

计算机组成原理

实验2 原码除法器设计

2024 · 春

哈工大



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

实验目的

- 理解基于恢复余数、加减交替的原码除法原理
- 掌握使用HDL设计实现原码除法器的方法
- 了解片上系统SoC的基本架构



实验内容

1. 基于恢复余数法/加减交替法，设计一个8bit的整数原码除法器

要求：

① 输入输出均使用8bit原码表示

② 不得使用 /、% 运算符

③ 其余未规定细节可自行决定

2. 将所设计的原码除法器扩展到32bit，并集成到SoC中



实验原理

1. 整数原码除法

- 设有 $[x]_{\text{原}} = x_0x_1x_2 \cdots x_n$, $[y]_{\text{原}} = y_0y_1y_2 \cdots y_n$, 则:

$$\left[\frac{x}{y}\right]_{\text{原}} = (x_0 \oplus y_0) \cdot \frac{x_1x_2 \cdots x_n}{y_1y_2 \cdots y_n} = (x_0 \oplus y_0) \cdot \frac{x^*}{y^*}$$

其中, x_0 、 y_0 —— $[x]_{\text{原}}$ 、 $[y]_{\text{原}}$ 的符号位

x^* 、 y^* —— $[x]_{\text{原}}$ 、 $[y]_{\text{原}}$ 的绝对值

- 商的符号 = 被除数的符号 **异或** 除数的符号
- 商的值 = 被除数的绝对值 除以 除数的绝对值
- 余数的符号与被除数相同

实验原理

2. 恢复余数法

- ① 被除数加上 $[-y^*]_{\text{补}}$
- ② 若被除数(余数)为负, 商上0, 加上 $[y^*]_{\text{补}}$ 以恢复余数;
若被除数(余数)为正, 商上1;
- ③ 若左移次数小于 n , 不含符号的位宽
令被除数(余数)左移1位;
否则结束

【例】 $x = 13$, $y = -6$, 求 $z = [x \div y]_{\text{原}}$

$$[x]_{\text{原}} = 01101,$$

$$[y^*] = 00110, [-y^*]_{\text{补}} = 11010$$

$$z_0 = x_0 \oplus y_0 = 0 \oplus 1 = 1, \text{ 故商为负}$$

被除数(余数)	商	说明
0 0 0 0 1 1 0 1 +1 1 0 1 0		$+[-y^*]_{\text{补}}$
1 1 0 1 0 1 1 0 1 +0 0 1 1 0	___ 0	余数为负, 不够减, 上商0 恢复余数: $+ [y^*]$
0 0 0 0 0 1 1 0 1 0 0 0 0 1 1 0 1 0	__ 0 _	恢复余数后, 商和余数左移1位 $+ [-y^*]_{\text{补}}$
1 1 0 1 1 1 0 1 0 +0 0 1 1 0	__ 0 0	余数为负, 不够减, 上商0 恢复余数: $+ [y^*]$
0 0 0 0 1 1 0 1 0 0 0 0 1 1 0 1 0 0	_ 0 0 _	恢复余数后, 商和余数左移1位 $+ [-y^*]_{\text{补}}$
1 1 1 0 1 0 1 0 0 +0 0 1 1 0	_ 0 0 0	余数为负, 不够减, 上商0 恢复余数: $+ [y^*]$
0 0 0 1 1 0 1 0 0 0 0 1 1 0 1 0 0 0	0 0 0 _	恢复余数后, 商和余数左移1位 $+ [-y^*]_{\text{补}}$
0 0 0 0 0 1 0 0 0 0 0 0 0 1 0 0 0 0	0 0 0 1 0 0 1 _	余数为正, 上商1 商和余数左移1位 $+ [-y^*]_{\text{补}}$
1 1 0 1 1 0 0 0 0 +0 0 1 1 0	0 0 1 0	余数为负, 不够减, 上商0 恢复余数: $+ [y^*]$
0 0 0 0 1 0 0 0 0		

实验原理

2. 恢复余数法

- ① 被除数加上 $[-y^*]_{\text{补}}$
- ② 若被除数(余数)为负, 商上0, 加上 $[y^*]_{\text{补}}$ 以恢复余数;
若被除数(余数)为正, 商上1;
- ③ 若左移次数小于 n , 不含符号的位宽
令被除数(余数)左移1位;
否则结束

【例】 $x = 13$, $y = -6$, 求 $z = [x \div y]_{\text{原}}$

$$[x]_{\text{原}} = 01101,$$

$$[y^*] = 00110, [-y^*]_{\text{补}} = 11010$$

$$z_0 = x_0 \oplus y_0 = 0 \oplus 1 = 1, \text{ 故商为负}$$

被除数(余数)	商	说明
$\begin{array}{r} 0001101 \\ + 11010 \end{array}$		$+[-y^*]_{\text{补}}$
$\begin{array}{r} 11011101 \\ + 00110 \end{array}$	---0	余数为负, 不够减, 上商0 恢复余数: $+ [y^*]$
$\begin{array}{r} 00001101 \\ \underline{00011010} \\ + 11010 \end{array}$	--0_	恢复余数后, 商和余数左移1位 $+ [-y^*]_{\text{补}}$
$\begin{array}{r} 11101010 \\ + 00110 \end{array}$	--00	余数为负, 不够减, 上商0 恢复余数: $+ [y^*]$
$\begin{array}{r} 00011010 \\ \underline{00110100} \\ + 11010 \end{array}$	_00_	恢复余数后, 商和余数左移1位 $+ [-y^*]_{\text{补}}$
$\begin{array}{r} 00000100 \\ \underline{00001000} \\ + 11010 \end{array}$	_001 001_	余数为正, 上商1 商和余数左移1位 $+ [-y^*]_{\text{补}}$
$\begin{array}{r} 11011000 \\ + 00110 \end{array}$	<u>0010</u>	余数为负, 不够减, 上商0 恢复余数: $+ [y^*]$
$\begin{array}{r} 00001000 \end{array}$		



实验原理

3. 加减交替法

- ① 被除数加上 $[-y^*]_{\text{补}}$
- ② 若被除数(余数)为负, 商上0,
被除数(余数)左移1位后 $+[y^*]_{\text{补}}$;
若被除数(余数)为正, 商上1,
被除数(余数)左移1位后 $[-y^*]_{\text{补}}$;
- ③ 若左移次数小于n, 回到②; 否则结束

【例】 $x = 13, y = -6$, 求 $z = [x \div y]_{\text{原}}$
 $[x]_{\text{原}} = 01101$,
 $[y^*] = 00110, [-y^*]_{\text{补}} = 11010$
 $z_0 = x_0 \oplus y_0 = 0 \oplus 1 = 1$, 故商为负

被除数(余数)	商	说明
$\begin{array}{r} 00001101 \\ +11010 \end{array}$		$+[y^*]_{\text{补}}$
$\begin{array}{r} 110101101 \\ 101011010 \\ +00110 \end{array}$	$\begin{array}{r} _ _ _ 0 \\ _ _ 0 _ \end{array}$	余数为负, 不够减, 上商0 商和余数左移1位 $+[y^*]$
$\begin{array}{r} 110111010 \\ 101110100 \\ +00110 \end{array}$	$\begin{array}{r} _ _ 00 \\ _ 00 _ \end{array}$	余数为负, 不够减, 上商0 商和余数左移1位 $+[y^*]$
$\begin{array}{r} 111010100 \\ 110101000 \\ +00110 \end{array}$	$\begin{array}{r} _ 000 \\ 000 _ \end{array}$	余数为负, 不够减, 上商0 商和余数左移1位 $+[y^*]$
$\begin{array}{r} 000001000 \\ 000010000 \\ +11010 \end{array}$	$\begin{array}{r} 0001 \\ 001 _ \end{array}$	余数为正, 上商1 商和余数左移1位 $+[y^*]_{\text{补}}$
$\begin{array}{r} 110110000 \\ +00110 \end{array}$	$\underline{0010}$	余数为负, 不够减, 上商0 $+[y^*]$
$\begin{array}{r} 000010000 \end{array}$		

实验原理

3. 加减交替法

- ① 被除数加上 $[-y^*]_{\text{补}}$
- ② 若被除数(余数)为负, 商上0,
被除数(余数)左移1位后 $+[y^*]_{\text{补}}$;
若被除数(余数)为正, 商上1,
被除数(余数)左移1位后 $+[-y^*]_{\text{补}}$;
- ③ 若左移次数小于n, 回到②; 否则结束

【例】 $x = 13$, $y = -6$, 求 $z = [x \div y]_{\text{原}}$

$$[x]_{\text{原}} = 01101,$$

$$[y^*] = 00110, [-y^*]_{\text{补}} = 11010$$

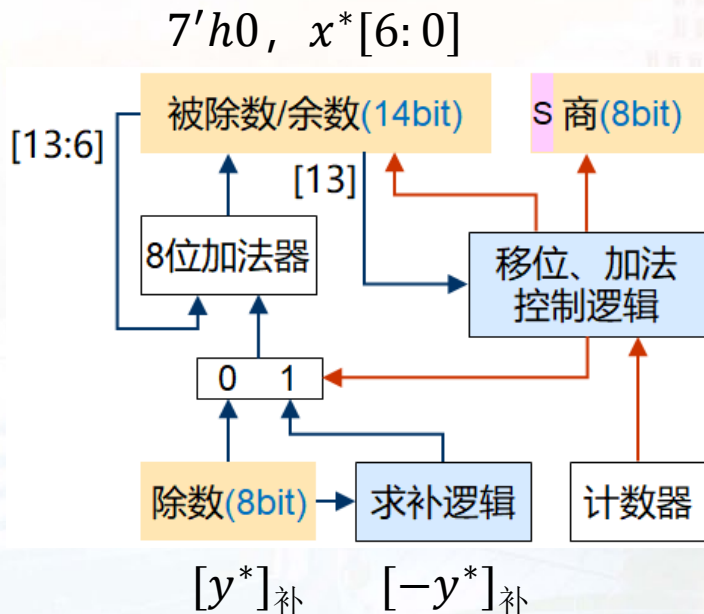
$$z_0 = x_0 \oplus y_0 = 0 \oplus 1 = 1, \text{ 故商为负}$$

被除数(余数)	商	说明
<div>0 0 0 1 1 0 1</div> <div>+ 1 1 0 1 0</div>		$+[-y^*]_{\text{补}}$
<div>1 1 0 1 1 1 0 1</div> <div><div>1 0 1 1 1 0 1 0</div></div> <div>+ 0 0 1 1 0</div>	<div>__ 0</div> <div>__ 0 _</div>	余数为负, 不够减, 上商0 商和余数 左移1位 $+ [y^*]$
<div>1 1 1 0 1 0 1 0</div> <div><div>1 1 0 1 0 1 0 0</div></div> <div>+ 0 0 1 1 0</div>	<div>__ 0 0</div> <div>_ 0 0 _</div>	余数为负, 不够减, 上商0 商和余数 左移1位 $+ [y^*]$
<div>0 0 0 0 0 1 0 0</div> <div><div>0 0 0 0 1 0 0 0</div></div> <div>+ 1 1 0 1 0</div>	<div>_ 0 0 1</div> <div>0 0 1 _</div>	余数为正, 上商1 商和余数 左移1位 $+ [-y^*]_{\text{补}}$
<div>1 1 0 1 1 0 0 0</div> <div>+ 0 0 1 1 0</div>	<div><u>0 0 1 0</u></div>	余数为负, 不够减, 上商0 $+ [y^*]$
<div>0 0 0 0 1 0 0 0</div>		

实验原理

4. 除法器设计

- 原码除法的基本操作：求补、加法、左移、异或



① 复位时

- 计数器初始为6，其余清零

② 运算时

- 商符S通过异或求得
- 控制逻辑根据被除数 (余数) 的符号，选择加法的数据源，并控制上商
- 控制逻辑根据计数器值，控制被除数 (余数) 的移位次数

实验原理

4. 除法器设计

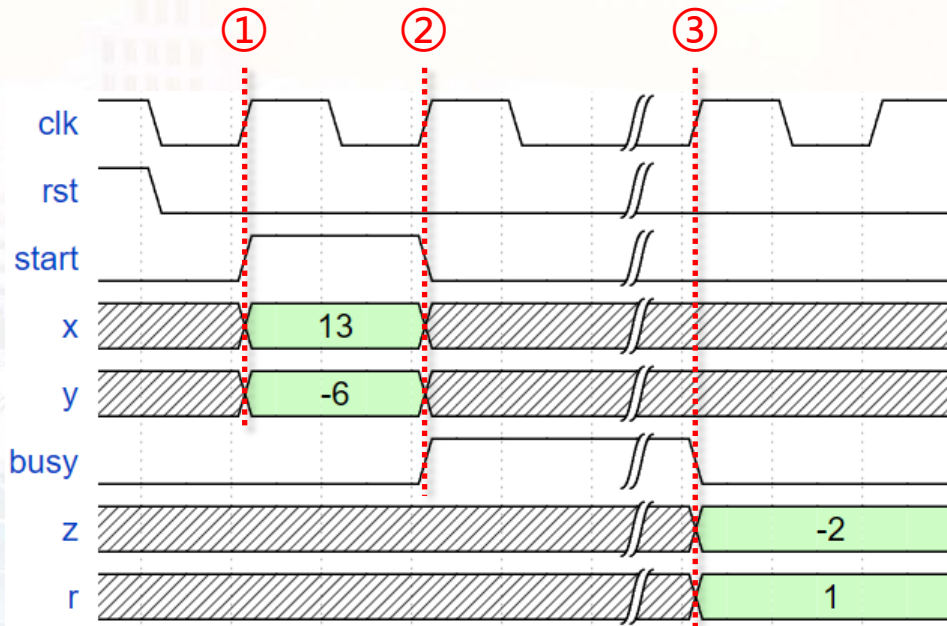
序号	名称	位宽	属性	功能描述
1	clk	1	输入	时钟信号
2	rst	1	输入	复位信号（高电平有效）
3	x	8	输入	被除数（原码）
4	y	8	输入	除数（原码）
5	start	1	输入	启动信号
6	z	8	输出	商（原码）
7	r	8	输出	余数（原码）
8	busy	1	输出	忙标志信号

实验原理

4. 除法器设计

➤ 除法器接口信号时序

- ① start表示何时开始运算：
start有效时，x、y也有效，
此时即将要进行除法运算
- ② start、x、y仅有效1个clk，
除法器缓存数据后，拉高
busy信号，表示正在运算，
此时不可接收新的数据
- ③ 运算结束，拉低busy，输出
商和余数



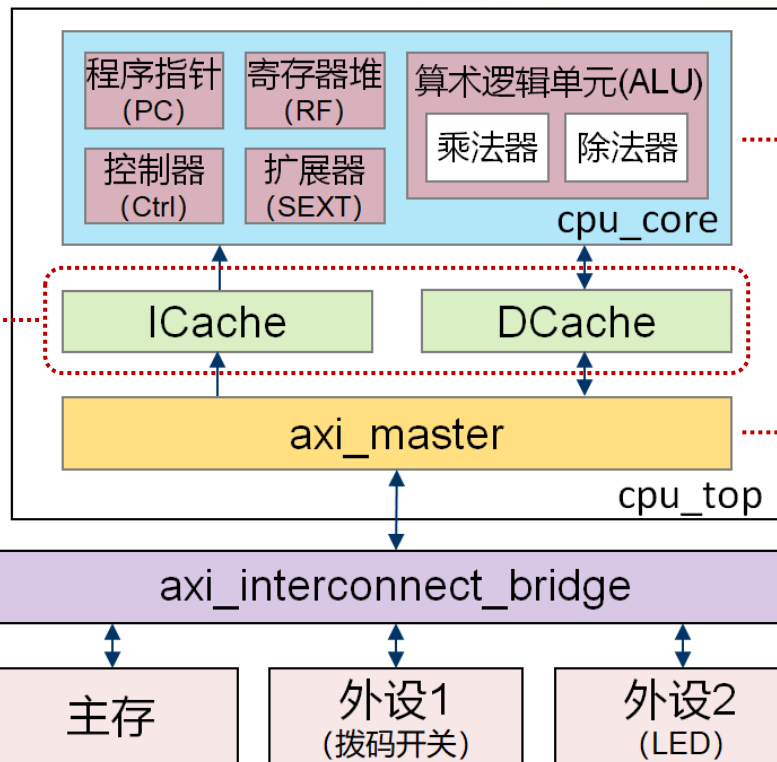
实验原理

5. SoC架构简介

除去乘/除法器，
其他模块均已提供

一级存储 (Cache)
哈佛结构

二级存储 (主存)
冯·诺依曼结构



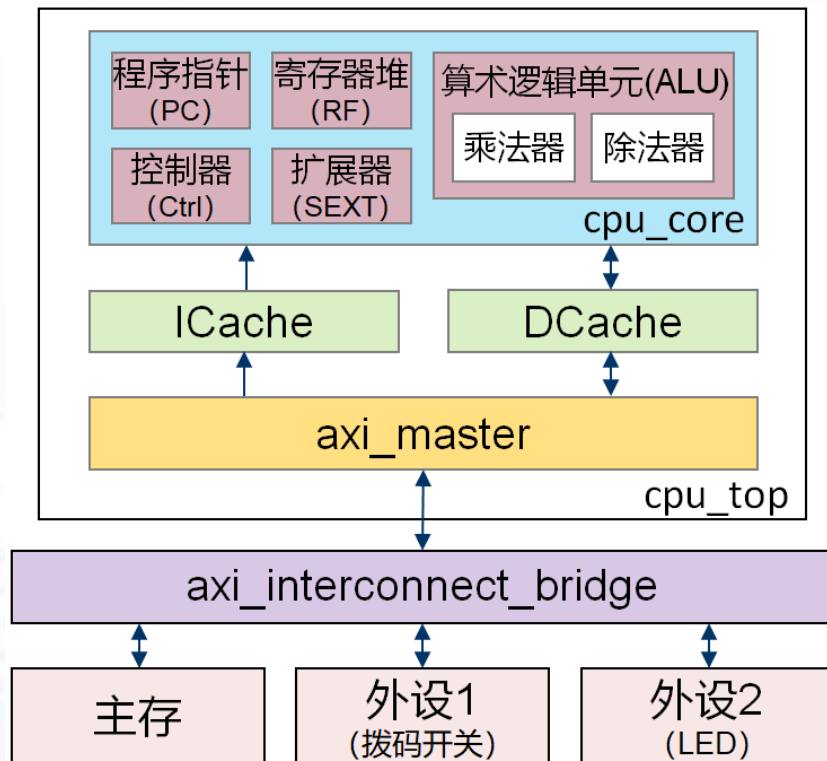
RV32多周期处理器核

协议转换, 使CPU
支持AXI总线接口

实现多设备互连

实验原理

5. SoC架构简介



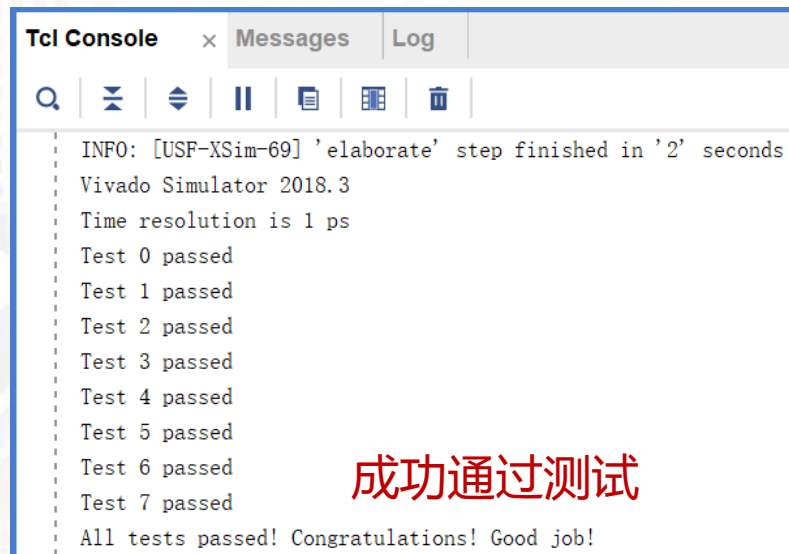
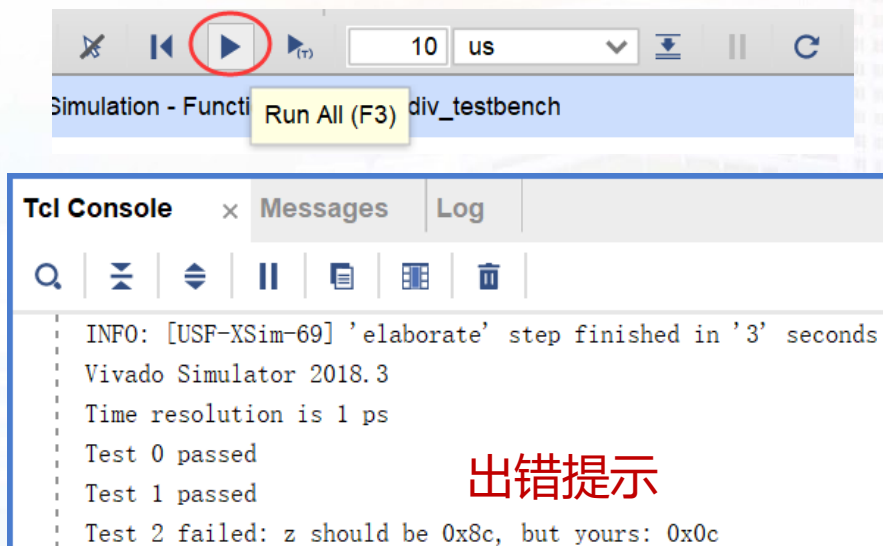
指令执行过程:

- cpu_core以PC为地址，向ICache取指
- ➔ ICache命中：直接返回指令
- ICache缺失：向axi_master发出读请求
- ➔ axi_master向AXI总线桥发出读请求
- ➔ AXI总线桥根据请求地址，向相应的设备（主存）发出读请求，从而取出指令
- ➔ 指令依次经过AXI总线桥、axi_master、ICache，最终进入cpu_core
- ➔ cpu_core对指令进行译码、执行和写回

实验步骤

1. 在lab2_div模板工程中，完成原码除法器设计

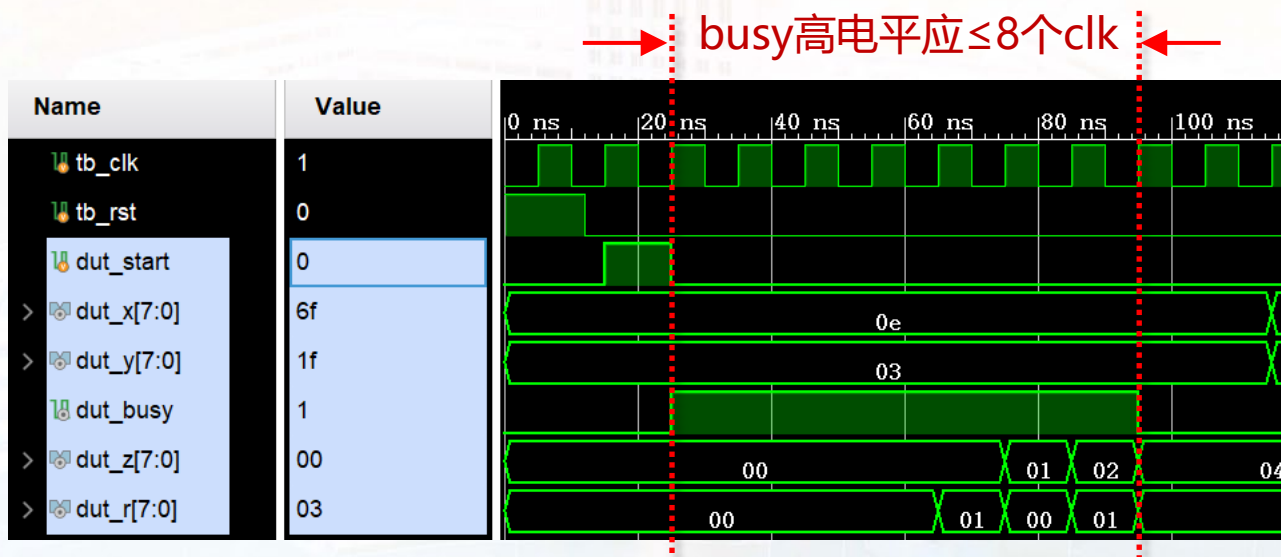
- ① 下载lab2_div工程，基于恢复余数法or加减交替法，完成divider模块
- ② 运行功能仿真，并对除法器进行调试和验证



实验步骤

1. 在lab2_div模板工程中，完成原码除法器设计

- ① 下载lab2_div工程，基于恢复余数法or加减交替法，完成divider模块
- ② 运行功能仿真，并对除法器进行调试和验证

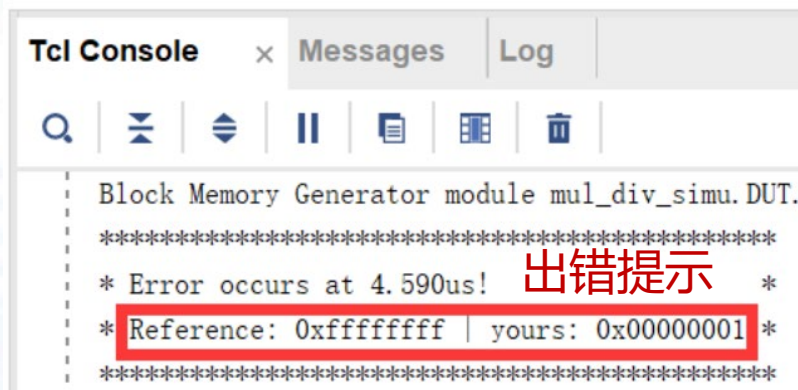


实验步骤

2. 扩展除法器到32位，并集成到miniRV_axi工程

- ① 将8位原码除法器扩展成32位
- ② 下载SoC工程miniRV_axi，将扩展后的除法器连接到SoC工程
- ③ 运行功能仿真，并对除法器进行调试和验证

注意不要修改乘/除法器以外的其余模块



```
Tcl Console x Messages Log
[Icons]
Block Memory Generator module mul_div_simu.DUT.
*****
* Error occurs at 4.590us! 出错提示 *
* Reference: 0xffffffff | yours: 0x00000001 *
*****
```

验收与提交

- **验收内容**

- 课上检查是否通过lab2_div工程中的所有测试用例：2.5分
- 课上检查集成到SoC后的除法器是否通过所有测试用例：0.5分

- **提交内容**

- 原码除法器的.v源文件(含8位和32位)：1分
- 实验报告（按模板完成）：3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