SystemVerilog 与 Verilog 描述状态机(FSM)之比较

杨厚一,徐东明

(西安邮电学院 电子与信息工程系,陕西 西安 710121)

摘要:由于状态机不仅是一种电路的描述工具,而且也是一种思想方法,因而在电路设计的系统级和 RTL 级有着广泛的应用。如何编写出高质量、易维护和可复用的 RTL 级代码,这既对硬件工程师提出了新的挑战,又对硬件描述语言的抽象层次、语义及语法也提出了更高的要求。本文详细描述了如何使用新的 SystemVerilog 来构建 FSM 的寄存器传输级(RTL)编码技术,并且将现存有效的 RTL 编码风格与新的增强的 SystemVerilog 编码风格进行比较,以显示 SystemVerilog 在构建 FSM 中的优势。

关键词:SystemVerilog;Verilog;状态机(FSM);寄存器传输级(RTL);编码风格(Coding Style)

中图分类号:TP332.1 **文献标识码**:A **文章编号**:1007 - 3264(2008)03 - 0106 - 05

引言

随着芯片制造工艺的迅猛发展,片上系统(SOC)已经成为当今集成电路设计的主流。在系统芯片的设计流程中,像系统定义、软硬件划分,设计实现等,集成电路设计界一直在考虑如何满足 SOC 的设计要求,一直在寻找一种能同时实现较高层次的软件和硬件描述的系统级设计语言。正是在这种情况下,由 Accellera 标准组织在 Verilog 的基础上进行扩展从而形成了 SystemVerilog。SystemVerilog 是一种硬件描述和验证语言(HDVL),它基于 IEEE 1364 - 2001 Verilog 硬件描述语言(HDL),并对其进行了根本性的修改,包括扩充了 C 语言数据类型和结构、压缩和非压缩数组、接口、断言等,这些都使得 SystemVerilog 在一个更高的抽象层次上提高了设计的建模能力。利用 SystemVerilog 新增的如枚举类型、新的过程语句等特性,来描述如状态机(FSM)这样的时序逻辑,可以方便设计,增强代码的阅读性和可维护性,并且能够提升设计效果,提高验证水平。同时,增强了 Verilog 跨软件平台翻译的一致性,避免了因使用不同工具而导致设计产生错误情况的发生。本文首先介绍了状态机的一些基本概念,然后讨论了状态机的编码风格,最后重点比较 SystemVerilog 与 Verilog 描述有限状态机的异同。

1 状态机的基本概念

状态机是由组合逻辑和时序逻辑构成的时序机,特别适合描述那些发生有先后顺序,或者有逻辑规律的事件。而且状态机的 HDL 的编码风格较规范。因此,在数字电路的设计中有着广泛的应用。下面介绍状态机的一些主要概念。

状态机的基本要素有 3 个,它们分别是:状态、输出和输入。

◆状态:也叫状态变量。在逻辑设计中,使用状态划分逻辑顺序和时序规律。

◆输出:指在某一个状态时特定发生的事件。

◆输入:指状态机进入每个状态的条件,有的状态机没有输入条件,其中的状态转移较为简单;有的状态机有输入条件,只有当某个输入条件发生时才能转移到相应的状态。

根据状态机的输出是否与输入条件相关,可将

收稿日期:2007 - 09 - 29

作者简介:杨厚一(1982 -),男,陕西安康人,西安邮电学院电子与信息工程系硕士研究生;
徐东明(1963 -),男,湖北武汉人,西安邮电学院电子与信息工程系教授。

状态机分为两大类:摩尔(Moore)型状态机和米勒(Mealy)型状态机。

◆摩尔状态机:摩尔状态机的输出仅依赖于当前状态,而与输入条件无关。如下图所示。

◆米勒型状态机:米勒型状态机的输出不仅依赖于当前状态,而且还取决于该状态的输入条件,如图 1 所示。

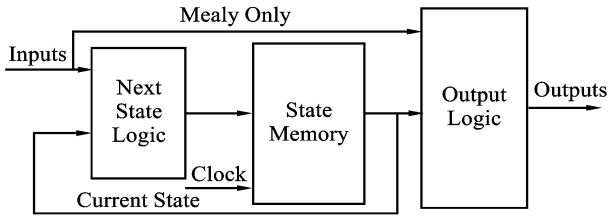

图 1　状态机的结构图

根据状态机状态的数量,可将状态机分为有限状态机(Finite State Machine,FSM)和无限状态机(Infinite State Machine,ISM)。而这所讨论的状态机主要是指有限状态机,用 FSM 表示。

2　FSM 性能的评判标准和编码风格

2.1　RTL 级 FSM 性能的评判标准

使用 HDL 进行状态机的编写灵活性很大,但也必须遵循相应的规范。通过使用一些规范的描述方法,可以使 HDL 描述的状态机更安全、稳定、易于维护。那么规范的 RTL 级 FSM 编码的评价标准有哪些,这里拣选最重要的几个方面进行讨论。

◆FSM 的安全性高,鲁棒性好。

所谓 FSM 安全是指 FSM 不会进入死循环,特别是不会进入非预知的状态。这里面有两层含义,第一:要求该 FSM 的综合实现结果无毛刺等异常扰动;第二:要求状态机要完备,即使收到异常扰动进入非设计状态,也能很快恢复到常状态。

◆FSM 速度快,满足设计的频率要求。这里“速度”指设计在芯片上稳定运行所能够达到的最高频率。)

◆FSM 面积小,满足设计的面积要求。(这里“面积”是指一个设计所消耗 FPGA/CPLD 的逻辑资源数量。)

◆FSM 设计要清晰易懂、易维护,方便调试。

不规范的 FSM 写法很难让其他人解读,甚至过一段时间后设计者也发现很难维护。所以,各条标准要综合考虑,根据设计的要求进行权衡。其实科学的设计目标应该是:在满足设计时序要求(包含对设计最高频率的要求)的前提下,占用最小的芯片面积,或者在所规定的面积下,使设计的时序余量更大,频率更高。

2.2　RTL 级 FSM 的编码风格

在了解了高质量 FSM 编码的要求之后,使用 HDL 编写 RTL 级 FSM 时,就应该尽可能的遵守一定的编写规则,使编写出来的代码易于阅读、维护、修改和整理文档。更为重要的是这样的代码才能最大限度的使各种工具发挥其最大的作用,实现最佳的仿真和综合结果。现将 RTL 级编写 FSM 的通用规则罗列如下:

◆使每一个状态机独立的在一个模块中。这样做的目的是保持每一个状态机与其它可综合逻辑分开,从而可以简化状态机的定义、修改和调试。

◆使用有意义的参数给状态机的状态进行赋值。这样做的目的是定义和使用符号化状态名使得 Verilog 代码便以阅读,并且可以减轻重新定义状态的任务。

◆在时序逻辑的 Always 过程块中仅可以使用非阻塞赋值(用符号“ < = ”表示)。这是因为非阻塞赋值模拟实际硬件流水线并行处理行为,从而可以排除绝大多数潜在的竞争状态。

◆在组合逻辑的 Always 过程块中仅可以使用阻塞赋值(用符号“ = ”表示)。组合逻辑的 Always 过程块常用来更新状态机的下一个状态或 \ 和输出。

◆FSM 的编码。Binary(二进制编码)、Gray - code(格雷码)编码使用最少的触发器,较多的组合逻辑,而 One - hot(独热码)编码则反之。One - hot 编码的最大优势在于状态的比较仅需要比较一个比特,一定程度上简化了比较逻辑,减少了毛刺产生的概率。由于 CPLD 更多地提供组合逻辑资源,而 FPGA 更多地提供触发器资源,所以 CPLD 多使用 Gray - code,而 FPGA 多使用 One - hot 编码。另一方面,对于小型设计使用 Gray - code 和 Binary 编码更有效,而大型状态机使用 One - hot 更高效。

3　Verilog 和 SystemVerilog 描述 FSM 方法及区别

3.1　Verilog 和 SystemVerilog 如何描述 FSM

其实,不论是用 Verilog 还是用 SystemVerilog 编写该状态机,常用的描述方法有一段式、两段式和三段式。而推荐使用的是两段式和三段式。两段式

是指将状态机的组合逻辑和时序逻辑在两个 Always 过程块中分别描述。这样做的好处不仅便于阅读、理解、维护,更重要的是利于综合器优化代码,利于用户添加合适的时序约束条件,利于布局布线器来实现设计。而三段式与两段式相比,关键在于根据状态转移规律,在上一状态根据输入条件判断出当前状态的输出,从而在不插入额外时钟节拍的前提下,实现了寄存器输出。

这里我们就以一个串行比特流中搜索 3'b110 为例,当在该串行比特流中检测到该序列时就给出一个指示信号。状态如图 2 所示:

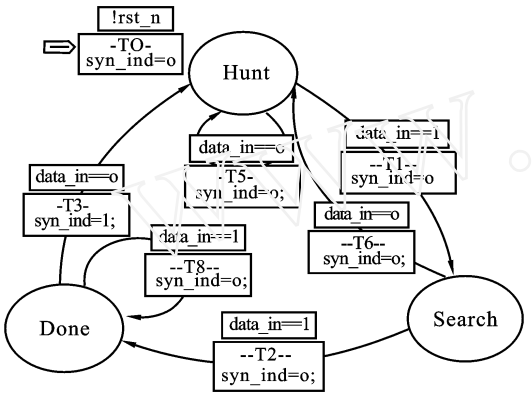

图 2 串行比特流中搜索 3'b110 的状态图

根据上面的状态图,先看由 Verilog 如何编写该状态机,这里仅给出了部分代码以供比较使用。

```
……
reg [2:0] State, Next;
parameter [2:0] Hunt = 3'b001, Search = 3'b010,
Done = 3'b100;

always @(posedge clock or negedge rst - n) // Sequential Logic
    if (! rst - n) State <= Hunt; // reset to hunt state
    else State <= Next;

always @(State or data - in)
  begin
    case (State)
      ……
      Default: Next = State;
    endcase
end
      ……
```

例 1 使用 Verilog 描述的状态机

接下来,再看由 SystemVerilog 编写该状态机。同样,这里也仅给出了部分代码。

```
……
enum logic [2:0] {Hunt = 3'b001,
    Search = 3'b010,
    Done = 3'b100} State, Next;

always - ff @(posedge clock, negedge rst - n)
    if (! rst - n) State <= Hunt; // reset to hunt state
    else State <= Next;

always - comb begin: set - next - state
    Next = State; // the default for each branch below
    unique case (1'b1) // reversed case statement
      ……
    endcase
end: set - next - state
    ……
```

例 2 使用 SystemVerilog 描述的状态机

3.2 Verilog 与 SystemVerilog 构建 FSM 的比较

以上两个例子结构上大体是一致的,状态的转移用同步时序逻辑描述,判断状态的转移条件用组合逻辑实现,也就是上面提到的两段式描述方法。但仔细比较就会发现这两种描述方式之间的区别。就这两者之间的区别本文将做详细的讨论。

3.2.1 Always 过程中所描述逻辑类型的明确程度不同

在 Verilog 中的 Always 过程中,具体要生成什么逻辑需要软件进行检测。而在 SystemVerilog 中,由于已经明确声明了过程块中所要生成的逻辑类型,如 Always - FF 将生成触发器;Always - Comb 将生成组合逻辑;Always - Latch 将生成锁存器。因此,软件工具将检测过程块中的内容并且当所设计的内容与过程语句声明的类型不一致时将产生告警。

3.2.2 是否需要声明敏感列表以及生成的敏感列表的内容不同

在 Verilog 组合逻辑的 Always 过程块内,所有语句右边的变量需要明确地被声明在敏感列表中,否则将会产生综合前和综合后仿真的不一致。为了避免这种情况,Verilog - 2001 标准通过使用 @ * 或者 @(*) 为 @事件控制指定一个通配符,通配符的主要作用是不用声明敏感列表中的信号就可以自动生成组合逻辑敏感列表。虽然 Verilog @ * 可以简化复杂的组合逻辑设计,但是 @ * 结构并不要求它所设计的内容遵循可综合的组合逻辑设计规则,即 @ * 结构是不可综合的。此外 Verilog @ * 结构当调用函数来设计大型的组合逻辑时,不能够生成完

整的敏感列表(函数中所触发的信号未被包含在敏感列表中)。

而 SystemVerilog 中的 Always - Comb 过程块却很好的解决了这个问题。首先,Always - Comb 继承了 Verilog 中 Always 过程块的属性,与 Verilog - 2001 标准中的 @ * 相似的是 Always - Comb 也不需要声明组合逻辑的敏感列表,而会自动生成组合逻辑的敏感列表。而与 Verilog - 2001 标准中的 @ * 区别有即使在过程块中被调用函数所读取的信号,也会被包含敏感列表中,从而避免了综合前和综合后仿真的不一致;此外,也是最重要的一点,Always - Comb 过程块语句是可综合的。最后,又因为 Always - Comb 的语义规则是标准的,所有的软件工具将生成相同的敏感列表,从而可以实现 SystemVerilog 的跨平台移植,而不会出现因不同的工具而出现不同结果的情况。

3.2.3　新抽象数据类型的使用

Verilog 使用"reg"数据类型作为一般的变量来在 Initial 和 always 过程块中描述硬件行为。而关键字 reg 是一个误区经常迷惑那些学习 Verilog 的新手。"reg"看起来似乎生成硬件中的"寄存器",以触发器的形式生成时序逻辑类型。而事实上,使用 reg 变量与生成的硬件类型之间没有任何对应关系,而是根据上下文的内容来决定生成的硬件是由何种逻辑类型构成。

SystemVerilog 为 Verilog 加入了几个新的数据类型,以便能够在更抽象的层次上建模。如 logic 数据类型就比 Verilog 的线网和寄存器数据类型更加灵活、更直观,它使得在任何抽象层次上建模硬件都更加容易。关键字 logic 实际上并不是一个变量类型,而是一个数据类型,指示该信号有四个状态值,分别为 0、1、X 和 Z。由于 logic 数据类型没有暗示将要生成的硬件类型,因此,logic 类型能够以下面的任何一种方法赋值:

通过任意数目的过程赋值语句赋值,能够替代 Verilog 的 reg 类型;

通过单一的连续赋值语句赋值,能够有限制地替代 Verilog 的 wire 类型;

连接到一个单一原语的输出,能够有限制地替代 Verilog 的 wire 类型;

由于 logic 数据类型能够被用来替代 Verilog 的 reg 或 wire(具有限制),这就使得能够在一个更高的抽象层次上建模,并且随着设计的不断深入能够加入一些设计细节而不必改变数据类型的声明。logic 数据类型不会表示信号的强度也不具有线逻辑的解析功能,因此 logic 数据类型比 Verilog 的 wire 类型更能有效地仿真和综合。

3.2.4　Unique 修饰符的使用

在由 Verilog 设计 FSM 的例 1 中,当前状态和下一个状态是 3 比特的寄存器型,而 3 比特的寄存器型的所有组合有 8 种情况,而组合逻辑过程块的 Case 语句中被使用的 State 仅有 3 种情况。为了明确指示哪些 Case 表达式未被使用,在 Verilog 中有两种通用的建模方法:一、未使用的状态通过 Default 语句赋与逻辑值 X。二、通过声明一个特殊的综合属性 Full - Case。就第一种情况而言,综合工具会识别所有落入 Default 语句中的 Case 表达式为未被使用的状态,这样就可以使综合工具实施额外的优化来提高综合结果的质量。而第二种情况,综合属性 Full - Case 相对于仿真工具而言它将提供更多关于设计的信息给综合工具。这个特殊的指示被用来告知综合工具,所有的 Case 语句被完全的定义了,对于所有未被使用的 Case 的输出赋值被视为"不关心",而对于仿真工具它相当于一个注释,不起任何作用。因此,使用这个指示时在综合前和综合后的设计的功能可能会不一致。

而 SystemVerilog 的当前状态和下一个状态在枚举数据类型中已经被明确地定义,就例 2 而言它的可能取值为枚举列表中的 3 种情况,分别为 Hunt、Search 和 Done。这也就是说,所有使用的状态都被包含在枚举列表中,而未使用的状态都在枚举列表之外。因此,前状态和下一个状态不会进入到未声明的状态中。在 Case 语句中使用 Unique 修饰符是非常重要的。因为独热码一次仅有一比特置位,在 Case 表达式中仅有一个 Case 选项将与数值 1 相匹配。使用 Unique 修饰符说明了 3 件事:

第一:Unique case 指明了所有的 Case 选项都可以被并行估算,而没有优先级编码产生。软件工具如综合编译器可以优化 Case 选项的编码逻辑,来生成一个更小、更有效的硬件实现。

第二:Unique case 指明了在 Case 选项中不应该有重叠的部分。在仿真工具运行期间,如果 Case 表达式的值同时满足两个或两个以上 Case 选项时,将会产生告警。这一语法检测可以在设计的早期发现设计的错误。而综合属性如 Paralle - case 并不能提供这种重要的语法检测。

第三:Unique case 指明了在仿真期间 Case 表达式出现的所有值必须被 Case 选项所覆盖。对于 U-

nique case 而言,如果 Case 表达式中出现的一个值不会触发 Case 语句的一个分支被执行,将会产生告警。该语法检测同样可以在设计的早期发现设计的错误。这与综合属性 Full - Case 相似,但是该综合指示并不能要求如仿真工具来执行任何语法检测。

综上所述,通过将 Verilog 构建 FSM 同 SystemVerilog 构建 FSM 的各自特点进行了比较,显而易见,用 SystemVerilog 来构建 FSM 虽然未能大大简化编码的数量,但是它方便了设计,提升了设计效果和验证水平。并且还提供了额外的语法检测,使设计中的错误在设计早期就被发现并及时修正,从而可以有效的避免综合前仿真和综合后仿真不一致的出现。

4 结束语

本文就现存有效的编写 RTL 级有限状态机(FSM)的代码风格进行了讨论,并在此基础上就 Verilog 构建 FSM 与 SystemVerilog 构建 FSM 的异同进行了详细的比较。突显了 SystemVerilog 新的扩展内容对编写 RTL 级代码性能的巨大提升。本文不仅就 SystemVerilog 的一些语法进行了深入的讨论,并且也给出了一般逻辑设计的一些规范。因此,对 RTL 级的设计,特别是 RTL 级状态机的设计具有重要的指导意义。可以预见,在不久的将来,SystemVerilog 替代 Verilog 将是发展的必然趋势。

参 考 文 献

[1] Stuart Sutherland, Simon DavidMann and Peter Flake, "SystemVerilog for Design," Second edition, 2003.

[2] S. Golson, "State Machine Design Techniques for Verilog and VHDL," Synopsys Journal of high - Level Design, September 1994.

[3] Clifford E. Cummings, "Synthesizable Finite State Machine Design Techniques Using the New SystemVerilog 3.0 Enhancements," March 6th, 2003.

[4] Clifford E. Cummings, "The Fundamentals of Efficient Synthesizable Finite State Machine design using NC - Verilog and Build Gates," September 16 - 18, 2002.

[5] Clifford E. Cummings, "State Machine Coding Styles for Synthesis," SNUG'98 (Synopsys Users Group San Jose, CA, 1998) Proceedings, section - TB1 (3rd paper), March 1998.

[6] Clifford E. Cummings, "Coding and Scripting Techniques for FSM designs with Synthesis Optimized, Glitch - Free Outputs," SNUG (Synopsys Users Group Boston, MA 2000) Proceedings, September 2000.

[7] SystemVerilog 3.0 Accellera's Extensions to Verilog, Accellera, 2002.

[8] Bhasker J. Verilog HDL Synthesis A practical Primer, Star Galaxy Press, Allentown, PA, August 1998, ISBN 0 - 9650391 - 5 - 3.

[9] 吴继华,王诚. Verilog 设计与验证[M]. 北京:人民邮电出版社,2006.

[10] 唐杉,徐强,王莉薇. 数字 IC 设计 - 方法、技巧与实践[M]. 北京:机械工业出版社,2006.

Comparison between system verilog and verilog in modeling Finite State Machine (FSM)

YANG Hou - yi, XU Dong - ming

(Department of Electronic and Information Engineering, Xi'an University of Post and Telecommunications, Xi'an710121, China)

Abstract: Finite State Machine (FSM) is not only a tool for describing the circuit, but a way for thinking, therefore, it is widely used in the areas of digital circuit design, such as system level modeling and Register Transfer Level (RTL) modeling. So how to code efficient, maintainable and reusable RTL coding, is not only come up with new challenge to the hardware engineer, but to the abstract level, syntax and grammar of hardware description language(HDL). This paper details RTL coding and synthesis techniques of Finite State Machine (FSM) design using the new SystemVerilog capabilities. What is more, efficient existing RTL coding styles are compared to new SystemVerilog enhanced coding styles, so as to show the advantage of SystemVerilog in modeling FSM.

Key words: system verilog; verilog; Finite State Machine(FSM); Register Transfer Level(RTL); coding style