在npc中移植lightsss

从炸盘的痛苦中解脱出来

分享人：彭培智

学号：ysyx_24080020

What's the LightSSS?

lightSSS是开源香山的工具组件之一，官网介绍如下：

- 在仿真调试过程中，并不需要完整过程的波形，只需要在仿真出错的时候把前面一段时间内的波形保存即可。Verilator等仿真器提供了持久化保存电路状态的snapshot功能，仿真出错时可以调用这个功能生成相应的状态文件，复现错误的时候可以用这个文件从特定电路状态开始跑仿真。但这个功能的局限性有以下几点：
  - 只能保存RTL电路的状态，其他仿真的部分比如参考模型、DRAMSim3等的状态不能被保存
  - 而且当电路规模比较大的时候，保存电路状态的存储开销比较大
- 为此我们开发了轻量级的仿真快照工具lightSSS，它可以在仿真进程出错时自动保存出错点附近的波形和debug信息而不需要经过电路状态文件这个中间层。

lightSSS 原理

这里我简单说下，详细讲解可观看 王植鑫同学的分享会————《香山LightSSS剖析》， 强烈推荐 :)

利用fork函数(对当前进程进行bit级复制)来保留程序在不同状态的快照，每隔一段时间就fork新的子进程，并将多余的子进程结束，每个子进程被创建出来是都会被阻塞，直到父进程出错时，去唤醒最老的子进程，结束最老子进程的阻塞状态，并开启波形跑子进程。 这样就得到了从子进程被创建的那个状态到出错的状态之间的所有波形信息。

缘从何起?

我在排查sdram的bug时，因为sdram的地址空间比较大，开启波形仿真时，会生成巨大的波形文件，非常占据磁盘空间，而且使用gtkwave打开太大的波形文件会非常缓慢。

所以刚开始我设置了开始仿真的步长，即对周期进行计数，当周期数大于某个数值时才保存波形，但是对于不同的错误，我并不能事先知道出错时大致的周期范围，所以我在调试bug时花费了大量的时间去定位出问题时周期范围，哪怕使用二分法去定位，依旧耗费不少时间。

直到从一生一芯的分享会中得知了今天的主角`lightSSS`！

开始移植

找到香山项目工具集的子项目仓库地址：OpenXiangshan/difftest: https://github.com/OpenXiangShan/difftest

在 `src/test/csrc/common/` 目录中有 `lightsss.h lightsss.cpp` 两个文件，添加到自己的npc目录环境中，记得要修改`Makefile`将这两个文件来编译他们。

简单看一下这两个文件： lightsss.h lightsss.cpp

在哪里添加创建子进程/唤醒子进程等的逻辑?

因为我们的difftest可以测试指令维度的正确性，所以我们可以在`exec_npc()`函数中每条指令执行完成后添加创建子进程的逻辑，并且判断全局变量`npc_state`的状态（会在difftest函数中修改状态，和ref不等就会被修改）来修改共享内存中的`flag，notgood`变量，子进程此时在`shwait`函数中会跳出死循环，是否生成波形的逻辑也根据以上的两个变量来决定。这样就的到了预期的只在bug附近的波形文件, 大功告成，最后别忘了在父进程退出之前清理子进程，以免造成内存泄漏。

一些额外的修改

此时你会发现当父进程`assert`失败，或者自己手动`ctrl + c`中断进程执行后，子进程并没有被清理，造成了内存泄漏，必须去手动清理遗留的子进程，非常的麻烦。
聪明如你马上就想到信号处理函数了：

```
```

此时就需要对`lightsss.h lightsss.cpp`进行一些细微的改动，因为c++的特性，成员函数会自带一个指针，不能与`signal`函数的参数匹配，所以要讲用到的成员函数和成员变量声明成`static`类型，并在类的外部额外定义对应的函数或变量。

结束