УДК 681.3

Моделирование композиционного микропрограммного устройства управления с разделением кодов и кэш-памятью микрокоманд

Баркалов А.А., Ковалев С.А., Бабаков Р.М., Николаенко Д.В.
Университет Зеленогурский (Польша),
Донецкий национальный технический университет
Государственный университет информатики и искусственного интеллекта
a.barkalov@iie.uz.zgora.pl, cpld@mail.ru

Abstract

Barkalov A.A., Kovalev S.A., Babakov R.M., Nikolaenko D.V. Modelling of compositional micriprogram control unit with division of codes and cache-memory of microinstructions. *The approach to simulation modeling of compositional microprogram control unit with division of codes and cache memory of microinstructions, based on using of hardware description language is offered. The received model allows to investigate logical correctness of modelled structure, to make its synthesis on a crystal of programmable chip and also to estimate hardware expenses at various parametres of model.*

Общая постановка проблемы

Важной составной частью любой цифровой системы является устройство управления (УУ), координирующее работу всех блоков системы [1]. Оптимизация быстродействия схем УУ позволяет расширить область применения цифровых систем и является актуальной научно-технической задачей для промышленности средств ВТ.

Одним из способов реализации устройств управления являются композиционные микропрограммные устройства управления (КМУУ) с разделением кодов [2]. Особенностью данных структур является использование гетерогенного элементного базиса – ПЛИС для реализации схемы адресации и ПЗУ (ППЗУ) для реализации управляющей памяти. Это приводит к тому, что относительно медленный базис ПЗУ составляет значительную часть длительности такта работы КМУУ [1, 2]. Следовательно, увеличение быстродействия схемы КМУУ может быть достигнуто за счет снижения времени доступа к управляющей памяти.

В работе [3] показана возможность использования кэш-памяти микрокоманд для уменьшения средней длительности такта работы КМУУ с разделением кодов. При этом в структуру устройства добавляется модуль кэш-памяти, увеличивающий аппаратурные затраты в схеме устройства.

Для определения эффективности структур КМУУ с кэш-памятью должны быть известны численные значения аппаратурных затрат и быстродействия их логических схем, получение которых возможно с помощью специализированных САПР цифровых систем [4]. Подобные САПР, как правило, требуют представления моделируемой системы на языке описания аппаратуры VHDL [4, 5]. Таким образом, оценке эффективности структур КМУУ с разделением кодов и кэш-памятью предшествует этап разработки VHDL-моделей структур КМУУ с кэш-памятью. На сегодняшний день вопросы VHDL-моделирования кэш-памяти как структурного элемента КМУУ с разделением кодов являются неисследованными.

Постановка задач и целей исследования

В настоящей работе ставится научная задача разработки программной модели КМУУ с разделением кодов, содержащей модуль кэш-памяти полностью ассоциативного типа с алгоритмом замещения данных со случайным выбором замещаемой строки (стратегия замещения Random). Целью разработки данной модели являются:

1. Анализ корректности логической организации КМУУ с разделением кодов и кэш-памятью микрокоманд.
2. Получение для разработанной модели УУ числовых характеристик быстродействия и аппаратурных затрат при реализации в базисе программируемых БИС типа FPGA и CPLD.
3. Получение, при необходимости, бинарных прошивок для синтеза разработанных структур в FPGA или CPLD.

Используемые средства моделирования

Одним из современных средств описания логической организации цифровых устройств сегодня является язык описания аппаратуры VHDL. Вопросы синтеза цифровых устройств с

использованием VHDL достаточно широко рассмотрены в литературе [5], а сам язык поддерживается большинством САПР вычислительной техники.

В качестве программной среды моделирования использован пакет Active-VHDL компании ALDEC [4]. Являясь интегрированной средой моделирования, Active-VHDL включает различные программы для ввода проектов, VHDL и Verilog компиляторы, единое ядро моделирования, программные модули отладки, графический и текстовый вывод результатов моделирования, множество различных вспомогательных инструментов.

В качестве средства синтеза и имплементации может быть использован пакет Xilinx ISE 9.2i фирмы Xilinx [5-7]. Допустимо также использование и других пакетов синтеза и имплементации, ориентированных на продукцию различных производителей интегральных схем.

Общий подход к построению модели

Чтобы достичь всех поставленных целей, проектируемую VHDL-модель будем реализовывать структурным стилем, используя операторы из синтезируемого подмножества VHDL [4, 5]. Верхний уровень описания имеет интерфейс, включающий входные и выходные сигналы. Наличие данного интерфейса позволяет использовать данное описание как синтезируемый элемент иерархии более общего устройства (например, совместно с операционным автоматом).

Согласно структуре устройства, предложенной в [3], модель будет включать следующие основные структурные элементы:

– схема адресации, формирующая код следующей микрокоманды при переходах между операторными линейными цепями (ОЛЦ) [1];

– счетчик микрокоманд, выполняющий инкремент номера микрокоманды внутри текущей ОЛЦ;

– регистр памяти, хранящий код текущей ОЛЦ;

– модуль кэш-памяти, предназначенный для временного хранения наиболее часто используемых микрокоманд;

– управляющая память, предназначенная для хранения микрокоманд.

Помимо рассмотренных блоков, в процессе функционирования КМУУ участвует ряд дополнительных блоков, реализация которых может быть различной и в настоящей работе не рассматривается. К таким блокам, например, относятся [2, 3]:

– операционный автомат, формирующий сигналы логических условий (ЛУ);

– генератор синхронизации;

– триггер запуска-останова, останавливающий процесс функционирования КМУУ по завершении микропрограммы;

– формирователь сигнала «Сброс»

и другие.

Отметим, что, согласно принципу гетерогенности элементного базиса КМУУ, схема адресации, счетчик, регистр памяти и модуль кэш-памяти традиционно реализуются в базисе ПЛИС, для чего их VHDL-описания должны быть реализованы структурным стилем. Остальные блоки реализуются вне кристалла ПЛИС, и их VHLD-описание может быть выполнено поведенческим стилем.

Также отметим, что для обеспечения совместного функционирования блоков в составе модели их следует объединять на более высоком уровне иерархии VHDL-описания.

Разработка модели КМУУ

Рассмотрим VHDL-реализацию каждого из перечисленных блоков. Особенностью приведенных описаний является использование так называемых generic-констант, позволяющих параметризировать проектируемые модули.

Схема адресации

Данный блок формирует код следующей ОЛЦ Ψ и код входа ОЛЦ Ф на основании кода текущей ОЛЦ и сигналов логических условий.

Внешнее описание блока имеет вид:

```
entity CA is  -- Circuit of addressing
generic (
R_Tau: natural; -- Разрядность кода ОЛЦ
R_LC:  natural; -- Количество ЛУ
R_T: natural);   -- Разрядность номера МК
port (X: in bit_vector (1 to R_LC);
      T: in bit_vector (1 to R_Tau);
      Phi: out bit_vector (1 to R_T);
      Ksi: out bit_vector (1 to R_Tau));
end entity;
```

Внутреннее описание блока:

```
architecture CA_A of CA is
begin
process (X, T)
-- Инверсные значения ЛУ:
variable nX: bit_vector (1 to R_LC);
-- Инверсия кода текущей ОЛЦ
variable nT: bit_vector (1 to R_Tau);
-- Код следующей ОЛЦ
variable D1: bit_vector (1 to R_Tau);
-- Номер входа в ОЛЦ
variable D2: bit_vector (1 to R_T);

begin
       nX := not X;
       nT := not T;

-- Пример функций переходов
D1(1) := nT(1) and T(2) and nT(3) and nT(4);
D2(1) := '0';

-- Формирование выходных сигналов
Ksi <= D1(1 to R_Tau);
Phi <= D2(1 to R_T);
end process;
end architecture CA_A;
```

Счетчик микрокоманд

По сигналу управляемой синхронизации, поступающей из модуля кэш-памяти, в случае y_0=1 счетчик загружает значение из схемы адресации. В случае y_0=0 счетчик инкрементирует внутреннее содержимое.

Внешние описание счетчика:

```
entity CT is
  generic (R_T: natural); -- Разрядность
  port (D: in bit_vector (1 to R_T);
        Int_Clk: in bit;
        R: in bit;
        y0:  in bit;
        O: out bit_vector (1 to R_T));
end entity CT;
```

Внутреннее описание счетчика:

```
architecture CT_A of CT is
begin
process (Int_Clk, R)
variable contents: bit_vector (1 to R_T);
variable i: natural;
variable carry: bit;
begin
 if r='1' then
    contents:=(others=>'0');
  else
-- ------------------------------
  if y0='0' and Int_Clk='1' then
     carry:='1';
f1:  for i in R_T downto 1 loop
  if carry='1' then
     if contents(i)='1' then
        contents(i):='0';
        carry:='1';
     else
        contents(i):='1';
        carry:='0';
     end if;
  end if;
end loop f1;
  end if;
  if y0='1' and Int_Clk='1' then
     contents := D;
   end if;
-- ------------------------------
   end if;
  O <= contents;
-- ------------------------------
end process;
end architecture CT_A;
```

Регистр памяти

По сигналу синхронизации в слчуае y_0=1 регистр загружает значение из схемы адресации. В случае y_0=0 регистр остается неизменным.

Внешние описание счетчика:

```
entity RM is
  generic (R_Tau: natural); -- Разрядность
  port (D: in bit_vector (1 to R_Tau);
        Int_Clk: in bit;
        R: in bit;
        y0:  in bit;
        O: out bit_vector (1 to R_Tau));
end entity RM;
```

Внутренняя архитектура регистра:

```
architecture RM_A of RM is
begin
process (Int_Clk, R)
variable contents: bit_vector (1 to R_Tau);
begin
  if r='1' then
     contents:=(others=>'0');
  else
-- ------------------------------------
    if y0='1' and Int_Clk='1' then
       contents := D;
    end if;
-- ------------------------------------
    end if; -- if r='1'
  O <= contents
-- ------------------------------------
end process;
end architecture RM_A;
```

Данные блоки, а также модуль кэш-памяти, который будет рассмотрен ниже, объединяются в блоке CMCU, являющимся блоком более высокого уровня иерархии. Объединение блоков происходит на структурном уровне с использованием внутренних шин. Блок CMCU принимает в качестве настроечных констант все значения, необходимые для параметризации внутренних элементов блока.

Внешнее описание блока имеет вид:

```
entity CMCU is -- КМУУ с разделением кодов
generic (
-- Разрядность счетчика:
R_T:           natural;
-- Разрядность кода ОЛЦ:
R_Tau:         natural;
-- Разрядность адреса микрокоманды :
R_Adr:        natural;
-- Разрядность микрокоманды:
R_MI:          natural;
-- Количество МК в строке кэш-памяти:
N_MI:          natural;
-- Количество строк кэш-памяти:
N_Lines:       natural;
-- Разрядность кода слова в адресе МК:
R_Word:        natural;
-- Разрядность номера строки кэш-памяти:
R_Line:        natural;
-- Число тактов задержки при кэш-промахе
Delay:         natural;
-- Количество сигналов логических условий
R_LC:          natural);

port ( -- Сигнальный интерфейс блока
-- Сигналы логических условий:
X_bus: in bit_vector (1 to R_LC);
-- Внешняя синхронизация:
Clock: in bit;
-- Сигнал инициализации:
Rst: in bit;
-- Шина данных из управляющей памяти:
In_Bus: in bit_vector (1 to R_MI*N_MI);
-- Формируемые микрооперации:
Out_Bus: out bit_vector (1 to R_MI-1);
-- Разрешение чтения блока данных из УП:
RE: out bit;
-- Адрес микрокоманды (поступает в УП):
Adr: out bit_vector (1 to R_Adr));

end entity CMCU;
```

Внутренняя архитектура блока:

```
architecture CMCU_A of CMCU is
-- ------------------------------------
component CA
  generic (R_Tau: natural;
           R_LC:  natural;
           R_T: natural);
  port (X: in bit_vector (1 to R_LC);
        T: in bit_vector (1 to R_Tau);
        Phi: out bit_vector (1 to R_T);
        Ksi: out bit_vector (1 to R_Tau));
end component CA;
-- ------------------------------------
component CT
  generic (R_T: natural);
  port (D: in bit_vector (1 to R_T);
       Int_Clk: in bit;
       R: in bit;
       y0: in bit;
       O: out bit_vector (1 to R_T));
end component CT;
-- ------------------------------------
component RM is
  generic (R_Tau: natural);
  port (D: in bit_vector (1 to R_Tau);
        Int_Clk: in bit;
        R: in bit;
        y0:  in bit;
        O: out bit_vector (1 to R_Tau));
end component RM;
-- ------------------------------------
component CACHE – Модуль кэш-памяти
 generic (R_Adr:      natural;
          Word_Size:  natural;
          N_Words:    natural;
          R_Line:     natural;
          R_Word:     natural;
          N_Lines:    natural;
          Delay:      natural);
port (ADR: in bit_vector (1 to R_Adr);
      Clk: in bit;
      SysClk: out bit;
      In_Bus: in bit_vector (1 to R_MI*N_MI);
Cache_Out_Bus: out bit_vector (1 to R_MI-1);
      y0: out bit;
      RE: out bit;
      Reset: in bit);
end component CACHE;
-- ------------------------------------
-- Внутренние шины:
-- Код следующей ОЛЦ
signal Ksi_bus: bit_vector (1 to R_Tau);
-- Код входа следующей ОЛЦ
signal Phi_bus: bit_vector (1 to R_T);
-- Текущий адрес микрокоманды
signal Adr_bus: bit_vector (1 to R_T+R_Tau);
-- Микрооперация y0
signal y0: bit;
-- Внутренняя (управляемая) синхронизация
signal SysClock: bit;
-- ------------------------------------
begin --Структурное объединение компонентов

L1: component CA -- Схема адресации
    generic map (R_Tau, R_LC, R_T)
    port map (X_bus, Adr_Bus (1 to R_Tau),
              Phi_bus, Ksi_bus);

L2: component CT
    generic map (R_T)
    port map (Phi_bus, SysClock, Rst, y0,
            Adr_bus(R_Tau+1 to R_T+R_Tau));

L3: component RM
    generic map (R_Tau)
    port map (Ksi_bus, SysClock, Rst, y0,
              Adr_bus (1 to R_Tau));

L4: component CACHE
    generic map (R_Adr, R_MI, N_MI, R_Line,
                 R_Word, N_Lines, Delay)
    port map (Adr_Bus, Clock, SysClock,
              In_Bus, Out_Bus, y0, RE, Rst);

Adr <= Adr_bus;

end architecture CMCU_A;
```

Отметим, что данный блок, равно как и все его компоненты, описан с использованием синтезируемого подмножества языка VHDL и может быть синтезирован на кристалле ПЛИС.

Разработка модели модуля кэш-памяти

В разработанной модели КМУУ используется модуль кэш-памяти полностью ассоциативного типа с алгоритмом замещения Random. Это значит, что в случае кэш-промаха любой блок данных может быть размещен в любой строке кэш-памяти, а номер замещаемой строки выбирается псевдослучайным образом. Также отметим, что количество строк в модуле кэш-памяти не обязательно должно быть кратно степени двойки и может быть произвольным.

В настоящей работе предлагается представлять модуль кэш-памяти структурным объединением следующих блоков:

– блок данных, реализующий загрузку и хранение кэшированных микрокоманд;

– блок проверки кэш-попаданий, реализующий хранение и сравнение тэгов блоков микрокоманд и определяющий ситуации кэш-попаданий и кэш-промахов;

– блок управления синхронизацией, блокирующий прохождение синхроимпульсов в регистровые схемы устройства на требуемое количество тактов в случае кэш-промаха;

– счетчик номеров строк, реализующий последовательный синхронный перебор номеров строк и выступающий в качестве генератора псевдослучайных значений.

Рассмотрим VHDL-реализации каждого из перечисленных блоков.

Блок данных

Основу блока составляет массив из N_Lines строк, каждая из которых в свою очередь является массивом из N_Words слов. Отдельное слово представляет собой последовательность из Word_Size бит и может хранить одну микрокоманду ГСА.

Внешнее описание блока:

```
entity Cache_Data_Block is
generic (
-- Разрядность слова данных
Word_size: natural;
-- Разрядность кода строки
R_Line: natural;
-- Разрядность поля слова в адресе МК
R_Word: natural;
-- Количество слов в строке кэш-памяти
N_Words: natural;
```

```
-- Количество строк в кэш-памяти
N_Lines: natural);

port (
  -- Блок данных из управляющей памяти
  In_Bus: in bit_vector
            (1 to Word_Size*N_Words);
  -- Код обрабатываемой строки
  Str: in natural range 0 to N_Lines-1;
  -- Номер запрашиваемого слова
  Word: in bit_vector (1 to R_Word);
  -- Разрешение записи в кэш
  Data_WE: in bit;
  -- Выходная шина (запрошенная МК)
  Out_Bus: out bit_vector (1 to Word_Size));

end entity Cache_Data_Block;
```

Внутренняя архитектура:

```
architecture      Cache_Data_Block_A      of
Cache_Data_Block is
begin

process (In_Bus, Str, Data_WE, Word)

type Word_Element is array (0 to N_Words-1)
of bit_vector (1 to Word_Size);
type Data_Block is array (0 to N_Lines-1) of
Word_Element;

variable Data: Data_Block;    -- Блок данных
variable   Temp_Line:   bit_vector   (1   to
Word_Size*N_Words); -- Буфер чтения
variable Slovo: natural;
variable mult: natural;

begin

-- Вычисляем номер слова по его коду
Slovo := 0;
mult := 1;
m1_1: for i in R_Word downto 1 loop
       if Word(i)='1' then
         Slovo := Slovo + mult;
        end if;
       mult := mult * 2;
      end loop m1_1;
-- -------------------------------------
if (Data_WE='1') then  -- Если запись в кэш
    Temp_Line := In_Bus;
    m1_2: for i in 0 to N_Words-1 loop
    Data(Str)(i):=Temp_Line(1 to Word_Size);
    Temp_Line := Temp_Line sll Word_Size;
    end loop m1_2;
end if;

-- Чтение слова данных
out_Bus <= Data (Str)(Slovo);

end process;
end architecture Cache_Data_Block_A;
```

Блок проверки кэш-попаданий

Анализ ситуаций кэш-попаданий и кэш-промахов происходит путем анализа содержимого блока регистров тэгов и блока битов достоверности. Каждый бит достоверности по сигналу «Сброс» устанавливается в ноль, обозначая соответствующую строку кэш-памяти как неиспользуемую. При первой записи данных в данную строку кэша бит достоверности устанавливается в единицу и свидетельствует о том, что строка содержит реальный блок данных.

Внешнее описание блока:

```
entity Hit_Checker is
generic (R_Adr:     natural;
         R_Line:    natural;
         R_Word:    natural;
         N_Lines:   natural;
         N_Words:   natural);
port (Strr: in natural range 0 to N_Lines-1;
      Tag: in bit_vector (1 to R_Adr-R_Word);
      Word: in bit_vector (1 to R_Word);
      Reset: in bit;
      Out_string: out natural range
                  0 to N_Lines-1;
      Miss_Flag: out bit);
end entity Hit_Checker;
```

Внутреннее описание блока:

```
architecture Hit_Checker_A of Hit_Checker is
begin
process (Tag, Word, Reset)

type Tag_Array is array (natural range 0 to
N_Lines-1) of bit_vector (1 to R_Adr-R_Word);
variable Tag_Reg: Tag_Array;
variable Val_bit: bit_vector (0 to N_Lines-
1);
variable hit: natural range 0 to N_Lines-1;
variable Stroka: natural range 0 to N_Lines-
1;
variable  Result_line:  natural  range  0  to
N_Lines-1;
variable Result_bit: bit;
variable mult: natural;
variable i: natural range 0 to N_Lines-1;
begin
-- -------------------------------------
if (Reset='1') then
    hit:=0;
    Val_Bit:=(others=>'0');
else
    Stroka := 0;
    hit:=0;
m0: for i in 0 to N_Lines-1 loop
 if val_bit(i)='1' and Tag_Reg(i)=Tag then
       hit := hit + 1;
       Stroka := i;
 end if;
end loop m0;

if (hit=0) then - Если кэш-промах
   i := Strr;
   Tag_Reg(i) := Tag;
   Val_Bit(i) := '1';
   Result_Bit := '1';
   Result_Line := Strr;
end if;

if (hit=1) then -- Если кэш-попадание
    Result_Line := Stroka;
    Result_Bit := '0';
end if;
  Miss_Flag <= Result_Bit;
  Out_String <= Result_Line;
end if; -- Else
end process;
end architecture Hit_Checker_A;
```

Блок управления синхронизацией

В случае кэш-промаха данный блок блокирует прохождение синхросигнала в регистровые схемы КМУУ и операционного автомата на время, достаточное для чтения блока данных из ПЗУ управляющей памяти и записи его в кэш-память. В остальных случаях блок пропускает внешние синхроимпульсы без изменений.

Внешнее описание блока:

```
entity Clock_Controller is
-- Количество тактов задержки
generic (Delay: natural);

port (
-- Признак кэш-промаха
Miss_Flag: in bit;
-- Внешняя синхронизация
Clk: in bit;
-- Сигнал «Сброс»
Reset: in bit;
-- Разрешение чтения из УП
RE: out bit;
-- Разрешение записи в блок данных кэша
Data_WE: out bit;
-- Внутренняя (управляемая) синхронизация
SysClk: out bit);

end entity Clock_Controller;
```

Внутренняя архитектура блока:

```
architecture Clock_Controller_A of
Clock_Controller is
begin

process (Miss_Flag, Reset, Clk)
constant Number_of_Tacts: natural := Delay;
variable Counter: natural;
variable Synch: bit;
variable ReadEnable: bit;
begin

 if Reset='1' then
      synch:='1';
      counter:=number_of_tacts ;
 else
      synch := '0';
      if Miss_Flag='1'
         and counter = number_of_tacts then
          Counter:=0;
          Synch:='1';
          ReadEnable := '1';
      end if;
   if Clk='1' and Counter<Number_Of_Tacts
      and Miss_Flag='1' then
      synch := '0';
      Counter:=Counter+1;
   end if;

   if Clk='1' and Counter=Number_Of_Tacts-1
      then
      Data_WE <= '1';
      Synch := '0';
   end if;

   if Counter=Number_Of_Tacts
      and ReadEnable='1' then
      Synch:='1';
      Data_WE <= '0';
      ReadEnable := '0';
   end if;

   if Miss_Flag='0' and counter =
      number_of_tacts and ReadEnable='0' then
       Synch:='1';
       end if;
 end if; -- if Reset='1'

   RE <= ReadEnable;
   SysClk <= synch and Clk;

end process;
end architecture Clock_Controller_A;
```

Счетчик строк кэш-памяти

При использовании алгоритма замещения данных Random в модуле кэш-памяти должен быть предусмотрен блок, генерирующий псевдослучайные значения номеров замещаемых строк. В случае КМУУ фактор случайности определяется случайным формированием сигналов логических условий, которые, в свою очередь, приводят к случайным переходам в ГСА. Таким образом, ситуации кэш-промахов также возникают случайным образом и не могут быть достоверно предсказаны заранее.

По этой причине сам генератор псевдослучайных значений не обязательно должен реализовывать какой-либо псевдослучайный алгоритм генерации. В разработанной модели генератор представляет собой инкрементирующий счетчик, производящий последовательный циклический перебор всех номеров строк кэш-памяти. Важной особенностью счетчика является то, что инкремент его содержимого происходит не только в случае кэш-промаха, а в каждом такте внешней (неуправляемой) синхронизации. При этом псевдослучайность генерации номеров строк обеспечивается асинхронной работой счетчика и схемы формированием значений сигналов логических условий.

Внешнее описание счетчика:

```
entity counter is
generic (R_Line: natural;
          N_Lines: natural);
port (Reset: in bit;
      Clock: in bit;
      Line: out natural range 0 to N_Lines-1);
end entity counter;
```

Внутренне описание:

```
architecture counter_A of counter is
begin
process (Reset, Clock)
variable value: natural range 0 to N_Lines-1;
begin
if (Reset='1') then
    value := 0;
    end if;
if (clock='1') and (Reset='0') then
   if value = N_Lines-1 then
      value := 0;
   else
      value := value + 1;
   end if;
end if;
```

```
Line <= value;
end process;
end architecture counter_A;
```

Блок верхнего уровня иерархии

Рассмотренные структурные элементы модуля кэш-памяти должны быть объединены в блоке более высокого уровня иерархии подобно рассмотренным выше блокам КМУУ.

Внешнее описание блока:

```
entity CACHE is
generic (R_Adr:       natural;
         Word_Size:   natural;
         N_Words:     natural;
         R_Line:      natural;
         R_Word:      natural;
         N_Lines:     natural;
         Delay:       natural);

port (ADR: in bit_vector (1 to R_Adr);
      Clk: in bit;
      SysClk: out bit;
      In_Bus: in bit_vector (1 to
                Word_Size*N_Words);
      Cache_Out_Bus: out bit_vector (1 to
                Word_Size-1);
      y0: out bit;
      RE: out bit;
      Reset: in bit);
end entity CACHE;
```

Внутреннее описание блока:

```
architecture CACHE_A of CACHE is
-- ------------------------------------
component Cache_Data_Block is of Cache of
Microinstructions
generic (Word_size: natural;
         R_Line: natural;
         R_Word: natural;
         N_Words: natural;
         N_Lines: natural);
port (In_Bus: in bit_vector (1 to
                Word_Size*N_Words);
Str: in natural range 0 to N_Lines-1;
Word: in bit_vector (1 to R_Word);
Data_WE: in bit;
Out_Bus: out bit_vector (1 to Word_Size));
end component Cache_Data_Block;
-- ------------------------------------
component Hit_Checker is
generic (R_Adr:     natural;
        R_Line:    natural;
        R_Word:    natural;
        N_Lines:   natural;
        N_Words:   natural);
port (Strr: in natural range 0 to N_Lines-1;
      Tag: in bit_vector (1 to R_Adr-R_Word);
      Word: in bit_vector (1 to R_Word);
      Reset: in bit;
      Out_string: out natural range
                0 to N_Lines-1;
      Miss_Flag: out bit);
end component Hit_Checker;
-- ------------------------------------
component Clock_Controller is
generic (Delay: natural);
port (Miss_Flag: in bit;
      Clk: in bit;
      Reset: in bit;
      RE: out bit;
      Data_WE: out bit;
      SysClk: out bit);
end component Clock_Controller;
-- ------------------------------------
component counter is
generic (R_Line: natural;
         N_Lines: natural);
port (Reset: in bit;
      Clock: in bit;
     Line: out natural range 0 to N_Lines-1);
end component counter;
-- ------------------------------------
signal Miss: bit;
signal Data_WE: bit;
signal Word_Bus: bit_vector (1 to R_Word);
signal TAG_Bus: bit_vector (1 to R_Adr-
R_Word);
signal Cache_Out_Line: bit_vector (1 to
Word_Size);
signal Used_Line: natural range 0 to N_Lines-
1;
signal Replaced_Line: natural range 0 to
N_Lines-1;
begin

Tag_Bus <= Adr (1 to R_Adr-R_Word);
Word_Bus <= Adr (R_Adr-R_Word+1 to R_Adr);

Cache1: component Cache_Data_Block
generic map (Word_Size, R_Line, R_Word,
             N_Words, N_Lines)
port map (In_Bus, Used_Line, Word_Bus,
          Data_WE, Cache_Out_Line);

Cache2: component Hit_Checker
generic map (R_Adr, R_Line, R_Word, N_Lines,
            N_Words)
port map (Replaced_Line, Tag_Bus, Word_Bus,
          Reset, Used_Line, Miss);

Cache3: component Clock_Controller
generic map (Delay)
port map (Miss, Clk, Reset, RE, Data_WE,
          SysClk);

Cache4: component counter
generic map (R_Line, N_Lines)
port map (Reset, Clk, Replaced_Line);


Cache_Out_Bus<=Cache_Out_Line(1 to Word_Size-
1);
y0 <= Cache_Out_Line(Word_Size);

end architecture CACHE_A;
```

Основные результаты моделирования

Как показали проведенные авторами исследования, стиль описания разработанной модели позволяет выполнить ее синтез на базе ПЛИС. Для этого в разделе Generic VHDL-описания блока CMCU следует задать фактические значения настроечных констант. Также в описании схемы адресации булевы выражения должны быть сформированы для конкретного алгоритма управления (в настоящей работе приведены в сокращении).

В процессе исследований в качестве кристалла для синтеза была выбрана микросхема V600FG680 серии VIRTEX фирмы XILINX [8]. Экспериментально проверено, что площадь

кристалла и количество контактов ввода-вывода достаточны для реализации рассмотренной модели. При увеличении сложности схемы по причине увеличения параметров схемы КМУУ или модуля кэш-памяти, либо по причине размещения на кристалле дополнительных узлов (например, операционного автомата) может быть выбрана любая подходящая микросхема данного класса [6-8].

Для синтеза устройства на базе разработанной VHDL-модели в качестве примера выбрана тестовая ГСА, содержащая 27 операторных вершин, разделенных на 10 ОЛЦ и имеющая 4 логических условия.

Пусть значения настроечных констант блока CMCU соответствуют табл. 1.

Таблица 1. Значения настроечных констант блока CMCU

Константа	Значение
R_T	2
R_Tau	4
R_Adr	6
R_MI	3
N_MI	4
N_Lines	4
R_Word	2
R_Line	2
Delay	5
R_LC	4

Синтезированная схема КМУУ с разделением кодов и кэш-памятью микрокоманд, построенная при данных значениях, обладает параметрами, приведенными в табл. 2. Параметры указаны в терминах использованной программы синтеза и имплементации Xilinx ISE [6]. В колонке перед наклонной чертой указано значение для полученной схемы, после наклонной черты – предельно допустимое значение для выбранной микросхемы ПЛИС.

Таблица 2. Основные характеристики проекта

Параметр	Значение
Number of Slice Latches	67 / 13824
Number of 4 input LUTs	224 / 13824
Number of occupied Slices	143 / 6912
Number of bonded IOBs	26 / 512
Total equivalent gate count for design	2018
Additional JTAG gate count for IOBs	1296

Заключение

Анализ полученных данных показывает, что разработанная VHDL-модель КМУУ с разделением кодов и кэш-памятью микрокоманд позволяет получить как технические характеристики проекта, так и, при необходимости, синтезированное устройство.

В плане дальнейших исследований с использованием полученной модели авторы видят анализ аппаратурных затрат в схеме КМУУ при различных характеристиках реализуемого алгоритма управления и используемого модуля кэш-памяти.

Литература

1. Баркалов А. А, Палагин А.В. Синтез микропрограммных устройств управления. – Киев: ИК НАН Украины, 1997. – 136 с.

2. Баркалов А.А. Синтез устройств управления на программируемы логических устройствах. – Донецк: ДНТУ, 2002. – 262 с.

3. Баркалов А.А., Ковалев С.А., Бабаков Р.М., Николаенко Д.В. Организация композиционных микропрограммных устройств управления с разделением кодов и кэш-памятью // Искусственный интеллект. – 2007. – №3. – С. 135-138.

4. Проектирование цифровых систем с использованием языка VHDL: Уч. пособие / В. В. Семенец, И. В. Хаханова, В. И. Хаханов. – Харьков: ХНУРЭ, 2003. – 492 с.

5. Тарасов. И.Е. Разработка цифровых устройств на основе ПЛИС Xilinx с применением языка VHDL. – М.: Горячая Линия – Телеком, 2005. – 256 с.

6. Зотов В.Ю.. Проектирование цифровых устройств на основе ПЛИС фирмы XILINX в САПР WebPACK ISE. – М.: Горячая Линия – Телеком, 2003. – 624 с.

7. Кузелин М.О., Кнышев Д.А., Зотов В.Ю. Современные семейства ПЛИС фирмы XILINX. Справочное пособие. – М.: «Горячая Линия – Телеком», 2004. – 440 с.

8. Programmable Logic Data Book. – Xilinx. – 2000. – 463 p.

Поступила в редколлегию 02.03.2009