Окончание. Начало в № 8'2003

Технология разработки алгоритмически сложных цифровых систем с помощью автоматического синтеза микропрограммных автоматов

**Михаил Долинский,
Игорь Коршунов,
Алексей Толкачев,
Игорь Ермолаев,
Вячеслав Литвинов**

4. Примеры применения

В процессе отладки и апробации разработанной системы синтеза микропрограммных автоматов были написаны микропрограммы и синтезированы схемы для большого количества устройств, некоторые из них коротко описаны ниже. Отметим, что после генерации схемы всегда осуществлялась ее симуляция в HLCCAD/IEESD, генерация синтезируемого VHDL-описания микропрограммного автомата и загрузка его в ПЛИС фирмы Altera. На всех этапах использовались тесты, подготовленные еще для микропрограммы.

4.1. Устройство для кодирования-декодирования кодов Хэмминга

Постановка задачи

Помехоустойчивый код — двоичный код, позволяющий обнаруживать и корректировать ошибки при передаче данных. Код Хэмминга обнаруживает ошибки в двух битах и позволяет корректировать один бит. Контрольные разряды занимают позиции с номерами $i = 2^j$ (j = 0, 1, 2, ...). Значения контрольных разрядов равны сумме разрядов, номера которых больше номера контрольного разряда и таких, что двоичное представление номера содержит единицу в разряде j, начиная с младших разрядов: $B_2 = B_3 + B_6 + B_7 + B_{10} + B_{11}$ (если длина исходного слова 7 бит).

Если слово, полученное после подстановки контрольных разрядов, передано правильно, то выражение, полученное в результате сложения контрольных разрядов и суммы, вычисленной по вышеуказанному правилу, будет равно 0: $S_2 = B_2 + B_3 + B_6 + + B_7 + B_{10} + B_{11}$ (длина исходного слова 7 бит).

Если какой-то разряд исказился, некоторые значения выражений будут равны 1. Пусть исказился B_i, тогда первое значение (S_1) станет равно 1, если двоичное представление i содержит 1 в младшем разряде. Второе значение (S_2), если во втором и т. д. Величина …, S_3, S_2, S_1 — номер искаженного разряда.

Необходимо спроектировать устройство, которое осуществляет кодирование и декодирование кодов Хэмминга в случае слов длиной 7 бит.

Решение задачи

Описание интерфейса проектируемого устройства:

Ic — вход для кодирования (7 бит);

Id — вход для декодирования (11 бит);

Oc — закодированное слово Ic (11 бит);

Od — декодированное слово Id (7 бит).

В случае длины слов 7 бит имеем следующие формулы для вычисления значений контрольных разрядов:

B1 = B3 + B5 + B7 + B9 + B11;

B2 = B3 + B6 + B7 + B10+ B11;

B4 = B5 + B6 + B7;

B8 = B9 + B10+ B11.

Так как процессы кодирования и декодирования должны выполняться параллельно и независимо друг от друга, то целесообразно реализовать эти процессы в виде двух МПА.

Микропрограмма для процесса кодирования:

```
Ic ContIn  7
Oc ContOut 11

start:

    ; Запись информационных разрядов
    mov    Oc[2],   Ic[0]
    mov    Oc[4]:3, Ic[1]:3
    mov    Oc[8]:3, Ic[4]:3

    ; Вычисление контрольного разряда 1 и запись в Oc[0]
    ; B1 = B3 + B5 + B7 + B9 + B11
    xor    Oc[2], Oc[4], s
    xor    s, Oc[6]
    xor    s, Oc[8]
    xor    s, Oc[10], Oc[0]

    ; Вычисление контрольного разряда 2 и запись в Oc[1]
    ; B2 = B3 + B6 + B7 + B10+ B11
    xor    Oc[2], Oc[5], s
    xor    s, Oc[6]
    xor    s, Oc[9]
    xor    s, Oc[10], Oc[1]

    ; Вычисление контрольного разряда 4 и запись в Oc[3]
    ; B4 = B5 + B6 + B7
    xor    Oc[4], Oc[5], s
    xor    s, Oc[6], Oc[3]

    ; Вычисление контрольного разряда 8 и запись в Oc[7]
    ; B8 = B9 + B10+ B11
    xor    Oc[8], Oc[9], s
    xor    s, Oc[10], Oc[7]

    jmp start
```

Микропрограмма для процесса декодирования:

```
Id  ContIn  11
Od  ContOut 7

tmp reg    7

s  flag
s1  flag
s2  flag
s3  flag
s4  flag

start:

        ;Сброс сигналов об ошибках
        clr     s1
        clr     s2
        clr     s3
        clr     s4

        ;Проверка контрольного разряда 1
        ; S = B2 + B3 + B5 + B7 + B9 + B11
        xor     Id[0], Id[2], s
        xor     s, Id[4]
        xor     s, Id[6]
        xor     s, Id[8]
        je      s, Id[10], E1
        ;Установка сигнала о ошибке
        set     s1

        ;Проверка контрольного разряда 2
E1:     ; S = B2 + B3 + B6 + B7 + B10+ B11
        xor     Id[1], Id[2], s
        xor     s, Id[5]
        xor     s, Id[6]
        xor     s, Id[9]
        je      s, Id[10], E2
        ;Установка сигнала о ошибке
        set     s2

        ;Проверка контрольного разряда 4
E2:     ; S = B4 + B5 + B6 + B7
        xor     Id[3], Id[4], s
        xor     s, Id[5]
        je      s, Id[6], E4
        ;Установка сигнала о ошибке
        set     s3

        ;Проверка контрольного разряда 8
E4:     ; S = B8 + B9 + B10+ B11
        mov     s, Id[7]
        xor     Id[7], Id[8], s
        xor     s, Id[9]
        je      s, Id[10], E8
        ;Установка сигнала о ошибке
        set     s4

        ;Формирование декодированного слова
E8:     mov     tmp[0],   Id[2]
        mov     tmp[1]:3, Id[4]:3
        mov     tmp[4]:3, Id[8]:3

        ;Поиск разряда в котором произошла ошибка
        jnz     s4, m1
        jnz     s3, m01
        jnz     s2, m001
        jmp     finish  ;нет ошибок
m001:   jz      s1, finish ;ошибка контрольного разряда
        ; Ошибка в разряде 0011 (бит 0)
        not     tmp[0]  ;инверсия ошибочного разряда
        jmp     finish
m01:    jz      s2, m010
        jz      s1, m0110
        ; Ошибка в разряде 0111 (бит 3)
        not     tmp[3]  ;инверсия ошибочного разряда
        jmp     finish
m0110:  ; Ошибка в разряде 0110 (бит 2)
        not     tmp[2]  ;инверсия ошибочного разряда
        jmp     finish
m010:   jz      s1, finish ;ошибка контрольного разряда
        ; Ошибка в разряде 0101 (бит 1)
        not     tmp[1]  ;инверсия ошибочного разряда
        jmp     finish
m1:     jz      s2, m100
        jz      s1, m1010
        ; Ошибка в разряде 1011 (бит 6)
        not     tmp[6]  ;инверсия ошибочного разряда
        jmp     finish
m1010:  ; Ошибка в разряде 1010 (бит 5)
        not     tmp[5]  ;инверсия ошибочного разряда
        jmp     finish
m100:   jz      s1, finish ;ошибка контрольного разряда
        ; Ошибка в разряде 1001 (бит 4)
        not     tmp[4]  ;инверсия ошибочного разряда

        ;Вывод декодированного слова на выходные контакты
finish:
        mov   Od, tmp

        jmp start
```

4.2. Устройство для распознавания кода Морзе

Постановка задачи

Необходимо спроектировать устройство, декодирующее текстовую информацию из сигналов азбуки Морзе. Каждая буква представляется в виде последовательности из точек и тире:

A .-	J .---	S ...
B -...	K -.-	T -
C -.-.	L .-..	U ..-
D -..	M --	V ...-
E .	N -.	W .--
F ..-.	O ---	X -..-
G --.	P .--.	Y -.--
H	Q --.-	Z --..
I ..	R .-.	

Характеристики сигналов:

- Сигналы в виде «точек» и «тире» поступают на вход DI. Значение сигнала DI считывается по переднему фронту сигнала Clk.
- Сигналы «точка» и «тире» разделяются паузами.
- Сигнал «точка» соответствует одной «1» между двумя паузами.
- Сигнал «тире» соответствует трем «1» между двумя паузами.
- Пауза соответствует значению «0» на входе DI.
- Сигналы одной и той же буквы разделяются одной паузой.
- Между двумя буквами три паузы. На выходе Code необходимо устанавливать порядковый номер символа («A» = 0, «B» = 1, т. д.) в промежутке между межбуквенными паузами.

DI — сигналы азбуки Морзе.

Clk — синхронизирующий сигнал (DI).

Reset — асинхронный вход инициализации («1»).

Code — код декодированного символа.

Ready — признак готовности символа («0» — готов, «1» — нет).

Решение задачи

Микропрограмма:

```
DI      ContIn  1
dClk    ContIn  1
oCode ContOut 5
Ready  ContOut 1

ax      flag
pause   reg 2

init:       clr Ready
start:
rep:        call GetInfo
            je      pause, 3, rep
            clr Ready
            jz ax, Get0

            call GetInfo
            jeq     pause, 3, OUT_T
            jz ax, Get10

            call    GetInfo
            jeq     pause, 3, OUT_M
            jnz     ax, OUT_O

            call    GetInfo
            jeq     pause, 3, OUT_G
            jz      ax, OUT_Z
            jmp     OUT_Q

Get10:
            call GetInfo
            jeq pause, 3, OUT_N
            jz ax, Get100

            call GetInfo
            jeq pause, 3, OUT_K
            jz ax, OUT_C
            jmp OUT_Y

Get100:
            call GetInfo
            jeq pause, 3, OUT_D
            jz ax, OUT_B
            jmp OUT_X

Get0:
            call GetInfo
            jeq pause, 3, OUT_E
            jz ax, Get00

            call GetInfo
            jeq pause, 3, OUT_A
            jz ax, Get010

            call GetInfo
            jeq pause, 3, OUT_W
            jz ax, OUT_P
            jmp OUT_J

Get010:
            call GetInfo
            jeq pause, 3, OUT_R
            jz ax, OUT_L
            jmp finish

Get00:
            call GetInfo
            jeq pause, 3, OUT_I
            jz ax, Get000

            call GetInfo
            jeq pause, 3, OUT_U
            jz ax, OUT_F
            jmp finish

Get000:
            call GetInfo
            jeq pause, 3, OUT_S
            jz ax, OUT_H
            jmp OUT_V

OUT_A:
            mov oCode, 0
            jmp finish
OUT_B:
            mov oCode, 1
            jmp finish
OUT_C:
            mov oCode, 2
            jmp finish
OUT_D:
            mov oCode, 3
            jmp finish
OUT_E:
            mov oCode, 4
            jmp finish
OUT_F:
            mov oCode, 5
            jmp finish
OUT_G:
            mov oCode, 6
            jmp finish
OUT_H:
            mov oCode, 7
            jmp finish
OUT_I:
            mov oCode, 8
            jmp finish
OUT_J:
            mov oCode, 9
            jmp finish
OUT_K:
            mov oCode, 10
            jmp finish
OUT_L:
            mov oCode, 11
            jmp finish
OUT_M:
            mov oCode, 12
            jmp finish
OUT_N:
            mov oCode, 13
            jmp finish
OUT_O:
            mov oCode, 14
            jmp finish
OUT_P:
            mov oCode, 15
            jmp finish
OUT_Q:
            mov oCode, 16
            jmp finish
OUT_R:
            mov oCode, 17
            jmp finish
OUT_S:
            mov oCode, 18
            jmp finish
OUT_T:
            mov oCode, 19
            jmp finish
```

```
OUT_U:
            mov oCode, 20
            jmp finish
OUT_V:
            mov oCode, 21
            jmp finish
OUT_W:
            mov oCode, 22
            jmp finish
OUT_X:
            mov oCode, 23
            jmp finish
OUT_Y:
            mov oCode, 24
            jmp finish
OUT_Z:
            mov oCode, 25
            jmp finish

finish:     set     Ready
            jmp start


;Подпрограмма обработки входной информации
;Выходные значения
; pause — количество пауз
; ax   — 1 — «точка», 0 — «тире»
GetInfo proc

            set pause
IncP:       inc pause
            jeq pause, 3, blank
Clk0:       jnz dClk, Clk0
Clk1:       jz  dClk, Clk1
            jz  DI, IncP

Clk0_1:     jnz dClk, Clk0_1
Clk1_1:     jz  dClk, Clk1_1
            jnz  DI, Clk0_2
            set ax
            ret

Clk0_2:     jnz dClk, Clk0_2
Clk1_2:     jz  dClk, Clk1_2
            clr ax
blank:      ret

endp
```

4.3. Управляемый делитель частоты

Постановка задачи

На вход A подается меандр (периодический сигнал с равными длительностями 0 и 1). На вход B — число — коэффициент деления частоты. На выходе C нужно получить меандр с частотой A/B. На стабилизацию отводится один период выходной частоты, после чего должен следовать нулевой полупериод. Если B = 0, на C всегда должен быть 0.

Решение задачи

Микропрограмма:

```
  a     contIn   1
  b     contIn   8
  c     contOut  1
  cur   reg      8

start:
        clr     c
        ; ожидание на b сигнала отличного от нуля
l0:     jz      b, l0
        mov     cur, b
        ; ожидание переднего фронта
l1:     jz      a, l1
        ; ожидание заднего фронта
l2:     jnz     a, l2
        dec     cur
        ; пропускаем первые b периодов
        jnz     cur, l1
        ; ожидание переднего фронта
l3:     jz      a, l3
        ; инвертируем выходной сигнал
        not     c
        jz      b, start
        jmp     l0
```

4.4. Стек

Постановка задачи

Требуется спроектировать 8-разрядный стек с глубиной до 255 элементов, который позволяет:

- положить элемент в стек;
- взять значение вершины стека;
- убрать элемент из стека;
- узнать количество элементов в стеке;
- контролировать ошибки (попытка взять элемент из пустого стека);
- сбросить состояние стека и признак ошибки.

Решение задачи

Описание интерфейса проектируемого стека:

Input — значение элемента;

Output — значение вершины стека;

Count — количество элементов (для пустого стека — 0);

Reset — сброс стека (сброс при 1);

Push — положить элемент с Input в стек (по переднему фронту);

Pop — убрать элемент со стека (по переднему фронту);

Error — признак ошибки (1 — ошибка).

Стек представляет собой регистр длиной 256×8 = 2048. Вершина стека — биты 0–7. По команде Push все биты регистра сдвигаются влево на 8 позиций и на освободившееся место записывается новый элемент. По команде Pop все биты регистра сдвигаются вправо на 8 позиций.

Микропрограмма, реализующая стек:

```
Input       ContIn      8
Push        ContIn      1
Pop         ContIn      1

Output      ContOut     8
Count       ContOut     8
Error       ContOut     1

Stack       reg         2048

start:
            clr         Count
InitPop:
            jnz         Pop, InitPop
InitPush:
            jnz         Push, InitPush
Wait:
            jnz         Push, PushCmd
            jz          Pop, Wait
                                    ; Pop
            clr         Error
            jz          Count, StackEmpty
            shr         Stack, 8
            mov         Output, Stack[0]:8
            dec         Count

            jmp         InitPop
                                    ; Push
PushCmd:
            clr         Error
            inc         Count
            jc          StackFull
            shl         Stack, 8
            mov         Stack[0]:8, Input
            mov         Output, Input
            jmp         InitPop
                                    ; Full
StackFull:
            mov         Count, 255
                                    ; Empty
StackEmpty:
            set         Error
            jmp         InitPop
```

4.5. Устройство для проверки корректности штрих-кода

Постановка задачи

Требуется спроектировать устройство для проверки 13-разрядного штрих-кода системы EAN-13.

Пусть дан штрих-код «4601546003119». Для проверки правильности штрих-кода существует следующий алгоритм:

1. Складываются цифры, стоящие на четных местах штрих-кода 6+1+4+0+3+1=15.
2. Полученная сумма умножается на три 15×3=45.
3. Складываются цифры, стоящие на нечетных местах штрих-кода (кроме последней контрольной цифры) 4+0+5+6+0+1=16.
4. Полученные два числа складываются 45+16=61.
5. Отбрасываются десятки 61–60=1.
6. Полученное в пункте 5 число вычитается из десяти 10–1=9 (если в 5 пункте получили 0 — результат 0). Результат должен совпадать с контрольной цифрой (13-й цифрой). Если это не так, то указанный «штрих-код» — грубая подделка, а качество товара скорей всего невысоко.

Примеры

«4601546003119» — штрих-код верен.

«4601546003118» — штрих-код неверен.

Решение задачи

Описание интерфейса проектируемого стека:

Code1-Code12 — первые 12 цифр штрих-кода;

Code13 — 13-я цифра штрих-кода (контрольная);

Error — некорректные входные данные (1 — данные некорректны);

Status — результат проверки (1 — штрих-код неверен или некорректные входные данные).

Микропрограмма:

```
Code1   ContIn  4
Code2   ContIn  4
Code3   ContIn  4
Code4   ContIn  4
Code5   ContIn  4
Code6   ContIn  4
Code7   ContIn  4
Code8   ContIn  4
Code9   ContIn  4
Code10  ContIn  4
Code11  ContIn  4
Code12  ContIn  4
Code13  ContIn  4

Status  ContOut 1
Error   ContOut 1

sum1    reg     8
sum     reg     8
tmp     reg     8

Start:
        ; Проверка корректности входных данных
        ja      Code1, 9, not_digit
        ja      Code2, 9, not_digit
        ja      Code3, 9, not_digit
        ja      Code4, 9, not_digit
        ja      Code5, 9, not_digit
        ja      Code6, 9, not_digit
        ja      Code7, 9, not_digit
        ja      Code8, 9, not_digit
        ja      Code9, 9, not_digit
        ja      Code10, 9, not_digit
        ja      Code11, 9, not_digit
        ja      Code12, 9, not_digit
        ja      Code13, 9, not_digit
        clr     Error

        ; Проверка корректности штрих-кода
        clr     tmp
        clr     sum
        mov     tmp[0] : 4, Code2
        add     sum, tmp
        mov     tmp[0] : 4, Code4
        add     sum, tmp
        mov     tmp[0] : 4, Code6
        add     sum, tmp
        mov     tmp[0] : 4, Code8
        add     sum, tmp
        mov     tmp[0] : 4, Code10
        add     sum, tmp
        mov     tmp[0] : 4, Code12
        add     sum, tmp
        add     sum, sum, sum1
        add     sum1, sum
        clr     sum
        mov     tmp[0] : 4, Code1
        add     sum, tmp
        mov     tmp[0] : 4, Code3
        add     sum, tmp
        mov     tmp[0] : 4, Code5
        add     sum, tmp
        mov     tmp[0] : 4, Code7
        add     sum, tmp
        mov     tmp[0] : 4, Code9
        add     sum, tmp
        mov     tmp[0] : 4, Code11
        add     sum, tmp
        add     sum, sum1
```

```
for:      ja      10, sum, done
          sub     sum, 10
          jmp     for
done:     jz      sum, except
          sub     10, sum, sum
except:   jeq     sum[0] : 4, Code13, set_s
          clr     Status
          jmp     start
set_s:    set     Status
          jmp     start
not_digit:
          set     Error
          clr     Status
          jmp     start
```

5. Отладка микропрограмм, настраиваемая на исполнительные устройства

Нам представляется, что в практике разработок вполне реальной является следующая ситуация. Имеется набор исполнительных устройств — независимо функционирующих компонентов операционного автомата или, в общем случае, независимо функционирующих операционных автоматов. Создание требуемой цифровой системы фактически заключается в написании и отладке микропрограммы, реализующей требуемый комплекс функций. Какие способы решения подобных проблем предлагаются нами?

При наличии определенных пользователем (или заданных ему техническим заданием) операционных устройств и формата микрокоманды можно реализовать модели имеющихся операционных устройств с помощью языка микропрограммных автоматов: блоки памяти операционных устройств определяем с помощью команд резервирования памяти REG и FLAG, а функции операционных устройств реализуем как последовательности микрокоманд на ассемблере MPA, объединенные в микропрограммы или макрокоманды, с определяемыми пользователями именами. Далее пользователи могут отлаживать микропрограммы, управляющие имеющимися операционными устройствами. По завершению отладки всех микропрограмм (или параллельно их отладке) пользователь может разработать генератор реальных машинных кодов для отлаживаемых подпрограмм. Эта задача может быть сильно упрощена при разумном вводе пользователем имен макрокоманд и подпрограмм, обеспечивающем табличное кодирование мнемоник в соответствующие поля микрокоманд. Кроме того, для ускорения создания генератора машинных кодов микропрограмм можно использовать разработанный нами ассемблер, настраиваемый на целевую архитектуру RtASM [8].

Для отладки генератора микропрограмм можно использовать дизассемблер и симуляцию дизассемблированных кодов с целью достижения эквивалентности результатов симуляции исходного текста и его ассемблированного (дизассемблированного) представления.

6. Использования в учебном процессе

Описанный в данной работе программный комплекс активно используется в учебном процессе математического факультета Гомельского государственного университета (Белоруссия). Ниже приведены примеры лабораторных и курсовых работ.

1. Разработка микропрограмм алгоритмически сложных цифровых устройств.
2. Исследования прикладных алгоритмов на выделение новых инструкций для расширения языка микропрограммных автоматов (МПА).
3. Анализ эффективности аппаратной реализации инструкций (операционных устройств) языка МПА.

7. Направления текущих разработок

Разработанная система автоматического синтеза микропрограммных автоматов продолжает развиваться в следующих направлениях:

- поддержка языка программирования C как средства описания алгоритмов функционирования устройств;
- генераторы типизированных параметрических систем;
- управляемая оптимизация генерируемых МПА.

Очевидно, что использование языка более высокого уровня, чем ассемблер, будет способствовать повышению производительности разработки микропрограмм. Для реализации этого направления в настоящее время разрабатываются язык CMPDL, в основе которого лежат конструкции языка C, которые могут быть скомпилированы в конструкции уже реализованного языка MPDL, в том числе арифметические и логические выражения, операторы условия, выбора, циклов, безусловного перехода и некоторые другие. Одновременно разрабатывается компилятор CMPDL->MPDL на базе созданного нами универсального синтаксического анализатора UniSAn.

Дальнейшим развитием такого подхода к повышению производительности разработчиков сложных алгоритмических устройств может быть создание визуальных редакторов для специализированных устройств, например, фильтров, шифраторов-дешифраторов, компрессоров-декомпрессоров, аудио- и видеокодеков и т. д. Разработчик в визуальном редакторе определяет параметры нужного ему устройства и получает исходный текст микропрограммы на CMPDL или MPDL, из которой по описанной выше технологии строится синтезируемое VHDL-описание микропрограммного автомата, реализующего требуемые функции.

Третьим важным направлением развития является анализ и совершенствование генерируемых по алгоритмам схем с целью увеличения количества параметров генерации и передачи в руки пользователей возможностей анализировать и выбирать для генерируемой схемы компромиссы между производительностью, аппаратными затратами, потребляемой мощностью, стоимостью и т. д.

Заключение

Описанные метод и средства разработки алгоритмически сложных устройств на основе автоматической генарации схем по отлаженным микропрограммам существенно сокращают сроки разработки, в то же время обеспечивая высокое быстродействие и приемлемое качество (в смысле аппаратных затрат) созданных схем. К сожалению, ввиду отстутствия реальных заказов, апробация и испытания в настоящее время проводились только на учебных примерах. Хочется надеяться, что настоящая публикация поможет нам найти разработчиков, решающих задачи соответствующей сложности.

Литература

1. Dolinsky M. S., Ziselman I. M., Harrasov A. A. Computer-Aided design of microprogrammed devices // Automatic Control and Computer Sciences. New York: Allerton Press. 1997. Vol. 31. № 5.
2. Долинский М. С., Зисельман И. М., Харрасов А. Р. Автоматический синтез микропрограммных автоматов // Автоматика и вычислительная техника. Рига. 1997. № 5.
3. Долинский М. С., Харрасов А. А. Средства разработки цифровых устройств методом синтеза микропрограммных автоматов // Электоника. Минск. 1998. № 11–12.
4. Коршунов И. В. Автоматический синтез микропрограммных автоматов // Новые математические методы и компьютерные технологии в проектировании, производстве и научных исследованиях. Материалы III Республиканской научно-технической конференции студентов и аспирантов 13–18 марта 2000 г. Гомель.
5. Коршунов И. В. Метод автоматического построения микропрограммных автоматов // Новые математические методы и компьютерные технологии в проектировании, производстве и научных исследованиях. Материалы IV Республиканской научно-технической конференции студентов и аспирантов 19–22 марта 2001 г. Гомель.
6. Долинский М., Литвинов В., Галатин А., Ермолаев И. HLCCAD — среда редактирования, симуляции и отладки аппаратного обеспечения. Компоненты и Технологии. 2003. № 1.
7. Долинский М., Ермолаев И., Толкачев А., Гончаренко И. WInter — среда отладки программного обеспечения мультипроцессорных систем. Компоненты и Технологии. 2003. № 2.
8. ht tp: //NewIT.gsu.unibel.by.
9. Баранов С. И. Синтез микропрограммных автоматов. Л.: Машиностроение. 1979.
10. Майоров С. А., Новиков Г. И. Структура электронных вычислительных машин. Л.: Машиностроение. 1979.
11. Каган Б. М. Электронные вычислительные машины и системы. М.: Энергия. 1979.
12. Семенов Н., Каршенбойм И. Микропрограммные автоматы на базе специализированных ИС // Chip news. 2000. № 7.
13. Лобанов В. Технический минимум пользователя САПР MAX+PLUS II // Chip news. 2001. № 1.
14. Лазарев В. Г., Пийль Е. И. Синтез управляющих автоматов. М.: Энергоатомиздат. 1989.
15. Tolkachev A., Dolinsky M. Hardware Implementation of Complex Data Processing Algorithms. EUROMICRO/DSD-2003.