Spade：一种基于表达式并支持流水线的硬件描述语言

Frans Skarman 和 Oscar Gustafsson 电气工程系，
林雪平大学，瑞典林雪平 SE-581 83 电子邮箱：
{frans.skarman,oscar.gustafsson}@liu.se

arXiv:2304.03079v1 [cs.AR] 6 Apr 2023

摘要——**Spade 是一种新的开源硬件描述语言（HDL），旨在提高开发者的工作效率，同时不牺牲硬件描述语言所提供的底层控制能力。它是一种独立语言，借鉴了现代软件语言的设计理念，并为常见的硬件结构增添了实用的抽象机制。它还配备了一套便捷的工具，包括功能强大的编译器、支持依赖管理的构建系统、调试工具以及编辑器集成。**

索引词——**硬件描述语言、语言与编译器、设计自动化**

I. 引言

数字硬件的开发传统上使用 Verilog 或 VHDL 完成，这两种语言都起源于 20 世纪 80 年代。尽管它们自诞生以来不断发展，但仍缺少语言设计领域后来出现的许多进步。

Spade[1] 是一种新型的开源硬件描述语言（HDL），旨在借鉴软件语言的思想，并加入面向常见硬件结构的语言级机制，从而减少数字硬件的开发工作量，同时保留硬件描述语言所提供的底层控制能力。

Spade 受到 Rust 和函数式编程语言的启发，采用基于表达式的设计，并拥有丰富的类型系统。它支持结构体和元组等积类型，以及以枚举形式呈现的和类型。该语言还支持线性类型检查，可用于确保内存端口等硬件资源恰好被使用一次。

Spade 内置了针对流水线、存储器和寄存器等常见硬件结构的构造和抽象。流水线化允许用户指定要执行的计算，并通过显式语句分隔流水线的各级，而无需为每个流水线级分别定义变量。对这样的流水线进行重定时无需更改任何变量，只需移动分级语句。此外，流水线的延迟在语言中是显式的，并由编译器进行检查，以确保更改不会流水线不会影响计算结果。为更准确地反映所描述的硬件，默认情况下，所有逻辑均为组合逻辑，唯一的时序元件是显式实例化的寄存器和存储器。

本文其余部分安排如下：第II节介绍相关工作，并重点说明 Spade 的不同之处。第III节介绍基本语义；第IV节介绍如何使用线性类型对输入和输出端口建模。第V节讨论为 Spade 提供的软件，最后在第VI节作总结。

II. 相关工作

近年来，出现了若干新的硬件描述语言。一种常见做法是将该语言作为领域专用语言（DSL）嵌入 Scala、Python 或 Ruby 等传统软件编程语言中。在这种方式下，硬件描述语言由一组硬件构造库组成，用户在宿主语言中实例化这些构造，以描述硬件。这样，用户便能在硬件描述中利用宿主语言的强大功能，例如使用软件控制流结构生成参数化硬件，或在硬件描述中采用面向对象或函数式编程方法。此外，这种方式还让硬件设计者能够使用宿主语言的所有工具，例如依赖管理器、构建工具和集成开发环境。此类语言包括嵌入 Scala 的 Chisel [1], SpinalHDL [2] 和 DFiant [3] 、嵌入 Python 的 Amaranth [4] 、嵌入 Dart 的 ROHD [5], 以及嵌入 Ruby 的 RubyRTL [6] 。

一些语言采用传统编程语言的子集，并将其编译为硬件。其中一个例子是 Clash [7], 它利用纯函数式语言与硬件之间的自然映射，将 Haskell的一个较大子集编译为硬件。另一个例子是 PipelineC [8] ，它是一种带有自动流水线化功能、类似 C 的硬件描述语言。虽然它并不是常规的C，但与 C足够相近，因此许多 PipelineC 程序可以由标准 C 编译器进行编译和“模拟”。

TL-Verilog [9] 是一个SystemVerilog扩展，支持时序抽象建模，其中行为和时序是分离的。它为流水线和FIFO等常见硬件构造提供语言特性。

此外，还有几种完全独立于现有语言的语言。例如，Bluespec [10] 通过受保护的原子操作来描述硬件。另一个例子是Silice [11], 它包含

[1]https://gitlab.com/spade-lang/spade/

本文发表于*2023*年第三届开源设计自动化研讨会 *(OSDA)*，该研讨会与2023年欧洲设计、自动化与测试会议 *(DATE)*联合举办，地点为比利时安特卫普，时间为2023年4月17日。如需了解更多信息或获取相关材料，请访问研讨会网站上的论文网页：**https://osda.ws/r/2QkTi** （扫描右侧二维码即可）。
版权归作者所有。本文采用知识共享署名许可协议 (CC BY) 4.0 版授权，详情见 https://creativecommons.org/licenses/by/4.0。第三届开源设计自动化研讨会 *(OSDA)* 论文集，第7–12页，2023年。

常见硬件结构的抽象，同时又不失对所生成硬件的控制。此外，它还提供了一种更高层次的描述风格，可将设计表达为一系列操作，并采用类似软件的控制流。

最后，描述硬件的一种常见替代方案是高级综合（HLS）：将通常为软件设计的高级语言编译成硬件。这种设计方法与前文所述的硬件描述语言截然不同。具体而言，高级综合工具通常无法对生成的硬件进行精细控制。尤其是，电路的精确时序通常会被抽象掉，而高级综合生成的模块之间则通过同步信号在运行时进行同步。

现在，一个自然的问题是：是否有必要再推出一种硬件描述语言？Spade 又提供了哪些现有工作所不具备的特性？首先，Spade 并非嵌入在另一种语言中的领域专用语言，因此有别于 Chisel 和 Amaranth 等语言。这使语言设计拥有更大的自由度，不受领域专用语言表达能力的限制。例如，大多数嵌入式硬件描述语言都不得不创造新的“关键字”，以避免与宿主语言冲突；典型例子是 Chisel 使用 when 和 otherwise，而不是更熟悉的 if 和 else。然而，这个问题不止于表层关键字。例如，Scala 支持模式匹配，但在 Chisel 和 SpinalHDL 中，这一特性只能在“构建时”用于 Scala 值，而不能在硬件中用于“运行时”值。最后，潜在用户初次学习这门语言时，无需同时学习宿主语言和领域专用语言。

与Clash和 PipelineC 不同，Spade 不受限于遵循宿主语言的执行模型，这意味着该语言可以直接针对硬件描述进行设计，不受任何限制。

Spade以逐周期的方式描述电路的行为，相较于 Bluespec（其基本抽象是原子规则）以及 PipelineC（其编译器会自动执行流水线化），提供了更多的控制能力。

最后，Spade 在理念上类似于 TL-Verilog。然而，通过设计一种新语言，而不是基于 Sys- temVerilog 构建，可以更自由地探索新的语言结构。

III. 基本语义

为了描述 Spade 的编程模型和语义，这里使用了一个示例。清单 1展示了 Spade 代码，该代码会以可配置的间隔让发光二极管闪烁。第 1–2 行定义了一个实体 ，名为 blink，它接受一个类型为`clock`的时钟信号、一个类型为`bool`的复位信号，以及一个 20 位整数`int<20>`，用于指定计数器的最大值。返回值是一个驱动 LED 的单个`bool` 值。实体 关键字表示该单元 可以包含时序逻辑。也可以编写函数`fn`，它只允许使用组合逻辑；还可以编写流水线 来定义一个流水线。流水线的细节将在后文讨论。实例化单元时，函数、实体和

清单 1. 让发光二极管闪烁的 Spade 代码 1 实体 blink(clk:
clo
res
cou

流水线，向代码的读取器表明实例化背后可能发生什么。

第 4 行定义了一个名为 counter的寄存器，该寄存器由clk信号驱动，并由 rst信号复位为0。Spade中的寄存器是显式构造，而不是从使用诸如上升_沿之类的内容中推断出来的。寄存器与对存储器的写操作是该语言中仅有的时序构造，其他所有内容都描述组合电路。

在 Spade 中，电路的行为是按时钟周期描述的。设计中寄存器的新值由当前时钟周期内的寄存器值构成的表达式给出。Spade 中的所有变量都是不可变的：只能赋值一次。由于 Spade 是一种面向表达式的语言，因此可以只赋值一次；这意味着大多数语句都是表达式，并会产生值 [12]。例如，计数器 寄存器被设为第 4 行等号后表达式的值；在本例中，该表达式是一个跨越第 5–10 行的 if 表达式。如果计数器已达到最大值，if 表达式返回的值为 0，否则就是计数器的下一个值。与传统的命令式硬件描述语言相比，不可变变量和基于表达式的控制流更贴近最终生成的硬件：变量对应导线，而“控制流”则通过多路复用器实现。该语言也没有显式的 return 关键字；单元体中的最后一个表达式就是实体的输出，如第 11 行所示。

对 `trunc` 函数的调用在第 9 行将 (计数器 + 1) 的值从 21 位截断为 20 位。这是必要的，因为 Spade 会在大多数算术运算中使用可能的最大值，以防止意外溢出。有关类型系统的更多详情，请参见第 III-B

Spade具有与大多数现代软件语言相似的作用域规则：变量仅在其定义之后可见，这使得意外创建组合逻辑循环变得更加困难。此外，这些作用域规则使开发者更容易找到变量的定义。定义总是位于其使用位置之前，并且通常与其赋值归为一组。不过，有时创建变量之间的依赖循环仍然很有用，而这可以使用 decl 关键字显式完成。

A. 流水线

流水线化是大多数硬件设计中的重要构造，它使设计能够以延迟为代价保持较高的时钟频率和吞吐量。然而，尽管流水线化十分重要，大多数硬件描述语言都要求用户手动构建

列表 2。Spade 中流水线化构造的示例。

```
1  流水线 (4)      X(clk:   时钟 ,   a:  int <32>,  b:  int <32>)
2      -> 整数 <33>
3  {
4          'initial
5          let x = 实例 (3)    子流水线 (时钟 ,  a);
6          let p = a * b;
7      reg * 3;
8          let s = x + f(a, p);
9      reg;
10         s  +阶段( initial ).a
11 }
12
13 流水线 (3) 子流水线 (...) -> 整数 <32>{...}
```

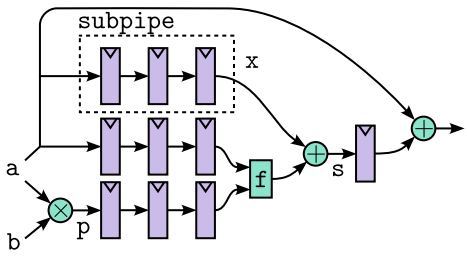


图1。由流水线中的硬件描述x 由清单中的代码 2。

它们的流水线。这一过程既繁琐又容易出错，因为必须确保计算所使用的值对应正确的时间步。在某些情况下，设计人员会在变量名称中采用特定模式，并使用临时的静态检查工具来验证流水线化是否正确 [13]。

另一方面，Spade在语言层面支持流水线描述，用户可以描述每个阶段执行的计算，而编译器负责管理阶段之间寄存器的插入。清单 2 展示了使用流水线化构造来描述图中硬件的示例。1。与之前讨论的 实体一样，前两行描述了流水线的外部接口：名称、输入和输出。它还指定了深度，在本例中为4，即流水线中的寄存器数量，因此也是输入与输出之间的延迟。虽然编译器可以推断深度，但由于它属于公共 API，因此必须手动指定。这使用户无需阅读主体即可查看流水线的接口。

第4 行命名当前流水线级 `initial`，允许稍后在代码中引用它。第 5 行使用 `subpipe` 实例化了另一个名为实例的关键字。在实例化流水线时，必须指定流水线的深度，编译器会将其与被实例化流水线头部中指定的深度进行检查。如果稍后更改被实例化流水线的深度，编译器会报错，迫使开发者考虑该更改是否会影响实例化代码。编译器还会检查子流水线中的变量是否可用；如果在结果准备好之前使用该结果，则会报错。例如，如果 `x` 在其定义之后紧接着的流水线级中被使用，就会产生示例清单3中所示的错误消息。

第 6 行定义了一个名为 p 的新变量，它是 a 和 b 的乘积，并且根据类型推断，其类型为 int<64>。接下来是一个 reg 语句，它的行为不同于 the

列表 3。流水线错误消息示例错误： 在 x 准备好之前使用它 -->

之前描述的实体中的reg语句。在流水线中，reg语句会告知编译器插入一个流水线级，为其上方可见的所有值进行寄存。第7 行的 reg 语句之后，任何对 p 的引用都会指向该值的延迟版本。subpipe计算出的 x值也是如此，但是，由于它是通过一个深度为三的流水线计算得出的，前三级寄存器已存在于子流水线中，因此不会被插入实例化流水线中。寄存器上的 times three 运算符reg *3 指定应插入三级，而不进行任何新的计算。最后，在第 10行，s 被加到来自第一级的 a 值上。stage 关键字用于绕过一些流水线寄存器，并直接引用信号在另一个流水线级中的呈现形式，在本例中即名为initial的流水线级。流水线级引用既可以引用“未来”流水线级，也可以引用“过去”流水线级；既可以像此处这样按名称引用，也可以使用相对偏移量，例如 stage(-2)。

流水线功能将计算的描述与流水线本身的描述解耦。在没有反馈的流水线中，开发者可以根据需要轻松增减流水线级，而不改变流水线的输出值。如果此类更改可能影响项目其他部分的结果，那么在调用处注明深度可确保用户通过编译器错误消息了解这些潜在问题。对于带反馈的流水线，即便需要一些手动操作来确保计算正确，对流水线阶段进行结构化描述仍有助于设计迭代。

B. 类型和模式匹配

Spade 是一种具有类型推断的静态类型语言。这意味着大多数代码中都可以省略类型，因为编译器会根据上下文推断出适当的类型，并在无法推断类型时报告错误。

与大多数语言一样，Spade 具有整数和布尔值等基本类型，以及数组、元组和结构体等复合类型。此外，该语言还支持受 Rust、Haskell 和ML 等软件语言启发的枚举类型。不同于其 C 或 VHDL 中的同名类型，这些枚举除了属于一组变体中的一个之外，还关联有数据。这种构造的一个常见用例是`Option`类型，它在标准库中的定义如清单4所示。它对所包含的类型`T` 是泛型的，并且取两个值中的一个：`Some` ，此时存在一个类型为`val`的值`T` ；或者是`None` ，此时不存在这样的值。可以将`Option` 类型看作一个与其验证的数据捆绑在一起的有效/无效信号。

在 Spade 中与枚举交互的主要方式是 match 表达式，它允许对值进行模式匹配。作为一个

列表4。Option 类型的定义。

```
> = ...; let b: Option <T> = ...; let
result = match(a, b) {(Some (val), _)
```

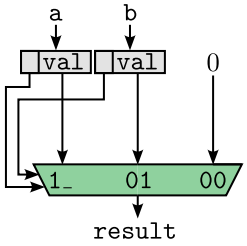


图2. 清单生成的硬件5。

示例，清单 5 展示了match-表达式在一个由 Option-值组成的元组上的使用方式： a 和 b。生成的硬件如图 2所示。如果 a 是 Some，其内部值会绑定到 val ，并从 match 表达式中返回。如果 a 是 None ， 但 b 是 Some，则返回其内部值。最后，如果 a 和 b 都是 None，则返回 0。枚举被编码为用于指定当前激活变体的位，即判别位，后面跟着包含负载的位。在本例中，由于该枚举包含两个变体，判别位只有一位，而其余位在存在时包含 val ， 否则未定义。

C. 存储器

如前文所述，Spade 要求寄存器定义包含一个表达式，用于根据上一个时钟周期的值计算整个寄存器的新值。然而，在处理类似存储器的结构时，这种抽象会产生问题，因为每个周期只会更新总状态中的一小部分。为缓解这一问题，Spade 为存储器提供了显式构造，目前通过标准库中定义的实体实现，编译器会对其进行单独处理。存储器通过 clocked_memory 实体实例化，该实体会创建一个具有固定数量写端口的存储器。通过类似定义的 read_memory 实体从该存储器中读数据。这意味着，与 VHDL 和 Verilog 不同，存储器会被显式实例化为存储器，而不是根据代码结构推断出来。下一节将给出一个使用存储器的示例。

IV. 端口

到目前为止，讨论一直围绕对值的计算展开。一个单元接收一组值作为输入，并生成一组值作为输出。在内部，值用于计算，并通过流水线进行流水线化处理。当处理存储器这类位于当前单元之外的对象时，这种方式就变得不太方便。该单元必须向存储器返回一个地址、一个写使能信号以及一个要写入存储器的值。然后，必须将从存储器读出的值作为输入传递给该单元。存储器反过来会生成一个输出值，而这个值必须反馈给控制单元。这种方法存在一些问题：控制信号和输出会受到流水线机制的延迟影响。除非手动进行缓解（例如通过分级引用），否则这种流水线化会引入额外延迟。此外，存储器控制信号与相应输出之间也没有清晰的关联。

为缓解这一问题，Spade 类型系统包含端口和线的概念。线有两种形式：可变线和不可变线，分别用 &mut 和& 表示。不可变线可用于通过单元传递值，而不会使这些值在流水线中产生延迟。可变线同样不会经过流水线处理，但允许在将该线作为输入的模块中设置线的值。最后，端口是线和其他端口的集合，它们被捆绑在一起。

代码清单 6 展示了如何使用 端口 功能，让两条流水线共享一个存储器。前几行定义了包含读、写和地址线的端口类型。第 8 行定义了一个 实体 ，其中实例化了实际的存储器。它返回一个读端口和一个写端口。为节省空间，第 9 行经过了删减，但其中实例化了可变线 r_地址和一个 WPort。第 10–11 行使用 时钟控制的_存储器 实体实例化了一个只有一个写端口的存储器；第 12–14 行则从存储器中异步读出一个值。在实体末尾，由可变地址和结果线组装出读端口，并与写端口一同返回。

流水线读取器和写入器是分别使用读端口和写端口的两条流水线。在第20行，读取器流水线设置要读取的地址，并在第21行读取所得的值。同样，写入器在第 27 行设置目标地址和写入值。

最后，在第 32–34 行实例化了这三个单元。存储器返回的端口被传递给读取器和写入器。

使用可变线时存在一些陷阱：返回可变线的单元会要求为该线设置一个值，否则该线的值可能未定义；如果只在满足条件时设置，则可能会推导出锁存器。类似地，如果一条线由多个驱动器驱动，彼此冲突的值可能会导致问题。虽然可以在仿真中发现这类问题，但最好让编译器捕获此类错误。

这一问题的解决方案受到 [14] 的启发，该方案使用仿射类型来确保高层次综合工具中的正确存储器访问模式。仿射类型可以保证一个值至多被使用一次，从而解决多驱动问题；然而，为确保所有可变线都恰好设置一次，还需要更强的线性类型概念 [15]，并在 Spade 中实现。

由于 Spade 以逐周期的方式描述行为，因此实现线性类型系统比一般情况更简单。线性类型的每个资源都必须在每个时钟周期恰好被消耗 一次。使用 set语句设置某个值，或将其传递给另一个单元（由该单元承担消耗要求）时，该值即被消耗。

线性类型检查在常规类型检查之后进行，这意味着编译器知道哪些资源属于线性类型，必须对其进行检查。对于每个产生线性类型资源的表达式，都会创建一棵树，其中叶节点表示基本线性类型，非叶节点表示复合线性类型，例如元组或结构体。任何

列表 6. 用端口在模块之间共享对存储器的访问。 1 struct port RPort { 2 a WP(enti 10 l(inst mer add port mer 写入 nne top(reac 5) 写

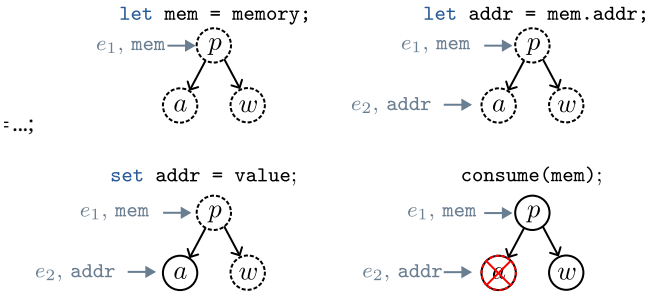


图 3. 清单7中代码的线性类型检查过程。清单 7. 图3的示例代码。1 let mem: MemPort =存储器 (); 2 let addr =mem.addr; 3 se

将资源设为别名的语句，例如表达式的let绑定或结构体上的字段访问，会创建一条从该别名指向相应树和节点的指针。

当遇到会消耗资源的表达式或语句（例如 set 语句），或将资源传递给单元时，对应于被消耗对象的节点会被标记为已消耗。如果该节点或其子节点已被标记为已消耗，则表示该资源被重复使用，系统将抛出错误。

这确保节点不会被使用多次，但不能保证它们恰好被使用一次。因此，在处理结束时，会对所有树进行最后一轮遍历，以确保每个节点都被消耗。如果遍历发现有未被消耗的叶节点，它就表示一个尚未设置的线性类型资源，并会报告错误。

为说明线性类型检查算法，图3展示了清单 7中的代码如何体现这一过程。在图中， e_x 表示在绑定到变量之前赋予子表达式的匿名名称。虚线节点表示该值未被消耗，实线节点表示该值已被消耗，带叉的节点表示该值被重复消耗。

V. SPADE 软件

一套完善的语言工具，无论是用于处理语言本身，还是与现有工具集成，都能极大地推动语言的普及。首先，Spade 编译器和这门语言从一开始就以生成实用且易于阅读的错误消息为目标，例如清单 3中所示的示例；编译器给出无用的诊断信息会被视为缺陷。这门语言的设计也倾向于报错，而不是产生可能令人意外的行为。例如，在可能溢出的算术运算之后，要求显式截断。

A. 编译器架构

Spade 编译器是一个使用 Rust 编写的多阶段编译器，可将输入的 Spade 代码编译为目标语言，目前为 SystemVerilog 的一个精简子集。编译过程从词法分析和语法分析开始，以生成抽象语法树（AST）。随后对 AST 遍历三次：首先收集程序中的所有类型，然后收集所有单元，最后将其降级为高级中间表示（HIR）。从 AST 到 HIR 的降级过程保留了 AST 的树结构，同时解析名称和作用域规则，并执行初步语义分析。生成 HIR 后，会进行类型推断，随后如前所述执行线性类型检查。利用 HIR 及其类型信息生成中级中间表示（ MIR ）。在这一步中会进行更多语义分析，并将树结构展平为简单语句列表。最后，从 MIR 生成 SystemVerilog。选择 SystemVerilog 作为目标语言，是因为开源和专有的仿真与综合工具都对它提供了良好支持；但编译器以独立于目标的方式编写，便于轻松试验或切换到不同的后端。尤其值得关注的后端是 CIRCT [16] 方言，例如 LLHD [17] 或 Calyx [18],，它们既能提供与语言无关的优化，也能生成 SystemVerilog 以外的其他输出语言的代码。

B. 工具链与生态系统

Spade 随附构建工具 Swim[2] ，用于管理项目文件、后端构建工具和依赖项。一个 Swim 项目由 Spade 文件和以 TOML 编写的配置文件组成。该文件包含构建工具参数、要纳入项目的原始 Verilog 文件以及外部依赖项等内容。随后，Swim 会管理项目文件的命名空间、依赖项的下载和版本控制

[2]https://gitlab.com/spade-lang/swim

清单 8. 使用 cocotb 编写的 Spade 测试平台示例。

```
1 # top=
外设
asy
dut
(10
" C
ass
```

以及通过便捷的接口调用综合和仿真工具。它还提供插件系统，用于扩展构建流程，例如运行命令生成 Spade 或 Verilog 代码、加载其他 Yosys 插件，或将输出位流打包到微控制器的可执行文件中，再由该微控制器对目标 FPGA 进行编程。

为了便于将Spade 集成到现有项目中，可以对单元进行注释，以防止名称重整。Spade单元也可以标记为外部单元，从而允许在 Spade 项目中使用现有的 IP模块。

一个 tree-sitter语法，以及一个基础的语言服务器协议（LSP）[19] 服务器可用，使任何支持 LSP 和/或 tree-sitter 的文本编辑器都能提供类似 IDE 的体验。

生成与输入 Spade代码结构相似的 Verilog方面的工作较为有限。不过，编译器确实会尝试保持名称可读，并提供将名称和表达式映射回其源代码位置的函数，以帮助调试。Swim会自动将VCD文件中的值转换为其高级 Spade值，而编译器会在输出的Verilog 中包含源代码映射，使时序报告等内容无需查看输出的 Verilog 即可读懂。

C. 测试平台与仿真

Spade语言本身主要用于硬件描述和综合，而不是仿真。然而，为了验证生成硬件的正确性，仿真和测试平台必不可少。Spade测试使用 cocotb [20], 一个基于 Python 的协同仿真测试平台环境来进行验证。cocotb库扩展了相关功能，用于将 Spade 值作为输入和输出写入被测单元，同时由编译器生成的 Verilog 使用现成的 Verilog仿真器进行仿真。

例如，清单 8 展示了一个内存映射定时器外设测试平台的一部分。第一行指定了要测试的模块，接下来的两行是标准的cocotb测试用例定义。第 4行将 cocotb 被测设计封装在一个 Spade 类中，该类扩展了 cocotb接口以添加Spade 特有的功能。第 6–7 行开始一个用于驱动被测设计时钟的任务。在第9–11 行，为模块输入设置的值会由 Spade编译器进行编译和求值。最后，第13 行断言输出符合预期，同样传入一个包含 Spade 表达式的字符串作为预期值。

为了实现这一点，Python 测试平台必须能够编译 Spade 代码；同时，为了允许在测试平台中使用项目内定义的类型和单元，必须将编译器的状态提供给测试平台。因此，项目构建完成后，编译器的状态会被序列化并存储在磁盘上。此外，Spade 编译器的部分功能会导出为一个 Python 模块，该模块读取已存储的状态，然后编译并计算测试平台中使用的表达式。所得的位向量用于驱动被测设计的输入，或与预期输出进行比较。

VI. 结论

Spade 是一种硬件描述语言，旨在简化 FPGA 或 ASIC 的开发。为此，它将流水线、寄存器和存储器等常见硬件构造明确地纳入语言本身。它深受现代软件语言的启发，例如集成了强大的类型系统和模式匹配。与目前大多数其他硬件描述语言不同，Spade 是一种独立的语言，拥有自定义编译器，并且不会将底层硬件抽象掉。最后，它还提供了实用的工具，例如能够生成详细错误消息的编译器、支持依赖管理的构建系统，以及可带来类似 IDE 使用体验的 LSP。

参考文献

[1] J. Bachrach 等，“Chisel：在嵌入式 Scala 语言中构建硬件”，载于设计自动化会议论文集，2012 年 6 月，第 1212–1221 页。[2] SpinalHDL 贡献者，“SpinalHDL”，https://github.com/SpinalHDL/ SpinalHDL，2022 年。[3] O. Port 和 Y. Etsion，“DFiant：一种数据流硬件描述语言”，载于国际现场可编程逻辑与应用会议论文集，2017 年。[4] Amaranth 贡献者，“Amaranth HDL”，https://github.com/ amaranth-lang/amaranth，2022 年。[5] M. Korbel，“快速开放硬件开发框架”，载于开源 *EDA* 技术研讨会论文集，2022 年。[6] J.-C. Le Lann、H. Badier 和 F. Kermarrec，“迈向硬件 DSL 生态系统：RubyRTL 及相关工具”，载于欧洲设计自动化与测试会议论文集——开源设计自动化研讨会，2020 年 3 月。[7] C. Baaij，“cλaSH 中的数字电路”，博士论文，特温特大学，2015 年 1 月。[8] J. Kemmerer，“PipelineC：介于 RTL 与 HLS 之间的简易开源硬件描述”，载于开源 *EDA* 技术研讨会论文集，2022 年。[9] S. F. Hoover，“事务级 Verilog 中的时序抽象电路设计”，载于*IEEE* 国际计算机设计会议论文集，IEEE，2017 年 11 月。[10] R. Nikhil，“Bluespec SystemVerilog：根据高级规范生成高效、正确的 RTL”，载于*ACM/IEEE* 国际形式化方法与协同设计模型会议论文集。IEEE，2004 年。[11] S. Lefebvre，“Silice”，https://github.com/sylefeb/Silice/tree/5003ec72，2022 年 11 月。[12] S. Klabnik 和 C. Nichols，《*Rust* 编程语言》，第 1 版，No Starch Press，2018 年 6 月，第 Glossary 章。[13] D. Luu，“编写安全的 Verilog”，https://danluu.com/pl-troll/。[14] R. Nigam 等，“利用时间敏感仿射类型实现可预测的加速器设计”，载于*ACM SIGPLAN* 编程语言设计与实现会议论文集。ACM，2020 年 6 月。[15] P. Wadler，“线性类型可以改变世界！”，载于《编程概念与方法》，1990 年。[16] CIRCT 项目，“CIRCT”，https://circt.llvm.org/，2022 年。[17] F. Schuiki 等，“LLHD：用于硬件描述语言的多层中间表示”，载于*ACM SIGPLAN* 编程语言设计与实现会议论文集，2020 年 6 月。[18] R. Nigam 等，“用于加速器生成器的编译器基础设施”，载于*ACM* 国际编程语言与操作系统体系结构支持会议论文集，2021 年 4 月。[19] Microsoft，“语言服务器协议”，https://microsoft.github.io/ language-server-protocol/。[20] cocotb 贡献者，“cocotb”，https://www.cocotb.org/，2022 年 11 月。