Продолжение. Начало в № 3 `2008

Проектирование в условиях временных ограничений: верификация проектов

Ростислав ГРУШВИЦКИЙ
RIGrushvitsky@mail.eltech.ru
Максим Михайлов
yamaksya@yandex.ru

Авторы продолжают рассмотрение вопросов верификации аппаратуры, и потому включили в данную статью краткий обзор возможностей применения открытой библиотеки верификации OVL (Open Verification Library).

Открытая библиотека верификации (OVL)

Как уже отмечалось во вступительной статье цикла, структурная и функциональная сложность современных систем весьма внушительна — миллионы логических вентилей, функции сотен и тысяч переменных. Это делает актуальной задачу верификации, то есть подтверждения того, что описание проекта, по которому микросхема будет реализована «в кремнии», полностью соответствует спецификации (техническому заданию) проектируемой системы [1].

Под спецификацией проекта понимают по возможности сжатое описание всех значимых аспектов поведения системы, например, с целью проверки того, что они не противоречат некоторому набору исходных требований. Это может быть описание в форме требований к характеристикам функционирования или к функциональному составу и условиям реализации.

Отмеченная сложность проектируемых систем означает невозможность «ручного» анализа правильности результатов разработки и указывает на то, что процедура верификации может быть очень длительной. Между тем, в современной электронной промышленности отрезок времени от начала разработки до выхода на рынок — один из наиболее критичных экономических показателей.

Использование в качестве инструмента верификации языка **PSL** хотя и оказывается эффективным, но все же сопряжено с некоторыми существенными трудностями. Применение языка требует высокой квалификации разработчика и хорошего знания базовых конструкций. При этом объем кода, создаваемого для целей верификации, весьма значителен.

Пожалуй, самым простым и доступным средством повышения скорости и надежности описания верифицируемого проекта являются мониторы утверждений библиотеки **OVL**, разработанной фирмой **Accellera**. Существует две версии библиотеки **OVL**, функциональность которых определяется, прежде всего, особенностями используемых языков описания аппаратуры — **VHDL** или **Verilog**. Разработанные фирмой модули покрывают широкий диапазон типовых проблем верификации. Отлаженное функционирование библиотечных элементов облегчает обнаружение ошибок. Полный список всех мониторов, предназначенных для проверки специфических свойств систем, находится в свободном доступе в Интернете по адресу **http://www.accellera.org/activities/ovl/**. Другим не менее важным ресурсом является сайт пользователей библиотекой **OVL** — **http://www.eda.org/ovl/**.

Мониторы утверждений — это объявленные в проекте компоненты, цель использования которых — гарантирование выполнения некоторых условий. Здесь прослеживается явное сходство с понятием утверждение (*assertion*). Мониторы должны проверять истинность указанных событий и в случае неудачи сообщать об этом проектировщику, а также классифицировать ошибку, в зависимости от серьезности обнаруженного расхождения между предполагаемым и свершившимся событием. При этом допустимо использование мониторов как для моделирования, так и для формальной верификации.

Набор мониторов призван совершенствовать верификационную процедуру и увеличить интерес к подобной методологии проектирования. При размещении мониторов утверждений в RTL-коде следует придерживаться следующих основных правил:

- Вставлять мониторы утверждений, чтобы зафиксировать все проектные предположения для RTL-кодирования или узловые проблемы проекта.
- Включать мониторы утверждений, когда модуль имеет внешний интерфейс. В этом случае должны гарантироваться и верифицироваться предположения о корректности поведения входных и выходных сигналов.
- Включать мониторы утверждений, когда предполагается подключение модулей сторонних фирм, так как проектировщики чаще всего не имеют представления о внутренней структуре таких блоков (как в случае с IP-ядрами) или могут не до конца понимать их работу. В этих случаях защита модуля мониторами утверждений может предотвратить некорректное использование модуля.

Обычно каждый монитор утверждений ассоциируется с определенным свойством, которое формулирует потенциальную проблему.

Обязательными параметрами для любого монитора **OVL** являются:

- **severity_level** — определяет уровень важности ошибки, зафиксированной монитором. Конкретные значения этого параметра зависят от используемой САПР. Например, в пакете QuestaSim в библиотеке std описаны следующие возможности: note, warning, error и failure.
- **options** — дополнительный параметр, характеризующий специфические свойства мониторов. Также во многом зависит от САПР.
- **msg** — строка, отображаемая на экране при срабатывании монитора.

Разные мониторы могут иметь различное количество и назначение портов. Общим является только то, что все эти порты входные.

Ознакомление с применением мониторов **OVL** удобнее всего начать с анализа конкретного примера. В качестве такового была выбрана модель управления светофором.

Описание системы

Контроллер светофора предназначен для управления дорожным движением на перекрестках путем переключения соответствующих сигналов. Эта задача — достаточно тривиальна и часто описывается в различных источниках для пояснения принципов использования языковых средств. В руководстве по работе с библиотекой **OVL** [2] приведена реализация подобного алгоритма на языке **Verilog**. Однако в данной статье прямому заимствованию этого кода авторы

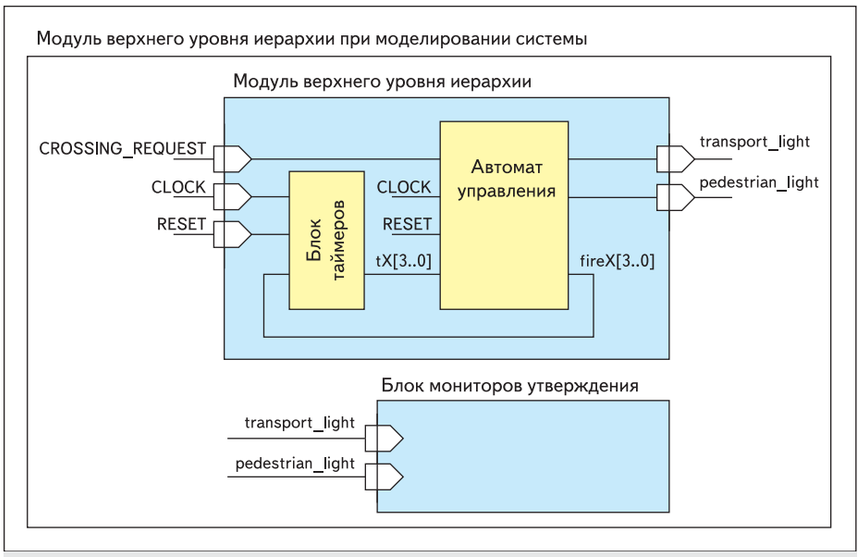

Рис. 1. Структурная схема устройства

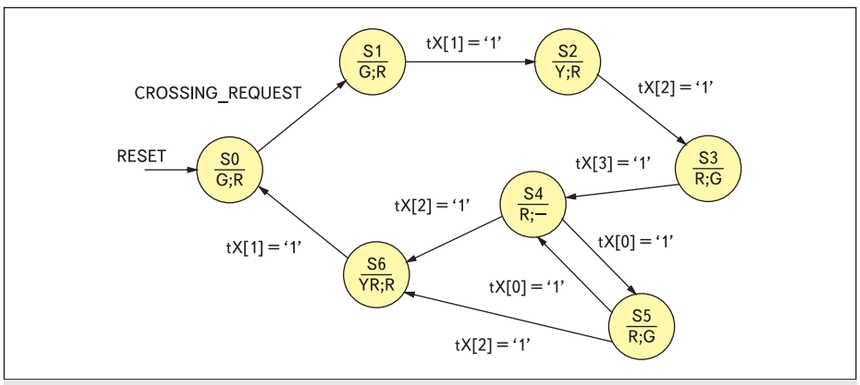

Рис. 2. Граф автомата управления светофором

предпочли написание собственной модели на языке **VHDL**. Структурная схема разрабатываемого устройства приведена на рис. 1.

Назначение отдельных модулей:

- Автомат управления, он реализует основной алгоритм работы светофора. Граф переходов и выходов автомата приведен на рис. 2.

По сигналу сброса ***RESET*** автомат переходит в состояние S0 и остается в нем до появления сигнала ***CROSSING_REQUEST*** (пешеход нажал кнопку — запрос на переход перекрестка). При переходе в состояние S1 запускается таймер 1 (***fireX[1] <= '1'***). При его переполнении активируется сигнал ***tX[1]*** и автомат переходит в состояние S2. Аналогичным образом происходит запуск таймера 2, переход в S3, запуск таймера 3, переход в S4. Переключение между состояниями S4–S5 (с периодом, управляемым таймером 0) имитирует мигание зеленого сигнала светофора для пешехода. Длительность мигания определяется таймером 2. Далее через промежуточное состояние S6 автомат возвращается в начальное состояние S0.

Выходные сигналы автомата (***TRANSPORT_LIGHT и PEDESTRIAN_LIGHT*** — сигналы светофора для транспорта и пешеходов соответственно) определяются текущим состоянием автомата. На рис. 2 цвета закодированы следующим образом: G (GREEN) — зеленый, Y (YELLOW) — желтый, R (RED) — красный.

- Блок таймеров реализует временные задержки сигналов светофора.
- Модуль верхнего уровня иерархии задает структуру светофора.
- Блок мониторов утверждения. Текст HDL-программы содержит только один монитор — ***assert_never***, подробная работа которого будет рассмотрена далее.

Таблица. Назначение сигналов

Сигнал	Назначение
CLK	Тактирование системы
RESET	Сброс ('1' — активное значение)
CROSSING_REQUEST	Запрос пешехода на переход через дорогу
TRANSPORT_LIGHT	Сигналы светофора для транспорта
PEDESTRIAN_LIGHT	Сигналы светофора для пешехода
tX[3..0]	Признаки срабатывания таймеров 3..0
fireX[3..0]	Сигналы включения таймеров 3..0

Назначение отдельных сигналов схемы, приведенной на рис. 1, представлено в таблице.

Листинг HDL-программы

Для удобства выполнения практических работ с приведенным далее HDL-кодом он может быть в любое время получен в электронном виде. Для этого достаточно связаться с авторами по указанным адресам электронной почты.

В компоненте DUT реализован **testbench** для проведения моделирования в среде **QuestaSim**. Кроме того, здесь же формируется логическое выражение, результат которого (сигнал BOOL) будет отслеживать монитор утверждений, и объявляется сам монитор — both_are_green:

Device_Under_Test.vhd

```
LIBRARY IEEE;
  USE IEEE.std_logic_1164.ALL;

LIBRARY WORK;
  USE WORK.tl_lib.ALL;
  USE WORK.ovl_assert.ALL; -- include OVL components description

ENTITY DUT IS
END DUT;

ARCHITECTURE etu OF DUT IS

  SIGNAL sCLK, sRESET, sCROSSING_REQUEST : std_logic;
  SIGNAL sTRANSPORT_LIGHT, sPEDESTRIAN_LIGHT : light;

SIGNAL sRESET_n : std_logic;

SIGNAL BOOL : BOOLEAN;

BEGIN

  sRESET_n <= NOT sRESET;

  traffic_light_inst : traffic_light
  PORT MAP (sCLK, sRESET, sCROSSING_REQUEST,
sTRANSPORT_LIGHT, sPEDESTRIAN_LIGHT);

  BOOL <= (sTRANSPORT_LIGHT = GREEN AND sPEDESTRIAN_
LIGHT = GREEN);

  both_are_green: assert_never
  GENERIC MAP (note, 0, «TRAFFIC ACCIDENT»)
  PORT MAP (clk => sCLK, reset_n => sRESET_n, test_expr => BOOL);

  clock_generator: PROCESS
  BEGIN
      sCLK <= '0';
      WAIT FOR 10 ns;
      sCLK <= '1';
      WAIT FOR 10 ns;
  END PROCESS;

  sRESET <= '1', '0' AFTER 20 ns;

  sCROSSING_REQUEST <= '0', '1' AFTER 60 ns, '0' AFTER 80 ns;

END etu;
```

Компонент traffic_light описывает структуру светофора: взаимосвязь контроллера светофора (controller) и четырех таймеров (timerX_inst):

Traffic_light.vhd

```
LIBRARY IEEE;
  USE IEEE.std_logic_1164.ALL;

LIBRARY WORK;
  USE WORK.tl_lib.ALL;

ENTITY traffic_light IS
```

```
  PORT (
                          CLK   : IN std_logic;
                        RESET   : IN std_logic;
            CROSSING_REQUEST    : IN std_logic;
             TRANSPORT_LIGHT    : OUT light;
             PEDESTRIAN_LIGHT   : OUT light
            );
END traffic_light;

ARCHITECTURE etu OF traffic_light IS

  SIGNAL TX, FIREX : std_logic_vector(3 DOWNTO 0);

  SIGNAL i : INTEGER RANGE 0 TO 3;

BEGIN

  ontroller : traffic_light_controller
  PORT MAP(
            RESET,
            CLK,
            CROSSING_REQUEST,
            TX,
            FIREX,
            TRANSPORT_LIGHT,
            PEDESTRIAN_LIGHT
            );

  timeout : FOR i IN 0 TO 3 GENERATE
     timerX_inst : timerX
     GENERIC MAP (timeout_value(i))
     PORT MAP(RESET, CLK, FIREX(i), TX(i));
  END GENERATE;

END etu;
```

В пакете tl_lib происходит объявление необходимых компонентов, типов, констант и т. д.:

Traffic_Light_Library.vhd

```
LIBRARY IEEE;
  USE IEEE.std_logic_1164.ALL;

PACKAGE tl_lib IS

  TYPE FSM_state IS (S0, S1, S2, S3, S4, S5, S6);
  TYPE light IS (GREEN, YELLOW, RED, YELLOW_and_RED, OFF);
  TYPE timer_state IS (INIT, COUNT, INTERRUPT);
  TYPE timeout_values IS ARRAY (integer range 0 to 3) OF std_logic_
vector(5 downto 0);
  CONSTANT timeout_value : timeout_values := (
                                                «000010»,
                                                «011010»,
                                                «011010»,
                                                «101000»
                                                );

  CONSTANT prescale : std_logic_vector(20 downto 0) :=
«100000000000000000000»;

  COMPONENT traffic_light_controller
     PORT (
                        Reset   : IN std_logic;
                          Clk   : IN std_logic;
              crossing_request  : IN std_logic;
                           tX   : IN std_logic_vector(3 downto 0);
                        fireX   : OUT std_logic_vector(3 downto 0);
               transport_light  : OUT light;
             pedestrian_light   : OUT light
              );
  END COMPONENT;

  COMPONENT timerX
  GENERIC (
             TIMEOUT_VALUE    : std_logic_vector(5 downto 0)
             );
  PORT (
            Reset   : IN std_logic;
              clk   : IN std_logic;
             fire   : IN std_logic;
                t   : OUT std_logic
          );
  END COMPONENT;

  COMPONENT traffic_light IS
  PORT (
                          CLK   : IN std_logic;
                      RESET_N   : IN std_logic;
          CROSSING_REQUEST      : IN std_logic;
           TRANSPORT_LIGHT      : OUT light;
           PEDESTRIAN_LIGHT     : OUT light
           );
  END COMPONENT;

END tl_lib;
```

Компонент traffic_light_controller реализует основной алгоритм устройства управления работой светофора (описан с помощью конечного автомата):

Traffic_Light_Controller.vhd

```
LIBRARY IEEE;
  USE IEEE.std_logic_1164.ALL;

LIBRARY WORK;
  USE WORK.tl_lib.ALL; -- traffic light package

ENTITY traffic_light_controller IS
  PORT (
                      Reset   : IN std_logic;
                        Clk   : IN std_logic;
            crossing_request  : IN std_logic;
                         Tx   : IN std_logic_vector(3 downto 0);
                      fireX   : OUT std_logic_vector(3 downto 0);
             transport_light  : OUT light;
           pedestrian_light   : OUT light
          );
END traffic_light_controller;

ARCHITECTURE etu OF traffic_light_controller IS

  SIGNAL current_state, next_state : FSM_state;

BEGIN

  transition_event: PROCESS (clk, reset)
  BEGIN
     IF reset = '1' THEN
        current_state <= S0;
     ELSIF RISING_EDGE( clk ) THEN
        current_state <= next_state;
     END IF;
  END PROCESS;

  transition_and_output_logic: PROCESS (reset, clk, current_state,
crossing_request, tX)
  BEGIN
     CASE current_state IS

        WHEN S0 => -- initial state after reset
                   -- transport traffic light is GREEN
                   -- pedestrian traffic light is RED

                   transport_light <= GREEN;
                   pedestrian_light <= RED;

                   IF ( crossing_request = '1' ) THEN
                      next_state <= S1;
                   ELSE
                      next_state <= S0;
                   END IF;

        WHEN S1 => -- pedestrian's road crossing request
                   -- transport traffic light is GREEN
                   -- pedestrian traffic light is RED

                   transport_light <= GREEN;
                   pedestrian_light <= RED;

                   IF ( tX(1) = '1' ) THEN
                      next_state <= S2;
                   ELSE
                      next_state <= S1;
                   END IF;

        WHEN S2 => -- timeout 1 exhausts
                   -- transport traffic light is YELLOW
                   -- pedestrian traffic light is RED

                   transport_light <= YELLOW;
                   pedestrian_light <= RED;

                   IF ( tX(2) = '1' ) THEN
                      next_state <= S3;
                   ELSE
                      next_state <= S2;
                   END IF;

        WHEN S3 => -- timeout 2 exhausts
                   -- transport traffic light is RED
                   -- pedestrian traffic light is GREEN

                   transport_light <= RED;
                   pedestrian_light <= GREEN;

                   IF ( tX(3) = '1' ) THEN
                      next_state <= S4;
                   ELSE
                      next_state <= S3;
                   END IF;

        WHEN S4 => -- timeout 3 exhausts (on transition from S3)
                   -- timeout 0 enhausts (on transition from S5)
                   -- transport traffic light is RED
                   -- pedestrian traffic light is OFF (emulating blink)

                   transport_light <= RED;
                   pedestrian_light <= OFF;

                   IF ( tX(2) = '1' ) THEN
                      next_state <= S6;
                   ELSIF ( tX(0) = '1' ) THEN
                      next_state <= S5;
                   ELSE
                      next_state <= S4;
                   END IF;

        WHEN S5 => -- timeout 0 exhausts
                   -- transport traffic light is RED
                   -- pedestrian traffic light is GREEN

                   transport_light <= RED;
                   pedestrian_light <= GREEN;

                   IF ( tX(2) = '1' ) THEN
                      next_state <= S6;
                   ELSIF ( tX(0) = '1' ) THEN
                      next_state <= S4;
                   ELSE
                      next_state <= S5;
                   END IF;

        WHEN S6 => -- timeout 2 enhausts
                   -- transport traffic light is YELLOW and RED
                   -- pedestrian traffic light is RED

                   transport_light <= YELLOW_and_RED;
                   pedestrian_light <= RED;

                   IF ( tX(1) = '1' ) THEN
                      next_state <= S0;
                   ELSE
                      next_state <= S6;
                   END IF;
     END CASE;
  END PROCESS;

  timers_control: PROCESS (clk, reset, current_state, crossing_request, tX)
  BEGIN
     IF reset = '1' THEN
        fireX <= «0000»;
     ELSIF RISING_EDGE( clk ) THEN
        CASE current_state IS

        WHEN S0 => IF ( crossing_request = '1' ) THEN
                      fireX <= «0010»;
                   ELSE
                      fireX <= «0000»;
                   END IF;

        WHEN S1 => IF ( tX(1) = '1' ) THEN
                      fireX <= «0100»;
                   ELSE
                      fireX <= «0000»;
                   END IF;

        WHEN S2 => IF ( tX(2) = '1' ) THEN
                      fireX <= «1000»;
                   ELSE
                      fireX <= «0000»;
                   END IF;

        WHEN S3 => IF ( tX(3) = '1' ) THEN
                      fireX <= «0101»;
                   ELSE
                      fireX <= «0000»;
                   END IF;

        WHEN S4 => IF ( tX(2) = '1' ) THEN
                      fireX <= «0010»;
                   ELSIF ( tX(0) = '1' ) THEN
                      fireX <= «0001»;
                   ELSE
                      fireX <= «0000»;
                   END IF;

        WHEN S5 => IF ( tX(2) = '1' ) THEN
                      fireX <= «0010»;
                   ELSIF ( tX(0) = '1' ) THEN
                      fireX <= «0001»;
                   ELSE
                      fireX <= «0000»;
                   END IF;

        WHEN S6 => fireX <= «0000»;

        END CASE;
     END IF;
  END PROCESS;

END etu;
```

Компонент timerX описывает поведение таймера, который используется для реализации временных задержек:

Timer.vhd

```
LIBRARY IEEE;
  USE IEEE.std_logic_1164.ALL;
  USE IEEE.std_logic_unsigned.ALL;

LIBRARY WORK;
  USE WORK.tl_lib.ALL; -- traffic light package

ENTITY timerX IS
  GENERIC (
            TIMEOUT_VALUE : std_logic_vector(5 DOWNTO 0)
           );
  PORT (
          Reset   : IN std_logic;
            Clk   : IN std_logic;
           Fire   : IN std_logic;
              T   : OUT std_logic
         );
END timerX;

ARCHITECTURE etu OF timerX IS

  SIGNAL state : timer_state;
  SIGNAL timer : std_logic_vector (26 DOWNTO 0);

BEGIN

t <= '1' WHEN (state = INTERRUPT) ELSE '0';

PROCESS (clk, reset, fire, timer)
BEGIN

  IF reset = '1' THEN
     timer <= (OTHERS => '0');
     state <= INIT;
  ELSIF RISING_EDGE( clk ) THEN
     CASE state IS
        WHEN INIT =>    IF fire = '1' THEN
                           state <= COUNT;
                        ELSE
                           state <= INIT;
                        END IF;

                        timer <= (OTHERS => '0');

        WHEN COUNT => IF timer = (TIMEOUT_VALUE & prescale)
THEN
                           state <= INTERRUPT;
                           timer <= (OTHERS => '0');
                        ELSE
                           state <= COUNT;
                           timer <= timer + '1';
                        END IF;

     WHEN INTERRUPT =>  state <= INIT;
                           timer <= (OTHERS => '0');

     END CASE;
  END IF;

END PROCESS;

END etu;
```

Компонент assert_never реализует монитор утверждения. Данное описание было скопировано с официального сайта фирмы **Accellera** (заголовок файла, содержащий комментарии, вырезан, так как не представляет особого интереса). Пояснение к работе этого монитора будет приведено после кода:

Assert_never.vhd

```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
USE ieee.Numeric_Std.ALL;

LIBRARY WORK;
USE WORK.ovl_assert.ALL;

ENTITY assert_never IS
  GENERIC (
-- pragma translate_off
  severity_lvl       : severity_level  := FAILURE;
-- pragma translate_on
  Options            : INTEGER      := 0
-- pragma translate_off
  ;
  Msg                : string           := «ASSERT NEVER VIOLATION»
-- pragma translate_on
  );
  PORT (clk, reset_n  : IN std_ulogic;
        test_expr     : IN boolean);
END assert_never;

ARCHITECTURE ovl OF assert_never IS
-- pragma translate_off
  SIGNAL valid : std_ulogic := '1';
  SIGNAL rst_n : std_ulogic;
-- pragma translate_on
BEGIN
-- pragma translate_off
--
  ASSERT valid = '1' REPORT msg SEVERITY severity_lvl;
--
  rst_n <= ovl_reset_n WHEN ovl_reset_n_enable ELSE
           reset_n;
--
  PROCESS
  BEGIN
     WAIT UNTIL clk'EVENT AND clk = '1';
        valid <= '1';
     IF (rst_n = '1') THEN
        IF (test_expr = TRUE) THEN
           valid <= '0';
        END IF;
     ELSE
        valid <= '1';
     END IF;
  END PROCESS;
-- pragma translate_on
END ovl; -- assert_never
```

Файл Ovl_assert.vhd также получен с www.accellera.org. Содержащийся в нем пакет декларирует все существующие мониторы библиотеки **OVL** и приведен здесь в сокращенном виде, так как для той цели, которую авторы поставили в данной статье, важен только один конкретный монитор:

Ovl_assert.vhd

```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
USE ieee.Numeric_Std.ALL;

PACKAGE ovl_assert IS

-- pragma translate_off
  SIGNAL ovl_reset_n                    : std_ulogic  := '1';
  SIGNAL ovl_reset_n_enable             : Boolean     := FALSE;
  SIGNAL ovl_end_of_simulation_signal   : std_ulogic  := '0';
-- pragma translate_on

  TYPE OVL_BOOLEAN_VECTOR IS ARRAY (INTEGER RANGE <>)
OF boolean;

  COMPONENT assert_never
  GENERIC (
-- pragma translate_off
     severity_lvl  : severity_level  := FAILURE;
-- pragma translate_on
     Options       : integer         := 0
-- pragma translate_off
     ;
     Msg           : string          := «ASSERT NEVER VIOLATION»
-- pragma translate_on
     );
  PORT (
           clk, reset_n     : IN std_ulogic;
           test_expr        : IN boolean);
  END COMPONENT;

END ovl_assert;
```

Монитор assert_never

Для монитора *assert_never* определены следующие порты:

- **clk** — сигнал тактирования, по фронту которого проверяется тестовое выражение test_expr.
- **reset_n** — сигнал сброса (активный уровень — '0').
- **test_expr** — свойство системы, записанное с помощью булевского выражение. Результат проверки выражения — TRUE или FALSE.

Монитор *assert_never* позволяет проверить свойство системы, которое никогда не должно быть оценено как истина (TRUE). Применительно к разрабатываемой системе таким свойством, например, может выступать — «сигнал светофора не должен быть зеленым одновременно для транспорта и пешехода». В том случае, если свойство нарушено, произойдет срабатывание монитора (вывод строкового параметра **msg** на экран). В зависимости от уровня важности ошибки моделирование может быть остановлено или продолжено. Вместе с сообщением в окно моделировщика выводится время срабатывания монитора.

Поддержка САПР

Одним из доказательств простоты применения средства **OVL** в сравнении с языком **PSL** является поддержка САПР. Поскольку мониторы утверждений библиотеки **OVL** построены на основе стандартных языков описания аппаратуры, то для работы с ними подойдет любая САПР, поддерживающая стандарт **IEEE-1076** (IEEE Standard VHDL Language Reference Manual). Количество средств моделирования описаний на стандартном языке **VHDL** явно превышает количество аналогичных продуктов с поддержкой **PSL** на данный момент. Однако, справедливости ради, стоит отметить, что в последнюю версию стандарта IEEE-1076 язык **PSL** уже интегрирован [5].

Один из самых распространенных — программный продукт фирмы **Mentor Graphics** — **ModelSim**. Однако, учитывая основную тематику статьи, лучше ориентироваться на аналогичную по функциональности версию с расширенными возможностями в области верификации — **QuestaSim**.

Для проведения моделирования с целью изучения принципов использования мониторов утверждений библиотеки **OVL** необходимо выполнить следующую последовательность действий:

1. Запустить программу **QuestaSim**.
2. Создать новый проект (File > New > Project) и добавить в него файлы:
   - assert_never.vhd (описание монитора утверждения *assert_never*);
   - ovl_assert.vhd (объявление мониторов утверждения и сигналов);
   - Device_Under_Test.vhd (верхний уровень иерархии при моделировании);
   - Timer.vhd (алгоритм работы таймера);
   - Traffic_Light.vhd (структура светофора);
   - Traffic_Light_Controller.vhd (автомат управления светофором);
   - Traffic_Light_Library.vhd (объявление компонентов, типов, констант и т. д.).

3. Скомпилировать проект.
4. Провести моделирование разработанной системы. Убедиться в правильности функционирования светофора.
5. Изменить в файле Traffic_Light_Controller.vhd для одного из состояний *Si* значения выходных сигналов *TRANSPORT_LIGHT* и *PEDESTRIAN_LIGHT* таким образом, чтобы они одновременно имели значение GREEN (имитация ошибки при написании кода контроллера светофора).
6. Скомпилировать проект.
7. Провести моделирование разработанной системы. В окне сообщений (transcript) основного окна **QuestaSim** наблюдать появление сообщения *"TRAFFIC ACCIDENT"*.

Искусственное внесение ошибки необходимо для визуализации работы монитора. В противном случае, при условии правильности описанной модели, монитор не сработает в ходе моделирования.

Результат выполнения моделирования HDL-программы с внесенной ошибкой приведен на рис. 3.

Положительный эффект применения монитора верификации в данном случае может показаться сомнительным, ведь состояние, в котором зеленый свет горит в оба направления, отчетливо видно на временной диаграмме. Однако простота рассматриваемого примера не должна вводить в заблуждение. Во-первых, сложность современных систем на порядок превосходит разобранный пример (по числу сигналов, сложности их поведения и т. д.). Во-вторых, анализируемое логическое условие в реальных системах выглядит сложнее, чем просто сравнение значений двух сигналов (для примера был выбран один из самых простых мониторов библиотеки **OVL**). И, наконец, в-третьих, использование мониторов позволяет в большей степени исключить человеческий фактор из процедуры проверки правильности описания — информация о свойствах системы располагается не в памяти специалиста по верификации (точнее не только в ней), а формально записана в коде устройства.

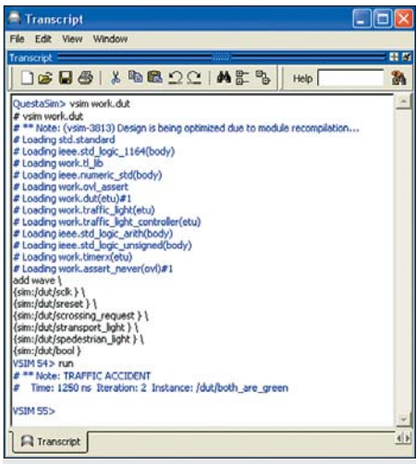

Рис. 4. Лог моделирования

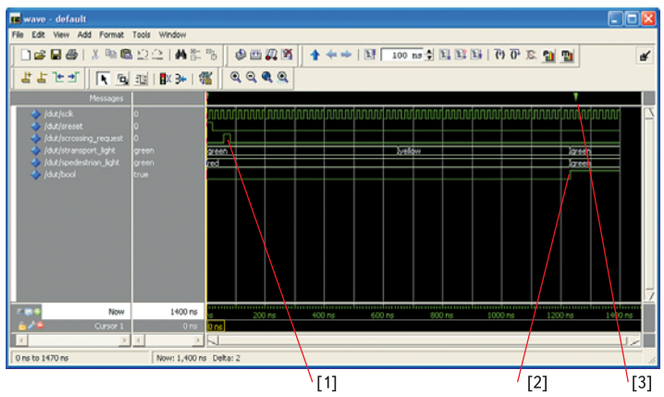

Рис. 3. Временная диаграмма: 1 — появление запроса пешехода на переход дороги; 2 — возникновение условия срабатывания монитора утверждений assert_never; 3 — срабатывание монитора на очередном такте и появление соответствующего сообщения в окне *transcript* (рис. 4)

Логическим продолжением рассмотренного примера будет обсуждение вопросов применения мониторов библиотеки **OVL** и описание их возможностей, а также перспектив развития верификационных средств. Этому будет посвящена следующая статья цикла «Проектирование в условиях временных ограничений: Верификация проектов».

Окончание следует

Литература

1. Функционально-временная верификация сложных цифровых систем // Открытые системы. 2002. № 6.
2. Assertion Monitor Reference Manual. V. 03.10.14.
3. Максфилд К. Проектирование на ПЛИС. Архитектура, средства и методы. Курс молодого бойца. М.: Додека-XXI, 2007.
4. Foster H., Krolnik A., Lacey D. Assertion-Based Design. Kluwer Academic Publisher, 2004.
5. Accellera Continues to Promote Increased Electronic Design Productivity with Revised VHDL Standard, Oct. 9, 2006. http://www.haifa.ibm.com/projects/verification/sugar/papers/accellera_vhdlVHDL_press_release_psl_inside2006.pdf