## 形式化验证方法学的介绍与应用

奥卡思微电科技

**ExASIC**

微信号 ExASIC

功能介绍 分享数字集成电路设计中的经验和方法。分享让工作更轻松。

本文提纲：

一、验证背景介绍

二、形式化验证简介

三、形式化验证和仿真验证相结合的综合验证方法

四、形式化验证实例

五、公司介绍

**一、验证背景介绍**

在芯片设计和基于FPGA的设计流程中数字前端验证是人力花费最多的一个环节，可以说芯片数字前端的设计流程就是设计和验证的交替迭代。验证占整个芯片设计流程70%的人力、物力资源，流片失败的原因中70%是由于验证不充分导致的功能错误。

前端验证的主要内容包括：功能，时钟，功耗等方面。整个设计划分成合适大小的模块，初步完成RTL级（寄存器传输级，Register Transfer Level）的硬件描述语言文件之后，功能验证人员会在这个时候开始前端验证。

当功能验证完成以后，后端人员会将RTL文件综合生成门级网表文件（Gate  Netlist），然后进行布局布线，在后端生成网表之后，前端验证人员还需要进行基于标准延时格式（SDF，Standard Delay Format）文件的网表门级仿真，验证物理实现的功能正确性。

数字前端验证主要分为仿真，形式化验证和硬件加速。主要特点如下：

* 仿真：建立仿真环境，创建输入激励，然后对比仿真和预期结果；
* 形式化验证：建立形式化验证模型，生成约束和属性，并验证属性的正确性；
* 硬件加速：用可编程逻辑实现仿真的硬件加速。

**二、形式化验证简介**

**1、形式化验证理论**

形式化方法：目前形式化方法的主要研究对象是计算机系统的设计和验证，其主要目的是帮助工程师构造正确可靠的计算机系统，形式化方法最基本的特点是利用数学的概念、方法和工具来解决设计的正确性问题，作为形式化方法的主要数学基础包括各种逻辑学、集合论、代数理论、图论。一般而言，形式化方法就是用具有形式语义的记号和工具明确地表述所要设计的计算机系统的设计要求，即给出系统规范，并根据系统规范利用上述记号和工具对系统具有的性质和最终实现的正确性进行严格的证明。

形式化验证是形式化方法在数字硬件设计领域中的应用。从数学上完备地证明或验证电路的实现方案是否符合电路设计描述的功能；形式化验证是使用基于数学变换的静态分析来确定硬件或软件行为的正确性的技术集合，与动态验证技术（如软件仿真）对比，形式化验证比形式规范向前迈进了一步，它对设计的系统进行分析，以便证明系统具有需要的性质。

**2、形式化验证的分类**

形式化验证主要可分为三大类：等效性检验、模型检验、定理证明

模型检验：主要是检查RTL代码是否满足规范中规定的一些特性。在规定这些特性时一般使用特性规范语言，目前使用基于断言的验证语言。由于这种方法可以在不需要仿真的前提下检查设计中所有可能出现的情况是否满足规定的特性，所以使用这种方法不会遗漏任何的边界情况；模型检查形式化验证比较复杂，因为它不仅依赖于用户生成的断言，而且还依赖于为给定问题选择正确的算法，其基本思想是利用计算树逻辑（CTL）公式表达程序或电路的时序性质，用FSM表示程序或电路的状态转移的抽象的结构，通过遍历FSM来检查时态逻辑公式的正确性。

等价性检验：主要是验证在一个设计经过变换之后，穷尽地检验变化前后的功能的一致性，这里的设计变化可以是综合、布局布线、测试、时钟树的插入、扫描链的重排序，FPGA到ASIC的转换等；等价性检查的基本原理是建立被比较的两个模型之间的关系，检验的依据是数学的定理和公理，以及设计实现所利用的标准单元库的精确描述，用形式化算法分析彼此的数据结构来进行比较。

定理证明：定理证明方法是一种将模型抽象为逻辑公式，然后使用自动的逻辑推理技术来验证电路是否正确的技术,定理证明方法十分严格，跟数理逻辑结合十分紧密，一般使用高阶逻辑（Higher-Order Logic，HOL）系统来进行证明。

**3、形式化验证与其他验证手段相比的优势和限制**

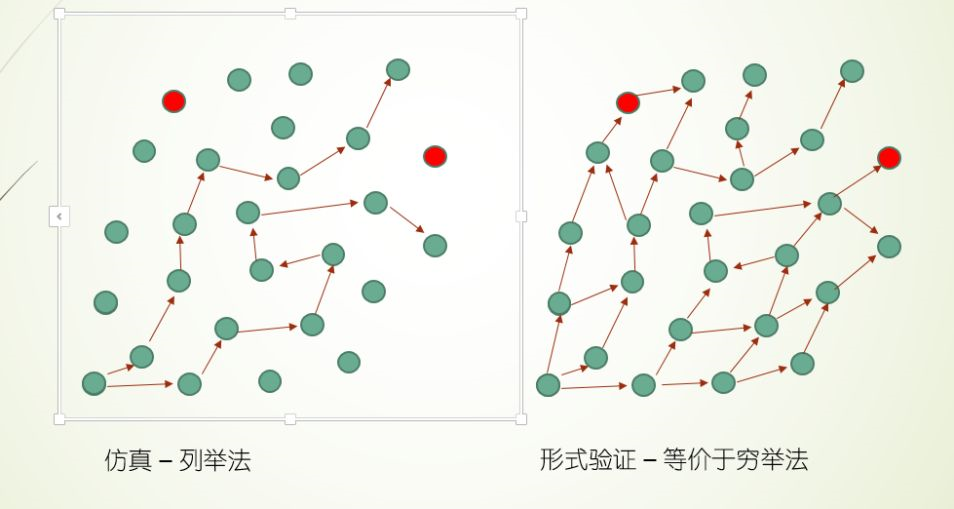
仿真的特点：

* 基于仿真模型对输入激励的执行，本质上是例举法，验证可靠性取决于激励的质量
* 需要生成激励；
* 需要知道预期结果；

形式化验证的特点：

* 基于对形式化模型的数学和逻辑分析，可靠性取决于约束和属性的质量；
* 需要生成约束和属性；

下图描述了仿真验证和形式化验证的区别：



形式化验证的优势：

* 形式化验证能覆盖完整的设计状态空间，基于仿真的验证只是检验了在使用某些测试向量时，系统不会出现漏洞，但无法保证当使用其他测试向量时，漏洞不会出现；形式化验证是对指定描述的所有可能的情况进行验证，因此有效地克服了模拟验证的不足。形式化验证的完备性取决于属性的完备性，而后者可以基于设计要求建立清晰明确的模型；
* 由于形式化验证技术采用数学上的方法将待验证电路和功能描述或参考设计直接进行比较，因此测试者不必考虑如何获得测试向量，一条属性的真伪结论基于严格的数学证明，证明为真的属性任何激励下进行仿真都不会出错；
* 由于属性可以基于设计内部的任何信号和运行场景，形式化验证还具有白箱验证的优点，而仿真只能在输出端确定对错；

形式化验证的限制：

* 形式化验证的缺点是所谓的“状态爆炸”问题（State Explosion），这个问题是由形式化验证，具体而言是形式化验证中最为常用的模型验证（Model Checking）的工作机制决定的，是形式化验证的根本性问题，无法通过优化算法或者工具完全解决；
* 形式化验证到目前为止仍然不能有效的验证电路的性能，如电路的时延和功耗等。

**4、形式化验证的目前进展**

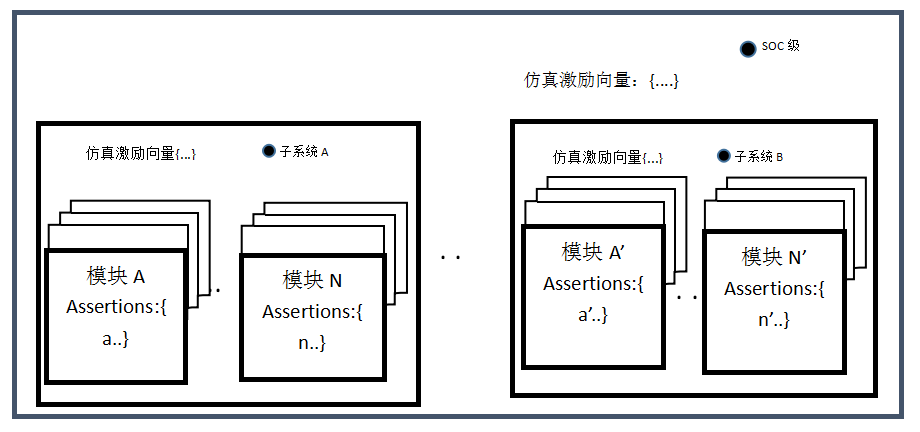
最近二三十年来，形式化方法在各个领域都取得了巨大进步，基础研究的进展加上技术进步的推动，使得形式化方法的发展呈现以下几个方面的趋势：1. 研究更加一般的概念、模型和理论，2. 研制更加实用的方法和工具，3. 方法和工具的集成。

由于形式化方法特别是模型检验已经表现出巨大的发展潜力，模型验证已被用于验证大量的实际设计，包括检查奔腾微处理器的浮点运算单元的错误。近年来模型检验的应用扩大，除硬件设计之外，还包括IEEE标准的验证、各种协议验证、大型通讯软件验证等，各大EDA厂商如Cadence、Synopsys、Mentor/Siemens等争相把模型验证商品化，开发商用的模型检验软件甚至成为了投资热点，目前正在抓紧时机开发各种形式化、半形式化的工具。

目前EDA行业主要的针对芯片设计的形式化验证软件工具主要有：AveMC (成都奥卡思微电科技) , Jasper (Cadence) ，VC-formal (Synopsys) ,QuestaFormal (Mentor) 和360dv-verify (Onespin) 等。

**三、形式化验证和仿真验证相结合的综合验证方法**

由于形式化验证和仿真验证都存在各自的限制和优势，二种验证方法缺一不可。形式化验证和仿真验证相结合的的应用场景是：模块级验证：对大部分模块采用形式化验证方法；全芯片系统级：采用软件仿真、硬件仿真等验证手段，而用于模块级的形式化验证的断言模型可做为monitor，帮助定位设计的错误，如下图：



模块级的功能验证采用模型检查的方式对规范中的特性向量assertions进行验证，而且验证的assertions可作为子系统/SOC级的monitor或者checker的目标点；对于子系统/SOC级则可以采用传统的仿真方式，通过测试向量的激励来验证设计的正确性；另外对于同一模块的不同版本之间统一性验证，可以采用等价性验证的方式，来保证设计始终保持其正确性。

**四、形式化验证实例**

本章描述基于一个简单的交通信号控制子系统的设计，使用AveMC工具来构建完整的形式化验证流程，验证该设计的正确性：内容大致分为以下几个部分：1、**设计简介**：介绍该信号灯控制子系统的设计规格需求；2、**验证计划简介**：针对该设计制定一个完整的验证计划；3、**AveMC使用简介**：介绍软件的使用流程；4、**实验部分**：以举例的方式详细介绍使用AveMC工具实施验证计划。5、**验证总结**：简述设计的验证结果。

1、设计简介

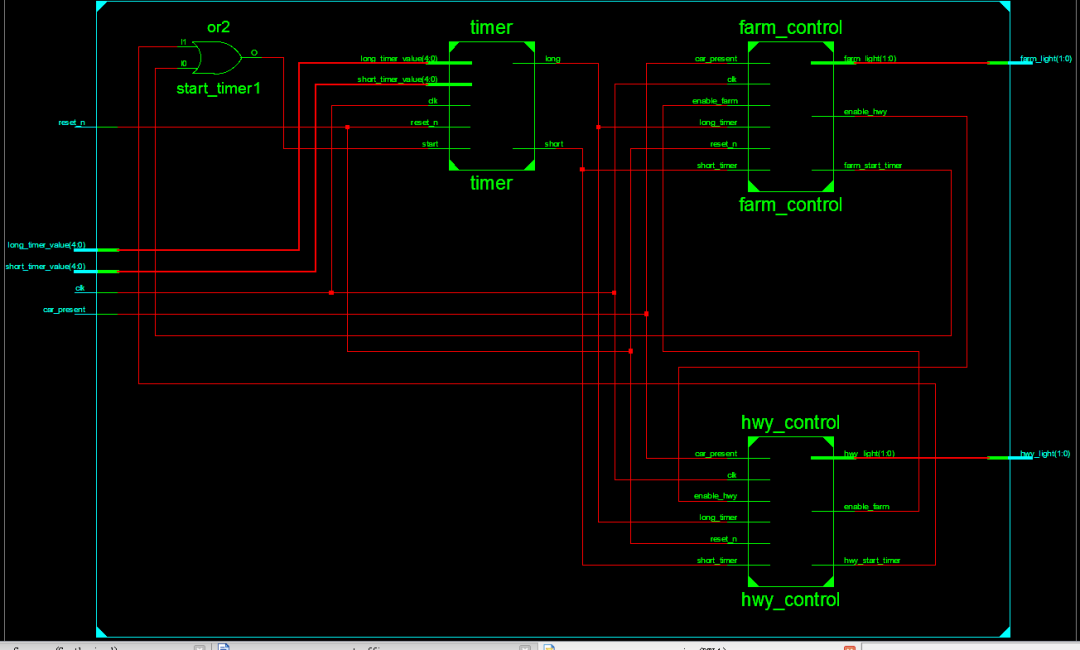
**系统规格描述**

本章描述了Verilog crossroad交通灯信号控制子系统，这个例子是1995年由Tom Shiple设计。目的是实现对两个方向的交通信号灯的控制，同时有优先级，并保证系统安全高效的运行，横向是highway road，有优先通过权，纵向是farm road，要求该子系统具有一些指定的特性（specifications），规格需求描述如下：

|  |  |
| --- | --- |
| 序号 | 规格描述 |
| 1 | 两个方向的交通灯信号均分别交替变化：绿灯→黄灯→红灯→绿灯... |
| 2 | 两个方向的通行是交替的，且遵循绿灯行，红灯停，黄灯警示的一般通行规律 |
| 3 | 任何时候两个方向的交通灯信号都是不同的 |
| 4 | 为了保证安全，一个方向为黄灯时，不允许另一个方向为绿灯 |
| 5 | 横向的highway road有优先通行的权利：即当纵向的farm road 为绿灯时，如果此时该方向没有车通行，需要变更控制灯信号 |
| 6 | 各个不同颜色的交通灯信号时间是可配置的 |

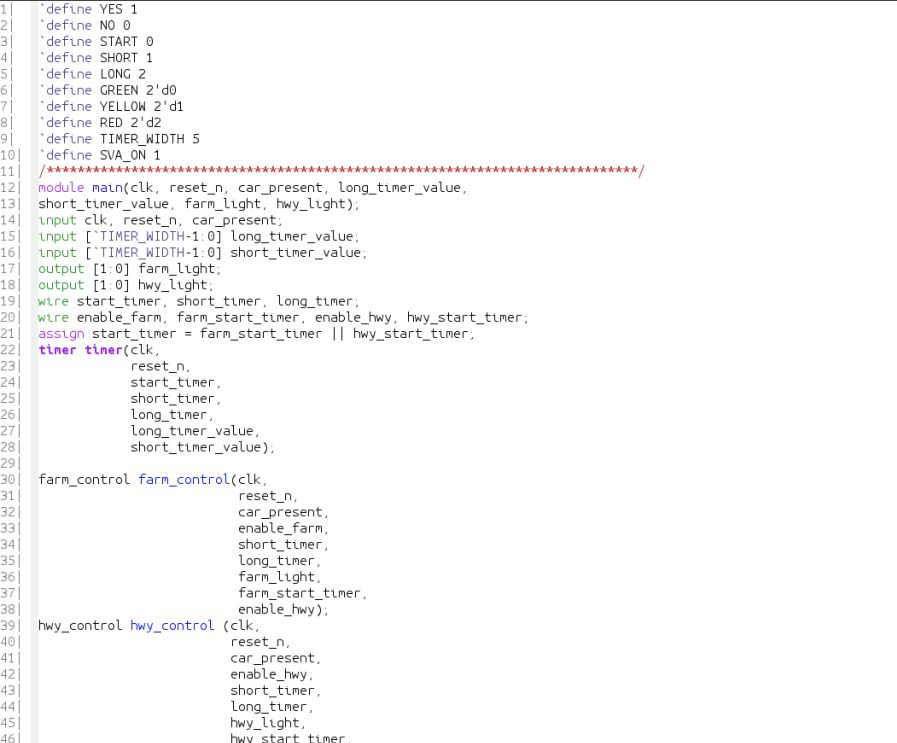
**设计描述**

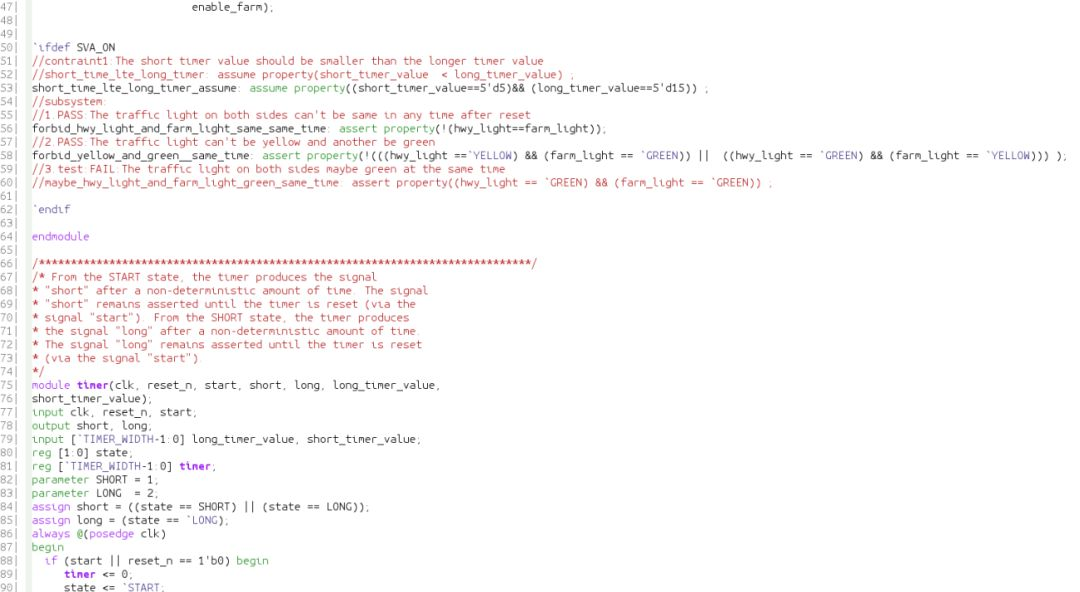
结合上述对crossroad交通灯信号控制子系统的规格要求，用Verilog语言对该系统进行设计，下图是该设计的实现框图：

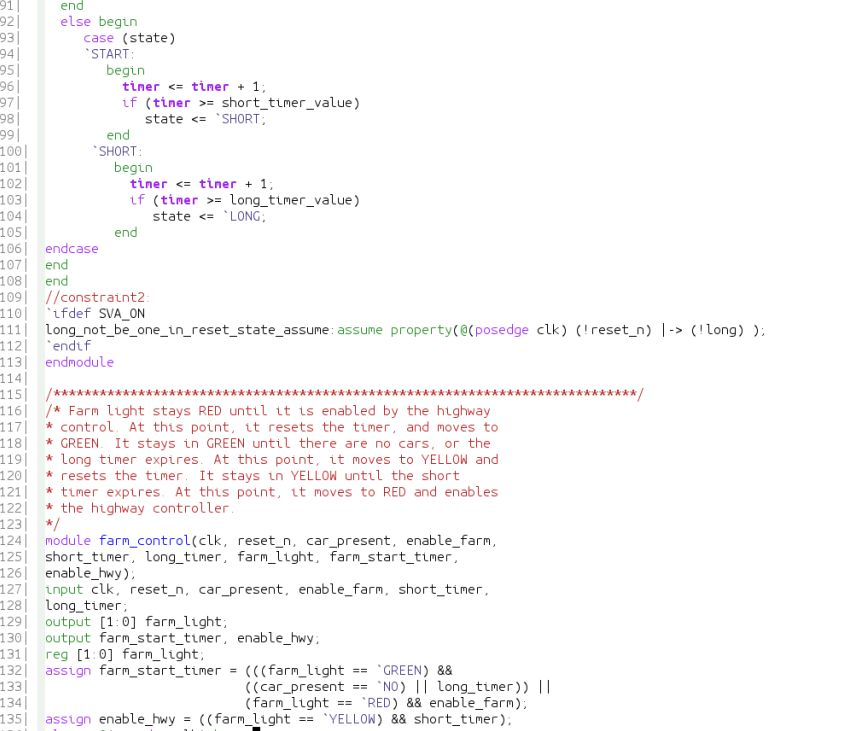


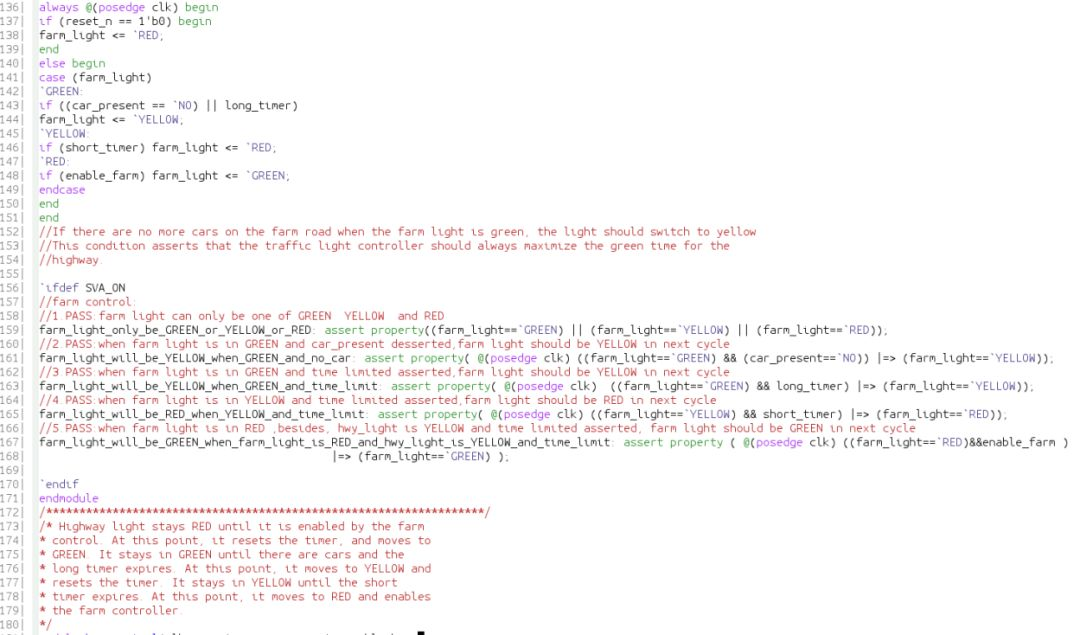
系统由三个子模块组成：timer(用来控制各个交通信号灯的产生时序)、farm\_control（用来控制纵向的交通信号灯）、hwy\_control(用来控制横向的交通信号灯)；系统的输入包括：clk（时钟）、reset\_n(复位)、car\_present(farm road方向有无车的标志输入)、long\_timer\_value(标志绿灯时长的限制时间输入)、short\_timer\_value(标志黄灯时长的限制时间输入)、farm\_light（farm  road 的信号灯输出）、hwy\_light(highwayroad的信号灯输出)。

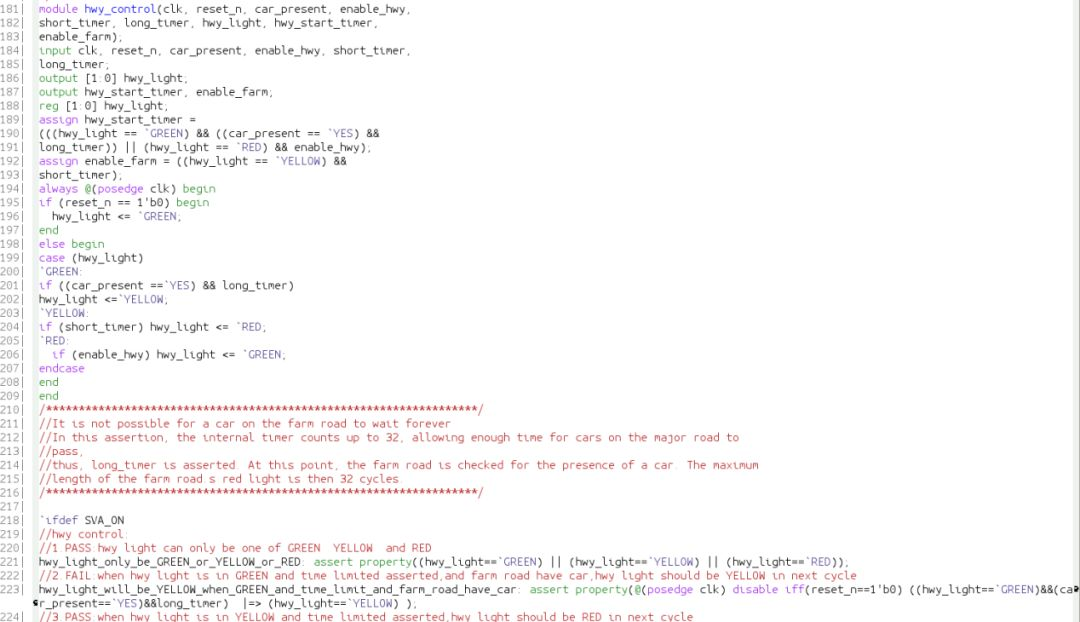
**详细的Verilog实现如下：**

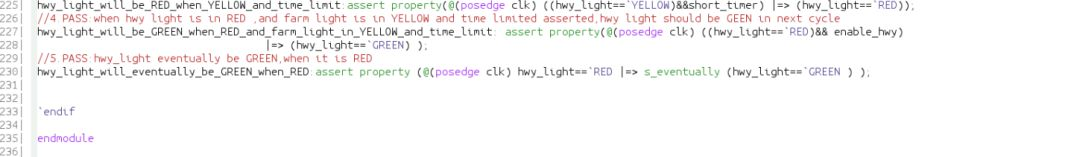












2、验证计划

**说明：**

为了充分验证整个子系统的功能正确性，制定了验证计划，对于模块级或较小子系统级的功能验证，采用形式化的验证方法可更可靠、高效。形式化验证过程中间会经过多个层次的抽象转移：设计规格→验证目标→SVA(OVL)描述→形式化验证工具模型，最后进行严格的数学方法证明，得出验证结果。

针对本设计的规格描述，对子系统及其中的各个子模块制定了后续的验证目标，验证时需要将这些设计规格描述转化成验证目标，再将验证目标转化成SVA（systemverilog assertion），并把这些assertions交给形式化验证工具AveMC进行验证，得到验证的结果。

**验证目标:**

验证目标制定的参照物是设计时的规格描述；将符合设计要求的关键点提取出来，作为验证目标，且需要做到完备，这样验证才完整；另外，如果在后续的验证中发现某些验证目标验证失败时，还可以将该目标相关的一些内部特性来作为额外的验证目标，来实现快速的debug。下表是关于该crossroad交通灯控制子系统的验证目标：

* 子系统:

规格目标:

⑴任何时候两个方向的交通灯信号都不同；

⑵为了保证安全，一个方向为黄灯时，不允许另一个方向为绿灯;

其他目标

⑴hwy\_light 和farm\_light的使能信号不会同时出现；

⑵hwy\_light和farm\_light的 start\_timer信号不会同时出现；

* FARM CONTROL子模块：

规格目标：

⑴farm Light信号灯只有绿、黄、红三种颜色；

⑵当farmLight信号灯为绿灯并且该方向没有车出现时，下个时钟该信号应该变黄灯；

⑶当farmLight信号灯为绿灯，且绿灯的最长时间限制已到，下个时钟周期该黄灯

⑷当farmLight信号灯为黄色时，且黄灯的限制时间已到，下个时钟周期该信号应该为红色；

⑸当farmLight信号灯为红色时，若此时hyw light的黄灯是亮着并且限制时间已到，下个时钟周期farm Light为绿灯；

...

* TIMER子模块：

规格目标：（无）

* HWY CONTROL子模块：

规格目标：

⑴hwy light信号灯只有绿、黄、红三种颜色；

⑵当hwyligh为绿灯，此时farm road有车，并且绿灯的最长限制时间已到，那下个时钟周期，hwyligh为黄灯；

⑶当hwy light为黄灯，并且黄灯的最长限制时间已到，那么下个时钟周期，hwy light为红灯；

⑷当hwy light为红灯，此时farm Light为黄灯，并且黄灯的限制时间已到，那么下个周期hwy light应该为绿灯；

(5)当hwy light为红灯时，最终将变成绿灯；

**SVA断言:**

针对验证目标，需要将其转化成形式化验证工具能识别的输入，在这里我们将它们转化成SystemVerilog Assertion,下表是与上面的每一个验证目标对应的断言（assertion）。

* 子系统：

规格目标：

(1)

forbid\_hwy\_light\_and\_farm\_light\_same\_same\_time:assert

property(!(hwy\_light==farm\_light));

(2)

forbid\_yellow\_and\_green\_same\_time:assert

property(!(((hwy\_light==`YELLOW) && (farm\_light ==`GREEN))||((hwy\_light == `GREEN) && (farm\_light == `YELLOW))) );

* FARM CONTROL子模块：

规格目标：

(1)

farm\_light\_only\_be\_GREEN\_or\_YELLOW\_or\_RED:assert

property((farm\_light==`GREEN)||(farm\_light==`YELLOW)|| (farm\_light==`RED));

(2)

farm\_light\_will\_be\_YELLOW\_when\_GREEN\_and\_no\_car:assert

property(@(posedge clk) ((farm\_light==`GREEN) && (car\_present==`NO)) |=> (farm\_light==`YELLOW));

(3)

farm\_light\_will\_be\_YELLOW\_when\_GREEN\_and\_time\_limit:assert

property(@(posedge clk)  ((farm\_light==`GREEN) &&long\_timer) |=> (farm\_light==`YELLOW));

(4)

farm\_light\_will\_be\_RED\_when\_YELLOW\_and\_time\_limit:assert

property(@(posedge clk) ((farm\_light==`YELLOW) && short\_timer) |=>(farm\_light==`RED));

(5)

farm\_light\_will\_be\_GREEN\_when\_farm\_light\_is\_RED\_and\_hwy\_light\_is\_YELLOW  \_and\_time\_limit:assert

property(@(posedge clk) ((farm\_light==`RED) && enable\_farm) |=> (farm\_light==`GREEN) );

* TIMER子模块：

规格目标：（无）

* HWY CONTROL子模块：

规格目标：

(1)

hwy\_light\_only\_be\_GREEN\_or\_YELLOW\_or\_RED:assert

property((hwy\_light==`GREEN) || (hwy\_light==`YELLOW) || (hwy\_light==`RED));

(2)

hwy\_light\_will\_be\_YELLOW\_when\_GREEN\_and\_time\_limit\_and\_farm\_road\_have\_car:assert

property(@(posedge clk) ((hwy\_light==`GREEN)&&(car\_present==`YES)&&long\_timer) |=> (hwy\_light==`YELLOW) );

(3)

hwy\_light\_will\_be\_RED\_when\_YELLOW\_and\_time\_limit:assert

property(@(posedge clk) ((hwy\_light==`YELLOW)&&short\_timer) |=>(hwy\_light==`RED));

(4)

hwy\_light\_will\_be\_GREEN\_when\_RED\_and\_farm\_light\_in\_YELLOW\_and\_time\_limit:assert

property(@(posedge clk) ((hwy\_light==`RED) && enable\_hwy) |=>(hwy\_light==`GREEN) );

(5)

hwy\_light\_will\_eventually\_be\_GREEN\_when\_RED:assert

property(@(posedge clk) hwy\_light==`RED |=> s\_eventually (hwy\_light==`GREEN ) );

3、AveMC使用简介

**验证流程：**

工程师将自己的Verilog设计和与设计相关的SVA提供给AveMC软件，软件会对其进行形式化验证，同时工程师可对整个软件的运行过程进行配置，然后将这一切交给软件进行严格的数学证明，给出证明的结果，对那些验证为fail的SVA，工程师可以利用工具对其进行调试，找出设计中存在的问题，接着修改此前的设计版本，得到新的设计版本，然后接着进行验证，如此循环迭代，直至最后所有的验证都通过，即设计得到充分性验证。

**典型使用流程：**

工程师使用AveMC软件读取设计,定义验证的对象,执行验证过程以及对验证失败的属性进行debug等，一系列操作都是通过软件的tcl命令执行，tcl命令可以在AveMc shell模式下交互式的执行，也可以以脚本的形式执行；

tcl命令可以被分为以下几种类别:1>Design setup:这些命令用来读入设计，定义时钟、复位、初始值和抽取设计中的assertions; 2>Assertion：用来增加、改变、移除属性、覆盖和约束; 3>Configuration:设置verification过程中的相关参数; 4>Verification:用来控制验证的过程; 5>Information：显示验证的状态信息以及设计和断言的信息; 6>Debug：产生波形和进行交互式debug。丰富、直观的命令形式可构建验证过程的所有场景。

下表是一个典型的验证流程的命令执行步骤：

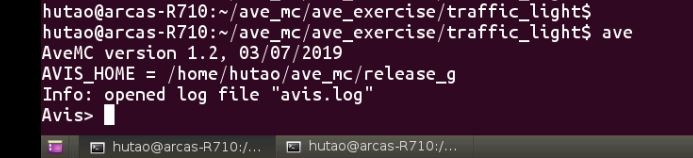
* 读入设计（in verilog2005 or in systemverilog2012）、指定top module（命令实现）；根据读入的设计，工具会进行语法分析、提取出设计中的OVL或者SVA断言、构建验证模型等（工具自动）；
* 定义时钟、复位：指定时钟及其周期、复位信号的有效值等（命令实现）；
* 定义约束；约束软件运行空间，模拟设计实际的运行环境（命令实现）
* 初始化：对寄存器等进行初始化赋值（命令实现）
* 配置verification过程；配置验证引擎、验证过程timeout时间等（命令实现）；
* 执行验证：（命令实现）；
* 报告结果：（命令实现）；
* Debug：使用工具的交互式图形界面；

4、实验部分

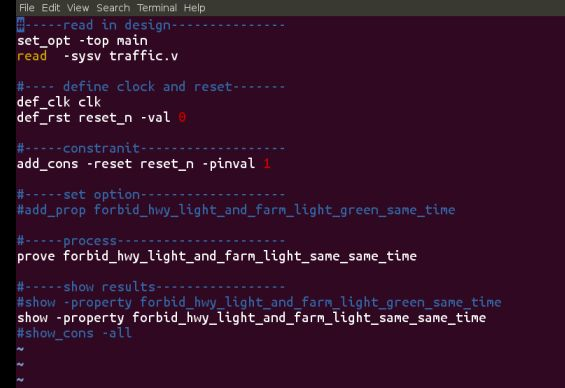
说明：范例使用脚本的形式运行AveMC软件，运行环境为Ubuntu。针对该交通灯信号控制子系统的设计，将使用两个例子来说明AveMC工具进行形式化验证的过程，一个验证成功的例子:来说明工具的使用流程，一个验证失败的例子来说明工具debug的过程。

**例子1：（验证成功）**

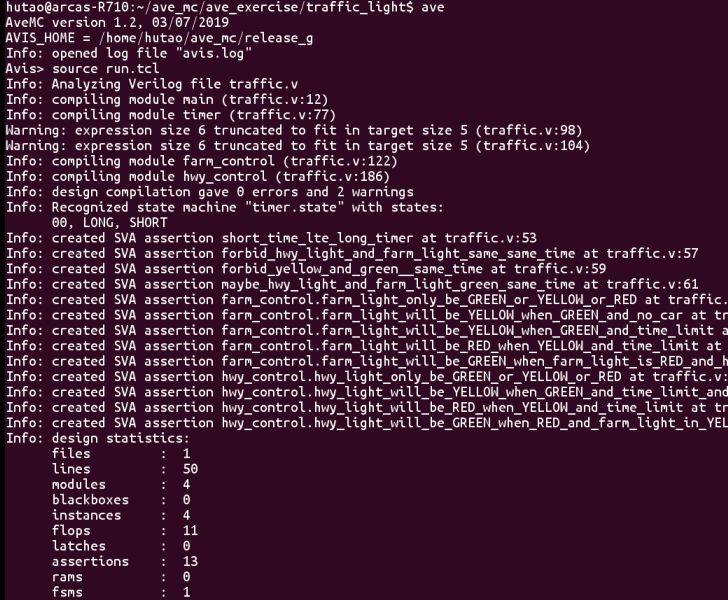
* 软件安装成功后，在shell命令行下输入“ave”,即可进入AveMC的软件运行环境，如下图：



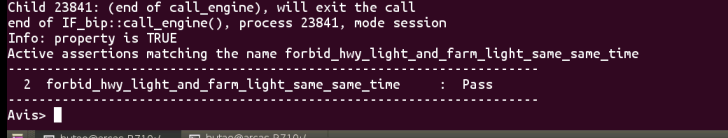
* 接着需要将我们执行的tcl脚本读入到工具中，输入命令“sourcerun.tcl”，tcl脚本的内容如下图：



* 从tcl的内容可以看出首先将设计读入到工具里面，然后定义了时钟与复位，接着添加约束：将复位信号约束为0，即验证过程中不发生复位，也可以在设计文件中添加约束（在设计文件中我们加了两条约束short\_time\_lte\_long\_timer\_assume和long\_not\_be\_one\_in\_reset\_state\_assume）；然后开始证明验证目标中对子系统验证的第一个断言,即forbid\_hwy\_light\_and\_farm\_light\_same\_same\_time，最后显示该属性的验证结果；上面的这一组简单的命令组合即可构建一个典型的验证流程；
* 回车执行source run.tcl命令后tcl脚本被读入工具中，并且会自动执行里面的命令，如下图：



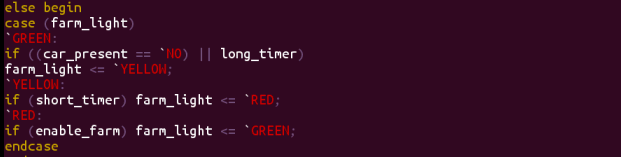
* 图中显示工具前端会分析Verilog设计文件里面的内容，并将分析的结果报告出来：比如HDL语法错误，assertions数量等，下图显示了执行tcl中最后一条命令的结果：

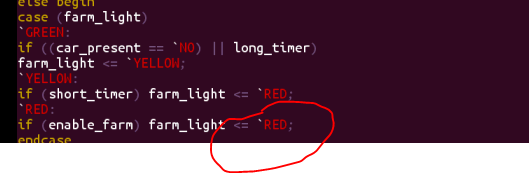


* 图中显示该assertion的状态是PASS，说明目前设计的交通灯信号控制系统满足了设计的第一个目标，后续可针对每一个目标asserton进行同样的验证过程，当所有的specificaton都满足，则可以说设计满足了要求。

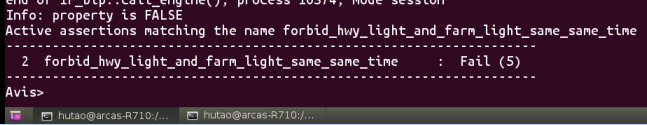
**例子2：（验证失败）**

* 我们针对第一个验证成功的例子，做了小的改动，如下图：

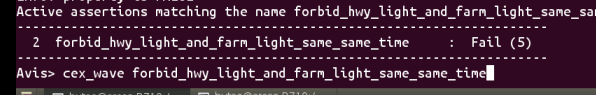




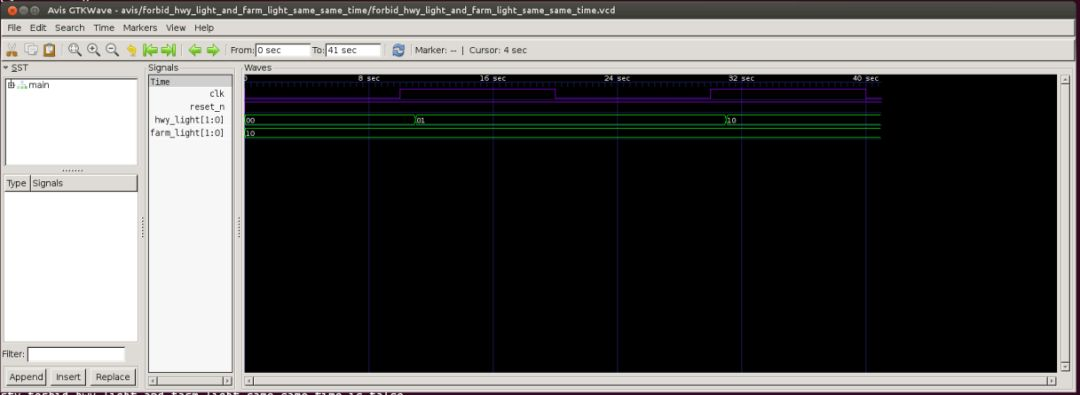
* 圆圈标注的是修改后的设计,原来的设计是如果highwayroad的交通为黄灯，并且时间已到，接下来其就该为红灯，同时farm road的交通灯接下来就是绿色，修改后farm road的交通灯接下来还是红色，很明显这与设计目标相违背，接下来我们执行与例子1相同的过程，得到的结果如下：



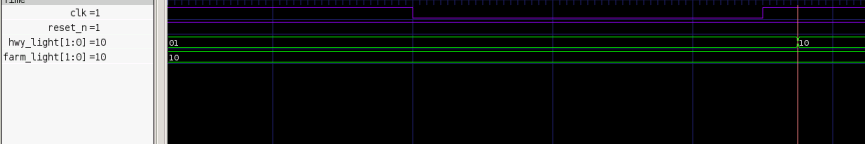
* 从验证的结果看，第一个验证目标失败，接着利用AveMC工具对设计进行debug，首先我们需要看在验证过程中的什么情况下出现了问题，在AveMC的环境中接着执行命令“cex\_waveforbid\_hwy\_light\_and\_farm\_light\_same\_same\_time”如下图；



* 回车执行命令会弹出如下的图形界面，工具会自动保存在验证过程中出问题时所有相关的信号的轨迹，在图形界面中，添加与属性验证相关的信号，进行分析；



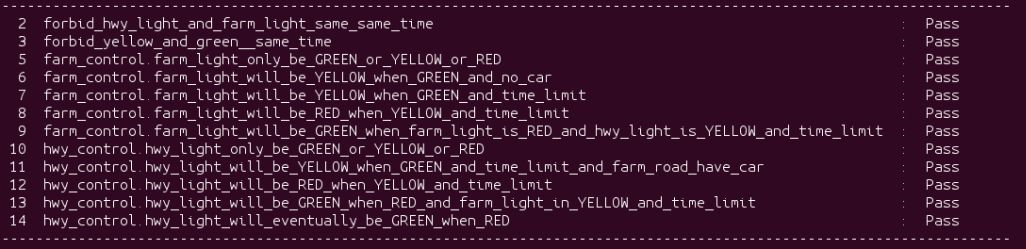
* 放大图形界面的窗口（下图），我们发现hwy\_light与farm\_light在图中的竖直标线处同时为2'b10，设计中定义2'b10为RED （`define RED 2'd2)，由此可知在设计中可能存在的问题：一种是farm light在红灯的转换过程有问题，另一种是 highway light 在红灯的过程中有问题；



* 有两种方式可快速的发现在HDL设计里的问题，一种是通过分析verilog编码在两个方向的交通灯在红灯时候的信号转化逻辑；另一种方式是可以在验证目标里增加farm light和highway light信号灯转换逻辑的验证目标，如果该目标验证失败，说明该处有问题，假如我们增加了一条assertion：当farm light为红灯时，满足条件后，其应该是绿灯，这样样也能定位到是farm light出了问题。两种方式都能很快定位到问题所在：即farm light在红灯变为绿灯时出错。

5、验证总结

针对上面的交通灯控制子系统，我们采用了三个约束，一个是复位信号的约束（验证过程中系统处于不复位的状态），另两个是与设计密切相关的参数设置的约束short\_time\_lte\_long\_timer\_assume和long\_not\_be\_one\_in\_reset\_state\_assume，依次来指明设计所处的真实环境；另外11个是验证目标断言assertion，来表明设计的需求规格；通过使用AveMC形式化验证工具，来完成对设计目标的功能验证与debug；最终我们所有的验证目标都完成了，下面是关于这一设计的所有验证目标结果截图：



**五、公司介绍**

成都奥卡思微电科技有限公司是中国本土的EDA公司，也是目前国内唯一专注于芯片数字前端的本土EDA公司。公司目前主要产品AveMC形式化验证EDA软件已经有多个客户使用。我们将致力于国产EDA的开发，为中国自主创”芯”事业添砖加瓦！

联系方式：王锐 wangrui@arcas-da.com

电话：028-83323800

网址：www.arcas-da.com