DASS: Combining Dynamic & Static Scheduling in High-Level Synthesis

Source: IEEE Trans on CADICS, 03/2022

Last Modified: 06/01/2023

0. Abstract

调度就是安排操作的时钟周期。

（1） 静态调度：在编译时决定，电路简单且资源可复用；

（2） 动态调度：在运行时决定，运行更快但控制复杂。

将动态调度无法提高性能表现的部分用静态调度实现，并将静态调度部分视为黑盒来设计动态调度部分。

1. Introduction

对静态调度来说，寻找可并行的操作可以降低延迟，并且调整操作的开始时间可以实现资源复用以减小面积。但在控制路径和延时变化的操作上，静态调度要采取保守的策略，因为这些信息在编译时是未知的。

对动态调度来说，具有握手信号可以在延时变化的操作上取得更低的时钟周期。但是握手信号可能造成较长的关键路径，进而降低时钟频率。同时，动态调度难以进行资源复用以及握手电路的存在，使得动态调度的面积消耗较大。

通过结合两种调度方式，通过用户定义来使得控制流简单且延时固定的部分通过静态调度实现。这些静态调度的部分可以被当作黑盒来放入动态调度中。

本文主要解决了两个挑战：

（1） 如何将静态调度部分结合进动态调度的外围电路中；

（2） 如何使得静态调度和动态调度可以正确且高效的进行内存共享。

2. Overview

静态调度具有较低的面积和性能。一个静态调度的硬件可以包括一组存储和一个有限状态机来监控和控制操作来实现资源共享。对于启动间隔来说，考虑到数据间依赖关系，必须采用保守的策略来编排时间。

动态调度具有较高的面积和性能。只要输入数据有效，动态调度就可以立即开始计算。

两种调度混合可以实现较低的面积和较高的性能。甚至考虑到时钟频率，可以取得更高的性能表现。

3. Background

3.1 Scheduling in HLS

多数 HLS 工具流包括两个部分：

（1） 前端：将源代码编译为一种中间表示；

（2） 后端：将中间表示表示变为电路实现。

调度过程在将中间表示变为控制/数据流图时进行。流图包括两层有向图：

（1） 顶层：通过控制流图表示，顶点是中间表示的基础块，边是控制流；

（2） 底层：通过数据流图表示，子顶点是数据操作，子边是数据依赖。

3.2 Static scheduling

编排是将控制/数据流图做时间编排。静态编排决定了每个操作的开始和结束时钟周期，同时也考虑到资源消耗。但是由于静态编排考虑的是最差的时序场景，所以显示了整体吞吐率和可达到的性能表现。

3.3 Dynamic scheduling

最早的动态编排工作自动化了将并行程序放在一个同步硬件网表中，但是需要手动设计和部署硬件优化。

之后的工作将程序放到异步电路中。对每个数据流图设计硬件节点并进行流水线处理。每个节点有自己的控制触发器，接着在同步电路中进行设计。

近期工作可以将任意输入代码转化为同步电路。利用硬件的并行性和握手信号完成动态编排来达到高吞吐率。本文使用该工作来实现动态编排的硬件。

动态编排的数据流可能包含多个数据流部件：

（1） 合并：从多个上级电路得到数据合并发送到下级电路；

（2） 分叉：从上级电路得到数据将其复制发放到多个下级电路；

（3） 加入：在输入全部有效时将其输出；

（4） 分支：根据输入数据将上级数据发放给不同夏季电路。

编排内存读写是动态编排的难点。在动态编排中，如果数据冲突没有解决，正确性和性能可能受到影响。读写队列是一种解决数据依赖并编排任意内存读写的方法，但是它消耗了大量的面积并添加了延迟。在本文设计中，内存架构可能包括了部分编排好的内存读写和未编排的内存读写。

3.4 Combining dynamic and static approaches

目前已有设计通常具有较高的限制条件，不能在一些复杂内存读写场景下进行应用。本文通过将静态模块打包放入延迟不敏感系统中来混和两种编排方式。

4. Methodology

4.1 Applicability of our approach

该方法最好用于满足以下条件的设计中：

（1） 利用运行时信息可以更好地优化吞吐率；

（2） 至少一个区域具有较为固定地延时；

（3） 上述区域可以进行资源共享。

4.2 Integrating SS Hardware into DS hardware

本节介绍了静态编排电路的包装器，来使得静态编排电路和动态编排电路可以连接到一起。在这个包装器中，我们支持并希望启动间隔大于 1。

对于动态编排电路来说，通常包含以下控制接口：

（1） pValid：输入信号，上一级输入信号有效；

（2） valid：输出信号，告知下一级电路现阶段结果有效；

（3） nReady：输入信号，下一级信号已经准备好接受下一个信号；

（4） ready：输出信号，告知上一级电路已经准备好接受下一个信号。

对于静态编排电路来说，通常包含以下控制接口：

（1） ap_ce：时钟使能信号；

（2） ap_ready：准备好接受下一个信号；

（3） ap_vld：输出有效。

因此包装器需要两个部分：

（1） 一个有效信号来发送信号给下一级电路并在下一级地反向压力时保护输出信号；

（2） 一个准备信号来将反向压力发送给上一级电路。

4.2.1 Constructing the valid signal

对静态电路来说，编排策略有两种：

（1） 暂停操作直到输入信号有效；

（2） 持续操作并标记无效输出。

考虑到暂停可能会损失性能，采取第二种策略在牺牲功耗的情况下优化性能表现。

使用一个移位寄存器来标志数据的有效性。每当接受一个新的输入，移位寄存器右移一个比特，这由 ap_ready 决定。输入有效信号被存进移位寄存器中，即 pValid 信号。移位寄存器的深度由延迟和启动间隔共同决定。而移位寄存器的输出信号和 ap_vld 共同决定了输出数据的有效与否，即决定了 valid 信号。

4.2.2 Constructing the ready signal

对于给到前一级的反向压力，直接将 ap_ready 连接到 ready 即可。

对于来自后一级的反向压力，需要在静态编排电路输出有效时暂停，即在 valid 为 1 且 nReady 为 0 时将时钟使能信号 ap_ce 置为 0。

4.2.3 Handling multiple inputs and multiple outputs

对于多输入情况来说，通过类似于动态编排电路中的加入操作使得输入信号同时输入到静态编排电路中。

对于多输出电路来说，每个输出具有独立的握手信号。对于输出有效信号 valid，每个均为将静态电路的输出有效 ap_valid 和移位寄存器的最后一位进行与操作来获得。对于 nReady 来说，每个下一级的未准备与对应输出有效相与都可以暂停时钟使能信号 ap_ce。

4.3 Shared memory between SS and DS circuits

对于动态编排电路来说，读写队列可以支持不寻常的内存读写。但在不会产生数据冲突的情况下，也可以通过直连到一个仲裁器作为内存控制器上。

将读写操作视为节点，读写队列和内存控制器视为部件，存储资源视为存储着向量的块。节点可以组成节点集，每个节点集必须在一块硬件区域中。对于包含了不可预测的内存读写的静态编排函数，顶层不能流水线实现。这是由于动态编排的外围电路忽略了静态编排电路内部的数据依赖。

节点集的划分规则为：

（1） 直连到一个内存控制器是一个节点集；

（2） 共享同一个读写队列是一个节点集。

对于动态电路来说，每个节点一次只会读写数据一次；对于静态电路来说，一次可能读写多个数据。如果一个使用读写队列的节点集是静态编排的，就不再需要读写队列了。

4.4 Constructing shared memory interface in DASS

对于静态编排电路来说，可以直接通过仲裁器连接到内存接口上；对于动态电路来说，需要内存控制器来转换握手信号以连接到内存接口上。仲裁器通过轮询调度在冲突时将数据提供给静态编排电路和动态编排电路。对于仲裁器的优先级，静态编排电路永远高于动态编排电路。

5. Implementation

首先根据用户定义划分静态编排和动态编排的部分，对于没有指定则默认动态编排。对于有迭代内依赖的情况则采用级联的编排策略。Vivado HLS 的结果静态编排电路会被添加包装器来获得握手信号。对静态编排的内存来说，如果有共享则通过仲裁器连接到内存，如果没有则直接连到内存。

对于动态编排，通过将已有的静态编排模块和其启动间隔和延迟作为黑盒与其余动态编排模块一起生成最终 RTL 设计。