KiT#24(7)_Olia.qxd 2/6/03 3:32 PM Page 62

Окончание. Начало в №3

Микропроцессор своими руками

Иосиф Каршенбойм

lk@lmail.loniis.spb.su

Начинаем проект в FPGA

При разработке и создании все более сложных и объемных цифровых схем появляется проблема повышения эффективности работы разработчиков и гибкости проекта, а основным требуемым критерием эффективности разработки является время, затраченное на разработку, или, иначе говоря, «time to market» — время от начала реализации проекта до выхода готового продукта на рынок. Поэтому, проект строится в виде иерархической пирамиды файлов. При таком подходе каждый «кирпич» пирамиды проектируется и отлаживается отдельно, что позволяет в дальнейшем пользоваться набором проверенных блоков, называемых библиотечными. Проект будем выполнять на языке AlteraHDL, так как он достаточно широко распространен и описан. Кроме того, для начинающих, он гораздо более доступен, чем другие языки группы VHDL. Так же более доступны и студенческие версии ПО.

Проект будет выполнен в виде текстовых файлов. Это позволит применить параметрическое выполнение описаний, что даст возможность применить файлы и в других проектах. В пакете ПО MaxPlus фирма Altera дает библиотеку параметризуемых мегафункций, позволяющих легко встраивать в проект счетчики, дешифраторы и пр. Описание этих мегафункций приведено в Help-файлах ПО.

Соглашение о названии сигналов (о наименовании цепей)

Для однообразия примем, что активное состояние сигналов будет равно 1. Если активное состояние сигнала будет равно 0, то в название сигнала на конце будет добавлена буква «*n*».

В файлах описаний применим следующие названия сигналов:

clk — частота, синхрочастота,
_ena — сигнал разрешения записи, в большинстве случаев его длительность будет равна длительности 1-го периода синхрочастоты,
reset — сигнал системного сброса.
_load — сигнал загрузки,
_node — название сигнала относится к внутреннему узлу схемы,
_rg — название сигнала относится к внутреннему регистру
_cnt — название сигнала относится к внутреннему счетчику.

Все сигналы внутри FPGA Altera могут быть только однонаправленными, поэтому у каждого блока обычно есть входы и выходы данных, поэтому

d[], data_a[], data_b[] — эти и другие подобные сигналы будут являться входными шинами данных для разрабатываемых блоков,
q[] — эти и другие подобные сигналы будут являться выходными шинами данных для разрабатываемых блоков (если это не оговорено отдельно).

Начинаем с разработки вспомогательных файлов.

Произведем разработку вспомогательных файлов. Начнем с регистра из D-триггеров.

Далее приведен текст файла Dffex_rg.tdf. Это регистр со входами сброса, установки и разрешения записи. Разрядность регистра задается в параметрах и, по умолчанию, принята равной 8 бит.

```
-- Version 1.0
-- Copyright Iosif Karshenboim, 2000
-- You may use or distribute this function freely,
-- provided you do not remove this copyright notice.
-- If you have questions or comments, feel free to
-- contact me by email at ik@lmail.loniis.spb.su

PARAMETERS
( m = 7, l = 0 );

SUBDESIGN Dffex_rg

( d[m..l]    : INPUT;
 clk         : INPUT;
 resn        : INPUT = VCC ;
 setn        : INPUT = VCC ;
 ena         : INPUT = GND ;
 q[m..l]     : OUTPUT; )

BEGIN
        FOR i IN l to m GENERATE
                q[i] = DFFE(d[i], clk, resn, setn, ena );
        END GENERATE;
END;
```

Выберем в качестве устройства для проекта микросхемы серии ACEX или FLEX10, а далее выбор конкретного устройства установим в режим AUTO.

Создайте рабочую папку, например My_cpu, запишите файл в эту папку, затем выполните компиляцию, чтобы убедиться в отсутствии ошибок, и создайте инклюдный файл Dffex_rg.inc. Так же будем поступать и со всеми другими файлами проекта о которых будет говориться здесь.

Далее создедим еще один вспомогательный файл — mdffex_rg.tdf. Это регистр, аналогичный предыдущему, со входами сброса, установки и разрешения записи. Здесь уже используется предыдущий файл, как основа для построения данного фай-

KiT#24(7)_Olia.qxd 2/6/03 3:32 PM Page 63

ла. От предыдущего этот регистр отличается тем, что имеет на входе мультиплексор данных. Сигнал sel выбирает какую шину data_a[DATA_WIDTH-1..0] или data_b[DATA_WIDTH-1..0] подключить ко входу регистра. Разрядность регистра задается в параметрах и, по умолчанию, принята равной 16 бит.

```
-- Version 1.0
-- Copyright Iosif Karshenboim, 2000
-- You may use or distribute this function freely,
-- provided you do not remove this copyright notice.
-- If you have questions or comments, feel free to
-- contact me by email at ik@lmail.loniis.spb.su

INCLUDE «dffex_rg.inc»;
PARAMETERS
(-- Параметры
 DATA_WIDTH = 16        -- разрядность шины данных
 );

SUBDESIGN mdffex_rg
( data_a[DATA_WIDTH-1..0],
 data_b[DATA_WIDTH-1..0],
 sel, clk, ena, reset                 : INPUT = GND ;
 q[DATA_WIDTH-1..0]                   : OUTPUT; )
BEGIN
 q[DATA_WIDTH-1..0] = dffex_rg (( !sel & data_a[DATA_WIDTH-1..0])
            # ( sel & data_b[DATA_WIDTH-1..0]), clk, !reset, , ena )
            WITH ( m = DATA_WIDTH-1, l = 0 ) ;
END;
```

Теперь создадим файл, описывающий стек.

Стек будет представлять набор регистров с мультиплексорами на входах. Регистры будут соединены каскадно. При этом, на входы регистра, через мультиплексор, начиная с 1-го, подаются выходы от регистра с меньшим номером или от регистра с большим номером. На вход 0-го регистра подается входная шина данных, а его выходы соединены с выходной шиной данных стека. Если на входах управления режимом работы блока — push и pop нет активного сигнала, то сохраняется предыдущее состояние. Если активен сигнал push, то в 0-й регистр записываются данные со входной шины данных, в остальные регистры записываются данные от «соседа» с меньшим номером и, следовательно, данные «проталкиваются» в стек, а если активен сигнал pop, то в регистр записываются данные от «соседа» с большим номером и данные извлекаются из стека.

Наш стек будет иметь параметры как по разрядности шины данных, так и по глубине. Такое выполнение файла позволит, при необходимости, легко модернизировать микропроцессор.

```
-- Version 1.0
-- Copyright Iosif Karshenboim, 09.06.2000
-- You may use or distribute this function freely,
-- provided you do not remove this copyright notice.
-- If you have questions or comments, feel free to
-- contact me by email at ik@lmail.loniis.spb.su

INCLUDE «mdffex_rg.inc»;

PARAMETERS
(-- Параметры
 STACK_WIDTH = 4,     -- разрядность шины адресов памяти команд в словах «CPU_WIDTH»
 DATA_WIDTH = 24      -- разрядность шины данных памяти команд
 );

SUBDESIGN stack
( data[DATA_WIDTH-1..0],
 push, pop, clk                        : INPUT = GND ;
 q[DATA_WIDTH-1..0]                    : OUTPUT; )

VARIABLE
 s[DATA_WIDTH-1..0][STACK_WIDTH-1..0]        : NODE;

BEGIN
 s[DATA_WIDTH-1..0][0] = mdffex_rg
        (.data_a[(data_width) — (1)..0] = data[DATA_WIDTH-1..0] ,
         .data_b[(data_width) — (1)..0] = s[DATA_WIDTH-1..0][1] ,
         .sel = pop , .clk = clk , .ena = push # pop )
         WITH ( DATA_WIDTH = DATA_WIDTH );

FOR i IN 1 to ( STACK_WIDTH — 2 ) GENERATE
   s[DATA_WIDTH-1..0][i] = mdffex_rg ( .data_a[(data_width) —
(1)..0] = s[DATA_WIDTH-1..0][i-1] ,
        .data_b[(data_width) — (1)..0] = s[DATA_WIDTH-1..0][i+1] ,
        .sel = pop , .clk = clk , .ena = push # pop )
        WITH ( DATA_WIDTH = DATA_WIDTH );
END GENERATE;

s[DATA_WIDTH-1..0][STACK_WIDTH-1] = mdffex_rg ( .data_a[(data_width) — (1)..0] = s[DATA_WIDTH-1..0][STACK_WIDTH-2] ,
        .data_b[(data_width) — (1)..0] = GND ,
        .sel = pop , .clk = clk , .ena = push # pop )
        WITH ( DATA_WIDTH = DATA_WIDTH );
q[DATA_WIDTH-1..0] = s[DATA_WIDTH-1..0][0] ;

END;
```

Основа стека — это двухмерный массив регистров с мультиплексорами s[DATA_WIDTH-1..0][STACK_WIDTH-1..0], обозначенный в файле как узел — NODE. Массив имеет ширину — DATA_WIDTH и глубину — STACK_WIDTH.

Для проверки работы стека был создан файл, аналогичный приведенному файлу stack, но для удобства проверки работы, выходы регистров, образующих стек были выведены как выходные шины — q0_[] …q3_[].

Симуляция работы блока stack приведена на рис. 2.

Сначала выполняется команда push, данные помещаются в стек, затем подается команда pop и данные извлекаются из стека.

Создадим файл, описывающий счетчик команд.

Нам нужен счетчик, который мог бы загружаться при выполнении команд JMP, CALL, RET. Значение адреса, которым будет загружаться счетчик, подадим по входной шине данных. Счетчик должен считать «вверх». Счетчик также должен иметь входной сигнал разрешения счета, так как для некоторых операций счет будет запрещен. Для счетчика команд применим библиотечный элемент — lpm_counter. Зададим параметр по разрядности шины адресов памяти команд — PS_WIDTHAD.

```
INCLUDE «lpm_counter.inc»;

PARAMETERS
( PS_WIDTHAD = 7);      -- разрядность шины адресов памяти команд

SUBDESIGN ps_cnt
(
clk, reset,
data_in[PS_WIDTHAD -1..0],
ps_cnt_load,
ps_cnt_ena                         : INPUT;
ps_addr[PS_WIDTHAD-1..0]           : OUTPUT;
)
VARIABLE
ps_cnt      : lpm_counter WITH (lpm_width = PS_WIDTHAD);

BEGIN
-- Счетчик адресов памяти команд «PS»
      ps_cnt.clock = clk;
      ps_cnt.aclr = reset;
      ps_cnt.data[PS_WIDTHAD-1..0] = data_in[PS_WIDTHAD -1..0];
-- программная установка
      ps_cnt.sload = ps_cnt_load;
      ps_cnt.cnt_en = ps_cnt_ena;
      ps_addr[PS_WIDTHAD-1..0] = ps_cnt.q[PS_WIDTHAD -1..0];

END ;
```

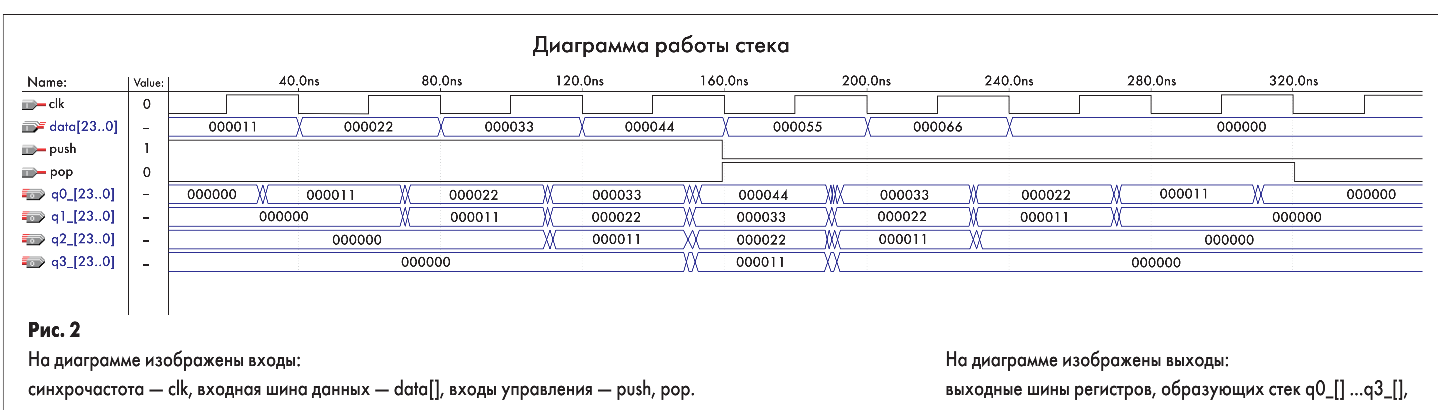

Рис. 2

На диаграмме изображены входы: синхрочастота — clk, входная шина данных — data[], входы управления — push, pop.

На диаграмме изображены выходы: выходные шины регистров, образующих стек q0_[] …q3_[],

KiT#24(7)_Olia.qxd 2/6/03 3:32 PM Page 64

Счетчик имеет входную шину данных, входы синхронной загрузки и разрешения счета. Выходы счетчика подключены к выходной шине блока. По умолчанию такой счетчик будет считать «вверх». Итак, счетчик адресов удовлетворяет поставленным требованиям.

Создадим файл, описывающий ALU — основу нашего микропроцессора.

Прежде чем приступить к описанию ALU, позволю себе небольшое отступление. Когда в известной басне слепых спрашивали, какой им представляется слон, то каждый из них давал свое описание. Так же происходит и с микропроцессорами. На вопрос, как и почему микропроцессоры работают, дилер из коммерческой фирмы не задумываясь ответит: «Потому что они импортные!». Программист начнет рассказывать об операционных системах, языках программирования, оптимизирующих компиляторах и т.д. Если же сейчас спросить нас с Вами в этой части статьи, то мы должны ответить: «Микропроцессор работает только потому, что есть генератор тактовой частоты, счетчик адресов памяти, сама память, кое-какие регистры, называемые «порты», а все остальное — дешифратор. Да и память, собственно, это тоже большой дешифратор». Так вот — лучшая часть этого дешифратора и называется ALU!

Наше ALU будет состоять из двух «логических» частей: первая будет вырабатывать сигналы управления для всего, что есть в данном проекте, а вторая будет коммутировать выходные шины блоков на общую внутреннюю шину микропроцессора.

Самый верхний приоритет в исполнении команд имеет вход запроса прерывания, и, если есть состояние IRQ=1, то необходимо выполнить переход по адресу вектора и сохранить адрес возврата в стеке. Если данную команду представить в виде элементарных команд, то она эквивалентна двум командам JMP <addr> и PUSH.

Если нет запроса прерывания, т.е. IRQ=0, то необходимо продешифрировать слово команды, поступившее с выходов блока PS. Биты [23..20] дадут поле кода операции, биты [19..16] — укажут на регистр — приемник информации, биты [3..0] — укажут на регистр — источник информации, биты [15..0] — укажут на адрес поля Mem.

```
TITLE «CPU ALU» ;

-- Version 1.0
-- Copyright Iosif Karshenboim, 2002
-- You may use or distribute this function freely,
-- provided you do not remove this copyright notice.
-- If you have questions or comments, feel free to
-- contact me by email at ik@lmail.loniis.spb.su

-- Имеет память команд — PS, память данных — DS, отдельные
шины адресов команд и данных.
-- Стек аппаратный на регистрах только для адреса возврата, нет
команд PUSH и POP

%
Код операции 4 бита, адрес регистра или условие перехода 4 би-
та, константа или данные 16 бит.

-- Описание команд
0 — NOP
1 — JMP < Y >          – переход без условий
2 — CALL
3 — RET                – возврат
8 — MOV Reg, Reg       – загрузка в регистр-приемник данных
из регистра-передатчика
9 — MOV Reg, [Mem]     – загрузка в регистр-приемник данных
из памяти
                       – по непосредственному адресу
A — MOV [Mem], Reg     – загрузка в память по непосредственно-
му адресу данных
                       – из регистра-передатчика
B — LDI Reg < C >      – непосредственная загрузка в регистр-
приемник данных из
                       – памяти команд по текущему адресу

Регистры имеют номера 0 — 15,
для записи в них есть сигналы wr_reg_ena[15..0],
регистр 0 — аккумулятор
регистр 1 — регистр ввода–вывода
регистр 2 — регистр ввода–вывода
регистр 3 — регистр ввода–вывода

%

PARAMETERS
(
-- Параметры «ALU», задаваемые по умолчанию
 PS_WIDTHAD = 7,        -- разрядность шины адресов памяти
команд в словах «CPU_WIDTH»
 DS_WIDTHAD = 7,        -- разрядность шины адресов памяти
данных в словах «CPU_WIDTH»
 PS_WIDTH = 24,   -- разрядность шины данных памяти команд
 DS_WIDTH = 16,   -- разрядность шины данных памяти команд
-------------------------------------------------------------------
 STACK_WIDTH = 8,
 STACK_DATA_WIDTH = 7,
-------------------------------------------------------------------
 IRQ_VECTOR = H»0»      -- адрес вектора запроса прерывания
-------------------------------------------------------------------
);
--//////////////////////////////////////////////////////////////////
CONSTANT VECTOR = IRQ_VECTOR;
--//////////////////////////////////////////////////////////////////
SUBDESIGN alu
(
-- «ALU»
-- выход от «ALU»
data_out[15..0],
ps_cnt_data[PS_WIDTHAD-1..0], -- данные для загрузки в счет-
чик команд
wr_reg_ena[1..0],
ps_cnt_load,
push,
pop,
ds_wr_ena   : OUTPUT ;

irq,      -- сигнал запроса прерываний для «ALU», действует 1 clk
-- выходы блоков — входы для АЛУ
data_in[23..0],
ds_q[15..0],
rg0_q[15..0],                         -- выходы регистров
rg1_q[15..0],
stack_q[PS_WIDTHAD-1..0],

ps_addr[PS_WIDTHAD — 1..0]      -- выходы счетчика адресов
                               : INPUT = GND ;

)
VARIABLE

ps_cnt_data_node[PS_WIDTHAD-1..0] : NODE ;
--==========================================
BEGIN
--==========================================

--//////////////////////////////////////////////////////////
-- ЗДЕСЬ НАЧАЛО
-- сначала выполняем обработку прерываний,
-- потом обрабатываеи операнд
--//////////////////////////////////////////////////////////

--//////////////////////////////////////////////////////////
-- Обработка прерываний
--//////////////////////////////////////////////////////////
IF irq THEN
-- абсолютный переход без условий по адресу вектора запроса
прерывания
 ps_cnt_data_node[PS_WIDTHAD-1..0] = VECTOR;
 ps_cnt_load = VCC ;
 ---------------
 -- в память записываем адрес возврата
  data_out[PS_WIDTHAD-1..0] = ps_addr[PS_WIDTHAD-1..0];
 data_out[15..PS_WIDTHAD] = GND ;
 push = VCC ;
 ---------------
ELSE
-- Нет запросов прерываний, или запросы этого уровня уже об-
работаны !
-- обработка операнда
--//////////////////////////////////////////////////////////////////
-- ОБРАБОТКА ОПЕРАНДА
--//////////////////////////////////////////////////////////////////
 CASE data_in[23..20] IS
==========================================
-- H»1» — JMP < Y >        — переход без условий
==========================================
   WHEN 1 =>
    -- выходы памяти команд, где находится абсолютный адрес
перехода
     ps_cnt_data_node[PS_WIDTHAD-1..0] = data_in[PS_WIDTHAD-
1..0];
     ps_cnt_load = VCC ;     -— переход без условий
==========================================
-- H»2» — CALL < Y >       — вызов без условий
------------------------------------------------------------------
-- состоит из двух действий
-- JMP < Вектор >                  — переход без условий
-- Push                            — занесение в стек
==========================================
   WHEN 2 =>
    -- выходы памяти команд, где находится абсолютный адрес
перехода
     ps_cnt_data_node[PS_WIDTHAD-1..0] = data_in[PS_WIDTHAD
-1..0];
     ps_cnt_load = VCC ;    -- переход без условий
     -- в память записываем адрес возврата
     data_out[PS_WIDTHAD — 1..0] = ps_addr[PS_WIDTHAD -
1..0];
     data_out[15..PS_WIDTHAD] = GND ;
     push = VCC ;
==========================================
-- H»3» — RET              — возврат из прерывания
==========================================
   WHEN 3 =>
    -- записываем абсолютный адрес возврата в счетчик адресов
```

KiT#24(7)_Olia.qxd 2/6/03 3:32 PM Page 65


```
    -- от выходов стека, где находится абсолютный адрес возврата из прерывания
    -- или вызова подпрограммы
    ps_cnt_data_node[PS_WIDTHAD-1..0] = stack_q[PS_WIDTHAD-1..0];
    ps_cnt_load = VCC ;                      -- переход без условий
 ---------------
    pop = VCC ;                              -- извлечь из стека
==============================================
-- H»8» — Mov Reg, Reg — загрузка данных из регистра в регистр 0 !!!
==============================================
    -- будем считать, что адреса регистра-передатчика расположены в битах [3..0] слова
    -- а адреса регистра-приемника расположены в битах [19..16] слова
    WHEN 8=>
    CASE data_in[19..16] IS
    WHEN 0 =>
        CASE data_in[3..0] IS
        WHEN 0 =>
        data_out[15..0] = rg0_q[15..0];  -- выходы регистра 0
        WHEN 1 =>
        data_out[15..0] = rg1_q[15..0];  -- выходы регистра 1
        END CASE;
        wr_reg_ena[0] = VCC;
WHEN 1 =>
        CASE data_in[3..0] IS
        WHEN 0 =>
        data_out[15..0] = rg0_q[15..0];  -- выходы регистра 0
        WHEN 1 =>
        data_out[15..0] = rg1_q[15..0];  -- выходы регистра 1
        END CASE;
        wr_reg_ena[1] = VCC;
    END CASE;
==============================================
-- H»9» — Mov Reg, [Mem] — загрузка в регистр-приемник данных из памяти
==============================================
    WHEN 9 =>
    CASE data_in[19..16] IS
    WHEN 0 =>
        data_out[15..0] = ds_q[15..0]; -- выходы памяти данных
        wr_reg_ena[0] = VCC;
    WHEN 1 =>
        data_out[15..0] = ds_q[15..0]; -- выходы памяти данных
        wr_reg_ena[1] = VCC;
    END CASE;
==============================================
-- H»A» — Mov [Mem], Reg — загрузка данных из регистра-передатчика в память
==============================================
    WHEN H»A» =>
    CASE data_in[3..0] IS
    WHEN 0 =>
        data_out[15..0] = rg0_q[15..0];      -- выходы регистра 0
    WHEN 1 =>
        data_out[15..0] = rg1_q[15..0];      -- выходы регистра 1
    END CASE;
    ds_wr_ena = VCC;
==============================================
-- H»B» — LDI Reg < C > — непосредственная загрузка
-- в регистр-приемник данных из памяти команд по текущему адресу
==============================================
    WHEN H»B» =>
    CASE data_in[19..16] IS
        WHEN 0 =>
        data_out[15..0] = data_in[15..0];
-- выходы памяти команд
        wr_reg_ena[0] = VCC;
        WHEN 1 =>
        data_out[15..0] = data_in[15..0];
-- выходы памяти команд
        wr_reg_ena[1] = VCC;
    END CASE;
==============================================
 END CASE;
END IF;
--==============================================
-- Шина данных на счетчик адресов памяти команд «PS»
 ps_cnt_data[PS_WIDTHAD-1..0] = ps_cnt_data_node[PS_WIDTHAD-1..0] ;
--==============================================
END;
```

Создадим файл программы, которую будет выполнять микропроцессор.

Для описания памяти команд применим библиотечную функцию lpm_rom, и для ее инициализации применим файл step1.mif. Файл step1 будет содержать коды команд микропроцессора, требуемые для отладки проекта.

Чтобы нам было легче ориентироваться в диаграмме временной симуляции проекта, создадим несколько файлов инициализации памяти команд, причем каждый файл будет проверять работу не более чем одной или двух команд микропроцессора.

Файл step1 будет проверять работу команд NOP и JMP. Коды команд и их мнемоника приведены в таблице 1. По этой программе микропроцессор должен выполнить два шага и вернуться в нулевой адрес. Далее процесс доолжен повторяться циклически.

Для проверки микропроцессора по программе Step1, создадим файл инициализации памяти программ микропроцессора step1.mif

```
WIDTH = 24;
DEPTH = 128;

ADDRESS_RADIX = HEX;
DATA_RADIX = HEX;

CONTENT BEGIN
        [0..1]      :       000000;
%       NOP         %
        2           :       100000;
%       JMP <000000> %
        [3..7f]     :       000000;
%       NOP         %
END;
```

Остальные команды микропроцессора могут быть проверены аналогично. Подробное описание работы с этой программой будет приведено в разделе, описывающем этап симуляции работы микропроцессора.

Создадим файл, описывающий микропроцессора. Верхний файл проекта.

Теперь мы можем собрать верхний файл проекта — непосредственно микропроцессор. В микропроцессор входят файлы:

1. dffex_rg — этот файл будет выполнять все функции, где необходимы регистры: порт ввода-вывода, DS и др.,
2. stack — этот файл будет выполнять функцию стека,
3. ps_cnt — этот файл будет выполнять функцию счетчика адресов памяти команд,
4. alu — этот файл будет выполнять функцию АЛУ,
5. для описания памяти команд применим библиотечную функцию lpm_rom, и для ее инициализации применим файл step1.mif.

Далее приведен текст файла описания микропроцессора uP.tdf

```
TITLE «Тестовый проект микропроцессор — uP» ;
-- Version 1.0
-- Copyright Iosif Karshenboim, 2002
-- You may use or distribute this function freely,
-- provided you do not remove this copyright notice.
-- If you have questions or comments, feel free to
-- contact me by email at ik@lmail.loniis.spb.su
%
-- Описание команд
-- Код операции 4 бита.
0 — NOP
1 — JMP < Y >          – переход без условий
2 — CALL
3 — RET                – возврат
8 — MOV Reg, Reg       – загрузка в регистр-приемник данных из регистра-передатчика
9 — MOV Reg, [Mem]     – загрузка в регистр-приемник данных из памяти по непосредственному адресу
A — MOV [Mem], Reg     – загрузка в память по непосредственному адресу данных из регистра-передатчика
B — LDI Reg, < C >     – загрузка в регистр-приемник константы — данных из памяти команд по текущему адресу

Регистры имеют номера 0 — 15,
для записи в них есть сигналы wr_reg_ena[15..0],
регистр 0 — аккумулятор
регистр 1 — регистр ввода–вывода
регистр 2 — регистр ввода–вывода
%

INCLUDE «stack.inc»;
INCLUDE «dffex_rg.inc»;
INCLUDE «alu.inc»;
INCLUDE «ps_cnt.inc»;
INCLUDE «lpm_rom.inc»;

PARAMETERS
(
-- Параметры «ALU»
 PS_WIDTHAD = 7, -- разрядность шины адресов памяти команд в словах «CPU_WIDTH»
```

Таблица 1. Коды, выполняемые микропроцессором, по программе step1

Адрес	Код	Мнемоника	Комментарий
0000	000000	NOP	Нет операции, переходим в следующий адрес
0001	000000	NOP	Нет операции, переходим в следующий адрес
0002	100000	JMP <000000>	Безусловный переход по адресу 000000
0003	000000	NOP	Нет операции, переходим в следующий адрес

```
PS_WIDTH = 24, -- разрядность шины данных памяти команд
DS_WIDTH = 16, -- разрядность шины данных памяти команд
---------------------------------------------------------------------
STACK_WIDTH = 8,-- глубина вложений в стек
---------------------------------------------------------------------
IRQ_VECTOR = H»0»,      -- адрес вектора запроса прерывания
---------------------------------------------------------------------
PS_FILE = «step1.mif» -- это файл команд микропроцессора
-- PS_FILE = «step2.mif»
);

CONSTANT STACK_DATA_WIDTH = PS_WIDTHAD; -- разрядность стека равна разрядности шины адресов памяти команд

--/////////////////////////////////////////////////////////////////
SUBDESIGN uP
(
-- системные сигналы
clk, reset,

-- сигнал запроса прерываний для «ALU», действует 1 clk
irq                          : INPUT = GND ;
-------------------
ps_cnt_load                  : OUTPUT ; -- загрузка счетчика адресов
upalu_data_in[PS_WIDTH-1..0] : OUTPUT ;
-------------------
rg0_q[15..0],                           -- выходы регистров
rg1_q[15..0]                 : OUTPUT ;
-------------------
)
VARIABLE

ps_cnt_node[PS_WIDTHAD-1..0],
rg0_q_node[15..0],
rg1_q_node[15..0]            : NODE ;

upalu : alu WITH ( -- Параметры «ALU»
        PS_WIDTHAD = PS_WIDTHAD, -- разрядность шины адресов памяти команд в словах
        PS_WIDTH = PS_WIDTH,         -- разрядность шины данных памяти команд
        DS_WIDTH = DS_WIDTH,         -- разрядность шины данных памяти данных
        ---------------------------------------------------------------
        STACK_WIDTH = STACK_WIDTH,
        STACK_DATA_WIDTH = STACK_DATA_WIDTH,
        ---------------------------------------------------------------
        IRQ_VECTOR = IRQ_VECTOR    ); -- адрес вектора запроса прерывания

--=============================================
BEGIN
--=============================================
-- Здесь ко микропроцессора подключен вход запроса прерывания
upalu.irq = irq;
---------------------------------------------------------------------
-- Здесь ко входу АЛУ подключен счетчик адресов PS
upalu.ps_addr[PS_WIDTHAD-1..0] = ps_cnt_node[PS_WIDTHAD-1..0];
---------------------------------------------------------------------
-- Здесь ко входу АЛУ подключен регистр как память данных DS
upalu.ds_q[DS_WIDTH-1..0] = dffex_rg ( upalu.data_out[DS_WIDTH-1..0], clk, !reset, , upalu.ds_wr_ena )

WITH ( m = DS_WIDTH-1, l = 0 ) ;
---------------------------------------------------------------------
-- Здесь ко входу АЛУ подключены два регистра как порты ввода–вывода
-- считаем, что порты имеют ту же разрядность, что и память данных
rg0_q_node[15..0] = dffex_rg ( upalu.data_out[DS_WIDTH-1..0], clk, !reset, , upalu.wr_reg_ena[0] )
                                        WITH ( m = DS_WIDTH-1, l = 0 ) ;
upalu.rg0_q[DS_WIDTH-1..0] = rg0_q_node[15..0];
rg0_q[15..0] = rg0_q_node[15..0];

rg1_q_node[15..0] = dffex_rg ( upalu.data_out[DS_WIDTH-1..0], clk, !reset, , upalu.wr_reg_ena[1] )
                                        WITH ( m = DS_WIDTH-1, l = 0 ) ;
upalu.rg1_q[DS_WIDTH-1..0] = rg1_q_node[15..0];
rg1_q[15..0] = rg1_q_node[15..0];
---------------------------------------------------------------------
-- Здесь ко входу АЛУ подключен Стек
upalu.stack_q[PS_WIDTHAD-1..0] =
    stack ( .data[PS_WIDTHAD-1..0] = upalu.data_out[PS_WIDTHAD-1..0] ,
    .push = upalu.push, .pop = upalu.pop, .clk = clk )
    WITH ( STACK_WIDTH = STACK_WIDTH , data_width = STACK_DATA_WIDTH );
---------------------------------------------------------------------
-- Здесь сам счетчик адресов PS
ps_cnt_node[PS_WIDTHAD-1..0] = ps_cnt ( .clk = clk, .reset = reset,
    -- данные для загрузки в счетчик команд
    .data_in[PS_WIDTHAD-1..0] = upalu.ps_cnt_data[PS_WIDTHAD-1..0],
    .ps_cnt_load = upalu.ps_cnt_load,      .ps_cnt_ena = VCC )
    WITH (PS_WIDTHAD = PS_WIDTHAD );

---------------------------------------------------------------------
-- Здесь ко входу АЛУ подключена память команд — PS как асинхронный блок
upalu.data_in[PS_WIDTH-1..0] = lpm_rom ( .address[PS_WIDTHAD-1..0] = ps_cnt_node[PS_WIDTHAD-1..0],
                              .memenab = VCC )
        WITH (LPM_WIDTH = PS_WIDTH,    -- разрядность шины данных памяти команд
        LPM_WIDTHAD = PS_WIDTHAD, -- разрядность шины адресов памяти команд
        LPM_FILE = PS_FILE,
        LPM_ADDRESS_CONTROL = «UNREGISTERED»,
        LPM_OUTDATA = «UNREGISTERED» );

--=============================================
-- Все, что здесь написано нужно только для отладки
ps_cnt_load = upalu.ps_cnt_load;
upalu_data_in[PS_WIDTH-1..0] = lpm_rom ( .address[PS_WIDTHAD-1..0] = ps_cnt_node[PS_WIDTHAD-1..0],
                              .memenab = VCC )
        WITH (LPM_WIDTH = PS_WIDTH ,-- разрядность шины данных памяти команд
        LPM_WIDTHAD = PS_WIDTHAD, -- разрядность шины адресов памяти команд
        LPM_FILE = PS_FILE,
        LPM_ADDRESS_CONTROL = «UNREGISTERED»,
        LPM_OUTDATA = «UNREGISTERED»);
--=============================================

END;
```

В качестве DEVICE для проекта — выберем серию ACEX, а для выбора микросхемы установим режим AUTO. Произведем компиляцию проекта. Далее, создадим файл симуляции проекта микропроцессора с файлом команд step1, такой как на рис. 3. И произведем симуляцию проекта.

Синхрочастота, поступающая на тактовый вход микропроцессора, имеет длительность импульса 20 нсек и каждый положительный фронт синхрочастоты пронумерован. Все описание работы будет опираться на соответствующий фронт синхрочастоты. Будем называть фронты так: фронт1, фронт2 и т. д.

При действии фронта1 микропроцессор находится в состоянии сброса. Счетчик адреса обнуляется и из PS извлекается первая команда NOP, и никаких изменений не происходит. Под фронт2 и фронт3 из PS так же извлекаеются команды — NOP. Таким образом выполняются строки 0..1 из файла step1. Когда счетчик адреса памяти команд находится в состоянии 2, из памяти команд извлекается команда, имеющая код 100000. Это команда — JMP <000000>. Получив код этой команды, АЛУ производит дешифрацию кода операции, и выставляет сигнал на запись нового адреса в счетчик команд. Код адреса из АЛУ подается на счетчик команд. Под фронт4 производится запись в счетчик команд нового адреса. Далее

Диаграмма работы микропроцессора с файлом step1

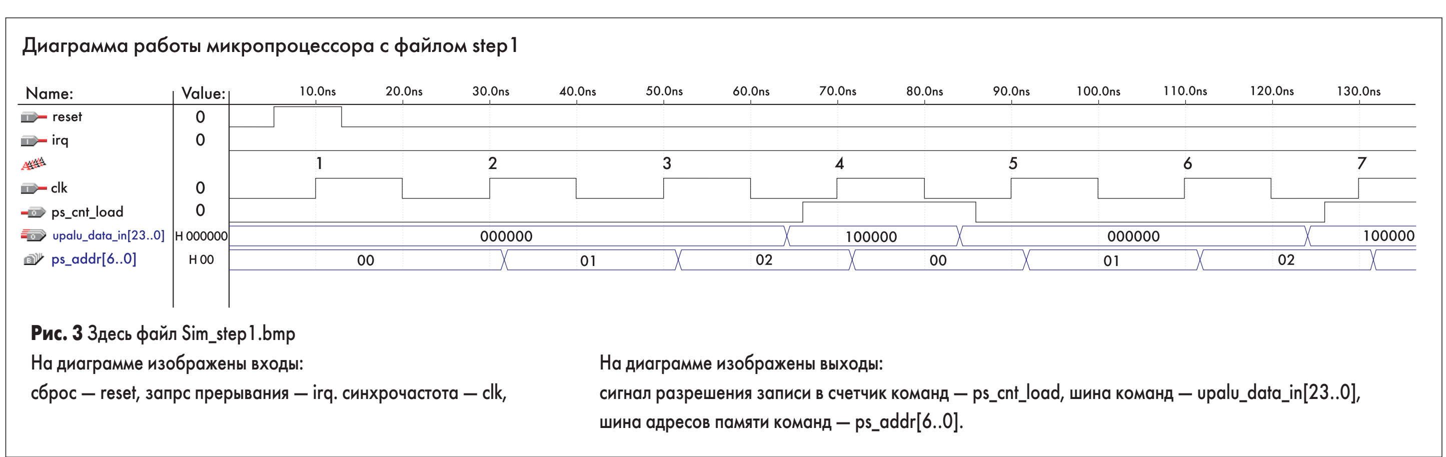

Рис. 3 Здесь файл Sim_step1.bmp

На диаграмме изображены входы:
сброс — reset, запрс прерывания — irq. синхрочастота — clk,

На диаграмме изображены выходы:
сигнал разрешения записи в счетчик команд — ps_cnt_load, шина команд — upalu_data_in[23..0],
шина адресов памяти команд — ps_addr[6..0].

KiT#24(7)_Olia.qxd 2/6/03 3:32 PM Page 67

на шинах адреса устанавливается новый адрес. По этому адресу извлекается новый код команды. Поскольку мы выполняли команду JMP <000000>, то микропроцессор перешел в адрес 0000 и на этом программа зацикливается в соответствии с файлом step1. Результат симуляции показывает, что наш микропроцессор успешно выполнил команды NOP и JMP.

Далее читатели могут самостоятельно выполнить подобные действия для проверки работы микропроцессора с другими командами.

Для полной проверки микропроцессора, создадим файл инициализации памяти программ микропроцессора step2.mif. В этом файле будут проверяться все команды микропроцессора и так же будет проверяться работа по прерываниям. Произведем компиляцию проекта с данным файлом инициализации памяти.

```
WIDTH = 24;
DEPTH = 128;

ADDRESS_RADIX = HEX;
DATA_RADIX = HEX;

CONTENT BEGIN
    [0..2]   :  000000;
    3        :  200010;  %  CALL <0010>        %
    [4..6]   :  000000;
    7        :  100000;  %  JMP <0000>         %
    [8..F]   :  000000;
    10       :  B055AA;  %  LDI Reg0, < 55AA > %
    11       :  B133CC;  %  LDI Reg1, < 33CC > %
    12       :  810000;  %  MOV Reg1, Reg0     %
    13       :  A10000;  %  MOV [Mem], Reg1    %
    14       :  B01111;  %  LDI Reg0, < 1111 > %
    15       :  900000;  %  MOV Reg0, [Mem]    %
    16       :  000000;  %  NOP                %
    17       :  300000;  %  RET                %
    [18..7F] :  000000;
END;
```

Создадим новый файл симуляции проекта.

Временная диаграмма симуляции работы микропроцессора с файлом команд step2 приведена на рис. 4.

Синхрочастота, поступающая на тактовый вход микропроцессора, также как и в предыдущем случае имеет длительность импульса 20 нсек и каждый положительный фронт синхрочастоты пронумерован. При действии фронта1 микропроцессор находится в состоянии сброса. Счетчик адреса обнуляется и из PS извлекается первая команда NOP и никаких изменений не происходит. Под фронт2, фронт3 и фронт4 из PS так же извлекаеютя команды — NOP. Таким образом выполняются строки 0..2 из файла step2. Счетчик команд находится в состоянии 3 и из этого адреса извлекается код команды 200010, что соответствует команде CALL <0010>. Под фронт5 АЛУ формирует импульс записи нового адреса в счетчик команд и следующий адрес будет 10. Так же под фронт5 производится запись в стек адреса возврата, на выходной шине стека появляется значение 3. Под фронт6 начинает выполняться подпрограмма, см. файл step2, с адреса 10. Здесь выполняется команда LDI Reg0, < 55AA >. АЛУ формирует строб записи в регистр 0. Данные на вход регистра поступают по внутренней шине микропроцессора. Аналогично под фронт7 выполняется команда LDI Reg1, < 33CC >. Под фронт8 выполняется команда MOV Reg1, Reg0. Под фронт9 выполняется команда MOV [Mem], Reg1. Под фронт10 выполняется команда LDI Reg0, < 1111 >. Эта команда нужна нам для того, чтобы показать, что следующая за ней команда MOV Reg0, [Mem], выполняемая под фронт11 снова изменит содержимое этого регистра. Под фронт12 выполняется команда NOP. Под фронт13 из памяти команд выбрана команда RET и выполняются следующие действия: из стека извлекается адрес возврата и заносится в счетчик команд. Стек выполняет команду POP и на его выходе значение с 3 меняется на 0, т. е. стек приходит в то состояние, которое у него было до вызова подпрограммы, см фронт0..4. Далее происходят следующие действия: под фронт14 микропроцессор вернулся в адрес 3 и снова начал выполнять вызов подпрограммы по адресу 10. Но под фронт16 на входе АЛУ появился сигнал запроса прерывания irq. По этому сигналу производятся действия, аналогичные действиям по команде CALL <0000>. АЛУ формирует импульс записи нового адреса — адреса вектора прерывания в счетчик команд и следующий адрес будет 0. Так же под этот фронт производится запись в стек адреса возврата, на выходной шине стека появляется значение 11, а предыдущее значение стека — 3 будет «протолкнуто» внутрь стека. Далее микропроцессор выполняет команды, которые будут извлекаться из PS из адресов, по которым выполнен переход, по адресу вектора прерывания. Таким образом, мы видим, что микропроцессор выполняет все команды в соответствии с заданием

Несколько слов об отладке программ во встроенных микропроцессорах.

Во-первых, используемые микросхемы FPGA — перепрограммируемые, то есть в случае обнаружения ошибки всегда есть возможность исправить эту ошибку без переделки всего устройства.

Во-вторых, при отладке программы во встроенных микропроцессорах есть много дополнительных аппаратных возможностей, используя которые, можно достичь быстрого положительного результата.

Поскольку весь проект «самодельного» микропроцессора для нас открыт, и мы знаем его устройство, то мы можем облегчить отладку программ, при помощи дополнительных аппаратных ресурсов, так как в процессе отладки можно использовать FPGA с большим количеством ячеек, чем нужно для функционирования проекта. Это позволит выделять отладочные ресурсы. К таким ресурсам можно отнести:

А) дополнительные поля памяти для вызова тестовых и мониторных программ,

Б) тестовые входы и выходы для микропроцессора, позволяющие снимать состояния сигналов и задавать воздействия,

В) дополнительные поля памяти могут использоваться для включения в систему встроенных логических анализаторов см. Л15, что является очень мощным средством отладки,

Г) дополнительные счетчики, тестовые триггеры и т. д.

Далее, ко встроенным микропроцессорам может быть «пристроен» блок, выполняющий функции внутрисхемного эмулятора. С его помощью пользователю становятся доступны внутренние регистры процессора, осуществляется останов по адресу, шаговый режим и пр.

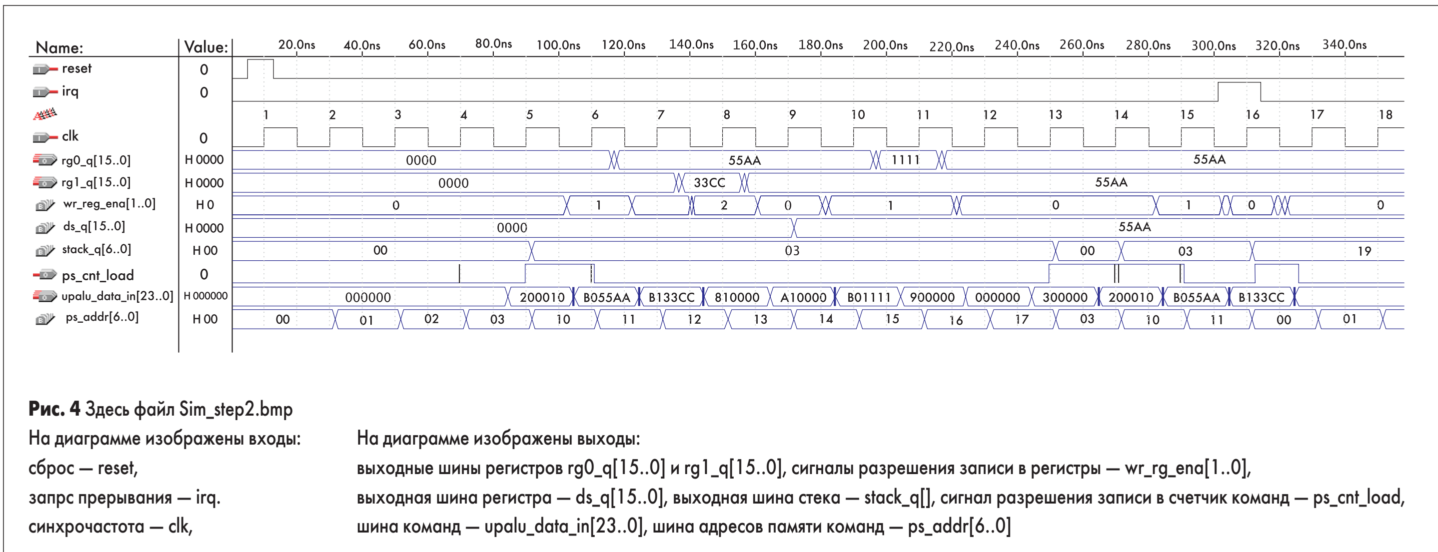

Рис. 4 Здесь файл Sim_step2.bmp

На диаграмме изображены входы:
сброс — reset,
запрс прерывания — irq.
синхрочастота — clk,

На диаграмме изображены выходы:
выходные шины регистров rg0_q[15..0] и rg1_q[15..0], сигналы разрешения записи в регистры — wr_rg_ena[1..0],
выходная шина регистра — ds_q[15..0], выходная шина стека — stack_q[], сигнал разрешения записи в счетчик команд — ps_cnt_load,
шина команд — upalu_data_in[23..0], шина адресов памяти команд — ps_addr[6..0]

KiT#24(7)_Olia.qxd 2/6/03 3:32 PM Page 68

Все эти возможности позволяют проводить успешную отладку программы не только в программном симуляторе, но и на «живом железе».

Что мы получили.

По итогам компиляции проекта, из файла рапорта, получаем следующий результат:

Тестовый проект микропроцессор — uP

** DEVICE SUMMARY **

Chip/ POF	Device	Input Pins	Output Pins	Bidir Pins	Memory Bits	Memory %Utilized	LCs	LCs %Utilized
up	EPF10K30ETC144-1	3	57	0	6144	25%	182	10%
User Pins:		3	57	0				

Для реализации проекта потребовалось всего 182 ячейки, при том, что проект содержит три 16-битных регистра.

Сравнительная характеристика.

Для сравнения можно сказать, что широко известные микропроцессоры, такие как MCS51 выполняют одну команду за 12 тактов, а их более «продвинутые» версии — за 4 такта, и только новейшие CYGNAL за 1 такт. Чтобы сравнить производительности различных микропроцессоров с тем, что мы разработали, воспользуемся диаграммой, приведенной в Л16, и добавим к ней данные по нашему проекту (см. рис. 5).

Заключение

Мы получили 16-битный RISC-микропроцессор, работающий на частоте 50 МГц, и выполняющий все команды за один такт. И, следовательно, спроектированный микропроцессор имеет производительность 50 MIPS. Данный проект может послужить основой для разработки микропроцессора с требуемым набором команд и, с требуемыми для вычислений, ресурсами. На основе такого микропроцессора может быть разработан микроконтроллер, путем добавления к проекту блоков периферийных устройств, из имеющихся библиотечных мегафункций. Проект имеет все возможности, чтобы его можно было легко доработать до реальной конструкции. Конечно, при значительном увеличении числа команд сложность АЛУ тоже возрастет, а быстродействие несколько уменьшится. Однако применение конвейера команд позволяет преодолеть эти трудности. Но конвейер команд, тайм-слоты задержки переходов и прочие профессиональные «хитрости» выходят за рамки данной статьи.

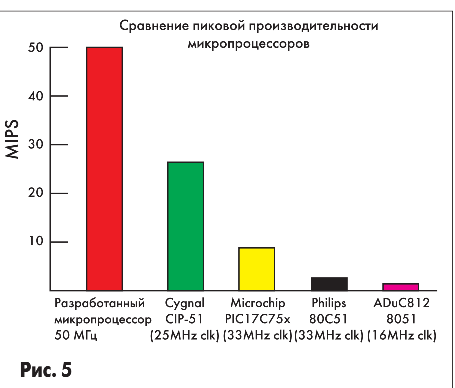

Рис. 5

Литература

1. Дмитрий Фомин «Кремний на бумаге» или fabless?
Живая электроника россии, 2001, том 1 http://www.elcp.ru
2. А.П. Антонов Язык Описания цифровых устройств AlteraHDL. М. РадиоСофт, 2002.
3. И.Кривченко. Системы на кристалле: общее представление и тенденции развития. Компоненты и технологии, №6 2001.
4. Processors drive (or dive) into programmable-logic devices. By Markus Levy. EDN От 07.20.2000
5. Каршенбойм И.Г. Микроконтроллер для встроенного применения — NIOS. Конфигурация шины и периферии. «Компоненты и технологии» №2, 3, 4, 5 2002
6. Altera™. Nios Soft Core Embedded Processor, data sheet June 2000, ver.1
7. Altera™. Nios_2_0_CPU_datasheet
8. Altera™. Nios 32-Bit Programmer's Reference Manual January 2002 Version 2.0
9. Altera™. Nios 16-Bit Programmer's Reference Manual January 2002 Version 2.0
10. Altera™. ARM-Based Embedded Processor Device Overview, data sheet February 2001, ver.1.2
11. Altera™. Nios_tutorial.pdf
12. Altera™. Nios_an188_Custom_Instructions
13. Altera™. Simultaneous Multi-Mastering with the Avalon Bus
14. И. Каршенбойм, Н. Семенов Микропрограммые автоматы на базе специализированных ИС. — Chip News, №7, 2000.
15. Каршенбойм И.Г. «Встроенный» логический анализатор — инструмент разработчика «встроенных» систем. Схемотехника №12, 2001.
16. 8051CYGNAL.pdf