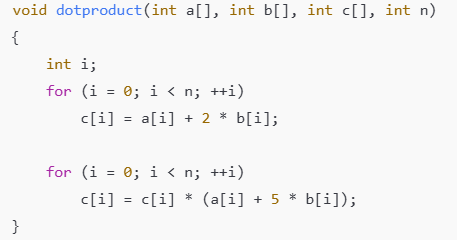
**数字集成电路设计自动化——dotproduct实例综合设计**

# **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。



项目主要目标包括：

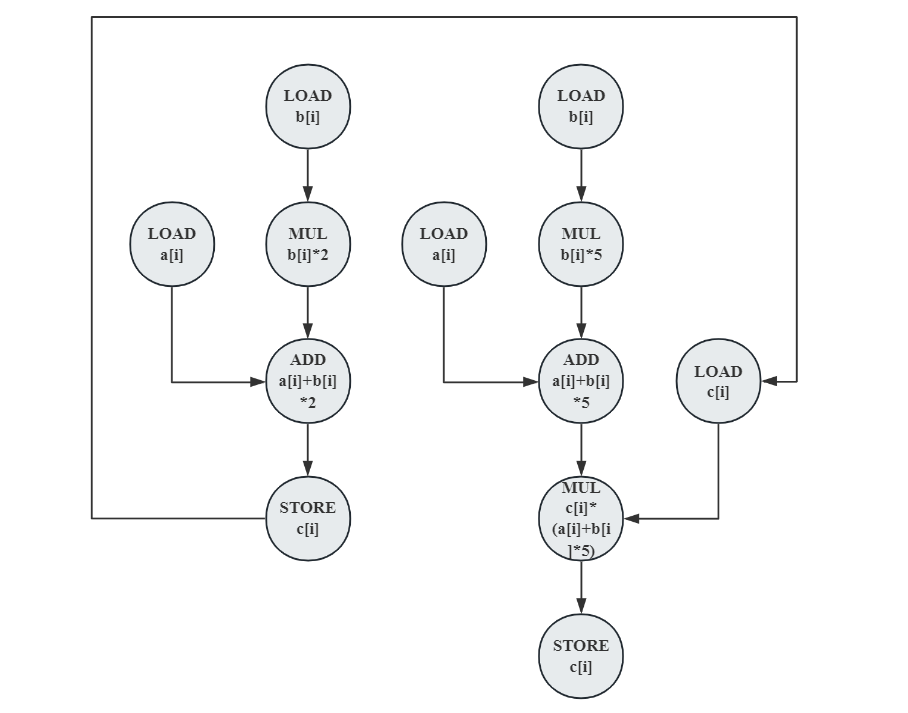
最小资源配置下的硬件实现：系统仅包含一个乘法器、一个加法器及一个读写模块（Load/Store Unit），每类操作每周期仅允许执行一次；

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。

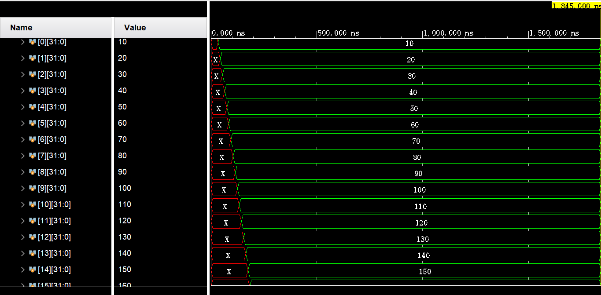
这个模块设计了addr, din, dout端口，所以在一个时钟周期下，LOAD/STORE操作只能进行一次（在Multi-Resource部分会扩展）。

# **六、Min-Resource RTL代码与结果分析**

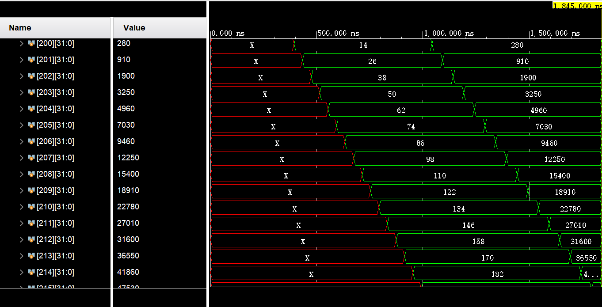
Min-Resource分为四个模块（Top, Datapath, Controller, Sram）及一个testbench组成。

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。

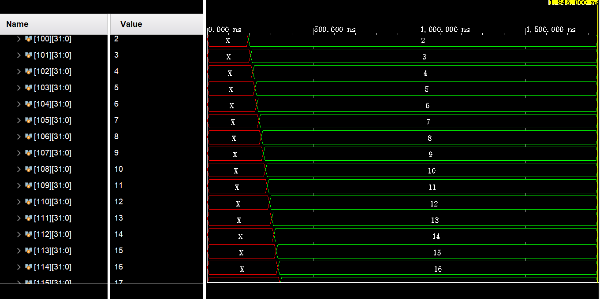
在testbench中，我们使用了16个数据的a[]和b[]数组进行测试，测试的波形图如下：



输入的a数组存于SRAM的0~99行内



c数组存于SRAM的200~299行内



输入的b数组存于SRAM的100~199行内

电脑萤幕画面

AI 生成的内容可能不正确。

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：

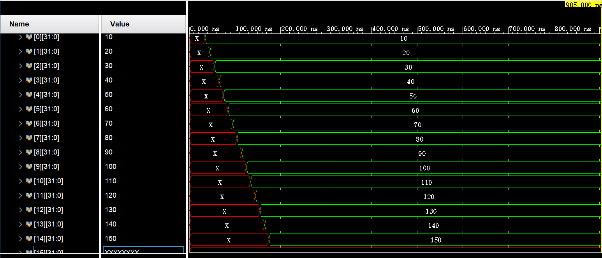
输入阶段（S0）：利用双读写模块，每个时钟周期可同时写入 a[i] 和 a[i+1]、b[i] 和 b[i+1]，从而将 SRAM 初始化所需时钟周期降为 n（原为 2n）；

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*

*图片包含 日程表

AI 生成的内容可能不正确。*

*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*

*图片包含 图形用户界面

AI 生成的内容可能不正确。*

*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）*

*图形用户界面

AI 生成的内容可能不正确。*

*图形用户界面

AI 生成的内容可能不正确。*

*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。