复旦大学集成电路与微纳电子创新学院

2024~2025 学年第二学期期末考试试卷

A 卷

课程名称：EDA 系统软件分析和设计方法学 课程代码：MICR130037.01

开课院系：集成电路与微纳电子创新学院 考试形式：课程论文

姓名：祝彦翔 学号：22307130073 专业：微电子科学与工程

提示：请同学们秉持诚实守信宗旨，谨守考试纪律，摒弃考试作弊。学生如有违反学校考试纪律的行为，学校将按《复旦大学学生纪律处分条例》规定予以严肃处理。

内容	需求分析	类关系图	团队分工	实现思路	测试方法	总分
得分						

（以下为试卷正文**或课程论文题目**）

1. 问题：实现一个并行化概念性的混合仿真器（Mix-mode simulator），要求尽可能提高混合仿真的效率

（装订线内不要答题）

1. 引言

本报告针对 Fudan University EDA(2025)课程项目——混合事件模拟实现。

项目完成人为祝彦翔，学号 22307130073，邮箱 22307130073@m.fudan.edu.cn，gitee 仓库地址 https://gitee.com/m18652829333/eda-program.git。

本项目旨在模拟混合模拟和数字信号事件（A、D、M 类型），并生成符合要求的输出文件。具体代码和测试实例可见作者 gitee 仓库(https://gitee.com/m18652829333/eda-program.git)下 eda-program/PJ2 目录。

2. 任务需求分析

项目目标是实现一个事件驱动的模拟系统，处理 A（模拟）、D（数字）和 M（混合）事件，并根据输入参数生成正确输出。

2.1 事件类型

D（数字事件）： 包含 V（0 或 1）、dT（时间间隔）、T（发生时间）、bM（与 M 事件的关系，1 为产生、-1 为消耗、0 为无关系）。

A（模拟事件）： 包含 V（0 到 1 的浮点值，配置文件确定）、Vth（阈值，0 到 1，配置文件确定）、dT（0 到 0.1 的间隔，随机生成）、T（发生时间）、bM。

M（混合事件）： 虚拟事件，根据 A 和 D 事件的条件生成，不包含在最终结果中，用于同步。

2.2 规则和交互

M 事件不会生成新 M 事件；bM=-1 的 M 事件可覆盖 bM=0 的潜在 D/A 事件；若同时发生 bM_A=1 和 bM_D=1 的 M 事件，A 到 D 条件优先，设置 bM_A=1，bM_D=-1，也即在同一时刻 A 优先级高于 D 事件。

A 事件产生 D 事件： A 事件的时间间隔是配置文件中固定的，当 A 事件的 V 跨过了 Vth 时会触发一个新的 D 事件覆盖掉原有的 D 事件，这个 D 事件的 V 取决于旧的 D 事件的 V 和触发新 D 事件的 A 事件的跨越阈值方向。

D 事件产生 A 事件： D 事件的时间一到一定会产生一个新的 A 事件，这个 A 事件不再和旧的 A 事件比较有无跨过 Vth。

2.3 期末项目完成思路

在期末项目中，对于仿真的时间有模拟，引入了仿真 A 事件和 D 事件的仿真时间 tA, tD。其中 tA 是通过要求文档 con.txt 传入，而 tD 则是在代码中实时生成随机整数。

在期中的项目中，利用了串行锁步法对事件进行模拟，逻辑的核心是一个事件队列的结构数组 Event。Event 数组大小为 2，存取当前运行目标 A 和 D 事件。主代码核心即为维护 Event 结构数组，包括队列的数据弹出、按时间进行数据重排等。另外，在期中项目中已经开发了两个接口代码 config.c/.h 和 simulator.c/.h，这两个接口分别作用于（config）处理输入文件 con.txt 把里面的数据存为全局变量，（simulator）用于生成新的 A 事件和 D 事件。由于生成事件的逻辑并没有变化，所以这两个模块可以直接在期末项目——快速混合仿真中直接使用。

针对于期末项目，为了实现一个尽量快的混合事件仿真模块，我的工程思路为先生成串行下

的可工作版本，再对这个版本进行并行化改进，实现 A 事件和 D 事件之间的并行化处理，以及多个 A 事件内及多个 D 事件内的并行。

3. 两种简易方法的简述

3.1 简单串行递进方法

这种方法采取了和期中锁步法相同的逻辑，对锁步法的代码进行了改进，引入了 tA 和 tD 事件生成的仿真时间。

在第一个 D 事件生成之前，串行的生成各个 A 事件，直到生成一个 bM=1 的 A 事件。在这之后进行 A 事件和 D 事件的共同仿真，我沿用了 PJ1 中的 Event 事件队列，在 MMSimulator.c 中对该队列进行实时维护。A 事件和 D 事件的时间停顿是串行进行的，所以可以得到在锁步机制下，总仿真时长可表示为：T = T_A + T_D + T_1，其中 T_A 为所有 A 事件的总时间，T_D 为所有 D 事件的总时间，T_1 则为回退 D 事件、逻辑级运行时间等。

但是显然，由于没有引入并行处理，串行下的锁步机制并不能最大化发挥仿真资源。于是我们可以引入初步并行化的 A/D 事件相互并行的并行逻辑。

3.2 A/D 事件相互并行方法

在本并行逻辑下，我们只构建两个线程，分别处理 A 事件和 D 事件。而 A 事件和 D 事件内部并没有进行并行处理。

同样我们可以分析它的工作过程。在得到第一次 D 事件之前，所有的事件都只会是 A 事件，这些 A 事件仍然是串行生成的。D 事件生成后，开启 A/D 事件的同步仿真，在本项目中 D 事件的 dT 要大于 A 事件的 dT。我们规定每次等待 tD 的时间。

我们可以考虑几种情况，如果 tD 时间后，生成的 A 事件的时间长度已经超出了新生成的 D 事件。这时分析就会很简单，因为在 t[n]到 t[n+1]之间的事件都已经知道，只要遍历生成的在 D 事件时间之前的 A 事件有无产生 M 事件。若有的话把 D 事件回退到这个时刻，而没有 A 事件生成 M 事件的话，那 D 事件就会产生 A 事件，回到循环中继续推进。

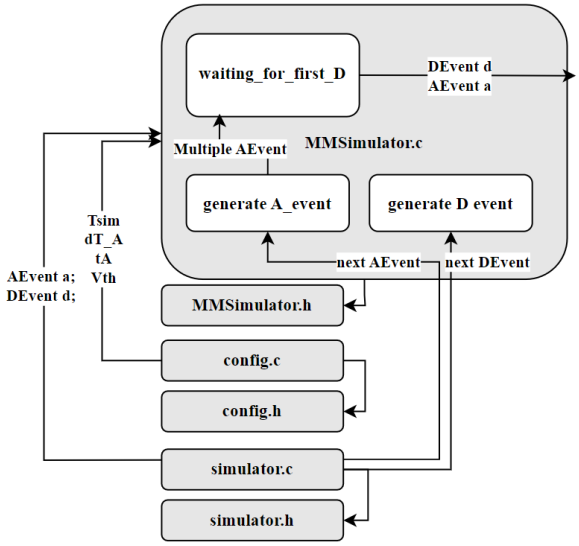
而如果在经过了 tD 时间后，A 事件仍没有推进到新生成的 D 事件的位置，那先分析在已经生成的 A 事件有无已经生成 M 事件，没有的话再进一步生成多个 A 事件（每次等待 tA 时间）直到 A 事件最终推进到 D 事件的时间。在这期间或者 D 事件产生 M 事件，或者 A 事件产生 M 事件，进而再次进入循环向前推进。

在实际情况下分析，可以发现这种简单的并行方式相比于串行锁步法有一定的提升。但是发现在相同仿真时间下 A 事件普遍时间步进要小于 D 事件，于是经常要在仿真 tD 时间后再等待几轮 tA，这段时间可以进一步简化。

下面着重展示高度并行化仿真的实现及思路。

4. 高度并行化仿真（最终方案）

4.1 实现接口化的类关系图



图表 1 接口关系图

在高度并行化的仿真模拟下，我们的模块与前者基本类似。Simulator 和 config 仍然是用于生成新的 A/D 事件和读取 con.txt 参数。

而主体 MMSimulator.c 代码中则包括了重要的函数 generate_A_event 和 generate_D_event，这两个函数用于在线程中生成 A 事件和 D 事件，在这个函数中包括了仿真时间停顿和事件生成等。下面展示了仿真时间停顿以供验证。

```c
void* generate_D (void* arg) {
    DThreadArgs* args = (DThreadArgs*)arg;

    usleep(args->tD * 1000);

    DEvent d_parallel = generate_next_D(d, d.T);

    pthread_mutex_lock(&lock);
    d_event_queue[args->idx] = d_parallel;
    pthread_mutex_unlock(&lock);
    free(arg);
    return NULL;
}
```

```c
void* generate_A(void* arg) {
    int index = *(int*)arg;

    usleep(tA * 1000);

    AEvent a_parallel = generate_next_A(a, a.T, Vth);

    pthread_mutex_lock(&lock);
    a_event_queue[index] = a_parallel;
    pthread_mutex_unlock(&lock);
    free(arg);
    return NULL;
}
```

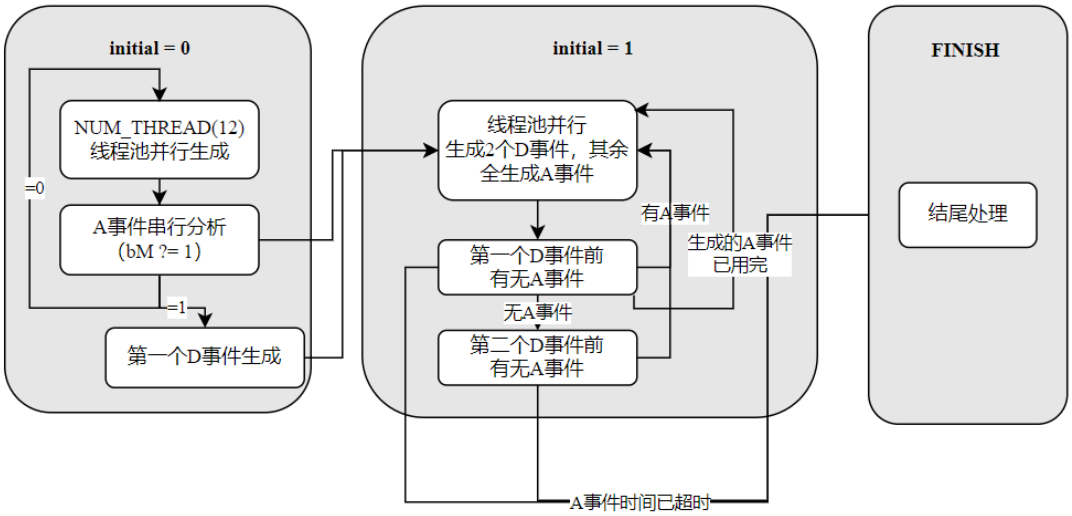
图表 2 仿真时间安插

而 waiting_for_first_D 实现的是在第一个 D 事件仍未生成时的逻辑操作，每批次并行生成 NUM_THREADS 个 A 事件，如果没有生成 D 事件的话继续生成下一个批次直到有 D 事件生成。

而主体的 A 和 D 事件混合逻辑在 main 函数中，可见软件实现思路部分。

4.2 软件实现思路含数据结构，高性能实现方法

4.2.1 架构级优化



图表 3 软件并行化版本架构图

上图为高度并行化方法的实现思路，在第一个 D 事件没有生成之前，构造一个 NUM_THREAD(即 12)线程的线程池，进行 waiting_for_first_D 操作。

进入 A / D 混合仿真状态后，实时动态的线程分配，线程数 threads 为 max(NUM_THREADS, 3+dT_D1/dT_A)，也即是线程的数目至少大于等于 3+dT_D1/dT_A。在每次多线程的实现过程中，总会有 2 个线程分配给 D 事件的生成，其余的（threads-2）个线程都分配给 A 事件的生成，也即是 A 事件至少会生成 1+dT_D1/dT_A 个，这表明这个线程之后，A 事件肯定推进到时了第一个 D 事件的时间之后，而且由于 NUM_THREADS=12 很大概率是 A 事件能推进到第二个 D 事件之后的。

多线程得到结果之后，进行事件的数据及逻辑分析。进入逻辑分支后循环对刚刚产生的 A 事件逐个分析：

1. 这个 A 的时间超出 Tsim （打印所有积压的 A 事件）
2. 这个 A 时间第一次越过第一个 D 事件时间 （打印所有积压 A 事件，打印 M[D]事件，改变后续 A 事件的时间后继续推进）
3. 这个 A 时间第一次越过第二个 D 事件时间 （打印所有积压 A 事件，打印 M[D]事件）
4. 这个 A 是线程产生的最后一个 A 事件（清空积压 A 事件）
5. 这个 A 事件的 bM=1 产生了 M 事件 （打印所有积压 A 事件，打印 M[A]事件）
6. 其它（无任何操作）--A 事件被积压，暂不打印

如果时间达到了 Tsim 的最终时刻，会直接跳出循环打印出 FINISH，合上输出文件。若非到达仿真最后时间，逻辑分支分析出来之后会再重新进入下一次的多线程分析，如此循环直到仿真结束。

4.2.2 逻辑级优化

期末项目的目标是在最短的时间内得到仿真结果，于是在架构优化的同时，我对于我整体的 C 代码的运行速度也尝试进行了提升。下图为我的优化项。其中 C 代码利用 O2 的优化编译方式对整体提速效果是比较明显的，可以提升约 2%的仿真速度（在本地实验的 con.txt 下）。

优化项	优化手段	效果说明

I/O 输出优化	fprintf → fwrite	提高写入效率，减少格式开销
编译器优化	使用 -O2	提升整体执行性能
内存管理优化	减少 malloc 使用	降低内存碎片和分配开销
全局变量优化	封装、替换为局部变量	减少耦合、提升可维护性

5. 团队分工情况及里程碑设置

本项目由祝彦翔独立完成，包括项目的思路构建，三个版本代码的优化，代码改错及报告生成等工作。

这里采用模块化设计方式迭代开发，将系统拆分为独立模块，接口清晰。数据结构多使用结构数组，充分发挥 C 的编程优势。在主代码中也尽量将不同的功能拆分到了相应的函数中去，让代码结构整体更清晰。

里程碑设计：

1. 思路构建，从串行向并行不断进化，不断更新软件版本
2. 修改期中项目的代码，加入期末要求，实现串行锁步法
3. 改为 A/D 相互并行法，实现逻辑
4. 改为高度并行化仿真，实现逻辑
5. 修改内部程序问题
6. 撰写报告等

6. 软件测试方法含白盒测试说明，软件失效边界条件

我使用了 5 种不同的测试用例作为白盒测试，运行编译出来的可执行文件后得到三种不同的代码版本下的运行时间。其中 5 组白盒测试的时间 Tsim 均为 1000.0ms，构造长时间验证证实可行性及减小随机误差。

三种方法下运行 5 种白盒测试结果如下：

白盒测试结果/s

	Method1	Method2	Method3
dT=0.08, Vth=0.2, Tsim=1000.0, tA=8	113.3127	92.7469	49.9383
dT=0.06, Vth=0.2, Tsim=1000.0, tA=8	147.8556	110.8364	59.8554
dT=0.08, Vth=0.1, Tsim=1000.0, tA=8	114.3553	94.4532	44.5944
dT=0.08, Vth=0.03, Tsim=1000.0, tA=8	115.1708	92.3719	41.0342
dT=0.08, Vth=0.01, Tsim=1000.0, tA=8	114.7685	90.3283	39.5906

图表 4 白盒测试用时

可见在串行锁步式改为 A/D 相互并行下，速度的提升明显，但是由于 A 事件的生成仍然在很大程度限制仿真的速度，所以明显仍有很大提升的空间。将 A 事件与 D 事件再进行各自的并行处理后，很明显整体速度进一步翻倍，达到非常好的效果。

而边界失效条件上，考虑有 Tsim 和线程数两个因素。在本项目中 Tsim 最大使用过 1000.0ms 进行测试可成功，当 Tsim 过大可能超过项目中设定的结构数组最大范围，导致数组越界，

但是除非 Tsim 非常大这种情况基本不会发生。另外考虑到不同的机器下线程的分配条件是不同的，在 eda 虚拟机中使用（本项目中设定的）12 线程是绰绰有余的，如果运行是在一些老机器上可能需要调小线程数，但必须保证要大于 3，不然第三种方法无法使用，只能使用前两种仿真方法。

7. 个人收获

本次项目我从设计到完成都参与了，收获颇多。在完成过程中，遇到了很多问题，包括仿真架构的设计方法和代码的具体细节 bug 都有遇到，但是在助教和同学的帮忙下都顺利解决了。在最后迭代出的最终版本下也跑出了非常满意的仿真速度。所以最好的学习是通过实践完成的，通过这个契机让我能有体会到了独立开发完整软件的一个流程。