УДК 681.325.53: 37 : 004.5

*Н.Я. КАКУРИН, В.В. ВАРЕЦА, С.Н. КОВАЛЕНКО*

# ПАРАЛЛЕЛЬНАЯ СТРАТЕГИЯ ИСПОЛЬЗОВАНИЯ ШАГОВ В ДВУХШАГОВЫХ ПРЕОБРАЗОВАТЕЛЯХ КОДА

Рассматривается структура и возможности программного пакета «Converter» для анализа преобразования чисел и обоснованного выбора основных системных параметров проектируемого преобразователя.

## 1. Постановка задачи

Основными параметрами преобразователей кодов по методу накопления эквивалентов является быстродействие (число тактов преобразования) и аппаратурные затраты (число корпусов ИС или число вентилей) [1,2]. К достоинствам преобразователей кодов этого типа относится возможность изменения соотношения между быстродействием и аппаратурными затратами за счет выбора числа шагов преобразования, значений шагов и стратегии использования различных шагов преобразования.

Стратегия использования шагов преобразования может быть последовательной или параллельной.

При последовательной стратегии показания разрядных счетчиков, значения которых равны или превышают значение большего шага, уменьшают на значение этого шага.

Если же во всех преобразуемых разрядах значения цифр оказываются меньше этого шага, то происходит переход на более меньший шаг, и в дальнейшем вновь ведется уменьшение значений цифр до тех пор, пока не будут обнулены все разрядные счетчики.

Последовательная стратегия по сравнению с параллельной структурно реализуется достаточно просто. Аппаратные затраты на построение основного блока ПК – формирователя эквивалентов – будут небольшими.

Оценку числа тактов преобразования целых чисел в многошаговых ПК с последовательной стратегией выполняют по формулам:

$$N_1^{цел} = K\text{-}1;$$

$$N_2^{цел} = ](K\text{-}1)/a[+a\text{-}1; \qquad (1)$$

$$N_3^{цел} = ](K\text{-}1)/b[+](b\text{-}1)/a[+a\text{-}1;$$

$$N_4^{цел} = ](K\text{-}1)/c[+](c\text{-}1)/b[+](b\text{-}1)/a[+a\text{-}1,$$

где К – основание системы счисления на входе; a, b, c – соответственно второй, третий и четвертый шаги преобразования (первый шаг всегда равен 1); $N_1^{цел}$, $N_2^{цел}$, $N_3^{цел}$, $N_4^{цел}$ - соответственно максимальное число тактов преобразования одно-; двух-; трех-; четырехшагового ПК целых чисел

Для значений шагов преобразования должны выполняться следующие ограничения:

$$1<a<b;\ a<b<c;\ b<c \le K\text{-}1.$$

Недостатком использования максимальных значений $N_i^{цел}$ ($i = \overline{1,4}$) для оценки быстродействия являются завышенные требования, которые для определенных подмножеств преобразуемых чисел будут значительны и потребуют применения в структуре ПК быстродействующих интегральных схем [3,4].

В целях реальной оценки быстродействия разработан программный пакет «Converter», позволяющий промоделировать процесс преобразования чисел для параллельной стратегии и определить конкретное число тактов преобразования, а не оценку сверху.

## 2. Программная реализация режима преобразования чисел

В качестве инструмента для разработки программного пакета «Converter» была использована среда программирования C++ Builder 6. Данная среда представляет собой так называемую среду быстрой разработки (RAD-среда), которая берет на себя всю рутинную работу, связанную с подготовкой программы к работе[5]. Она автоматически генерирует соответствующий программный код и позволяет сосредоточиться не на оформлении интерфейса, а на логике работы будущей программы.

Данный программный пакет может работать в двух режимах: в режиме преобразования чисел и в режиме определения минимального числа тактов преобразования в функции от набора шагов.

Для реализации режима преобразования чисел необходимо создать главную форму. Для выполнения данного действия производится вызов пунктов меню File>New>Application. Созданная таким образом форма получит автоматически имя Form1.

Для осуществления работы приложения в двух режимах на форму помещается компонент набора страниц PageControl. На компоненте создаются две страницы: одна с именем “Режим преобразования чисел”, а другая с именем “Режим определения минимального числа тактов преобразования в функции от набора шагов ”.

В данном подразделе будет рассмотрена первая страница компонента набора страниц PageControl. На данной странице расположен компонент GroupBox1. На компоненте GroupBox1 расположены компоненты CspinEdit, которые имеют следующие назначения:

1) Компонент NumSteps – определяет количество шагов преобразователя. Свойству MaxValue присвоено значение 20, а свойству MinValue – 1, т.е. количество шагов преобразователя может быть от 1 до 20. Свойству Value присвоено значение 3, т.е. по умолчанию количество шагов преобразователя является равным 3.

2) Компонент BasisSystem – задает основание системы счисления для числа, которое необходимо преобразовать. Свойство MaxValue равно 30, а свойство MinValue имеет значение, равное 2, т.е. основание системы может изменяться от 2 до 30. Свойству Value присвоено значение, равное 10, т.е. по умолчанию установлена десятичная система счисления.

3) Компонент NumDigs – задает количество разрядов числа, которое необходимо преобразовать. У компонента свойство MaxValue равно 50, а свойство MinValue – 1, т.е. число, которое необходимо преобразовать, может содержать от 1 до 50 разрядов. Свойство Value равно 10, т.е. по умолчанию число состоит из 10 разрядов.

На компоненте GroupBox1 также расположены компоненты StringGrid, которые имеют следующие назначения:

1) Компонент NumGrid – содержит число, которое необходимо преобразовать.

2) Компонент WeightGrid – содержит набор шагов весов преобразования. Первому элементу данного компонента изначально присваивается 1.

Для вывода информационной текстовой информации на компоненте GroupBox1 расположены компоненты Label, которые имеют следующие назначения: Label1 – “Веса шагов преобразования”, Label2 – содержит текстовую подпись “Количество шагов преобразователя”, Label3 – “Основание системы счисления”, Label4 – “Разрядность числа” и Label5 – “Число, которое необходимо преобразовать”.

Для получения информации о десятичном представлении параметра Sn и двоичном представлении значения накапливающего сумматора служат компоненты Label, которые имеют имена S10 и Sm. Данные компоненты расположены на компоненте GroupBox2. Также на данном компоненте расположены компоненты Label_Sk и Label_Sm, которые используются для отображения вспомогательной текстовой информации. Компонент Label_Sk отображает строку “Параметр Sn в десятичном представлении”. Здесь обозначение n определяет основание системы счисления. Компонент Label_Sm используется для отображения строки текста “Параметр Sm в двоичном представлении”.

Для отображения результата преобразования числа используется компонент ListView, который находится на закладке Win32. Данному компоненту присвоено имя ListView1. Также у данного компонента создано четыре столбца: для отображения номера такта, параметра Xn…X1, параметра Sn и параметра Sm, определяющего значение накапливаю-

щего сумматора. После выполнения данных действий окно формы приложения имеет вид, приведенный на рис. 1.

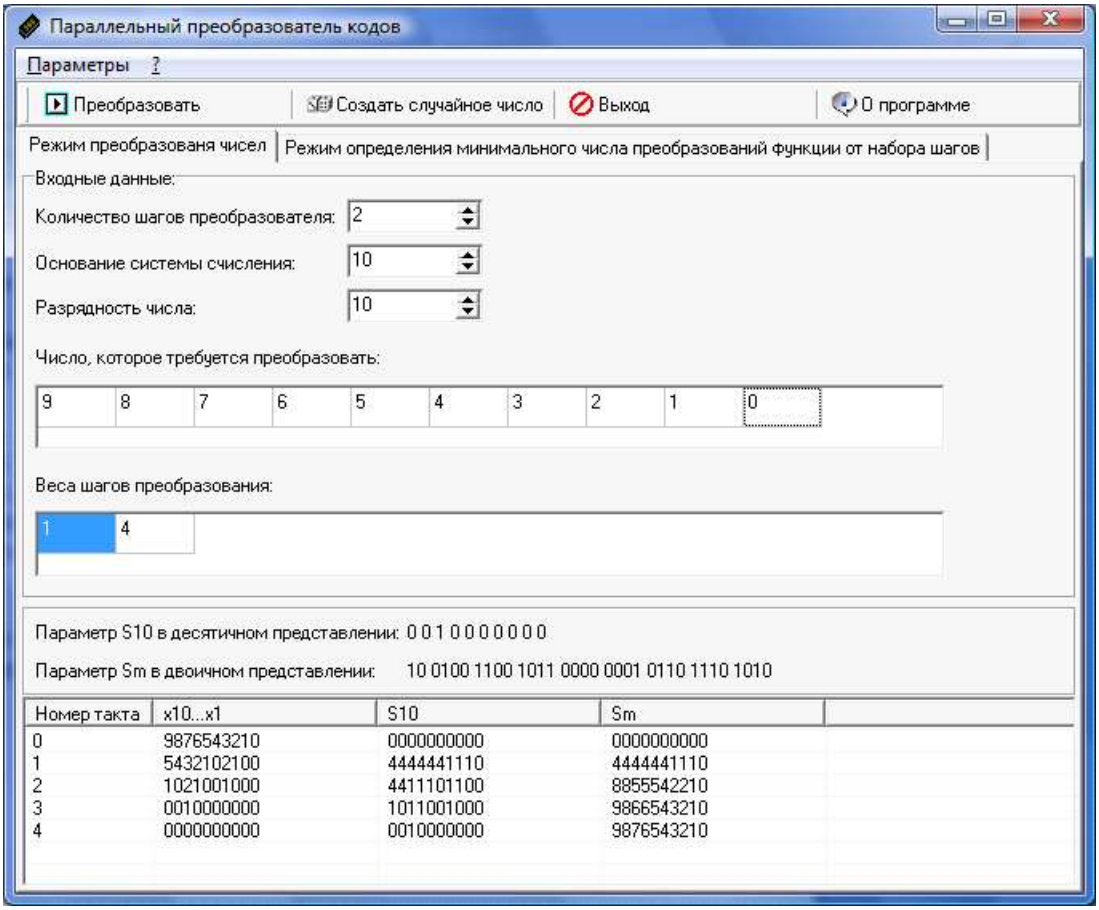


Рис. 1. Форма приложения в режиме преобразования чисел и результат преобразования для K=10; n=10 и набора шагов 1,4

При нажатии на кнопку “Преобразовать” осуществляется операция преобразования числа, задаваемого в компоненте NumGrid с использованием весов, которые задаются в компоненте WeightGrid. Если программа находится в режиме преобразования чисел, то происходит вызов функции ParallConvert. Результатом работы данной функции является текстовая информация, описывающая процесс преобразования. В процессе своей работы функция использует следующие структуры данных:

1) переменная K – определяет основание системы счисления;

2) переменная numDigs – определяет количество разрядов числа, которое необходимо преобразовать;

3) переменная numDigsStep – определяет количество шагов преобразователя;

4) вектор numArray – хранит число, которое необходимо преобразовать;

5) вектор weightArray – хранит веса шагов преобразователя;

6) переменные step1 и step2 хранят состояние преобразователя на текущем и последующем шагах.

Для описания переменных step1 и step2 введен новый тип данных – структура Step.

Данная структура имеет следующий вид:

```
struct Step
{ ArrayDigs xArray;
 ArrayDigs sKArray;
 ArrayDigs sMArray;};
```

Здесь вектор xArray – определяет параметр Xn…X1 преобразователя, вектор sKArray – параметр $S_n$; вектор sMArray – значение сумматора.

### 3. Функционирование программы в режиме преобразования чисел

Режим преобразования чисел используется для преобразования числа из К-ичной системы счисления в двоичную. Программа выдает не только окончательный результат преобразования, но и промежуточные результаты в виде таблицы: состояния разрядов счетчиков Xn…X1, величину эквивалента и состояние накапливающего сумматора на каждом такте преобразования. Для того чтобы произвести расчет в данном режиме, необходимо запустить программу путем активизации исполняемого модуля Converter.exe.

После запуска программы появится пять текстовых полей для ввода данных, три из которых снабжены кнопками инкремента. С помощью этих элементов управления необходимо задать исходные данные для режима преобразования чисел: количество шагов преобразователя, основание системы счисления на входе, разрядность преобразуемого числа, собственно само число, которое необходимо преобразовать, и значения шагов преобразования. После ввода всех необходимых данных необходимо нажать на кнопку «Преобразовать».

Структурная схема алгоритма в режиме преобразования чисел приведена на рис.2.

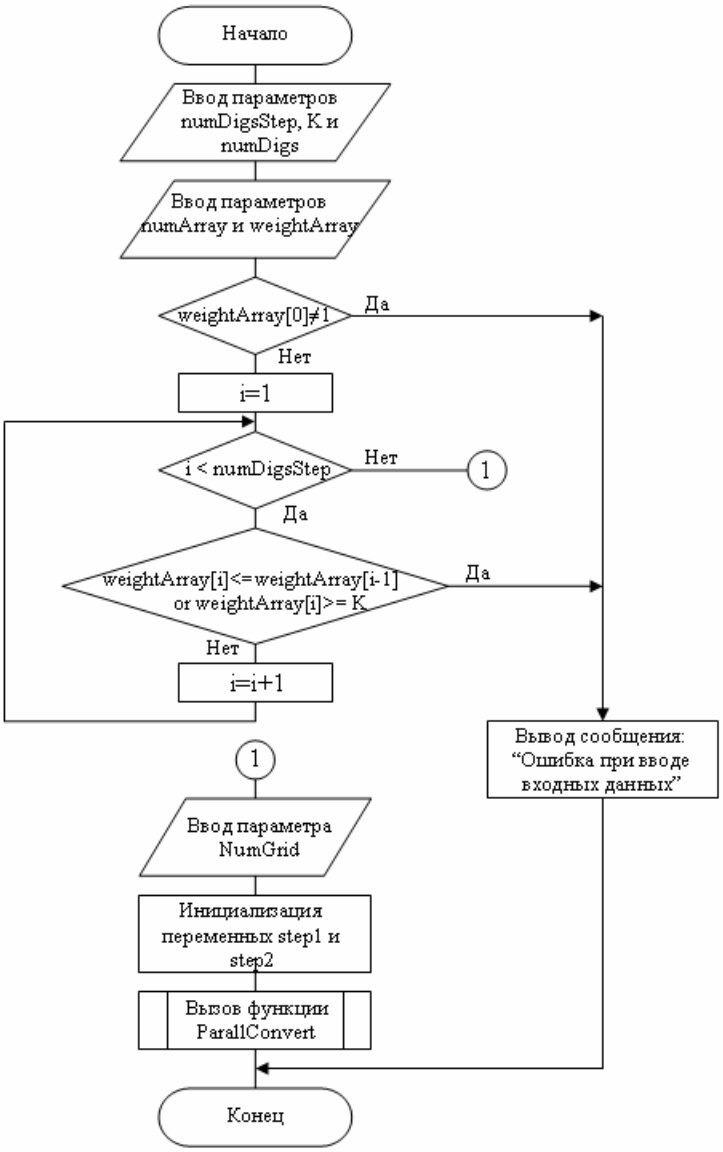


Рис. 2. Структурная схема алгоритма в режиме преобразования чисел

Если введены некорректные данные, то выводится сообщение. После того как программа произведет расчет, на экране появится результат преобразования введенного числа. Пример расчета для десятичной системы счисления десятиразрядного числа 9876543210 представлен на рис.1.

Фрагмент кода функции параллельного преобразования чисел ParallConvert в соответствии с синтаксисом языка C++ приведен ниже:

```
void TForm1::ParallConvert()
{
Step * pStep1 = &step1;
    Step * pStep2 = &step2;
TListItem * Lis tItem;
ListView1->Items->BeginUpdate()
ListView1->Items->Clear();
ArrayDigs::iterator i;
    int step = 0;
    while (1)
    {
ListItem = ListView1->Items->Add();
ListItem->Caption = IntToStr(step);
ListItem->SubItems->Add(ArrayToString(pStep1->xArray, true, ShowSymbDig->Checked));
 ListItem->SubItems->Add(ArrayToString(pStep1->sKArray, ShowZeroS->Checked, ShowSymbDig->Checked));
 ListItem->SubItems->Add(ArrayToString(pStep1->sMArray, ShowZeroS->Checked, ShowSymbDig->Checked));

for (i = pStep1->xArray.begin(); i != pStep1->xArray.end(); ++i)
                        if ((*i) != 0)
                                break;
                if (i == pStep1->xArray.end())
                        break;

                ArrayDigs::iterator iterX2 = pStep2->xArray.begin();
                ArrayDigs::iterator iterSK2 = pStep2->sKArray.begin();

                for (i = pStep1->xArray.begin(); i != pStep1->xArray.end(); ++i)
                {
                        int s = numDigsStep;
                        while (—s >= 0)
                                if ((*i) >= weightArray[s])
                                {
                                        *iterX2 = (*i) - weightArray[s];
                                        *iterSK2 = weightArray[s];
                                        ++iterX2;
                                        ++iterSK2;
                                        break;
                                }
                        if (s < 0)
                        {
                                *iterX2 = 0;
                                *iterSK2 = 0;
                                ++iterX2;
                                ++iterSK2;
                        }
                }
                ArrayDigs::iterator iterSM1 = pStep1->sMArray.begin();
```

```
                    iterSK2 = pStep2->sKArray.begin();

    for (ArrayDigs::iterator iterSM2 = pStep2->sMArray.begin(); iterSM2 != pStep2->sMArray.end();
++iterSM2)
                    {
                        *iterSM2 = (*iterSM1) + (*iterSK2);
                        ++iterSM1;
                        ++iterSK2;
                    }
                    Step * pTemp = pStep1;
                    pStep1 = pStep2;
                    pStep2 = pTemp;
                    ++step;
        }
     ListView1->Items->EndUpdate();
     }
```

Данный фрагмент программы позволит при необходимости ее модернизации определить и переписать соответствующие участки программного кода.

## 4. Результаты исследования двухшагового ПК с параллельным использованием шагов преобразования

С помощью программного средства «Converter» был проведен анализ зависимости числа тактов преобразования двухшаговых ПК с основаниями $К_1 = 10$ (табл.1) и $К_2 = 12$ (табл.2) для различных наборов шагов.

Анализ таблиц показывает, что для основания $К_1 = 10$ наименьшее значение числа тактов преобразования 4 обеспечивают два набора шагов 1,3 и 1,4; а для $К_2 = 12$ наименьшее значение числа тактов преобразования 5 обеспечивают три набора шагов 1,3; 1,4 и 1,5.

Таблица 1

| Номер набора | Набор шагов | $N_2^{мин}$ |
|---|---|---|
| 1 | 1,2 | 5 |
| 2 | 1,3 | 4 |
| 3 | 1,4 | 4 |
| 4 | 1,5 | 5 |
| 5 | 1,6 | 5 |
| 6 | 1,7 | 6 |
| 7 | 1,8 | 7 |
| 8 | 1,9 | 8 |

Таблица 2

| Номер набора | Набор шагов | $N_2^{мин}$ |
|---|---|---|
| 1 | 1,2 | 6 |
| 2 | 1,3 | 5 |
| 3 | 1,4 | 5 |
| 4 | 1,5 | 5 |
| 5 | 1,6 | 6 |
| 6 | 1,7 | 6 |
| 7 | 1,8 | 7 |
| 8 | 1,9 | 8 |
| 9 | 1,10 | 9 |
| 10 | 1,11 | 10 |

Результаты, полученные с помощью программного пакета «Converter» в виде табл.1; табл. 2, обеспечиваются режимом определения минимального числа тактов преобразования в функции от набора шагов.

Достоверность результатов может быть проверена путем наведения курсора мыши на соответствующий набор шагов и нажатия левой кнопки мыши.

Детальные результаты моделирования (по тактам) преобразования с параллельным использованием шагов преобразования для $К_1 = 10$ и набора шагов 1,2, а также для набора шагов 1,4 ($К_1 = 10$) приведены в табл.3 и табл.4 соответственно.

Результаты, приведенные в табл.1 и табл.2, для наглядности отображены на рис.3 и 4.

Таблица 3

| i | $X_{10}$ | $X_9$ | $X_8$ | $X_7$ | $X_6$ | $X_5$ | $X_4$ | $X_3$ | $X_2$ | $X_1$ |
|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| 1 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | 0 | 0 |
| 2 | 5 | 4 | 3 | 2 | 1 | 0 | 0 | 0 | 0 | 0 |
| 3 | 3 | 2 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 4 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 5 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

Таблица 4

| i | $X_{10}$ | $X_9$ | $X_8$ | $X_7$ | $X_6$ | $X_5$ | $X_4$ | $X_3$ | $X_2$ | $X_1$ |
|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| 1 | 5 | 4 | 3 | 2 | 1 | 0 | 0 | 0 | 0 | 0 |
| 2 | 1 | 0 | 2 | 1 | 0 | 0 | 0 | 0 | 0 | 0 |
| 3 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 4 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

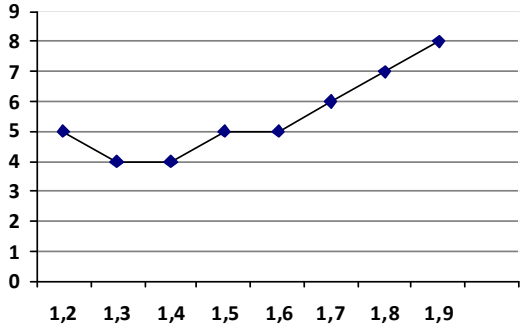


Рис. 3. Зависимость $N_2^{мин}$ для $K_1 = 10$ от набора шагов 1,а

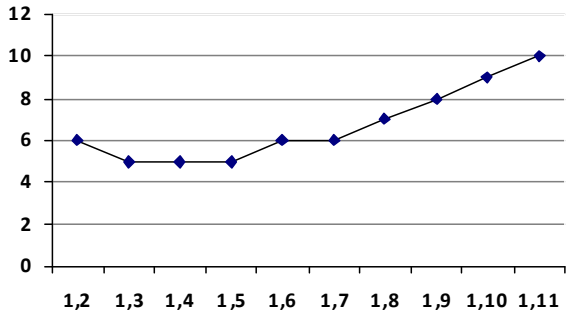


Рис. 4. Зависимость $N_2^{мин}$ для $K_2 = 12$ от набора шагов 1,а

**Выводы**

*Основные результаты.* Разработан программный пакет «Converter», обеспечивающий два режима исследования (режим преобразования чисел и режим определения минимального числа тактов преобразования в функции от набора шагов) для различных значений основания системы счисления К=2-30; для наборов шагов от 1 до 20 и разрядности преобразуемых чисел n=1-50.

*Сравнение с лучшими аналогами.* Предложенный программный пакет обеспечивает анализ процесса преобразования чисел для параллельной стратегии использования шагов преобразования. По сравнению с последовательной стратегией использования шагов число тактов преобразования параллельной стратегии сокращается на 20-33%.

*Практическая значимость.* Программное средство позволяет ускорить проведение этапа системного анализа проектируемых преобразователей кодов за счет параллельного использования шагов преобразования, уменьшить аппаратурные затраты проектируемых преобразователей кодов и сократить число тактов преобразования.

**Список литературы**: **1**. А.С. 1126946 5G06F 5/02. Преобразователь двоично-К-ичного кода в двоичный код / А.Н. Слобожанин // Открытия, изобретения. 1984. №44. С.250. **2**.А.С. 1647908 5НОЗМ 7/12. Преобразователь двоично-К-ичного кода в двоичный код /Н.Я.Какурин, Ю.К. Кирьяков, А.Н. Макаренко // Открытия, изобретения. 1991. № 17. С. 262-263. **3.** *Какурин Н.Я., Макаренко А.Н., Старчевский Д.Л.* Проектирование алгоритмов функционирования преобразователей двоично-десятичных кодов последовательного типа. Часть 1. Проектирование алгоритмов преобразования // АСУ и приборы автоматики. 2005. Вып. 128. С.76-182. **4.** *Какурин Н.Я., Макаренко А.Н., Старчевский Д.Л.* Проектирование алгоритмов функционирования преобразователей двоично-десятичных кодов последовательного типа. Часть 2. Проектирование схемных реализаций // АСУ и приборы автоматики. 2005. Вып. 131. С.167-175. **5.** *Бондарев В.М.* Программирование на С++. Харьков. «Компания СМИТ». 2004. 284 с.

*Поступила в редколлегию 02.11.2007*

**Какурин Николай Яковлевич**, канд. техн. наук, профессор кафедры автоматизации проектирования вычислительной техники ХНУРЭ. Научные интересы: прикладная теория цифровых автоматов, автоматизация проектирования цифровых устройств. Адрес: Украина, 61166, Харьков, пр.Ленина, 14, тел. 70-21-326.

**Вареца Виталий Викторович,** студент группы КСС-04-2 ХНУРЭ. Научные интересы: проектирование программного обеспечения, автоматизация проектирования цифровых устройств. Адрес: Украина, 61166, Харьков, пр.Ленина, 14, тел. 70-21-326.

**Коваленко Сергей Николаевич,** соискатель кафедры АПВТ ХНУРЭ. Научные интересы: цифровые датчики, устройства преобразования кодов, автоматизации проектирования цифровых устройств. Адрес: Украина, 61166, Харьков, пр.Ленина, 14, тел. 70-21-326.

УДК 681.326:519.713

*А.Н. ПАРФЕНТИЙ, В.И. ХАХАНОВ, Е.И. ЛИТВИНОВА*

# МОДЕЛИ ИНФРАСТРУКТУРЫ СЕРВИСНОГО ОБСЛУЖИВАНИЯ ЦИФРОВЫХ СИСТЕМ НА КРИСТАЛЛАХ

Предлагается алгебро-логический метод встроенного сервисного обслуживания функциональностей цифровых систем на кристаллах, ориентированный на решение задач диагностирования дефектов цифровых модулей и восстановления работоспособности матриц памяти. Метод характеризуется матричным заданием модели покрытия неисправностей тестом и дает возможность получать полное и минимальное множества дефектов в функциональностях SoC.

## 1. Архитектура инфраструктуры сервисного обслуживания

Сервисное обслуживание функциональных (F-IP) блоков цифровой системы на кристалле предполагает интегрированное взаимодействие технологий граничного сканирования на основе стандарта IEEE 1500 [1,2] с дополнительными нефункциональными инфраструктурными I-IP-модулями [3-6], которое гарантирует прозрачность и робастность выполнения функций SoC, технологичность изготовления, надежность и встроенный ремонт изделия в процессе его эксплуатации. Инфраструктура всключает следующие I-IP-модули [5]: 1) наблюдение за состоянием внутренних линий в процессе функционирования и сервисного обслуживания; 2) тестирование функциональных модулей путем подачи проверяющих наборов; 3) диагностирование отказов и дефектов путем анализа информации, полученной на стадии тестирования; 4) восстановление работоспособности функциональных модулей при диагностировании дефектов; 5) измерение характеристик и определение параметров функционирования изделия; 6) надежность и отказоустройчивость функционирования изделия в процессе эксплуатации.

Однако реализация всех компонентов инфраструктуры не всегда оправдана экономически по причине высокой стоимости разработки. Поэтому конкретная реализация компонентов из шестерки упомянутого выше полного множества является функцией от обслуживаемой функциональности и сервисных свойств, необходимых для обеспечения работоспособности SoC. На рис. 1 предложена инфраструктура SoC [3-5], ориентированная на решение задач [7,8]: 1) моделирование одиночных дефектов; 2) построение и хранение таблицы неисправностей; 3) логический и сигнатурный анализ состояний линий; 4) безусловное