软件学报 ISSN 1000-9825, CODEN RUXUEW
Journal of Software,2021,32(6):1799−1817 [doi: 10.13328/j.cnki.jos.006250]

E-mail: jos@iscas.ac.cn
http://www.jos.org.cn
Tel: +86-10-62562563

芯片开发功能验证的形式化方法*

姚广宇[1,2], 张 南[1,2], 田 聪[1,2], 段振华[1,2], 刘灵敏[1,2], 孙风津[1,2]

[1](西安电子科技大学 计算理论与技术研究所,陕西 西安 710071)

[2](综合业务网理论及关键技术国家重点实验室(西安电子科技大学),陕西 西安 710071)

通讯作者: 张南, E-mail: nanzhang@xidian.edu.cn; 田聪, E-mail: ctian@mail.xidian.edu.cn; 段振华, E-mail: zhhduan@mail.xidian.edu.cn

摘 要: 在芯片设计领域,采用模型驱动的 FPGA 设计方法是目前较为安全可靠的一种方法.但是,基于模型驱动的 FPGA 设计需要证明 FPGA 设计模型和生成 Verilog/VHDL 代码的一致性;同时,芯片设计的正确性、可靠性和安全性也至关重要.目前,多采用仿真方法对模型和代码的一致性进行验证,很难保证设计的可靠性和安全性,并存在验证效率低、工作量大等问题.提出一种新型验证设计模型和生成代码一致性的方法,该方法利用 MSVL 语言进行系统建模,并通过模型提取命题投影时序逻辑公式描述的系统的性质,通过统一模型检测的原理,验证模型是否满足性质的有效性.进而,应用信号灯控制电路系统作为验证实例,对验证方法做了检验和说明.

关键词: 芯片设计;模型驱动;功能一致性;MSVL 建模;命题投影时序逻辑

中图法分类号: TP311

中文引用格式: 姚广宇,张南,田聪,段振华,刘灵敏,孙风津.芯片开发功能验证的形式化方法.软件学报,2021,32(6):1799−1817. http://www.jos.org.cn/1000-9825/6250.htm

英文引用格式: Yao GY, Zhang N, Tian C, Duan ZH, Liu LM, Sun FJ. Formal method of functional verification for chip development. Ruan Jian Xue Bao/Journal of Software, 2021,32(6):1799−1817 (in Chinese). http://www.jos.org.cn/1000-9825/6250.htm

Formal Method of Functional Verification for Chip Development

YAO Guang-Yu[1,2], ZHANG Nan[1,2], TIAN Cong[1,2], DUAN Zhen-Hua[1,2], LIU Ling-Min[1,2], SUN Feng-Jin[1,2]

[1](Institute of Computing Theory and Technology, Xidian University, Xi'an 710071, China)

[2](State Key Laboratory of Integrated Services Networks (Xidian University), Xi'an 710071, China)

Abstract: In the field of chip design, the use of model-driven FPGA design methods is currently a safer and more reliable method. However, model-driven FPGA design needs to prove the consistency of the FPGA design model and the generated Verilog/VHDL code. Further, the chip design correctness, performance, reliability, and safety are critical. At present, simulation methods are often used to verify the consistency of models and codes. It is difficult to ensure the reliability and safety of the design, and there are problems such as low verification efficiency and heavy workload. This study proposes a new method to verify the consistency of the design model and the generated code. This method uses the MSVL language to model the system, and propositional projection temporal logic (PPTL) formula to describe the properties of the system, then based on the principle of unified model checking, verifies whether the model meets the validity of the property. Furthermore, a signal light control system is used as a verification example to verify and explain the verification method.

Key words: chip design; model-driven; functional consistency; MSVL modeling; propositional projection temporal logic

* 基金项目: 国家重点研发计划(2018AAA0103202); 国家自然科学基金(61751207, 61732013); 陕西省重点科技创新团队(2019TD-001)

Foundation item: National Key Research and Development Program of China (2018AAA0103202); National Natural Science Foundation of China (61751207, 61732013); Key Science and Technology Innovation Team of Shaanxi Province(2019TD-001)

本文由"形式化方法与应用"专题特约编辑姜宇副教授推荐.

收稿时间: 2020-08-30; 修改时间: 2020-10-26; 采用时间: 2020-12-19; jos 在线出版时间: 2021-02-07

随着集成电路的规模越来越大,集成密度越来越高,相应的设计越来越复杂,电路设计已经由手工设计转向使用硬件描述语言(Verilog 或 VHDL)来完成电路设计.而使用硬件描述语言设计的电路可以经过行为综合、工艺映射、打包、布局和布线,快速的烧录到现场可编程门阵列(field-programmable gate array,简称 FPGA)[1,2]上进行测试,这也是现在集成电路设计的主流技术.

由于采用 FPGA 设计专用集成电路(application specific integrated circuit,简称 ASIC),用户可以在确认系统正确后才进行流片,所以能得到合格的芯片,它是 ASIC 电路中设计周期较短、开发费用较低、风险较小的方法之一.所以,FPGA 被广泛用于电信、汽车导航、通信卫星、航空航天等领域芯片开发[3−6].

模型驱动的 FPGA 设计方法能够利用模型驱动的优点,使设计更加安全可靠.特别是,能够使用图形语言描述和验证业务需求和系统设计,使得设计更合理、更稳定、更具适应性.因为设计是基于模型的,并且在构建模型前进行全面地分析,基于模型的规格说明设计系统时,更容易得到符合需求的设计模型.

而模型驱动的 FPGA 设计需要证明 FPGA 设计模型与硬件描述语言生成代码的一致性.这是芯片开发领域一个关键性问题.

国防科大[7]提出了一种系统级模型和寄存器级(register transfer level,简称 RTL)模型顺序等效检查方法.所提出的方法基于带有数据路径等效检查方法的有限状态机,使用机器学习技术从所有路径中识别有限状态机的对应路径序偶对.然后,通过符号仿真比较相应的路径序偶对,将所有路径分开对应的路径序偶对,避免了路径序偶对的盲目比较.这种方法可以处理截然不同的系统级模型和 RTL 级模型设计,并大大降低了基于路径的有限状态机等效性检查问题的复杂性.得克萨斯大学奥斯汀分校的 Vasudevan 等人[8]提出了一种用于顺序等效检查的系统级规范及其在 RTL 中实现的新颖技术,在源代码级别上为这些系统构建了 Kripke 结构或状态转换图,来进行模型和 RTL 代码的等价性验证.圣保罗大学的 Marquez 等人[9]提出了一种用于电子系统级规范和 RTL 实现之间的时序等价性检查形式和算法,该方法基于深度状态序列的概念以及序列之间的功能覆盖关系,用带数据路径的有限状态机分别对高级参考模型和待验证设计的行为进行建模,对模型的状态机中的每一个深度状态序列,如果都能在待验证设计的状态机中找到在功能上覆盖它的深度状态序列,说明待验证设计的状态机在功能上覆盖模型的状态机,从而验证了电子系统级规范和 RTL 描述之间的时序等价性.波特兰州立大学的 Yang[10]提出了一种等价性检查框架,该框架通过 3 个阶段来进行电子系统级规范和 RTL 实现的等价性验证:(1) 编译器转换;(2) 为每个操作分配一个时钟周期,并满足用户指定的约束;(3) 局部优化和 RTL 生成,然后逐个检查每个阶段的等价性.目前,已提出的仿真方法和一致性验证方法能够在一定程度上保障模型和代码的一致性,但是这些方法都存在验证效率低、工作量大等问题.本文提出了一种新的基于代码运行时验证方法来检查设计模型和生成代码的一致性.

运行时验证(runtime verification)[11]是近年来发展起来的一种新兴的验证技术.在运行时验证中,所关注的只有系统在运行时所表现出的行为,并在系统实际运行的时候对其监控,最终可根据系统运行时的状态信息来验证所期望的性质是否满足.

我们拟提出基于建模仿真验证语言(modeling simulation verification language,简称 MSVL)[12,13]的运行时验证方法,来验证模型和生成代码的一致性.该方法用 MSVL 语言描述系统的模型,用命题投影时序逻辑(propositional projection temporal logic,简称 PPTL)[14,15]公式来描述期望的性质.在实践中,我们通常将一个 Verilog 程序自动转换为 MSVL 程序 M_1,将期望的性质 P 的非转换为另一个 MSVL 程序 M_2,从而得到一个新的 MSVL 程序 M_3=M_1 and M_2.为了执行 M_3,我们将 M_3 通过 MSVL 编译器 MC[16]进行编译,得到可执行代码 M_3.exe.对 M_3.exe 选择合适的输入,运行 M_3.exe,若程序运行为真,则我们找到一个违反性质的一条路径;否则,我们仅知道当前执行是满足性质的.该方法的主要优点是:(1) 可以对大规模电路设计进行验证;(2) 验证过程可以自动化实现.

本文第 1 节主要介绍建模仿真验证语言 MSVL 的基本语法.第 2 节主要介绍命题投影时序逻辑的主要知识.第 3 节主要介绍使用的 V2M 转换工具.第 4 节主要介绍 UMC4M 验证工具框架和验证方法.第 5 节主要通过信号灯控制电路系统的例子进行建模和验证.第 6 节是对论文方法的总结概述.

1 建模仿真验证语言 MSVL

MSVL 是一种建模仿真和验证语言,它是投影时序逻辑的一个可执行子集.通过 MSVL,可以对软硬件系统进行建模,并根据使用逻辑公式描述的系统期望性质,进行软硬件系统的形式化验证.

MSVL 的语句定义如下.

语句名称	语法定义
空语句	empty
skip 语句	skip
区间长度语句	*len*(*n*)
下一状态赋值语句	*x*:=*e*
当前状态赋值语句	*x*<==*e*
选择语句	p_1 or p_2
并行语句	$p_1 \| \| p_2$
合取语句	p_1 and p_2
If 条件语句	**if** *b* **then** {p_1} **else** {p_2}
等待语句	*await*(*b*)
Always 语句	*always*(*p*)
顺序语句	$p_1;p_2$
投影语句	$(p_1,...,p_n)prjp$
Next 语句	*next*(*p*)
框架变量语句	*frame*(*x*)
While 循环语句	while (*b*) {*p*}
局部变量语句	*local x*:*p*

上述定义中,*e* 表示算术表达式,*b* 表示布尔表达式,它们的定义与 C 语言中的定义基本相同.另外,MSVL 中也定义了丰富的数据类型,包括整型、浮点型、布尔型、字符型、数组、指针和结构体.从程序上看,MSVL 简单易行,类似于 C 语言;从逻辑角度看,每一条语句又都是一个投影时序逻辑公式.

目前已经开发了 MSVL 编译器 MC 以及解释器,解释器支持 MSVL 程序一次执行过程的仿真和对 MSVL 程序所有路径的建模以及验证.

2 命题投影时序逻辑

命题投影时序逻辑(propositional projection temporal logic,简称 PPTL)是一种基于区间的时序逻辑,它的主要时序操作符是 *prj* 和 *next*.而命题区间时序逻辑(propositional interval temporal logic,简称 PITL)也是一种基于区间的时序逻辑,它的主要时序操作符是 *chop* 和 *next*.文献[17]证明,PPTL 在有穷和无穷区间都是可判定的.这一结论使我们可以将 PPTL 应用于模型检测.PPTL 公式的语法可定义如下:

$$P::=p|\circ P|\neg P|P_1\vee P_2|(P_1,\ldots,P_m)prjQ,$$

其中,$p\in$可数的命题变元集合 *Prop*,*P* 和 $P_1,\ldots,P_m,Q$ 都是 PPTL 合式公式.

一个 PPTL 公式 *P* 可用一个三元组 $I=(\sigma,i,j)$来解释,其中,$\sigma=\langle s_0,s_1,\ldots,\rangle$定义为非空的有穷或无穷状态序列,$|\sigma|$表示区间长度.若$\sigma$为无穷区间,则$|\sigma|=\omega$.定义$\preceq$为$\leqslant-\{(\omega,\omega)\}$.*i* 和 *j* 是整数并且 $i\preceq j\leqslant|\sigma|$.定义$(\sigma,i,j)\vDash P$ 表示 *P* 在σ的某一子区间$\langle s_i,\ldots,s_j\rangle$上可满足,其中,$s_i$ 表示当前状态,s_i:*Prop*$\rightarrow$*B*={true,flase}.可满足关系可定义如下.

1) $I\vDash p\Leftrightarrow$对任意原子命题 *p* 有 $s_i[p]$=true;
2) $I\vDash\neg P\Leftrightarrow I\nvDash P$;
3) $I\vDash P\vee Q\Leftrightarrow I\vDash P$ 或 $I\vDash Q$;

4) $I\models\circ P\Leftrightarrow i<j$ 并且$(\sigma,i+1,j)\models P$;

5) $I\models(P_1,\ldots,P_m)prjQ\Leftrightarrow$存在整数 $i=r_0\leqslant\ldots\preceq r_m\leqslant j$,使得$(\sigma,r_{l-1},r_l)\models P_l(1\leqslant l\leqslant m)$;且对如下两种$\sigma'$,有$(\sigma',0,|\sigma'|)\models Q$:

i. $r_m=j$ 且$\sigma'=\sigma\downarrow(r_0,\ldots,r_h),0\leqslant h\leqslant m$;

ii. $r_m<j$ 且 $\sigma'=\sigma\downarrow(r_0,\ldots,r_m)\cdot\sigma_{(r_{m+1},\ldots,j)}$.

如果$(\sigma,0,|\sigma|)\models P$,则表示公式 P 在区间σ上被满足,简写为$\sigma\models P$.若存在$\sigma,\sigma\models P$,则称公式 P 是可满足的;若对任意$\sigma,\sigma\models P$,则称公式 P 是有效的.因此,公式 P 有效等价于公式$\neg P$ 不可满足.

3 V2M 转换工具

V2M 转换工具是一个将 Verilog 语言自动转换成 MSVL 语言的工具,其基本原理是:将 Verilog 源程序通过词法分析和语法分析转换成语法树结构,该语法树结构可以作为一种中间表示形式,然后对中间表示的数据结构通过转换算法转换为 MSVL 程序.其中:词法分析可以将源程序识别成 token 流,进而交给语法处理器去处理;还可以自动将空格或者回车等无用成分进行过滤;并且识别错误信息,进行错误处理.词法分析的输出是 token 流,将词法分析的输出作为语法分析的输入,即可进行 Verilog 源程序的语法分析.语法分析的作用是分析 token 流,并且要排除语法错误,进而根据 token 流分析源代码的结构,最后生成语法树,作为一种中间表示.经过词法分析和语法分析的转换,Verilog 源程序已经从一种硬件描述语言的程序变成了以语法树作为中间表示的一种存储结构,将中间表示转换成 MSVL 程序的过程,就是将中间存储结构翻译成 MSVL 语言.Verilog 中的模块和函数对应 MSVL 语句中的 function 模块,在 Verilog 中,各种基本语句在 MSVL 语言中都有其一一对应的表达形式.这就是 V2M 工具的基本原理和基本组成结构.图 1 呈现了 Verilog 到 MSVL 转换器的工具流程图.

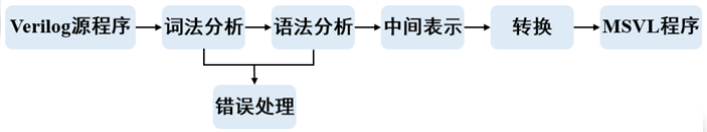

Fig.1 Flow chart of V2M tool

图 1 V2M 工具流程图

4 UMC4M 验证工具

UMC4M 是一种基于 MSVL 动态执行的代码级运行时验证工具,其原理是结合模型检测的一些策略,将程序的时序性质验证转换为 MSVL 程序源代码的动态执行.该工具输入为 MSVL 程序和 PPTL 公式,MSVL 程序用以描述系统的模型,而 PPTL 公式用以描述系统的性质.UMC4M 利用统一模型检测(UMC)原理[18],将检测模型是否满足性质的问题转化为程序的动态执行问题.验证过程如图 2 所示,主要步骤简要描述如下.

1) 利用工具 V2M 将待验证的 Verilog 程序转换为 MSVL 程序 M_1;

2) 利用 PPTL 公式 P 描述待验证的性质;

3) 利用工具 P2M[17]将$\neg p$ 转化为 MSVL 程序 M_2;

4) 利用 MSVL 编译器 MC 对新得到的 MSVL 程序 $M_3=M_1$ and M_2 进行编译,生成二进制代码 M_3.exe;

5) 选择电路中所有二进制变量一组合适的值(01 串)作为 M_3.exe 的输入;

6) 执行程序 M_3.exe:如果运行成功,则发现反例路径,即输入变量导致的运行轨迹;如果运行失败,则说明本次运行满足性质.

该方法与模型检测方法不同,模型检测方法是要检测模型是否满足性质的有效性,就是要检测性质是否被模型中的所有路径所满足.相比之下,对运行时验证方法,如所看到的程序的一次执行依赖于程序的一组输入,而且仅检查了一条执行路径.这种方法简单易行,可以对大规模的程序电路进行验证.但是该方法的挑战是:如何选择合适的输入集,使得程序运行能够覆盖所有路径.对于 n 个输入变量来说,它的输入集合是 2^n,这就迫使我

们要进行 2^n 次运行时验证.

该方法与测试方法不同,测试方法无论是白盒、黑盒或者其他方法,其基本原理都是基于枚举所有的测试用例的方法.虽然运行时验证方法也要通过选择不同的输入来检验模型是否满足期望的性质,但是一个性质能够代表一些测试用例的集合而不是一个单一的测试用例;同时,一个性质能够描述程序执行的动态性质,例如安全性($\Box p$)、活性($\Diamond q$)以及弱公平性($\Box(p\rightarrow\Diamond q)$)等.

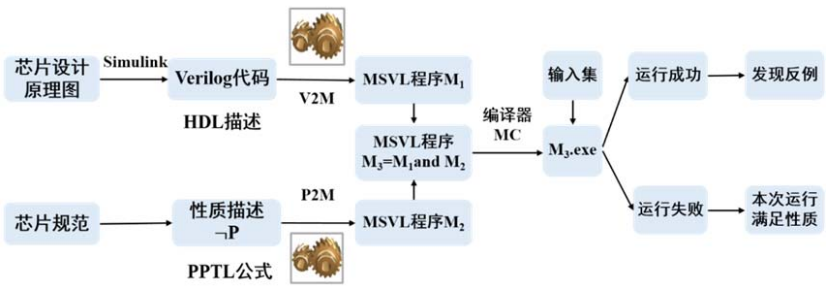

Fig.2 UMC4M verification tool

图 2 UMC4M 验证工具

5 信号灯控制电路系统的建模与验证

利用基于 MSVL 的运行时验证技术和 UMC4M 验证工具,可以进行模型和代码等价性的验证.本节以信号灯控制电路系统为例,说明该方法在小电路系统验证上的可行性.

5.1 系统功能需求分析

实现一个简易交通信号灯控制电路系统,该系统能够确保东西主干道和南北支干道上的车辆能够交替安全行驶.系统需要实现如下功能:系统关闭时,4 个方向黄灯持续亮,以警示来往车辆注意行驶.系统启动后,任意方向的信号灯都是红黄绿交替闪亮,周期为 60s.为确保安全,主干道和支干道的绿灯不能同时亮.根据功能需求,可以将整个系统划分为秒脉冲发生模块、分频计时器模块和状态控制模块,具体原理图如图 3 所示.

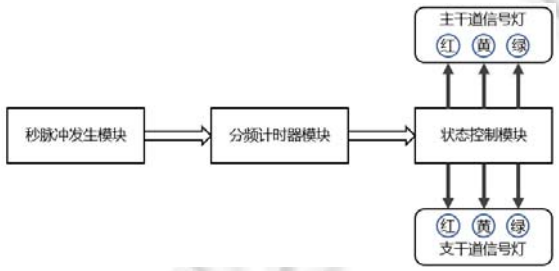

Fig.3 Schematic diagram of signal light control system

图 3 交通信号灯控制系统原理图

系统在启动前会一直保持警示状态,启动后,系统处于工作状态,并在达到相应的时间条件下进行状态变换,如图 4 所示.

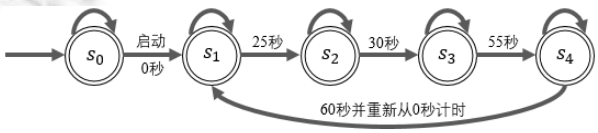

Fig.4 System state transition diagram

图 4 系统状态迁移图

具体迁移过程如下.

1) 状态 s_0:系统关闭时,4 个方向黄灯持续亮;
2) 状态 s_1:主干道绿灯亮 25s,支干道红灯亮;
3) 状态 s_2:主干道黄灯亮 5s,支干道红灯亮;
4) 状态 s_3:主干道红灯亮,支干道绿灯亮 25s;
5) 状态 s_4:主干道红灯亮,支干道黄灯亮 5s.

5.2 V2M工具转换

在分析了信号灯控制系统功能需求之后,利用 Verilog 代码对秒脉冲发生模块、分频计时器模块、状态控制模块进行功能描述,描述后的顶层代码如图 5 所示,全部代码在附录 1 中可以查看.

```
module signal_light_top(count,clk,rst,light1,light2);

input clk,rst;

output[2:0] light1,light2;

output[5:0] count;

wire[5:0] count;


counter u2(clk,rst,count);

signal_light u1(clk,rst,count,light1,light2);


endmodule
```

Fig.5 The top-level code of the signal light control system

图 5 信号灯控制系统顶层代码

通过命令./v2m.exe *signal_light_top.v signal_light_top.m* 调用 V2M 工具,将 Verilog 代码转换成 MSVL 代码,得到 *signal_light_top.m*.得到的部分 MSVL 代码如图 6 所示,全部 MSVL 代码可以在附录 2 中查看.

```
frame(signal_light_top_clk,signal_light_top_rst,reg_signal_light_top_light1,signal_light
(
output("input variable is signal_light_top_clk") and skip;
output("input variable is signal_light_top_rst") and skip;
output("output variable is signal_light_top_light1[0]") and skip;
output("output variable is signal_light_top_light1[1]") and skip;
output("output variable is signal_light_top_light1[2]") and skip;
output("output variable is signal_light_top_light2[0]") and skip;
output("output variable is signal_light_top_light2[1]") and skip;
output("output variable is signal_light_top_light2[2]") and skip;
output("output variable is signal_light_top_count[0]") and skip;
output("output variable is signal_light_top_count[1]") and skip;
output("output variable is signal_light_top_count[2]") and skip;
output("output variable is signal_light_top_count[3]") and skip;
output("output variable is signal_light_top_count[4]") and skip;
output("output variable is signal_light_top_count[5]") and skip;
int every_i<==0 and skip;
unsigned int TEMP<==0 and skip;
int cycle_num<==0 and skip;
input(cycle_num) and skip;
output(cycle_num) and skip;
int reg_counter_count<==0 and int counter_count[0] and skip;
int reg_signal_light_light1<==0 and int signal_light_light1[0] and skip;
int reg_signal_light_light2<==0 and int signal_light_light2[0] and skip;
int counter_counter_count<==0 and skip;
int signal_light_signal_light_light1<==0 and skip;
int signal_light_signal_light_light2<==0 and skip;
int signal_light_state[0] and skip;
int reg_signal_light_state<==0 and skip;
int now and now<==0 and skip;
int*signal_light_top_clk and skip;
signal_light_top_clk:=(int*)malloc(cycle_num*sizeof(int));
now<==0 and skip;
while(now<cycle_num)
{
if(now%24=0)then
{
input(TEMP) and skip
}
else
```

Fig.6 MSVL code of the signal light control system

图 6 信号灯控制系统的 MSVL 代码

在利用 V2M 工具进行转换的过程中,需要判断转换结果正确与否.可以采用如下方法:转化的结果是 MSVL 代码,可以对 MSVL 代码使用 MSVL 编译器进行编译执行,如果执行的结果与源 Verilog 代码在 iverilog 下仿真的结果一致,则证明转换正确,不一致则转换错误.具体可通过对比波形和对比 01 结果串两种方式进行判断.

以交通信号灯控制电路系统为测试用例,对判断转化的 MSVL 代码是否正确的过程进行说明.

首先对测试用例在 iverilog 软件上进行编译和逻辑仿真,结果如图 7 所示.

主模块中有两个 1 位的输入信号 *clk*,*rst* 和两个 3 位的输出信号 *light*1,*light*2.*clk* 与 *rst* 代表着时钟信号和复位信号,*light*1 与 *light*2 代表两个方向的信号灯的状态.

将交通信号灯控制电路系统的源代码用 V2M 转换器进行转换,转换后的 MSVL 代码在编译器上进行编译执行,然后输入与 Verilog 仿真输入相同的数据,通过 01 结果串对比,即,将 MSVL 程序运行结果的每位的输出值与 iverilog 下仿真值进行比较.为了便于观察也可通过波形比较他们结果是否一致,从图 7 和图 8 中我们可以看出两者结果一致,证明 V2M 转换器在转换交通信号灯控制电路系统这一测试用例时,能正确转换.

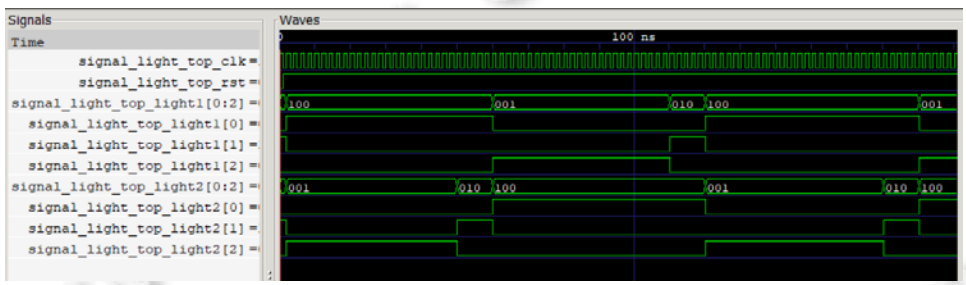

Fig.7 Verilog simulation result

图 7 Verilog 仿真结果

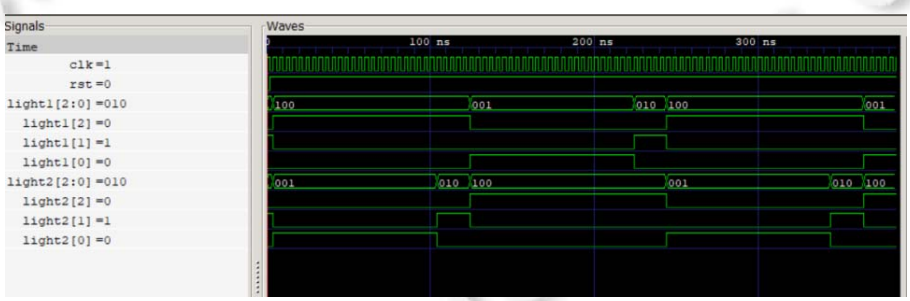

Fig.8 MSVL simulation result

图 8 MSVL 仿真结果

5.3 提取性质

从系统的功能需求中提取待验证的性质,并使用 PPTL 公式描述.

使用 $p_0,p_1,p_2,p_3,p_4,p_5,p_6$ 分别表示启动系统、主干道红灯亮、主干道黄灯亮、主干道绿灯亮、支干道红灯亮、支干道黄灯亮和支干道绿灯亮这 7 个原子命题,则可以提取以下几条性质.

性质 1. 如果系统没启动,那么保持所有方向黄灯亮:

$$Q_1=\Box(\neg p_0\rightarrow(p_2\wedge p_5)).$$

性质 2. 将来某个时刻主干道绿灯亮,且将来某个时刻支干道绿灯亮:

$$Q_2=\Box(p_0\rightarrow(\Diamond p_3\wedge\Diamond p_6)).$$

性质 3. 在任意时刻,主干道和支干道绿灯不能同时亮:

$$Q_3=\Box\neg(p_3\wedge p_6).$$

性质 4. 在任意时刻,任一方向的红黄绿这 3 个灯同时只能亮一个,以主干道为例:

$$Q_4=\Box((p_1\oplus p_2\oplus p_3)\wedge\neg(p_1\wedge p_2\wedge p_3)).$$

5.4 性质描述文件

根据 UMC4M 工具的输入要求,将 PPTL 公式转化为 *signal_light.p* 文件.性质 1 的.*p* 文件的格式如下.

〈/

define *p*0:*signal_light_top_rst*=1;

define *p*1:*signal_light_top_light*1[2]=1;

define *p*2:*signal_light_top_light*1[1]=1;

define *p*3:*signal_light_top_light*1[0]=1;

define *p*4:*signal_light_top_light*2[2]=1;

define *p*5:*signal_light_top_light*2[1]=1;

define *p*6:*signal_light_top_light*2[0]=1;

alw(!$p_0\rightarrow$(p_2 and p_5))

/〉

性质 2 的.*p* 文件的格式如下:

〈/

define p_0:*signal_light_top_rst*=1;

define p_1:*signal_light_top_light*1[2]=1;

define p_2:*signal_light_top_light*1[1]=1;

define p_3:*signal_light_top_light*1[0]=1;

define p_4:*signal_light_top_light*2[2]=1;

define p_5:*signal_light_top_light*2[1]=1;

define p_6:*signal_light_top_light*2[0]=1;

alw($p_0\rightarrow$(*som*(p_3) and *som*(p_6)))

/〉

性质 3 的.*p* 文件的格式如下:

〈/

define p_0:*signal_light_top_rst*=1;

define p_1:*signal_light_top_light*1[2]=1;

define p_2:*signal_light_top_light*1[1]=1;

define p_3:*signal_light_top_light*1[0]=1;

define p_4:*signal_light_top_light*2[2]=1;

define p_5:*signal_light_top_light*2[1]=1;

define p_6:*signal_light_top_light*2[0]=1;

alw(!(p_3 and p_6))

/〉

性质 4 的.*p* 文件的格式如下:

〈/

define p_0:*signal_light_top_rst*=1;

define p_1:*signal_light_top_light*1[2]=1;

define p_2:*signal_light_top_light*1[1]=1;

define p_3:*signal_light_top_light*1[0]=1;

define p_4:*signal_light_top_light*2[2]=1;

define p_5:*signal_light_top_light*2[1]=1;

define p_6:*signal_light_top_light*2[0]=1;

alw((!p_1 and !p_2 and p_3) or (!p_1 and p_2 and !p_3) or (p_1 and !p_2 and !p_3))

/⟩

5.5 利用工具进行验证

将 MSVL 程序与*.p* 文件作为 UMC4M 工具的输入,进行运行时验证,在终端输入命令:

UMC4M.exe *signal_light.p signal_light.m*

等待运行结束就可以得到运行结果.以性质 2 验证为例,其验证结果如图 9 所示.

```
ThreadPool init success!!!
input variable is signal_light_top_clkinput variable is signal_light_top_rstoutp
ut variable is signal_light_top_light1[0]output variable is signal_light_top_lig
ht1[1]output variable is signal_light_top_light1[2]output variable is signal_lig
ht_top_light2[0]output variable is signal_light_top_light2[1]output variable is
signal_light_top_light2[2]output variable is signal_light_top_count[0]Verificati
on Result: Valid!!!
release ThreadPool success!!!
output variable is signal_light_top_count[1]The runtime verification time is: 23
4ms
All time is: 1857ms
```

Fig.9 Validation result

图 9 验证结果

5.6 实验结果

然后,根据运行结果判断 PPTL 公式对于 MSVL 程序是否有效,从而可以判断被提取的性质对于 Verilog 代码是否有效.所有性质的验证结果见表 1.

Table 1 Experimental result

表 1 实验结果

性质	PPTL 公式	验证结果	运行验证时间	运行总时间
性质 1	Q_1=□(¬p_0→(p_2∧p_5))	有效	312ms	4353ms
性质 2	Q_2=□(p_0→(◇p_3∧◇p_6))	有效	226ms	1857ms
性质 3	Q_3=□¬(p_3∧p_6)	有效	218ms	1832ms
性质 4	Q_4=□((p_1⊕p_2⊕p_3)∧¬(p_1∧p_2∧p_3))	有效	253ms	2884ms

6 结 论

本文通过使用基于 MSVL 的代码级运行时验证技术,完成了对模型建模和功能验证,将模型的 Verilog 代码通过 V2M 工具转换成 MSVL 程序,完成 MSVL 建模,再使用 PPTL 公式将模型的性质提取出来,最后使用 UMC4M 验证工具,将性质和 MSVL 程序一起运行,得到验证结果.本文采用信号灯控制电路系统作为实验用例,完成了整体实验流程和最后实验结果的正确输出.将来的工作是:通过该验证方法,可以对更多的模型进行功能验证.

References:

[1] Alexander MJ, Cohoon JP, Ganley JL, Robins G. Performance-Oriented placement and routing for field-programmable gate arrays. In: Proc. of the European Design Automation Conf. (EURO-DAC). IEEE, 1995. 80–85.

[2] Andraka R. A survey of CORDIC algorithms for FPGA based computers. In: Proc. of the 1998 ACM/SIGDA 6th Int'l Symp. on Field Programmable Gate Arrays. New York: Association for Computing Machinery, 1998. 191–200.

[3] Parsons S, Taylor DE, Schuehler DV, Franklin MA, Chamberlain RD. High speed processing of financial information using FPGA devices. United States Patent US8655764 B2. 2013.

[4] Guan Y, Yuan Z, Sun G, Cong J. FPGA-Based accelerator for long short-term memory recurrent neural networks. In: Proc. of the 22nd Asia and South Pacific Design Automation Conf. IEEE, 2017. 629−634.

[5] Ma YF, Suda N, Cao Y, Seo J, Vrudhula S. Scalable and modularized RTL compilation of convolutional neural networks onto FPGA. In: Proc. of the 26th Int'l Conf. on Field Programmable Logic and Applications. IEEE, 2016. 1−8.

[6] Tolba MF, AbdelAty AM, Soliman NS, Said LA, Madian AH, Azar AT, Radwan AG, *et al*. FPGA implementation of two fractional order chaotic systems. AEU—Int'l Journal of Electronics and Communications, 2017,78:162−172.

[7] Hu J, Li T, Li S. Equivalence checking between SLM and RTL using machine learning techniques. In: Proc. of the 17th Int'l Symp. on Quality Electronic Design. Santa Clara: IEEE, 2016. 129−134.

[8] Vasudevan S, Abraham JA, Viswanath V, Tu JJ. Automatic decomposition for sequential equivalence checking of system level and RTL descriptions. In: Proc. of the 4th ACM and IEEE Int'l Conf. on Formal Methods and Models for Co-design. Napa: IEEE, 2006. 71−80.

[9] Marquez CIC, Strum M, Chau WJ. Formal equivalence checking between high-level and RTL hardware designs. In: Proc. of the 14th Latin American Test Workshop (LATW). Cordoba: IEEE, 2013. 1−6.

[10] Yang Z. Scalable equivalence checking for behavioral synthesis [Ph.D. Thesis]. Broadway: Portland State University, 2015.

[11] Wang Z, Li C, Hu S, Man J. A framework on runtime verification for software behavior. In: Proc. of the 2nd Int'l Conf. on Intelligent System Design and Engineering Application. Sanya: IEEE, 2012. 20−23.

[12] Duan ZH, Yang X, Koutny M. Framed temporal logic programming. Science of Computer Programming, 2008,70(1):31−61.

[13] Duan ZH, Koutny M. A framed temporal logic programming language. Journal of Computer Science and Technology, 2004,19(3): 341−351.

[14] Duan ZH. Temporal logic and temporal logic programming. Beijing: Science Press, 2005.

[15] Duan ZH, Tian C, Zhang N. A canonical form based decision procedure and model checkingapproach for propositional projection temporal logic. Theoretical Computer Science, 2016,609:544−560.

[16] Yang K, Duan ZH, Tian C, Zhang N. A compiler for MSVL and its applications. Theoretical Computer Science, 2018,749:2−16.

[17] Duan ZH, Tian C. A practical decision procedure for propositional projection temporal logic with infinite models. Theoretical Computer Science, 2014,554:169−190.

[18] Duan ZH, Tian C. A unified model checking approach with projection temporal logic. In: Proc. of the Int'l Conf. on Formal Engineering Methods. Berlin, Heidelberg: Springer-Verlag, 2008. 167−186.

附录 1. 信号灯控制电路系统 Verilog 代码

```
顶层模块
`include "counter.v"
`include "signal_light.v"
module signal_light_top(count,clk,rst,light1,light2);
input clk, rst;
output [2:0] light1, light2;
output [5:0] count;
wire [5:0] count;
counter u2(clk,rst,count);
signal_light u1(clk,rst,count,light1,light2);
endmodule
计数器模块
module counter(clk,rst,count);
output [5:0] count;
input clk, rst;
reg [5:0] count;
always@(posedgeclk or negedgerst)
  begin
    if (!rst)
    count<='d0;
    else if (count<'d60)
    count<=count+1;
    else
    count<='d1;
  end
endmodule
```

```
信号灯模块
module
signal_light(clk,rst,count,light1,light2);
input clk, rst;
input [5:0] count;
output [2:0] light1, light2;
reg [2:0] light1, light2;
reg [2:0] state;
parameter
  Idle=3′b000,
  S1=3′b001,
  S2=3′b010,
  S3=3′b011,
  S4=3′b100;
always@(posedgeclk)
  begin
    if (!rst)
      begin
        state<=Idle;
        light1<=3′b010;
        light2<=3′b010;
      end
    else
      case(state)
Idle: if (rst)
        begin
          state<=S1;
          light1<=3′b100;
          light2<=3′b001;
        end
S1: if (count==′d24)
  begin
    state<=S2;
    light1<=3′b100;
    light2<=3′b010;
  end
S2: if (count==′d29)
  begin
    state<=S3;
    light1<=3′b001;
    light2<=3′b100;
  end
S3: if (count==′d54)
  begin
    state<=S4;
    light1<=3′b010;
    light2<=3′b100;
  end
S4: if (count==′d59)
  begin
    state<=S1;
    light1<=3′b100;
    light2<=3′b001;
  end
default: state<=Idle;
endcase
  end
endmodule
```

附录 2. 转换得到的信号灯控制电路系统 MSVL 代码

```
function Bit_not(int a,intRValue)
{if (a=1) then {RValue:=0}
else
{RValue:=1}};
function counter(int counter_clk,intcounter_rst)
{if (counter_clk=Old_counter_clk+1
OR counter_rst=Old_counter_rst−1)
then {if (counter_rst≤0)
then {reg_counter_count:=0;
TEMP<==reg_counter_count and skip;
every_i:=0;
```

while (*every_i*≤5)
{*counter_count*[*every_i*]:=*TEMP*%2;
every_i:=*every_i*+1;
TEMP:=*TEMP*/2}}
else {
if (*reg_counter_count*<60)
then {*reg_counter_count*:=*reg_counter_count*+1;
TEMP<==*reg_counter_count* and *skip*;
every_i:=0;
while (*every_i*≤5)
{*counter_count*[*every_i*]:=*TEMP*%2;
every_i:=*every_i*+1;
TEMP:=*TEMP*/2}}
else {*reg_counter_count*:=1;
TEMP<==*reg_counter_count* and *skip*;
every_i:=0;
while (*every_i*≤5)
{*counter_count*[*every_i*]:=*TEMP*%2;
every_i:=*every_i*+1;
TEMP:=*TEMP*/2}}}}
else {empty};
counter_counter_count:=*reg_counter_count*;
Old_counter_clk_temp:=*counter_clk*;
Old_counter_rst_temp:=*counter_rst*};
function *signal_light*(*int signal_light_clk*,*intsignal_light_rst*,*intreg_signal_light_count*)
{*frame*(*signal_light_count*) and (
int signal_light_count[6] and *skip*;
TEMP<==*reg_signal_light_count* and *skip*;
every_i:=0;
while (*every_i*≤5)
{*signal_light_count*[*every_i*]:=*TEMP*%2;
every_i:=*every_i*+1;
TEMP:=*TEMP*/2};
if (*signal_light_clk*=*Old_signal_light_clk*+1)
then {**if** (*signal_light_rst*≤0)
then {*reg_signal_light_state*:=*signal_light_Idle*;
TEMP<==*reg_signal_light_state* and *skip*;
every_i:=0;
while (*every_i*≤2)
{*signal_light_state*[*every_i*]:=*TEMP*%2;
every_i:=*every_i*+1;
TEMP:=*TEMP*/2};

*reg_signal_light_light*1:=2;
TEMP<==*reg_signal_light_light*1 and *skip*;
every_i:=0;
while (*every_i*≤2)
{*signal_light_light*1[*every_i*]:=*TEMP*%2;
every_i:=*every_i*+1;
TEMP:=*TEMP*/2};
*reg_signal_light_light*2:=2;
TEMP<==*reg_signal_light_light*2 and *skip*;
every_i:=0;
while (*every_i*≤2)
{*signal_light_light*2[*every_i*]:=*TEMP*%2;
every_i:=*every_i*+1;
TEMP:=*TEMP*/2}}
else {**if** (*reg_signal_light_state*=*signal_light_Idle*)
then {**if** (*signal_light_rst*≥1)
then {*reg_signal_light_state*:=*signal_light_S*1;
TEMP<==*reg_signal_light_state* and *skip*;
every_i:=0;
while (*every_i*≤2)
{*signal_light_state*[*every_i*]:=*TEMP*%2;
every_i:=*every_i*+1;
TEMP:=*TEMP*/2};
*reg_signal_light_light*1:=4;
TEMP<==*reg_signal_light_light*1 and *skip*;
every_i:=0;
while (*every_i*≤2)
{*signal_light_light*1[*every_i*]:=*TEMP*%2;
every_i:=*every_i*+1;
TEMP:=*TEMP*/2};
*reg_signal_light_light*2:=1;
TEMP<==*reg_signal_light_light*2 and *skip*;
every_i:=0;
while (*every_i*≤2)
{*signal_light_light*2[*every_i*]:=*TEMP*%2;
every_i:=*every_i*+1;
TEMP:=*TEMP*/2}}
else {empty}}
else {**if** (*reg_signal_light_state*=*signal_light_S*1)
then {**if** (*reg_signal_light_count*=25)
then {*reg_signal_light_state*:=*signal_light_S*2;
TEMP<==*reg_signal_light_state* and *skip*;

every_i:=0;
while (*every_i*≤2)
{*signal_light_state*[*every_i*]:=*TEMP*%2;
every_i:=*every_i*+1;
TEMP:=*TEMP*/2};
*reg_signal_light_light*1:=4;
TEMP<==*reg_signal_light_light*1 and *skip*;
every_i:=0;
while (*every_i*≤2)
{*signal_light_light*1[*every_i*]:=*TEMP*%2;
every_i:=*every_i*+1;
TEMP:=*TEMP*/2};
*reg_signal_light_light*2:=2;
TEMP<==*reg_signal_light_light*2 and *skip*;
every_i:=0;
while (*every_i*≤2)
{*signal_light_light*2[*every_i*]:=*TEMP*%2;
every_i:=*every_i*+1;
TEMP:=*TEMP*/2}}
else {empty}}
else {**if** (*reg_signal_light_state*=*signal_light_S*2)
then {**if** (*reg_signal_light_count*=30)
then {*reg_signal_light_state*:=*signal_light_*S3;
TEMP<==*reg_signal_light_state* and *skip*;
every_i:=0;
while (*every_i*≤2)
{*signal_light_state*[*every_i*]:=*TEMP*%2;
every_i:=*every_i*+1;
TEMP:=*TEMP*/2};
*reg_signal_light_light*1:=1;
TEMP<==*reg_signal_light_light*1 and *skip*;
every_i:=0;
while (*every_i*≤2)
{*signal_light_light*1[*every_i*]:=*TEMP*%2;
every_i:=*every_i*+1;
TEMP:=*TEMP*/2};
*reg_signal_light_light*2:=4;
TEMP<==*reg_signal_light_light*2 and *skip*;
every_i:=0;
while (*every_i*≤2)
{*signal_light_light*2[*every_i*]:=*TEMP*%2;
every_i:=*every_i*+1;

TEMP:=*TEMP*/2}}
else {empty}}
else
{**if** (*reg_signal_light_state*=*signal_light_S*3)
then {**if** (*reg_signal_light_count*=55)
then {*reg_signal_light_state*:=*signal_light_S*4;
TEMP<==*reg_signal_light_state* and *skip*;
every_i:=0;
while (*every_i*≤2)
{*signal_light_state*[*every_i*]:=*TEMP*%2;
every_i:=*every_i*+1;
TEMP:=*TEMP*/2};
*reg_signal_light_light*1:=2;
TEMP<==*reg_signal_light_light*1 and *skip*;
every_i:=0;
while (*every_i*≤2)
{*signal_light_light*1[*every_i*]:=*TEMP*%2;
every_i:=*every_i*+1;
TEMP:=*TEMP*/2};
*reg_signal_light_light*2:=4;
TEMP<==*reg_signal_light_light*2 and *skip*;
every_i:=0;
while (*every_i*≤2)
{*signal_light_light*2[*every_i*]:=*TEMP*%2;
every_i:=*every_i*+1;
TEMP:=*TEMP*/2}}
else {empty}}
else {**if** (*reg_signal_light_state*=*signal_light_S*4)
then {**if** (*reg_signal_light_count*=60)
then {*reg_signal_light_state*:=*signal_light_S*1;
TEMP<==*reg_signal_light_state* and *skip*;
every_i:=0;
while (*every_i*≤2)
{*signal_light_state*[*every_i*]:=*TEMP*%2;
every_i:=*every_i*+1;
TEMP:=*TEMP*/2};
*reg_signal_light_light*1:=4;
TEMP<==*reg_signal_light_light*1 and *skip*;
every_i:=0;
while (*every_i*≤2)
{*signal_light_light*1[*every_i*]:=*TEMP*%2;
every_i:=*every_i*+1;

TEMP:=*TEMP*/2};
*reg_signal_light_light*2:=1;
TEMP<==*reg_signal_light_light*2 and *skip*;
every_i:=0;
while (*every_i*≤2)
{*signal_light_light*2[*every_i*]:=*TEMP*%2;
every_i:=*every_i*+1;
TEMP:=*TEMP*/2}}
else {empty}}
else {*reg_signal_light_state*:=*signal_light_Idle*;
TEMP<==*reg_signal_light_state* and *skip*;
every_i:=0;
while (*every_i*≤2)
{*signal_light_state*[*every_i*]:=*TEMP*%2;
every_i:=*every_i*+1;
TEMP:=*TEMP*/2
}}}}}}}}
else {empty};
*signal_light_signal_light_light*1:=*reg_signal_light_light*1;
*signal_light_signal_light_light*2:=*reg_signal_light_light*2;
Old_signal_light_clk_temp:=*signal_light_clk*)};

frame(*signal_light_top_clk*,*signal_light_top_rst*,*reg_signal_light_top_light*1,*signal_light_top_light*1,*reg_signal_light_top_light*2,*signal_light_top_light*2,*reg_signal_light_top_count*,*signal_light_top_count*,*TEMP*,*cycle_num*,*now*,*Old_counter_clk*,*Old_counter_clk_temp*,*Old_counter_rst*,*Old_counter*_rst*_temp*,*Old_signal_light_clk*,*Old_signal_light_clk_temp*,*signal_light_Idle*,*signal_light_S*1,*signal_light_S*2,*signal_light_S*3,*signal_light_S*4,*counter_counter_count*,*signal_light_signal_light_light*1,*signal_light_signal_light_light*2,*reg_signal_light_state*,*signal_light_state*,*reg_counter_count*,*counter_count*,*reg_signal_light_light*1,*signal_light_light*1,*reg_signal_light_light*2,*signal_light_light*2,*every_i*) and
(output (“input variable is *signal_light_top_clk*”) and *skip*;
output (“input variable is *signal_light_top_rst*”) and *skip*;
output (“output variable is *signal_light_top_light*1[0]”) and *skip*;
output (“output variable is *signal_light_top_light*1[1]”) and *skip*;
output (“output variable is *signal_light_top_light*1[2]”) and *skip*;
output (“output variable is *signal_light_top_light*2[0]”) and *skip*;
output (“output variable is *signal_light_top_light*2[1]”) and *skip*;
output (“output variable is *signal_light_top_light*2[2]”) and *skip*;
output (“output variable is *signal_light_top_count*[0]”) and *skip*;
output (“output variable is *signal_light_top_count*[1]”) and *skip*;
output (“output variable is *signal_light_top_count*[2]”) and *skip*;
output (“output variable is *signal_light_top_count*[3]”) and *skip*;
output (“output variable is *signal_light_top_count*[4]”) nd *skip*;
output (“output variable is *signal_light_top_count*[5]”) and *skip*;

int every_i<==0 and *skip*;
unsigned *int TEMP*<==0 and *skip*;
int cycle_num<==8 and *skip*;
input (*cycle_num*) and *skip*;
output (*cycle_num*) and *skip*;
int reg_counter_count<==0 and *int counter_count*[6] and *skip*;
*int reg_signal_light_light*1<==0 and *int signal_light_light*1[3] and *skip*;
int reg*_signal_light_light*2<==0 and *int signal_light_light*2[3] and *skip*;
int counter_counter_count<==0 and *skip*;
*int signal_light_signal_light_light*1<==0 and *skip*;
*int signal_light_signal_light_light*2<==0 and *skip*;
int signal_light_state[3] and *skip*;
int reg_signal_light_state<==0 and *skip*;
int now and *now*<==0 and *skip*;
*int***signal_light_top_clk* and *skip*;
signal_light_top_clk:=(*int**)*malloc*(*cycle_num***sizeof*(*int*));
now<==0 and *skip*;
while (*now*<*cycle_num*)
{**if** (*now*%24=0) **then** {input (*TEMP*) and *skip*}
else {empty};
signal_light_top_clk[*now*]<==*TEMP*%2 and *skip*;
TEMP<==*TEMP*/2 and *skip*;
now<==*now*+1 and *skip*};
now<==0 and *skip*;
*int***signal_light_top_rst* and *skip*;
signal_light_top_rst:=(*int**)*malloc*(*cycle_num***sizeof*(*int*));
now<==0 and *skip*;
while (*now*<*cycle_num*){**if** (*now*%24=0) **then**
{input (*TEMP*) and *skip*}
else {empty};
signal_light_top_rst[*now*]<==*TEMP*%2 and *skip*;
TEMP<==*TEMP*/2 and *skip*;
now<==*now*+1 and *skip*};
now<==0 and *skip*;
*int signal_light_top_light*1[3],*reg_signal_light_top_light*1 and *reg_signal_light_top_light*1<==0 and *skip*;
*int signal_light_top_light*2[3],*reg_signal_light_top_light*2 and *reg_signal_light_top_light*2<==0 and *skip*;
int signal_light_top_count[6],*reg_signal_light_top_count* and *reg_signal_light_top_count*<==0 and *skip*;
int signal_light_Idleand signal_light_Idle<==0 and *skip*;
signal_light_Idle<==0 and *skip*;
*int signal_light_S*1 and *signal_light_S*1<==0 and *skip*;
*signal_light_S*1<==1 and *skip*;
*int signal_light_S*2 and *signal_light_S*2<==0 and *skip*;

*signal_light_S*2<==2 and *skip*;
*int signal_light_S*3 and *signal_light_S*3<==0 and *skip*;
*signal_light_S*3<==3 and *skip*;
*int signal_light_S*4 and *signal_light*_S4<==0 and *skip*;
*signal_light_S*4<==4 and *skip*;
int Old_counter_clkand Old_counter_clk:=0;
int Old_counter_clk_tempand Old_counter_clk_temp:=0;
int Old_counter_rstand Old_counter_rst:=0;
int Old_counter_rst_tempand Old_counter_rst_temp:=0;
int Old_signal_light_clkand Old_signal_light_clk:=0;
int Old_signal_light_clk_tempand Old_signal_light_clk_temp:=0;
while (*now*<*cycle_num*){
counter(*signal_light_top_clk*[*now*],*signal_light_top_rst*[*now*]);
reg_signal_light_top_count:=*counter_counter_count*;
TEMP<==*reg_signal_light_top_count* and *skip*;
every_i:=0;
while (*every_i*≤5)
{*signal_light_top_count*[*every_i*]:=*TEMP*%2;
every_i:=*every_i*+1;
TEMP:=*TEMP*/2};
signal_light(*signal_light_top_clk*[*now*],*signal_light_top_rst*[*now*],*reg_signal_light_top_count*);
reg_signal_light_top_light1:=*signal_light_signal_light_light*1;
TEMP<==*reg_signal_light_top_light*1 and *skip*;
every_i:=0;
while (*every_i*≤2)
{*signal_light_top_light*1[*every_i*]:=*TEMP*%2;
every_i:=*every_i*+1;
TEMP:=*TEMP*/2};
*reg_signal_light_top_light*2:=*signal_light_signal_light_light*2;
TEMP<==*reg_signal_light_top_light*2 and *skip*;
every_i:=0;
while (*every_i*≤2)
{*signal_light_top_light*2[*every_i*]:=*TEMP*%2;
every_i:=*every_i*+1;
TEMP:=*TEMP*/2};
Old_counter_clk:=*Old_counter_clk_temp*;
Old_counter_rst:=*Old_counter_rst_temp*;
Old_signal_light_clk:=*Old_signal_light_clk_temp*;
output (“interval”) and *skip*;
output (*signal_light_top_clk*[*now*]) and *skip*;
output (*signal_light_top_rst*[*now*]) and *skip*;
output (*signal_light_top_light*1[2]) and *skip*;

output (*signal_light_top_light*1[1]) and *skip*;
output (*signal_light_top_light*1[0]) and *skip*;
output (*signal_light_top_light*2[2]) and *skip*;
output (*signal_light_top_light*2[1]) and *skip*;
output (*signal_light_top_light*2[0]) and *skip*;
output (*signal_light_top_count*[5]) and *skip*;
output (*signal_light_top_count*[4]) and *skip*;
output (*signal_light_top_count*[3]) and *skip*;
output (*signal_light_top_count*[2]) and *skip*;
output (*signal_light_top_count*[1]) and *skip*;
output (*signal_light_top_count*[0]) and *skip*;
now:=*now*+1})

姚广宇(1995－),男,博士生,CCF 学生会员,主要研究领域为 FPGA 布局,形式化验证.

张南(1984－),女,博士,副教授,博士生导师,CCF 高级会员,主要研究领域为形式化验证.

田聪(1981－),女,博士,教授,博士生导师,CCF 杰出会员,主要研究领域为软件安全:程序漏洞分析,恶意代码检测.

段振华(1948－),男,博士,教授,博士生导师,CCF 会士,主要研究领域为高可信软件开发及验证.

刘灵敏(1995－),男,硕士生,主要研究领域为 Verilog 代码和模型功能一致性验证.

孙风津(1995－),男,硕士生,主要研究领域为 Verilog-MSVL 代码转换技术,形式化验证.