УДК 519.713:681.3

П.Н. Бибило, С.Н. Кардаш, Н.А. Кириенко, Д.А. Кочанов, П.В. Леончик, Д.Я. Новиков, В.И. Романов, Д.И. Черемисинов

Программный комплекс *MICEL* высокоуровневого и логического синтеза параллельных алгоритмов логического управления

Описан комплекс программных средств, позволяющий вести высокоуровневое и логическое проектирование устройств управления, поведение которых задано на языках описания аппаратуры *VHDL* и *PRALU*.

The MICEL advanced programming tools are described. These tools can perform a high-level and logical design of control devices, behavior of which is specified in VHDL and PRALU languages of the hardware description.

Описано комплекс програмних засобів, який дозволяє здійснювати високорівневе та логічне проектування пристроїв керування, поведінку яких задано на мовах описання апаратури *VHDL* та *PRALU*.

Введение. Широкое внедрение систем автоматизации проектирования (САПР) цифровых управляющих и вычислительных систем позволяет выбрать наиболее эффективные маршруты проектирования с использованием как зарубежных, так и отечественных САПР. Схемная реализация параллельных алгоритмов логического управления – это одна из трудных задач проектирования. Для этого класса алгоритмов могут быть использованы современные языки *VHDL*, *Verilog* описания цифровой аппаратуры, так как эти языки дают возможность описания параллельных процессов. Схемная реализация алгоритмов, представленных на этих языках, в современных синтезаторах *Leonardo-Spectrum* (далее – *Leonardo*), *XST*, *Synplify* осуществляется по компилятивному (локальному) способу – каждый оператор (конструкция) языка заменяется соответствующей логической подсхемой либо логическим элементом. Это не всегда эффективно, так как для некоторых классов описаний алгоритмов управления имеются более совершенные методы схемной реализации, дающие схемы меньшей сложности. Например, в работе [1] развита методология схемной реализации параллельных алгоритмов, ориентированная на выделенный класс алгоритмов управления и позволяющая эффективно «кодировать» параллелизм при синтезе логических схем, т.е. данная методология позволяет осуществить глобальный подход к синтезу. Методология опирается на язык ПРАЛУ, предназначенный для описания алгоритмов, в которых используются только двоичные (булевы) переменные, а также операции действия (присвоения значений переменным) и ожидания событий, выраженных также в терминах булевых переменных.

В статье описывается программный комплекс *Micel*, позволяющий проводить схемную реализацию параллельных алгоритмов логического управления, представленных на языке ПРАЛУ. Комплекс *Micel* интегрирован по входным данным с синтезатором логических схем *Leonardo* [2] и имеет в своем составе оптимизационные блоки, отсутствующие в *Leonardo* – это прежде всего минимизация и декомпозиция систем частичных булевых функций. Показывается, что совместное использование *Micel* и *Leonardo* позволяет проводить повторный логический синтез и получать логические схемы с лучшими показателями (быстродействие, сложность), чем использование только одного синтезатора *Leonardo*. Так как все используемые в системе *Micel* промежуточные данные конвертируются в синтезируемый *VHDL*, то *Leonardo* может быть заменен другим синтезатором (*XST*, *Synplify*), который по *VHDL*-описаниям алгоритмов строит логические схемы, например схемы *FPGA*.

Входные и выходные данные программного комплекса *Micel*

ПРАЛУ-описания

Описания параллельных алгоритмов логического управления на языке ПРАЛУ состоят из множества предложений. Каждое предложение

может состоять из нескольких цепочек. В листинге 1 приведен пример ПРАЛУ-описания, в котором каждое предложение, кроме четвертого, состоит из одной цепочки. Четвертое предложение состоит из двух цепочек.

Согласно [1] *элементарной* называется цепочка вида

$$\mu_i : -k_i' \rightarrow k_i'' \rightarrow \nu_i , \qquad (1)$$

где операция $-k_i'$ или $\rightarrow k_i''$ в ней может отсутствовать. В общем случае элементарная цепочка состоит из четырех частей: μ_i – множество начальных меток цепочки; $-k_i'$ – операция ожидания события k_i'; $\rightarrow k_i''$ – операция действия; ν_i – множество заключительных меток цепочки.

Символы k_i', k_i'' представляют собой элементарные конъюнкции булевых переменных. Конъюнкции k_i' образуются из литералов булевых переменных множества X входных переменных, конъюнкции k_i'' – из литералов переменных множества Y выходных переменных. Если конъюнкция k_i' (k_i'') отсутствует, то предполагается, что она тождественно равна единице. Операция $-k_i'$ представляет собой операцию ожидания события k_i'. Выполнение этой операции сводится к ожиданию события, когда переменные, входящие в конъюнкцию k_i', примут значения, обращающие k_i' в единицу. Операция действия $\rightarrow k_i''$ означает присвоение таких значений переменным конъюнкции k_i'' , при которых k_i'' обращается в единицу. Двоеточие служит разделителем, а стрелка перед ν_i играет роль операции внесения элементов в текущее множество запуска цепочек. Множество M, называемое *множеством запуска* цепочек, получается в результате объединения множеств начальных и заключительных меток всех цепочек.

В первом предложении предполагаем, что когда переменные $x1$, $t1$, $t2$ примут значение ноль, а переменная $x2$ – значение единица, выполнится операция действия, состоящая в том, что переменные $y1$, $t1$ принимают значение единица, а переменная $y2$ – значение ноль. После этого ожидается событие, заключающееся в том, что переменная $x2$ принимает значение ноль. В случае, когда такое событие произойдет, активными одновременно (параллельно) становятся предложения с номерами 2, 3, 4.

Листинг 1.

```
TITLE example3
FORMAT PRL
AUTHOR Bibilo
DATE 12/09/08
PROJECT USIM
DCL_PIN
EXT
INP
x1 x2
OUT
y1 y2
INTER
t1 t2
END_PIN
BLOCK example3main
1:    -^x1*x2*^t1*^t2 > y1*^y2*t1 -^x2
> 2.3.4;
2:   -t1 > ^y1 > 5;
3.5: -x2*t1  > ^t1 > 8;
4:   -^x1*^t2 > ^y1*^t1 > 6;
     -x1*^t2 > y2*t2 > 9;
6:   -^x2*^t1*^t2 > t2 > 9;
8.9: -^t1*t2 > ^y2 > 7;
7:   -x1*t2 > ^t1*^t2 > 1;
END_BLOCK example3main
END_example3
```

VHDL-описания систем частичных булевых функций

Системы частичных булевых функций задаются на булевых наборах значений аргументов, значениями частичных булевых функций могут быть 0,1, неопределенное значение «–». В табл. 1 задан пример выделенного класса систем частичных функций. Для этого класса характерно то, что на каждом из наборов значений входных переменных все функции системы либо определены (0,1), либо не определены (–). Такие системы функций легко описываются на *VHDL*. В листинге 2 представлено функциональное *VHDL*-описание системы частичных функций, заданных в табл. 1. Заметим, что для используемого типа `std_logic_vector` «–» и есть неопределенное значение (*don't care*).

Таблица 1. Система частичных булевых функций

a(3)	a(2)	a(1)	a(0)	s(2)	s(1)	s(0)
0	0	0	0	0	0	0
0	0	0	1	–	–	–
0	0	1	0	–	–	–
0	0	1	1	–	–	–
0	1	0	0	–	–	–
0	1	0	1	0	0	1
0	1	1	0	0	1	0
0	1	1	1	–	–	–
1	0	0	0	0	0	0
1	0	0	1	0	1	0
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	–	–	–

Таблица 2. Система ДНФ полностью определенных булевых функций

a(3)	a(2)	a(1)	a(0)	s(2)	s(1)	s(0)
1	0	1	0	1	0	0
–	0	–	1	0	1	0
–	1	–	0	0	1	0
0	1	0	1	0	0	1

Листинг 2. *VHDL*-описание системы частичных функций

```
  library ieee;
  use ieee.std_logic_1164.all;
  entity umn is
    port (a : in  std_logic_vector (3
downto 0);
          s    : out std_logic_vector (2
downto 0));
  end umn;
  architecture BEHAVIOR of umn is
  begin
    s <=
      "000" when a = "0000" else
      "001" when a = "0101" else
      "010" when a = "0110" else
      "000" when a = "1000" else
      "010" when a = "1001" else
      "100" when a = "1010" else
      "---";
  end BEHAVIOR;
```

VHDL-описания уровня RTL

В листинге 3 приводится *RTL*-описание многоуровневого представления системы полностью определенных функций, реализующих систему частичных функций, представленную в листинге 2. В *RTL*-описании схем комбинационной логики используются только четыре логических оператора *VHDL and* (И), *or* (ИЛИ), *not* (НЕ), *xor* (исключающее ИЛИ).

Листинг 3. *RTL*-описание системы полностью определенных функций

```
  library IEEE;
  use IEEE.STD_LOGIC_1164.all;
  entity umn is
     port (
        a : IN std_logic_vector (3
DOWNTO 0) ;
        s : OUT std_logic_vector (2
DOWNTO 0)) ;
  end umn ;
  architecture BEHAVIOR of umn is
   signal nx75, nx78, nx80, nx3, nx4_1,
nx4, nx5, nx2, nx92,
  nx4_2, nx96, nx97, nx6, nx8, nx7:
std_logic ;

  begin
     nx3 <= NOT a(2) ;
     nx4_dup_0 <= NOT a(0) ;
     nx4 <= nx3 AND nx4_1 ;
     nx5 <= NOT nx75 ;
     s(2) <= nx4 AND nx5 ;
     nx2 <= NOT a(1) ;
     nx92 <= NOT a(3) ;
     nx75 <= nx2 OR nx92 ;
     nx4_2 <= NOT nx78 ;
     nx96 <= NOT nx80 ;
     nx97 <= nx4_2 AND nx96 ;
     nx6 <= NOT a(3) ;
     nx8 <= nx97 AND nx6 ;
     nx7 <= NOT a(1) ;
     s(0) <= nx8 AND nx7 ;
     nx78 <= NOT a(2) ;
     nx80 <= NOT a(0) ;
     s(1) <= a(0) XOR a(2) ;
  end BEHAVIOR ;
```

Чтобы получить систему ДНФ по *RTL*-описанию, надо провести элиминацию внутренних переменных.

Внутренние данные

Внутренними моделями при схемной реализации ПРАЛУ-описаний являются модели параллельного, секвенциального и последовательностного автоматов. Данные модели подробно представлены в работе [1] и в статье не описываются. Заметим лишь, что последовательностный автомат представляется в виде комбинационной схемы, функции которой задаются

в виде ДНФ, и элементов памяти – регистра *RS*-триггеров. Последовательностный автомат описывается на языке *SF* [3] – внутренним языком *Micel*.

Архитектура *Micel*

Программный комплекс состоит из трех подсистем, ориентированных на различные виды входных данных и на решение разных задач проектирования.

***Подсистема* 1** преобразования и оптимизации параллельных алгоритмов логического управления. Взаимодействие подсистемы 1 с синтезатором *Leonardo* показано на рис. 1.

***Подсистема* 2** оптимизации представлений систем полностью определенных булевых функций позволяет реализовать маршрут проектирования 2 (рис. 2).

***Подсистема* 3** оптимизации систем частичных функций реализует маршрут проектирования 3 (рис. 2).

Исходя из общей концепции создания программного комплекса с дружественным интерфейсом, управление проектированием в системе *Micel* осуществляется с использованием нескольких мастеров (*Wizard* в терминологии пользовательских интерфейсов, применяемой в современных условиях, в частности компанией *Microsoft*). Мастер – это специальная программа, обеспечивающая интерактивное решение требуемой задачи, например, проектирования, на основе определенного, заранее сформулированного и последовательно реализуемого сценария. Применение техники мастеров существенным образом ограничивает возможности выбора исполнения тех или иных предусмотренных в системе функций, но вместе с тем, дисциплинирует пользователя и облегчает процесс овладения системой. Применение мастеров особенно полезно в условиях, когда последовательность шагов сценария диалога не сильно разветвлена. В соответствии с выделенными подсистемами в системе *Micel* используются разные мастера, каждый из которых ответственен за свой маршрут проектирования.

***Маршрут проектирования* 1.** Требуемый алгоритм управления формулируется в виде описания на языке ПРАЛУ, затем осуществляется проверка синтаксической корректности и проводится моделирование. Построенный таким образом алгоритм конвертируется в язык *VHDL* [4]. Далее полученный алгоритм управления представляется в виде параллельного автомата.

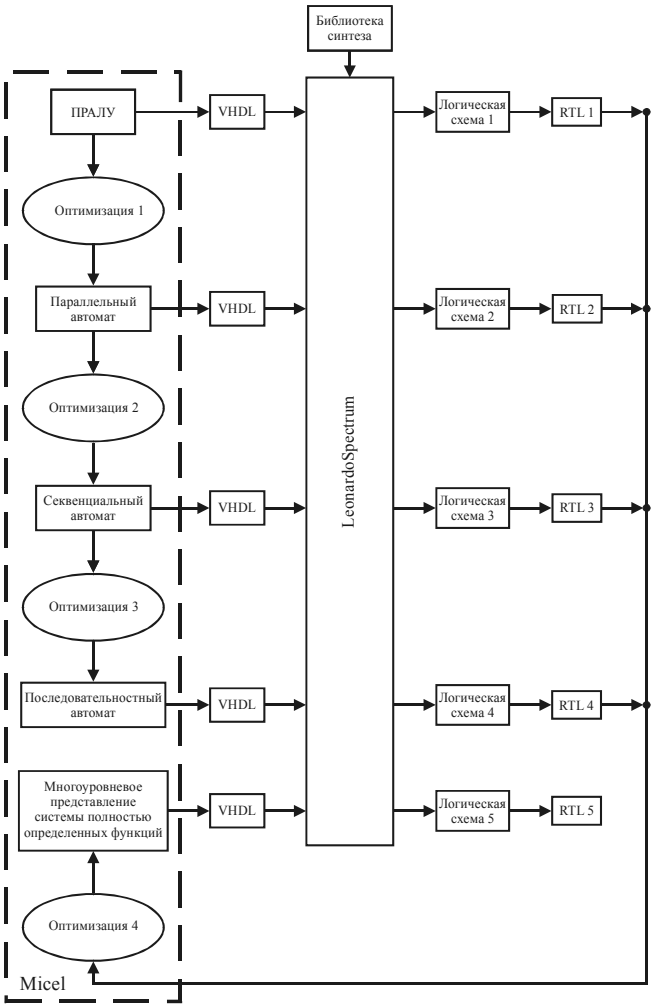

Рис. 1. Маршрут 1 проектирования

Параллельный автомат – модель, которая, в отличие от традиционной модели конечного автомата, может одновременно находиться в нескольких состояниях. Переходы происходят не между отдельными состояниями, а между подмножествами. В рамках модели параллельного автомата осуществляется проверка различных свойств алгоритма, связанных с его «параллельностью». В ходе этой проверки с целью достижения требуемых свойств пользователь может

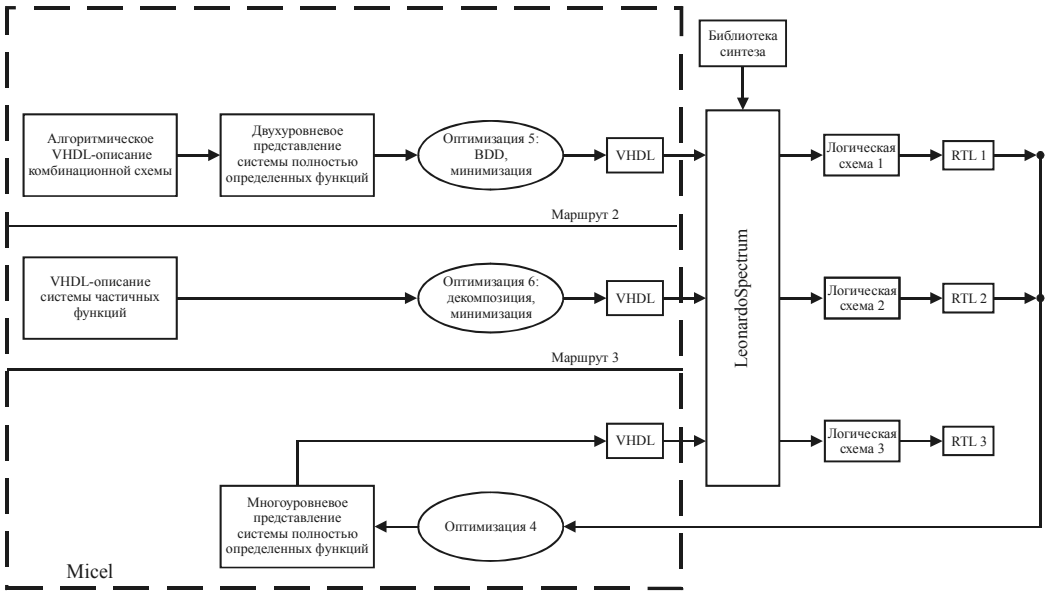

Рис. 2. Маршруты 2 и 3 проектирования

осуществить коррекцию исходного описания в терминах языка параллельных автоматов либо вернуться к предыдущему шагу и преобразовать исходное ПРАЛУ-описание. После того, как удовлетворяющая необходимым свойствам модель окажется определенной, осуществляется построение соответствующего ей представления в языке *VHDL*.

Переход от параллельного к секвенциальному автомату связан с выполнением кодирования состояний параллельного автомата (оптимизация 2). *Секвенциальный автомат* – динамическая логическая модель дискретной системы со многими переменными, определяемая формально [5] как множество S секвенций s_i. Каждая секвенция s_i имеет форму $f_i|-k_i$, определяет причинно-следственное отношение между некоторым сложным событием, представленным булевой функцией f_i (заданной в ДНФ), и простым событием, представленным конъюнктивным термом k_i; $|-$ является символом рассматриваемого отношения. Выражение $f_i|-k_i$ интерпретируется следующим образом: если в некоторый момент времени f_i принимает значение единица, то непосредственно вслед за этим k_i также принимает значение единица. При этом значения всех переменных в k_i определяются однозначно.

Секвенциальный автомат также может быть конвертирован в *VHDL*-описание. По полученному секвенциальному автомату осуществляется построение модели в виде *SF*-описания классического последовательностного автомата, представляемого в виде комбинационной схемы, функции которой задаются в виде системы ДНФ полностью определенных булевых функций, и элементов памяти – в виде регистра *RS*-триггеров. Как и ранее, такой объект может быть транслирован в *VHDL*-описание.

Четыре вида *VHDL*-моделей являются синтезируемыми, по ним могут быть получены логические схемы 1–4 в синтезаторе *Leonardo*. Пользователь может выбрать лучший для него вариант по критериям сложности и/или быстродействия. Попытка получения лучшего решения связана с повторным синтезом: для каждой из четырех логических схем может быть получено соответствующее *RTL*-описание (*Register Transfer Level* – уровень регистровых передач). Используя блок «оптимизация 4» можно получить многоуровневое представление комбинационной части логической схемы 1–4, подать его на вход *Leonardo* и получить логическую схему 5 – результат *повторного* синтеза. Как показали экспериментальные исследования [6], повторный синтез часто бывает эффективным именно для комбинационных схем.

***Маршрут проектирования* 2**. Исходное алгоритмическое *VHDL*-описание, составленное из операторов назначения сигнала и логических операций, преобразуется в *SF*-описание, задающее систему полностью определенных булевых функций. Система преобразуется в сис-

тему ДНФ, которая в блоке «оптимизация 5» минимизируется в классе ДНФ либо оптимизируется *BDD*-представление этой системы функций. Для минимизации в классе ДНФ используются программы [7] раздельной либо совместной минимизации системы функций. Критерии оптимизации – число литералов в записи системы функций либо общее число элементарных конъюнкций, на которых заданы функции системы. *BDD*-представление (*binary decision diagrams* – диаграммы двоичного выбора) [8] является одним из видов *RTL*-представлений и строится на основе разложения Шеннона системы функций по всем переменным. В отечественной литературе для таких разложений употребляется термин «бинарная программа» [9]. Оптимизация *BDD* заключается в нахождении одинаковых коэффициентов Шеннона при разложении всех функций системы по тем же самым переменным. Полученные после оптимизации минимизированные ДНФ либо оптимизированные по числу вершин *BDD*-представления конвертируются в *VHDL*-описания.

***Маршрут проектирования* 3**. Исходным объектом проектирования в маршруте 3 является система частичных булевых функций (см. рис. 2). Блок «оптимизация 6» имеет две возможности: провести совместную минимизацию исходной системы в классе ДНФ и получить минимизированную систему ДНФ полностью определенных функций и провести декомпозицию системы функций по двухблочному разбиению множества переменных. Алгоритм декомпозиции реализует известный метод [10] декомпозиции системы частичных функций, основанный на построении графа отношения несовместимости и раскраске графа в минимальное число цветов.

Выделяемый блок при декомпозиции может быть реализован на ПЗУ(n, m) , где n – число входных полюсов ПЗУ (размерность адреса), m – число выходных полюсов (размерность слова хранимых данных) емкости 4096 бит. В конкретной реализации допускаются следующие значения: $n = 12$, $m = 1$; $n = 11$, $m = 2$; $n = 10$, $m = 4$; $n = 9$, $m = 8$; $n = 8$, $m = 16$. Тем не менее, декомпозиция может быть проведена и по большему, чем 12, числу входов в отделяемом от схемы блоке.

Пользователь может выбрать указанное им число ПЗУ с требуемыми параметрами, после чего оставшаяся часть схемы представляется одним функциональным блоком и может быть реализована впоследствии схемой в заданном библиотечном базисе. Полученная после оптимизации 6 система булевых функций является полностью определенной, поэтому в маршруте 3 имеется программа верификации, цель которой – проверка отношения реализуемости, т.е. реализует ли оптимизированная система полностью определенных функций исходную систему частичных функций.

После выделения ПЗУ и оптимизации логическая схема конвертируется в *VHDL*-описание, которое может быть подано на вход синтезатора *Leonardo*. Функциональные блоки, реализуемые на ПЗУ, представляются в виде таблиц истинности – по сути, представляют собой «прошивки» ПЗУ.

Эксперименты

Для каждого из маршрутов 1 – 3 проведен вычислительный эксперимент.

***Маршрут* 1.** Результаты эксперимента для маршрута 1 представлены в табл. 3. В строках «Конвертация» представлены результаты прямого синтеза схем по различным *VHDL*-моделям, представленным в столбцах, а в строках «Оптимизация *RTL*» представлены результаты повторного синтеза по многоуровневым представлениям функций, оптимизированным в блоке «Оптимизация 4» по соответствующим *RTL*1 – *RTL*4. Использовалась библиотека логических элементов, описанная в [6]. В табл. 3: T – число триггеров; S – площадь комбинационной части логической схемы; L – число библиотечных логических элементов (исключая триггеры); τ – задержка схемы (нс). Жирным шрифтом выделены лучшие решения. Рассмотрим пример ПРАЛУ-описания с именем *vertical*. Конвертация этого описания в *VHDL* и последующий синтез позволяет получить логическую схему 1, состоящую из $T = 37$ триггеров,

Таблица 3. Реализация параллельных алгоритмов логического проектирования в маршруте 1

Схема	Процесс проектирования	ПРАЛУ-описание (логическая схема 1)				Параллельный автомат (логическая схема 2)				Секвенциальный автомат (логическая схема 3)				*SF* – описание (логическая схема 4)			
		T	*S*	*L*	τ	*T*	*S*	*L*	τ	*T*	*S*	*L*	τ	*T*	*S*	*L*	τ
Vertical	Конвертация	37	546	186	16,10	33	970	306	41,50	15	566	189	33,95	15	639	221	29,05
	Оптимизация *RTL*	**37**	**532**	**174**	**19,10**	33	854	261	44,35	15	882	276	29,25	15	691	240	26,40
*Zak*135*m*	Конвертация	25	406	135	12,10	22	659	207	34,80	12	421	141	21,45	12	411	137	21,10
	Оптимизация *RTL*	25	387	120	18,20	22	551	178	31,90	**12**	**387**	**120**	**18,20**	12	480	153	18,40
Pott1	Конвертация	13	202	71	8,20	13	343	107	23,45	7	140	43	7,05	7	149	48	9,40
	Оптимизация *RTL*	13	186	62	8,80	13	236	72	9,00	**7**	**124**	**36**	8,10	7	127	38	**6,95**
Traffic	Конвертация	3	32	12	4,35	16	253	90	8,30	9	175	60	12,50	8	146	47	-
	Оптимизация *RTL*	**3**	**27**	**9**	**2,70**	16	214	67	11,60	9	173	58	10,00	8	-	-	11,80
*Pott*3	Конвертация	7	126	43	9,35	14	459	162	25,95	8	207	65	14,40	10	220	73	11,80
	Оптимизация *RTL*	**7**	**111**	**42**	**9,10**	14	364	136	22,60	8	184	63	12,70	10	207	72	9,80

$L = 186$ комбинационных логических элементов; с площадью $S = 546$ условных единиц; задержка схемы $\tau = 16{,}10$ нс. Повторный синтез по *RTL*1 (см. рис. 1) позволяет улучшить результат (см. строку ниже) по площади – получить схему с параметром $S = 532$, но задержка схемы при этом увеличивается на 3 нс, $\tau = 19{,}10$ нс. Таким образом, использование конвертации в *VHDL* и повторный синтез могут быть целесообразными с различных уровней представления данных.

***Маршрут* 2.** Результаты эксперимента представлены в табл. 4. Тестовые примеры взяты из множеств тестовых примеров *MCNC BENCHMARK SET* и *itc*99-*poli*2, символом * помечены примеры автоматов, в отличие от примеров комбинационных схем. Синтез схем проводился в той же библиотеке, что и в эксперименте для маршрута 1; обозначения *S*, τ имеют тот же смысл, что и в табл. 3; *n*, *m* – число входных и выходных полюсов схемы соответственно. Эксперимент показывает, что переход к двухуровневым представлениям функций и оптимизация в *Micel* позволяют увеличить быстродействие логических схем, а иногда и их сложность.

***Маршрут* 3.** В качестве примеров исходных данных были выбраны два описания *verg*1, *verg*2 (табл. 5) таблиц микрокоманд отечественных микроконтроллеров. Таблицы описывались как системы неполностью определенных (частичных) булевых функций, пример описаний дан в листинге 2. В табл. 5: *n*, *m* – число входов и функций, *k* – число наборов (строк таблицы микрокоманд), τ – задержка схемы (нс). Синтез проводился в другой библиотеке проектирования, площадь *S* подсчитывалась в других условных единицах. Эксперимент показал, что оптимизация в маршруте 3 позволяет уменьшить сложность схемы, но задержка при этом, как правило, увеличивается.

Таблица 4. Реализация комбинационных схем и автоматов в маршруте 2

Схема			*Leonardo*		Совместная минимизация + *Leonardo*		Раздельная минимизация + *Leonardo*		*BDD* + *Leonardo*	
Имя	*n*	*m*	*S*	τ	*S*	τ	*S*	τ	*S*	τ
*F*51m	8	8	116	9,90	**113**	8,90	**113**	8,90	118	8,95
*Frg*1	28	3	**296**	20,40	320	**19,30**	320	19,30	–	–
*b*01*	4	2	155	**11,15**	144	12,90	163	15,35	**125**	12,70
*b*02*	3	1	**48**	9,70	53	**9,50**	53	**9,50**	51	10,60
*B*06*	4	6	**98**	14,40	133	11,70	136	10,30	155	**9,70**
LAL	26	19	**313**	13,50	354	**12,60**	354	**12,60**	–	–
*PM*1	16	13	**135**	9,30	141	**6,20**	141	**6,20**	–	–
*C*8	28	18	**226**	6,30	227	**6,20**	227	**6,20**	–	–
*CM*162*A*	14	5	79	10,20	**77**	**7,50**	**77**	**7,50**	–	–
*b*03*	6	1	**441**	28,80	543	**26,30**	542	26,40	–	–
*B*07*	3	1	**1275**	**39,20**	3265	44,95	3362	46,40	–	–
*B*09*	3	1	**455**	**19,20**	1013	33,30	1301	35,10	–	–

Таблица 5. Реализация систем частичных функций в маршруте 3

Схема				Синтез по исходному описанию			Синтез минимизированного «остатка» после выделения ПЗУ(14,7)			Синтез по минимизированной Системе ДНФ		
Имя	*n*	*m*	*k*	*S*	*L*	τ	*S*	*L*	τ	*S*	*L*	τ
*Verg*1	17	61	2004	1211630	2820	**12,60**	460736	1086	5,26	**342356**	**798**	13,73
*Verg*2	18	63	2130	1324215	3177	**13,21**	550706	1311	5,21	**383538**	**905**	13,73

Заключение. В статье описана архитектура, входные, выходные данные и решаемые задачи в системе *Micel*. Эксперименты с практическими примерами показывают, что интеграция данной системы с промышленными синтезато-

рами позволяет варьировать параметрами схемы (сложность, быстродействие) и выбирать лучшие схемные решения при схемной реализации как параллельных алгоритмов логического управления, так и *RTL*-описаний систем полностью определенных функций и табличных форм частичных функций.

1. *Закревский А.Д.* Параллельные алгоритмы логического управления. – Минск: Ин-т техн. кибернетики НАН Беларуси, 1999. – 202 с.
2. *Бибило П.Н.* Системы проектирования интегральных схем на основе языка *VHDL*. StateCAD, ModelSim, LeonardoSpectrum. - М.: СОЛОН-Пресс, 2005. – 384 с.
3. *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.
4. *Бибило П.Н.* Представление ПРАЛУ-описаний параллельных алгоритмов логического управления на языке *VHDL* // Микроэлектроника. – 2006. – Т. 35. – № 4. – С. 306–320.
5. *Закревский А.Д.* Логический синтез каскадных схем. – М.: Наука, 1981. – 416 с.
6. *Бибило П.Н.*, *Романов В.И.* Новые эксперименты повторного синтеза комбинационных схем // Микроэлектроника. – 2008. – Т. 37. – № 3. – С. 228–240.
7. *Торопов Н.Р.* Минимизация систем булевых функций в классе ДНФ // Логическое проектирование. – Минск: Ин-т техн. кибернетики НАН Беларуси, 1999. – Вып. 4. – С. 4–19.
8. *Akers S.B.* Binary Decision Diagrams // IEEE Trans. on Computers. – 1978. – **27**. – № 6. – P. 509–516.
9. *Кузнецов О.П.* О программной реализации логических функций и автоматов // Автоматика и телемеханика. – 1977. – № 7. – С. 63–74; № 9. – С. 138–149.
10. *Бибило П.Н.*, *Енин С.В.* Синтез комбинационных схем методами функциональной декомпозиции. – Минск: Наука и техника, 1987. – 189 с.

Поступила 20.12.2008

Тел. для справок: (37517) 284-2084, 231-8403 (Минск)
E-mail: *bibilo@newman.bas-net.by*

●

Окончание статьи А.А. Баркалова и др.

1. *Baranov S.* Logic Synthesis for Control Automata. – Kluwer Academic Publishers, 1994. – 312 p.
2. *Грушвицкий Р.И.*, *Мурсаев А.Х.*, *Угрюмов Е.П.* Проектирование систем с использованием микросхем программируемой логики. – СПб: БХВ. – Петербург, 2002. – 608 с.
3. *Соловьев В.В.* Проектирование цифровых схем на основе программируемых логических интегральных схем. – М.: Горячая линия–ТЕЛЕКОМ, 2001. – 636 с.
4. *Barkalov A.*, *Titarenko L.* Logic Synthesis for Compositional Microprogram Control Units. – Berlin: Springer, 2008. – 272 p.
5. *Altera* devices overview. – http://www.altera.com/products/devices/common/dev-family_overview.html
6. *Xilinx* CPLDs. – http://www. ilinx.com/products/silicon_solutions/cplds/index.htm
7. *Баркалов А.А.*, *Зеленёва И.Я.*, *Лаврик А.С.* Использование особенностей ПЛИС для оптимизации схемы устройства управления / Наук. пр. Донецького нац. техн. ун-ту. Серія «Інформатика, кібернетика і обчислювальна техніка» (ІКОТ–2008). Вип. 9 (132) – Донецьк: ДонНТУ. – 2008. – С. 178–182.
8. *Оптимизация* устройства управления с преобразователем адреса микрокоманд / А.А. Баркалов, С.А. Ковалев, А.А. Красичков и др. // Материалы Девятого Междунар. науч.-практ. семинара. В 3-х кн. – Таганрог. Кн. 3. – 2008. – С. 12–20.
9. *CoolRunner* CPLD Datasheet. – http://www.xilinx.com/support/documentation/coolrunner-ii.htm
10. *Maxfield C.* The Design Warrior's Guide to FPGAs. – Amsterdam: Elseveir, 2004. – 541 p.

Поступила 31.03.2009

Тел. для справок: (XXX) 301-0723 (Донецк)
E-mail: *A.Barkalov@iie.uz.zgora.pl*

●