Шпаргалка для перехода от AHDL к VHDL

Иосиф Каршенбойм

lk@lmail.loniis.spb.su

Михаил Косткин

kostkin@asicdesign.ru

Вступление

Когда-то в популярной книжке по математике была напечатана такая шутка. Математик задал Физику задачу: «Имеем чайник, воду, плиту и спички. Как получить кипяток?» Физик ответил — «Поставить чайник на плиту, налить воду, зажечь плиту и вскипятить воду». «Правильно, — сказал Математик, — а вот более сложная задача — чайник с водой уже стоит на плите, что нужно сделать в этом случае?» Физик ответил, чего, мол, проще, надо зажечь огонь и готово. «Нет, — сказал Математик, — надо вылить из чайника воду, и тогда мы придем к задаче, которую уже умеем решать». То же происходит и в реальной жизни, когда есть необходимость перевести старый проект из AHDL в VHDL. Можно просто взять и сделать проект заново, а можно попытаться переписать строчку за строчкой. Для того чтобы этот процесс произошел быстрее, можно прибегнуть к данной шпаргалке.

Необходимость перехода от языка AHDL к VHDL

Сравнение проектирования на AHDL и VHDL приведено в статье Г. Варфоломеева [1]. В данной статье достаточно подробно описаны все аспекты перехода с AHDL на VHDL. Хочется добавить только следующее: в случае применения ModelSim в качестве инструмента моделирования при проведении симуляции появляется возможность читать данные из файла и эти данные выдавать на проверяемый проект в качестве входных воздействий. Очень полно и подробно о таких методах отладки и проверки проектов, написанных на VHDL, можно прочитать в учебнике [2]. Ссылки на подобные решения можно также найти, например, на сайтах **w ww. asicdesign. ru** и **w ww. actel. ru**. Результаты моделирования тоже могут быть записаны в файл. Допустим, что проект представляет собой многоканальный HDLC-контроллер. Число каналов — 30. Данные, которые необходимо пропустить через HDLC-контроллер, представляют собой, по меньшей мере, последовательность из 10 байт в каждом канале. Ядро контроллера при обработке каждого канала имеет не менее 10 состояний автомата. В итоге получаем: 30×10×10 = 3000 участков на диаграмме симуляции для проверяемого устройства. Разбираться с такой диаграммой (в случае применения AHDL и программного средства разработки MaxPlus или Quartus с его встроенным симулятором) довольно затруднительно.

В случае применения ModelSim для аналогичного проекта, выполненного на VHDL, мы получим, например, 30 файлов с выходными данными. Причем, в случае необходимости, можно записать в файл и состояния автомата на требуемых участках диаграммы или на всех участках диаграммы, а также состояния внутренних сигналов проекта и т. д. Результаты моделирования можно сравнить с математической моделью обработки данных и сделать вывод о работоспособности создаваемого проекта.

Существуют программы-перекодировщики, которые автоматически переводят файлы из AHDL в VHDL или в Verilog. Примером такой программы может служить Xport.exe. Этот перекодировщик успешно переводит файлы из одного формата в другой, но при перекодировке в выходном файле он не делает параметрических функций. То есть если в AHDL-файле проект выполнен с параметрами, которые позволяют его гибко использовать, то при перекодировке, в выходном VHDL-файле эти параметры ввести будет уже невозможно, и они будут жестко заданы в соответствии с теми значениями, которые были установлены в момент перекодировки. Именно поэтому ручная перекодировка в большинстве случаев будет более предпочтительной.

Шпаргалка

Для того чтобы было проще перейти от AHDL к VHDL, попробуем противопоставить известным и хорошо изученным в AHDL-выражениям аналогичные выражения VHDL. Тогда-то мы и придем к задаче, которую уже умеем решать. Данная статья не является учебным пособием, описывающим язык VHDL. Ее цель — облегчить переход к программированию на VHDL для тех разработчиков, которые уже имеют опыт работы на AHDL.

Рассмотрим структуру AHDL-файла. Он может содержать следующие части:

1. Title Statement (optional).
2. Include Statement (optional).
3. Constant Statement (optional).
4. Define Statement (optional).
5. Parameters Statement (optional).
6. Function Prototype Statement (optional).
7. Options Statement (optional).
8. Assert Statement (optional).
9. Subdesign Section.

10. Variable Section (optional).
    10.1. If Generate Statement (optional).
    10.2. Node Declaration (optional).
    10.3. Instance Declaration (optional).
    10.4. Register Declaration (optional).
    10.5. State Machine Declaration (optional).
    10.6. Machine Alias Declaration (optional).
    10.7. Assert Statement (optional).
11. Logic Section.
    11.1. Defaults Statement (optional).

В части файла, называемой Logic Section, находятся следующие выражения, которые могут следовать в файле в любом порядке и повторяться произвольное число раз:

1. Boolean Equation
2. Case Statement
3. For Generate Statement
4. If Generate Statement
5. If Then Statement
6. In-Line Logic Function Reference
7. Truth Table Statement
8. Assert Statement

Далее попробуем заменять каждую часть AHDL-файла на соответствующее выражение для VHDL-файла.

1. Пропустим Title Statement, так как аналога в VHDL-файле нет и вместо этого воспользуемся символом комментария «--» и напишем:

```
-- My project… и т. д.
```

2. Для VHDL-файла заголовочными являются файлы, описывающие, с какими библиотеками придется работать компилятору, а не описания библиотечных и «самодельных» компонентов проекта. Типичный вид части VHDL-файла, описывающего библиотеки:

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all ;
use ieee.std_logic_unsigned.all ;
use work.uart_signals.all ;
USE std.standard.ALL;
USE std.textio.All;
```

Поэтому часть AHDL-файла, пункт 2, Include Statement, в которой дано описание библиотечных и «самодельных» компонентов для данного проекта, переносится в конструкцию PACKAGE, где компоненты описываются, например, так:

```
COMPONENT DFFE
        PORT (d    : IN STD_LOGIC;
              clk  : IN STD_LOGIC;
              clrn : IN STD_LOGIC := '1';
              prn  : IN STD_LOGIC := '1';
              ena  : IN STD_LOGIC;
              q    : OUT STD_LOGIC );
        END COMPONENT;
```

Собственно раздел PACKAGE может быть размещен в одном файле с entities.

То же самое относится и к части 6 AHDL-файла — Function Prototype Statement. Вместо декларирования прототипа функции, принятого в AHDL, в VHDL объявляется компонент так, как указано выше.

3. Выделим часть 3 AHDL-файла, в которой описываются константы, применяемые для данного файла, — Constant Statement. В VHDL константы желательно размещать в конструкции PACKAGE. Выражение из AHDL-файла, например:

```
CONSTANT DATA_WIDTH = Data_width;
```

заменяем на выражение для VHDL, при этом необходимо учесть тип данных константы:

```
constant constant_name : type := value;
```

4. Выделим часть 4 AHDL-файла, в которой описываются параметры, применяемые для данного файла, — Parameters Statement. В VHDL настраиваемые параметры Entity имеют название GENERIC и инициализируются при использовании компонента. Выражение из AHDL-файла, например:

```
PARAMETERS ( Data_width = 32, Data_len = 8 );
```

заменяем на выражение для VHDL, при этом необходимо учесть тип данных параметров. Для VHDL параметры могут относиться ко всему файлу, тогда они связаны с ENTITY и выглядят, например, так:

```
ENTITY Adder IS
  GENERIC -— Interface constants, can be modified by configuration
    (MaxValue_g  : integer := 255);
  PORT
    (A    : in  Integer range 0 to MaxValue_g;
     B    : in  Integer range 0 to MaxValue_g;
     Y    : out Integer range 0 to MaxValue_g);
  END ENTITY Adder;
```

5. Выделим часть 8 AHDL-файла, Assert Statement, в которой описываются сообщения, выдаваемые при компиляции файла. Выражение из AHDL-файла, например:

```
ASSERT
        REPORT «  CNT_WIDTH равен — %, DATAB_WIDTH
равно — %» CNT_WIDTH, DATAB_WIDTH
        SEVERITY INFO;
```

заменяем на выражение для VHDL, при этом необходимо учесть тип данных, которые проверяются в выражении ASSERT, например, вот так:

```
ASSERT A_s = '1'
  REPORT «A_s /= '1', it is « & Bit2Strg_c(A_s)
SEVERITY NOTE;
```

При этом параметр после SEVERITY определяет уровень сообщения (NOTE, ERROR или другие), которое генерируется при выполнении команды ASSERT, что позволяет, настроив симулятор необходимым образом, либо выдать сообщение, указывающее пользователю, что при заданном условии команда ASSERT выполнилась, либо остановить симуляцию, если выполнение команды ASSERT происходит при возникновении ошибки.

6. Выделим часть 9 AHDL-файла, Subdesign Section, в которой описывается название проекта, например:

```
SUBDESIGN __design_name
 (
        clk, …);
```

и заменим эту часть на соответствующее выражение в VHDL:

```
ENTITY __entity_name IS
  GENERIC(…                              );
  PORT( __input_name, __input_name  : IN STD_LOGIC;
        __input_vector_name         : IN STD_LOGIC_VECTOR
                                      (__high  downto  __low);
        __bidir_name, __bidir_name    : INOUT  STD_LOGIC;
        __output_name, __output_name  :  OUT   STD_LOGIC);
END __entity_name;
```

И далее, добавим следующую часть кода, описывающего архитектуру:

```
ARCHITECTURE a OF __entity_name IS
        SIGNAL __signal_name : STD_LOGIC;
        SIGNAL __signal_name : STD_LOGIC;
BEGIN
        -- Process Statement
        -- Concurrent Procedure Call
        -- Concurrent Signal Assignment
        -- Conditional Signal Assignment
        -- Selected Signal Assignment
        -- Component Instantiation Statement
        -- Generate Statement
END a;
```

8. Выделим часть 10 AHDL-файла, Variable Section, в которой описываются переменные для данного проекта, например:

```
VARIABLE
    reset, Inp_shift_rg8    : node;
    phase_cnt_tx            : lpm_counter with ( lpm_width = 4,
                              lpm_direction = «UP» ) ;
    tx_process              : machine with states ( idle_tx, trm_shift
) ;
```

и т. д.

Поступаем следующим образом:

А. То, что в AHDL-файле называлось NODE, теперь назовем SIGNAL и поместим либо в ARCHITECTURE (см. п. 7), либо вынесем в отдельный файл PACKAGE. Это позволит уменьшить объем основного файла проекта.

Б. Библиотечные и «самодельные» компоненты в VHDL заменяются декларированием соответствующего компонента.

Пример файла PACKAGE.

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all ;
use ieee.std_logic_unsigned.all ;

PACKAGE uart_signals IS

COMPONENT__component_name
        PORT(__input_name, __input_name      : IN STD_LOGIC;
             __bidir_name, __bidir_name      : INOUT
                                               STD_LOGIC;
             __output_name, __output_name    : OUT
                                               STD_LOGIC);
END COMPONENT;
        ……
SIGNAL
        phase_cnt_tx_sclr,
        …
        rx_process_idle   : STD_LOGIC;
END uart_signals;
```

Если компонент имеет параметры, тогда он декларируется так:

```
COMPONENT __component_name
        GENERIC(__parameter_name  :  string  :=  __default_value;
                __parameter_name  : integer := __default_value);
        PORT( __input_name: IN  STD_LOGIC; …. );
END COMPONENT;
```

В. Несколько сложнее обстоит дело с декларацией сигналов статических автоматов.

Вот часть кода, описывающего работу статического автомата:

```
type State_Typ is (RST, RedOn, YelOn, GrnOn);
 signal State_s  : State_Typ;
begin

 Explicit_Lbl : process
 begin
  wait until Clk'event and Clk = '1';
  if Reset = '1' and State_s = GrnOn then
   Red   <= '0';
   Yellow <= '0';
   Green  <= '0';
   State_s  <= RST;

  elsif Reset = '0' and
    (State_s = RST or State_s = GrnOn) then
   Red   <= '1';
   Yellow <= '0';
   Green  <= '0';
   State_s  <= RedOn;

  elsif State_s = …..

  end if;
 end process Explicit_Lbl;
end Explicit_a;
```

Как видно из приведенного примера, необходимо указать тип данных для сигнала State_s и перечислить те состояния, которые он принимает.

9. В части AHDL-файла, называемой Logic Section, могут находиться выражения If Generate Statement. Данная часть AHDL-файла аналогична следующему выражению в VHDL:

```
__generate_label:
IF __expression GENERATE
   __statement;
END GENERATE;
```

10. В части AHDL-файла, называемой Logic Section, могут находиться булевы выражения типа:

```
__node_name = __node_name & __node_name;
__node_name = __node_name # __node_name;
__node_name = __node_name $ __node_name;
```

из AHDL заменяются в VHDL на следующие выражения для сигналов:

```
__sygnal_name <= __ sygnal_name AND __sygnal_name;
__sygnal_name <= __sygnal_name OR   __sygnal_name;
__sygnal_name <= __sygnal_name XOR __sygnal_name;
```

а для переменных аналогичные выражения будут выглядеть так:

```
__variable_name := __variable_name AND __variable_name;
__variable_name := __variable_name OR   __variable_name;
__variable_name := __variable_name XOR __variable_name;
```

11. В части AHDL-файла, называемой Logic Section, могут находиться выражения «IF — THEN». Выражение из AHDL «IF — THEN» выглядит следующим образом:

```
IF __expression_1 THEN
            __statement_1;
ELSIF __expression_2 THEN
            __statement_2;
ELSE
            __statement_3;
END IF;
```

В AHDL данное выражение заставляет выполняться выражения __statement_1, __statement_2 или __statement_3 только в зависимости от выполнения или невыполнения условий __expression_1 или __expression_2.

В VHDL выражение «IF — THEN» имеет несколько иное значение, чем в AHDL, поэтому аналогом данного выражения в VHDL будет являться выражение, называемое Concurrent_signal:

```
__label:
__signal <= __expression WHEN __boolean_expression ELSE
            __expression WHEN __boolean_expression ELSE
            __expression;
```

Если речь идет об обработке сигналов «внутри» выражения PROCESS:

```
__process_label:
PROCESS
            VARIABLE __variable_name : STD_LOGIC;
BEGIN
            WAIT UNTIL __clk_signal = '1';
            -- Signal Assignment Statement
            -- Variable Assignment Statement
            -- Procedure Call Statement
            -- If Statement
            -- Case Statement
            -- Loop Statement
END PROCESS __process_label;
```

то тогда возможно применение выражения «IF __expression THEN», такого же вида, что и в AHDL. Отличия будут только в выражении __expression. В AHDL все переменные NODE имеют один и тот же тип данных, который можно сопоставить с типом данных SYGNAL в VHDL. Назовем сигнал, который будет проверяться в выражении «IF __expression THEN», именем my_sygnal и проверим его на соответствие высокому уровню. Выражение в AHDL будет выглядеть так:

```
IF my_sygnal = = 1 THEN
  ...
```

Поскольку в AHDL имеется только один тип данных для сигналов, и при проверке сигнала ему автоматически приписывается значение True в том случае, если данный сигнал имеет высокий уровень, выражение может выглядеть и так:

```
IF my_sygnal  THEN
  ...
```

Для VHDL имеется несколько типов данных, поэтому для данных типа SYGNAL выражение будет выглядеть так:

```
IF my_sygnal  = '1' THEN
…
```

а для данных типа BOOLEAN выражение будет выглядеть так:

```
IF my_bool  THEN
…
```

12. В части AHDL-файла, называемой Logic Section, могут находиться выражения CASE. Выражение CASE в AHDL:

```
CASE __expression IS
            WHEN __constant_value =>
              __statement;
              __statement;
            WHEN OTHERS =>
              __statement;
              __statement;
END CASE;
```

аналогично выражению «IF — THEN», для выражения CASE могут быть применены те же самые правила. Если это выражение не входит в какое-либо выражение PROCESS (см. объявление PROCESS выше), то такое выражение AHDL заменяется на выражение VHDL, называемое Selected signal:

```
__label:
WITH __expression SELECT
      __signal <= __expression WHEN __constant_value,
                  __expression WHEN __constant_value,
                  __expression WHEN __constant_value,
```

Если речь идет об обработке сигналов «внутри» выражения PROCESS, то тогда аналогично изложенному выше применяется функция «CASE __expression IS» такого же вида, что и в AHDL.

13. В части AHDL-файла, называемой Logic Section, могут находиться выражения, называемые таблицами истинности. Выражение из AHDL, соответствующее таблице истинности, выглядит следующим образом:

```
TABLE
      __node_name, __node_name    => __node_name,
                                          __node_name;
      __input_value, __input_value  => __output_value,
                                          __output _value;
      __input_value, __input_value  => __output_value,
                                          __output _value;
END TABLE;
```

Данному выражению нет прямой аналогии в VHDL, поэтому необходимо воспользоваться выражениями типа Concurrent_signal, «IF — THEN» или CASE, в зависимости от того, входит ли таблица истинности в выражение PROCESS или нет.

Заключение

Языки AHDL и VHDL относятся к одной группе языков описания аппаратуры. Именно поэтому большую часть выражений языка AHDL можно перевести непосредственно в выражения языка VHDL. Однако полного соответствия между выражениями в этих двух языках, к сожалению, нет. Поэтому часть выражений языка AHDL, которым нет прямой аналогии в VHDL, придется перерабатывать в аналогичные по смыслу конструкции языка VHDL. Поскольку VHDL по отношению к AHDL является языком более высокого уровня, то необходимо также учитывать и дополнительные факторы, такие, как, например, разные типы данных, отличия в описании статических автоматов и др. Однако достаточно хотя бы один раз преодолеть все эти трудности, чтобы потом сказать: «от AHDL до VHDL всего один шаг».

Литература

1. Глеб Варфоломеев. Проектирование в системе Max+Plus II: VHDL против AHDL // Компоненты и Технологии. 2002. № 7.
2. Ben Cohen. VHDL Coding Styles and Methodologies. Second Edition. Kluwer academic publishers. Boston-Dordrecht-London. 1999.