PicoBlaze — семейство восьмиразрядных микропроцессорных ядер,

реализуемых на основе ПЛИС фирмы Xilinx

Этой публикацией мы открываем цикл статей, в которых рассматриваются встраиваемые микропроцессорные модули (ядра), предназначенные для применения в составе проектов, реализуемых на основе ПЛИС фирмы Xilinx.

Валерий Зотов

walerry@euro.ru

Современные семейства программируемых логических интегральных схем (ПЛИС) предоставляют широкие возможности для реализации проектируемого устройства на базе одного кристалла. Это обусловлено, в первую очередь, внедрением новых технологий производства, позволяющих значительно увеличить объем логических и трассировочных ресурсов кристаллов. При выполнении разработки «системы на кристалле» (System-on-Chip), реализующей в одном корпусе ПЛИС функции процессора и периферийных устройств, целесообразно использовать готовые микропроцессорные ядра. Их применение позволяет ощутимо сократить длительность цикла проектирования разрабатываемой системы.

Микропроцессорные ядра, предоставляемые фирмой Xilinx

Фирма Xilinx наряду с выпуском новых семейств ПЛИС, отличающихся высокими техническими характеристиками, предоставляет разработчикам готовые отлаженные модули микропроцессорных ядер с различной архитектурой. В рамках программы AllianceCORE пользователям доступны ядра с архитектурой широко применяемых микропроцессоров различных производителей, таких, как Z80 фирмы Zilog, PIC семейств 125x, 1655x, 165x фирмы Microchip, 8051 и др. Кроме того, фирма Xilinx предлагает семейства ядер с оригинальной архитектурой, оптимизированной для реализации на основе ПЛИС различных серий (Soft Processor). К числу таких ядер относятся семейства PicoBlaze и MicroBlaze. Общие характеристики микропроцессорных ядер этих семейств представлены в таблице.

Таблица. Общие характеристики микропроцессорных ядер семейств PicoBlaze и MicroBlaze

Тип микропроцессорного ядра	Архитектура	Разрядность шин	Производительность	Объем требуемых ресурсов	Поддерживаемые семейства ПЛИС	Средства разработки
PicoBlaze	RISC 8 разрядов	8 разрядов данных и адресов	40 MIPS 116 МГц	35 CLBs	Spartan-II, Spartan-IIE, Virtex, Virtex-E, Virtex-II, CoolRunner-II	Ассемблер
MicroBlaze	RISC 32 разряда	32 разряда данных и команд	100 D-MIPS 150 МГц	225 CLBs	Spartan-II, Spartan-IIE, Virtex, Virtex-E, Virtex-II, Virtex-II Pro	Development Kits (компилятор, ассемблер, отладчик)

Семейство микропроцессорных ядер PicoBlaze является свободно распространяемым (бесплатным). Для их получения следует обратиться к Web-странице http://www.xilinx.com/ipcenter/processor_central/picoblaze/index.htm. Чтобы скачать все необходимые материалы, относящиеся к конкретному ядру этого семейства, необходимо выполнить процедуру бесплатной регистрации.

Изучение элементов семейства PicoBlaze начнем с ядра, предназначенного для применения в системах, выполняемых на основе ПЛИС серий Spartan-II, Spartan-IIE, Virtex, Virtex-E.

Основные характеристики микропроцессорного ядра PicoBlaze, реализуемого на основе ПЛИС семейств Spartan-II, Spartan-IIE, Virtex, Virtex-E

Отличительными особенностями микропроцессорного ядра PicoBlaze, предназначенного для применения в ПЛИС семейства Spartan-II, Spartan-IIE, Virtex, Virtex-E являются:

- гибкая архитектура с раздельными шинами данных и команд;
- разрядность шины данных — 8 бит;
- разрядность шины адресов — 8 бит;
- разрядность шины команд — 16 бит;
- восьмиразрядное арифметическо-логическое устройство (АЛУ), реализующее логические функции, операции сложения, вычитания и сдвига;
- поддержка 49 команд;
- постоянное время выполнения всех команд — два машинных цикла;
- шестнадцать регистров общего назначения;
- пятнадцатиуровневый стек;

- возможность поддержки до 256 входных и выходных портов;
- встроенное ППЗУ микропрограмм, выполненное на основе блочной памяти ПЛИС Block SelectRAM, объем которого составляет 256×16 разрядов;
- поддержка прямого, косвенного и непосредственного режимов адресации;
- реализация в виде модулей исходного описания на языке VHDL с учетом оптимального размещения и трассировки в кристалле соответствующего семейства;
- минимальный объем ресурсов кристалла, используемый для реализации микропроцессорного ядра, позволяет без труда разместить в кристалле другие функциональные модули проектируемой системы, включая интерфейсы ввода-вывода (в ПЛИС семейства Spartan-IIE ядро PicoBlaze занимает всего лишь 76 секций (slices), что составляет 9% от полного объема логических ресурсов кристалла XC2S50E и 2,5% от логической емкости ПЛИС XC2S300E);
- интерфейс микропроцессорного ядра обеспечивает оптимальное его сопряжение с периферийными модулями, реализуемыми на основе свободных логических ресурсов кристалла;
- достаточно высокая производительность, достигающая 40 MIPS (в зависимости от типа используемого кристалла);
- наличие ассемблера, формирующего необходимые файлы различного формата, обеспечивает высокую скорость и наглядность процесса разработки программ;
- наличие входа сброса (инициализации), позволяющего перевести микропроцессор в начальное состояние;
- полная совместимость компонентов ядра со всей линией средств разработки проектов и программирования ПЛИС ISE (Integrated Synthesis Environment) фирмы Xilinx (WebPACK ISE [1–6, 10], Base ISE, Foundation ISE и Alliance ISE);
- возможность моделирования ядра в составе разрабатываемых проектов с помощью системы ModelSim XE [7–9], которая включена в состав свободно распространяемого пакета САПР WebPACK ISE.

Структура проекта микропроцессорного ядра PicoBlaze, реализуемого на основе ПЛИС семейств Spartan-II, Spartan-IIE, Virtex, Virtex-E

Структура микропроцессорного ядра PicoBlaze, предназначенного для применения в составе проектов, выполняемых на основе ПЛИС семейства Spartan-II, Spartan-IIE, Virtex, Virtex-E, показана на рис. 1. Она включает в себя программную память и модуль центрального процессорного устройства (ЦПУ) (исполнительного устройства) PicoBlaze. В качестве ППЗУ микропрограмм используется один модуль блочной памяти Block SelectRAM кристаллов указанных семейств, объем которого составляет 4 кбит. Этот модуль блочной памяти конфигурируется как однопортовое ОЗУ с организацией 256×16 разрядов.

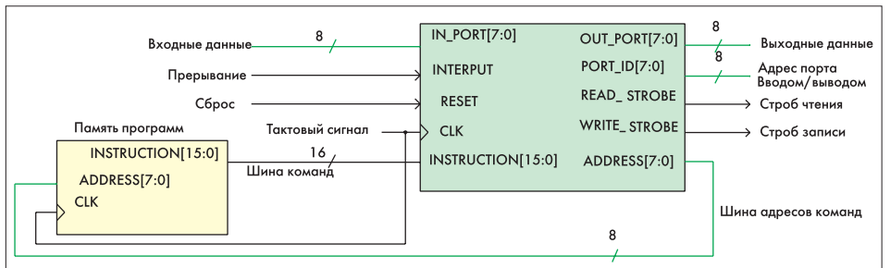

Рис. 1. Структура микропроцессорного ядра PicoBlaze, реализуемого на основе ПЛИС семейств Spartan-II, Spartan-IIE, Virtex, Virtex-E

Микропроцессорное ядро PicoBlaze предоставляется пользователям в виде архива, в котором содержится комплект файлов, включающий в себя необходимые модули VHDL-описаний, ассемблер, тестовые примеры, иллюстрирующие использование компонентов ядра. Кроме того, в состав комплекта входят дополнительные файлы, которые могут быть использованы для включения в состав проекта универсального асинхронного приемопередатчика UART (Universal Asynchronous Receiver-Transmitter).

Каждый элемент структуры микропроцессорного ядра PicoBlaze (рис. 1) представлен в форме соответствующего компонента, выполненного в виде макроса с относительным размещением. Функции исполнительного модуля реализует компонент KCPSM (Constant (k) Coded Programmable State Machine). Описание этого модуля на языке VHDL содержится в файле kcpsm.vhd. Для применения компонента KCPSM в качестве элемента проектируемой системы необходимо, прежде всего, включить в состав ее описания выражения декларации, которые выглядят следующим образом.

```
component kcpsm
    Port (
        address : out std_logic_vector(7 downto 0);
        instruction : in std_logic_vector(15 downto 0);
        port_id : out std_logic_vector(7 downto 0);
        write_strobe : out std_logic;
        out_port : out std_logic_vector(7 downto 0);
        read_strobe : out std_logic;
        in_port : in std_logic_vector(7 downto 0);
        interrupt : in std_logic;
        reset : in std_logic;
        clk : in std_logic
        );
end component;
```

В приведенных выражениях декларации используется следующая система обозначений интерфейсных цепей компонента. Идентификаторы clk, reset и interrupt описывают соответственно входы тактового сигнала, сброса и прерывания, а read_strobe и write_strobe соответствуют выходам сигналов, сопровождающих операции чтения и записи данных в порты ввода-вывода. Векторы in_port и out_port представляют соответственно входную и выходную шины данных. Вектор address соответствует выходной шине адресов команд, instruction — входной шине команд, port_id — выходной шине адресов портов ввода-вывода.

Для создания одного экземпляра компонента kcpsm, представляющего модуль ЦПУ, необходимо в состав структурного описания архитектуры проектируемой системы включить следующий оператор:

```
inst_name_processor: kcpsm
    port map(
        address => address_signal,
        instruction => instruction_signal,
        port_id => port_id_signal,
        write_strobe => write_strobe_signal,
        out_port => out_port_signal,
        read_strobe => read_strobe_signal,
        in_port => in_port_signal,
        interrupt => interrupt_signal,
        reset => reset_signal,
        clk => clk_signal
        );
```

Вместо идентификатора *inst_name_processor* следует задать метку, определяющую позиционное обозначение экземпляра компонента. Кроме того, названия сигналов, указанные в операторе, должны соответствовать именам сигналов, которые используются в описании проектируемого устройства.

Таким же способом, в виде компонента с названием prog_rom, описывается модуль программной памяти в составе разрабатываемой системы. Вначале выполняется декларация этого компонента, которая выглядит следующим образом.

```
component prog_rom
    Port (
        address : in std_logic_vector(7 downto 0);
        instruction : out std_logic_vector(15 downto 0);
        clk : in std_logic
        );
end component;
```

В описании интерфейса компонента prog_rom идентификатор clk соответствует входу тактового сигнала, address — входной шине адресов команд, instruction — выходной шине команд. Создание экземпляра компонента, представляющего программную память, выполняется следующим оператором.

```
inst_name_program: prog_rom
    port map(
        address => address_signal,
        instruction => instruction_signal,
        clk => clk_signal
        );
```

При практическом использовании этого компонента следует учитывать, что его название должно совпадать с идентификатором файла, имеющего расширение PSM, в котором записан исходный текст программы.

Помимо отдельных компонентов микропроцессорного ядра PicoBlaze пользователю также доступно VHDL-описание объекта EMBEDDED_KCPSM, который представляет собой подсистему, состоящую из модуля ЦПУ и подключенной к нему программной памяти. Описание этого объекта является наглядной иллюстрацией практического использования

компонентов микропроцессорного ядра, и поэтому далее полностью приводится его текст.

```
-- EMBEDDED_KCPSM.VHD
--
----------------------------------------------------------------------
--
-- Standard IEEE libraries
--
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
--
----------------------------------------------------------------------
--
entity embedded_kcpsm is
   Port (      port_id : out std_logic_vector(7 downto 0);
         write_strobe : out std_logic;
          read_strobe : out std_logic;
             out_port : out std_logic_vector(7 downto 0);
              in_port : in std_logic_vector(7 downto 0);
            interrupt : in std_logic;
                reset : in std_logic;
                  clk : in std_logic);
end embedded_kcpsm;
--
----------------------------------------------------------------------
--
-- Start of test achitecture
--
architecture connectivity of embedded_kcpsm is
--
----------------------------------------------------------------------
--
-- declaration of KCPSM
--
 component kcpsm
   Port (      address : out std_logic_vector(7 downto 0);
           instruction : in std_logic_vector(15 downto 0);
               port_id : out std_logic_vector(7 downto 0);
         write_strobe : out std_logic;
             out_port : out std_logic_vector(7 downto 0);
          read_strobe : out std_logic;
              in_port : in std_logic_vector(7 downto 0);
            interrupt : in std_logic;
                reset : in std_logic;
                  clk : in std_logic);
 end component;
--
-- declaration of program ROM
--
 component prog_rom
   Port (    address : in std_logic_vector(7 downto 0);
           instruction : out std_logic_vector(15 downto 0);
                   clk : in std_logic);
 end component;
--
----------------------------------------------------------------------
--
-- Signals used to connect KCPSM to program ROM
--
signal      address : std_logic_vector(7 downto 0);
signal instruction : std_logic_vector(15 downto 0);
--
----------------------------------------------------------------------
--
-- Start of test circuit description
--
begin

 processor: kcpsm
   port map(    address => address,
            instruction => instruction,
                port_id => port_id,
           write_strobe => write_strobe,
               out_port => out_port,
            r ead_strobe => read_strobe,
                in_port => in_port,
              interrupt => interrupt,
                  reset => reset,
                    clk => clk);

 program: prog_rom
   port map(    address => address,
            instruction => instruction,
                    clk => clk);

end connectivity;

----------------------------------------------------------------------
--
-- END OF FILE EMBEDDED_KCPSM.VHD
```

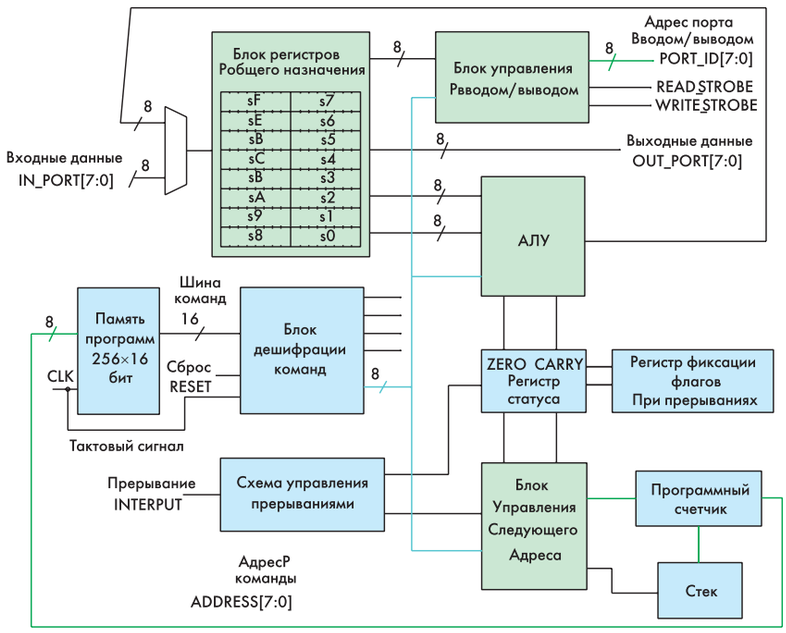

Рис. 2. Архитектура микропроцессорного ядра PicoBlaze, реализуемого на основе ПЛИС семейств Spartan-II, Spartan-IIE, Virtex, Virtex-E

Приведенное описание объекта EMBEDDED_KCPSM имеет стандартную структуру исходного модуля, выполненного с использованием языка VHDL. В начале описания указаны ссылки на используемые стандартные библиотеки и пакеты. Затем следуют операторы, описывающие интерфейс объекта. Далее приводится структурное описание архитектуры объекта.

Объект EMBEDDED_KCPSM может применяться как автономно, так и в виде модуля, входящего в состав разрабатываемой системы. В последнем случае модуль EMBEDDED_KCPSM используется в форме одноименного компонента. Для декларации компонента *embedded_kcpsm* в составе VHDL-представления проектируемой системы используется следующая конструкция.

```
component embedded_kcpsm
      Port (
               port_id : out std_logic_vector(7 downto 0);
               write_strobe : out std_logic;
               read_strobe : out std_logic;
               out_port : out std_logic_vector(7 downto 0);
               in_port : in std_logic_vector(7 downto 0);
               interrupt : in std_logic;
               reset : in std_logic;
               clk : in std_logic
               );
end embedded_kcpsm;
```

В описании интерфейса компонента *embedded_kcpsm* используется та же система обозначений, что и для компонента *kcpsm*. Конкретный экземпляр рассматриваемого компонента в структурном описании проектируемой системы создается с помощью оператора, текст которого представлен ниже. При практическом использовании этого оператора следует заменить метку *inst_name_embedded_processor* идентификатором, который, как правило, соответствует позиционному обозначению экземпляра компонента в составе описания проекта. При описании внешних связей компонента embedded_kcpsm должны быть указаны названия соответствующих сигналов, используемых в проекте.

```
inst_name_embedded_processor: embedded_kcpsm
      port map(
               port_id => port_id_signal,
               write_strobe => write_strobe_signal,
               read_strobe => read_strobe_signal,
               out_port => out_port_signal,
               in_port => in_port_signal,
               interrupt => interrupt_ signal,
               reset => reset_signal,
               clk => clk_signal
               );
```

Архитектура микропроцессорного ядра PicoBlaze, реализуемого на основе ПЛИС семейств Spartan-II, Spartan-IIE, Virtex, Virtex-E

Архитектура микропроцессорного ядра PicoBlaze основана на концепции раздельных шин данных и команд (гарвардская архитектура). Такая организация магистралей процессора позволяет добиться высокой скорости выполнения операций. Структурное представление архитектуры микропроцессорного ядра PicoBlaze, предназначенного для применения в кристаллах семейств Spartan-II, Spartan-IIE, Virtex, Virtex-E, показано на рис. 2.

Основными элементами архитектуры микропроцессорного ядра PicoBlaze, реализуемого на основе ПЛИС семейств Spartan-II, Spartan-IIE, Virtex, Virtex-E, являются:

- восьмиразрядное АЛУ;
- блок регистров общего назначения;
- регистр статуса (флагов);
- регистр фиксации флагов при выполнении прерываний;
- программный счетчик (счетчик команд);
- блок управления вводом-выводом;
- стек;
- схема управления прерываниями;
- блок управления выбором адреса следующей команды;

- блок дешифрации команд;
- память программ.

Блок АЛУ выполняет логические и арифметические операции над восьмиразрядными операндами. В качестве операндов может использоваться содержимое любого из шестнадцати регистров общего назначения, а также константы, указанные непосредственно в тексте команды. Результат выполнения операции заносится в тот же регистр, который являлся источником первого операнда.

Регистр статуса содержит значения флагов (признаков), формируемых блоком АЛУ при выполнении арифметических и логических операций. Этот регистр содержит два разряда. В первый разряд записывается состояние флага нулевого результата ZERO Flag, а во второй — значение флага переноса (заема) CARRY Flag. Флаг нулевого результата ZERO Flag переключается в установленное состояние (состояние высокого логического уровня) в случае, если результатом арифметической или логической операции является нуль. При получении других результатов этот флаг сбрасывается в состояние низкого логического уровня. Флаг переноса CARRY Flag устанавливается в том случае, если в результате операции сложения из самого старшего разряда происходит перенос или заем при выполнении вычитания. Анализ состояния флагов, записанных в соответствующие разряды регистра статуса, производится при выполнении условных команд. Кроме того, регистр статуса принимает участие в процессе выполнения операций сдвига.

Регистр фиксации флагов предназначен для сохранения текущих значений признаков результата операции АЛУ, записанных в регистр статуса, перед выполнением процедуры обслуживания прерывания.

Блок регистров общего назначения содержит шестнадцать восьмиразрядных регистров, обозначаемых по порядку s0... sF. Эти регистры предназначены для хранения данных, поступающих из входных портов ввода-вывода, операндов и результатов выполнения операций. Все регистры имеют одинаковый статус (полностью равноправны). Любой из них может использоваться в качестве аккумулятора.

Схема управления прерываниями формирует комбинацию управляющих сигналов, необходимых для выполнения процедуры обработки прерываний. Микропроцессорное ядро PicoBlaze изначально содержит единственную цепь сигнала прерывания. Для создания комплексной системы прерываний, поддерживающей комбинацию нескольких линий сигналов, следует использовать дополнительную логику, которая реализуется на основе свободных ресурсов кристалла.

Блок управления вводом-выводом предназначен для формирования адреса входного или выходного порта PORT_ID[7:0], к которому производится обращение, а также сигналов WRITE_STROBE и READ_STROBE, указывающих тип выполняемой операции (записи или чтения в указанный порт). Адрес порта ввода-вывода может задаваться в программе в виде абсолютного значения или в форме ссылки на один из регистров общего назначения, содержимое которого определяет номер порта. Во время выполнения операции ввода INPUT данные из входного порта, номер которого определяет комбинация значений сигналов в шине адреса порта PORT_ID[7:0], могут быть загружены в любой из шестнадцати регистров общего назначения. Эта процедура сопровождается формированием импульсного сигнала READ_STROBE. При осуществлении операции вывода OUTPUT информация из любого регистра общего назначения может быть передана в выходной порт, номер которого указывает комбинация значений сигналов в шине адреса порта PORT_ID[7:0]. Выполнение операции вывода сопровождается стробом записи WRITE_STROBE.

Память программ представляет собой запоминающее устройство, реализуемое в виде однопортового ОЗУ на основе блочной памяти кристалла Block SelectRAM, в котором хранится последовательность выполняемых команд.

Блок дешифрации команд на основании данных, поступающих с выходов программной памяти, формирует совокупность управляющих сигналов, необходимых для выполнения соответствующей операции, которые подаются на все блоки микропроцессорного ядра.

Управление последовательностью выполнения команд в составе программы осуществляется с помощью программного счетчика. Сигналы на его выходах образуют адрес выборки следующей команды. Эти сигналы по шине адресов команд передаются на адресные входы программной памяти. Режим работы программного счетчика (счет или загрузка) определяется состоянием сигналов, формируемых в блоке управления выбором следующего адреса команды. В основном режиме при отсутствии прерываний, команд переходов, вызовов и возврата из подпрограмм содержимое программного счетчика автоматически увеличивается на единицу при исполнении текущей операции. Таким образом, реализуется последовательная выборка и выполнение команд программы.

В процессе выполнения команд перехода в программный счетчик производится загрузка значения, соответствующего адресу, по которому осуществляется передача управления в программе. После исполнения команды перехода программный счетчик продолжает работу в инкрементном режиме, но, уже начиная с нового значения, которое было записано при ее выполнении.

При вызове подпрограмм также осуществляется принудительное изменение содержимого программного счетчика. В него загружается значение, которое соответствует начальному адресу выполняемой подпрограммы. Перед этим прежнее содержимое программного счетчика заносится в стек (в регистр, адрес которого определяется текущим значением указателя стека). В процессе выполнения команд подпрограммы производится последовательное инкрементирование нового содержимого программного счетчика. После завершения подпрограммы, при выполнении команды возврата в основную программу, из стека извлекается последнее записанное значение адреса, которое увеличивается на единицу и загружается в программный счетчик. Тем самым осуществляется переход к выполнению очередной команды, следующей после вызова подпрограммы. Так как глубина стека составляет пятнадцать уровней, то он может хранить одновременно до пятнадцати адресов возврата. Поэтому в процессе исполнения подпрограмм допускаются вложенные вызовы других подпрограмм. При очередном вложенном вызове подпрограммы содержимое программного счетчика заносится в следующий регистр стека. Процесс выполнения каждой вложенной подпрограммы завершается «выталкиванием» из стека последнего записанного значения, которое, увеличиваясь на единицу, заносится в программный счетчик.

Процедура обработки прерывания выполняется подобно вызову подпрограммы, но имеются несколько отличий. Одно из них заключается в том, что при наличии активного уровня сигнала на входе прерывания в программный счетчик загружается адрес вектора соответствующей процедуры обработки. При этом кроме записи в стек адреса текущей исполняемой команды производится сохранение состояния признаков результата операции АЛУ (содержимого регистра статуса) на момент прерывания в регистре фиксации флагов. После завершения процесса обработки прерывания происходит автоматическое восстановление значений флагов в регистре статуса.

В начальный момент времени функционирования микропроцессорного ядра, а также при подаче внешнего сигнала сброса, выходы программного счетчика устанавливаются в нулевое состояние (низкого логического уровня). Таким образом управление передается первой команде программы.

Команды, поддерживаемые микропроцессорным ядром PicoBlaze

Совокупность команд, поддерживаемых микропроцессорным ядром PicoBlaze, можно разбить по функциональному признаку на шесть групп. К первой группе относятся команды, управляющие последовательностью выполнения операций в программе, и команды обработки подпрограмм. В эту группу входят команды переходов JUMP, вызова подпрограмм CALL и возврата из подпрограмм RETURN. Каждая из перечисленных команд представлена как в безусловной форме, так и в условном формате (то есть может исполняться только при выполнении определенного условия).

Вторую группу образуют логические команды. Эти команды выполняют поразрядные операции «Логическое И» (поразрядное умножение) AND, «Логическое ИЛИ» (поразрядное сложение) OR, «Исключающее ИЛИ» XOR. В качестве операндов могут выступать содержимое любого регистра общего назначения и константа, указанная в тексте команды, а также содержимое двух регистров блока РОН. К этой же группе относится команда загрузки значения в регистр общего назначения

LOAD. С помощью этой команды в выбранный регистр может записываться константа, указанная в тексте команды, или содержимое другого регистра блока POH.

В третью группу входят арифметические команды. Команды ADD и ADDCY предназначены для получения суммы двух операндов без учета и с учетом входного переноса соответственно. Операция вычитания также может выполняться без учета и с учетом заема с помощью команд SUB и SUBCY соответственно. Значение одного из операндов, участвующих в арифметических операциях, содержится в регистре общего назначения, номер которого указан в тексте команды. В качестве второго операнда может использоваться значение, заданное непосредственно в коде команды, или содержимое другого регистра блока POH.

Четвертую группу составляют команды сдвига. Они позволяют реализовать операции логического (арифметического) или циклического сдвига данных, записанных в регистре общего назначения, номер которого указан в тексте команды. В процессе выполнения одной команды сдвиг производится на один разряд. Сдвиг может осуществляться как влево (в сторону младшего разряда), так и вправо (в сторону старшего разряда). Операции циклического сдвига могут выполняться с участием разряда переноса регистра статуса.

Пятая группа включает в себя команды ввода-вывода. Эти команды предназначены для организации обмена данными между регистрами, входящими в блок POH, и портами ввода-вывода. Выполнение операций чтения данных из входного порта в регистр общего назначения (INPUT) и записи информации из регистра в выходной порт (OUTPUT) рассмотрено выше, в разделе описания архитектуры микропроцессорного ядра PicoBlaze.

Последняя группа объединяет команды, используемые для обслуживания прерываний. В нее входят команды разрешения ENABLE INTERRUPT и запрета прерываний DISABLE INTERRUPT.

Более подробно синтаксис и выполнение команд микропроцессорного ядра PicoBlaze, предназначенного для применения в проектах, реализуемых на основе ПЛИС семейств Spartan-II, Spartan-IIE, Virtex, Virtex-E, будут рассмотрены в следующей публикации цикла.

Программирование микропроцессорного ядра PicoBlaze

Исходный текст программы в кодах ассемблера записывается в файл с расширением psm, название которого должно содержать не более восьми символов. Из этого файла ассемблер формирует VHDL-описание начального содержимого блочного ОЗУ, исполняющего роль программной памяти. Результирующий файл с расширением vhd включается в состав разрабатываемого проекта и используется далее в процессе синтеза, моделирования, размещения и трассировки в кристалле.

Ассемблер выполнен в виде DOS-приложения с названием KCPSM.EXE. Этот файл должен располагаться в рабочем каталоге проекта вместе с шаблонами ROM_form.vhd и ROM_form.coe. Для активизации ассемблера используется командная строка, которая имеет формат kcpsm *<название_файла_программы>[.psm]*

Инициализация программной памяти микропроцессорного ядра с помощью набора данных, которые соответствуют машинным кодам разработанной программы, выполняется автоматически в процессе загрузки конфигурационной последовательности проекта в кристалл.

Для практического изучения функционирования и применения рассматриваемого микропроцессорного ядра PicoBlaze можно воспользоваться универсальным инструментальным модулем *SET-StarterKit*, подробное описание которого публиковалось ранее в нашем журнале [11].

Литература

1. Зотов В. WebPACK ISE — свободно распространяемый пакет проектирования цифровых устройств на базе ПЛИС фирмы Xilinx // Компоненты и технологии. 2001. № 6.
2. Зотов В. WebPACK ISE: Интегрированная среда разработки конфигурации и программирования ПЛИС фирмы Xilinx. Создание нового проекта // Компоненты и технологии. 2001. № 7.
3. Зотов В. Схемотехнический редактор пакета WebPACK ISE. Создание принципиальных схем и символов // Компоненты и технологии. 2001. № 8.
4. Зотов В. Синтез проектов, реализуемых на базе ПЛИС FPGA фирмы Xilinx, в САПР WebPACK ISE // Компоненты и технологии. 2002. № 3.
5. Зотов В. Реализация проектов на базе ПЛИС семейств FPGA фирмы Xilinx в САПР WebPACK ISE // Компоненты и технологии. 2002. № 4.
6. Зотов В. Конфигурирование ПЛИС семейств FPGA фирмы Xilinx в САПР WebPACK ISE // Компоненты и технологии. 2002. № 5.
7. Зотов В. ModelSim — система HDL-моделирования цифровых устройств // Компоненты и технологии. 2002. № 6.
8. Зотов В. Функциональное моделирование цифровых устройств, проектируемых на базе ПЛИС фирмы Xilinx в среде САПР WebPACK ISE // Компоненты и технологии. 2002. № 7.
9. Зотов В. Временное моделирование цифровых устройств, проектируемых на базе ПЛИС фирмы Xilinx в среде САПР WebPACK ISE // Компоненты и технологии. 2002. № 8.
10. Зотов В. Новая версия свободно распространяемого пакета проектирования WebPACK ISE фирмы Xilinx // Компоненты и технологии. 2003. № 1.
11. Зотов В. Инструментальный комплект *SET-StarterKit* для практического освоения методов проектирования цифровых устройств на основе ПЛИС семейств FPGA фирмы Xilinx // Компоненты и технологии. 2002. № 9.

Система команд микропроцессорного ядра PicoBlaze, реализуемого на основе ПЛИС семейств Spartan-II, Spartan-IIE, Virtex, Virtex-E

Продолжаем ознакомление с микропроцессорным модулем PicoBlaze, предназначенным для применения в проектах, реализуемых на основе ПЛИС семейств Spartan-II, Spartan-IIE, Virtex, Virtex-E. Настоящая статья представляет подробное описание системы команд этого ядра.

Валерий Зотов

walerry@euro.ru

Команды управления последовательностью выполнения операций в программе

Команды переходов *JUMP* позволяют осуществить передачу управления в программе по указанному адресу. Эти команды обычно используются для организации ветвления и формирования циклов в ходе выполнения алгоритма программы. Микропроцессорное ядро PicoBlaze поддерживает пять модификаций инструкции переходов *JUMP*: одну безусловную и четыре условных. Форматы различных вариантов команд переходов *JUMP* представлены в таблице 1. Единственным параметром команд переходов является однобайтовая константа, значение которой определяет адрес перехода. При мнемонической форме записи команд этот параметр задается в виде двух шестнадцатеричных символов.

Таблица 1. Форматы команд переходов

Поле кода операции								Поле адреса перехода								Мнемоника	Выполняемая операция
1	0	0	0	X	X	0	1	A	A	A	A	A	A	A	A	JUMP aa	Безусловный переход
1	0	0	1	0	0	0	1	A	A	A	A	A	A	A	A	JUMP Z,aa	Переход при условии, что флаг ZERO Flag находится в установленном состоянии
1	0	0	1	0	1	0	1	A	A	A	A	A	A	A	A	JUMP NZ,aa	Переход при условии, что флаг ZERO Flag находится в сброшенном состоянии
1	0	0	1	1	0	0	1	A	A	A	A	A	A	A	A	JUMP C,aa	Переход при условии, что флаг CARRY Flag находится в установленном состоянии
1	0	0	1	1	1	0	1	A	A	A	A	A	A	A	A	JUMP NC,aa	Переход при условии, что флаг CARRY Flag находится в сброшенном состоянии
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	Номер разряда микрокоманды	

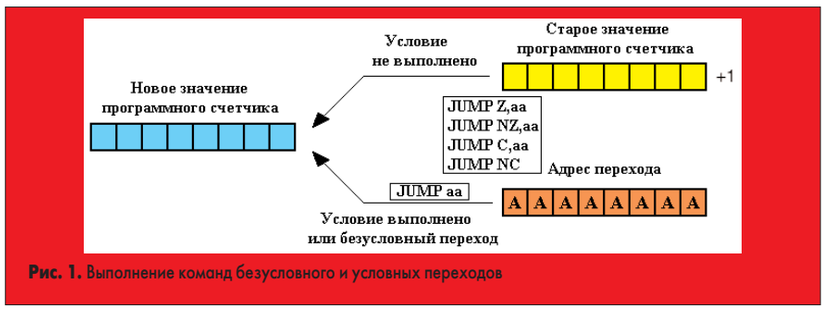

Рис. 1. Выполнение команд безусловного и условных переходов

Процесс выполнения команд переходов иллюстрирует рис. 1. При исполнении операции безусловной передачи управления *JUMP aa* в программный счетчик записывается значение адреса перехода *aa*. При условных переходах загрузка в программный счетчик нового адреса, указанного в команде, производится только при выполнении соответствующего условия. Если заданное условие не выполнено, то программный счетчик продолжает работу в инкрементном режиме, то есть его прежнее содержимое увеличивается на единицу. В процессе реализации команд безусловного и условных переходов состояние флагов регистра статуса не изменяется.

Команда *JUMP Z,aa* инициирует переход к выполнению инструкции, расположенной по указанному адресу, при условии, что флаг нулевого результата ZERO Flag находится в состоянии логической единицы. При исполнении команды *JUMP NZ,aa* управление передается инструкции с указанным адресом только в случае, если флаг нулевого результата ZERO Flag установлен в состояние логического нуля. Команда *JUMP C,aa* загружает новое значение адреса в программный счетчик при условии, что состояние флага переноса/займа CARRY Flag соответствует значению логической единицы. С помощью команды *JUMP NC,aa* осуществляется переход к выполнению инструкции с указанным адресом, если флаг переноса/займа CARRY Flag находится в сброшенном состоянии (логического нуля).

Команды вызова подпрограмм *CALL* позволяют передать управление по адресу, который соответствует первой инструкции вызываемой процедуры. Обращение к подпрограмме может происходить как в безусловном порядке, так и при выполнении заданного условия. Форматы безусловного и условных вариантов команд вызова подпрограмм *CALL* приведены в таблице 2. Адрес вызываемой процедуры задается в том же виде, что и адрес в командах перехода *JUMP*. Команды безусловного и условных об-

Таблица 2. Форматы команд вызова подпрограмм

Поле кода операции								Поле адреса подпрограммы								Мнемоника	Выполняемая операция
1	0	0	0	X	X	1	1	A	A	A	A	A	A	A	A	CALL aa	Безусловный вызов подпрограммы
1	0	0	1	0	0	1	1	A	A	A	A	A	A	A	A	CALL Z,aa	Вызов подпрограммы при условии, что флаг ZERO Flag находится в установленном состоянии
1	0	0	1	0	1	1	1	A	A	A	A	A	A	A	A	CALL NZ,aa	Вызов подпрограммы при условии, что флаг ZERO Flag находится в сброшенном состоянии
1	0	0	1	1	0	1	1	A	A	A	A	A	A	A	A	CALL C,aa	Вызов подпрограммы при условии, что флаг CARRY Flag находится в установленном состоянии
1	0	0	1	1	1	1	1	A	A	A	A	A	A	A	A	CALL NC,aa	Вызов подпрограммы при условии, что флаг CARRY Flag находится в сброшенном состоянии
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	Номер разряда микрокоманды	

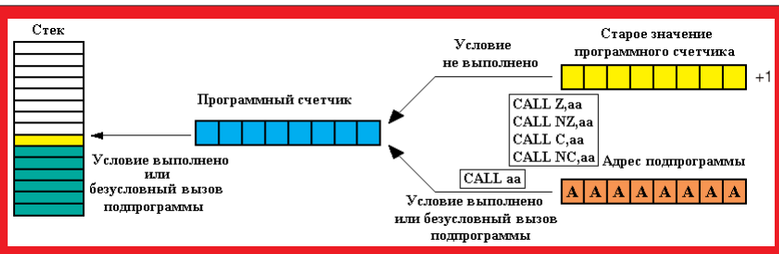

Рис. 2. Выполнение команд безусловного и условных вызовов подпрограмм

ращений к подпрограммам не влияют на состояние флагов регистра статуса.

На рис. 2 отображена последовательность действий, которые производятся при различных вызовах подпрограмм. Выполнение команды безусловного обращения к подпрограмме *CALL aa* начинается с записи в стек текущего содержимого программного счетчика (то есть адреса исполняемой инструкции вызова процедуры). Сохраненное значение используется впоследствии, при завершении подпрограммы и возврате в основную программу или подпрограмму, из которой вызывалась текущая процедура. Затем в программный счетчик заносится значение адреса, указанное в команде. Это значение определяет начальный адрес блока памяти, в котором хранится вызываемая подпрограмма.

Микропроцессорным ядром PicoBlaze поддерживается четыре варианта команд условного обращения к подпрограмме. Исполнение команды условного вызова подпрограммы начинается с проверки выполнения соответствующего условия. Если это условие выполнено, то далее производится последовательность операций, рассмотренная выше для команды безусловного обращения к подпрограмме (рис. 2). В случае невыполнения заданного условия содержимое программного счетчика увеличивается на единицу, указывая тем самым адрес следующей команды основной программы. Команда *CALL Z,aa* передает управление подпрограмме, расположенной по указанному адресу, если флаг нулевого результата ZERO Flag установлен в состояние логической единицы. С помощью команды *CALL NZ,aa* осуществляется вызов подпрограммы при условии, что флаг нулевого результата ZERO Flag находится в состоянии логического нуля. Команда *CALL C,aa* загружает в программный счетчик указанный адрес подпрограммы, если состояние флага переноса/займа CARRY Flag соответствует значению логической единицы. При исполнении команды *CALL NC,aa* обращение к подпрограмме производится только при условии, что флаг переноса/займа CARRY Flag находится в сброшенном состоянии.

Команды возврата из подпрограммы *RETURN* предназначены для передачи управления основной программе или подпрограмме, из которой вызывалась текущая процедура. Эта операция может осуществляться как в безусловной форме, так и в зависимости от выполнения заданного условия.

Таблица 3 представляет форматы безусловного и условных вариантов команд возврата из подпрограммы *RETURN*.

Процесс выполнения команд возврата из подпрограмм в условной и безусловной форме показан на рис. 3. Безусловное завершение выполняемой подпрограммы и передача управления основной программе или подпрограмме, из которой производилось обращение к этой процедуре, осуществляется с помощью команды *RETURN*. При этом из стека извлекается последнее записанное значение, увеличивается на единицу и загружается в программный счетчик. Таким образом, в программный счетчик заносится адрес очередной команды основной программы (или подпрограммы при вложенном вызове процедуры), следующей после инструкции обращения к подпрограмме.

При реализации команды условного возврата из подпрограммы, прежде всего, осуществляется контроль выполнения соответствующего условия. При получении положительного результата проверки далее выполняется та же последовательность действий, что и при безусловном возврате. Если условие не выполнено, то текущее содержимое программного счетчика увеличивается на единицу. Таким образом, новое значение программного счетчика в этом случае соответствует адресу следующей команды текущей исполняемой подпрограммы.

Команда *RETURN Z* возвращает управление вызывающей программе или подпрограмме в случае, если флаг нулевого результата ZERO Flag принимает значение логической единицы. С помощью команды *RETURN NZ* осуществляется возврат из подпрограммы при условии, что флаг нулевого результата ZERO Flag сброшен в состояние логического нуля. Команда *RETURN C* выполняет операцию возврата из подпрограммы, если состояние флага переноса/займа CARRY Flag соответствует значению логической единицы. При исполнении команды *RETURN NC* управление передается вызывающей программе или подпрограмме только при условии, что флаг переноса/займа CARRY Flag находится в сброшенном состоянии. При выполнении команд возврата из подпрограммы состояние флагов регистра статуса не изменяется.

Таблица 3. Форматы команд безусловного и условного возврата из подпрограммы

Поле кода операции								Поле адреса								Мнемоника	Выполняемая операция
1	0	0	0	X	X	0	0	0	0	0	0	0	0	0	0	RETURN	Безусловный возврат из подпрограммы
1	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	RETURN Z	Возврат из подпрограммы при условии, что флаг ZERO Flag находится в установленном состоянии
1	0	0	1	0	1	0	0	0	0	0	0	0	0	0	0	RETURN NZ	Возврат из подпрограммы при условии, что флаг ZERO Flag находится в сброшенном состоянии
1	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	RETURN C	Возврат из подпрограммы при условии, что флаг CARRY Flag находится в установленном состоянии
1	0	0	1	1	1	0	0	0	0	0	0	0	0	0	0	RETURN NC	Возврат из подпрограммы при условии, что флаг CARRY Flag находится в сброшенном состоянии
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	Номер разряда микрокоманды	

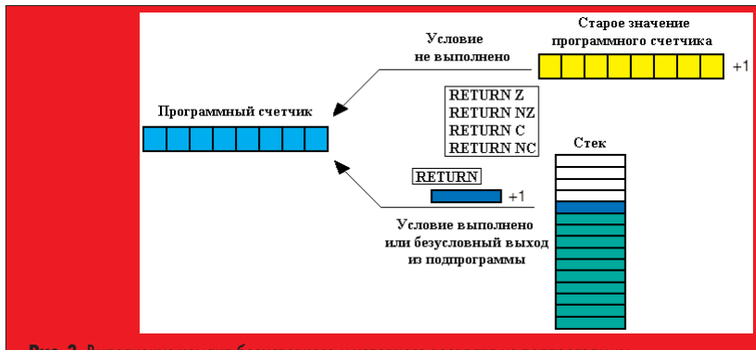

Рис. 3. Выполнение команд безусловного и условного возврата из подпрограммы

Таблица 4. Форматы команд поразрядных операций «Логическое И»

Поле кода операции				Поле номера регистра				Поле константы								Мнемоника	Выполняемая операция
0	0	0	1	n	n	n	n	K	K	K	K	K	K	K	K	AND sN, kk	Поразрядное «Логическое И» содержимого регистра sN и константы kk
Поле префикса кода операции				**Поле номера первого регистра**				**Поле номера второго регистра**				**Поле кода операции**				**Мнемоника**	**Выполняемая операция**
1	1	0	0	n	n	n	n	m	m	m	m	0	0	0	1	AND sN,sM	Поразрядное «Логическое И» содержимого регистров sN и sM
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	Номер разряда микрокоманды	

Таблица 5. Форматы команд поразрядных операций «Логическое ИЛИ»

Поле кода операции				Поле номера регистра				Поле константы								Мнемоника	Выполняемая операция
0	0	1	0	n	n	n	n	K	K	K	K	K	K	K	K	OR sN, kk	Поразрядное «Логическое ИЛИ» содержимого регистра sN и константы kk
Поле префикса кода операции				**Поле номера первого регистра**				**Поле номера второго регистра**				**Поле кода операции**				**Мнемоника**	**Выполняемая операция**
1	1	0	0	n	n	n	n	m	m	m	m	0	0	1	0	OR sN,sM	Поразрядное «Логическое ИЛИ» содержимого регистров sN и sM
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	Номер разряда микрокоманды	

Таблица 6. Форматы команд поразрядных операций «Исключающее ИЛИ»

Поле кода операции				Поле номера регистра				Поле константы								Мнемоника	Выполняемая операция
0	0	1	1	n	n	n	n	K	K	K	K	K	K	K	K	XOR sN, kk	Поразрядное «Исключающее ИЛИ» содержимого регистра sN и константы kk
Поле префикса кода операции				**Поле номера первого регистра**				**Поле номера второго регистра**				**Поле кода операции**				**Мнемоника**	**Выполняемая операция**
1	1	0	0	n	n	n	n	m	m	m	m	0	0	1	1	XOR sN,sM	Поразрядное «Исключающее ИЛИ» содержимого регистров sN и sM
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	Номер разряда микрокоманды	

Группа логических команд

В командах, входящих в эту группу, используются два параметра, которые определяют значения операндов. Первым операндом всегда является содержимое регистра общего назначения, номер которого *N* указан в тексте команды. В качестве номера регистра *N* может использоваться любое число в диапазоне от 0 до 15, которое при мнемонической записи команд задается в шестнадцатеричном формате (0 — F). В этот же регистр записывается результат выполненной операции. В качестве второго операнда используется либо константа, значение которой указывается непосредственно в коде команды, либо содержимое другого регистра общего назначения. При мнемонической форме записи команд значение константы задается в виде двух шестнадцатеричных символов.

При выполнении поразрядных логических операций флаг переноса/займа CARRY Flag всегда сбрасывается в состояние логического нуля. Значение флага ZERO Flag определяется полученным результатом. Если все разряды результирующего слова принимают значение логического нуля, то флаг нулевого результата ZERO Flag устанавливается в состояние, соответствующее логической единице. В противном случае, если хотя бы в одном разряде результата присутствует единичное значение, то флаг нулевого результата ZERO Flag переключается в сброшенное состояние.

Форматы команд поразрядных операций «Логическое И» (поразрядное умножение) *AND* представлены в таблице 4.

Команда *AND sN,kk* выполняет операцию «Логическое И» над соответствующими разрядами содержимого регистра с номером *N* и константы *kk*. Для поразрядного умножения содержимого двух регистров общего назначения с номерами *N* и *M* предназначена команда *AND sN,sM*. Выполнение этих команд иллюстрирует рис. 4.

Операции поразрядного сложения двух операндов осуществляются с помощью инструкций *OR*, форматы которых определены в таблице 5.

Команда *OR sN,kk* выполняет поразрядную операцию «Логическое ИЛИ» содержимого регистра с номером *N* и константы *kk* (рис. 5). Поразрядное сложение содержимого двух регистров общего назначения с номерами *N* и *M* реализует команда *OR sN,sM*.

Форматы инструкций *XOR*, предназначенных для выполнения поразрядной операции «Исключающее ИЛИ», представлены в таблице 6.

Инструкция *XOR sN,kk* выполняет поразрядную операцию «Исключающее ИЛИ», в которой участвует содержимое регистра с номером *N* и константа *kk*, значение которой указывается непосредственно в команде. Команда *XOR sN,sM* осуществляет аналогичную операцию, операндами которой является содержимое двух регистров общего назначения с номерами *N* и *M*. Выполнение поразрядных операций «Исключающее ИЛИ» демонстрирует рис. 6.

Инструкция *LOAD*, которая также относится к группе логических команд, предназначена для загрузки данных в выбранный регистр общего назначения (табл. 7). Источником данных может служить содержимое любого регистра общего назначения или константа, указанная непосредственно в коде команды.

Команда *LOAD sN,kk* записывает значение константы *kk*, указанной в инструкции, в регистр общего назначения с номером *N*. Команда *LOAD sN,sM* производит загрузку данных

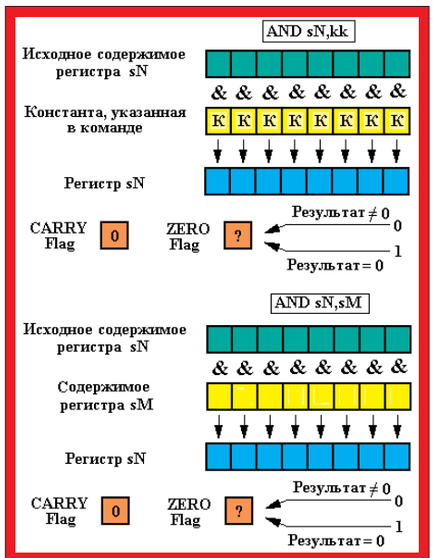

Рис. 4. Выполнение поразрядных операций «Логическое И»

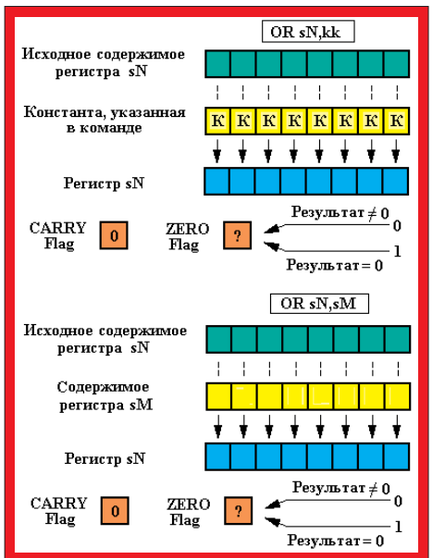

Рис. 5. Выполнение поразрядных операций «Логическое ИЛИ»

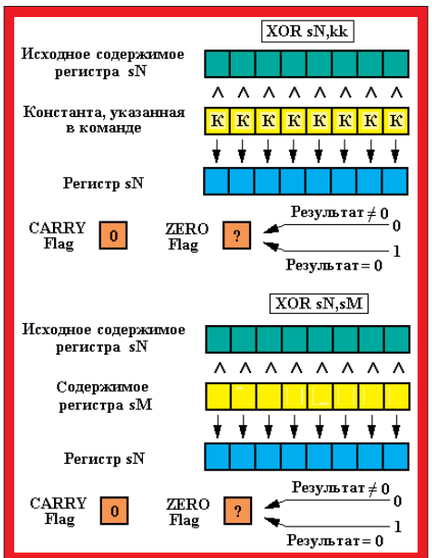

Рис. 6. Выполнение поразрядных операций «Исключающее ИЛИ»

Таблица 7. Форматы инструкции загрузки данных в регистр общего назначения

Поле кода операции				Поле номера регистра				Поле константы								Мнемоника	Выполняемая операция
0	0	0	0	n	n	n	n	K	K	K	K	K	K	K	K	LOAD sN, kk	Загрузка константы kk в регистр sN
Поле префикса кода операции				**Поле номера первого регистра**				**Поле номера второго регистра**				**Поле кода операции**				**Мнемоника**	**Выполняемая операция**
1	1	0	0	n	n	n	n	m	m	m	m	0	0	0	0	LOAD sN,sM	Загрузка содержимого регистра sM в регистр sN
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	Номер разряда микрокоманды	

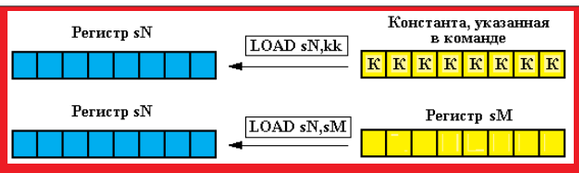

Рис. 7. Выполнение операций загрузки данных в регистр общего назначения

Таблица 8. Форматы команд сложения двух операндов без учета переноса

Поле кода операции				Поле номера регистра				Поле константы								Мнемоника	Выполняемая операция
0	1	0	0	n	n	n	n	K	K	K	K	K	K	K	K	ADD sN, kk	Сложение содержимого регистра sN и константы kk
Поле префикса кода операции				**Поле номера первого регистра**				**Поле номера второго регистра**				**Поле кода операции**				**Мнемоника**	**Выполняемая операция**
1	1	0	0	n	n	n	n	m	m	m	m	0	1	0	0	ADD sN,sM	Сложение содержимого регистров sN и sM
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	Номер разряда микрокоманды	

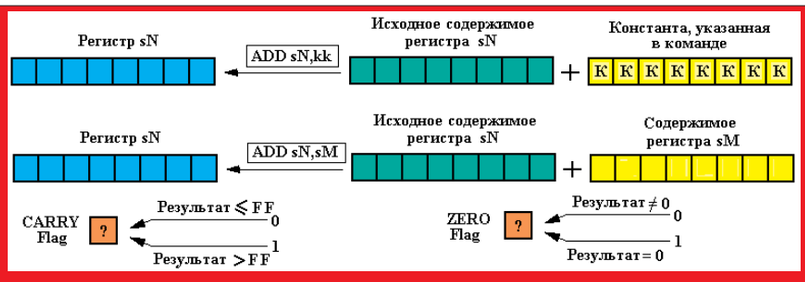

Рис. 8. Выполнение операций сложения двух операндов без учета переноса

из регистра с номером *M* в регистр с номером *N*. Операции загрузки данных в регистр не оказывают влияния на флаги регистра статуса. Рис. 7 отображает процесс выполнения операции загрузки данных в регистр общего назначения.

Группа арифметических команд

Арифметические команды микропроцессорного ядра PicoBlaze обеспечивают выполнение операций сложения и вычитания над двумя восьмиразрядными операндами. Первым параметром в арифметической команде всегда является номер одного из регистров общего назначения, содержимое которого используется в качестве первого операнда. Этот же регистр используется для записи результата выполненной операции. В роли второго операнда может выступать константа, значение которой задается непосредственно в коде команды, или содержимое любого регистра общего назначения. При выполнении арифметических операций состояние флагов регистра статуса зависит от полученного результата.

Форматы команд сложения двух операндов без учета переноса представлены в таблице 8.

Команда *ADD sN,kk* выполняет сложение содержимого регистра с номером *N* и константы *kk*. Для получения суммы содержимого двух регистров общего назначения с номерами *N* и *M* предназначена команда *ADD sN,sM*. На рис. 8 показана последовательность действий, выполняемых при сложении двух операндов без учета переноса.

Сложение двух операндов с учетом переноса, полученного при выполнении предыдущей операции, производится с помощью инструкции *ADDCY*. Форматы двух вариантов этой команды приведены в таблице 9.

Команда *ADDCY sN,kk* суммирует содержимое регистра с номером *N* с константой *kk* с учетом состояния флага переноса перед выполнением этой операции. Сложение содержимого двух регистров общего назначения с учетом значения флага переноса осуществляется с помощью команды *ADDCY sN,sM* (рис. 9).

После исполнения команд сложения (с учетом и без учета переноса) флаг нулевого результата ZERO Flag переключается в установленное состояние, если сумма равна нулю. При получении результата, отличного от нулевого, флаг ZERO Flag сбрасывается в состояние логического нуля. Флаг переноса/займа CARRY Flag принимает значение логической единицы, если сумма операндов превышает значение 255 (FF). В противном случае этот флаг переключается в сброшенное состояние.

Операция вычитания без учета займа выполняется с помощью инструкции *SUB*, форматы двух вариантов которой представлены в таблице 10.

Команда *SUB sN,kk* позволяет вычесть значение константы *kk* из содержимого регистра с номером *N*. Для вычисления разности содержимого двух регистров общего назначения с номерами *N* и *M* следует использовать команду *SUB sN,sM*. Процесс выполнения команд вычитания без учета займа отображен на рис. 10.

Для вычисления разности двух операндов с учетом значения займа, образовавшегося при выполнении предыдущей операции, предназначена инструкция *SUBCY*, форматы вариантов которой приведены в таблице 11.

Команда *SUBCY sN,kk* выполняет вычитание значений константы *kk* и флага займа CARRY Flag из содержимого регистра с номером *N*. Вычисление разности содержимого двух регистров общего назначения с номерами *N* и *M* с участием значения флага займа CARRY Flag производится с помощью команды *SUBCY sN,sM* (рис. 11).

Таблица 9. Форматы команд сложения двух операндов с учетом переноса

Поле кода операции				Поле номера регистра				Поле константы								Мнемоника	Выполняемая операция
0	1	0	1	n	n	n	n	K	K	K	K	K	K	K	K	ADDCY sN, kk	Сложение содержимого регистра sN и константы kk с учетом переноса
Поле префикса кода операции				**Поле номера первого регистра**				**Поле номера второго регистра**				**Поле кода операции**				**Мнемоника**	**Выполняемая операция**
1	1	0	0	n	n	n	n	m	m	m	m	0	1	0	1	ADDCY sN,sM	Сложение содержимого регистров sN и sM с учетом переноса
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	Номер разряда микрокоманды	

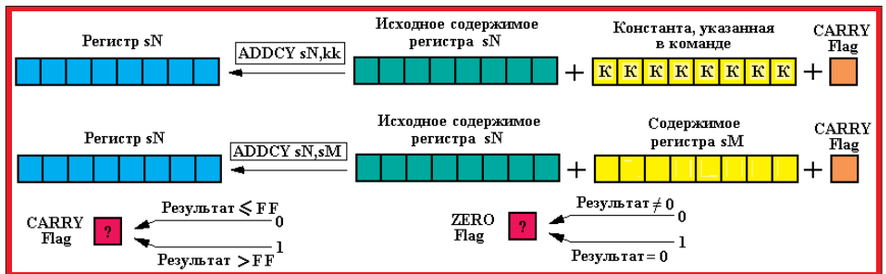

Рис. 9. Осуществление операций сложения двух операндов с учетом переноса

Таблица 10. Форматы команд вычитания без учета займа

Поле кода операции				Поле номера регистра				Поле константы								Мнемоника	Выполняемая операция
0	1	1	0	n	n	n	n	K	K	K	K	K	K	K	K	SUB sN, kk	Вычитание из содержимого регистра sN константы kk
Поле префикса кода операции				**Поле номера первого регистра**				**Поле номера второго регистра**				**Поле кода операции**				**Мнемоника**	**Выполняемая операция**
1	1	0	0	n	n	n	n	m	m	m	m	0	1	1	0	SUB sN,sM	Вычитание содержимого регистра sM из содержимого регистра sN
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	Номер разряда микрокоманды	

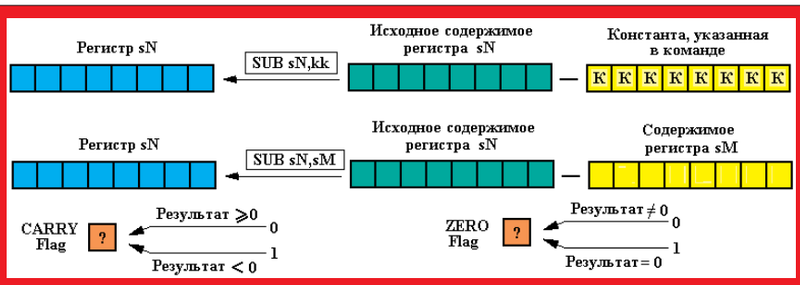

Рис. 10. Вычисление разности двух операндов без учета займа

Таблица 11. Форматы инструкций вычитания с учетом займа

Поле кода операции				Поле номера регистра				Поле константы								Мнемоника	Выполняемая операция
0	1	1	1	n	n	n	n	K	K	K	K	K	K	K	K	SUBCY sN, kk	Вычитание из содержимого регистра sN константы kk с учетом заема
Поле префикса кода операции				**Поле номера первого регистра**				**Поле номера второго регистра**				**Поле кода операции**				**Мнемоника**	**Выполняемая операция**
1	1	0	0	n	n	n	n	m	m	m	m	0	1	1	1	SUBCY sN,sM	Вычитание содержимого регистра sM из содержимого регистра sN с учетом заема
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	Номер разряда микрокоманды	

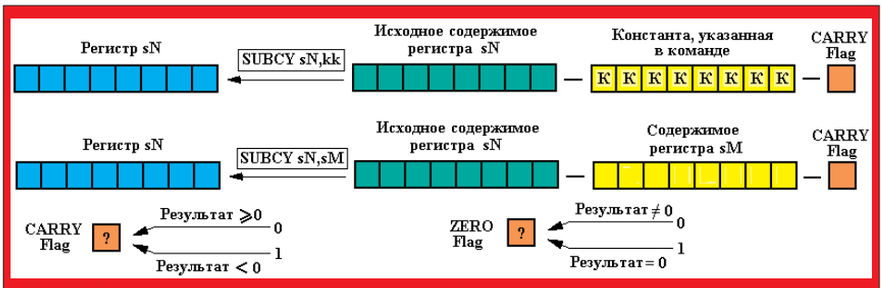

Рис. 11. Вычисление разности двух операндов с учетом займа

Таблица 12. Форматы команд логического или циклического сдвига данных

Поле кода операции				Поле номера регистра				Поле направления сдвига				Поле типа сдвига			Мнемоника	Выполняемая операция
1	1	0	1	n	n	n	n	0	0	0	0	1	1	0	SR0 sN	Логический сдвиг содержимого регистра sN вправо на один разряд с записью 0
1	1	0	1	n	n	n	n	0	0	0	0	1	1	1	SR1 sN	Логический сдвиг содержимого регистра sN вправо на один разряд с записью 1
1	1	0	1	n	n	n	n	0	0	0	0	1	0	1	0	SRX sN	Логический сдвиг содержимого регистра sN вправо с сохранением последнего разряда
1	1	0	1	n	n	n	n	0	0	0	0	1	0	0	0	SRA sN	Циклический сдвиг содержимого регистра sN вправо через разряд переноса/заема
1	1	0	1	n	n	n	n	0	0	0	0	1	1	0	0	RR sN	Циклический сдвиг содержимого регистра sN вправо без участия бита переноса
1	1	0	1	n	n	n	n	0	0	0	0	0	1	1	0	SL0 sN	Логический сдвиг содержимого регистра sN влево на один разряд с записью 0
1	1	0	1	n	n	n	n	0	0	0	0	0	1	1	1	SL1 sN	Логический сдвиг содержимого регистра sN влево на один разряд с записью 1
1	1	0	1	n	n	n	n	0	0	0	0	0	0	1	0	SLX sN	Логический сдвиг содержимого регистра sN влево с сохранением последнего разряда
1	1	0	1	n	n	n	n	0	0	0	0	0	0	0	0	SLA sN	Циклический сдвиг содержимого регистра sN влево через разряд переноса/заема
1	1	0	1	n	n	n	n	0	0	0	0	0	1	0	0	RL sN	Циклический сдвиг содержимого регистра sN влево без участия бита переноса
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	Номер разряда микрокоманды	

После выполнения команд вычитания (с учетом и без учета займа) состояние флагов регистра статуса изменяется в зависимости от полученного результата. Флаг нулевого результата ZERO Flag переключается в установленное состояние, если разность равна нулю. При получении результата, отличного от нулевого, флаг ZERO Flag сбрасывается в состояние логического нуля. Флаг переноса/займа CARRY Flag принимает значение логической единицы при отрицательном значении разности операндов. В противном случае этот флаг переключается в сброшенное состояние.

Команды сдвига данных

Команды этой группы предназначены для выполнения операций логического (арифметического) или циклического сдвига данных, хранящихся в регистре общего назначения с указанным номером, на один разряд. Форматы инструкций сдвига данных представлены в таблице 12.

Команды *SR0 sN* и *SR1 sN* выполняют логический сдвиг содержимого регистра с номером sN вправо на один разряд с записью соответственно нулевого и единичного бита в «освободившийся» (крайний левый) разряд. Команда *SRX sN* позволяет осуществить операцию логического сдвига данных вправо в регистре sN с сохранением состояния последнего (крайнего левого) разряда. На рис. 12 в наглядной форме показан процесс выполнения различных сдвиговых операций, поддерживаемых микропроцессорным ядром PicoBlaze.

Циклический сдвиг данных в регистре общего назначения на один разряд вправо, выполняемый через разряд переноса/займа CARRY Flag регистра статуса, осуществляется с помощью команды *SRA sN*. Операция циклического сдвига вправо без участия бита переноса реализуется с помощью команды *RR sN*.

Команды *SL0 sN* и *SL1 sN* предназначены для выполнения операций логического сдвига данных влево с занесением нулевого и единичного бита соответственно в «освободившийся» (крайний правый) разряд используемого регистра sN. Для логического сдвига влево содержимого регистра sN с сохранением значения последнего (крайнего правого) разряда предусмотрена команда *SLX sN*.

Операции циклического сдвига данных влево в регистре sN выполняются с помощью инструкций *SLA sN* и *RL sN*. При использовании команды *SLA sN* циклический сдвиг производится через разряд переноса/займа CARRY Flag регистра статуса. Команда *RL sN* позволяет осуществить циклический сдвиг данных влево без участия разряда переноса регистра статуса.

При выполнении сдвиговых операций в разряд регистра статуса, который содержит значение флага переноса/займа CARRY Flag, записывается бит данных, «выталкиваемый» из регистра общего назначения в процессе сдвига. Состояние флага нулевого результата ZERO Flag зависит от значения содержимого используемого регистра общего назначения после выполнения операции сдвига. Если во всех разрядах этого регистра присутствуют нулевые

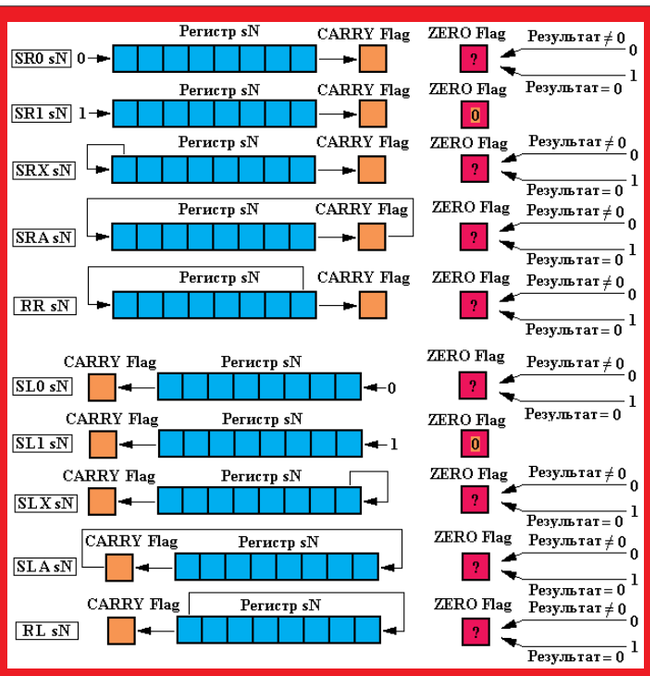

Рис. 12. Выполнение операций логического или циклического сдвига данных

значения, то флаг ZERO Flag переключается в установленное состояние. В противном случае флаг нулевого результата ZERO Flag сбрасывается.

Команды ввода-вывода

Команды ввода-вывода предназначены для организации чтения данных из входного порта в заданный регистр общего назначения и передачи информации из указанного регистра в выходной порт. Инструкции ввода-вывода включают в себя два параметра. Значение первого параметра определяет номер регистра общего назначения, используемого в качестве приемника данных при операции ввода или в качестве источника при выполнении операции вывода. Второй параметр позволяет указать адрес порта ввода-вывода, к которому производится обращение. Адресация ко входным и выходным портам в инструкциях ввода-вывода может осуществляться с помощью восьмиразрядной константы, значение которой задается непосредственно в команде, или содержимого регистра общего назначения с указанным номером. Форматы команд ввода-вывода приведены в таблице 13.

Команда *INPUT sN, kk* выполняет передачу данных из входного порта с адресом *kk* в регистр общего назначения с номером *sN*. Чтение входных данных из порта ввода-вывода, адрес которого указывает содержимое регистра *sM*, производится при использовании инструкции *INPUT sX,(sY)*.

Передача содержимого регистра общего назначения с номером *sN* в выходной порт с адресом *kk* осуществляется с помощью команды *OUTPUT sX, kk*. Для записи выходных данных из регистра *sN* в порт, адрес которого определяется содержимым регистра *sM*, предназначена команда *OUTPUT sN,(sM)*. Рис. 13 поясняет выполнение операций ввода-вывода с различными видами адресации.

Команды обслуживания прерываний

В эту группу входят команды возврата из процедуры обработки прерываний и установки режима обслуживания прерываний в программе.

Инструкция возврата из процедуры обслуживания прерывания *RETURNI* предназначена для передачи управления программе, при выполнении которой поступил запрос прерывания. Возврат к выполняемой программе производится в безусловном порядке после завершения процедуры обработки прерывания. Параметры команды возврата из процедуры обслуживания прерывания позволяют также определить дальнейший режим (запрета или разрешения) обработки прерываний. Форматы инструкций обслуживания прерываний представлены в таблице 14.

Команда *RETURNI ENABLE* возвращает управление из процедуры обслуживания прерывания основной программе с разрешением последующих прерываний. Для выхода из процедуры обработки прерывания и установки режима запрета прерывания следует использовать команду *RETURNI DISABLE*. Механизм возврата из процедуры обслуживания прерываний показан на рис. 14. При завершении процесса обработки прерывания в программный счетчик из стека загружается значение адреса, которое было записано последним. Таким образом, в программный счетчик заносится адрес команды, которая выполнялась в момент поступления запроса на прерывание. Кроме того, восстанавливается состояние флагов в регистре статуса, в котором они находились при вызове процедуры обслуживания запроса на прерывание.

Команды *ENABLE INTERRUPT* и *DISABLE INTERRUPT* позволяют изменить режим

Таблица 13. Форматы команд ввода-вывода

Поле кода операции				Поле номера регистра				Поле адреса порта ввода/вывода								Мнемоника	Выполняемая операция
1	0	1	0	n	n	n	n	K	K	K	K	K	K	K	K	INPUT sN, kk	Чтение данных из порта ввода/вывода с адресом kk в регистр sN
1	1	1	0	n	n	n	n	K	K	K	K	K	K	K	K	OUTPUT sN, kk	Запись данных из регистра sN в порт ввода/вывода с адресом kk
Поле кода операции				**Поле номера первого регистра**				**Поле номера второго регистра**				**Нулевые разряды**				**Мнемоника**	**Выполняемая операция**
1	0	1	1	n	n	n	n	m	m	m	m	0	0	0	0	INPUT sN,(sM)	Чтение данных из порта ввода/вывода с адресом, определяемым регистром sM, в регистр sN
1	1	1	1	n	n	n	n	m	m	m	m	0	0	0	0	OUTPUT sN,(sM)	Запись данных из регистра sN в порт с адресом, определяемым регистром sM
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	Номер разряда микрокоманды	

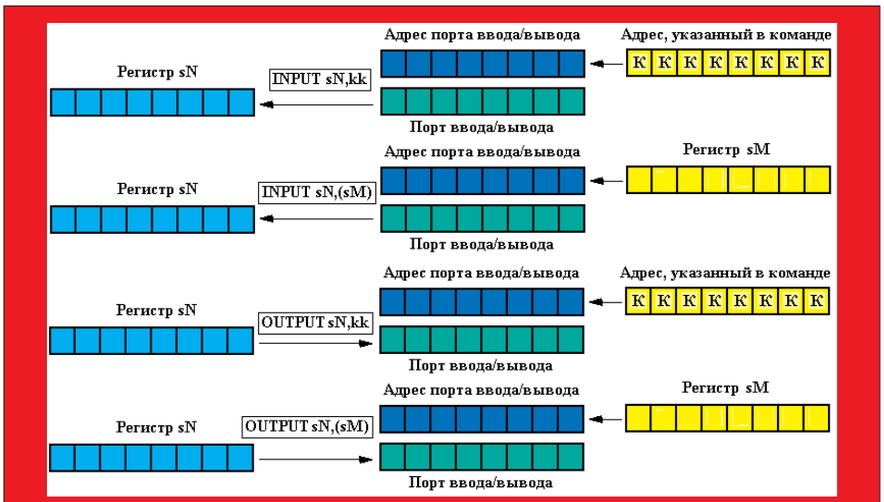

Рис. 13. Выполнение инструкций ввода-вывода с различными видами адресации

Таблица 14. Форматы команд обслуживания прерываний

Поле режима обработки прерываний								Поле кода операции								Мнемоника	Выполняемая операция
1	0	0	0	0	0	0	0	1	1	1	1	0	0	0	0	RETURNI ENABLE	Возврат из процедуры обработки и установка режима запрета прерывания
1	0	0	0	0	0	0	0	1	1	0	1	0	0	0	0	RETURNI DISABLE	Возврат из процедуры обработки и установка режима разрешения прерывания
1	0	0	0	0	0	0	0	0	0	1	1	0	0	0	0	ENABLE INTERRUPT	Установка режима разрешения прерывания
1	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	DISABLE INTERRUPT	Установка режима запрета прерывания
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	Номер разряда микрокоманды	

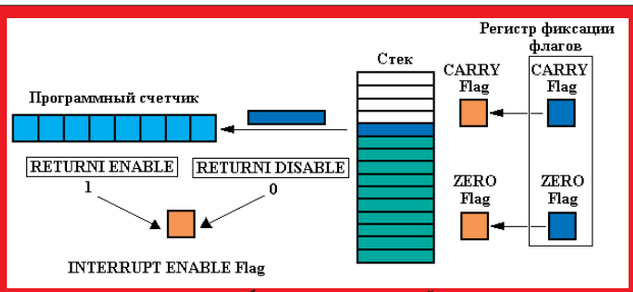

Рис. 14. Выполнение возврата из процедуры обслуживания прерываний

обработки прерываний на любом этапе выполняемой программы. При этом следует учитывать, что до завершения текущей процедуры обслуживания прерывания должен быть установлен режим запрета прерываний.

Заключение

Несмотря на то, что приведенное выше описание системы команд микропроцессорного ядра PicoBlaze относится, прежде всего, к модулю, предназначенному для употребления в проектах, которые реализуются на основе ПЛИС серий Spartan-II, Spartan-IIE, Virtex и Virtex-E, оно в значительной мере применимо и к другим элементам этого семейства. Все рассмотренные команды поддерживаются и остальными версиями микропроцессорного ядра PicoBlaze, различаясь только длиной (разрядностью) и форматом инструкций. Отличительные особенности микропроцессорных модулей PicoBlaze, которые предназначены для использования в кристаллах серий Virtex-II и CoolRunner-II, будут проанализированы в следующих публикациях цикла.

Литература

1. Зотов В. PicoBlaze — семейство восьмиразрядных микропроцессорных ядер, реализуемых на основе ПЛИС фирмы Xilinx // Компоненты и технологии. 2003. № 4.

Особенности микропроцессорного ядра PicoBlaze, предназначенного для применения в проектах, реализуемых на основе ПЛИС семейства Virtex-II

Микропроцессорное ядро PicoBlaze, предназначенное для применения в проектах, выполняемых на базе ПЛИС серии Virtex-II, отличается от других представителей этого семейства наиболее широкими функциональными возможностями. Учитывая, что этот модуль является результатом дальнейшего развития микропроцессорного ядра PicoBlaze, реализуемого на основе кристаллов семейств Spartan-II, Spartan-IIE, Virtex, Virtex-E, то основное внимание в настоящей статье сосредоточено на отличиях его характеристик, архитектуры и системы команд по сравнению с базовым ядром, описание которого приведено в предыдущих статьях [1, 2].

Валерий Зотов

walerry@euro.ru

Основные характеристики микропроцессорного ядра PicoBlaze для Virtex-II

Основные характеристики семейства встраиваемых микропроцессорных ядер PicoBlaze [1] в большинстве своем относятся ко всем представителям этой серии. Однако ядро, предназначенное для применения в кристаллах семейства Virtex-II, имеет ряд отличительных особенностей. К их числу относятся следующие характеристики:

- разрядность шины адресов — 10 бит;
- разрядность шины команд — 18 бит;
- объем блока регистров общего назначения — 32 восьмиразрядных регистра;
- объем встроенного ППЗУ микропрограмм, реализуемого на основе блочной памяти ПЛИС Block SelectRAM, составляет 1024×18 разрядов;
- объем ресурсов кристалла, необходимых для реализации микропроцессорного ядра PicoBlaze в ПЛИС семейства Virtex-II, ограничивается 84 секциями (slices), что составляет 9% от полного объема логических ресурсов кристалла XC2V40E и 0,25% от логической емкости ПЛИС XC2V6000E;
- повышенная производительность, достигающая 40–70 MIPS (в зависимости от типа используемого кристалла).

Структура проекта микропроцессорного ядра PicoBlaze для Virtex-II

Структура встраиваемого микропроцессорного ядра PicoBlaze, предназначенного для использования в составе проектов, реализуемых на базе кристаллов семейства Virtex-II, приведена на рис. 1. Она отличается от структуры, рассмотренной в одной из предыдущих публикаций [1], только типом и параметрами модулей, которые входят в ее состав.

Исполнительный модуль выполнен в форме компонента KCPSM2, представляющего собой макрос с относительным размещением, описание которого на языке VHDL содержится в файле kcpsm2.vhd.

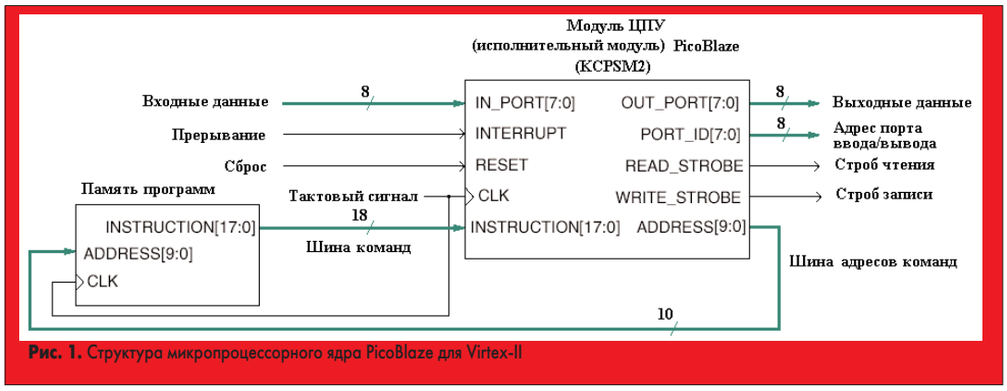

Рис. 1. Структура микропроцессорного ядра PicoBlaze для Virtex-II

Отличия интерфейса этого компонента от макроса, который является основой микропроцессорного ядра PicoBlaze, реализуемого на базе ПЛИС семейств Spartan-II, Spartan-IIE, Virtex, Virtex-E [1], заключаются в размерности векторов, представляющих шины адресов и команд. Выражения декларации компонента KCPSM2 в составе VHDL-описания проектируемой системы выглядят следующим образом.

```
component kcpsm2
    Port (
            address : out std_logic_vector(9 downto 0);
            instruction : in std_logic_vector(17 downto 0);
            port_id : out std_logic_vector(7 downto 0);
            write_strobe : out std_logic;
            out_port : out std_logic_vector(7 downto 0);
            read_strobe : out std_logic;
            in_port : in std_logic_vector(7 downto 0);
            interrupt : in std_logic;
            reset : in std_logic;
            clk : in std_logic
            );
end component;
```

В приведенных здесь и далее выражениях декларации используется система обозначений интерфейсных цепей компонентов микропроцессорного ядра PicoBlaze, подробно описанная ранее [1]. Включение экземпляра компонента, представляющего исполнительный модуль, в состав структурного описания архитектуры проектируемой системы осуществляется с помощью соответствующей конструкции [1] для компонента KCPSM, в которой следует изменить название компонента на kcpsm2.

Функции программной памяти, как и в ПЛИС семейств Spartan-II, Spartan-IIE, Virtex, Virtex-E, выполняет один модуль блочного ОЗУ Block SelectRAM. Но в отличие от кристаллов указанных семейств, информационная емкость одного модуля блочной памяти ПЛИС семейства Virtex-II составляет 18 Кбит, что позволило в четыре раза увеличить максимальный объем загружаемой микропроцессорной программы. Модуль программной памяти представлен в виде компонента с названием prog_rom, который фактически описывает ППЗУ с информационной емкостью 1024×18 разрядов, реализуемое на основе однопортового блочного ОЗУ с аналогичной организацией. Для декларации этого компонента следует воспользоваться конструкцией, которая имеет следующий вид:

```
component prog_rom
    Port (
            address : in std_logic_vector(9 downto 0);
            instruction : out std_logic_vector(17 downto 0);
            clk : in std_logic
            );
end component;
```

Создание экземпляра компонента, представляющего программную память, выполняется с помощью оператора, который используется для аналогичного компонента, рассмотренного в первой статье цикла [1]. При включении выражений декларации и создания экземпляра компонента, соответствующего программной памяти, в состав описания проектируемой системы следует вместо идентификатора prog_rom указать название компонента, совпадающее с идентификатором файла, в котором должен быть записан исходный текст программы. Необходимость такой замены объясняется тем, что ассемблер формирует описание содержимого ППЗУ программ на языке VHDL в виде файла с расширением vhd, название которого совпадает с идентификатором файла, содержащего исходный текст программы.

Сопряжение основных компонентов микропроцессорного ядра PicoBlaze демонстрирует VHDL-описание объекта EMBEDDED_KCPSM2, структуру которого составляет исполнительный модуль и подключенный к нему модуль программной памяти. Ниже приведен полный текст описания объекта EMBEDDED_KCPSM2, имеющий ту же структуру, что и описание объекта EMBEDDED_KCPSM [1].

```
-- EMBEDDED_KCPSM2.VHD
--
------------------------------------------------------------------------------
--
-- Standard IEEE libraries
--
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
--
------------------------------------------------------------------------------
--
entity embedded_kcpsm2 is
  Port (  port_id : out std_logic_vector(7 downto 0);
        write_strobe : out std_logic;
        read_strobe : out std_logic;
          out_port : out std_logic_vector(7 downto 0);
          in_port : in std_logic_vector(7 downto 0);
        interrupt : in std_logic;
            reset : in std_logic;
              clk : in std_logic);
end embedded_kcpsm2;
--
------------------------------------------------------------------------------
--
architecture connectivity of embedded_kcpsm2 is
--
------------------------------------------------------------------------------
--
-- Declaration of KCPSM-II
--
  component kcpsm2
    Port (      address : out std_logic_vector(9 downto 0);
          instruction : in std_logic_vector(17 downto 0);
            port_id : out std_logic_vector(7 downto 0);
          write_strobe : out std_logic;
            out_port : out std_logic_vector(7 downto 0);
          read_strobe : out std_logic;
            in_port : in std_logic_vector(7 downto 0);
          interrupt : in std_logic;
              reset : in std_logic;
                clk : in std_logic);
    end component;
--
-- declaration of program ROM
--
  component prog_rom
    Port (      address : in std_logic_vector(9 downto 0);
          instruction : out std_logic_vector(17 downto 0);
                clk : in std_logic);
    end component;
--
------------------------------------------------------------------------------
--
-- Signals used to connect KCPSM-II to program ROM
--
signal address : std_logic_vector(9 downto 0);
signal instruction : std_logic_vector(17 downto 0);
--
------------------------------------------------------------------------------
--
-- Connection of macros
--
begin

  processor: kcpsm2
    port map(      address => address,
              instruction => instruction,
                  port_id => port_id,
             write_strobe => write_strobe,
                out_port => out_port,
              read_strobe => read_strobe,
                in_port => in_port,
              interrupt => interrupt,
                  reset => reset,
                    clk => clk);
  program: prog_rom
    port map(      address => address,
              instruction => instruction,
                    clk => clk);

end connectivity ;
------------------------------------------------------------------------------
--
-- END OF FILE EMBEDDED_KCPSM2.VHD
```

В случае применения объекта EMBEDDED_KCPSM2 в качестве элемента, входящего в состав разрабатываемой системы, следует воспользоваться выражениями декларации и создания экземпляра соответствующего компонента, которые приведены первой статье цикла [1] для макроса EMBEDDED_KCPSM. В этих выражениях следует только изменить название компонента embedded_kcpsm на embedded_kcpsm2. Все рассмотренные компоненты полностью совместимы с любой из конфигураций средств проектирования фирмы Xilinx серии ISE (Integrated Synthesis Environment), включая свободно распространяемую — WebPACK ISE [3].

Архитектура микропроцессорного ядра PicoBlaze для Virtex-II

Архитектура микропроцессорного ядра PicoBlaze, предназначенного для применения в кристаллах семейства Virtex-II, изображена на рис. 2. В структурном отношении она практически не отличается от архитектуры встраиваемого микропроцессорного модуля [1]. Различия проявляются на уровне отдельных структурных элементов. Эти отличия обусловлены, прежде всего, тем, что преимущества архитектуры ПЛИС семейства Virtex-II, по сравнению с кристаллами серий Spartan-II, Spartan-IIE, Virtex, Virtex-E, позволили расширить функциональные возможности структурных элементов соответствующего микропроцессорного ядра из семейства PicoBlaze.

Объем блока регистров общего назначения увеличен до тридцати двух восьмиразрядных регистров, которые обозначаются соответствующими порядковыми номерами s00 — s1F. Тем самым создаются предпосылки для повышения уровня оптимизации разрабатываемой микропроцессорной программы. Двукратное увеличение числа регистров общего назначения позволяет, прежде всего, сохранять больший объем промежуточных результатов вычислений, входных и выходных данных, не используя внешнего (по отношению к микропроцессорному ядру) ОЗУ. Замена обращений к внешней оперативной памяти обращениями к регистрам позволяет получить существенный выигрыш в быстродействии при выполнении программы за счет исключения операций перемещения данных между регистрами и ОЗУ и благодаря тому, что каждый из регистров общего назначения может выполнять функции аккумулятора.

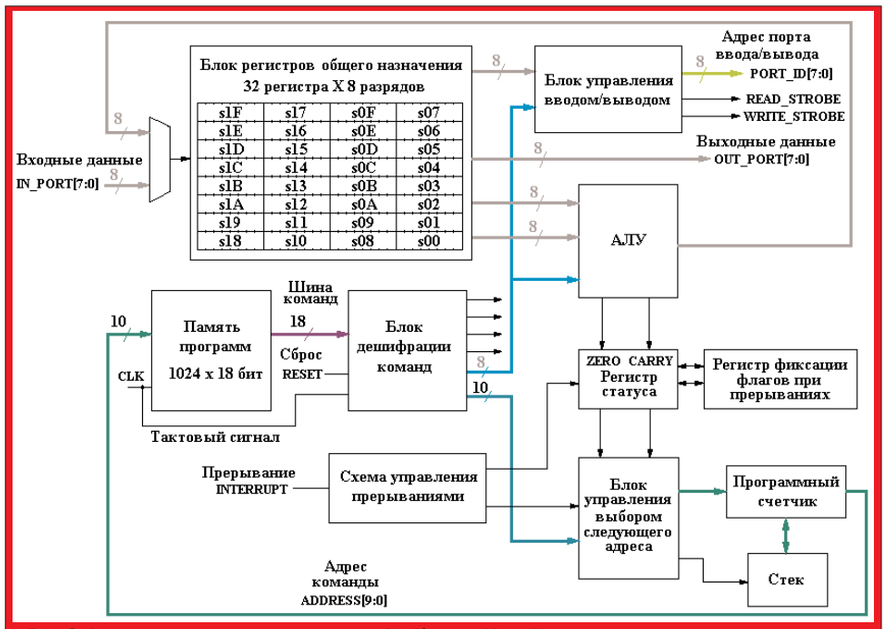

Рис. 2. Архитектура микропроцессорного ядра PicoBlaze для Virtex-II

Четырехкратное увеличение адресного пространства программной памяти потребовало расширения шины адресов до десяти разрядов. Тем самым обеспечивается возможность прямого обращения к любой ячейке ППЗУ, используемого для хранения микропроцессорных программ, в диапазоне адресов от 000 до 3FF. При этом длина команд возросла до восемнадцати разрядов, что повлекло за собой соответствующее расширение шины команд.

В соответствии с увеличением ширины шины адресов также изменена разрядность элементов архитектуры микропроцессорного ядра PicoBlaze, которые выполняют функции формирования и хранения адресов команд. Разрядность программного счетчика, регистров стека и блока управления выбором адреса следующей команды составляет десять бит.

Модернизация блока дешифрации команд, обусловленная необходимостью учета рассмотренных выше особенностей элементов архитектуры, привела, в частности, к изменению формата команд, поддерживаемых микропроцессорным ядром PicoBlaze, которое предназначено для применения в кристаллах семейства Virtex-II.

Изменения в процедуре обработки прерываний коснулись только значения адреса, по которому указывается вектор процедуры обслуживания прерывания. В связи с расширением диапазона адресов программной памяти в качестве адреса вектора прерывания, который в семействе микропроцессорных ядер PicoBlaze соответствует максимально допустимому значению адреса, используется значение 3FF.

Таблица 1. Форматы команд переходов микропроцессорного ядра PicoBlaze для Virtex-II

Поле кода операции								Поле адреса перехода									Мнемоника	Выполняемая операция
1	1	0	1	0	0	X	X	A	A	A	A	A	A	A	A	A	JUMP aaa	Безусловный переход
1	1	0	1	0	1	0	0	A	A	A	A	A	A	A	A	A	JUMP Z,aaa	Переход при условии, что флаг ZERO Flag находится в установленном состоянии
1	1	0	1	0	1	0	1	A	A	A	A	A	A	A	A	A	JUMP NZ,aaa	Переход при условии, что флаг ZERO Flag находится в сброшенном состоянии
1	1	0	1	0	1	1	0	A	A	A	A	A	A	A	A	A	JUMP C,aaa	Переход при условии, что флаг CARRY Flag находится в установленном состоянии
1	1	0	1	0	1	1	1	A	A	A	A	A	A	A	A	A	JUMP NC,aaa	Переход при условии, что флаг CARRY Flag находится в сброшенном состоянии
17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	Номер разряда микрокоманды

Таблица 2. Форматы команд вызова подпрограмм для микропроцессорного ядра PicoBlaze для Virtex-II

Поле кода операции								Поле адреса подпрограммы									Мнемоника	Выполняемая операция
1	1	0	1	1	0	X	X	A	A	A	A	A	A	A	A	A	CALL aaa	Безусловный вызов подпрограммы
1	1	0	1	1	1	0	0	A	A	A	A	A	A	A	A	A	CALL Z,aaa	Вызов подпрограммы при условии, что флаг ZERO Flag находится в установленном состоянии
1	1	0	1	1	1	0	1	A	A	A	A	A	A	A	A	A	CALL NZ,aaa	Вызов подпрограммы при условии, что флаг ZERO Flag находится в сброшенном состоянии
1	1	0	1	1	1	1	0	A	A	A	A	A	A	A	A	A	CALL C,aaa	Вызов подпрограммы при условии, что флаг CARRY Flag находится в установленном состоянии
1	1	0	1	1	1	1	1	A	A	A	A	A	A	A	A	A	CALL NC,aaa	Вызов подпрограммы при условии, что флаг CARRY Flag находится в сброшенном состоянии
17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	Номер разряда микрокоманды

Система команд микропроцессорного ядра PicoBlaze для Virtex-II

В состав системы команд микропроцессорного ядра PicoBlaze, встраиваемого в проекты, которые выполняются на основе ПЛИС семейства Virtex-II, входят те же 49 инструкций, что были подробно рассмотрены ранее [2]. При этом сохраняется классификация команд по функциональному признаку, в соответствии с которой они подразделяются на шесть уже известных групп. Изменения произошли в формате команд и их мнемонической записи. Необходимость преобразования формата инструкций обусловлена архитектурными особенностями, рассмотренными в предыдущем разделе. Как уже упоминалось, длина всех команд в двоичном представлении составляет восемнадцать разрядов. В большинстве инструкций поменялась длина полей и, соответственно, коды выполняемых операций. В некоторых командах изменилось взаимное расположение полей. В последующих разделах для каждой функциональной группы команд будут представлены соответствующие форматы и мнемоника инструкций рассматриваемого представителя семейства микропроцессорных ядер PicoBlaze.

Команды управления последовательностью выполнения операций в программе для ядра PicoBlaze для Virtex-II

В командах безусловного и условных переходов JUMP изменилась структура поля кода операции и длина поля адреса перехода. В соответствии с разрядностью шины адресов длина поля адреса перехода увеличена до десяти двоичных разрядов. При мнемонической форме записи команд передачи управления в программе параметр, определяющий адрес перехода, задается в виде последовательности, состоящей из трех шестнадцатеричных символов. Форматы команд безусловного и условных переходов JUMP в новой редакции представлены в таблице 1.

Модификация команд обращения к подпрограммам *CALL* также затронула структуру поля кода операции и длину поля адреса вызываемой процедуры. В новой редакции поле адреса вызываемой подпрограммы включает в себя десять двоичных разрядов. При мнемонической форме записи команд обращения к подпрограммам значение параметра, указывающее начальный адрес вызываемой процедуры, представляется в виде трехзначного шестнадцатеричного числа. Новые версии форматов команд безусловного и условных вызовов подпрограмм *CALL* приведены в таблице 2.

Преобразование инструкций возврата из подпрограммы *RETURN* коснулось тех же

полей, что и в командах вызова подпрограмм. Для всех вариантов команд завершения выполняемой подпрограммы и передачи управления основной программе или подпрограмме, из которой производилось обращение к этой процедуре, изменены коды выполняемой операции. Мнемоническая форма записи команд *RETURN* сохранилась без изменений. В таблице 3 представлены модифицированные форматы команд безусловного и условного возврата из подпрограммы *RETURN*.

Группа логических команд микропроцессорного ядра PicoBlaze для Virtex-II

В формате инструкций, которые относятся к группе логических команд, следует обратить внимание на два основных отличия по сравнению с аналогичными инструкциями, представленными в прошлом номере журнала [2]. Во-первых, длина поля кода операции увеличилась на один разряд и составляет пять бит. Во-вторых, поля команд, предназначенные для определения номеров регистров, содержимое которых используется в качестве операндов, также содержат пять двоичных разрядов. Изменение длины полей, в которых указываются номера регистров, связано с двукратным увеличением объема блока регистров общего назначения. В качестве номера регистра *NN*, который указывается в тексте инструкций, может использоваться любое число в диапазоне от 0 до 31, которое при мнемонической форме записи команд задается в виде последовательности из двух шестнадцатеричных символов (00-1F).

Новая редакция форматов команд поразрядных операций «Логическое И» (поразрядное умножение) *AND*, выполняемых над содержимым регистра общего назначения и константой *kk*, значение которой задается непосредственно в тексте инструкции, а также над содержимым двух регистров общего назначения, приведена в таблице 4.

Модифицированные форматы инструкций *OR*, предназначенных для выполнения операций поразрядного сложения двух операндов (поразрядное «Логическое ИЛИ»), определены в таблице 5 для двух вариантов. В первом случае операндами является содержимое любого из тридцати двух регистров общего назначения и константа *kk*, значение которой указывается в соответствующем поле команды, а во втором — содержимое двух регистров с номерами *NN* и *MM*.

Новая редакция форматов команд XOR, используемых для выполнения поразрядной операции «Исключающее ИЛИ» с участием содержимого регистра общего назначения с номером *NN* и константы *kk* или содержимого двух регистров с номерами *NN* и *MM*, представлена в таблице 6.

Форматы инструкций *LOAD*, предназначенных для загрузки константы или содержимого какого-либо регистра в выбранный регистр общего назначения, в новой редакции приведены в таблице 7.

Таблица 3. Форматы команд безусловного и условного возврата из подпрограммы для микропроцессорного ядра PicoBlaze для Virtex-II

Поле кода операции								Поле адреса										Мнемоника	Выполняемая операция
1	0	0	1	0	0	X	X	0	0	0	0	0	0	0	0	0	0	RETURN	Безусловный возврат из подпрограммы
1	0	0	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	RETURN Z	Возврат из подпрограммы при условии, что флаг ZERO Flag находится в установленном состоянии
1	0	0	1	0	1	0	1	0	0	0	0	0	0	0	0	0	0	RETURN NZ	Возврат из подпрограммы при условии, что флаг ZERO Flag находится в сброшенном состоянии
1	0	0	1	0	1	1	0	0	0	0	0	0	0	0	0	0	0	RETURN C	Возврат из подпрограммы при условии, что флаг CARRY Flag находится в установленном состоянии
1	0	0	1	0	1	1	1	0	0	0	0	0	0	0	0	0	0	RETURN NC	Возврат из подпрограммы при условии, что флаг CARRY Flag находится в сброшенном состоянии
17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	Номер разряда микрокоманды	

Таблица 4. Форматы команд поразрядных операций «Логическое И» микропроцессорного ядра PicoBlaze для Virtex-II

Поле кода операции					Поле номера регистра					Поле константы								Мнемоника	Выполняемая операция
0	0	0	0	1	n	n	n	n	n	K	K	K	K	K	K	K	K	AND sNN, kk	Поразрядное «Логическое И» содержимого регистра sNN и константы kk
Поле кода операции					**Поле номера первого регистра**					**Поле номера второго регистра**					**Нулевые разряды**			**Мнемоника**	**Выполняемая операция**
0	1	0	0	1	n	n	n	n	n	m	m	m	m	m	0	0	0	AND sNN,sMM	Поразрядное «Логическое И» содержимого регистров sNN и sMM
17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	Номер разряда микрокоманды	

Таблица 5. Форматы команд поразрядных операций «Логическое ИЛИ» микропроцессорного ядра PicoBlaze для Virtex-II

Поле кода операции					Поле номера регистра					Поле константы								Мнемоника	Выполняемая операция
0	0	0	1	0	n	n	n	n	n	K	K	K	K	K	K	K	K	OR sNN, kk	Поразрядное «Логическое ИЛИ» содержимого регистра sNN и константы kk
Поле кода операции					**Поле номера первого регистра**					**Поле номера второго регистра**					**Нулевые разряды**			**Мнемоника**	**Выполняемая операция**
0	1	0	1	0	n	n	n	n	n	m	m	m	m	m	0	0	0	OR sNN,sMM	Поразрядное «Логическое ИЛИ» содержимого регистров sNN и sMM
17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	Номер разряда микрокоманды	

Таблица 6. Форматы команд поразрядных операций «Исключающее ИЛИ» микропроцессорного ядра PicoBlaze для Virtex-II

Поле кода операции					Поле номера регистра					Поле константы								Мнемоника	Выполняемая операция
0	0	0	1	1	n	n	n	n	n	K	K	K	K	K	K	K	K	XOR sNN, kk	Поразрядное «Исключающее ИЛИ» содержимого регистра sNN и константы kk
Поле кода операции					**Поле номера первого регистра**					**Поле номера второго регистра**					**Нулевые разряды**			**Мнемоника**	**Выполняемая операция**
0	1	0	1	1	n	n	n	n	n	m	m	m	m	m	0	0	0	XOR sNN,sMM	Поразрядное «Исключающее ИЛИ» содержимого регистров sNN и sMM
17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	Номер разряда микрокоманды	

Таблица 7. Форматы инструкции загрузки данных в регистр общего назначения микропроцессорного ядра PicoBlaze для Virtex-II

Поле кода операции					Поле номера регистра					Поле константы								Мнемоника	Выполняемая операция
0	0	0	0	0	n	n	n	n	n	K	K	K	K	K	K	K	K	LOAD sNN, kk	Загрузка константы kk в регистр sNN
Поле кода операции					**Поле номера первого регистра**					**Поле номера второго регистра**					**Нулевые разряды**			**Мнемоника**	**Выполняемая операция**
0	1	0	0	0	n	n	n	n	n	m	m	m	m	m	0	0	0	LOAD sNN,sMM	Загрузка содержимого регистра sMM в регистр sNN
17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	Номер разряда микрокоманды	

Группа арифметических команд микропроцессорного ядра PicoBlaze для Virtex-II

В структуре полей и мнемонической форме записи арифметических команд микропроцессорного ядра PicoBlaze, предназначенного для применения в кристаллах семейства Virtex-II, произошли те же изменения (по сравнению с форматами аналогичных команд, приведенными ранее [2]), что и в логических инструкциях, рассмотренных в предыдущем разделе.

Модифицированные варианты форматов команд сложения *ADD* содержимого регистра с номером *NN* и константы *kk* или содержимого двух регистров общего назначения с номерами *NN* и *MM* без учета переноса представлены в таблице 8.

Новая версия форматов инструкций *ADDCY*, предназначенных для вычисления суммы двух операндов с учетом значения

Таблица 8. Форматы команд сложения двух операндов без учета переноса для микропроцессорного ядра PicoBlaze для Virtex-II

Поле кода операции					Поле номера регистра					Поле константы								Мнемоника	Выполняемая операция
0	0	1	0	0	n	n	n	n	n	K	K	K	K	K	K	K	K	ADD sNN, kk	Сложение содержимого регистра sNN и константы kk
Поле кода операции					**Поле номера первого регистра**					**Поле номера второго регистра**					**Нулевые разряды**			**Мнемоника**	**Выполняемая операция**
0	1	1	0	0	n	n	n	n	n	m	m	m	m	m	0	0	0	ADD sNN,sMM	Сложение содержимого регистров sNN и sMM
17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	Номер разряда микрокоманды	

Таблица 9. Форматы команд сложения двух операндов с учетом переноса для микропроцессорного ядра PicoBlaze для Virtex-II

Поле кода операции					Поле номера регистра					Поле константы								Мнемоника	Выполняемая операция
0	0	1	0	1	n	n	n	n	n	K	K	K	K	K	K	K	K	ADDCY sNN, kk	Сложение содержимого регистра sNN и константы kk с учетом переноса
Поле кода операции					**Поле номера первого регистра**					**Поле номера второго регистра**					**Нулевые разряды**			**Мнемоника**	**Выполняемая операция**
0	1	1	0	1	n	n	n	n	n	m	m	m	m	m	0	0	0	ADDCY sNN,sMM	Сложение содержимого регистров sNN и sMM с учетом переноса
17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	Номер разряда микрокоманды	

Таблица 10. Форматы команд вычитания без учета заема для микропроцессорного ядра PicoBlaze для Virtex-II

Поле кода операции					Поле номера регистра					Поле константы								Мнемоника	Выполняемая операция
0	0	1	1	0	n	n	n	n	n	K	K	K	K	K	K	K	K	SUB sNN, kk	Вычитание из содержимого регистра sNN константы kk
Поле кода операции					**Поле номера первого регистра**					**Поле номера второго регистра**					**Нулевые разряды**			**Мнемоника**	**Выполняемая операция**
0	1	1	1	0	n	n	n	n	n	m	m	m	m	m	0	0	0	SUB sNN,sMM	Вычитание содержимого регистра sMM из содержимого регистра sNN
17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	Номер разряда микрокоманды	

Таблица 11. Форматы инструкций вычитания с учетом заема для микропроцессорного ядра PicoBlaze для Virtex-II

Поле кода операции					Поле номера регистра					Поле константы								Мнемоника	Выполняемая операция
0	0	1	1	1	n	n	n	n	n	K	K	K	K	K	K	K	K	SUBCY sNN, kk	Вычитание из содержимого регистра sNN константы kk с учетом заема
Поле кода операции					**Поле номера первого регистра**					**Поле номера второго регистра**					**Нулевые разряды**			**Мнемоника**	**Выполняемая операция**
0	1	1	1	1	n	n	n	n	n	m	m	m	m	m	0	0	0	SUBCY sNN,sMM	Вычитание содержимого регистра sMM из содержимого регистра sNN с учетом заема
17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	Номер разряда микрокоманды	

Таблица 12. Форматы команд логического или циклического сдвига данных микропроцессорного ядра PicoBlaze для Virtex-II

Поле кода операции					Поле номера регистра					Поле направления сдвига					Поле типа сдвига			Мнемоника	Выполняемая операция
1	0	1	0	0	n	n	n	n	n	0	0	0	0	1	1	1	0	SR0 sNN	Логический сдвиг содержимого регистра sNN вправо на один разряд с записью 0
1	0	1	0	0	n	n	n	n	n	0	0	0	0	1	1	1	1	SR1 sNN	Логический сдвиг содержимого регистра sNN вправо на один разряд с записью 1
1	0	1	0	0	n	n	n	n	n	0	0	0	0	1	0	1	0	SRX sNN	Логический сдвиг содержимого регистра sNN вправо с сохранением последнего разряда
1	0	1	0	0	n	n	n	n	n	0	0	0	0	1	0	0	0	SRA sNN	Циклический сдвиг содержимого регистра sNN вправо через разряд переноса/заема
1	0	1	0	0	n	n	n	n	n	0	0	0	0	1	1	0	0	RR sNN	Циклический сдвиг содержимого регистра sNN вправо без участия бита переноса
1	0	1	0	0	n	n	n	n	n	0	0	0	0	0	1	1	0	SL0 sNN	Логический сдвиг содержимого регистра sNN влево на один разряд с записью 0
1	0	1	0	0	n	n	n	n	n	0	0	0	0	0	1	1	1	SL1 sNN	Логический сдвиг содержимого регистра sNN влево на один разряд с записью 1
1	0	1	0	0	n	n	n	n	n	0	0	0	0	0	0	1	0	SLX sNN	Логический сдвиг содержимого регистра sNN влево с сохранением последнего разряда
1	0	1	0	0	n	n	n	n	n	0	0	0	0	0	0	0	0	SLA sNN	Циклический сдвиг содержимого регистра sNN влево через разряд переноса/заема
1	0	1	0	0	n	n	n	n	n	0	0	0	0	0	1	0	0	RL sNN	Циклический сдвиг содержимого регистра sNN влево без участия бита переноса
17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	Номер разряда микрокоманды	

флага переноса, полученного при выполнении предыдущей операции, приведена в таблице 9.

Форматы инструкций *SUB*, используемых для выполнения операции вычитания с участием тех же операндов, что и в командах сложения, без учета заема, в новой редакции представлены в таблице 10.

Модифицированные варианты форматов команд *SUBCY*, предназначенных для вычисления разности двух операндов с учетом значения заема, образовавшегося при выполнении предыдущей операции, приведены в таблице 11.

Команды сдвига данных для микропроцессорного ядра PicoBlaze для Virtex-II

В формате команд, предназначенных для выполнения операций сдвига данных, произошли те же изменения, что и в формате логических инструкций — длина полей кода операции и номера регистра увеличилась на один бит и составляет пять двоичных разрядов. Отличия в мнемонической форме записи команд сдвига от однотипных выражений, приведенных в предыдущей статье [2], заключаются только в форме представления параметра, определяющего номер регистра общего назначения, над содержимым которого выполняется соответствующая операция. Номер регистра, участвующего в операции сдвига, задается в виде последовательности, состоящей из двух шестнадцатеричных символов. Преобразованные форматы инструкций логического (арифметического) и циклического сдвига данных, находящихся в регистре общего назначения с указанным номером, представлены в таблице 12.

Команды ввода-вывода микропроцессорного ядра PicoBlaze для Virtex-II

Структура инструкций ввода-вывода, используемых для организации чтения данных из входного порта в заданный регистр общего назначения и передачи информации из указанного регистра в выходной порт, отличается от структуры аналогичных команд, приведенной ранее [2], длиной полей кода операции и номеров регистров, каждая из которых составляет по пять двоичных разрядов. Значения номеров регистров общего назначения, используемых в операциях ввода-вывода, при мнемонической форме записи указываются в виде двухразрядного шестнадцатеричного числа. Новые варианты форматов команд ввода-вывода с различными видами адресации приведены в таблице 13.

Команды обслуживания прерываний микропроцессорного ядра PicoBlaze для Virtex-II

В формате инструкций обслуживания прерываний изменилось взаимное расположение поля кода операции и поля режима обра-

Таблица 13. Форматы команд ввода-вывода микропроцессорного ядра PicoBlaze для Virtex-II

Поле кода операции					Поле номера регистра					Поле адреса порта ввода/вывода								Мнемоника	Выполняемая операция
1	0	0	0	0	n	n	n	n	n	K	K	K	K	K	K	K	K	INPUT sNN, kk	Чтение данных из порта ввода/вывода с адресом kk в регистр sNN
1	0	0	0	1	n	n	n	n	n	K	K	K	K	K	K	K	K	OUTPUT sNN, kk	Запись данных из регистра sNN в порт ввода/вывода с адресом kk
Поле кода операции					**Поле номера первого регистра**					**Поле номера второго регистра**					**Нулевые разряды**			**Мнемоника**	**Выполняемая операция**
1	1	0	0	0	n	n	n	n	n	m	m	m	m	m	0	0	0	INPUT sNN,(sMM)	Чтение данных из порта ввода/вывода с адресом, определяемым регистром sMM, в регистр sNN
1	1	0	0	1	n	n	n	n	n	m	m	m	m	m	0	0	0	OUTPUT sNN,(sMM)	Запись данных из регистра sNN в порт с адресом, определяемым регистром sMM
17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	Номер разряда микрокоманды	

Таблица 14. Форматы команд обслуживания прерываний микропроцессорного ядра PicoBlaze для Virtex-II

Поле кода операции									Поле режима обработки прерываний								Мнемоника	Выполняемая операция
1	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	1	RETURNI ENABLE	Возврат из процедуры обработки и установка режима запрета прерывания
1	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	RETURNI DISABLE	Возврат из процедуры обработки и установка режима разрешения прерывания
1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	1	ENABLE INTERRUPT	Установка режима разрешения прерывания
1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	DISABLE INTERRUPT	Установка режима запрета прерывания
17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	Номер разряда микрокоманды	

ботки прерываний. Кроме того, длина каждого из этих полей команды в двоичном представлении стала составлять девять разрядов. При этом мнемоника инструкций, используемых для обработки прерываний, осталась прежней.

Новая редакция форматов команд возврата из процедуры обслуживания прерываний *RETURNI* и установки режима обслуживания прерываний в программе *ENABLE INTERRUPT* и *DISABLE INTERRUPT* представлена в таблице 14.

Завершая рассмотрение особенностей встраиваемого микропроцессорного модуля PicoBlaze, реализуемого в кристаллах серии Virtex-II, следует упомянуть о новой версии ассемблера, включенной в состав комплекта, предоставляемого пользователю. Новый вариант программы, формирующей файлы описания содержимого программной памяти, учитывает изменения в формате команд и параметры блочного ОЗУ используемых ПЛИС. В следующей статье цикла будут представлены отличительные особенности наиболее компактной версии микропроцессорного ядра PicoBlaze, предназначенной для использования в проектах, выполняемых на базе кристаллов семейства CoolRunner-II.

Литература

1. Зотов В. PicoBlaze — семейство восьмиразрядных микропроцессорных ядер, реализуемых на основе ПЛИС фирмы Xilinx // Компоненты и технологии. 2003. № 4.
2. Зотов В. Система команд микропроцессорного ядра PicoBlaze, реализуемого на основе ПЛИС семейств Spartan-II, Spartan-IIE, Virtex, Virtex-E // Компоненты и технологии. 2003. № 5.
3. Зотов В. Проектирование цифровых устройств на основе ПЛИС фирмы Xilinx в САПР WebPack ISE. М.: Горячая линия — Телеком. 2003.

Особенности микропроцессорного ядра PicoBlaze, предназначенного для применения в проектах, реализуемых на основе ПЛИС семейства CoolRunner-II

В предыдущих публикациях данного цикла [1–3] были рассмотрены элементы семейства 8-разрядных микропроцессорных ядер PicoBlaze, предназначенные для использования в проектах, которые выполняются на базе ПЛИС серий FPGA фирмы Xilinx. Несмотря на то, что кристаллы серий CPLD обладают значительно меньшими функциональными возможностями по сравнению с представителями семейств FPGA, некоторые из них могут применяться для реализации встраиваемых микропроцессорных систем. Для этих целей фирма Xilinx предлагает наиболее компактную версию микропроцессорного ядра PicoBlaze, которая предназначена для применения в проектах, реализуемых на основе ПЛИС семейства CoolRunner-II.

Валерий Зотов

walerry@euro.ru

Встраиваемый микропроцессорный модуль для ПЛИС семейства CoolRunner-II разработан на основе микропроцессорного ядра PicoBlaze, предназначенного для реализации на базе кристаллов семейств Spartan-II, Spartan-IIE, Virtex, Virtex-E. Поэтому в настоящей статье при рассмотрении характеристик, архитектуры и системы команд микропроцессорного ядра PicoBlaze для ПЛИС семейства CoolRunner-II представлены только его отличительные особенности по сравнению с базовым ядром, описание которого было опубликовано ранее [1–2].

Данное семейство кристаллов отличается от других серий ПЛИС CPLD, выпускаемых фирмой Xilinx, высоким быстродействием, низкой потребляемой мощностью и наличием микросхем с достаточно большим объемом логических ресурсов. Более подробно характеристики кристаллов семейства CoolRunner-II представлены в отдельной статье [4].

Основные характеристики ядра PicoBlaze, реализуемого на основе ПЛИС семейства CoolRunner-II

Микропроцессорное ядро PicoBlaze, предназначенное для разработки систем на основе кристаллов семейства CoolRunner-II, обладает основными техническими характеристиками, схожими с характеристиками базового модуля, представленными в первой публикации цикла [1]. Наиболее существенные различия проявляются в вопросе применения встроенного ППЗУ микропрограмм, объеме блока регистров общего назначения и производительности. Рассматриваемый представитель семейства PicoBlaze отличается от микропроцессорного ядра для ПЛИС семейств Spartan-II, Spartan-IIE, Virtex, Virtex-E следующими особенностями:

- возможность применения в проектах, выполняемых на основе кристаллов CoolRunner-II с числом макроячеек 256 и более (XC2C256, XC2C384, XC2C512);
- объем блока регистров общего назначения — восемь регистров по восемь разрядов;
- отсутствие встроенного ППЗУ микропрограмм;
- четырехуровневый стек;
- объем ресурсов кристалла, необходимых для реализации микропроцессорного ядра PicoBlaze в ПЛИС CoolRunner-II, составляет 212 макроячеек, что соответствует 83% от полного объема логических ресурсов кристалла XC2C256 (при этом используется 155 регистров и 53 пользовательских выводов ПЛИС, что составляет 61 и 45% от полного объема этих ресурсов).

Структура проекта ядра PicoBlaze, реализуемого на основе CoolRunner-II

В кристаллах семейства CoolRunner-II, как и в ПЛИС CPLD других серий, отсутствуют ресурсы выделенной блочной памяти. Использование же основных логических ресурсов ПЛИС (триггеров, входящих в состав макроячеек) для формирования программной памяти встраиваемого процессорного модуля крайне неэффективно. Поэтому в версии микропроцессорного ядра PicoBlaze, предназначенной для реализации в кристаллах семейства CoolRunner-II, исключен модуль встроенного ППЗУ микропрограмм. Таким образом, структура рассматриваемого варианта встраиваемого микропроцессорного ядра PicoBlaze, в отличие от рассмотренных ранее представителей этого семейства, содержит только исполнительный модуль.

Комплект микропроцессорного ядра PicoBlaze, предназначенного для семейства CoolRunner-II, содержит три группы файлов, упакованных в архив. Каждая из этих групп расположена в отдельном каталоге (разделе) архива. Первая группа содержит файлы всех необходимых модулей описаний на языке VHDL. Файлы этой группы расположены в разделе VHDL. Вторая группа включает в себя исполняемый программный модуль ассемблера asm.exe и файл его исходного описания на языке C.

В эту же группу включен файл, содержащий тестовую программу на языке ассемблера, и файлы, полученные в результате обработки этой программы ассемблером. Данная группа файлов находится в разделе C. Третья группа файлов, расположенная в разделе DEMO_TEST, представляет собой тестовый проект, иллюстрирующий использование компонентов ядра. Демонстрационный проект и все компоненты ядра полностью совместимы с любой из конфигураций средств проектирования фирмы Xilinx серии ISE (Integrated Synthesis Environment), включая свободно распространяемую — WebPACK ISE [3].

Исполнительный модуль выполнен в форме компонента *picoblaze*, представляющего собой макрос с относительным размещением, описание которого на языке VHDL содержится в файле *picoblaze.vhd*. В этом описании используется ряд компонентов следующего (более низкого) уровня иерархии, которые в большинстве своем соответствуют элементам архитектуры микропроцессорного ядра PicoBlaze. Описания этих компонентов находятся в соответствующих файлах, которые расположены в том же разделе, что и файл описания компонента верхнего уровня иерархии *picoblaze*. Названия файлов, содержащих описания компонентов нижнего уровня иерархии, как правило, совпадают с именами этих компонентов. Выражения декларации компонента *picoblaze* в составе VHDL-описания проектируемой системы выглядят следующим образом.

```
component picoblaze
   Port (
         address : out std_logic_vector(7 downto 0);
         instruction : in std_logic_vector(15 downto 0);
         port_id : out std_logic_vector(7 downto 0);
         write_strobe : out std_logic;
         out_port : out std_logic_vector(7 downto 0);
         read_strobe : out std_logic;
         in_port : in std_logic_vector(7 downto 0);
         interrupt : in std_logic;
         reset : in std_logic;
         clk : in std_logic
        );
 end component;
```

В приведенных выражениях декларации используется та же система обозначений интерфейсных цепей компонентов микропроцессорного ядра PicoBlaze, что и для компонентов базового варианта, подробно описанная ранее [1]. Включение экземпляра компонента, представляющего исполнительный модуль *picoblaze*, в состав структурного описания архитектуры проектируемой системы осуществляется с помощью следующей конструкции.

```
inst_name_processor: picoblaze
   port map(
              address => address_name,
              instruction => instruction_name,
              port_id => port_id_name,
              write_strobe => write_strobe_name,
              out_port => out_port_name,
              read_strobe => read_strobe_name,
              in_port => in_port_name,
              interrupt => interrupt_event,
              reset => reset_name,
              clk => clk_name
            );
```

В приведенном шаблоне следует вместо идентификатора *inst_name_processor* задать метку, которая, как правило, соответствует позиционному обозначению создаваемого экземпляра компонента. Кроме того, в операторе port map нужно указать названия сигналов, которые используются в описании проектируемого устройства.

Для отладки разрабатываемой программы методом моделирования ядра в составе создаваемого проекта с помощью системы ModelSim XE [5–8] можно использовать компонент виртуального ПЗУ микропрограмм. Этот компонент представляет ПЗУ информационной емкостью 4 Кбит с организацией 256×16 разрядов. В качестве шаблона для декларации и создания экземпляра компонента виртуального ПЗУ микропрограмм можно использовать соответствующие выражения, приведенные для модуля программной памяти ядра PicoBlaze, реализуемого на базе ПЛИС семейств Spartan-II, Spartan-IIE, Virtex, Virtex-E [1]. Название компонента виртуального ПЗУ микропрограмм должно совпадать с именем файла, в котором содержится текст отлаживаемой программы на языке ассемблера.

Тестовый проект, включенный в состав комплекта микропроцессорного ядра PicoBlaze, можно использовать в качестве образца VHDL-описания отладочной системы. Основу этой системы образует VHDL-описание объекта DEMO, в структуру которого входит исполнительный модуль и подключенный к нему модуль виртуальной программной памяти. Ниже приведен полный текст описания тестовой системы, который также демонстрирует методику применения компонента *picoblaze* в составе проектируемого устройства.

```
----------------------------------------------------------------------
-- Standard IEEE libraries
--
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
----------------------------------------------------------------------
entity demo is
   Port (output : out std_logic_vector(7 downto 0);
         reset : in std_logic;
         clk : in std_logic);
   end demo;
----------------------------------------------------------------------
-- Start of test achitecture
--
architecture Behavioral of demo is
----------------------------------------------------------------------
 component picoblaze
   Port (
         address : out std_logic_vector(7 downto 0);
         instruction : in std_logic_vector(15 downto 0);
         port_id : out std_logic_vector(7 downto 0);
         write_strobe : out std_logic;
         out_port : out std_logic_vector(7 downto 0);
         read_strobe : out std_logic;
         in_port : in std_logic_vector(7 downto 0);
         interrupt : in std_logic;
         reset : in std_logic;
         clk : in std_logic
         );
   end component;
--
-- declaration of program ROM
--
 component demo_test
   Port (
         address : in std_logic_vector(7 downto 0);
         dout : out std_logic_vector(15 downto 0);
         clk : in std_logic
         );
   end component;
----------------------------------------------------------------------
-- Signals used to connect picoblaze to program ROM and I/O logic
--
signal address : std_logic_vector(7 downto 0);
signal instruction : std_logic_vector(15 downto 0);
signal port_id : std_logic_vector(7 downto 0);
signal out_port : std_logic_vector(7 downto 0);
signal in_port : std_logic_vector(7 downto 0);
signal write_strobe : std_logic;
signal read_strobe : std_logic;
signal interrupt_event : std_logic;
--signal reset : std_logic;
--
----------------------------------------------------------------------
-- Start of circuit description
--
begin
-- Inserting picoblaze and the program memory
--
   processor: picoblaze
         port map(
                  address => address,
                  instruction => instruction,
                  port_id => port_id,
                  write_strobe => write_strobe,
                  out_port => out_port,
                  read_strobe => read_strobe,
                  in_port => in_port,
                  interrupt => interrupt_event,
                  reset => reset,
                  clk => clk
                  );

   program: demo_test
         port map(
                  address => address,
                  dout => instruction,
                  clk => clk);
   --
   -- Unused inputs on processor
   --
   in_port <= «00000000»;
   interrupt_event <= '0';
   -- reset <= '0';
   --
   -- adding the output registers to the processor

   IO_registers: process(clk)
   begin

   -- waveform register at address 01

   if clk'event and clk='1' then
        if port_id(0)='1' and write_strobe='1' then
             output <= out_port;
        end if;
   end if;
 end process IO_registers;
end Behavioral;
----------------------------------------------------------------------
-- END OF FILE demo.VHD
--
```

В структуре представленного описания можно выделить три раздела. В первом разделе описания указаны ссылки на используемые стандартные библиотеки и пакеты. Следующий раздел содержит операторы, описывающие интерфейс объекта DEMO. В третьем разделе приводится структурное описание архитектуры этого объекта. Этот раздел состоит из пяти секций. Первая секция содержит выражения декларации используемых компонентов. Во второй секции представлены выражения декларации внутренних сигналов устройства. В третьей секции описывается собственно структура объекта DEMO. Четвертая секция содержит выражения, которые определяют значения сигналов на неиспользуемых входах микропроцессорного ядра. Пятая секция описывает процесс формирования выходных сигналов объекта DEMO.

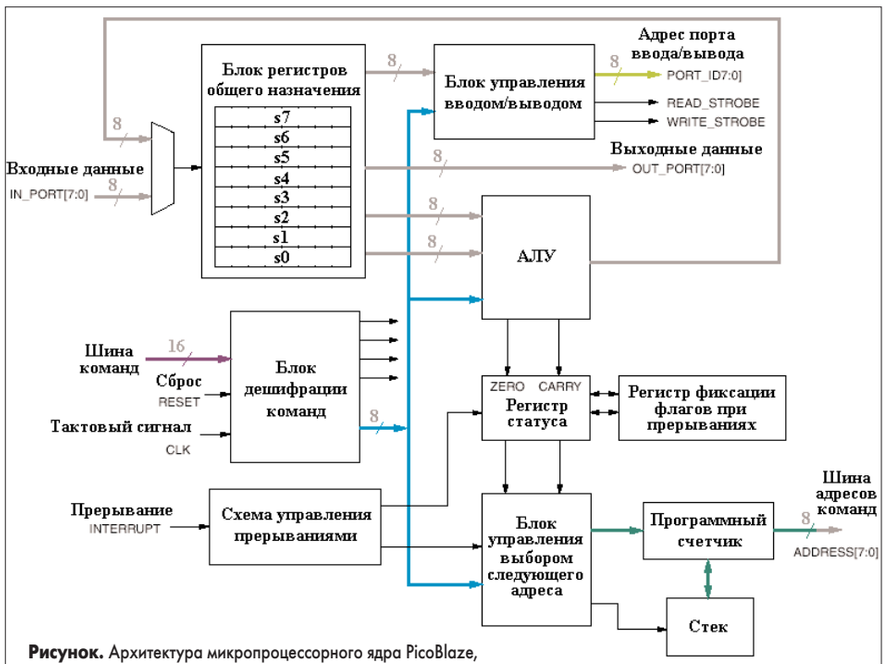

Рисунок. Архитектура микропроцессорного ядра PicoBlaze, реализуемого на основе ПЛИС семейства CoolRunner-II

Архитектура ядра PicoBlaze, реализуемого на основе CoolRunner-II

Архитектура микропроцессорного ядра PicoBlaze, предназначенного для применения в кристаллах семейства CoolRunner-II, изображена на рисунке. В структурном отношении она отличается от архитектуры базового микропроцессорного модуля [1] только отсутствием встроенного блока программной памяти. Кроме того, имеются различия, которые проявляются уже на уровне отдельных структурных элементов. Эти отличия обусловлены, прежде всего, ограниченным объемом ресурсов ПЛИС семейства CoolRunner-II по сравнению с кристаллами серий Spartan-II, Spartan-IIE, Virtex, Virtex-E. Поэтому одной из основных задач при разработке рассматриваемого варианта микропроцессорного ядра PicoBlaze являлась минимизация ресурсов кристалла, необходимых для его реализации. Решение этой задачи достигнуто за счет сокращения функциональных возможностей отдельных структурных элементов.

Вдвое уменьшен объем блока регистров общего назначения, который в новой версии содержит восемь восьмиразрядных регистров, обозначаемых соответствующими порядковыми номерами s0–s7.

Почти в четыре раза (с пятнадцати уровней до четырех) сокращена глубина стека программного счетчика. Тем самым накладывается более жесткое ограничение на количество вложенных вызовов процедур в разрабатываемой программе.

Модернизация блока дешифрации команд, обусловленная необходимостью снижения объема используемых ресурсов ПЛИС, привела, в частности, к изменению формата команд, поддерживаемых микропроцессорным ядром PicoBlaze, которое предназначено для применения в кристаллах семейства CoolRunner-II.

Система команд ядра PicoBlaze, реализуемого на основе CoolRunner-II

Базовая система команд микропроцессорного ядра PicoBlaze, встраиваемого в проекты, выполняемые на основе ПЛИС семейства CoolRunner-II, включает в себя 49 инструкций [2]. При классификации команд по функциональному признаку они подразделяются на шесть уже известных групп. Изменения произошли только в формате команд. В ряде инструкций поменялась длина полей и коды выполняемых операций. При этом полная длина команд не изменилась и по-прежнему составляет шестнадцать двоичных разрядов. В некоторых командах изменилось взаимное расположение полей. Мнемоническая форма записи инструкций сохранилась без изменений.

При необходимости разработчик может расширить существующую систему команд, дополнив ее собственными инструкциями. Для этого нужно внести соответствующие изменения в файлы исходного описания микропроцессорного ядра *picoblaze.vhd* и ассемблера *asm.cpp*.

В последующих разделах будут представлены соответствующие форматы инструкций для каждой функциональной группы, которые входят в базовую систему команд рассматриваемого представителя семейства микропроцессорных ядер PicoBlaze.

Таблица 1. Форматы команд переходов ядра PicoBlaze, реализуемого на основе CoolRunner-II

Поле кода операции								Поле адреса перехода								Мнемоника	Выполняемая операция
1	1	0	1	0	0	X	X	A	A	A	A	A	A	A	A	JUMP aa	Безусловный переход
1	1	0	1	0	1	0	0	A	A	A	A	A	A	A	A	JUMP Z,aa	Переход при условии, что флаг ZERO Flag находится в установленном состоянии
1	1	0	1	0	1	0	1	A	A	A	A	A	A	A	A	JUMP NZ,aa	Переход при условии, что флаг ZERO Flag находится в сброшенном состоянии
1	1	0	1	0	1	1	0	A	A	A	A	A	A	A	A	JUMP C,aa	Переход при условии, что флаг CARRY Flag находится в установленном состоянии
1	1	0	1	0	1	1	1	A	A	A	A	A	A	A	A	JUMP NC,aa	Переход при условии, что флаг CARRY Flag находится в сброшенном состоянии
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	Номер разряда микрокоманды	

Таблица 2. Форматы команд вызова подпрограмм для ядра PicoBlaze, реализуемого на основе CoolRunner-II

Поле кода операции								Поле адреса подпрограммы								Мнемоника	Выполняемая операция
1	1	0	1	1	0	X	X	A	A	A	A	A	A	A	A	CALL aa	Безусловный вызов подпрограммы
1	1	0	1	1	1	0	0	A	A	A	A	A	A	A	A	CALL Z,aa	Вызов подпрограммы при условии, что флаг ZERO Flag находится в установленном состоянии
1	1	0	1	1	1	0	1	A	A	A	A	A	A	A	A	CALL NZ,aa	Вызов подпрограммы при условии, что флаг ZERO Flag находится в сброшенном состоянии
1	1	0	1	1	1	1	0	A	A	A	A	A	A	A	A	CALL C,aa	Вызов подпрограммы при условии, что флаг CARRY Flag находится в установленном состоянии
1	1	0	1	1	1	1	1	A	A	A	A	A	A	A	A	CALL NC,aa	Вызов подпрограммы при условии, что флаг CARRY Flag находится в сброшенном состоянии
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	Номер разряда микрокоманды	

Команды управления последовательностью выполнения операций в программе для ядра PicoBlaze, реализуемого на базе CoolRunner-II

В командах безусловного и условных переходов *JUMP* изменились значения и структура поля кода операции. Форматы команд безусловного и условных переходов *JUMP* в новой редакции представлены в таблице 1.

Модификация команд обращения к подпрограммам *CALL* также затронула структуру поля кода операции. Поле адреса вызываемой подпрограммы осталось без изменений. Новые версии форматов команд безусловного и условных вызовов подпрограмм *CALL* приведены в таблице 2.

Преобразование инструкций возврата из подпрограммы *RETURN* проявилось в изменении структуры полей и значений кода выполняемой операции. В таблице 3. представлены модифицированные форматы команд безусловного и условного возврата из подпрограммы *RETURN*.

Группа логических команд ядра PicoBlaze, предназначенного для кристаллов CoolRunner-II

В формате инструкций, относящихся к группе логических команд, произошли следующие изменения по сравнению с аналогичными инструкциями, представленными ранее [2]. Во-первых, все команды этой группы имеют одинаковую длину поля кода операции. В новой редакции это поле включает в себя пять двоичных разрядов. Во-вторых, длина полей команд, в которых указываются номера регистров, используемых при выполнении операции, уменьшилась на один бит и составляет три двоичных разряда. Изменение длины полей, предназначенных для определения номеров регистров, обусловлено двукратным сокращением объема блока регистров общего назначения. В качестве номеров регистров *N* и *M*, которые указываются при мнемонической форме записи инструкций, могут использоваться любые числа в диапазоне от 0 до 7.

Новая редакция форматов команд поразрядных операций «Логическое И» (поразрядное умножение) AND, выполняемых над содержимым одного из регистров общего назначения и константой *kk*, значение которой задается непосредственно в тексте инструкции, а также над содержимым двух регистров общего назначения, приведена в таблице 4.

Модифицированные форматы инструкций OR, предназначенных для выполнения операций поразрядного сложения двух операндов (поразрядное «Логическое ИЛИ»), определены в таблице 5 для двух вариантов. В первом случае операндами является содержимое любого из восьми регистров общего назначения и константа *kk*, значение которой указывается в соответствующем поле команды, а во втором — содержимое двух регистров с номерами *N* и *M*.

Новая редакция форматов команд XOR, используемых для выполнения поразрядной операции «Исключающее ИЛИ» с участием содержимого регистра общего назначения с номером *N* и константы *kk* или содержимого двух регистров с номерами *N* и *M*, представлена в таблице 6.

Форматы инструкций *LOAD*, предназначенных для загрузки константы или содержимого какого-либо регистра в выбранный регистр общего назначения, в новой редакции приведены в таблице 7.

Группа арифметических команд ядра PicoBlaze, предназначенного для CoolRunner-II

В структуре полей арифметических команд ядра PicoBlaze, предназначенного для применения в кристаллах CoolRunner-II, произошли те же изменения (по сравнению с форматами аналогичных команд, приведенными ранее [2]), что и в логических инструкциях, рассмотренных в предыдущем разделе.

Модифицированные варианты форматов команд сложения *ADD* содержимого регистра с номером *N* и константы *kk* или содержимого двух регистров общего назначения с номерами *N* и *M* без учета переноса представлены в таблице 8.

Новая версия форматов инструкций *ADDCY*, предназначенных для вычисления суммы двух операндов с учетом значения флага переноса, полученного при выполнении предыдущей операции, приведена в таблице 9.

Таблица 3. Форматы команд безусловного и условного возврата из подпрограммы для ядра PicoBlaze, реализуемого на основе CoolRunner-II

Поле кода операции								Поле адреса								Мнемоника	Выполняемая операция
1	0	0	1	0	0	X	X	0	0	0	0	0	0	0	0	RETURN	Безусловный возврат из подпрограммы
1	0	0	1	0	1	0	0	0	0	0	0	0	0	0	0	RETURN Z	Возврат из подпрограммы при условии, что флаг ZERO Flag находится в установленном состоянии
1	0	0	1	0	1	0	1	0	0	0	0	0	0	0	0	RETURN NZ	Возврат из подпрограммы при условии, что флаг ZERO Flag находится в сброшенном состоянии
1	0	0	1	0	1	1	0	0	0	0	0	0	0	0	0	RETURN C	Возврат из подпрограммы при условии, что флаг CARRY Flag находится в установленном состоянии
1	0	0	1	0	1	1	1	0	0	0	0	0	0	0	0	RETURN NC	Возврат из подпрограммы при условии, что флаг CARRY Flag находится в сброшенном состоянии
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	Номер разряда микрокоманды	

Таблица 4. Форматы команд поразрядных операций «Логическое И» ядра PicoBlaze, реализуемого на основе CoolRunner-II

Поле кода операции					Поле номера регистра			Поле константы								Мнемоника	Выполняемая операция
0	0	0	0	1	n	n	n	K	K	K	K	K	K	K	K	AND sN, kk	Поразрядное «Логическое И» содержимого регистра sN и константы kk
Поле кода операции					**Поле номера первого регистра**			**Поле номера второго регистра**			**Нулевые разряды**					**Мнемоника**	**Выполняемая операция**
0	1	0	0	1	n	n	n	m	m	m	0	0	0	0	0	AND sN,sM	Поразрядное «Логическое И» содержимого регистров sN и sM
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	Номер разряда микрокоманды	

Таблица 5. Форматы команд поразрядных операций «Логическое ИЛИ» ядра PicoBlaze, реализуемого на основе CoolRunner-II

Поле кода операции					Поле номера регистра			Поле константы								Мнемоника	Выполняемая операция
0	0	0	1	0	n	n	n	K	K	K	K	K	K	K	K	OR sN, kk	Поразрядное «Логическое ИЛИ» содержимого регистра sN и константы kk
Поле кода операции					**Поле номера первого регистра**			**Поле номера второго регистра**			**Нулевые разряды**					**Мнемоника**	**Выполняемая операция**
0	1	0	1	0	n	n	n	m	m	m	0	0	0	0	0	OR sN,sM	Поразрядное «Логическое ИЛИ» содержимого регистров sN и sM
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	Номер разряда микрокоманды	

Таблица 6. Форматы команд поразрядных операций «Исключающее ИЛИ» ядра PicoBlaze, реализуемого на основе CoolRunner-II

Поле кода операции					Поле номера регистра			Поле константы								Мнемоника	Выполняемая операция
0	0	0	1	1	n	n	n	K	K	K	K	K	K	K	K	XOR sN, kk	Поразрядное «Исключающее ИЛИ» содержимого регистра sN и константы kk
Поле кода операции					**Поле номера первого регистра**			**Поле номера второго регистра**			**Нулевые разряды**					**Мнемоника**	**Выполняемая операция**
0	1	0	1	1	n	n	n	m	m	m	0	0	0	0	0	XOR sN,sM	Поразрядное «Исключающее ИЛИ» содержимого регистров sN и sM
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	Номер разряда микрокоманды	

Таблица. 7. Форматы инструкции загрузки данных в регистр общего назначения ядра PicoBlaze, реализуемого на основе CoolRunner-II

Поле кода операции					Поле номера регистра			Поле константы								Мнемоника	Выполняемая операция
0	0	0	1	1	n	n	n	K	K	K	K	K	K	K	K	XOR sN, kk	Поразрядное «Исключающее ИЛИ» содержимого регистра sN и константы kk
Поле кода операции					**Поле номера первого регистра**			**Поле номера второго регистра**			**Нулевые разряды**					**Мнемоника**	**Выполняемая операция**
0	1	0	1	1	n	n	n	m	m	m	0	0	0	0	0	XOR sN,sM	Поразрядное «Исключающее ИЛИ» содержимого регистров sN и sM
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	Номер разряда микрокоманды	

Таблица 8. Форматы команд сложения двух операндов без учета переноса для ядра PicoBlaze, реализуемого на основе CoolRunner-II

Поле кода операции					Поле номера регистра			Поле константы								Мнемоника	Выполняемая операция
0	0	1	0	0	n	n	n	K	K	K	K	K	K	K	K	ADD sN, kk	Сложение содержимого регистра sN и константы kk
Поле кода операции					**Поле номера первого регистра**			**Поле номера второго регистра**			**Нулевые разряды**					**Мнемоника**	**Выполняемая операция**
0	1	1	0	0	n	n	n	m	m	m	0	0	0	0	0	ADD sN,sM	Сложение содержимого регистров sN и sM
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	Номер разряда микрокоманды	

Таблица 9. Форматы команд сложения двух операндов с учетом переноса для ядра PicoBlaze, реализуемого на основе CoolRunner-II

Поле кода операции					Поле номера регистра			Поле константы								Мнемоника	Выполняемая операция
0	0	1	0	1	n	n	n	K	K	K	K	K	K	K	K	ADDCY sN, kk	Сложение содержимого регистра sN и константы kk с учетом переноса
Поле кода операции					**Поле номера первого регистра**			**Поле номера второго регистра**			**Нулевые разряды**					**Мнемоника**	**Выполняемая операция**
0	1	1	0	1	n	n	n	m	m	m	0	0	0	0	0	ADDCY sN,sM	Сложение содержимого регистров sN и sM с учетом переноса
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	Номер разряда микрокоманды	

Таблица 10. Форматы команд вычитания без учета заема для ядра PicoBlaze, реализуемого на основе CoolRunner-II

Поле кода операции					Поле номера регистра			Поле константы								Мнемоника	Выполняемая операция
0	0	1	1	0	n	n	n	K	K	K	K	K	K	K	K	SUB sN, kk	Вычитание из содержимого регистра sN константы kk
Поле кода операции					**Поле номера первого регистра**			**Поле номера второго регистра**			**Нулевые разряды**					**Мнемоника**	**Выполняемая операция**
0	1	1	1	0	n	n	n	m	m	m	0	0	0	0	0	SUB sN,sM	Вычитание содержимого регистра sM из содержимого регистра sN
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	Номер разряда микрокоманды	

Таблица 11. Форматы инструкций вычитания с учетом заема для ядра PicoBlaze, реализуемого на основе CoolRunner-II

Поле кода операции					Поле номера регистра			Поле константы								Мнемоника	Выполняемая операция
0	0	1	1	1	n	n	n	K	K	K	K	K	K	K	K	SUBCY sN, kk	Вычитание из содержимого регистра sN константы kk с учетом заема
Поле кода операции					**Поле номера первого регистра**			**Поле номера второго регистра**			**Нулевые разряды**					**Мнемоника**	**Выполняемая операция**
0	1	1	1	1	n	n	n	m	m	m	0	0	0	0	0	SUBCY sN,sM	Вычитание содержимого регистра sM из содержимого регистра sN с учетом заема
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	Номер разряда микрокоманды	

Таблица 12. Форматы команд логического или циклического сдвига данных ядра PicoBlaze, реализуемого на основе CoolRunner-II

Поле кода операции					Поле номера регистра			Поле направления сдвига				Поле типа сдвига			Мнемоника	Выполняемая операция
1	0	1	0	0	n	n	n	0	0	0	0	1	1	0	SR0 sN	Логический сдвиг содержимого регистра sN вправо на один разряд с записью 0
1	0	1	0	0	n	n	n	0	0	0	0	1	1	1	SR1 sN	Логический сдвиг содержимого регистра sN вправо на один разряд с записью 1
1	0	1	0	0	n	n	n	0	0	0	0	1	0	1	0	SRX sN	Логический сдвиг содержимого регистра sN вправо с сохранением последнего разряда
1	0	1	0	0	n	n	n	0	0	0	0	1	0	0	0	SRA sN	Циклический сдвиг содержимого регистра sN вправо через разряд переноса/заема
1	0	1	0	0	n	n	n	0	0	0	0	1	1	0	0	RR sN	Циклический сдвиг содержимого регистра sN вправо без участия бита переноса
1	0	1	0	0	n	n	n	0	0	0	0	0	1	1	0	SL0 sN	Логический сдвиг содержимого регистра sN влево на один разряд с записью 0
1	0	1	0	0	n	n	n	0	0	0	0	0	1	1	1	SL1 sN	Логический сдвиг содержимого регистра sN влево на один разряд с записью 1
1	0	1	0	0	n	n	n	0	0	0	0	0	0	1	0	SLX sN	Логический сдвиг содержимого регистра sN влево с сохранением последнего разряда
1	0	1	0	0	n	n	n	0	0	0	0	0	0	0	0	SLA sN	Циклический сдвиг содержимого регистра sN влево через разряд переноса/заема
1	0	1	0	0	n	n	n	0	0	0	0	0	1	0	0	RL sN	Циклический сдвиг содержимого регистра sN влево без участия бита переноса
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	Номер разряда микрокоманды	

Форматы инструкций *SUB*, используемых для выполнения операции вычитания с участием тех же операндов, что и в командах сложения, без учета заема, в новой редакции представлены в таблице 10.

Модифицированные варианты форматов команд *SUBCY*, предназначенных для вычисления разности двух операндов с учетом значения заема, образовавшегося при выполнении предыдущей операции, приведены в таблице 11.

Команды сдвига данных для ядра PicoBlaze, предназначенного для кристаллов CoolRunner-II

В формате команд, предназначенных для выполнения операций сдвига данных, изменилась длина полей кода операции и номера регистра. Преобразованные форматы инструкций логического (арифметического) и циклического сдвига данных, находящихся в регистре общего назначения с указанным номером, представлены в таблице 12.

Команды ввода-вывода ядра PicoBlaze для CoolRunner-II

Структура инструкций ввода-вывода, используемых для организации чтения данных из входного порта в заданный регистр общего назначения и передачи информации из указанного регистра в выходной порт, отличается от структуры аналогичных команд, приведенной ранее [2], длиной полей кода операции и номеров регистров. В новой редакции команд ввода-вывода указанные поля содержат соответственно пять и три двоичных разряда.

Новые варианты форматов команд ввода-вывода с различными видами адресации входных и выходных портов приведены в таблице 13.

Команды обслуживания прерываний ядра PicoBlaze для CoolRunner-II

В формате инструкций обслуживания прерываний изменилось взаимное расположение и длина поля кода операции и поля режима обработки прерываний.

Новая редакция форматов команд возврата из процедуры обслуживания прерываний *RETURNI* и установки режима обработки прерываний в программе *ENABLE INTERRUPT* и *DISABLE INTERRUPT* представлена в таблице 14.

Для практического освоения рассмотренного варианта микропроцессорного ядра PicoBlaze и для аппаратной отладки проектов, включающих это ядро, можно воспользоваться инструментальным комплектом *CoolRunner-II Design Kit*, возможности которого были рассмотрены на страницах «КиТ» [9].

На этом завершается рассмотрение характеристик, архитектуры и системы команд встраиваемых восьмиразрядных микропроцессорных модулей семейства PicoBlaze. В следующей публикации цикла будут обсуждаться вопросы использования ассемблера для данного семейства микропроцессорных ядер.

Литература

1. Зотов В. PicoBlaze — семейство восьмиразрядных микропроцессорных ядер, реализуемых на основе ПЛИС фирмы Xilinx // Компоненты и технологии. 2003. № 4.
2. Зотов В. Система команд микропроцессорного ядра PicoBlaze, реализуемого на основе ПЛИС семейств Spartan-II, Spartan-IIE, Virtex, Virtex-E // Компоненты и технологии. 2003. № 5.
3. Зотов В. Особенности микропроцессорного ядра PicoBlaze, предназначенного для применения в проектах, реализуемых на основе ПЛИС семейства Virtex-II // Компоненты и технологии. 2003. № 6.
4. Зотов В. CoolRunner-II — новое поколение высокопроизводительных ПЛИС CPLD фирмы Xilinx с микромощным потреблением // Схемотехника. 2003. № 5–10.
5. Зотов В. Проектирование цифровых устройств на основе ПЛИС фирмы Xilinx в САПР WebPack ISE. М.: Горячая линия — Телеком. 2003.
6. Зотов В. ModelSim — система HDL-моделирования цифровых устройств // Компоненты и технологии. 2002. № 6.
7. Зотов В. Функциональное моделирование цифровых устройств, проектируемых на базе ПЛИС фирмы Xilinx в среде САПР WebPACK ISE // Компоненты и технологии. 2002. № 7.
8. Зотов В. Временное моделирование цифровых устройств, проектируемых на базе ПЛИС фирмы Xilinx в среде САПР WebPACK ISE // Компоненты и технологии. 2002. № 8.
9. Зотов В. Инструментальный комплект *CoolRunner-II Design Kit* для практического освоения методов программирования ПЛИС семейств CPLD фирмы Xilinx // Компоненты и технологии. 2003. № 2.

Таблица 13. Форматы команд ввода-вывода ядра PicoBlaze, реализуемого на основе CoolRunner-II

Поле кода операции					Поле номера регистра			Поле адреса порта ввода/вывода								Мнемоника	Выполняемая операция
1	0	0	0	0	n	n	n	K	K	K	K	K	K	K	K	INPUT sN, kk	Чтение данных из порта ввода/вывода с адресом kk в регистр sN
1	0	0	0	1	n	n	n	K	K	K	K	K	K	K	K	OUTPUT sN, kk	Запись данных из регистра sN в порт ввода/вывода с адресом kk
Поле кода операции					**Поле номера первого регистра**			**Поле номера второго регистра**			**Нулевые разряды**					**Мнемоника**	**Выполняемая операция**
1	1	0	0	0	n	n	n	m	m	m	0	0	0	0	0	INPUT sN,(sM)	Чтение данных из порта ввода/вывода с адресом, определяемым регистром sM, в регистр sN
1	1	0	0	1	n	n	n	m	m	m	0	0	0	0	0	OUTPUT sN,(sM)	Запись данных из регистра sN в порт с адресом, определяемым регистром sM
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	Номер разряда микрокоманды	

Таблица 14. Форматы команд обслуживания прерываний ядра PicoBlaze, реализуемого на основе CoolRunner-II

Поле кода операции					Поле режима обработки прерываний											Мнемоника	Выполняемая операция
1	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	RETURNI ENABLE	Возврат из процедуры обработки и установка режима запрета прерывания
1	0	1	1	0	0	0	0	0	0	0	0	0	0	0	1	RETURNI DISABLE	Возврат из процедуры обработки и установка режима разрешения прерывания
1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	ENABLE INTERRUPT	Установка режима разрешения прерывания
1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	1	DISABLE INTERRUPT	Установка режима запрета прерывания
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	Номер разряда микрокоманды	

Разработка программ на языке ассемблера

для семейства микропроцессорных ядер PicoBlaze

В предыдущих публикациях данного цикла [1–4] были рассмотрены архитектура и система команд элементов семейства встраиваемых 8-разрядных микропроцессорных ядер PicoBlaze, которые предназначены для использования в проектах, выполняемых на основе различных серий ПЛИС фирмы Xilinx. Настоящая статья посвящена вопросам подготовки и трансляции программ на языке ассемблера для этого семейства микропроцессорных ядер.

Валерий Зотов

walerry@euro.ru

Подготовка исходного текста программы на языке ассемблера микропроцессорного ядра PicoBlaze

Для подготовки исходного текста программы на языке ассемблера можно воспользоваться любым универсальным текстовым редактором, поддерживающим стандартный текстовый формат. Наиболее целесообразно для этих целей использовать программы Notepad или Wordpad, которые являются стандартными принадлежностями операционных систем Windows.

В тексте программы на языке ассемблера микропроцессорного ядра PicoBlaze могут присутствовать следующие элементы:

- команды микропроцессорного ядра;
- директивы ассемблера;
- метки;
- комментарии.

Команды микропроцессорного ядра — это инструкции, предназначенные для выполнения исполнительным модулем PicoBlaze операций, которые предусмотрены алгоритмом разрабатываемой программы. Синтаксис этих команд совпадает с мнемонической формой записи инструкций микропроцессорных ядер PicoBlaze, рассмотренных ранее [2–4]. В процессе трансляции ассемблер преобразует команды микропроцессорного ядра в исполняемый код.

Директивы представляют собой команды, предназначенные для управления работой ассемблера и процессом трансляции. Директивы ассемблера не транслируются в исполняемый код. Более подробно назначение и использование директив ассемблера микропроцессорного ядра PicoBlaze рассматривается в следующем разделе.

Метки предназначены для идентификации строк программы, содержащих инструкции микропроцессорного ядра, на которые имеются ссылки из других команд. Метки представляют собой последовательность символов, оканчивающуюся двоеточием, которая располагается перед текстом инструкции. В составе меток могут использоваться прописные и строчные буквы латинского алфавита (A–Z, a–z), цифры (0–9) и символы подчеркивания (_). Метки не должны содержать пробелов или символов-разделителей, как, например, точки, запятые, точки с запятой. В качестве меток следует задавать мнемонические идентификаторы, которые повышают информативность разрабатываемой программы. Метки рекомендуется использовать в инструкциях перехода JUMP и вызова подпрограмм CALL вместо явного указания адреса перехода или адреса вызываемой процедуры. Таким образом, при написании и редактировании программы не требуются вычисления абсолютных значений адресов команд в ППЗУ микропрограмм. Применение меток не только позволяет избежать ошибок, связанных с указанием некорректных значений адресов в командах, но и обеспечивает мобильность разрабатываемой программы за счет отсутствия ее привязки к конкретным адресам программной памяти.

Комментарии используются, как правило, для внесения различных пояснений в разрабатываемую программу, которые облегчают ее чтение и понимание. Комментарии не оказывают никакого влияния на процесс трансляции программы и формируемый исполняемый код. Комментарием считается любая последовательность символов, которая начинается с точки с запятой. Ассемблер микропроцессорного ядра PicoBlaze полностью игнорирует ту часть строки, которая расположена вслед за точкой с запятой.

При написании программ на языке ассемблера необходимо соблюдать следующие правила. Каждая команда микропроцессорного ядра или директива ассемблера должна располагаться в отдельной строке. Параметры инструкций микропроцессорного ядра или директив ассемблера отделяются от текста команд пробелом. Если формат инструкции или директивы требует указания двух параме-

тров, то второй параметр отделяется от первого запятой и пробелом. В исходном тексте разрабатываемой программы не следует оставлять пустых строк. Если все же требуется включить пустую строку в текст программы, например, для разделения основной части программы и подпрограммы, то следует сделать это в виде комментария. Для этого в любую позицию пустой строки нужно вставить точку с запятой.

По окончании редактирования исходного текста разрабатываемой программы необходимо сохранить его в виде файла на диске. Название этого файла должно состоять не более чем из восьми символов и соответствовать требованиям операционной системы MS-DOS. Расширение создаваемого файла, в который записывается текст программы, зависит от используемой версии ассемблера. В свою очередь, версия ассемблера определяется типом микропроцессорного ядра PicoBlaze, для которого разрабатывается программа. Для микропроцессорных ядер, реализуемых на основе ПЛИС семейств Spartan-II, Spartan-IIE, Virtex, Virtex-E и Virtex-II, исходный текст программы должен содержаться в файле с расширением *psm*. Текст программы на языке ассемблера для микропроцессорного ядра PicoBlaze, используемого в кристаллах семейства CoolRunner-II, следует записывать в файл с расширением *asm*. Создаваемый файл с исходным текстом программы должен располагаться в том же разделе диска, что и соответствующая программа-ассемблер. Для этих целей рекомендуется использовать рабочий каталог проекта, разрабатываемого в среде САПР фирмы Xilinx серии ISE (Integrated Synthesis Environment) [5], в составе которого используется соответствующее микропроцессорное ядро. Кроме собственно программы ассемблера, в этот каталог для проектов, выполняемых на основе ПЛИС семейств Spartan-II, Spartan-IIE, Virtex, Virtex-E и Virtex-II, необходимо переписать файлы шаблонов ROM_form.vhd и ROM_form.coe.

Директивы ассемблера микропроцессорного ядра PicoBlaze

Ассемблер микропроцессорного ядра PicoBlaze поддерживает три директивы: *CONSTANT*, *NAMEREG* и *ADDRESS*.

Директива *CONSTANT* предназначена для декларации и определения значения константы, которая может использоваться в качестве параметра в командах микропроцессорной программы. Константы, определенные с помощью данной директивы, могут выступать в качестве операндов в арифметических и логических инструкциях и в качестве адресов входных и выходных портов в инструкциях ввода-вывода. Применение константы вместо явного числового значения позволяет, прежде всего, повысить информативность программы, написанной на языке ассемблера. Кроме того, использование констант облегчает процесс отладки и редактирования разрабатываемой программы. Если константа представляет параметр, который многократно встречается в различных местах программы, то для изменения значения этого параметра достаточно отредактировать численное значение в строке декларации константы, вместо того чтобы вносить исправления во всех командах, где используется данный параметр. Тем самым, исключается возможность появления в программе ошибок, которые возникают из-за того, что не во всех инструкциях, где встречается данный параметр, указано его корректное значение.

Формат командной строки для директивы *CONSTANT* выглядит следующим образом.

```
CONSTANT <идентификатор_константы>, <числовое_значение_константы >
```

В состав идентификатора константы могут входить прописные и строчные буквы латинского алфавита (A–Z, a–z), цифры (0–9) и символы подчеркивания (_). При этом в имени константы не должно быть пробелов или каких-либо символов-разделителей. Рекомендуется использовать мнемонические идентификаторы констант, отражающие функциональное назначение соответствующего параметра. Константа, определяемая с помощью директивы *CONSTANT*, содержит восемь двоичных разрядов. Числовое значение константы указывается в виде последовательности, состоящей из двух шестнадцатеричных символов (0–9, A–F, a–f). Директива *CONSTANT* может располагаться в любом месте программы. При этом ее действие распространяется на всю программу, то есть определяемая константа может использоваться в командах, расположенных как после, так и до строки, содержащей соответствующую директиву *CONSTANT*.

Приведенный ниже фрагмент программы иллюстрирует применение директивы *CONSTANT*.

```
            CONSTANT max_value, 15
            CONSTANT adr_in_port, 1D
            CONSTANT adr_out_port, 1F
            INPUT s1, adr_in_port
            LOAD s2, s1
            SUB s2, max_value
            JUMP NC, out_max
            OUTPUT s1, adr_out_port
            JUMP next_inst
out_max:    LOAD s3, max_value
            OUTPUT s3, adr_out_port
next_inst:  LOAD s3, s1
```

В представленном фрагменте программы константа *adr_in_port* определяет значение адреса используемого входного порта ввода-вывода, а константа *adr_out_port* — выходного порта. Константа с идентификатором *max_value* используется для указания максимально допустимого значения некоторого контролируемого параметра, значения которого считываются из входного порта с адресом *adr_in_port*.

Директива *NAMEREG* позволяет переименовать любой из регистров общего назначения. По умолчанию в качестве названий регистров общего назначения используются идентификаторы, состоящие из символа *s* и порядкового номера регистра. Для повышения информативности разрабатываемой ассемблерной программы целесообразно задать новые, мнемонические имена для используемых регистров общего назначения. В качестве идентификаторов, определяемых разработчиком, рекомендуется указывать названия, отражающие функциональное назначение соответствующего регистра или характер данных, записанных в этот регистр. Использование мнемонических названий регистров делает более прозрачным процесс отладки микропроцессорных программ. Применение директивы *NAMEREG* позволяет также избежать ошибок, обусловленных просчетами при указании номеров регистров общего назначения в различных командах программы. Определение новых названий регистров с помощью директивы *NAMEREG* повышает мобильность разрабатываемых программ. Программа, написанная на ассемблере с использованием директивы *NAMEREG*, требует минимальных исправлений при ее переносе от одного элемента семейства микропроцессорных ядер PicoBlaze к другому. Вместо переопределения регистров во всех соответствующих инструкциях разработанной программы, в которых они используются, достаточно внести изменения только в строки, содержащие директивы *NAMEREG*.

Командная строка для директивы *NAMEREG* имеет следующий формат.

```
NAMEREG <текущий_идентификатор_регистра>, <новый_идентификатор_регистра>
```

Идентификаторы регистров общего назначения могут содержать те же символы (прописные и строчные буквы латинского алфавита и цифры), что и названия констант, определяемых с помощью директивы *CONSTANT* . В то же время, в названиях регистров не допускается использование пробелов или символов-разделителей. Следует обратить внимание на то, что новое обозначение регистра может использоваться только в той части программы, которая следует после строки, содержащей директиву *NAMEREG* для соответствующего регистра. Таким образом, директива *NAMEREG* не имеет обратного действия. После переименования регистра его старое обозначение не может использоваться как идентификатор этого регистра.

В качестве примера применения директивы *NAMEREG* далее приводится модифицированный вариант фрагмента программы, демонстрирующего использование директивы *CONSTANT*.

```
            CONSTANT max_value, 15
            CONSTANT adr_in_port, 1D
            CONSTANT adr_out_port, 1F
            NAMEREG s1, in_data_reg
            NAMEREG s2, differenc_reg
            INPUT in_data_reg, adr_in_port
            LOAD differenc_reg, in_data_reg
            SUB differenc_reg, max_value
            JUMP NC, out_max
            OUTPUT in_data_reg, adr_out_port
            JUMP next_inst
out_max:    LOAD s3, max_value
            OUTPUT s3, adr_out_port
next_inst:  LOAD s3, in_data_reg
```

В представленном фрагменте программы переопределены идентификаторы двух регистров общего назначения. Регистр s1, который выполняет функцию хранения данных,

считанных из входного порта ввода-вывода, обозначен как in_data_reg. Для регистра s2, используемого в операции вычисления разности текущего и максимального значений, с помощью директивы *NAMEREG* задано новое название — differenc_reg.

Директива *ADDRESS* предназначена для указания нового адреса, начиная с которого должны располагаться команды программы, следующие за строкой, содержащей эту директиву. Таким образом, данная директива позволяет явно указать адресное пространство для размещения в ППЗУ любой части разрабатываемой программы. Директиву *ADDRESS* целесообразно использовать для выделения соответствующих секций в адресном пространстве ППЗУ микропрограмм, предназначенных для записи подпрограмм. Кроме того, данная директива необходима для записи вектора обработки прерывания.

Формат командной строки для директивы ADDRESS выглядит следующим образом.

```
ADDRESS <числовое_значение_адреса>
```

Числовое значение адреса в директиве *ADDRESS* указывается в виде последовательности, состоящей из двух или трех шестнадцатеричных символов, в зависимости от используемого типа микропроцессорного ядра. Для ядер PicoBlaze, реализуемых на базе кристаллов семейств Spartan-II, Spartan-IIE, Virtex, Virtex-E и CoolRunner-II, адрес задается в виде двухразрядного шестнадцатеричного числа. В программах, предназначенных для микропроцессорного ядра семейства Virtex-II, значение адреса содержит три шестнадцатеричных разряда.

В качестве примера далее приводится фрагмент программы, который наглядно поясняет способ практического применения директивы *ADDRESS*.

```
                CONSTANT max_value, 15
                CONSTANT adr_in_port, 1D
                CONSTANT adr_out_port, 1F
                NAMEREG s1, in_data_reg
                NAMEREG s2, differenc_reg
                INPUT in_data_reg, adr_in_port
                LOAD differenc_reg, in_data_reg
                SUB differenc_reg, max_value
                JUMP NC, out_data
                JUMP next_inst
out_data:       OUTPUT in_data_reg, adr_out_port
next_inst:      LOAD s3, in_data_reg
                ; …
                ; …
                ADDRESS B0
interrupt_routine:  ADD differenc_reg, 01
                OUTPUT differenc_reg, adr_out_port
                RETURNI ENABLE
                ;
                ADDRESS FF
                JUMP interrupt_routine
```

ADDRESS FF предписывает ассемблеру транслировать код вектора прерывания по адресу FF. Необходимость этого обусловлена архитектурными особенностями микропроцессорных ядер PicoBlaze для семейств Spartan-II, Spartan-IIE, Virtex, Virtex-E и CoolRunner-II. В программах, которые разрабатываются для микропроцессорного ядра на основе семейства Virtex-II, код вектора прерывания должен находиться по адресу 3FF. С помощью директивы *ADDRESS B0* дается указание ассемблеру транслировать код процедуры обработки прерывания, начиная с адреса B0. Представленный ниже текст файла листинга, сформированного ассемблером при трансляции рассматриваемого фрагмента программы, отражает результаты воздействия директивы *ADDRESS* на генерируемый код. Первые две шестнадцатеричные цифры в начале каждой строки указывают адрес размещения кода соответствующей команды в ППЗУ микропрограмм.

```
Addr Code

                    CONSTANT max_value, 15
                    CONSTANT adr_in_port, 1D
                    CONSTANT adr_out_port, 1F
                    NAMEREG s1, in_data_reg
                    NAMEREG s2, differenc_reg
00  A11D            INPUT in_data_reg[s1], adr_in_port[1D]
01  C210            LOAD differenc_reg[s2], in_data_reg[s1]
02  6215            SUB differenc_reg[s2], max_value[15]
03  9D05            JUMP NC, out_data[05]
04  8106            JUMP next_inst[06]
05  E11F      out_data: OUTPUT in_data_reg[s1],adr_out_port[1F]
06  C310     next_inst: LOAD s3, in_data_reg[s1]
07                    ; …
07                    ; …
B0                    ADDRESS B0
B0  4201 interrupt_routine: ADD differenc_reg[s2], 01
B1  E21F              OUTPUT differenc_reg[s2],adr_out_port[1F]
B2  80F0              RETURNI ENABLE
B3                    ;
FF                    ADDRESS FF
FF  81B0              JUMP interrupt_routine[B0]
```

Трансляция программ, написанных на языке ассемблера микропроцессорного ядра PicoBlaze

Программа-ассемблер для микропроцессорного ядра PicoBlaze представляет собой DOS-приложение, которое функционирует в консольном режиме. Поэтому для его запуска рекомендуется активизировать сеанс DOS (режим командной строки) и установить в качестве текущего каталог, в котором располагаются ассемблер и транслируемая программа. Для выполнения этих операций целесообразно воспользоваться какой-либо программой управления файлами, например, управляющей оболочкой Windows Commander, которая позволяет быстро установить требуемый текущий раздел диска и запустить сеанс DOS.

Как уже упоминалось ранее [2–4], для каждого элемента семейства PicoBlaze используется своя версия ассемблера. Командная строка для запуска процесса трансляции разработанной программы состоит из названия соответствующей версии ассемблера и идентификатора файла, в котором содержится исходный текст транслируемой программы на языке ассемблера. Формат командной строки, активизирующей процедуру трансляции программ, предназначенных для микропроцессорного ядра PicoBlaze семейств Spartan-II, Spartan-IIE, Virtex, Virtex-E, выглядит следующим образом:

```
Kcpsm.exe <идентификатор_файла_с_исходным_текстом_программы_на_языке_ассемблера>[.psm]
```

Для трансляции программ микропроцессорного ядра PicoBlaze, встраиваемого в проекты, реализуемые на базе ПЛИС семейства Virtex-II, следует воспользоваться командной строкой, формат которой имеет следующий вид:

```
Kcpsm2.exe <идентификатор_файла_с_исходным_текстом_программы_на_языке_ассемблера>[.psm]
```

В приведенных выше форматах командной строки квадратные скобки служат для обозначения необязательного параметра. Таким образом, используемое по умолчанию расширение файла, содержащего исходный текст программы (PSM) можно не указывать.

Процедура трансляции программ микропроцессорного ядра PicoBlaze, используемого в кристаллах семейства CoolRunner-II, запускается с помощью командной строки, которая должна соответствовать следующему формату:

```
Asm.exe <идентификатор_файла_с_исходным_текстом_программы_на_языке_ассемблера>.asm
```

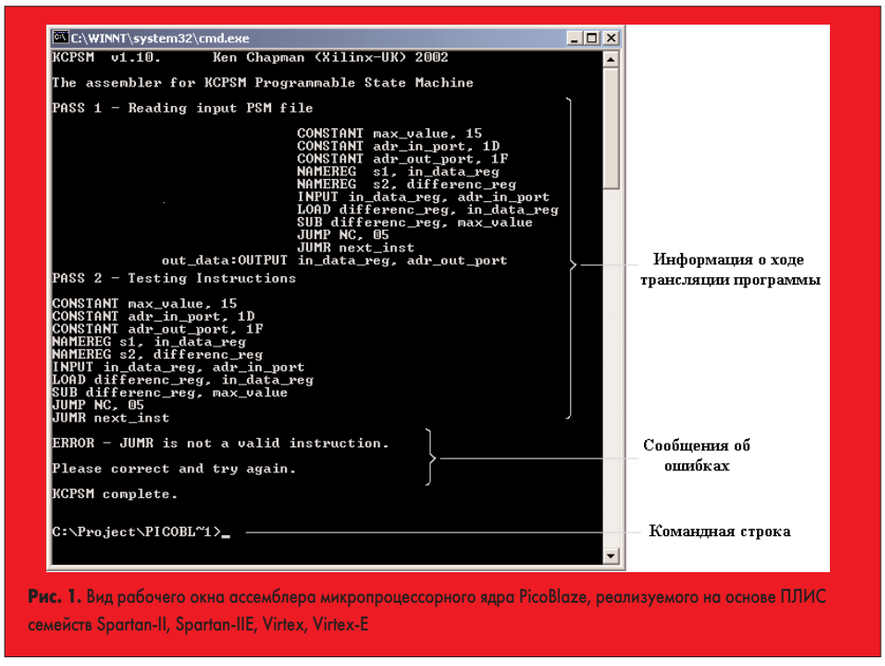

Рис. 1. Вид рабочего окна ассемблера микропроцессорного ядра PicoBlaze, реализуемого на основе ПЛИС семейств Spartan-II, Spartan-IIE, Virtex, Virtex-E

Процесс трансляции включает в себя несколько последовательных фаз. Информация о выполнении и результатах каждой фазы трансляции отображается непосредственно в окне DOS-приложения. В этом же окне выводятся сообщения о возможных ошибках, обнаруженных в транслируемой программе. На рис. 1 в качестве примера показан вид рабочего окна ассемблера KCPSM.

В случае отсутствия ошибок трансляция завершается выводом сообщения вида «KCPSM successful».

Файлы, формируемые ассемблером микропроцессорного ядра PicoBlaze

При успешном завершении процесса трансляции программы ассемблер формирует комплект файлов.

Рисунок 2 наглядно отображает информацию об исходных файлах, используемых при трансляции программ микропроцессорного ядра PicoBlaze, предназначенного для реализации в кристаллах семейств Spartan-II, Spartan-IIE, Virtex, Virtex-E, Virtex-II, и результирующих файлах, формируемых ассемблером KCPSM (KCPSM2).

Помимо основного исходного файла, содержащего текст программы на языке ассемблера, для выполнения трансляции необходимы файлы шаблонов ROM_form.vhd и ROM_form.coe. Файл ROM_form.vhd представляет собой шаблон описания содержимого ППЗУ микропрограмм на языке VHDL (VHSIC Hardware Description Language). В файле ROM_form.coe содержится шаблон описания содержимого ППЗУ микропрограмм в формате, воспринимаемом генератором ядер Xilinx CORE Generator.

При отсутствии ошибок в транслируемой программе ассемблерами KCPSM и KCPSM2 генерируется набор файлов, название большинства которых совпадает с идентификатором исходного файла, а расширение соответствует типу содержащейся в них информации. Всю совокупность файлов, формируемых этими версиями ассемблера, можно условно разбить на пять групп. К первой группе относятся файлы, в которые записывается содержимое в форматах, воспринимаемых средствами проектирования фирмы Xilinx серии ISE. В эту группу входят файлы описания содержимого программной памяти на языке VHDL и в формате, воспринимаемом генератором ядер Xilinx CORE Generator, которые имеют расширение *vhd* и *coe* соответственно. VHDL-описание содержимого ППЗУ микропрограмм используется на этапах синтеза и моделирования проектируемой системы. Вторую группу образуют файлы, описывающие содержимое ППЗУ микропрограмм в форматах, используемых утилитами, не входящими в состав САПР серии ISE. К этой группе относятся файлы, содержащие результаты трансляции в виде кодов в шестнадцатеричном и десятичном представлении, которые имеют расширение *hex* и *dec* соответственно. Третью группу составляют файлы отчетов о ходе

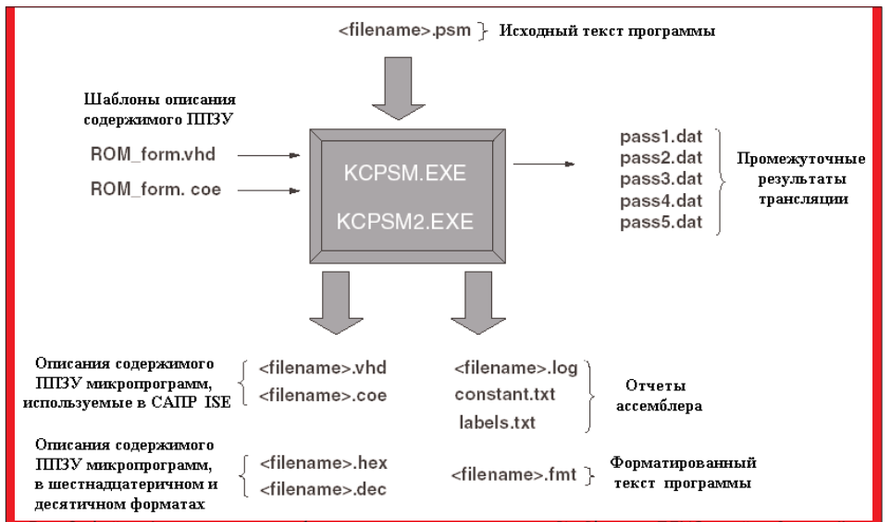

Рис. 2. Файлы, формируемые ассемблером микропроцессорного ядра PicoBlaze для ПЛИС семейств Spartan-II, Spartan-IIE, Virtex, Virtex-E, Virtex-II

и результатах трансляции. Файл с расширением *log* содержит детальную информацию о трансляции каждой строки ассемблерной программы. В файле *constant.txt* перечисляются все константы, используемые в программе, с указанием их значений. Файл *labels.txt* содержит информацию о метках, которые содержатся в программе, и адресах ППЗУ микропрограмм, которые им соответствуют. В четвертую группу входит единственный файл, имеющий расширение *fmt*, в котором записан отформатированный вариант исходного текста программы на языке ассемблера. К пятой группе относятся файлы *pass1.dat – pass5.dat*, содержащие промежуточные результаты, полученные на различных фазах процесса трансляции. Информация, которая содержится в этих файлах, может использоваться в процессе отладки программы.

Состав файлов, генерируемых ассемблером ASM, который предназначен для трансляции программ микропроцессорного ядра PicoBlaze, реализуемого на основе ПЛИС семейства CoolRunner-II, несколько отличается от рассмотренного выше. В первую очередь это связано с тем, что в данной версии ядра модуль программной памяти реализуется не внутри ПЛИС, а в виде внешнего элемента ПЗУ или ППЗУ. Для программирования такого элемента памяти ассемблер должен сформировать файл прошивки в формате, поддерживаемом различными типами программаторов. На рис. 3 показана структура комплекта выходных файлов, создаваемых ассемблером микропроцессорного ядра PicoBlaze, реализуемого на основе ПЛИС семейства CoolRunner-II, при успешном завершении процесса трансляции.

Все выходные файлы, формируемые ассемблером ASM, имеют одно и то же название, совпадающее с именем входного файла, в котором содержится исходный текст программы на языке ассемблера. В большинстве случаев основным выходным файлом является файл с расширением *mcs*, в котором содержится информация о прошивке внешнего ПЗУ или ППЗУ в формате Intel MCS-86. Файл, имеющий расширение *vhd*, представляет собой описание содержимого программной памяти на языке VHDL, соответствующего транслируемой программе. Этот файл используется в процессе отладки разрабатываемой программы методом моделирования ядра PicoBlaze в составе проек-

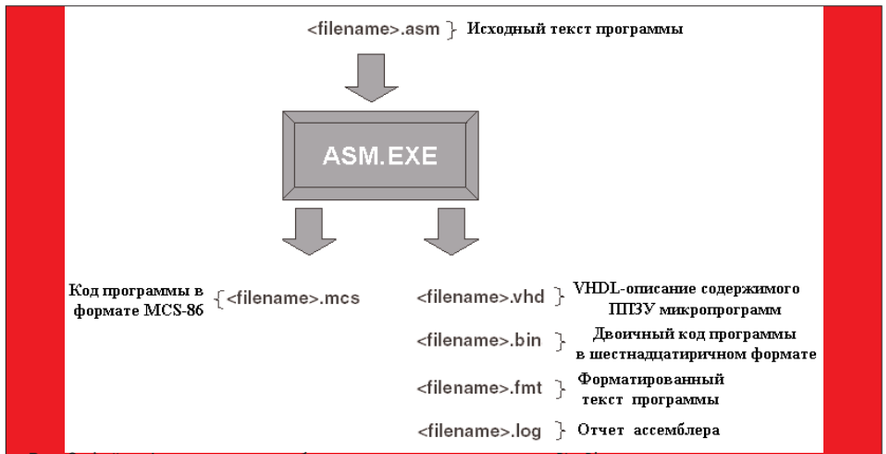

Рис. 3. Файлы, формируемые ассемблером микропроцессорного ядра PicoBlaze, реализуемого на основе ПЛИС семейства CoolRunner-II

тируемого устройства с помощью системы ModelSim XE [5–8]. Файл с расширением *bin* включает в себя результаты трансляции программы в виде двоичных кодов, которые представлены в шестнадцатеричном формате. Информация о выполнении и результатах процесса трансляции содержится в файле с расширением *log*. Форматированный вариант исходного текста программы на языке ассемблера записывается в файл, имеющий расширение *fmt*.

При отладке программ следует учитывать, что выходные файлы, формируемые различными версиями ассемблера, переписываются заново при каждом запуске процесса трансляции.

Типичные ошибки при написании программ на языке ассемблера микропроцессорного ядра PicoBlaze

В данном разделе рассматриваются наиболее распространенные ошибки, встречающиеся в исходном тексте программ на языке ассемблера для микропроцессорного ядра PicoBlaze. Для каждого типа ошибки приводится текст соответствующего сообщения, формируемого ассемблерами KCPSM и KCPSM2. Текст аналогичных сообщений ассемблера ASM несущественно отличается от образцов, представленных ниже, в основном меньшим объемом справочной информации.

Типовыми ошибками при написании программ на языке ассемблера микропроцессорного ядра PicoBlaze являются:

- опечатки в тексте команд микропроцессорного ядра или директив ассемблера;
- неправильные названия регистров, указываемые в качестве параметров команд;
- ошибки в указании параметров инструкций микропроцессорного ядра;
- отсутствие обязательных параметров команд микропроцессорного ядра или директив ассемблера;
- отсутствие метки строки, на которую имеются ссылки в командах программы;
- использование одной метки для двух или более строк;
- недопустимые значения числовых параметров директив ассемблера.

При выявлении ошибки в тексте команды микропроцессорного ядра или директивы ассемблер выводит на экран строку, формат которой выглядит следующим образом:

```
ERROR — <текст_неидентифицированной_команды_или_директивы> is not a valid instruction.
```

Обнаружение ошибки в названии используемого регистра сопровождается сообщением вида:

```
ERROR — Invalid register name: <неправильный_идентификатор_регистра>
Default register names are in the range 's0' to 'sF'. Note that NAMEREG directive replaces the default name, so check that user defined register names are consistant. User defined register names are case sensitive.
```

Если допущена ошибка при указании второго параметра инструкции микропроцессорного ядра или директивы, то ассемблер формирует следующее предупреждение:

```
ERROR — Invalid second operand: <неправильный_идентификатор_второго_параметра>
This does not match a valid register name or constant label. It is also invalid as an absolute value. Default register names are in the range 's0' to 'sF'. Note that NAMEREG directive replaces the default name, so check that user defined register names are consistant. Constant labels are defined using the CONSTANT directive. Absolute values must be specified as 2-digits hexadecimal in the range '00' to 'FF'. All user defined labels and names are case sensitive.
```

При обнаружении адреса порта ввода-вывода, выходящего за пределы допустимых значений, или несуществующей константы, используемой в качестве адреса порта ввода-вывода, выводится сообщение, которое имеет следующий вид.

```
ERROR — Invalid port address: <неправильный_адрес_порта_ввода-вывода>
This does not match a valid constant label. It is also invalid as an absolute port address. Constant labels are defined using the CONSTANT directive. Absolute port addresses must be specified as 2-digits hexadecimal in the range '00' to 'FF'. The second operand may also be a valid register name enclosed in brackets or a constant label. Default register names are in the range '(s0)' to '(sF)'. Note that NAMEREG directive replaces the default name. All user defined labels and names are case sensitive.
```

Если пропущен первый параметр команды микропроцессорного ядра или директивы, то ассемблер выводит строку, формат которой выглядит следующим образом:

```
ERROR — No operands specified for <инструкция,_в_которой_пропущен_параметр> instruction.
```

В случае отсутствия второго обязательного параметра команды микропроцессорного ядра или директивы строка сообщения принимает следующий вид:

```
ERROR — No second operand specified for <инструкция,_в_которой_пропущен_параметр> instruction.
```

Об отсутствии метки строки, на которую встречаются ссылки в командах перехода или вызова подпрограммы, или обнаружении неправильного значения адреса команды ассемблер информирует следующим сообщением:

```
ERROR — Address is not 2-digits: <текст_отсутствующей_метки>
Provide an absolute address or matching label. Note that labels are case sensitive. Absolute address must be in range '00' to 'FF'
```

Обнаружение в тесте программы двух или более строк с одинаковой меткой сопровождается следующим предупреждением:

```
ERROR — Duplicate label specified: <текст_повторно_используемой_метки>
All labels must be unique.
```

При выявлении ошибочного значения адреса, указанного в директиве *ADDRESS*, ассемблер выводит сообщение, которое выглядит следующим образом:

```
ERROR — Constant value '<ошибочное_значение_адреса>' is not 2-digits.
2-digit hexadecimal constant in range '00' to 'FF' required.
```

Если в директиве определения константы, указано значение, выходящее за допустимые границы, то ассемблер отреагирует следующим предупреждением:

```
ERROR — Constant value '<ошибочное_значение_константы>' is not valid.
2-digit hexadecimal constant in range '00' to 'FF' required.
```

На этом завершается рассмотрение встраиваемых микропроцессорных модулей семейства PicoBlaze. В следующей части данного цикла будет представлено семейство тридцатидвухразрядных микропроцессорных ядер MicroBlaze.

Литература

1. Зотов В. PicoBlaze — семейство восьмиразрядных микропроцессорных ядер, реализуемых на основе ПЛИС фирмы Xilinx // Компоненты и технологии. 2003. № 4.
2. Зотов В. Система команд микропроцессорного ядра PicoBlaze, реализуемого на основе ПЛИС семейств Spartan-II, Spartan-IIE, Virtex, Virtex-E // Компоненты и технологии. 2003. № 5.
3. Зотов В. Особенности микропроцессорного ядра PicoBlaze, предназначенного для применения в проектах, реализуемых на основе ПЛИС семейства Virtex-II // Компоненты и технологии. 2003. № 6.
4. Зотов В. Особенности микропроцессорного ядра PicoBlaze, предназначенного для применения в проектах, реализуемых на основе ПЛИС семейства CoolRunner-II // Компоненты и технологии. 2003. № 7.
5. Зотов В. Проектирование цифровых устройств на основе ПЛИС фирмы Xilinx в САПР WebPack ISE. М.: Горячая линия — Телеком. 2003.
6. Зотов В. ModelSim — система HDL-моделирования цифровых устройств // Компоненты и технологии. 2002. № 6.
7. Зотов В. Функциональное моделирование цифровых устройств, проектируемых на базе ПЛИС фирмы Xilinx в среде САПР WebPACK ISE // Компоненты и технологии. 2002. № 7.
8. Зотов В. Временное моделирование цифровых устройств, проектируемых на базе ПЛИС фирмы Xilinx в среде САПР WebPACK ISE // Компоненты и технологии. 2002. № 8.