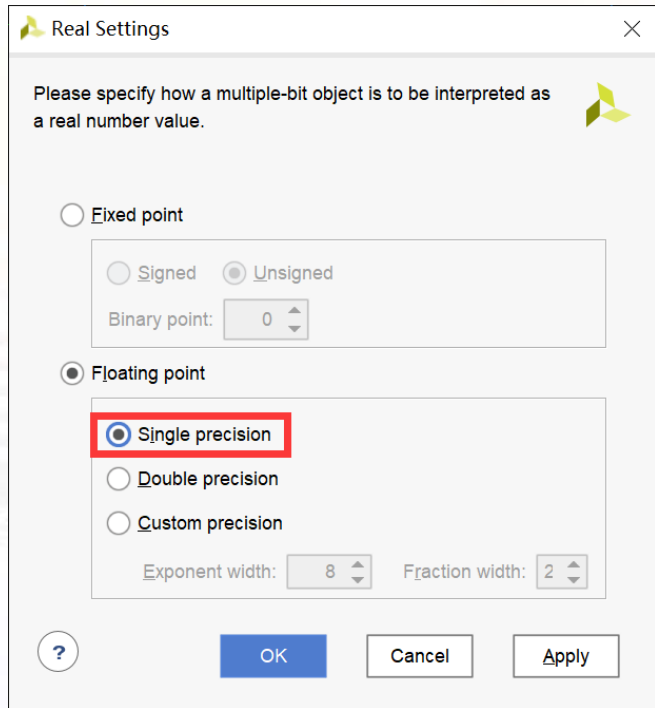
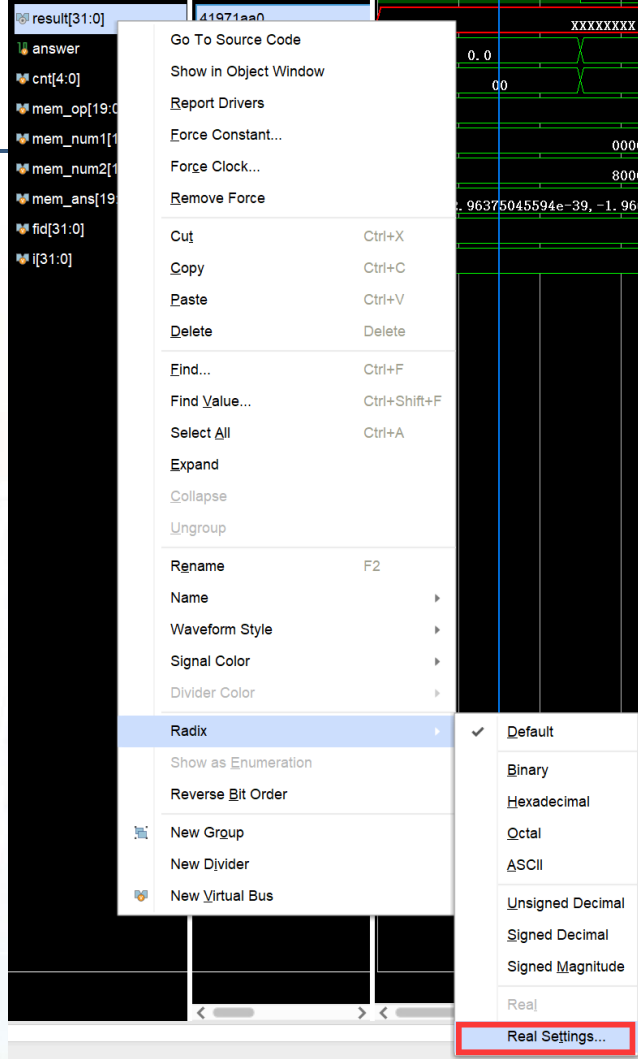
The screenshot illustrates the process of setting a simulation source file as the top module in the Xilinx IDE. The interface is divided into several panes:

- Sources Pane:** Displays the project hierarchy. Under 'Simulation Sources', the file 'fpu_tb (fpu_tb.sv)' is selected. A right-click context menu is open, with 'Set as Top' highlighted at the bottom.
- Project Summary Pane:** Shows project details such as 'Product family: Artix-7', 'Project part: xc7a100tfgg484-', 'Top module name: fp_unit_clk', and 'Target language: Verilog'.
- Simulation Sources Pane:** Lists the files in the simulation sources, including 'fpu_tb_clk (fpu_tb_clk.sv)' and 'fpu_tb (fpu_tb.sv)'.
- Utility Sources Pane:** Lists utility source files.
- Source File Properties Pane:** Shows the 'Hierarchy' tab, which includes options like 'Set as Top' and 'Set Global Include'.

The 'Set as Top' option is highlighted with a red rectangle, indicating the correct action to set the selected file as the top module for simulation.

仿真设置

2. 设置显示格式



验收&提交

- **课堂验收**

- 课上检查是否通过前5组（共10个）测试用例：2分

- **提交内容**

- 必做题：fpu.v 或 fpu_clk.v：1分
实验报告（按模板完成）：4分
 - 附加题：fpu.v 或 fpu_clk.v：+0.5分

- 将上述文件打包成.zip，以“**学号_姓名.zip**”命名提交到作业系统

- ◆ 注意：**如有雷同，双方均0分！**



HITSZ 实验与创新实践教育中心
Education Center of Experiments and Innovations, HITSZ