数字集成电路设计自动化——dotproduct 实例综合设计

一、项目描述

本项目围绕 dotproduct 点积运算程序展开，旨在实现其在硬件层面的综合设计。设计内容涵盖数据流控制、调度策略、控制逻辑以及硬件结构的搭建，并完成仿真验证。

```
void dotproduct(int a[], int b[], int c[], int n)
{
    int i;
    for (i = 0; i < n; ++i)
        c[i] = a[i] + 2 * b[i];

    for (i = 0; i < n; ++i)
        c[i] = c[i] * (a[i] + 5 * b[i]);
}
```

项目主要目标包括：

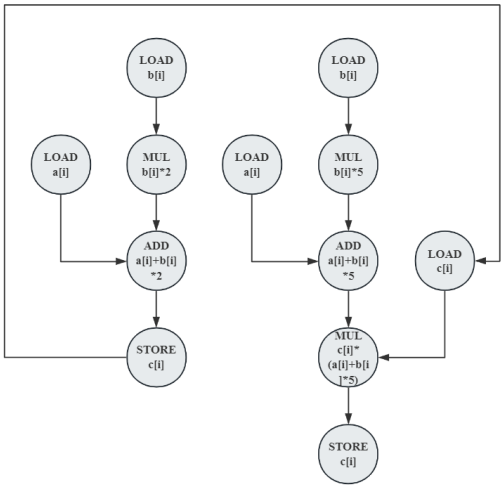
最小资源配置下的硬件实现：系统仅包含一个乘法器、一个加法器及一个读写模块（Load/Store Unit），每类操作每周期仅允许执行一次；

多资源并行配置下的性能优化：系统配备两个乘法器、两个加法器及两个读写模块，实现循环展开与并行计算；

硬件资源调度与控制策略研究：探索循环展开、循环合并、资源共享等设计空间优化方式。

二、数据流控制与调度

程序主体由两个嵌套循环构成，假设输入数组 a[]与 b[]已预存于 SRAM 中（SRAM 写入方式详见下节）。为简化建模，假设整数加法与乘法、以及 SRAM 读写操作均可在一个时钟周期内完成。



在最小资源配置下，由于 LOAD/STORE、MUL 和 ADD 模块每周期仅能使用一次，系统需进行有效的数据调度与流水安排。关键路径选择上，采用“先读 b[i]，再读 a[i]”的顺序以提升数据使用效率。

在未进行循环合并的前提下，两个循环依次执行。第一层循环需将中间结果 c[i] = a[i] + 2*b[i]写回 SRAM，第二层循环则读取 c[i]并进一步计算 c[i] = c[i] * (a[i] + 5*b[i])。若进

行循环合并，则中间变量可暂存于寄存器堆中，进一步节省读写周期。

三、寄存器分配

在本设计中我们构造了一个寄存器堆，包含 8 个寄存器：R1, R2, R3, R4 和 R7, R8, R9, R10。具体如下：

R1	从 SRAM 中读取的 a[i]值；
R2	从 SRAM 中读取的 b[i]值；
R3	保存 b[i]*2（第一个循环的中间值）；
R4	保存 a[i]+b[i]*2（第一个循环的中间值）；
R7	保存 b[i]*5（第二个循环的中间值）；
R8	从 SRAM 中读取的 c[i]值；
R9	保存 a[i]+b[i]*5（第二个循环的中间值）
R10	保存 c[i]*(a[i]+b[i]*5)（第二个循环的中间值）

在不采用流水线技术的前提下，两个循环在时间上无重叠，部分寄存器可复用，如 R3/R4 与 R7/R8，进一步减少硬件资源需求。

四、控制逻辑设计

本设计采用有限状态机（FSM）实现对点积运算各阶段的控制逻辑，整体流程被划分为 10 个状态（S0~S9）及一个完成信号状态（FINISH），分别对应输入读取、数据处理和结果写回等功能模块。

在硬件结构上，系统配置了一个读写模块、一个乘法模块和一个加法模块，通过时序控制依次完成以下两阶段计算：

阶段一（S1~S4）：计算中间值 c[i] = a[i] + 2 * b[i]；

阶段二（S5~S9）：基于第一阶段结果继续计算最终值 c[i] = c[i] * (a[i] + 5 * b[i])。

初始阶段（S0）负责将输入数组 a[] 和 b[] 写入 SRAM，每个时钟周期写入一个值。整个流程在每个时钟周期内顺序推进，完成每个数据点的处理后发出完成信号。

在资源受限（仅有一个乘法器和一个加法器）的条件下，整体时钟周期约为 11*n（其中 2*n 用于数据加载）。若增加硬件资源并行执行相关操作，可显著降低总周期数，提高处理效率。

五、SRAM 设计

SRAM 专门设计为项目的一个模块，实现对于 SRAM 数据的读与写操作。在这个项目中，我假设了 a[]和 b[]的数组长度不超过 100（根据实际应该场景可以调整这个设置）。SRAM 的宽度是 32bit 对应 int 的类型，长度为 300，其中前 100 个存储空间用来存 a[]，中间 100 个用来存 b[]，最后的 100 个用来存 c[]。

其中，a[]和 b[]的保存我单独在 S0 阶段进行，每个时钟周期保存一个数据，即在运行主程序前需要花费 2*n 个时钟周期先将两个数组保存下来。不过除此之外也可以通过其他方式比如直接读入文件比如 coe 的文件类型等，直接初始化 SRAM，这里不作探讨。

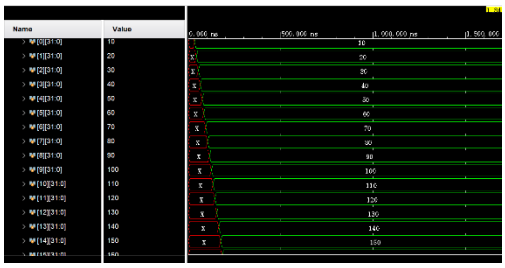
这个模块设计了 addr, din, dout 端口，所以在一个时钟周期下，LOAD/STORE 操作只能进行一次（在 Multi-Resource 部分会扩展）。

六、Min-Resource RTL 代码与结果分析

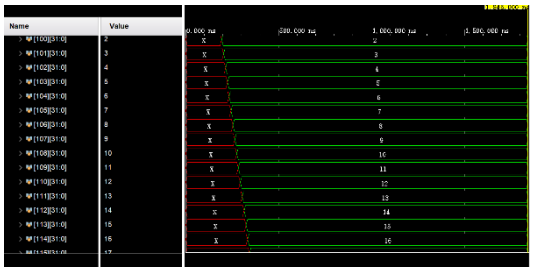
Min-Resource 分为四个模块（Top, Datapath, Controller, Sram）及一个 testbench 组成。

其中 Top 控制数据通路和控制系统，让两个模块之间的信号相互调度控制。Sram 在 Datapath 内实例化，专门控制 SRAM 数据的读写。在 Datapath 内部定义了寄存器堆，主要实现的是在接受到传入的 LOAD/STORE/MUL/ADD 等使能信号后，对寄存器堆及 SRAM 的数据操作。而 Controller 是控制上述的 S0~S9 的状态机，包括组织循环结构等。

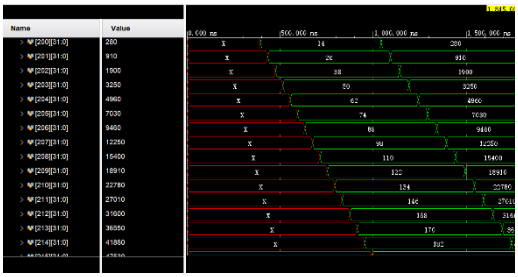
在 testbench 中，我们使用了 16 个数据的 a[]和 b[]数组进行测试，测试的波形图如下：



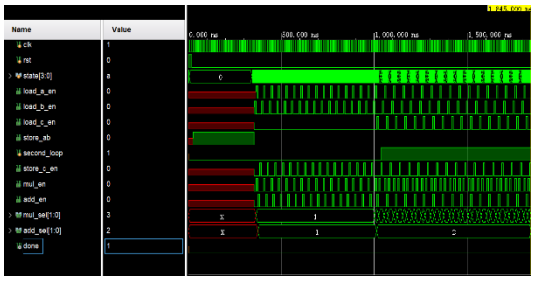
输入的 a 数组存于 SRAM 的 0~99 行内



输入的 b 数组存于 SRAM 的 100~199 行内



c 数组存于 SRAM 的 200~299 行内



State 为状态机的运作，其它为 load/store/mul/add 等信号的使能，由 controller 向 datapath 传递进行控制

由测试结果可见，c 数组的 SRAM 数据改变了两次，且每个时钟周期下存入一个数据。第一批存入的数据是第一个循环的 c[]结果，由于没有循环合并，寄存器堆并没有存下所有 c[]中间值的能力，所以需要存入 SRAM 内；第二批存入 SRAM 的是 c[]的结果，以备后续代码对 c[]可能的调用。而 controller 的各种调度信号也可见上图，这些信号会进入 datapath 控制不同的计算或是读写操作。

七、Multi-Resource RTL 代码与结果分析

本设计采用有限状态机对点积计算的全过程进行统一控制。控制器根据不同计算阶段，将整个操作流程划分为 10 个状态（S0~S9）及一个结束状态（FINISH），涵盖输入读取、数据处理、结果写回等主要环节。在基础模型中，系统配置了一个读写模块（Load/Store）、一个乘法器和一个加法器。其控制逻辑按阶段依次完成两部分计算。

为提升性能，系统进一步扩展为多资源并行结构（实现循环展开），配置 2 个读写模块 2 个乘法器和 2 个加法器。控制器的状态划分仍保持为 10 个阶段，但其内部控制逻辑进行了优化，以支持两个数据通路同时工作，实现双通道并行循环展开。

并行策略具体体现为：

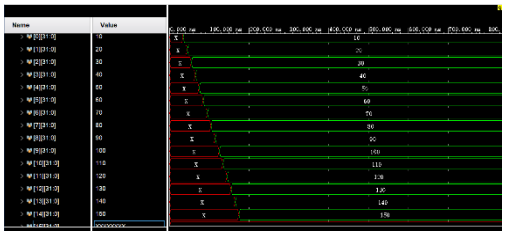
输入阶段（S0）：利用双读写模块，每个时钟周期可同时写入 a[i] 和 a[i+1]、b[i] 和 b[i+1]，从而将 SRAM 初始化所需时钟周期降为 n（原为 2n）；

计算阶段（S1~S9）：每 9 个周期可同时完成两组 c[i] 和 c[i+1] 的计算过程，实现吞吐量翻倍（在 n 为偶数，平均单数据计算时钟周期降为 4.5n；n 为奇数的情况下，n 逐渐增大下时钟周期也趋于 4.5n）；

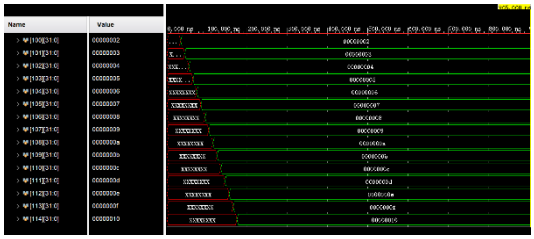
因此，在多资源配置下，包括 a，b 数组数据存入 SRAM，总的运算时钟周期约为 5.5*n。

与单资源结构相比，总体运算时间缩短了近一半，极大地提高了系统的运行效率。

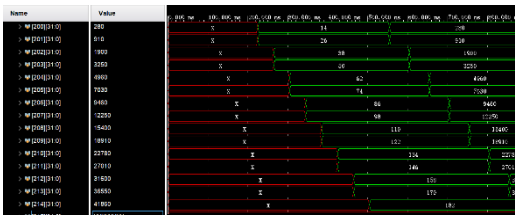
下面是 Multi-Resource 的仿真测试，可以看到对应位的 a 数组和 b 数组同时存入 SRAM 内，c 数组在 SRAM 内会有两次数据改变，第一次是第一个循环的中间数据，第二个是最终的结果。从波形图中可见，循环展开被实现：



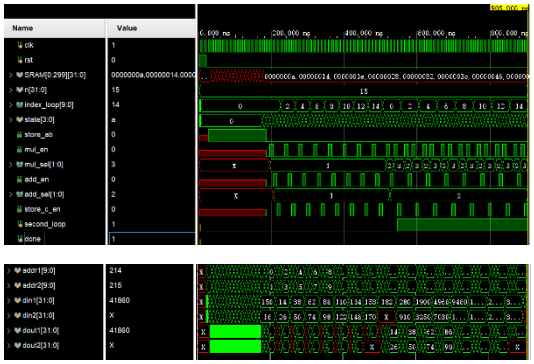
输入的 a 数组存于 SRAM 的 0~99 行内



输入的 b 数组存于 SRAM 的 100~199 行内（与 a 数组的对应位置同时存储）



c 数组存于 SRAM 的 200~299 行内（第一次数据是第一次循环下的结果，第二次数据是最终计算结果），可见结果两个一组在同一时钟周期内生成



控制逻辑，由控制器生成对数据通路进行存算控制

Multi-Resource 例下，演示了 2 组资源的并行计算方案。在硬件条件允许的情况下，可进一步扩展为 4 组、8 组或更多，构建更大规模的并行计算结构。在控制器设计上，通过引入循环调度、流水线（pipeline）控制及资源动态分配等技术，亦可进一步提升系统性能。

八、总结

这个项目是数字集成电路设计自动化的课程项目，由祝彦翔完成。

本文基于点积运算（dotproduct）任务，完成了从最小资源到多资源并行结构的完整硬件设计流程。通过对 FSM 状态控制、数据流调度、寄存器分配及 SRAM 模块进行精细设计，成功实现了点积计算在硬件中的高效执行。

在最小资源结构下，系统能在保证功能正确的前提下，以较低硬件开销完成任务，适用于资源受限的嵌入式系统；而在多资源配置下，通过循环展开与双通道并行机制，大幅减少了计算周期，适用于对性能要求更高的计算场景。

本设计展示了数字集成电路自动化设计中的关键路径优化思路，验证了控制逻辑与数据通路协同设计在算术计算任务中的应用潜力。

九、附件

项目的源码可见 github 仓库 https://github.com/Yanxiang-ZHU/C-Language-Control.git，包括两种方法下的 verilog 实现代码，详细 readme.md 解释文件及仿真波形。