ИНФОРМАТИКА

2006 **январь-март** **№ 1**

ЛОГИЧЕСКОЕ ПРОЕКТИРОВАНИЕ

УДК 519.7

П.Н. Бибило

СРАВНЕНИЕ СХЕМНЫХ РЕАЛИЗАЦИЙ VHDL-МОДЕЛЕЙ, ИСПОЛЬЗУЮЩИХ ЧАСТИЧНУЮ ОПРЕДЕЛЕННОСТЬ БУЛЕВЫХ ФУНКЦИЙ

Предлагаются модели не полностью определенных (частичных) булевых функций и систем таких функций. Показывается, что использование моделей частичных функций позволяет получать в системе синтеза схем LeonardoSpectrum более простые логические схемы по сравнению с VHDL-моделями, базирующимися на системах полностью определенных функций.

Введение

В настоящее время широкое распространение получил язык VHDL, который является входным языком современных САПР цифровых систем, реализуемых на СБИС. На этом языке разработано большое число моделей поведения цифровых систем различного класса и уровня сложности – функциональных моделей микропроцессоров, конечных автоматов, булевых функций и т. д.

Описания на языке VHDL булевых функций и их систем разработаны для класса полностью определенных функций [1]. Использование логических операторов языка позволяет легко записать алгебраические логические выражения, соответствующие поведению логического элемента. Скобочные алгебраические выражения дают возможность в стиле data flow описывать поведение комбинационных логических схем. В литературе было показано, что частичные булевы функции могут описывать поведение некоторых логических схем, так как в формальных описаниях фиксированные наборы входных переменных не могут появляться на входных полюсах таких схем. Эта неопределенность (свобода доопределения) служит источником оптимизации при синтезе, нахождение соответствующего доопределения позволяет получить более простые схемы. Доопределения используются при решении задачи совместной минимизации систем частичных функций в классе дизъюнктивных нормальных форм (ДНФ) [2, 3 и др.].

В данной работе предлагаются VHDL-модели частичных булевых функций и систем таких функций, показывается, что использование моделей частичных функций позволяет получать менее сложные логические схемы в промышленном синтезаторе LeonardoSpectrum [4]. Предполагается, что читатели знакомы с основными элементами языка VHDL.

1. VHDL-модель одной частичной функции

В языке VHDL введен девятизначный логический алфавит для моделирования и синтеза логических схем на базе перечислимого типа `std_logic`. Массив (вектор) значений типа `std_logic` обозначается как `std_logic_vector`. На базе типов `std_logic`, `std_logic_vector` предлагается строить VHDL-модели частичных функций. Дело в том, что одно из девяти значений данного типа называется don't care, обозначается как `'-'` и предназначается для оптимизации при логическом синтезе. Однако в литературе не показано, каким именно образом значение don't care используется при составлении VHDL-моделей и как влияет на результаты синтеза по таким моделям.

Рассмотрим частичную булеву функцию, заданную в табл. 1. Неопределенные значения функции $F(x1, x2, x3)$ показаны в табл. 1 черточкой. VHDL-модель частичной функции дана в листинге 1.

Таблица 1

$x1$	$x2$	$x3$	F
0	0	0	-
0	0	1	-
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	-
1	1	1	-

В тексте VHDL-программы используется тип `std_logic`, определенный в пакете `std_logic_1164`. Жирным шрифтом выделен оператор `case`, который и описывает таблицу значений функции. Оператор `case` – последовательный и может появляться в подпрограммах либо процессах, этим объясняется появление оператора `process`. Правая часть выражения $y := x1\&x2\&x3$; представляет собой конкатенацию битовых сигналов, оператор & в языке VHDL – это оператор конкатенации, логический оператор И в языке обозначается через `and`.

Листинг 1. **VHDL-модель частичной функции**

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;
ENTITY example1 IS
 PORT (x1, x2, x3 : IN std_logic;
       F : OUT std_logic);
END;
ARCHITECTURE BEHAVIOR OF example1 IS
BEGIN
 PROCESS (x1, x2, x3)
variable y : std_logic_vector (0 to 2);
variable f1 : std_logic;
BEGIN
y:= x1&x2&x3;
case y is
when   "010"  => f1:= '0';
when   "100"  => f1:= '0';
when   "011"  => f1:= '1';
when   "101"  => f1:= '1';
when  others  => f1:= '-';
end case;
F<=f1;
end process;
END BEHAVIOR;
```

Проведя синтез по VHDL-описанию, представленному в листинге 1, в промышленном синтезаторе LeonardoSpectrum, получим схему, не имеющую логических элементов, так как она вырождается до $F = x3$. В процессе преобразования алгоритмического описания в логические уравнения синтезатор LeonardoSpectrum использовал неопределенные значения `'-'` при минимизации и упростил логическое уравнение до вида $F = x3$. Легко проверить, что несущественным подмножеством переменных частичной функции, заданной в табл. 1, является подмножество $\{x1, x2\}$.

2. VHDL-модель системы частичных функций

Приведем более содержательный пример схемы circ_chast (рис. 1), описываемой в виде композиции систем частичных булевых функций.

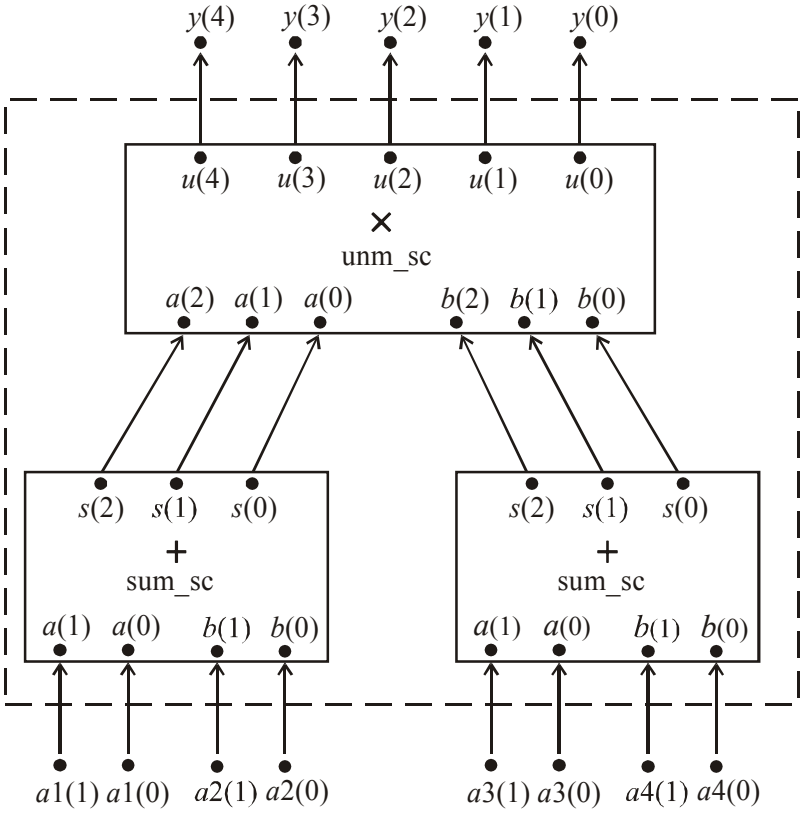

Рис. 1. Схема circ_chast

Подсхема sum_sc (листинг 2) представляет собой сумматор для сложения пары чисел, выбираемых из множества {0, 1, 2}, т. е. «неполный» сумматор. Данный сумматор описывается системой частичных функций, заданной в табл. 2.

***Листинг 2*. VHDL-модель «неполного» сумматора `sum_sc`**

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;
ENTITY sum_sc IS
 PORT (a, b : IN std_logic_vector (1 downto 0);
       s : OUT std_logic_vector (2 downto 0) );
END;
ARCHITECTURE BEHAVIOR OF sum_sc IS
BEGIN
 PROCESS (a, b)
variable y : std_logic_vector (3 downto 0);
variable s_pr :  std_logic_vector (2 downto 0);
BEGIN
y:= a&b;
case y is
when "0000"  => s_pr:= "000";
when "0001"  => s_pr:= "001";
when "0010"  => s_pr:= "010";
when "0100"  => s_pr:= "001";
```

```
when "0101"  => s_pr:= "010";
when "0110"  => s_pr:= "011";
when "1000"  => s_pr:= "010";
when "1001"  => s_pr:= "011";
when "1010"  => s_pr:= "100";
when others  => s_pr:= "---";
end case;
s<=s_pr;
end process;
END BEHAVIOR;
```

Если одно из слагаемых представляет собой число 3, то значения выходов сумматора не определены. Для кодирования каждого из складываемых чисел нужны две булевы переменные. Первое число задается в виде $a = (a(1), a(0))$, второе – в виде $b = (b(1), b(0))$, старшие разряды – $a(1)$, $b(1)$, старший разряд суммы – $s(2)$, для представления суммы $s = (s(2), s(1), s(0))$ достаточно трех булевых переменных.

Таблица 2

$a(1)$	$a(0)$	$b(1)$	$b(0)$	$s(2)$	$s(1)$	$s(0)$
0	0	0	0	0	0	0
0	0	0	1	0	0	1
0	0	1	0	0	1	0
0	0	1	1	–	–	–
0	1	0	0	0	0	1
0	1	0	1	0	1	0
0	1	1	0	0	1	1
0	1	1	1	–	–	–
1	0	0	0	0	1	0
1	0	0	1	0	1	1
1	0	1	0	1	0	0
1	0	1	1	–	–	–
1	1	0	0	–	–	–
1	1	0	1	–	–	–
1	1	1	0	–	–	–
1	1	1	1	–	–	–

Подсхема `umn_sc` представляет собой умножитель чисел, представленных в двоичном коде и выбираемых из множества {0, 1, 2, 3, 4}. Именно такие числа и могут появляться на входных портах данной подсхемы. Данный «неполный» умножитель описывается системой частичных функций, VHDL-модель умножителя дана в листинге 3. Для кодирования каждого из перемножаемых чисел нужны три булевы переменные. Первое число задается в виде $a = (a(2), a(1), a0))$, второе – в виде $b = (b(2), b(1), b(0))$, старшие разряды перемножаемых чисел – $a(2)$, $b(2)$, старший разряд произведения – $u(4)$. Для представления произведения $u = (u(4), u(3), u(2), u(1), u(0))$ достаточно пяти булевых переменных.

***Листинг 3*. VHDL-модель «неполного» умножителя `umn_sc`**

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;
ENTITY umn_sc IS
 PORT (a, b : IN std_logic_vector ( 2 downto 0);
         u : out std_logic_vector  ( 4 downto 0) );
END;
ARCHITECTURE BEHAVIOR OF umn_sc IS
BEGIN
 PROCESS (a, b)
```

```
variable y : std_logic_vector (5 downto 0);
variable u_pr :  std_logic_vector (4 downto 0);
BEGIN
y:= a&b;
case y is
when "000000"  => u_pr:= "00000";
when "000001"  => u_pr:= "00000";
when "000010"  => u_pr:= "00000";
when "000011"  => u_pr:= "00000";
when "000100"  => u_pr:= "00000";

when "001000"  => u_pr:= "00000";
when "001001"  => u_pr:= "00001";
when "001010"  => u_pr:= "00010";
when "001011"  => u_pr:= "00011";
when "001100"  => u_pr:= "00100";

when "010000"  => u_pr:= "00000";
when "010001"  => u_pr:= "00010";
when "010010"  => u_pr:= "00100";
when "010011"  => u_pr:= "00110";
when "010100"  => u_pr:= "01000";

when "011000"  => u_pr:= "00000";
when "011001"  => u_pr:= "00011";
when "011010"  => u_pr:= "00110";
when "011011"  => u_pr:= "01001";
when "011100"  => u_pr:= "10100";

when "100000"  => u_pr:= "00000";
when "100001"  => u_pr:= "00100";

when "100010"  => u_pr:= "01000";
when "100011"  => u_pr:= "10100";
when "100100"  => u_pr:= "10000";

when others  =>   u_pr:= "-----";
end case;
u<=u_pr;
end process;
END BEHAVIOR;
```

Структурное описание схемы, изображенной на рис. 1, дано в листинге 4.

***Листинг 4*. VHDL-модель `circ_chast`**

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

entity circ_chast is
  generic ( byte : natural := 2);
  port (
    a1, a2, a3, a4 : in  std_logic_vector(byte-1 downto 0);
      y : out std_logic_vector ( 2*byte  downto 0)  );
end circ_chast;
architecture beh of circ_chast is
component sum_sc
 PORT (a, b : IN std_logic_vector (1 downto 0);
       s : OUT std_logic_vector (2 downto 0) );
end component;
```

```
component umn_sc
 PORT (a, b : IN std_logic_vector (2 downto 0);
        u : out std_logic_vector(4 downto 0) );
 END component;
signal s1, s2 : std_logic_vector(byte  downto 0);
begin
p1: sum_sc port map ( a => a1, b => a2, s => s1);
p2: sum_sc port map ( a => a3, b => a4, s => s2);
p3: umn_sc port map ( a => s1, b => s2, u => y );
end beh;
```

3. Сравнение модели системы частичных функций с другими моделями

Использование функции `to_integer` перевода типа `std_logic_vector` в целочисленный тип `integer` (и функции обратного перевода) пакета `numeric_std` позволяет значительно упростить описание схемы. Соответствующее описание `circ_f` приводится в листинге 5. Использование настраиваемого параметра `byte` показывает путь параметризации этой схемы по разрядности.

***Листинг 5*. VHDL-модель `circ_f`**

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;
entity circ_f is
  generic ( byte : natural := 2);
  port (a1, a2, a3, a4 : in  std_logic_vector(byte-1 downto 0);
        y : out std_logic_vector(2*byte  downto 0));
end circ_f;
architecture beh of circ_f is
signal s1_int , s2_int, s3_int, s4_int : integer range 0 to 3;
signal y_int : integer range 0 to 49;
begin
s1_int <= to_integer (unsigned (a1));
s2_int <= to_integer (unsigned (a2));
s3_int <= to_integer (unsigned (a3));
s4_int <= to_integer (unsigned (a4));
y_int <= (s1_int +s2_int) * (s3_int + s4_int);
y <= std_logic_vector(to_unsigned(y_int, 5));
end beh;
```

Компактная VHDL-модель с использованием типов `std_logic_vector` портов (листинг 6) приводит к полным сумматорам и полному умножителю и требует шесть выходных полюсов.

***Листинг 6*. VHDL-модель `circ_full`**

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;
entity circ_full is
  generic ( byte : natural := 2);
  port (a1, a2, a3, a4 : in  std_logic_vector(byte-1 downto 0);
       y : out std_logic_vector (2*byte+1  downto 0) );
end circ_full;
architecture beh of circ_full is
begin
y <= std_logic_vector( (unsigned('0'& a1)+unsigned (a2))
                     *(unsigned('0'& a3)+unsigned(a4)) );
end beh;
```

Если же применять целочисленные типы входных портов, то описание схемы может быть весьма компактным. Назовем данную модель `circ_integ`. В этом случае надо проследить за представлением входных и выходных портов битовыми переменными в синтезированной схеме. Кодирование входных и выходных портов в синтезированных схемах рассмотрено в работе [1]. Модель описания поведения всей схемы в виде одной системы частичных функций назовем `chast`.

Сравнение схемных реализаций VHDL-моделей `circ_chast`, `circ_f`, `circ_full`, `circ_integ`, `chast` приведено в табл. 3. Модель `chast_DNF` будет объяснена в разд. 4.

Таблица 3

VHDL-модель	*S*	*L*	τ
`circ_chast`	124	32	5,57
`circ_f`	154	36	7,48
`circ_full`	160	37	7,61
`circ_integ`	154	36	7,48
`chast`	171	50	4,48
`chast_DNF`	**119**	**32**	**3,52**

Схемы синтезировались в библиотеке БМК, описанной в работе [1]. В табл. 3 используются следующие обозначения:

S – суммарное число элементарных ячеек, составляющих схему (суммарная площадь всех элементов схемы);

L – число элементов схемы;

τ – задержка схемы (наносекунды).

Описание схемы в виде одной системы частичных функций (модель `chast`) позволяет получить схему с меньшей задержкой, однако сложность схемы увеличивается – она распараллеливается. Анализируя табл. 3, можно сделать вывод об эффективности синтеза от исходной суперпозиции систем частичных функций (модель `circ_chast`). Можно заметить также, что результаты синтеза по VHDL-моделям `circ_f`, `circ_integ` не различаются. Эксперимент подтвердил, что использование моделей, приводящих к полностью определенным функциям, дает более сложные схемы по сравнению с моделями частичных функций. Однако лучший результат был получен в результате синтеза по VHDL-модели `chast_DNF`.

4. Синтез схем с использованием системы Custom Logic и синтезатора LeonardoSpectrum

Актуальным является вопрос о поиске полностью определенной функции, которая реализует исходную частичную функцию. В первом примере (листинг 1) получение такой функции возможно уже после выполнения процедуры чтения проекта, во время которой осуществляется переход к RTL-описанию. В более сложных случаях требуется специальная методика получения этих функций, которая состоит из двух этапов.

Этап 1. После синтеза схемы выполнить команду `unmap` в командном окне синтезатора LeonardoSpectrum, сохранить результат (логические уравнения, описывающие синтезированную схему) в виде VHDL-кода.

Этап 2. Конвертировать VHDL-код в описание на языке SF в системе проектирования Custom Logic [5]. С помощью проектных операций системы Custom Logic получить матричное двухуровневое (И-ИЛИ) представление (систему ДНФ) полностью определенной функции или системы функций. Взаимодействие систем LeonardoSpectrum и Custom Logic показано на рис. 2.

Если же выполнить еще и этап 3, то можно получить в LeonardoSpectrum логическую схему по двухуровневому представлению системы функций.

Этап 3. Конвертировать SF-описание в VHDL-код, который можно подать на вход LeonardoSpectrum, и провести повторный синтез от двухуровневого представления.

Применение данной методики к подсхеме `sum_sc` позволяет получить систему полностью определенных функций, заданную в табл. 4. Жирным шрифтом выделены доопределенные значения.

Таблица 4

a(1)	*a*(0)	*b*(1)	*b*(0)	*s*(2)	*s*(1)	*s*(0)
0	0	0	0	0	0	0
0	0	0	1	0	0	1
0	0	1	0	0	1	0
0	0	1	1	**0**	**1**	**0**
0	1	0	0	0	0	1
0	1	0	1	0	1	0
0	1	1	0	0	1	1
0	1	1	1	**0**	**1**	**0**
1	0	0	0	0	1	0
1	0	0	1	0	1	1
1	0	1	0	1	0	0
1	0	1	1	**0**	**1**	**0**
1	1	0	0	**0**	**1**	**0**
1	1	0	1	**0**	**1**	**0**
1	1	1	0	**0**	**1**	**0**
1	1	1	1	**0**	**1**	**0**

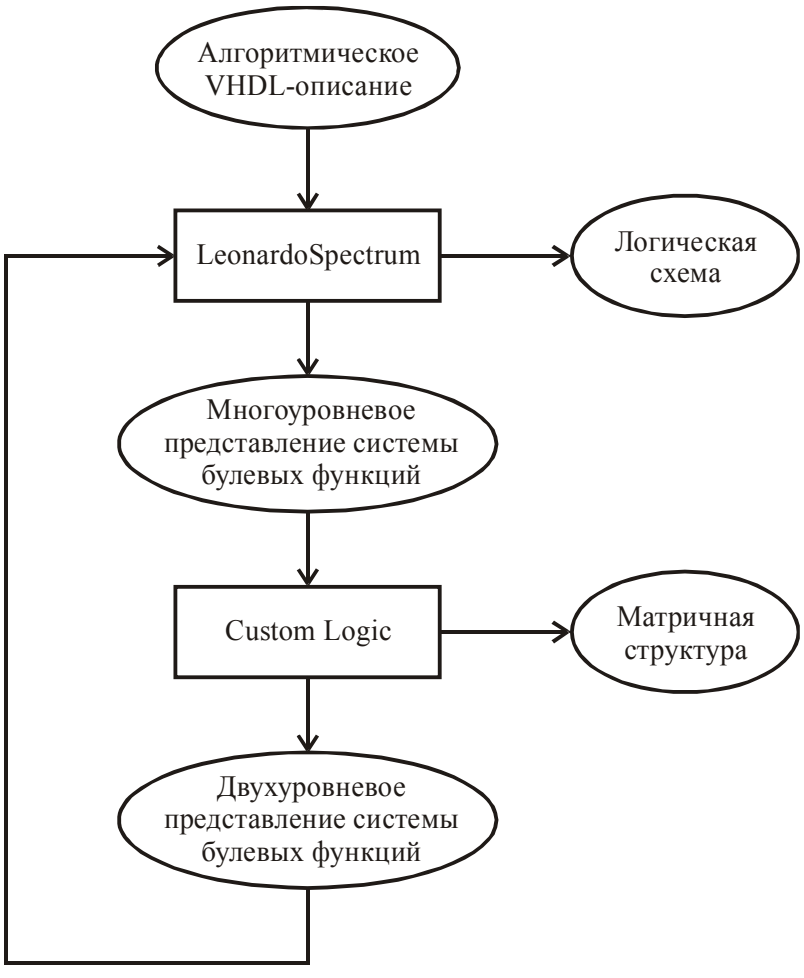

Рис. 2. Повторный синтез схем с использованием системы Custom Logic

Проведенные эксперименты по схемной реализации частичных функций в синтезаторе LeonardoSpectrum позволяют с большой долей уверенности полагать, что доопределение каждой из реализуемых функций происходит независимо. Был проделан эксперимент по схемной реализации системы частичных функций [2, с. 128]. Схемная реализация в библиотеке БМК исходной системы частичных функций привела к схеме с параметрами S=71, L=22, τ=8,40, схемная реализация системы полностью определенных функций [2, с. 130], полученной после совместной минимизации в классе ДНФ исходной системы частичных функций, – к схеме с параметрами S=39, L=13, τ=6,05. Данный эксперимент позволяет сделать вывод о том, что если схемной реализации подвергается система частичных функций, то во многих случаях целесообразно вне промышленных синтезаторов провести их совместную минимизацию в классе частичных функций, так как такой вид оптимизации отсутствует в LeonardoSpectrum.

Для примера системы частичных функций, реализующих схему (рис.1), предварительная совместная минимизация частичных функций в классе ДНФ позволяет уменьшить сложность схемы. Была построена система частичных функций, описывающая поведение схемы. Затем проведена совместная минимизация этой системы с помощью программы [6] и получена система ДНФ полностью определенных функций. VHDL-модель минимизированной системы была названа `chast_DNF`. По этой модели в LeonardoSpectrum построена логическая схема с параметрами S=119, L=32, τ=3,52 (см. табл. 3).

Заключение

В статье показано, что использование VHDL-моделей систем частичных функций может давать лучшие результаты по сравнению с другими моделями, приводящими к полностью определенным булевым функциям. Так как в промышленных синтезаторах осуществляется переход к полностью определенным функциям на этапе высокоуровневого синтеза, то во многих случаях целесообразно предварительное использование программ совместной минимизации систем частичных функций. Совместное использование систем синтеза Custom Logic и LeonardoSpectrum может привести в ряде случаев к более эффективным схемным реализациям.

Список литературы

1. Бибило П.Н. Синтез логических схем с использованием языка VHDL. – М.: СОЛОН-Р, 2002. – 384 с.

2. Закревский А.Д. Логический синтез каскадных схем. – М.: Наука, 1981. – 416 с.

3. Закревский А.Д., Поттосин Ю.В., Черемисинова Л.Д. Основы логического проектирования. В 2 кн. Кн. 2. Оптимизация в булевом пространстве. – Мн.: ОИПИ НАН Беларуси, 2004. – 240 с.

4. Бибило П.Н. Системы проектирования интегральных схем на основе языка VHDL. StateCAD, ModelSim, LeonardoSpectrum. - М.: СОЛОН-Пресс, 2005. – 384 с.

5. Система «Custom Logic» автоматизированного проектирования управляющей логики заказных цифровых СБИС / П.Н. Бибило, И.В. Василькова, С.Н. Кардаш и др. // Микроэлектроника. – 2003. – Т. 32. – № 5. – С. 379–398.

6. Торопов Н.Р. Минимизация систем булевых функций в классе ДНФ // Логическое проектирование. – Мн.: Ин-т техн. кибернетики НАН Беларуси, 1999. – Вып. 4. – С. 4–19.

Поступила 22.11.05

Объединенный институт проблем информатики НАН Беларуси,
Минск, Сурганова, 6
e-mail: bibilo@newman.bas-net.by

P.N. Bibilo

COMPARISON OF CIRCUIT IMPLEMENTATIONS OF VHDL MODELS OF INCOMPLETELY SPECIFIED BOOLEAN FUNCTIONS

In the paper VHDL models of incompletely specified Boolean functions and systems of such functions are suggested. Using the models of incompletely specified functions allows to obtain less complicated logical circuits in LeonardoSpectrum synthesizer as compared with the models resulting in completely specified functions. Complementary using Custom Logic synthesis system allows to organize one more design route by transition to two-level representations of implemented functions.