В направлении развития "от языков проектирования" наибольший интерес представляет Superlog, предложенный фирмой Co-design Automation. Superlog является расширением Verilog. Авторы проекта считают, что "проектировщики, работающие с языками проектирования, чрезвычайно легко адаптируются к новым возможностям"[49]. Среди привлекательных свойств языка (а точнее, системы моделирования Systemsim на его базе) авторы проекта также указывают на допустимость смешивания в одной программе фрагментов, записанных на C, Verilog и Superlog, мощные средства верификации.

Время покажет, какое из этих направлений будет более плодотворным, а может быть, как с VHDL и Verilog, возникнет ситуация их параллельного использования. В ближайшей же перспективе они, безусловно, будут развиваться параллельно.

Разумеется, развитие языков системного уровня не сможет привести к умалению роли Verilog и VHDL. Они по-прежнему остаются предпочтительными для детального моделирования аппаратуры и для создания проектов с преимущественно аппаратными средствами обработки. Более того, с повышением эффективности компиляторов они могут частично вытеснять языки ассемблерного уровня и схемотехнические приемы проектирования.

ГЛАВА 4

Примеры проектирования устройств с применением ПЛИС

4.1. Проектирование операционных устройств

4.1.1. Операционные устройства с микропрограммным управлением

Под операционным устройством понимают вычислительный узел, способный многократно выполнять любое преобразование из набора, предусмотренного для этого узла, каждый раз, когда на него поступает сигнал, инициирующий преобразование (команда). Часто реализация команды требует последовательного выполнения нескольких шагов. Это может быть связано с тем, что на некоторых шагах используются результаты, полученные на предыдущих шагах, или с тем, что данные поступают, а результаты должны выдаваться в определенной последовательности, или с наличием ограничений на затраты оборудования.

При высоких требованиях к производительности элементарные действия стараются распределять между несколькими параллельно работающими блоками, причем для алгоритмов, предусматривающих использование результатов некоторого шага на последующих шагах, применяют конвейерную реализацию (пример будет представлен далее). Однако при умеренных требованиях по производительности с целью уменьшения объема оборудования однотипные операции выполняют последовательно в одном и том же блоке, даже если операции функционально независимы (функционально независимыми называются операции, которые не используют результатов друг друга и, в принципе, могут исполняться параллельно).

Часто становится оправданной частичная перестройка функций операционных блоков в процессе исполнения команды. Элементарное действие в этом

случае называют микрооперацией. Несколько одновременно выполняемых действий объединяются микрокомандой. Последовательность микрокоманд, исполняемых в процессе реализации команды, называют микропрограммой.

Языки проектирования дискретных устройств, в принципе, позволяют описать любой алгоритм в последовательной форме, т. е. через последовательность операторов присваивания и принятия решений. Однако более удобный и чаще используемый подход к проектированию операционных блоков вычислительных устройств заключается в разделении устройства на два блока — устройство управления и операционный блок. При этом в операционном блоке выполняются преобразования данных, а устройство управления обеспечивает необходимую последовательность микроопераций, выполняемых в операционном блоке, передавая на входы операционного блока управляющие сигналы (микрокоманды). Последовательность действий, а значит управляющих сигналов, зависит как от результатов выполнения операций в операционном блоке, так и от внешних сигналов.

Управляющий блок удобно описывать в форме конечного автомата того или иного типа. Операционный блок обычно представляют как набор регистров, логических блоков (как правило, многофункциональных, управляемых), буферных схем, а также коммутируемых связей между ними.

Разработку операционных устройств рекомендуется выполнять в следующем порядке.

Составление содержательной граф-схемы алгоритма и разработка структуры операционного блока

Граф-схема алгоритма представляется совокупностью последовательно исполняемых операторов: так называемых операторных вершин, отображающих преобразования данных, и условных вершин, отражающих проверки входов и результатов исполнения предыдущих шагов с целью выбора пути продолжения исполнения. На основании анализа требуемого набора операций выделяют набор используемых функциональных модулей. Составление граф-схемы достаточно интуитивно. Тем не менее, можно обратить внимание на следующие моменты.

Нужно иметь в виду, что состав микроопераций, используемых при составлении алгоритма, учитывает наличие определенного набора стандартных решений, имеющегося в библиотеках системы проектирования. Для реализации совокупностей микроопераций, не представленных в библиотеке, следует предусмотреть разработку специализированных блоков, предполагая их описание в процессе создания проекта в поведенческой или иной форме.

Операторная вершина может содержать либо одну микрооперацию, либо несколько совместимых во времени операций. Объединение нескольких микроопераций в одной операторной вершине соответствует их одновременному исполнению и может стать средством повышения производительности проектируемого устройства. Разнесение совместимых операций в несколько последовательных операторных вершин позволяет реализовать их исполнение на одних и тех же компонентах, т. е. сократить количество оборудования за счет потери быстродействия. Поэтому важным этапом, в общем случае, является определение необходимого числа модулей того или иного типа для реализации требуемой микропрограммы.

Максимальная производительность достигается, когда принятая структура не накладывает ограничений на совместимость микроопераций. Это выполняется, лишь если каждый узел преобразования используется для выполнения функционально эквивалентных операций только над одним словом данных, но не над разными словами. За каждым регистром данных, сохраняющим результат для последующих шагов, закрепляется индивидуальное устройство логических преобразований. Тогда все функционально совместимые операции могут исполняться в одном такте [15].

Предельным противоположным случаем является подход, основанный на обобществлении преобразований (прежде всего, комбинационных схем) по отношению ко всем регистрам хранения промежуточных данных. На любое элементарное преобразование или пересылку отводится отдельный такт работы устройства, т. е. достигаются наименьшие затраты оборудования при минимальной производительности. Возможны смешанные варианты.

Даже для простейших двуместных преобразований слов данных, например сложения слов, возможны как параллельное преобразование всех разрядов, так и последовательная обработка групп разрядов в преобразователе меньшей разрядности с разделением во времени функций входов и выходов. Правда, для ПЛИС такой подход применительно к простой логике обычно неоправдан, т. к. базовая ячейка выполняет одновременно функции логики и функции хранения результата для последующих шагов алгоритма.

Рассмотрим более показательный пример — реализацию умножения. Независимо от способа реализации умножение двух чисел, представленных в двоичном коде, требует использования $n\times(m-1)$ полных одноразрядных сумматоров, где m и n — число разрядов сомножителей. Реализация умножения с использованием n-разрядных сумматоров порождает широкую номенклатуру параллельных умножителей, в том числе матричную и пирамидальную структуры, смешанные варианты [27]. Устройства этого класса способны выполнять умножения за один такт работы схемы. Подобными характеристиками обладают схемы, построенные на основе набора параллельных умножителей меньшей разрядности с последующим суммированием частичных произведений (см., например, листинг 3.53). Для повышения допустимой частоты поступления сомножителей используют конвейеризацию (см. листинг 3.30). Альтернативная версия — использование единственного сумматора. В этом случае умножение выполняется за число тактов, равное разрядности множителя. В каждом такте к ранее накопленному результату прибавляется произведение сдвинутого кода множимого на очеред-

ной разряд множителя. Известно достаточно много промежуточных вариантов, характеризующихся различными аппаратурными затратами, для которых время исполнения варьируется от одного такта до числа тактов, равного разрядности одного из сомножителей и даже больше, если выделены отдельные такты для сдвига операндов и частичных сумм.

Представленные примеры иллюстрируют тот факт, что перед разработчиком почти всегда имеется определенное поле выбора для определения соотношения затрат оборудования и производительности. Иногда приходится в процессе проектирования возвращаться к этапу формирования схемы алгоритма, если полученный вариант не удовлетворяет системным требованиям.

Порядок разработки операционных устройств с микропрограммным управлением иллюстрируется примером синтеза устройства умножения последовательного типа с одновременным анализом двух битов множителя. Пусть на этапе анализа системных требований установлено, что умножение должно инициироваться внешним сигналом и занимать около $N/2$ тактов работы системы. Тогда можно принять последовательный алгоритм с умножением в каждом такте множимого на два разряда множителя и накоплением полученных произведений в регистре результата.

Реализуемый в рассматриваемом примере алгоритм (граф-схема алгоритма представлена на рис. 4.1) состоит в том, что по сигналу `start` сомножители загружаются во внутренние регистры операционного блока — множимое в *Rg*1, а множитель в *Rg*2 и обнуляются регистр результата *Rg*3 и счетчик циклов *CT*. После этого выполняется $N/2+1$ циклов, где N — разрядность множителя, в каждом из которых вычисляется частичное произведение, т. е. произведение множимого на число, представленное очередной парой битов множителя, причем эти частичные произведения накапливаются в регистре результата.

В каждом i-м цикле оцениваются показатели

$$S_i = a1 \times M(2 \times i - 1) + a0 \times M(2 \times i - 2) + a_C \times C_{i-1};$$

$$C_i = \begin{cases} 0, \text{ если } S_i < 2; \\ 1, \text{ в противном случае;} \end{cases}$$

где M(j) — j-й разряд множителя; C_{i-1} — бит переноса, возникающий в предыдущем цикле при анализе предшествующей пары битов множителя; $a1$, $a0$ и a_C — относительные веса соответствующих битов, причем $a0 = a_C = 1$, $a1 = 2$.

Можно видеть, что показатель S_i представляет сумму численного эквивалента двухразрядного среза кода множителя и дополнительного члена $a_C \times C_{i-1}$, который равен единице, если значение подобного показателя в предыдущем цикле (S_{i-1}) таково, что не позволяет непосредственно сформировать в этом цикле частичное произведение. Соответствующая недостача должна корректироваться в текущем цикле.

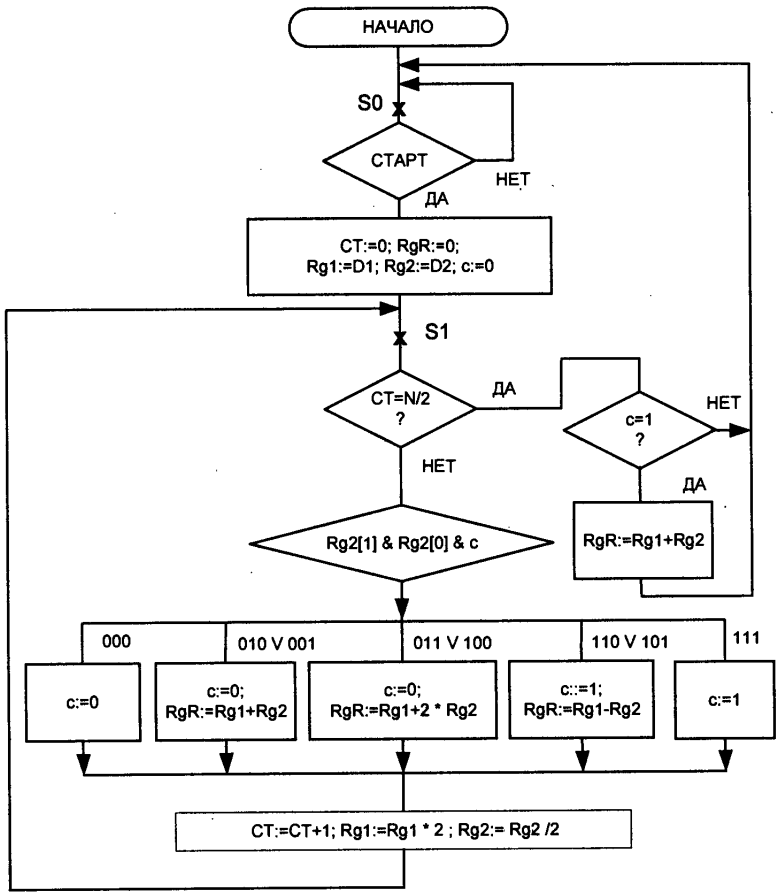


Рис. 4.1. Граф-схема алгоритма умножения

аждый цикл предусматривает умножение множимого на показатель S_i и рибавление такого произведения к ранее накопленной сумме частичных роизведений с учетом фактического веса учитываемых в текущем цикле зрядов множителя. Логика двухразрядного умножения сводится к слеующему.

сли $S_i = 0$ или $S_i = 4$, то содержимое *RgR* не изменяется. Если $S_i = 1$, то накопленной сумме прибавляется сдвинутый на $2 \times (i-2)$ разрядов код ножимого. Если $S_i = 2$, то к содержимому *Rg*3 прибавляется сдвинутый на

$2\times(i-1)$ разрядов код множимого. Если $S_i = 3$, то из содержимого ***RgR*** вычитается сдвинутый на $2\times(i-2)$ разрядов код множимого. Бит *C* устанавливается в единицу, если $S_i > 2$, и в нуль в остальных случаях, в том числе перед первым циклом.

Для удобства технической реализации код множимого в конце каждого цикла сдвигается на два разряда в сторону старших разрядов, а код множителя на два разряда в сторону младших, что позволяет избежать необходимости применения сдвигателей на произвольное переменное число разрядов. Тогда реализация каждого шага алгоритма накапливающего суммирования предусматривает анализ двух младших битов в регистре множителя. В зависимости от этого может выполняться одно из следующих действий: прибавление к накопленной частичной сумме кода, находящегося в данный момент в регистре множимого, вычитание из накопленной суммы этого же кода, либо прибавление к накопленной сумме удвоенного (сдвинутого на один разряд) кода регистра множимого.

Заметим, что общее число операций вида прибавление или отнимание кода множимого и результата может быть на единицу больше величины $N/2$. Дополнительное сложение требуется, если в цикле $N/2$ образуется недостача, т. е. *C* установлен в единицу. Это же является причиной необходимости увеличения разрядности регистра множимого и регистра результата до значения $N/2+2$.

Для рассматриваемого примера необходимо использовать следующий набор блоков:

- $(2\times N+2)$-разрядный регистр результата;
- N-разрядный сдвигающий регистр множителя с возможностью параллельной загрузки и со сдвигом на два разряда в сторону младших разрядов в каждом такте;
- $(2\times N+2)$-разрядный сдвигающий регистр множимого с возможностью параллельной загрузки и со сдвигом на два разряда в сторону старших разрядов в каждом такте;
- счетчик циклов, имеющий $N/2+1$ состояние;
- комбинационная схема, которая имеет два информационных $(2\times N+2)$-разрядных входа и один управляющий вход. В зависимости от управляющих сигналов выход комбинационной схемы может формироваться в одном из пяти вариантов:
  - нулевой код;
  - код с одного из входов (назовем его первым);
  - сумма кодов, поданных на входы;
  - разность кодов, поданных на входы;

  - сумма кода, поступающего на первый вход с удвоенным (т. е. сдвинутым на один разряд в сторону старших разрядов) кодом, поступающим на второй вход.

В соответствии с этим структурная схема умножителя приобретает вид, представленный на рис. 4.2. Далее рассматривается описание устройства на языке VHDL.

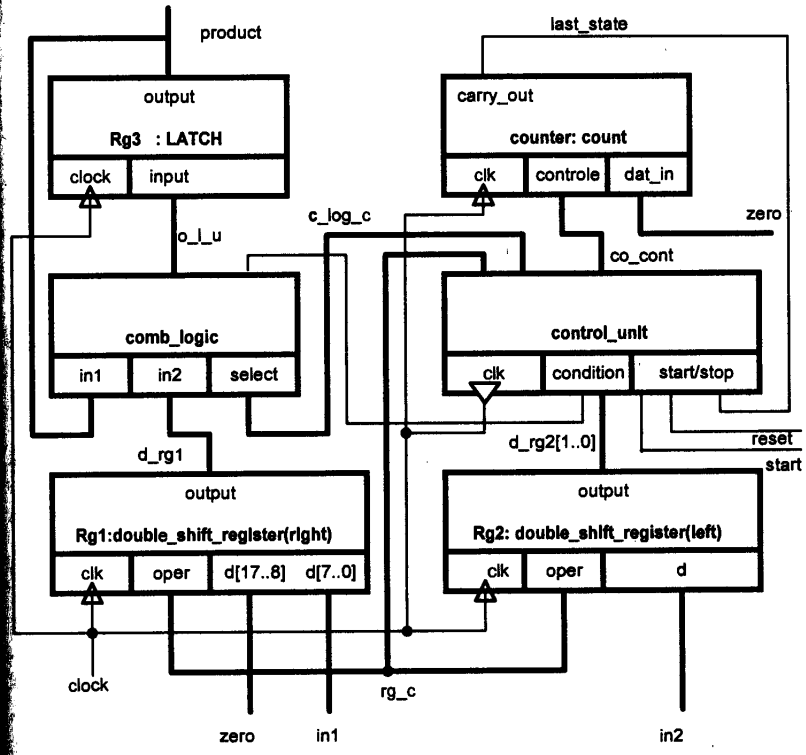


Рис. 4.2. Структурная схема умножителя

В VHDL-программе (см. листинг 4.1) линиям связи компонентов соответствуют сигналы. Идентификаторы и функциональное назначение этих сигналов определены в табл. 4.1.

Таблица 4.1. Идентификаторы и функциональное назначение сигналов VHDL-программы из листинга 4.1

Обозначение на схеме	Идентификатор в VHDL-программе	Функциональное назначение
q_rg1	q_rg1	Выходной код регистра множимого
q_rg2	q_rg2	Выходной код регистра множителя
last_state	last_state	Сигнал, сигнализирующий о том, что счетчик содержит код заданного числа циклов и означающий конец операции умножения
co_c	count_controle	Код, задающий операцию счетчика циклов
rg_c	register_controle	Код, задающий операцию сдвигающих регистров
c_l_c	comb_logic_control	Код, задающий операцию, реализуемую комбинационной схемой
product	product	Регистр накопления частичных сумм (произведение)
o_l_u	out_logical_unit	Выход комбинационной схемы

Разработка модели поведения цифрового автомата

Прежде всего, определяется набор состояний управляющего автомата и на схеме алгоритма помечаются эти состояния. Общее правило разметки состояний состоит в следующем [15]: *входу каждой вершины схемы алгоритма, следующей за операторной, сопоставляется состояние автомата.*

Однако в нашем примере общее число состояний может быть уменьшено, а соответственно, повышено быстродействие устройства. Операции, определенные на всех путях многовариантного выбора, совместимы с операциями, определенными в следующим за ними блоке, и могут быть выполнены одновременно с последними без каких-либо дополнительных аппаратурных затрат. Таким образом, управляющий автомат должен иметь два состояния, которые на схеме алгоритма обозначены $S0$ и $S1$ и помечены крестиками. В VHDL-программе состояние может быть представлено переменной или сигналом перечислимого типа. В программе (листинг 4.1) принято обозначение допустимых значений, совпадающее с обозначениями на схеме алгоритма, поэтому примем допустимые значения данных перечислимого типа как $S0$ и $S1$.

Управляющие автоматы могут быть представлены в различной форме. В данном примере, ориентированном на описание устройства в VHDL, применен

оператор выбора CASE, при этом в качестве ключа выбора варианта использована переменная, представляющая состояние автомата в текущий момент времени. Внутри каждого варианта определяется состояние перехода и значения выходных сигналов в соответствии с состоянием входов управляющего автомата. Следует обратить внимание на необходимость установки автомата в исходное состояние перед началом работы: выражение `IF reset='1' THEN state<=s0;`. Программа предполагает, что использована синхронизация управляющего автомата спадом тактирующего сигнала: выражение `IF clock='0' AND NOT clock'stable THEN`. Такая синхронизация для используемых операционных узлов обеспечит стабильность входных управляющих сигналов в моменты тактирования.

Разработка текстового описания устройства

Здесь мы кратко прокомментируем основные разделы VHDL-программы устройства умножения, рассматриваемого в качестве примера. Использовано смешанное структурно-поведенческое представление проекта.

В разделе ENTITY заданы внешние управляющие сигналы — тактовый `clock`, начальный сброс `reset`, запуск одиночной операции умножения `start`, а также входные и выходные информационные сигналы `in1`, `in2` и `output`.

Подраздел объявления компонентов архитектурного тела представляет используемые типовые операционные узлы — сдвигающий регистр со сдвигом на два разряда `double_shift_register`, счетчик `count` и регистр-защелку `latch`. Программы, описывающие поведение этих компонентов, должны быть заранее скомпилированы в рабочую библиотеку проекта. Не вдаваясь в детальное представление этих компонентов, будем считать, что они изменяют состояние синхронно с передним фронтом сигнала `clk`, а выполняемое при этом действие определяется для регистра и счетчика состоянием управляющих входов в соответствии с комментариями к декларации прототипов. В разделе описания сигналов предъявлены соединения между компонентами в соответствии с табл. 4.1.

Раздел операторов архитектурного тела `mixed OF ex_mult` содержит операторы параллельного типа, в том числе операторы объявления вхождений компонентов, оператор процесса, помеченный меткой `controle_unit`, представляющий поведенческое описание блока управления, и оператор присваивания по выбору, с помощью которого удалось описать работу всей комбинационной части операционного устройства. Каждое вхождение компонента представлено своим GENERIC MAP — объявлением параметров настройки конкретного образца, и PORT MAP — перечислением точек присоединения внешних выводов узла к линиям внутренних и сигналов к портам устройства.

```
LIBRARY IEEE;
USE IEEE.std_logic_1164.ALL;
USE IEEE.std_logic_unsigned.ALL;

  ENTITY ex_mult IS -- Entity проекта
     PORT( reset,clock,start:IN std_logic;
           IN1,IN2:IN std_logic_vector (7 DOWNTO 0);
            last_state:INOUT std_logic;
            output:OUT std_logic_vector (15 DOWNTO 0)
     );
      END ex_mult;

ARCHITECTURE mixed OF ex_mult IS
COMPONENT count  -- синхронный загружаемый счетчик
       GENERIC (number_of_states:integer;     -- число состояний
                number_of_bits:integer);      -- число разрядов);
       PORT( clk,                             -- синхронизация,
             reset,carry_in:std_logic;        -- сброс, перенос
            oper:IN std_logic_vector (1 DOWNTO 0);
                --   сигналы управления      -- "00" пусто
                     --"01" - +1
                     --"10"  загрузка
            dat_in: IN std_logic_vector (number_of_bits-1 DOWNTO 0);
            dat_out:OUT  std_logic_vector (number_of_bits-1 DOWNTO 0);
            carry_out:OUT std_logic );
END COMPONENT;
COMPONENT  latch -- регистр-защелка
       GENERIC ( number_OF_bits:INteger); -- число разрядов
       PORT (clock: IN std_logic; -- синхронизация
             input: IN std_logic_vector(17 DOWNTO 0);
              output: OUT std_logic_vector(17 DOWNTO 0));
END COMPONENT;

COMPONENT  double_shift_register -- регистр со сдвигом на два разряда
                                 -- и вдвиганием нуля в освободившиеся
  GENERIC( num_bits: integer; dir:string);
--  dir  := "left","right","bidir" ( Влево-вправо-двунаправленный)
  PORT(clk:IN std_logic;
     control:IN std_logic_vector (1 DOWNTO 0);  --  00 - пусто
                                                --  10 - сдвиг вправо
                                                --  01 - сдвиг влево
                                                --  11 - загрузка
       input:IN std_logic_vector  (num_bits-1 DOWNTO 0);
       output:OUT std_logic_vector  (num_bits-1 DOWNTO 0)
       );
END COMPONENT;
```



```
  TYPE states IS (s0,s1);
   SIGNAL zero: std_logic_vector(17 DOWNTO 0):="000000000000000000" ;
   SIGNAL one:std_logic:='1';
   SIGNAL  q_rg2: std_logic_vector (7 DOWNTO 0);
   SIGNAL regISter_control,count_control: std_logic_vector(1 DOWNTO 0);
   SIGNAL comb_logic_control: std_logic_vector(2 DOWNTO 0);
    SIGNAL q_rg1,OUT_logical_unit,product:std_logic_vector(17 DOWNTO 0);
    SIGNAL count_OUT: std_logic_vector (2 DOWNTO 0);
    SIGNAL state: states;
BEGIN
output<=product(15 DOWNTO 0);
-- функции блока комбинационной логики:
--  "000" - выход равен нулю
-- "100"-  повторение product
-- "101" - прибавление
-- "110" - вычитание
-- "111"-  прибавить удвоенное значение входа
WITH comb_logic_control SERECT
           out_logical_unit<= zero  WHEN "000",
                              product+q_rg1 WHEN "101",
                             product- q_rg1 WHEN "110",
                              product+q_rg1*"10" WHEN "111",
                               product WHEN others;
rg1:double_SHIFT_REGISTER
  GENERIC MAP(18, "left")
  PORT MAP(clk=>clock,oper=>register_control, input(17 DOWNTO 8)
       =>zero(9 DOWNTO 0),input(7 DOWNTO 0)=> in1,output=>q_rg1);

 rg2:double_shift_register
 GENERIC MAP( 8, "right")
  PORT MAP(clock,register_control,in2,q_rg2);

counter: count
       GENERIC map(number_of_states=>5,number_of_bits=>3)
       PORT map(  clk=>clock, reset=>reset,carry_in=>one,
           oper=> count_control,
            dat_in=>zero(2 DOWNTO 0),
            dat_out=>count_OUT,
            carry_out=>last_state);
rg3: latch
       GENERIC MAP( number_of_bits=>18)
     PORT MAP(clock=>clock, input=>out_logical_unit, output=> product);

controle_unit:  -- управляющий автомат
PROCESS(reset,clock)
     VARIABLE c:std_logic;
     VARIABLE condition:std_logic_vector (2 DOWNTO 0);
```

```
BEGIN
  condition:=q_rg2(1) & q_rg2(0) & c;
  IF reset='1' THEN state<=s0;
    ELSIF clock='0' and not clock'stable THEN
       CASE  state IS
           WHEN s0=> IF start='1' THEN register_control<="11";
                              state<=s1;
                              c:='0'; count_control<="11";
                              comb_logic_control<="000";
                     ELSE  state<=s0; register_control<="00";
                              c:='0'; count_control<="00";
                     END IF;
          WHEN s1=> IF last_state='1' THEN
                        state<=s0;
                      IF c='1' THEN comb_logic_control<="101";
                                    ELSE comb_logic_control<="100";
                       END IF; --carry
                   ELSE
                   CASE condition IS
                     WHEN "010"=> comb_logic_control<= "101";
                     WHEN "001"=>  comb_logic_control<=    "101";
                     WHEN "100" => comb_logic_control<=  "111";
                     WHEN "011"=>  comb_logic_control<=   "111";
                     WHEN "110"  =>  comb_logic_control<=   "110";
                     WHEN "101"=>comb_logic_control<= "110";
                     WHEN others=>comb_logic_control<="100";
                   END CASE;
                    c:=q_rg2(1) and (q_rg2(0) or c);
                 END IF;-- count_overflow
                   count_control<="01";
                   regISter_control<="10";
          END CASE; -- state
    END IF;---reset=1 or clock=0
  END PROCESS;-- END controle_unit
END mixed;
```

4.1.2. Операционные устройства конвейерного типа

Как отмечалось, повышение производительности операционных устройств для реализации многошаговых процедур, в частности таких, в которых на каждом шаге использованы результаты предыдущих шагов исполнения алгоритма, может быть достигнуто при конвейерной реализации блока обработки.

В качестве примера рассмотрим реализацию конвейерного устройства для выполнения операций сложения и вычитания над числами, представленными в формате с плавающей запятой. Примем следующее, достаточно распространенное представление:

$$x = 2^{p-128} \times M,$$

где x — представляемое число; p — код порядка, т. е. положительное целое число в восьмиразрядном двоичном представлении; M — мантисса, т. е. число, модуль которого принадлежит интервалу $[0{,}5;\ 1-2^{-23}]$, представленное в прямом двадцатичетырехразрядном коде, причем старший разряд знаковый, а запятая фиксируется между знаковым и старшим значащим разрядами.

Предполагаем, что входные и выходные данные нормализованы, т. е. приведены к форме, при которой старший значащий разряд мантиссы установлен в единицу.

Операции сложения и вычитания чисел, представленных в формате с плавающей запятой, предусматривают выполнение следующих действий:

1. Выделение операнда, у которого порядок больше, и вычисление разности порядков. Это необходимо для последующего выравнивания порядков при обработке мантисс.
2. Модификация кодов мантисс, т. е. перевод в дополнительный код мантисс чисел, которые будут суммироваться с отрицательным знаком (отрицательных слагаемых, отрицательных вычитаемых и положительных вычитателей).
3. Выравнивание порядков. Выполняется сдвиг мантиссы меньшего числа влево на число разрядов, равное разности порядков.
4. Сложение кодов мантисс. Для возможности обнаружения переполнения предварительно знаковые разряды операндов дублируются. Признак переполнения это несовпадение значений основного и дублирующего знаковых разрядов после сложения.
5. Коррекция переполнения. Если обнаружено переполнение, результат предыдущего этапа сдвигается на один разряд в сторону младших, а порядок увеличивается на единицу.
6. Модификация результата. Если результат отрицательный, результат преобразуется в прямой код.
7. Нормализация результата. Мантисса сдвигается так, что старшая единица в значащих разрядах результата размещается в старшем разряде. Знаковый разряд не модифицируется. Одновременно порядок результата уменьшается на столько единиц, на сколько разрядов сдвинута мантисса.

Можно обратить внимание, что большинство этапов должны выполняться последовательно. Тем не менее, первые два этапа функционально независи-

мы и требуют существенно различных средств, поэтому могут совмещаться во времени. Кроме того, преобразования по пунктам 4 и 5 достаточно просты и реализуются последовательно включенными комбинационными схемами за время, сравнимое с временем исполнения других этапов. Поэтому можно принять структуру устройства сложения чисел с плавающей точкой, представленную на рис. 4.3.

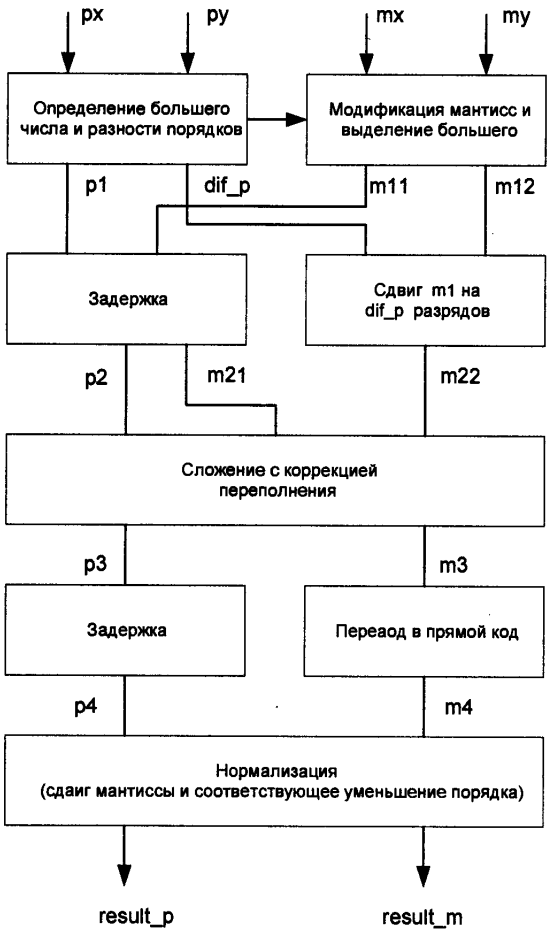


Рис. 4.3. Структура конвейерного сумматора чисел в формате с плавающей запятой

На рис. 4.3 и программе (листинг 4.2), отображающей поведение этого устройства, обозначено:

- `p1` и `m11` — порядок и модифицированная мантисса большего по модулю числа, а `p2` и `m21` — их задержанные значения;
- `m12` — модифицированная мантисса меньшего числа, а `m22` — ее значение после выравнивания порядков;
- `dif_p` — модуль разности порядков;
- `m3`, `m4`, `p3`, `p4` — результаты промежуточных этапов вычисления мантиссы и порядка.

Программа (листинг 4.2) представляет поведенческую интерпретацию поведения ступеней конвейера на языке VHDL. Для лучшей структуризации проекта описания ступеней выделены в отдельные программные единицы — процессы.

Для того чтобы обеспечить конвейерную обработку, выходы любой ступени представлены регистрами, причем все регистры синхронизируются общим тактовым сигналом. При описании в VHDL это находит свое отражение в том, что операторы присваивания выходным сигналам ступеней вложены в операторы PROCESS, инициируемые общим тактовым сигналом. Структура программы в языке Verilog HDL аналогична: следует записать подобную совокупность блоков, снабдив каждый из них префиксом событийного управления от фронта тактирующего сигнала.

Листинг 4.2

```
LISRARY IEEE;
USE ieee.std_logic_1164.ALL;
USE ieee.std_logic_arith.ALL;
USE ieee.std_logic_unsigned.ALL;

ENTITY pipe_add IS
   PORT (clk    : IN std_logic;
         add_subb    : IN std_logic; -- 0 - сложение
         mx,my : IN std_logic_vector(23 DOWNTO 0);
         px,py : IN std_logic_vector(7 DOWNTO 0)
         result_m : OUT std_logic_vector(23 DOWNTO 0);
         result_p:OUT std_logic_vector(7 DOWNTO 0);
);
END pipe_add;

ARCHITECTURE behave OF pipe_add IS
  SIGNAL m11,m12,m21,m22,m3,m4:std_logic_vector(23 DOWNTO 0);
  SIGNAL maxp,dif_p,p2,p3,p4: std_logic_vector(7 DOWNTO 0);
```

```
BEGIN
PROCESS(clk)-- приведение знаков; выделение большего
VARIABLE mod_mx,mod_my:std_logic_vector(23 DOWNTO 0);
 BEGIN
  IF mx(23)='0'THEN mod_mx:=mx;
   ELSE mod_mx:= mx(23)& (not mx(22 DOWNTO 0) +1);
  END IF;
  IF (my(23) xor ADD_subb) ='0' THEN mod_my:=my;
  ELSE mod_my:= my(23)& (not my(22 DOWNTO 0) +1);
  END IF;
IF px>= py  THEN  maxp<=px; dif_p<=px-py;
     m11<=mod_mx; m12<=mod_my;
  ELSE maxp<=py;  dif_p<=py-px;
     m12<=mod_mx; m11<=mod_my;
END IF;
END PROCESS;

PROCESS(clk) -- Выравнивание порядков
BEGIN
 p2<=maxp;    -- трансляция большего порядка
   m21<=m11; -- мантисса большего числа передается без изменений
 IF dif_p<24 THEN
   m22<=to_stdlogicvector(to_bitvector(m12) sla conv_integer(dif_p));
   ELSE m22<="000000000000000000000000";
END IF;
END PROCESS;
process (clk) -- сложение, коррекция при переполнении
variable sum,pr: std_logic_vector(24 downto 0);
variable switch:std_logic_vector (1 downto 0);
  begin
    sum:=m12(23)& m12 + m22(23)& m22;
    if not(sum(23) = sum(23)) then
       sum:=to_stdlogicvector(to_bitvector(sum) sla 1);
           p3<=p2+1;
    else  p3<=p2;
  end if;
  m3<=sum(23 downto 0);
end process;
process (clk) -- модификация результата
   Begin
   p4<=p3; --синхронизация данных
   if m3(23)='0' then  m4<= m3(23 downto 0);
     else  m4<= '1' & (not m3(22 downto 0))+1;
END IF;
 END PROCESS;
```



```
PROCESS (clk) -- нормализация
 VARIABLE i:integer;
   BEGIN
     FOR i IN 0 TO 22 LOOP -- определение позиции старшей единицы
        EXIT WHEN m3(22-i)='1';
      END LOOP;
   result_m <= to_stdlogicvector(to_bitvector(m3) sla i);
   result_p <=  p3+i;
END PROCESS;
END behave;
```

В зависимости от требований проекта можно объединять логику смежных ступеней (что приведет к уменьшению числа тактов преобразования, но может потребовать увеличения длительности такта), или наоборот, разбить основные этапы на более мелкие подэтапы, имея в виду включение регистровых схем между узлами, реализующими подэтапы. В общем случае при проектировании может потребоваться вмешательство разработчика для детальной проработки фрагментов и осуществление дополнительной их структурной декомпозиции для получения улучшенных характеристик реализации.

Список допустимых операций представленного арифметического устройства достаточно легко может быть расширен. Например, для реализации умножения на втором этапе следует предусмотреть умножение мантисс (выравнивание порядков при сложении может быть выполнено как умножение), а на третьем этапе выполнять сложение порядков. Блок нормализации никак не модифицируется. Естественно, требуется управление коммутацией данных внутри блоков, реализующих фрагменты алгоритма. Такие коммутаторы, распределенные по пути распространения данных, управляются кодом выполняемой операции, который "движется" по параллельному пути, составленному из элементов задержки, синхронно с перемещением данных, которые обрабатываются в этой операции.

Приведенные в данном разделе примеры относились к реализациям арифметических преобразований. Такой выбор был сделан, потому что обработка численных данных, по мнению авторов, понятна наибольшему числу возможных читателей книги и наиболее наглядна. Но такие подходы и подобные структурные реализации могут быть применены также и для создания систем обработки иной информации — логического анализа, поиска информации в сложных структурах данных, обработки текстов, изображений и т. д.

4.2. Реализация модулей памяти в ПЛИС

Все современные ПЛИС высокой сложности включают в свой состав блоки статической памяти (ЗУ) с произвольным доступом достаточно большого объема *(см. гл. 1)*. Внутренняя память ПЛИС организована по модульному

принципу и может реконфигурироваться в самые разнообразные структуры. Поэтому проблема реализации подсистем памяти в проектах состоит не столько в проектировании отдельных блоков памяти и даже не в создании средств доступа (буферизация адреса и данных, управление, сигнализация о состоянии и т. д.), сколько в выборе конфигурации из числа вариантов, предлагаемых в библиотеках типовых решений САПР, и их соответствующего представления в проекте.

Известно, что с точки зрения способа доступа к информации устройства памяти разделяются на:

- память с адресным доступом (оперативная RAM и постоянная ROM), существуют также однопортовая и многопортовая адресная память;
- память с последовательным доступом;
- память с ассоциативным доступом.

В книге [27] описаны принципы работы и типовые варианты структурной реализации этих видов памяти.

Данный раздел представляет обзор вариантов конфигурации и характеристику специфических особенностей блоков памяти на примере состава библиотеки параметризованных модулей (Library of Parameterized Modules, LPM) фирмы Altera.

4.2.1. Память с адресным доступом

Варианты однопортовой памяти с адресным доступом представлены в библиотеке САПР MAX+PLUS II несколькими параметризованными модулями:

- LPM_RAM_DQ — RAM с раздельными линиями входа и выхода данных;
- LPM_RAM_IO — RAM с общей линией входа и выхода данных;
- LPM_ROM — постоянная память.

Листинг 4.3 представляет VHDL-прототип модуля LPM_RAM_DQ.

Листинг 4.3

```
COMPONENT lpm_ram_dq
   GENERIC (LPM_WIDTH: POSITIVE;
      LPM_TYPE: STRING := "L_RAM_DQ";
      LPM_WIDTHAD: POSITIVE;
      LPM_NUMWORDS: STRING := Positive;
      LPM_FILE: STRING := "UNUSED";
      LPM_INDATA: STRING := "REGISTERED";
      LPM_ADDRESS_CONTROL: STRING := "REGISTERED";
      LPM_OUTDATA: STRING := "REGISTERED";
      LPM_HINT: STRING := "UNUSED");
```



```
   PORT (data: IN STD_LOGIC_VECTOR(LPM_WIDTH-1 DOWNTO 0);
      address: IN STD_LOGIC_VECTOR(LPM_WIDTHAD-1 DOWNTO 0);
      wr: IN STD_LOGIC := '1';
      inclock: IN STD_LOGIC := '1';
      outclock: IN STD_LOGIC := '1';
      q: OUT STD_LOGIC_VECTOR(LPM_WIDTth-1 DOWNTO 0));
END COMPONENT;
```

Рассмотрим основные параметры конфигурации модуля и вытекающие из этого особенности функционирования. Параметры LPM_WIDTH и LPM_NUMWORDS достаточно очевидны: они задают организацию памяти — разрядность и общее число хранимых слов. Разрядность кода адреса задается параметром LPM_WIDTHAD. Рекомендуется (но не обязательно), чтобы это значение было равно Log2 (LPM_NUMWORDS). Если LPM_WIDTHAD окажется меньше этой величины, то часть памяти становится попросту недоступна, а если больше, то на кодовых комбинациях, численный эквивалент которых превышает LPM_NUMWORDS, из памяти считывается неопределенное значение 'x'.

Параметр LPM_TYPE — это специфический параметр, обязательный для модулей библиотеки LPM, параметр просто повторяет имя ENTITY прототипа.

Параметр LPM_FILE определяет имя файла, который может содержать данные, записываемые в память при инициализации. Параметр LPM_HINT устанавливает дополнительные опции для синтеза. Если в проекте установлено значение USED, то при синтезе устройства памяти учитываются глобальные опции логического синтеза, определенные для остальной части проекта. Если установлено значение UNUSED, то эти опции не учитываются.

Параметры LPM_INDATA, LPM_ADDRESS_CONTROL, LPM_OUTDATA относятся к типу STRING и могут принимать значения REGISTERED и UNREGISTERED. Если принято значение REGISTERED, то соответствующие порты (data — вход данных, q — выход данных и address — вход адреса) внутри модуля сопровождаются буферными регистрами, причем адресный буфер и буфер входных данных стробируются сигналом inclock, а выходной буфер данных — сигналом outclock.

Здесь важно отметить, что если вход адреса стробирован, то работа устройства памяти осуществляется по конвейерному принципу. По фронту сигнала inclock адрес фиксируется в буферном регистре и подвергается внутренней дешифрации, но в матрице памяти в этот момент выбрана ячейка, адрес которой был записан в буфер в предыдущем цикле обращения. Именно в эту ячейку в режиме записи заносятся данные, которые в этот момент находятся в буфере входных данных, а в режиме чтения данные из этой ячейки поступают на выход. Указанные особенности иллюстрируются временными диаграммами, представленными на рис. 4.4.

Рис. 4.4, *а* показывает временную диаграмму процесса записи для случая стробированного адресного входа и входа данных. Строка диаграммы ram

отображает действия, выполняемые в каждом временном интервале в матрице памяти. Здесь стрелка отображает операцию занесения блока данных Di по адресу Ai в соответствии с обозначением предыдущих строк. После каждого тактового импульса запись данных, зафиксированных в буфере по фронту предыдущего тактового импульса, выполняется по адресу, который

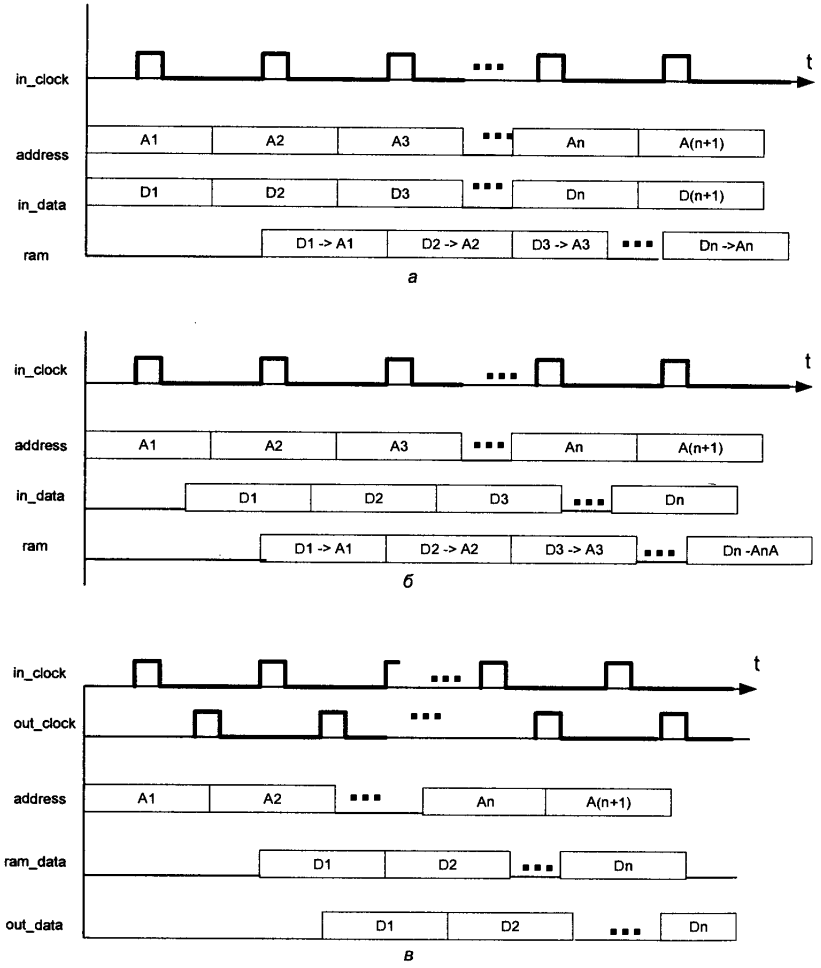


Рис. 4.4. Временные диаграммы работы конвейерных запоминающих устройств

также зафиксирован в буфере адреса в предыдущем такте работы. Рис. 4.4, *б* иллюстрирует случай нестробированного входа данных. Обратим внимание, что данные, которые в момент синхроимпульса присутствуют на входе, записываются по адресу, зафиксированному в предыдущем такте.

На рис. 4.4, *в* представлена диаграмма процесса чтения для режима стробируемых входа адреса и выхода данных. После каждого импульса in_clock на выходе матрицы памяти ram_data возникают данные, записанные в матрице по адресу, зафиксированному в буфере в предыдущем такте (A1 соответствует D1 и т. д.). Если выход не стробирован, эти данные непосредственно транслируются на выход. Для случая стробированного выхода данные фиксируются в выходном буфере по каждому импульсу out_clock.

Пример включения модуля LPM_RAM_DQ представлен в листинге 4.4. Библиотечный модуль сконфигурирован на реализацию блока памяти объемом 10 Кбайт с фиксацией по фронту синхроимпульса входов адреса и данных и нестробируемым выходом. Этот модуль включен в проект block_mema, представляющий устройство, способное выполнять блочные пересылки между модулем памяти и другими устройствами в синхронном режиме.

Замечание

Декларация прототипа в данном случае не требуется, т. к. содержится в пакете lpm_components, по определению скомпилированному в системную библиотеку LPM и доступную всем проектам, выполняемым в среде MAX+PLUS II.

По сигналу start начальный адрес передаваемого блока in_adr, длина блока block_length и признак чтение/запись wr фиксируются во внутренних регистрах. Каждый следующий такт работы инициирует чтение или запись данных по адресу, находящемуся в счетчике адреса с последующим инкрементом адреса. После выполнения запланированного числа обменов устройство переходит в состояние ожидания очередного запроса.

Листинг 4.4

```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
USE ieee.std_logic_arith.ALL;
LIBRARY lpm;
USE lpm.lpm_components.ALL;

ENTITY block_mema IS
   PORT(data: IN std_logic_vector (7 DOWNTO 0);
        in_adr,block_length    : IN integer RANGE 0 TO 1023;
        wr, clock,reset,start: IN std_logic;
        busy:out std_logic;
        q: OUT std_logic_vector (7 DOWNTO 0));
END block_mema;
```

```
ARCHITECTURE example OF block_mema IS
    SIGNAL address:integer RANGE 0 TO 1023;
    SIGNAL paddress: std_logic_vector (9 DOWNTO 0);
    TYPE state IS (idle, working);
    SIGNAL inclock,we: std_logic;
    SIGNAL s:state;
BEGIN
    paddress<=conv_std_logic_vector(address,10);
    inclock<= clock WHEN s=working ELSE
          '0';
    busy<= '1' WHEN s=working ELSE
          '0';
PROCESS (clock,reset)
  VARIABLE fin_address:integer;
BEGIN
  IF reset='1' THEN s<=idle;
  ELSIF  clock='1' AND clock'event THEN
      CASE s IS
      WHEN idle=>
           we<=wr;
           IF start='1' THEN s<=working;
              address<=in_adr;
              fin_address:=address+ block_length;
           ELSE s<=idle;
           END IF;
      WHEN working => address<=address+1;
           IF address=fin_address THEN s<=idle;
           END IF;
      END CASE;
  End IF;
End PROCESS;
  ram_inst: lpm_ram_dq
     GENERIC MAP (lpm_widthad => 10,lpm_outdata=>"unregistered",
      lpm_width => 8)
     PORT MAP (data => data, address => paddress, we => we,
                inclock => inclock, q => q);
END example;
```

Набор параметров настройки для модуля памяти с мультиплексированными линиями входа/выхода `lpm_ram_io` и модуля постоянной памяти `lpm_rom` ничем не отличаются от `lpm_ram_dq`. В число управляющих сигналов этих модулей введен сигнал разрешения, в отсутствии которого линии входа/выхода находятся в высокоимпедансном состоянии. Очевидно, что `lpm_rom` не имеет линий входа данных и сигнала разрешения записи.

Модули двухпортовой памяти достаточно давно введены в состав библиотек фирмы Altera. Классическая двухпортовая память представлена в библиотеках САПР MAX+PLUS II компонентом `csdpram`, прототип которого приведен в листинге 4.5.

Листинг 4.5

```
COMPONENT csdpram
   GENERIC (LPM_WIDTH: POSITIVE;
      LPM_WIDTHAD: POSITIVE;
      LPM_NUMWORDS: STRING ;= "UNUSED";
      FILE: STRING := "UNUSED");
   PORT (dataa, datab: IN STD_LOGIC_VECTOR(LPM_WIDTH-1 DOWNTO 0);
      addressa, addressb: IN STD_LOGIC_VECTOR(LPM_WIDTHAD-1 DOWNTO 0);
      clock, clockx2, wea, web: IN STD_LOGIC;
      qa, qb: OUT STD_LOGIC_VECTOR(LPM_WIDTH-1 DOWNTO 0);
      busy: OUT STD_LOGIC);
END COMPONENT;
```

Смысл параметров настройки совпадает со смыслом параметров настройки других модулей памяти.

Каждый порт представлен входом адреса (`addressa` для канала `A` и `addressb` для канала `B`), входом данных (`dataa` и `datab`, соответственно) и выходом данных (`qa` и `qb`). Модуль обеспечивает полностью симметричный режим работы. Возможно обращение к обоим портам в одном цикле как по чтению, так и по записи с произвольными адресами запроса. Единственное, и вполне естественное, ограничение — это недопустимость одновременной записи по одному адресу. В такой ситуации приоритет отдается запросу по каналу `A`, и одновременно выдается информационный сигнал `busy`, сообщающий устройству, подключенному к каналу `B`, об отказе в обслуживании. В подобном случае это устройство должно повторить попытку доступа в следующем цикле работы.

В последних типах микросхем (семейства ACTEX, Mercury) и более поздних версиях САПР предусматриваются широкие возможности конфигурирования многопортовой памяти [48]. Варианты конфигурирования многопортовой памяти для семейства Mercury приведены на рис. 4.5.

Вариант конфигурации (рис. 4.5, *а*) представляет симметричную двухпортовую память, подобную представленному в листинге 4.5 прототипу. Вариант (рис. 4.5, *б*) представляет упрощенный, несимметричный вариант, в котором порт `A` работает только на запись, а порт `B` только на чтение. Вариант конфигурации (рис. 4.5, *в*) соответствует четырехпортовой памяти с двумя портами ввода и двумя портами вывода. Возможны, кроме того, смешанные варианты.

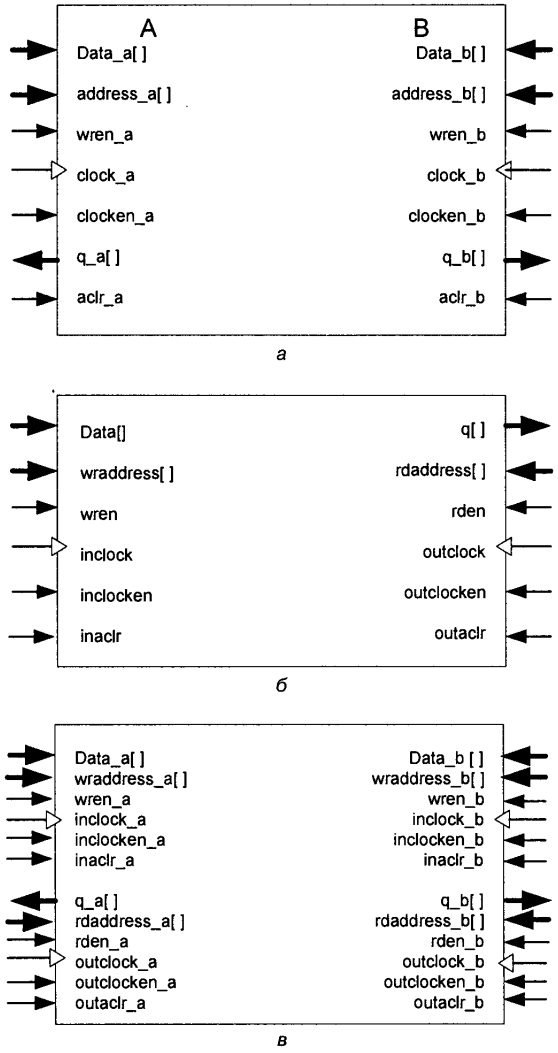


Рис. 4.5. Варианты конфигурации модуля двухпортовой памяти

4.2.2. Память с последовательным доступом

В памяти с последовательным доступом записываемые данные образуют некоторую очередь. Считываются данные из очереди одно за другим либо в том же порядке, в котором осуществлялась запись, либо в обратном порядке. Моделью последовательного ЗУ является цепочка последовательно соединенных элементов хранения, в которой данные передаются между соседними элементами.

С точки зрения порядка доступа можно выделить несколько вариантов:

- устройства, реализующие дисциплину "первый пришел — первым вышел" (FIFO);
- файловые ЗУ;
- циклические ЗУ;
- устройства, реализующие дисциплину "последний пришел — первым вышел" (LIFO).

Разница между FIFO-памятью и файловым ЗУ состоит в том, что в первом запись в "пустой" буфер сразу становится доступной для чтения. В файловых ЗУ данные поступают как бы в начало цепочки элементов хранения и появляются на выходе после числа обращений, равного количеству элементов в этой цепочке. Так как операции считывания и записи могут быть взаимно независимы, фактическое расположение данных в файловых ЗУ на момент считывания не связано с каким-нибудь внешним признаком. Поэтому записываемые данные объединяют в блоки, обрамляемые специальными символами конца и начала (файлы, отсюда и название). Процесс приема данных из файлового ЗУ начинается после того, как в последовательности выходных данных приемник обнаружит символ начала блока. В циклических ЗУ единицы записи доступны одно за другим с постоянным периодом, определяемым емкостью памяти. Считывание по дисциплине LIFO характерно для стековых запоминающих устройств.

Время доступа к требуемой единице хранения информации в последовательных ЗУ представляет собой случайную величину. В наихудшем случае для доступа к нужной записи может потребоваться "просмотр" всего объема хранимых данных.

В организации ЗУ с последовательным доступом известны два существенно различных подхода:

- продвижение информации в цепочке элементов (подобно регистрам сдвига);
- хранение информации в выбранных ячейках адресного ЗУ с управлением адресом доступа по определенному алгоритму.

При реализации блоков последовательной памяти в ПЛИС применяют оба подхода. Устройства на основе адресных ЗУ, как правило, более экономич-

ны. Но причиной использования регистровой реализации может быть недостаточный объем встроенной в ПЛИС памяти или даже ее отсутствие (например, при реализации на сравнительно дешевых микросхемах, таких как серии MAX7000 и MAX3000 фирмы Altera, XC9500 и CoolRanner фирмы Xilinx). Кроме того, следует заметить, что регистровые реализации при прочих равных условиях обеспечивают работу на большей тактовой частоте в сравнении со схемами на основе адресной памяти.

Рассмотрим эти подходы на примере памяти типа FIFO. Подобным образом строятся блоки памяти последовательного типа с иными дисциплинами доступа.

Описание на языке VHDL FIFO-буфера, построенного на использовании сдвига информации в последовательной цепочке регистров, представлено в листинге 4.6. Структурными элементами этого устройства является блок регистровой памяти (БРП) `men`, содержащий `length` регистров по `width` разрядов каждый, и `length`-разрядный регистр-указатель конца очереди `rg`, в котором всегда присутствует только одна единица, указывающая позицию в БРП, в которую будет заноситься очередное данное.

При записи новая информация записывается в БРП на место, указанное единицей в регистре-указателе, после чего содержимое `rg` сдвигается в сторону младших разрядов. Содержимое остальных регистров не изменяется. При чтении на выход выдается информация из последнего регистра БРП, а содержимое остальных переписывается в соседние регистры с большим порядковым номером. Информация в регистре-указателе сдвигается в сторону младших разрядов.

В случае одновременного запроса на запись и чтение все регистры БРП, кроме указанного единицей в регистре-указателе, принимают содержимое их предыдущих соседей. А регистр, который отмечен единицей в регистре-указателе, принимает входные данные. Содержимое регистра-указателя не изменяется.

Наличие единицы в старшем разряде регистра-указателя является сигналом "буфер пустой" (`empty`), а в младшем — сигналом "буфер полон" (`full`). Эти сигналы должны анализироваться источником и приемником данных, которые в соответствующих ситуациях приостанавливают запросы на доступ. Чтобы избежать потерю единицы в регистре-указателе, при сдвиге в сторону младших разрядов самый младший разряд сохраняет единицу, если она в нем была. Аналогично, при сдвиге в сторону старших разрядов самый старший разряд сохраняет единицу, если она в нем была.

Листинг 4.6

```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
use ieee.std_logic_arith.all;
```



```
entity fifo IS
generic (length,width:integer:=8);
port(start,wr,rd,clk:in STD_LOGIC;
     FULL, EMPTY:out STD_LOGIC;
     datin:in std_logic_vector(width-1 downto 0);
     datout:out std_logic_vector(width-1 downto 0));
END fifo;

ARCHITECTURE registered OF fifo IS
 TYPE memory is ARRAY (length-1 downto 0) of std_logic_vector(width-1
                                                              downto 0);
   signal men:memory;
   signal rg: std_logic_vector(length-1 downto 0);
begin
full<=rg(0);
empty<=rg(length-1);
datout<=men(length-1);
men(0)<= datin when wr='1' and rg(0)='1' else men(0);
Process(rd,wr,start,clk)
 variable v: std_logic_vector(1 downto 0);
 begin
   v:=rd&wr;
  if (start='1') then -- начальные установки
      rg<=conv_std_logic_vector(0,length);
      rg(length-1)<='1';
   elsif clk='1' and clk'event then
    for i in length-1 downto 1 loop -- сдвиг данных в регистровом массиве
      case  v is
           when "10"=> men(i)<=men(i-1);
           when "01"=> if rg(i)='1' then men(i)<=datin;
                       end if;
           when"11"=> if rg(i)='1' then men(i)<=datin;
                       else men(i)<=men(i-1);
                    end if;
           when others=>null;
       end case;
end loop;
   case v is -- логика регистра управления записью
    when "10" =>
                 rg(length-2 downto 1)<=rg(length-3 downto 0);
                 rg(length-1)<= rg(length-1) or rg(length-2);
                 rg(0)<='0';
    when "01"=> rg(length-2 downto 1) <= rg(length-1 downto 2);
                rg (0)<= rg(0) or rg(1);
                rg(length-1)<='0';
```

```
    when others=> null;
   end case;
  end if;--start
end process;
end registered;
```

В качестве примера реализации FIFO-буфера с размещением информации в блоке адресной памяти рассмотрим мегафункцию csfifo (Cycle-Shared FIFO Buffer) из библиотеки параметризованных модулей САПР MAX+PLUS II. VHDL-прототип этой функции представлен в листинге 4.7, а внутренняя структура — на рис. 4.6. Нетрудно убедиться, что декларация модуля csfifo практически совпадает с декларацией ENTITY модуля fifo, представленного в листинге 4.6. Отчасти изменены имена портов и введен дополнительный синхросигнал clockx2, имеющий удвоенную частоту в сравнении с сигналом clock.

Листинг 4.7

```
COMPONENT csfifo
   GENERIC (LPM_WIDTH: POSITIVE;
      LPM_NUMWORDS: STRING := "UNUSED");
   PORT (data: IN STD_LOGIC_VECTOR(LPM_WIDTH-1 DOWNTO 0);
      wreq, rreq, clock, clockx2, clr: IN STD_LOGIC;
      empty, full: OUT STD_LOGIC;
      q: OUT STD_LOGIC_VECTOR(LPM_WIDTH-1 DOWNTO 0)
     );
END COMPONENT;
```

Устройство содержит блок оперативной памяти RAM и два счетчика: CT1 — указатель адреса начала очереди и CT2 — указатель адреса конца очереди. Синхросигнал clockx2 делит период основного тактового сигнала clock на две фазы. Первая фаза отводится для чтения данных их памяти, а вторая — для записи данных. Если любое из этих действий в текущем цикле обращения не предусмотрено, в соответствующей фазе ничего не делается.

Если присутствует запрос на чтение, то в первой фазе на адресный вход RAM подается код со счетчика CT1, а выходной код RAM фиксируется в выходном регистре. После этого выполняется инкремент содержимого CT1. Если присутствует запрос на запись, то во второй фазе на адресный вход подается код со счетчика CT2, на входе данных RAM присутствует код, зафиксированный во входном регистре данных, и активизируется вход разрешения записи WE. После этого выполняется инкремент содержимого CT2. Если присутствуют запросы и на запись, и на чтение, то оба процесса выполняются последовательно в соответствующих фазах работы устройства.

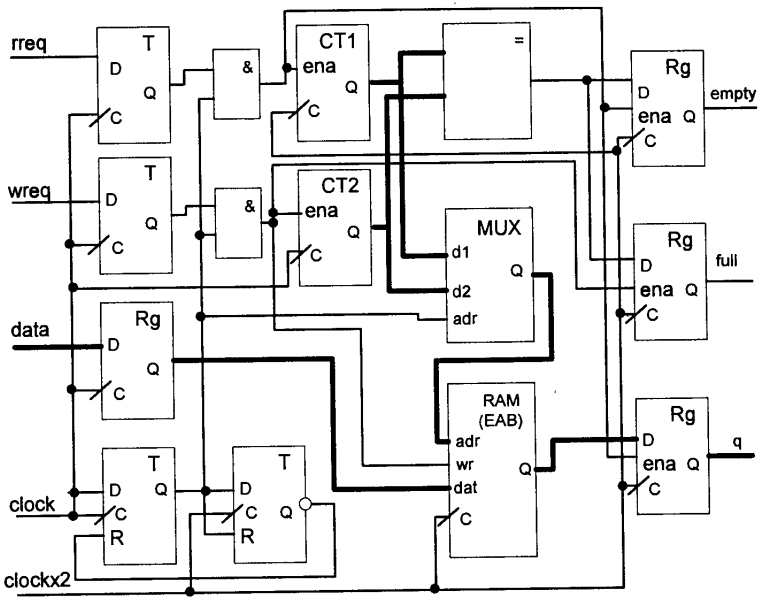


Рис. 4.6. Структурная организация библиотечного модуля csfifo

Если в конце цикла работы содержимое счетчиков CT1 и CT2 оказалось равным, то, в зависимости от типа цикла, должны выдаваться дополнительные сигналы состояния. Если такое событие произошло в цикле, в котором было только чтение, это является признаком очистки буфера (empty), а если в цикле, в котором была только запись, это является признаком полного заполнения буфера (full). В цикле чтения/записи такая ситуация возникнуть не может.

4.2.3. Память с ассоциативным доступом

В ассоциативной памяти поиск информации выполняется исходя из некоторого признака искомых данных, а не из ее предполагаемого расположения (адреса или номера в очереди).

В наиболее полной версии ячейка ассоциативной памяти содержит два поля — поле признака, называемого тегом или образом (Pattern), и поле данных. При выборке во всех ячейках памяти одновременно проверяется совпадение полей тегов с признаком, задаваемым входным словом (теговым адресом), и на выход выдаются слова, удовлетворяющие признаку. Признаком поиска может быть все входное слово или набор указанных разрядов входного слова. Для выделения существенных разрядов прибегают к маски-

рованию. Код маски может подаваться на вход одновременно с кодом признака, а в некоторых версиях сохраняется в ячейках памяти. Разряды, отмеченные в коде маски как несущественные, не учитываются при анализе искомого признака и тега на совпадение. Дисциплина выдачи слов в случае, если тегу удовлетворяют несколько слов, а также дисциплина записи новых данных, могут быть самыми разнообразными в зависимости от области применения [27].

Подвидом ассоциативной памяти является Content Addressable Memory (CAM), т. е. память, адресуемая по содержанию [48]. В CAM результатом обращения является номер ячейки памяти, содержимое в которой соответствует признаку поиска. Впрочем, если полученный таким образом адрес подать на блок памяти RAM, сохраняющий фактические данные, то получим полное ассоциативное запоминающее устройство.

Для пояснения принципов обращения к ассоциативной памяти рассмотрим рис. 4.7.

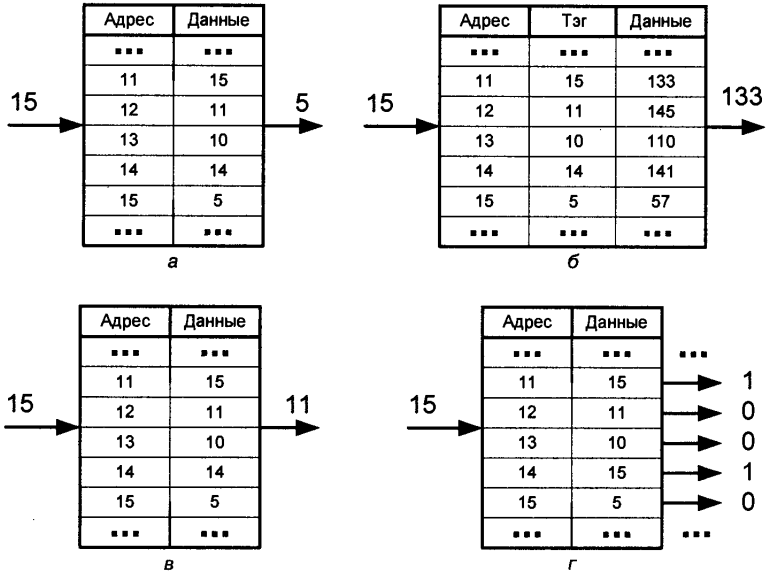


Рис. 4.7. Сравнение процессов выборки в адресной памяти (*а*), полной ассоциативной памяти (*б*) и CAM (*в*, *г*)

На рис. 4.7, *а* для сравнения представлена "классическая" память с адресным доступом. Здесь при выборке на выход выдается содержимое ячейки памяти, номер которой (адрес) подан на вход. На рис. 4.7, *б* изображена

полная ассоциативная память. В ней при выборке адрес является несущественным параметром. На выход выдается поле данных той ячейки, в которой обнаружено совпадение поля тега и входного слова. Заметим, что физически адрес не сохраняется в ячейке, а соответствующее поле на рисунках лишь отражает один из выходов дешифратора адреса.

Память типа CAM представлена в двух вариантах. В варианте, представленном на рис. 4.7, *в*, при чтении на выход передается код адреса ячейки памяти, в которой обнаружено совпадение образа и признака поиска. Причем, если обнаруживается несколько образов, удовлетворяющих признаку поиска, на выходные линии могут последовательно выдаваться несколько адресов, начиная с наименьшего. Вариант, представленный на рис. 4.7, *г*, называют схемой с декодированным адресным выходом. Здесь число выходных линий равно числу слов в памяти, причем выходной код содержит единицы в разрядах, соответствующих ячейкам, содержимое которых удовлетворяет признаку поиска, и нули в остальных разрядах.

В большинстве реализаций ассоциативной памяти физический адрес используется в процессе записи для задания места сохранения заносимых данных. Но известны устройства, в которых новая запись замещает наиболее долго не используемую (дисциплина Last Recently Used, LRU).

Наиболее распространенная область использования ассоциативной памяти — кэширование данных, т. е. запоминание копий информации, передаваемой между какими-либо устройствами (например процессором и основной памятью) для уменьшения числа передач по магистрали. Используется ассоциативная память также для модификации логических адресов в системах с виртуальной организацией памяти, при аппаратной интерпретации алгоритмов поиска данных.

Рис. 4.8 представляет конфигурацию внешних цепей мегафункции `altcam` из библиотеки параметризованных модулей САПР Quartus версии 1999.10 и выше, а табл. 4.2 — набор параметров настройки этой функции. В микросхемах семейства APEX20KE поддерживается троичное представление записываемой информации, включая уровни логического нуля, логической единицы и значение "не важно" (фактически, маскирование некоторых битов сравниваемых кодов). В более ранних семействах и в семействе Mercury поддерживается только поиск без маскирования.

Таблица 4.2. *Набор параметров настройки мегафункции* `altcam`

Параметр	Назначение	Значение по умолчанию
WIDTH	Число разрядов слов памяти	
WIDTHAD	Число разрядов входа адреса	
NUMWORDS	Число хранимых слов	2^ WIDTHAD

Таблица 4.2 (окончание)

Параметр	Назначение	Значение по умолчанию
LPM_FILE	Файл, задающий содержимое памяти, устанавливаемое при инициализации	UNUSED
MATCH_MODE	В режиме MULTIPLE допускается совпадение признака с содержимым нескольких ячеек, иначе наличие нескольких совпадений приводит к ошибочной работе	MULTIPLE
OUTPUT_REG	Задает использование стробируемого буферного регистра на выходе	UNREGISTERED
OUTPUT_ACLR	Задает использование входа сброса буферного регистра на выходе	ON
PATTERN_REG	Задает использование стробируемых буферных регистров на входах данных и адреса	INCLOCK
PATTERN_ACLR	Задает использование входа сброса буферного регистра на входе данных	ON
WRADDRESS_ACLR	Задает использование входа сброса буферного регистра на входе адреса	ON
WRX_REG	Задает использование стробируемого буферного регистра на входе маски	INCLOCK
WRX_ACLR	Задает использование входа сброса буферного регистра на входе маски	ON

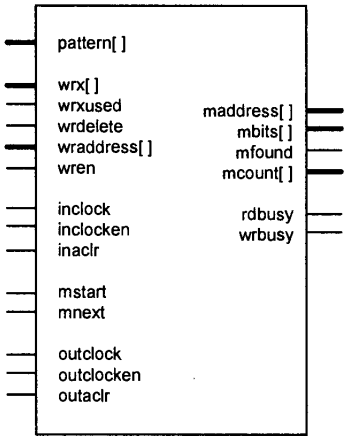


Рис. 4.8. Конфигурация внешних соединений мегафункции altcam

Для того чтобы записать новую информацию в память без применения маскирования, достаточно в течение двух периодов синхросигнала inclock поддерживать активный (единичный) уровень на входах разрешения тактового сигнала inclocken и разрешения записи wr_en, а также сохранять неизменными код адреса wraddress и данных pattern. Если используется режим маскирования битов, то одновременно активизируется вход wrxused, а на вход wrx подается код маски, в котором биты, установленные в единицу, отмечают разряды, не учитываемые при сравнении. Запись данных с маскированием должна продолжаться не менее трех периодов синхросигнала. Если подан активный сигнал на вход wrdelete, то содержимое ячейки, адрес которой присутствует на входе wraddress, стирается.

Реализация процесса чтения зависит от режима, задаваемого параметром конфигурации MATCH_MODE. В простейшем режиме SINGLE-MATCH MODE выявляются только однократные совпадения. Если окажется, что несколько образов удовлетворяют признаку, правильная работа не гарантируется. Признак поиска подается на вход pattern и фиксируется по сигналу inclock. Если образ, соответствующий признаку, имеется в памяти, то по сигналу outclock при единичном уровне на входе outclocken на выход выдается адрес ячейки, в которой обнаружено совпадение. При этом на выходе mbits присутствует дешифрированный адрес, т. е. код, содержащий единицу в разряде, номер которого соответствует адресу совпадения. Дополнительно при обнаружении совпадения в единицу устанавливается выход mfound.

В режиме MULTILPLE-MATCH допускается совпадение нескольких образов с признаком. Чтение в этом режиме требует минимум двух тактов сигнала. Доступ инициализируется подачей признака на вход pattern и единичного сигнала на вход mstart. К концу второго такта на выход maddress выдается наименьший адрес из числа адресов ячеек, в которых обнаружено совпадение, а на выход mbits — код, содержащий единицы во всех разрядах, соответствующих адресам ячеек, в которых обнаружено совпадение. Кроме того, при обнаружении совпадений их общее число выдается на выход mcount, а выход mfound устанавливается в единицу. Адреса следующих ячеек, в которых обнаружены совпадения, могут быть получены последовательно после очередных тактовых сигналов.

Режим FAST MULTIPLE-MATCH аналогичен предыдущему, но отличается тем, что для получения информации о первом совпадении требуется только один такт работы. Платой за такое ускорение является увеличение аппаратурных затрат почти вдвое.

Затраты на реализацию CAM емкостью 32 слова по 32 разряда для режима SINGLE_MATCH в микросхемах семейства APEX 20KE составляют один блок ESB и 35 логических макроячеек.

Использование CAM позволяет существенно повысить производительность систем, в которых реализуются поисковые алгоритмы.

4.3. Цифровые фильтры

Обычно цифровая обработка сигналов (ЦОС) предусматривает реализацию двух взаимодополняющих и взаимозависимых задач. Первая — это выделение некоторой совокупности признаков, характеризующих исследуемый сигнал, например вычисление корреляционных моментов, энергетических характеристик сигнала, спектральный анализ, частотная селекция и т. д. Следующая задача — логический анализ совокупности признаков с целью идентификации объекта или принятия определенных решений в зависимости от результатов анализа. Во многих случаях достаточно реализации лишь первой задачи. Особенно это относится к устройствам, результат обработки в которых непосредственно воспринимается человеком, например фильтрам в аудио- и видеосистемах.

Трудоемкость преобразований, реализуемых на этапе логического анализа, как правило, значительно уступает трудоемкости этапа выделения признаков. К тому же логический анализ трудно поддается распараллеливанию. Поэтому в настоящее время аппаратная реализация задач этого этапа практически не используется. В то же время аппаратная поддержка предварительных этапов обработки обеспечивает значительный рост производительности систем обработки сигналов и изображений и является во многих случаях оптимальным решением по критерию соотношения производительности и стоимости. Результаты предварительной обработки (отфильтрованный сигнал, спектральная функция и т. д.) можно передавать в процессорное ядро системы для логического анализа. В данной книге мы остановимся только на реализации типовых задач этапа выделения признаков в наблюдаемом сигнале, прежде всего задачи фильтрации.

Основной операцией ЦОС является дискретная свертка, вычисляемая как скалярное произведение двух векторов

$$y = \mathbf{X} \times \mathbf{V} = \sum_{(i)} x_i v_i, \qquad (4.1)$$

где **X** и **V** — векторы, причем $\mathbf{X}=\{x_i\}$, $i = 1, ..., n$ — последовательность отсчетов обрабатываемого сигнала.

Компонентами вектора $\mathbf{V}=\{v_i\}$, $i = 1, ..., n$ в зависимости от постановки задачи могут быть отсчеты того же или другого сигнала (вычисление коэффициентов корреляции и автокорреляции), совокупность значений базисной функции (спектральный анализ) или постоянные коэффициенты (классические цифровые фильтры частотной селекции).

Если вычисляется не одно, а одновременно несколько результирующих значений, то преобразование можно записать в векторно-матричной форме:

$$\mathbf{Y} = \mathbf{A} \times \mathbf{X}$$

что эквивалентно:

$$y_j = \mathbf{X} \times \mathbf{A} = \sum_{(i)} x_i a_{ji}, \qquad (4.2)$$

Выражение (4.2) является основой для вычисления корреляционных и спектральных функций, для многополосной фильтрации. Учет взаимной обусловленности коэффициентов матрицы **A** позволяет уменьшать число операций типа сложение/умножение и является основой для построения быстрых алгоритмов спектральных преобразований.

Основные пути и особенности выполнения операции дискретной свертки проиллюстрируем на примере реализации нерекурсивных цифровых фильтров, иначе называемых фильтрами с конечной импульсной характеристикой (КИХ-фильтры). При вычислении очередного отсчета результата КИХ-фильтра в качестве компонентов одного из векторов свертки используются отсчеты входного сигнала в предыдущие моменты времени, а второй вектор является упорядоченной совокупностью значений дискретной импульсной переходной характеристики, т. е.:

$$y(jT) = \sum_{i=0}^{n-1} x[(j-i)T] \times h(iT),$$

где T — период дискретизации, j — порядковый номер отсчета выходного сигнала, h — дискретная импульсная переходная характеристика фильтра.

Или в сокращенной форме записи:

$$y_j = \sum_{i=0}^{n-1} x_{j-i} h_i. \qquad (4.3)$$

Структурной моделью КИХ-фильтра является цепочка элементов, каждый из которых задерживает принимаемое значение на время одного периода дискретизации, а выход каждого соединен с входом следующего и схемой умножения на постоянный коэффициент. Выходы умножителей суммируются для получения очередного отсчета выходного сигнала.

В большинстве случаев используют КИХ-фильтры с линейной фазово-частотной характеристикой, потому что, как известно, такие фильтры сохраняют форму преобразуемого сигнала в полосе пропускания. Одной из особенностей фильтров с линейной фазово-частотной характеристикой является симметрия вектора коэффициентов $\{hi\}$ относительно его середины, т. е.

$h_{n/2+i} = h_{n/2-(i+1)}$, $i = 0, n/2$, при четном n;

$h_{(n-1)/2+i} = h_{(n-1)/2-i}$, $i = 1, (n-1)/2$, при нечетном n.

То есть значения отсчетов некоторых пар сигналов можно складывать до выполнения умножения на одинаковые коэффициенты [23]. Структурная модель такой реализации для $n = 8$ представлена на рис. 4.9.

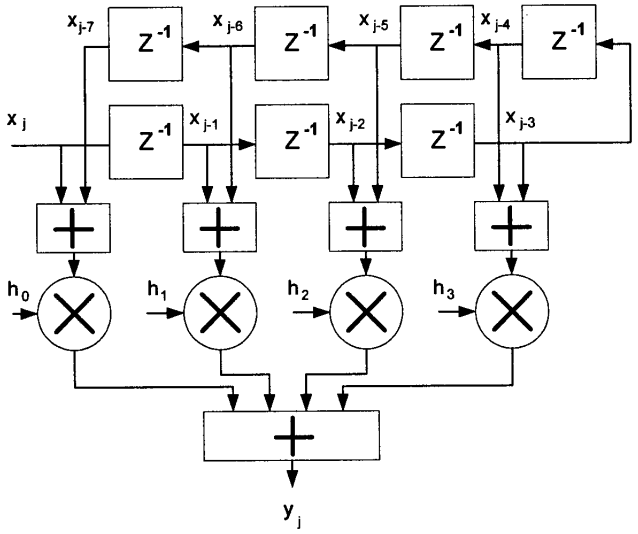


Рис. 4.9. Структурная модель КИХ-фильтра

Как и при реализации других вычислительных алгоритмов, вычисление дискретной свертки возможно как за счет последовательного выполнения операций умножение-накопление в единственном операционном блоке, так с использованием разнообразных структур параллельного и параллельно-последовательного типа.

Поведенческое описание устройства последовательного типа на языке VHDL представлено в листинге 4.8. Программа предполагает, что по внешнему сигналу `samlpe`, отмечающему появление очередного отсчета входного сигнала, происходит сдвиг информации в линии задержки и запись нового отсчета в первую ячейку этой линии. Импульс `samlpe` синхронизирован с тактовым сигналом, но появляется значительно реже тактового сигнала. В течение следующих `power/2` тактов (параметр `power` задает порядок фильтра и, соответственно, число элементов задержки) выполняется последовательное умножение значений отсчетов, сохраняемых в линии задержки, на соответствующие коэффициенты, и прибавление произведения к накопленной за n предыдущих тактов сумме. После этого накопленная сумма произведений передается на выход, а работа схемы приостанавливается до возникновения сигнала о появлении очередного отсчета. Предполагается,

что константа `xmax` (диапазон представления входных данных) и тип `data_array` определены в одном из доступных пакетов как

```
CONSTANT xmax: integer;
TYPE data_array IS ARRAY RANGE <> of integer RANGE -xmax TO +xmax;
```

Диапазон выходных данных фильтра может уточняться по результатам исследования типовых сигналов в проектируемой системе. В данном примере принято, что код результата расширяется на $\log_2$ `xmax`-разрядов по сравнению с кодом исходных данных: удвоение разрядности дает умножение, а дальнейшее расширение может требоваться при сложении.

Листинг 4.8

```
ENTITY fir IS
GENERIC (power:integer:=8; -- порядок фильтра, предполагается четный;
        h:data_array);
PORT ( clock,new_sample: IN bit;
          input: IN integer RANGE -xmax TO xmax;
         output:OUT integer RANGE -4*xmax*xmax TO 4*xmax*xmax);
ARCHITECTURE seqiential_behave OF fir IS
SIGNAL pipe:data_array(power-1 DOWNTO 0);
BEGIN Pipe_line_shifting: PROCESS (clock)
VARIABLE j:INTEGER;
  BEGIN IF (clock='1' and clock'event and new_sample='1') THEN
FOR j IN  power-1 DLOWNTO 1 LOOP
    Pipe(j)<=pipe(j-1);
END LOOP;
pipe(0)<= input;
end if;
END PROCESS;
multiply_and_accumulate:PROCESS(clock)
   VARIABLE i: integer;
   VARIABLE result: integer  RANGE -2*xmax TO 2*xmax);
Begin
 IF clock='1' and clock'event then
      If new_sample='0' THEN
             i:=0; result:=0;
       ELSIF i<power/2 THEN  result :=(pipe(I)+pipe(power-i))*h(i);
              i:=i+1;
        ELSIF i=power/2 THEN
             output<=result
   END IF;
END IF;
END PROCESS;
END sequential_behave;
```

Если последовательный вариант не удовлетворяет по быстродействию, то приходится прибегать к параллельным, предпочтительно к конвейерным, реализациям.

Учитывая параллельность путей распространения информации в структуре вычислителя свертки (см. рис. 4.9), можно расположить параллельно исполняемые преобразования в общих ярусах конвейерной структуры, причем результаты в каждом ярусе фиксируются в регистрах, синхронизируемых общим сигналом. На каждую двуместную операцию суммирования выделяется один ярус. В том числе сумматоры, необходимые для реализации одной операции умножения, можно расположить в нескольких различных ярусах.

Наиболее специфические фрагменты описания конвейерной схемы на языке VHDL представлены в листинге 4.9. Декларация ENTITY, а также описание линии задержки здесь опущены, т. к. в точности соответствуют таким же разделам программы, приведенной в листинге 4.8. Используется структурно-поведенческое представление, причем для реализации конвейерного умножителя предполагается включение модуля pipe_mul, представленного в листинге 3.30 *(см. разд. 3.2.6)*. Отметим, что все промежуточные суммы объявлены как сигналы, и присвоения этим сигналам значений записаны в общем теле процесса, что и соответствует конвейерной организации процедуры накопления результатов.

Листинг 4.9

```
ARCHITECTURE pipe_lined OF fir IS ....
COMPONENT  pipe_mul ....END COMPONENT;
...
SIGNAL z0,z1,z2,z3: integer range -2* xmax  to 2*xmax;
CONSTANT double:integer= 2*xmax*xmax;
SIGNAL  mult0,mult1, mult2,mult3: integer  RANGE – double TO double;
SIGNAL v1,v2: integer RANGE -2*double TO 2*double;
m0: pipe mul
         GENERIC MAP(width=>8)
         PORT MAP( z0, h0, clk, mult0);
 m1: pipe mul
          GENERIC MAP(width=>8)
           PORT MAP(z1, h1, clk,mult1);
m2: pipe mul
          GENERIC MAP(width=>8)
          PORT MAP( z2, h2, clk,mult2);
 m3: pipe mul
          GENERIC MAP(width=>8)
          PORT MAP(z3, h3, clk,mult3);
adding: PROCESS(clk)
    z0<=x(0)+x(7);
    z1<=x(1)+x(6);
```



```
    z2<=x(2)+x(5);
    z3<=x(3)+x(4);
    v1<= mult0+mult1;
    v2<= mult2+mult3;
    out<=v1+v2;
END PROCESS;
```

Если некоторые промежуточные данные, в том числе промежуточные суммы в описании умножителя, объявить как переменные, получим реализацию конвейера с укрупненными блоками, в каждом из которых будет исполняться несколько сложений. Уменьшение числа ступеней конвейера уменьшает число тактов задержки результата относительно моментов поступления данных, но это для большинства приложений не является существенным. В то же время, уменьшение объема преобразований в каждой ступени уменьшает необходимую длительность такта работы схемы и дает возможность обрабатывать более высокочастотные сигналы.

Замечание

Наряду с фильтрами с конечной импульсной характеристикой известны рекурсивные фильтры, иначе называемые фильтрами с бесконечной импульсной характеристикой (БИХ-фильтры). Их особенностью является то, что результат определяется как свертка не только входного сигнала, но и ранее полученных отсчетов выходного сигнала:

$$y_j = \sum_{i=0}^{m-1} x_{j-i}\, b_i - \sum_{i=1}^{n-1} y_{j-i}\, a_i$$

Реализация во многом подобна реализации КИХ-фильтра. Но имеется существенная проблема, а именно: в БИХ фильтре результат должен быть полностью сформирован за время, не превышающее период дискретизации входного сигнала. Это делает практически незффективными конвейерные реализации и, соответственно, ограничивает частотные возможности. Поэтому, хотя для построения подобных по характеристикам фильтров рекурсивные реализации обычно более экономичны в сравнении с КИХ-фильтрами, БИХ-фильтры в критических по времени приложениях, как правило, не используются. Это ни в коем случае не касается последовательных структур невысокого быстродействия.

Пока что мы рассматривали относительно общий случай, когда оба вектора свертки могут оперативно изменяться в процессе обработки. Хотя такой вариант используется достаточно часто, в том числе при построении адаптивных и корреляционных фильтров, а также устройств с одновременным воспроизведением функций нескольких фильтров одного входного сигнала, в большинстве своем разработчики имеют дело с постоянными коэффициентами преобразующего вектора. В этом случае устройство свертки (например, фильтр) может быть упрощено за счет замены операции умножения

прямым преобразованием кодов [23]. Действительно, умножение переменной на константу может быть обеспечено прямой выборкой результата из устройства памяти, хранящей коды произведений, по адресу, задаваемому кодом входной переменной. Однако, как правило, такой "прямолинейный" подход оказывается неприемлем. Во-первых, при больших порядках фильтра количество и объем встроенных в ПЛИС блоков памяти может быть попросту недостаточным, потому что выборка значений произведений из блоков памяти должна осуществляться параллельно, а во-вторых, быстродействие блоков памяти, как правило, хуже быстродействия логических компонентов ПЛИС. В подобных случаях целесообразным решением становится декомпозиция умножителя в базисе типовых ячеек ПЛИС.

Рассмотрим такое представление, ориентируясь на наиболее часто встречающуюся конфигурацию базовой ячейки ПЛИС, которая может настраиваться на воспроизведение произвольной функции четырех переменных. Тогда, например, для восьмиразрядного представления входных данных выражение (4.1) можно переписать в форме

$$y_j = \sum_{i=0}^{n-1}((x_{j-i}[7:4]\times h_i)/16 + (x_{j-i}[3:0]\times h_i)/256) = \sum_{i=0}^{n-1}(a_{j-i} + b_{j-i}), \quad (4.4)$$

где в квадратных скобках записан диапазон разрядов, учитываемых в преобразовании.

Предполагается деление по правилам деления целых чисел с отбрасыванием остатка, т. е. для представления a_{j-i} необходимо максимум 8 разрядов, и, значит, эта функция может быть воспроизведена с использованием 8 ячеек типа четырехвходовой LUT. Аналогично, для реализации b_{j-i} достаточно четырех ячеек. При значительном разбросе значений коэффициентов фильтра разрядность некоторых частичных произведений оказывается еще меньше, что является дополнительным резервом экономии оборудования.

Разложение (4.4) не является единственно возможным. Удачным оказалось объединение в группы одноименных разрядов различных компонентов вектора отсчетов преобразуемого сигнала:

$$y_j = \sum_{i=0}^{7}\sum_{k=0}^{3} x_{j-i}[k]\times h_i \times 2^k + \sum_{i=0}^{7}\sum_{k=4}^{7} x_{j-i}[k]\times h_i \times 2^k, \quad (4.5)$$

где k — номер разряда.

Внутренние суммы в этом выражении являются функциями четырех переменных, и каждый разряд такой суммы воспроизводится единственной ячейкой типа четырехвходовой LUT. Количество разрядов, необходимое для представления этих сумм, уменьшается с уменьшением номера разряда, что позволяет значительно сократить аппаратные затраты.

Рассмотрим этот подход подробнее на примере двумерной дискретной свертки, являющейся базовой операцией многих задач обработки изображений и, прежде всего, двумерной пространственной фильтрации.

Принцип цифровой фильтрации изображений основан на том, что значение показателя яркости или цветности элемента изображения (пиксела) модифицируется с учетом значений показателей соседних элементов [18]. Преобразования, реализуемые двумерным нерекурсивным фильтром, могут быть представлены соотношением:

$$z(i,j) = \sum_{k=ki}^{kk}\sum_{l=li}^{lk}(w_{k,l,i,j}\times x(i+k, j+l)).$$

где $x(i, j)$ и $z(i, j)$ — i-е элементы j-й строки входного и выходного изображений; $w_{k,l,i,j}$ — весовые коэффициенты, задающие вид преобразования; kk, ki, li, lk — границы диапазона элементов, учитываемых при формирования отсчетов выходного изображения.

В практических приложениях наиболее часто учитывают лишь ближайшие соседние элементы ($ki = li = -1$, $kk = lk = 1$) и предполагают пространственную инвариантность преобразования ($w_{k,l,i,j} = w_{k,l}$). Иными словами, выполняется точечное (т. е. поэлементное) умножение матрицы, составленной из смежных отсчетов входного сигнала, на матрицу коэффициентов преобразования

$$\mathbf{A} = \mathbf{X} \bullet \mathbf{W} = \begin{vmatrix} x_{i-1,j-1} & x_{i,j-1} & x_{i+1,j-1} \\ x_{i-1,j} & x_{i,j} & x_{i+1,j} \\ x_{i-1,j+1} & x_{i,j+1} & x_{i+1,j+1} \end{vmatrix} \bullet \begin{vmatrix} w_{-1,-1} & w_{0,-1} & w_{1,-1} \\ w_{-1,0} & w_{0,0} & w_{1,0} \\ w_{-1,1} & w_{0,1} & w_{1,1} \end{vmatrix}, \quad (4.6)$$

с последующим суммированием членов матрицы **A**.

Замечание

В рамках данной книги мы ограничимся этим случаем, хотя при дополнительных затратах ресурсов элементов программируемой логики возможна реализация фильтров, учитывающих большие зоны взаимного влияния отсчетов.

Не останавливаясь на способах выбора коэффициентов матрицы **W**, отметим, что варианты их задания обеспечивают разнообразные виды преобразований, в том числе $W_{i,j} = \text{const}$, что соответствует усредняющему фильтру. Матрица вида

$$\begin{vmatrix} k & 1 & k \\ 1 & M & 1 \\ k & 1 & k \end{vmatrix},$$

обеспечивает реализацию алгоритмов сглаживания, причем степень сглаживания зависит от соотношения параметров k и M.

Если в этой матрице знаки коэффициентов, кроме центрального, изменить на отрицательные, то получим обостряющий фильтр. Таким образом, многие функции обработки изображений могут быть реализованы в рамках одной и той же структуры за счет изменения коэффициентов.

Анализ типовых матриц преобразования и сопоставление выражения (4.5) с возможностями современных микросхем программируемой логики позволяет сделать следующие выводы.

В большинстве практических приложений отсчеты изображения представляются как беззнаковые целые в восьмибитовом формате, причем используются целочисленные значения параметров M и k из весьма ограниченного диапазона (обычно между 0 и 20). В принципе, это дает возможность применять умножители с малой разрядностью множителя. Однако для сохранения средней яркости изображения приходится нормировать результаты путем деления взвешенной суммы отсчетов на сумму весов. Деление является операцией, трудно реализуемой в среде программируемой логики. Поэтому целесообразно использовать умножения 8-битовых данных на 8-битовые коэффициенты, причем коэффициенты преобразований следует скорректировать в соответствии с соотношением:

$$w^{*}_{i,j} = ent((w_{i,j} / \sum_{(i,j)} w) \times 256).$$

Деление результата на 256 получается отбрасыванием младшего байта.

Последовательная реализация всех преобразований для каждого пиксела в одном умножителе-накопителе на современных ПЛИС не обеспечивает требований обработки в темпе поступления данных в видеосистему. Приходится переходить к параллельной или параллельно-последовательной реализации (например, последовательные преобразования данных для элементов одной строки и параллельное исполнение преобразований для разных строк) в сочетании с конвейеризацией вычислений.

Данные на вход фильтра поступают последовательно строка за строкой. Поэтому для реализации двумерной фильтрации необходимо использовать FIFO-буфер емкостью не менее двух строк изображения. Из возможных вариантов реализации такого буфера наиболее целесообразным представляется использование встроенных блоков памяти программируемых БИС. В частности, БИС семейства FLEX10K содержат встроенные блоки RAM суммарной емкостью до 24 Кбайт и допускающие режим READ-MODIFY-WRITE т. е. считывание данных и запись новых по одному адресу в одном цикле обращения. Возможность совмещения считывания и записи в одном цикле позволяет перейти от традиционной архитектуры FIFO к памяти с циклическим доступом, отличающейся постоянным местом хранения принимаемой информации и общим для обоих буферов счетчиком адреса доступа, который инкрементируется после каждого цикла и переходит к нулевому адресу после переполнения.

Число умножений, необходимое для вычисления значения одного пиксела для окна преобразования 3×3, равно 9. Однако большинство матриц преобразования симметричны — в приведенных примерах симметричны относительно центра матрицы, некоторые алгоритмы используют матрицы с вертикальной, горизонтальной или диагональной симметрией.

Сумму вида

$$S(i,j,r) = \sum_{k=-1}^{1} w_r \times x(i+k, j+1); \quad r \in \{-1,0,1\},$$

назовем сверткой по r-му столбцу окна преобразования изображения для точки i, j.

Очевидно, что в случае вертикально симметричной матрицы

$$S(i,j,1) = S(i+2, j, -1),$$

т. е. члены, соответствующие произведениям элементов правых колонок матрицы коэффициентов, и окна изображения, полученные при вычислении элемента изображения $z(i, j)$, можно использовать и для вычисления элемента $z(i+2, j)$. Это позволяет уменьшить число параллельно выполняемых умножителей до 6. Если алгоритм обработки предусматривает матрицу, симметричную и относительно центральной вертикали, и относительно центральной горизонтали, то число операций умножения на один пиксел можно довести до трех. Но это оказалось неэффективным, т. к. потребует увеличения объема памяти для хранения вычисленных неполных сумм произведений для последних двух строк пикселов. Компромиссом является использование только свойств вертикальной симметрии в сочетании с предварительным суммированием таких элементов текущего окна, которые умножаются на одинаковые коэффициенты, подобно одномерному фильтру. В этом случае требуется четыре умножения и память для сохранения значений отсчетов двух последних строк изображения.

Исходя из вышеизложенного, можно рекомендовать структурную схему вычислительного блока фильтра изображений, приведенную на рис. 4.10. Здесь каждый синхроимпульс вызывает загрузку нового отсчета на вход и сдвиг содержимого буферов. Буферные схемы BUF обеспечивают задержку на один период тактовой частоты каждая. Если в некоторый момент времени узел свертки (MULT_ADD) **A** вычисляет сумму произведений трех смежных элементов одного столбца изображения на центральные элементы матрицы преобразования, то в этот же момент узел свертки **B** вычисляет сумму произведений следующих трех элементов тех же строк на элементы правого столбца матрицы преобразования, формируя свертку по правой колонке этого же

окна преобразования. В качестве свертки левых столбцов матрицы преобразования и окна входного изображения используется эквивалентная ей свертка правых столбцов матриц, сформированная на два такта ранее.

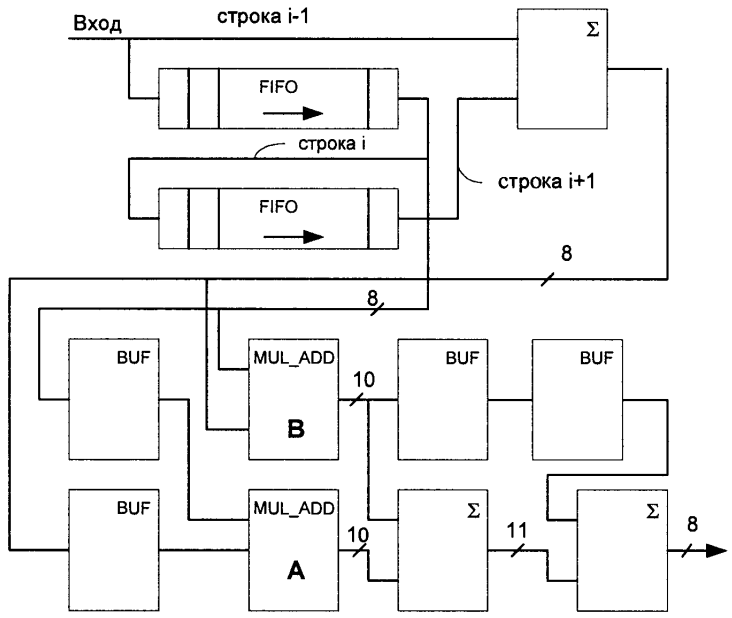


Рис. 4.10. Структура двумерного нерекурсивного фильтра

Реализация операции умножение-накопление может быть выполнена на основе комбинационного матричного умножителя или по конвейерной схеме. Однако для случая постоянства коэффициентов оказывается возможным отказаться от операции умножения как таковой и заменить блоки перемножения памятью или комбинационной логикой. Но и в реализациях на основе логических преобразований целесообразно разбивать схему по глубине на цепочку синхронизируемых блоков.

Как было показано при рассмотрении одномерных фильтров, при интерпретации умножения на константу с использованием ячеек типа LUT наиболее экономичная реализация получается, если выполнена декомпозиция выражения свертки так, чтобы были объединены одноименные или соседние разряды преобразуемых данных. В этом случае декомпозиция свертки по средней (нулевой) колонке окна в базисе четырехвходовых ячеек типа LUT имеет вид:

$$
\begin{aligned}
S_{i,j,0} = {} & (x_{i,j}[1..0]\ w_{00} + x^*_{i,j}[1..0]\ w_{01}) + \\
& + (x_{i,j}[3..2]\ w_{00} + x^*_{i,j}[3..2]\ w_{01}) \times 24 + \\
& + (x_{i,j}[5..4]\ w_{00} + x^*_{i,j}[5..4]\ w_{01}) \times 28 + \\
& + (x_{i,j}[7..6]\ w_{00} + x^*_{i,j}[7..6]\ w_{01}) \times 212 + \\
& + x^*_{i,j}[8]\ w_{0,1} \times 216 = \\
& = V_{10} + V_{20} + V_{30} + V_{40} + V_{50},
\end{aligned}
\tag{4.7}
$$

где $x^*_{i,j} = x_{i,j+1} + x_{i,j-1}$.

Как видно, при таком разложении совместному преобразованию подвергаются пары одноименных разрядов различных входных кодов.

Зависимость члена разложения V_{10} от младших битов аргументов $x_{i,j}$ и $x^*_{i,j}$ приведена в табл. 4.3. Таблицы функций V_{20}, V_{30} и V_{40} полностью тождественны, а выражение для члена V_{50} очевидно.

Таблица 4.3. *Зависимость члена разложения V_{10} от младших битов аргументов $x_{i,j}$ и $x^*_{i,j}$*

$x_{i,j}[1..0]$	$x^*_{i,j}[1..0]$	V_{10}
00	00	0
01	00	W_{00}
10	00	$2\times W_{00}$
11	00	$3\times W_{00}$
00	01	W_{01}
01	01	$W_{00}+W_{01}$
10	01	$2\times W_{00}+W_{01}$
11	01	$3\times W_{00}+W_{01}$
00	10	$2\times W_{01}$
01	10	$W_{00}+2\times W_{01}$
10	10	$2\times W_{00}+2\times W_{01}$
11	10	$3\times W_{00}+2\times W_{01}$
00	11	$3\times W_{01}$
01	11	$W_{00}+3\times W_{01}$
10	11	$2\times W_{00}+3\times W_{01}$
11	11	$3\times W_{00}+3\times W_{01}$

Для представления без потерь каждого скобочного выражения в (4.7) достаточно 11 разрядов, но относительный вклад членов разложения в окончательный результат уменьшается по логарифмическому закону с уменьшением первого индекса их обозначения. Поэтому практически без потери точности можно не воспроизводить восемь младших разрядов члена V_{10}, шесть младших разрядов V_{20} и т. д.

Выходы логических схем, воспроизводящих отдельные члены разложения (4.7), последовательно суммируются по пирамидальной схеме с учетом соотношения их весов.

Свертка по правому столбцу окна преобразования строится аналогично, а в качестве значения свертки по левому столбцу можно использовать результат свертки по правому столбцу ранее обработанного окна.

Для получения наилучших показателей по быстродействию и затратам оборудования при реализации сверток следует произвести детальный анализ информационных путей и выполнить коррекцию связей. Составление расписания продвижения данных в конвейерных структурах позволяет избежать возможных ошибок рассинхронизации потоков данных и скорректировать задержки параллельных каналов, выделить общие компоненты, присутствующие в параллельных потоках, и в некоторых случаях объединить их, а в данном случае также обеспечить объединение на входах сумматоров пирамиды суммирования термов, наиболее близких по формату представления.

Рассмотрим процедуру составления расписания для узла двумерной свертки, построенного из двух одномерных сверток вида (4.7).

Обозначим M_i — набор данных, в том числе промежуточных, необходимых для вычисления свертки среднего столбца окна изображения с центром в точке с координатами $\{i, j\}$, R_i — набор данных для вычисления свертки правого столбца, а L_i — левого столбца того же окна.

Предполагаем, что блоки LUT реализованы в виде комбинационной логики, а сумматоры и буферы — на триггерных ячейках с общей синхронизацией. Рассмотрим рис. 4.11.

Пусть в некоторый момент времени на вход узла поступают с FIFO-буферов данные, соответствующие (i+1)-м пикселам строк j−1, j и j+1. На линиях **A** и **B** этот момент присутствуют отсчет i-го пиксела j-й строки и суммы отсчетов i-х пикселов строк j−1 и j+1, в совокупности составляющие набор M_i для первого слоя конвейера. Но эти же данные являются исходными для вычисления правого столбца предыдущего окна (R_{i-1}) и левого столбца следующего окна (L_{i+1}). Одновременно на выходах сумматоров 1, 2, 9, 10 присутствуют суммы соответствующих членов разложения (4.7) для предыдущего отсчета M_{i-1}, R_{i-2}, L_i. Четыре верхние LUT воспроизводят умножение входного набора на боковой столбец матрицы коэффициентов, а четыре нижние — на центральный столбец матрицы коэффициентов. На выходах

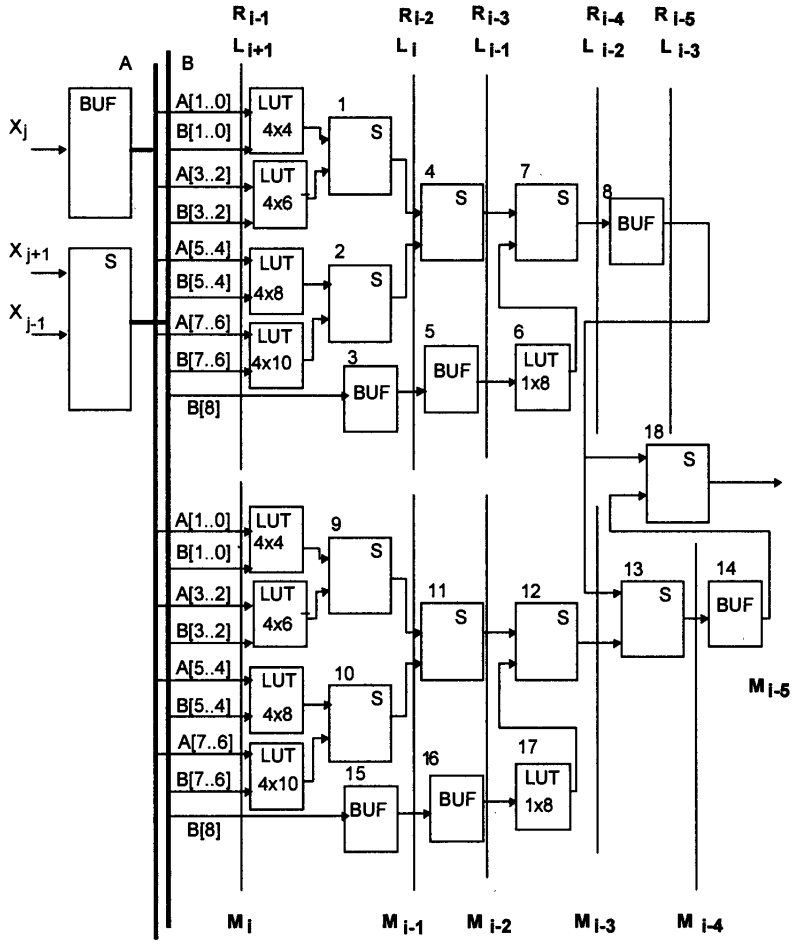


Рис. 4.11. Функциональная схема двумерного фильтра

сумматоров 4 и 11 находятся частичные суммы для набора данных M_{i-2}, R_{i-3}, L_{i-1} и т. д. Чтобы соответствующие данные синхронно появились в точках суммирования результатов свертки по столбцам, приходится включить буферы задержки на один такт 3, 5 и 8. Например, в рассматриваемый момент времени сумматор 13 складывает свертку центральных столбцов матриц, соответствующих (i−3)-му элементу строки со сверткой левых столбцов

тех же матриц, а на выходе сумматора 18 имеем окончательный результат обработки для (i–6)-го пиксела j-й строки.

При рассмотрении путей реализации в ПЛИС основных процедур ЦОС нельзя не обратить внимание на то, что большое число фирм предлагают готовые решения в этой области, относящиеся к категории IP (Intellectual Properties).

В частности, фирма Altera в составе библиотеки MegaCore предлагает набор из более 60 мегафункций ЦОС, включающий КИХ-фильтры, медианный фильтр, двумерные фильтры, процессоры быстрого преобразования Фурье и быстрого косинусного преобразования, аудио- и видеокодеки, модули сжатия сигналов и многое другое [32]. Модули оптимизированы и верифицированы для реализации в микросхемах семейств APEX и FLEX®, но многие из них могут быть экспортированы в системы проектирования других фирм в форме EDIF- и VHDL-файлов (хотя при конвертации эффективность реализации может ухудшаться).

В частности, для синтеза, тестирования и реализации КИХ-фильтров фирма Altera разработала специализированный компилятор. В рекламных материалах фирмы утверждается, что использование этого компилятора уменьшает время проектирования высокопроизводительных цифровых фильтров с нескольких недель до одного дня, а полученная реализация в десятки раз экономичнее, чем реализация на стандартных процессорах ЦОС. Компилятор имеет графический интерфейс и позволяет задавать вид частотной характеристики, специфицировать формат данных, порядок фильтра, коэффициенты интерполяции и децимации. В число опций проекта входит также вариант структурной реализации — параллельная или последовательная. Обеспечена стыковка со средствами проектирования системного уровня третьих фирм, например MATLAB, Simulink, а также системами моделирования на языках VHDL и Verilog.

Все это в значительной мере освобождает разработчика от рутинных работ по синтезу, описанию и отладке фрагментов ЦОС проекта. Тем не менее нам представляется, что материал данного раздела поможет разработчикам лучше осознать основные проблемы и пути реализации ЦОС в структурах программируемой логики, грамотно подойти к выбору того или иного варианта, а при возможности осуществить необходимые доработки, исходя из требований конкретного проекта.

4.4. Пример автоматизированного проектирования аппаратно-программной системы

Данный раздел посвящен рассмотрению полной проектной процедуры для ПЛИС типа SOPC. Современные методы и средства проектирования рассмотрим на примере разработки микропроцессорной системы, являющейся

модернизацией более ранней разработки устройства на базе микроконтроллера MCS-48. Реализация функций реального прибора, предназначенного для увлажнения дыхательной смеси, с целью сокращения объема примера существенно упрощена.

Так же, как в условиях реальных разработок, сузим область анализируемых вариантов. При переходе на новую элементную базу проектировщик должен стараться максимальным образом использовать ранее разработанные и проверенные решения, прежде всего, это касается фрагментов программного обеспечения. В рамках этого пожелания в данном проекте достаточно естественным представляется при анализе возможных вариантов реализации МП-ядра ограничиться рассмотрением вариантов, базирующихся на MCS-51, поскольку при этом упрощается процедура перевода большинства решений MCS-48 на MCS-51. В качестве элементов программируемой логики (кроме специально оговоренных случаев) будем ориентироваться на продукцию фирмы Altera. Это допустимо, поскольку проектируемый прибор не должен обладать каким-либо исключительными свойствами — малым потреблением мощности, обеспечением секретности разработки (наличием битов секретности), требованием радиационной стойкости и, соответственно, ориентации на ПЛИС с пережигаемыми перемычками и т. д.

В состав разрабатываемой системы должен входить 12-разрядный аналого-цифровой преобразователь, записывающий по запросу параллельный код в буферное ОЗУ емкостью 256 двенадцатиразрядных слов, 8 входов и 8 выходов для дискретных сигналов и блок сравнения двух входных восьмиразрядных кодов с содержимым регистра уставок, которое задается программным обеспечением МП-ядра и сохраняет значения текущих установок. Использование схем класса СИС и МИС должно быть в модернизированном варианте минимизировано за счет применения схем программируемой логики.

На начальном этапе проектирования осуществляется анализ и выделение общесистемных ограничений на основе требований на разработку микропроцессорной системы. Основным содержанием этапа является разделение функций между программным обеспечением, опирающимся на стандартные компоненты микроконтроллерной части системы, и специфическим аппаратным обеспечением. На этом этапе разрабатывается архитектура программной части будущей системы и производится распределение задач по возможным направлениям реализации. Поскольку в рамках данного раздела нас преимущественно интересуют вопросы, связанные с разработкой аппаратных фрагментов системы, то содержание и порядок разработки программного обеспечения подробно здесь рассматривать не будем. Для определенности будем считать, что для размещения ПО требуется не менее 16 Кбайт памяти. Стандартное для всех МП-систем оборудование (такое, как последовательный канал, таймеры), обычно требуемое и используемое при разработке и изготовлении реальных приборов, для упрощения примера из рассмотрения исключены.

Будем считать, что требования к системе с указанием распределения функций между SW и HW могут соответствовать рис. 4.12. Необходимость аппаратной реализации сравнения содержимого регистра уставок с данными каналов B и C, так же как аппаратная поддержка режима записи данных в блок памяти (RAM) и управление работой аналого-цифрового преобразователя (ADC) связаны со скоростными требованиями к обработке соответствующих данных. На рисунке требуемые блоки управления имеют названия `Cnt_Cmp`, `Cnt_ADC` и будут использоваться в дальнейшем описании проекта.

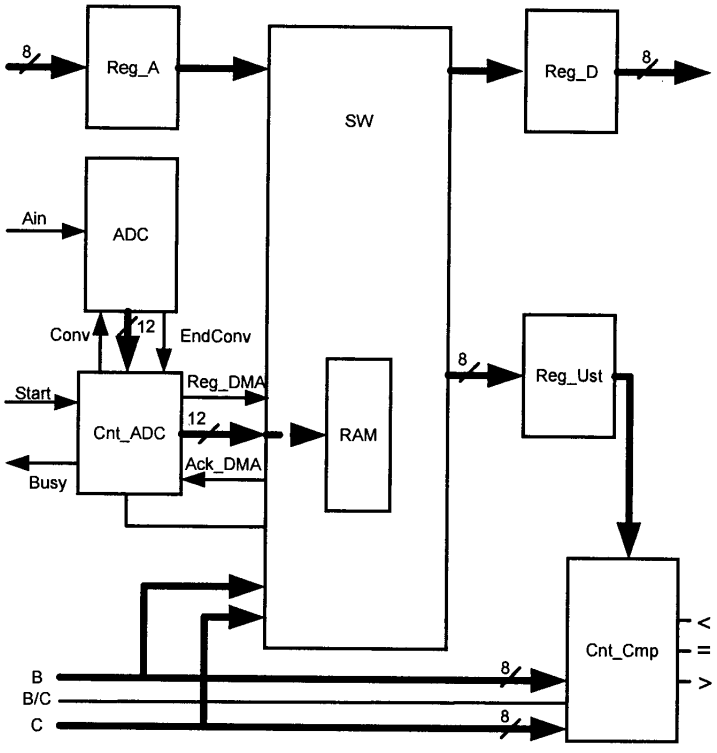


Рис. 4.12. Функциональная схема аппаратно реализуемых фрагментов проекта

Для этого этапа проектирования характерно (как минимум, предварительное) согласование внешних и внутренних (между SW и HW) интерфейсных функций разрабатываемой системы. В нашем примере сигналом, инициирующим запись блока данных (128 слов) в память, является сигнал `Start`,

а квитирующим сигналом, определяющим допустимость начала нового цикла записи, является нулевое значение сигнала `Busy`. Запись в память должна производиться в последовательности: старшие 8 разрядов результата преобразования, младшие 4 разряда того же результата. О завершении записи блока данных программное обеспечение МП-ядра должно быть проинформировано выставлением сигнала прерывания `Int`.

4.4.1. Рассмотрение технического задания на разрабатываемое устройство и выбор элементной базы

Реализация проекта возможна в различных вариантах. В соответствии с принятым соглашением анализироваться будут варианты, имеющие в качестве МП-ядра тот или иной вариант контроллера MCS-51.

К основным можно отнести следующие варианты реализации:

- использование обычного контроллера семейства MCS-51, автономных средств аналого-цифровой обработки и размещение всей дискретной части проекта в ПЛИС (для конкретности будем ориентироваться на ПЛИС фирмы Altera);
- использование БИС класса SOPC generic фирмы Altera и мегафункции фирмы CAST для реализации МП-ядра;
- использование микроконтроллера, совместимого по системе команд с MCS-51 со встроенными средствами аналого-цифровой обработки (для определенности будем ориентироваться на применение БИС ADuC812 фирмы Analog Devices), и реализация недостающих дискретных элементов в ПЛИС фирмы Altera;
- использование БИС класса SOPC фирмы Triscend семейства E5, содержащей в качестве МП-ядра встроенный микроконтроллер 8032 и конфигурируемую логику типа FPGA, и реализация внешнего аналого-цифрового преобразования (для определенности протокола будем ориентироваться на применение БИС AD7892 фирмы Analog Devices).

Рассмотрим эти варианты более подробно.

Первый вариант реализации (рис. 4.13) предполагает использование в качестве МП-ядра классической микросхемы MCS-51 (одну из БИС фирмы Atmel) и применение БИС ПЛ фирмы Altera для реализации недостающих дискретных компонентов. Необходимый объем памяти команд требует использования БИС семейства AT87F55 или AT89C55 либо применения внешней памяти команд (EEPROM Insruction), что в свою очередь предполагает включение в состав ПЛИС специального регистра, защелкивающего старшие разряды адреса, — `Reg_BD`. Требуемые объем и скорость заполнения буферной памяти заставляют при выборе типа ПЛИС ориентироваться на се-

мейства, содержащие встроенные блоки ОЗУ — т. е. типа FLEX10K. Последнее заставляет включить в состав устройства специальное загрузочное ПЗУ (память конфигурации ПЛИС). Суммарные затраты на комплектацию устройства составят сумму в несколько десятков долларов. Конструктивно (как видно из рисунка) система должна содержать 5 или 6 корпусов ИС.

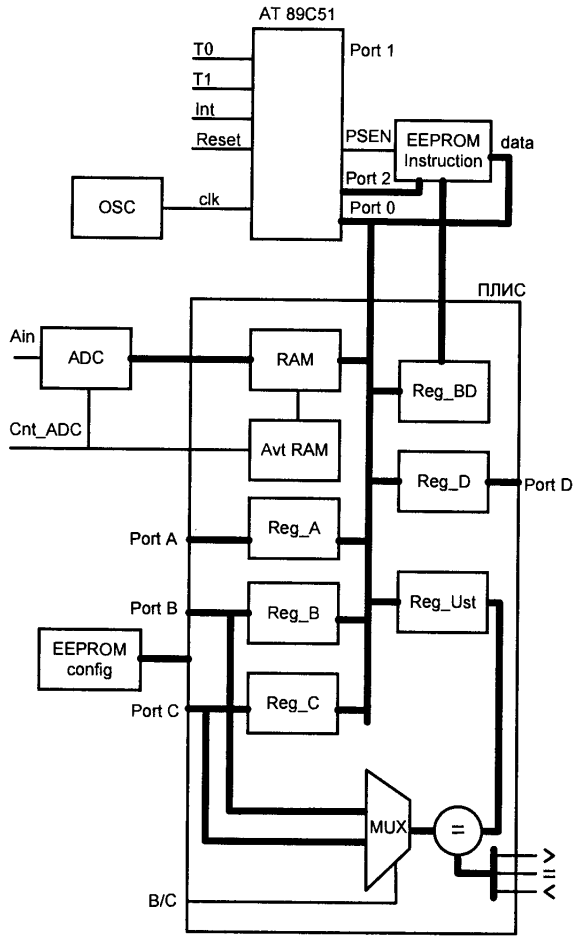


Рис. 4.13. Вариант реализации проекта на БИС МК фирмы Atmel и БИС ПЛ фирмы Altera

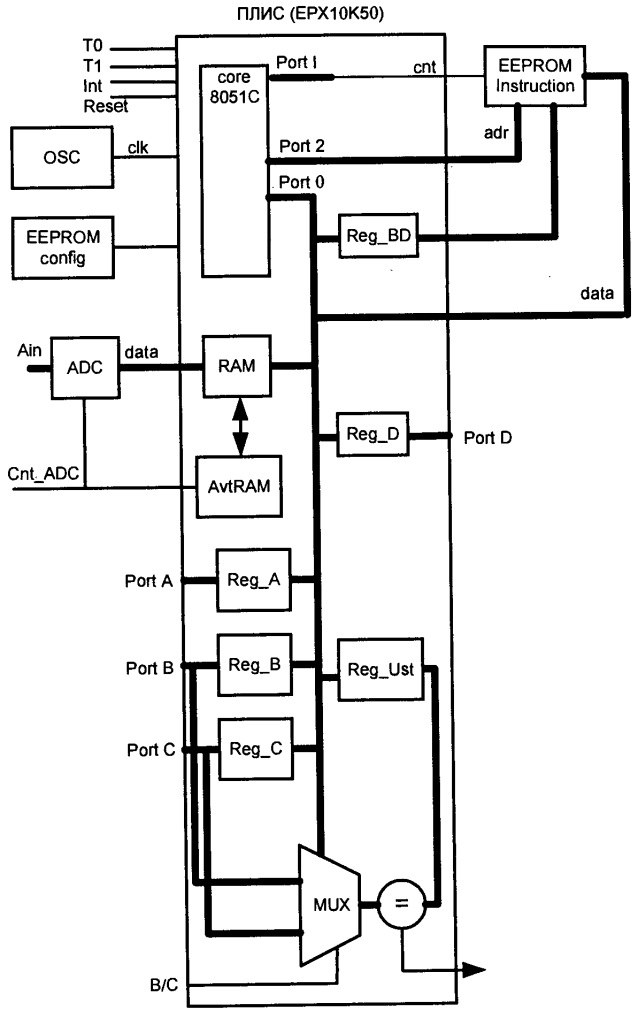


Рис. 4.14. Вариант реализации проекта на БИС ПЛ класса SOPC generic

Второй вариант реализации имеет укрупненную функциональную схему, соответствующую приведенной на рис. 4.14. Основу проекта составляет ПЛИС класса SOPC generic, конфигурация которой включает МП-ядро (на

базе стандартной мегафункции) и требуемую дополнительную логику: выходной регистр Reg_D, регистр формирования адреса памяти команд Reg_BD, входные регистры Reg_A, Reg_B и Reg_C, буфурное ОЗУ (RAM) и автоматы, управляющие работой ADC и ОЗУ (Avt_RAM). Помимо БИС ПЛ схема содержит дополнительные элементы: ИС времязадающего генератора (OSC), ИС ПЗУ конфигурации (EEPROM config), ИС ПЗУ команд (EEPROM Instuction), ИС аналогового коммутатора (MUX) и ИС АЦ преобразователя (ADC). Поскольку мегафункция микроконтроллера в этом варианте реализации требует (в зависимости от скоростных требований) от 2400 до 2860 логических ячеек, то для воплощения проекта понадобится БИС ПЛ класса не ниже 10K50. Стоимость ее в варианте семейства FLEX10K превысит 200 долларов, поэтому ориентация на такой проект будет экономически неоправданной при выпуске даже опытной партии устройств. Реализация этого варианта может иметь смысл только для ускорения проектных работ и использоваться для прототипизации проекта. Перевод проекта на более современные типы ПЛИС (например, схемы класса ACEX) существенно улучшает экономические показатели (стоимость БИС не превысит 30 долларов), однако это приводит к необходимости построения системы с различными (5 и 3,3) уровнями питающих напряжений.

Третий вариант реализации (рис. 4.15) является самым экономичным по числу требуемых для реализации системы числа микросхем. Основу схемы образуют две БИС (БИС ADuC812 фирмы Analog Devices и БИС ПЛ фирмы Altera). БИС ADuC812 фирмы Analog Devices разработчики отнесли к классу микропреобразователей (MicroConverter), поскольку она содержит на одном кристалле микроконтроллер, память, АЦП и ЦАП. Для реализации устройства дополнительно потребуется одна БИС конфигурационного ПЗУ (EPROM) и одна ИС времязадающего генератора (OSC). Структура устройства, конфигурируемого в БИС ПЛ, сохраняет элементы предыдущих вариантов. Стоимость БИС класса ADuC812 фирмы Analog Devices не превышает 15 долларов, но и стоимость БИС EPX10K10 фирмы Altera чуть больше 10 долларов. Приобретение средств, сопровождающих разработку, может потребовать затрат порядка 100 долларов.

Последний вариант (рис. 4.16), опирающийся на продукцию фирмы Triscend, потребует для своей реализации, помимо БИС семейства E5, четырех схем СИС (организация БИС семейства E5 позволяет объединить в одной БИС ПЗУ память команд МК и память конфигурации БИС ПЛ — EEPROM). Блоки, реализуемые конфигурируемой системной логикой (CSL) кристалла E5, функционально совпадают с блоками, размещенными в БИС ПЛ предыдущего варианта. Стоимость БИС семейства в зависимости от тактовой частоты и количества конфигурируемых ячеек системной логики может колебаться от 10 до 100 долларов. Сформулированные требования повлекут за собой использование ИС с логическими ресурсами, характерными для БИС типа TE5002. При выборе ИС аналого-цифрового преобразователя

для определенности остановимся, например, на продукции фирмы Analog Devices — схеме AD7892 (12-разрядном АЦП с возможностью приема информации в параллельной или последовательной формах).

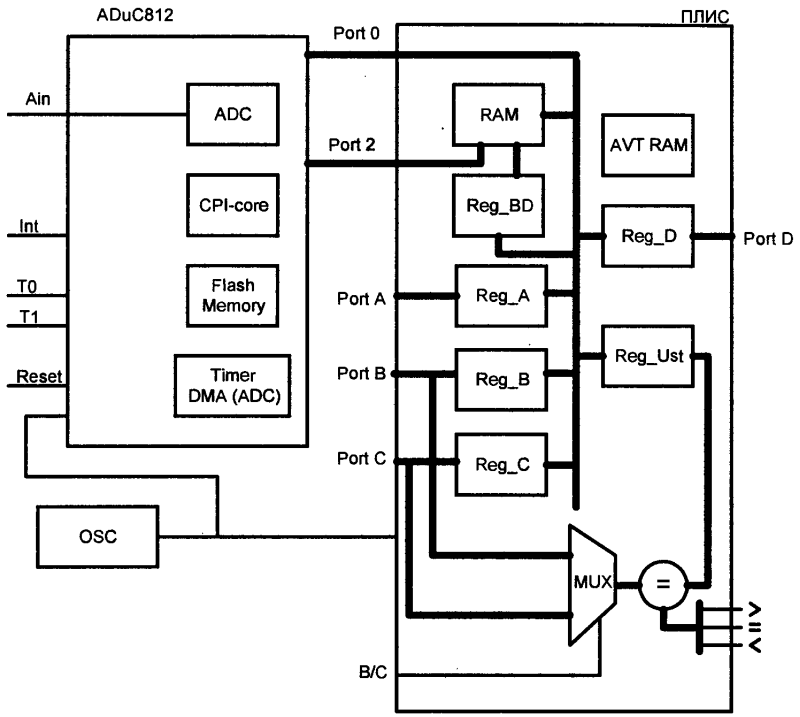


Рис. 4.15. Вариант реализации проекта на БИС ADuC812 и БИС ПЛ EPX10K10

С экономической точки зрения основные затраты необходимо разделять на затраты, необходимые для выпуска каждого экземпляра конечной продукции, и на расходы, связанные с начальными вложениями в проект. В последних, по-видимому, основную долю будут составлять затраты, связанные с приобретением лицензии на требуемые САПР и закупкой загрузочного оборудования. Общие затраты типа NRE будут максимальными для трех последних вариантов и могут достигать сумм порядка 170 долларов. Затраты, связанные с выпуском отдельных экземпляров конечной продукции, складываются из стоимости комплектующих и стоимости изготовления. Хотя стоимость комплектующих элементов будет минимальной для первого ва-

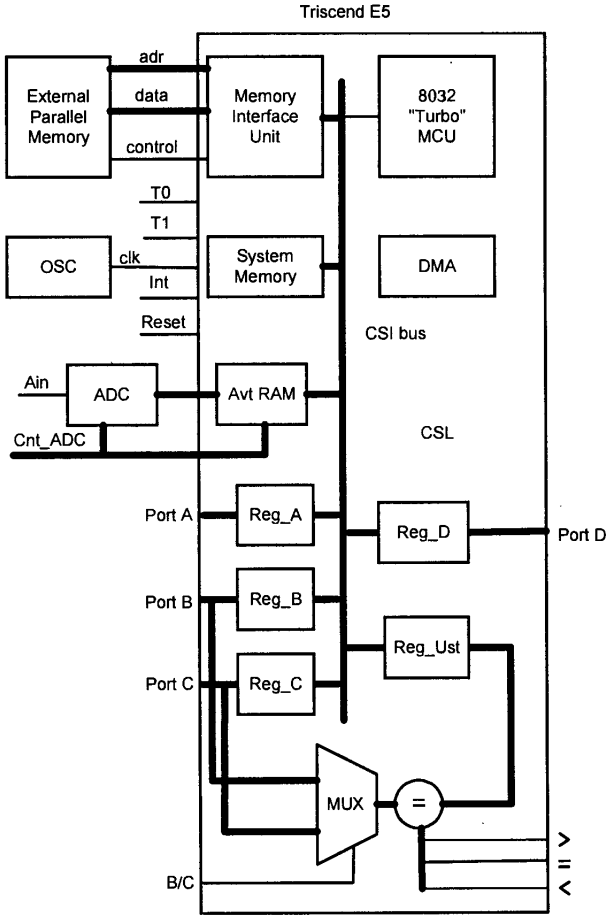


Рис. 4.16. Вариант реализации проекта на БИС ПЛ класса SOPC фирмы Triscend E5

рианта, стоимость изготовления и отладки аппаратуры для него в условиях отечественного рынка и состоянии производства будет сравнима с затратами для других вариантов. Основным достоинством последнего варианта, помимо минимизации числа корпусов ИС, является высокая надежность продукции, простота отладки опытной партии и легкость контроля конечной продукции при серийном изготовлении. Именно это и позволяет на нем остановиться. Дополнительными факторами, определяющими целесообразность реализации проекта на кристаллах типа SOPC (в примере — БИС фирмы Triscend), является возможность совершенствования и модернизации проекта, включая полное изменение аппаратной и программной начинки кристалла (т. е., практически, реализацию других проектов), без каких-либо конструктивных изменений.

Процедура декомпозиции проекта

После того как принято решение о базовых элементах проекта, разрабатывается предполагаемая архитектура аппаратной части будущей системы и производится распределение ресурсов по трем возможным направлениям реализации. Отдельные фрагменты проекта при ориентации на кристаллы класса SOPC могут строиться, используя возможности, предоставляемые:

- предопределенными ресурсами МП-ядра;
- ресурсами системной логики кристалла;
- ресурсами интегральных схем, внешних относительно кристалла.

Следует подчеркнуть, что идеология реализации системы на кристалле приводит к отличиям таких систем от систем в многокристальном исполнении. Первое отличие состоит в существенном увеличении скорости реализации отдельных команд (это связано, в частности, с тем, что элементы системы находятся на одном кристалле). Второе отличие состоит во введении дополнительных архитектурных элементов структуры (отсутствующих при классической реализации той же структуры). Такими элементами для кристаллов типа E5 являются контроллеры ПДП и сторожевой таймер.

После выбора способов реализации двух частей устройства можно переходить к детализации технического задания на проектирование SW и HW БИС SOPC.

Выбор САПР

Важное значение для быстрейшего и успешного завершения проекта имеет принятие на этом этапе обоснованного решения об привлекаемых средствах проектирования. На последних этапах проектирования (этапах монтирования проекта в БИС и подготовки конфигурационного файла) практически всегда приходится обращаться к базовой САПР фирмы-производителя кристалла SOPC. Для выбранного типа БИС SOPC семейства E5 фирмы Triscend необходимо ориентироваться на САПР под названием FastChip. Допустимые для проектирования в САПР FastChip направления проектного потока приведены на рис. 4.17.

Существование альтернативных вариантов выполнения проектного потока связано с возможностью или необходимостью привлечения на определенных ветвях проектирования САПР сторонних фирм. Поскольку в состав

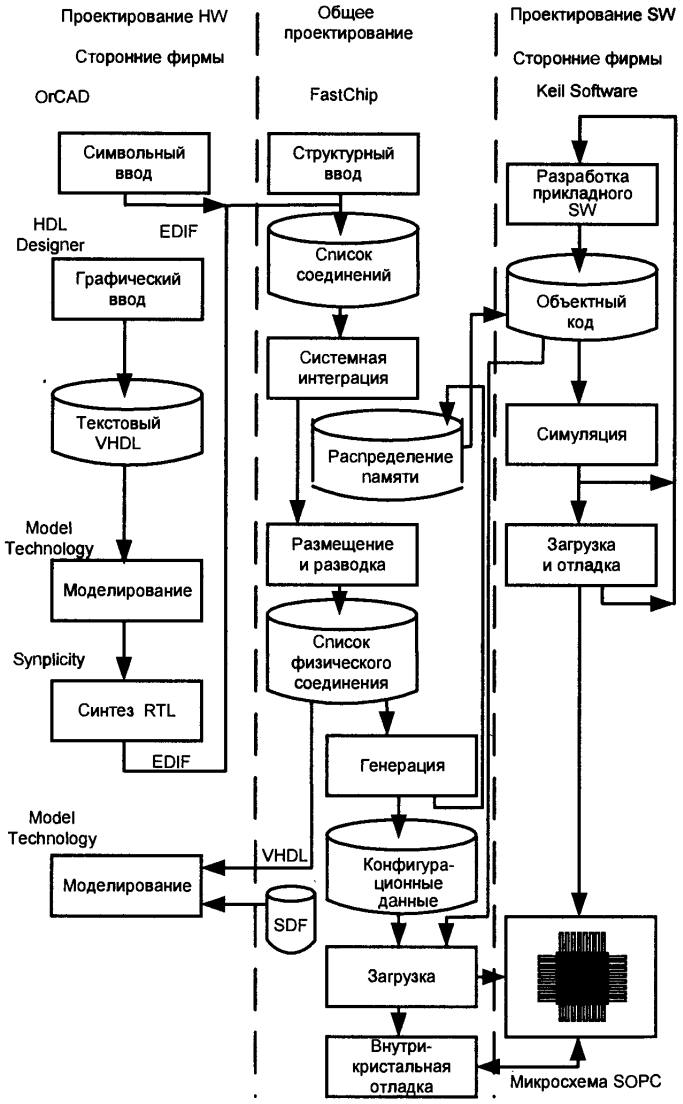


Рис. 4.17. Проектный поток в FastChip

FastChip не включены компиляторы для МК типа 8051, то подготовка программной части проекта обязательно требует привлечения каких-либо систем проектирования, ориентированных на разработку программного обеспечения для MCS-51. Формально, проектирование может выполняться на любой системе проектирования, но наилучшая стыковка с FastChip выполняется для небольшого числа систем проектирования. Наилучшее и гарантированное согласование обеспечивается при использовании инструментальных средств разработки фирмы Keil Software (**www.keil.com**) для семейства микроконтроллеров 8051. Средства проектирования фирмы Keil поддерживают все стадии разработки программного обеспечения. Программные средства включают средства компиляции с языков ассемблера и C, интегрированную среду разработки, содержащую отладчик и моделировщик. Для разработки однопроцессорных многозадачных систем могут привлекаться средства операционной системы реального времени RTX 51, входящей в состав САПР Keil Software. Поскольку вопросы разработки программного обеспечения микропроцессорных систем выходят за рамки данного раздела, далее будут затрагиваться только те аспекты разработки программного обеспечения, которые тесно связаны с реализацией аппаратных частей системы.

Реализацию аппаратной части проекта принципиально можно выполнить, опираясь только на возможности САПР FastChip Triscend. Однако трудно реализовать сложные проекты, оставаясь в рамках библиотеки стандартных решений, доступных из редактора САПР FastChip. Конечно, любой проект может быть представлен в форме соединенных между собой примитивов, которые входят в состав библиотеки САПР. Но принятый в редакторе FastChip способ объявления соединений блоков путем использования совпадающих имен для соединяемых входных и выходных контактов блоков оказывается не всегда удобным. Отсутствие отображения этих соединений на экране так же далеко не всегда приемлемо. Этот вариант соответствует одноуровневой организации аппаратной части системы. При составлении сложных проектов и для их документирования целесообразно привлечение САПР других фирм, которые могут поддержать иерархическое описание и построение проекта. Практическая реализация такой возможности требует выполнения ряда условий.

- Во-первых, в САПР FastChip предусмотрен импорт описания блоков, созданных в формате языка EDIF.

  В настоящий момент библиотеки примитивов фирмы включены в состав библиотек таких крупнейших фирм, как Innoveda и Cadence. Поэтому для графического ввода проектов (или их фрагментов) могут использоваться редакторы графического ввода ViewDraw фирмы Innoveda или OrCAD Capture фирмы Cadence.

  В структуру синтезирующих компиляторов фирм Synopsys или Synplicity включено представление о структуре строительных элементов FPGA фирмы Triscend, а также о правилах и возможностях их соединения меж-

ду собой. Поэтому для синтеза проектов, ориентированных на реализацию в SOPC E5, могут использоваться компиляторы FPGA Express или FPGA Compiler II фирмы Synopsys или компилятор Synplify фирмы Synplicity. Достоинство использования компиляторов этих фирм заключается в возможности составления проектов на языках высокого уровня VHDL и Verilog.

- Во-вторых, САПР FastChip не содержит встроенных средств для верификации проектов. Однако поскольку САПР FastChip позволяет экспортировать созданный проект в форме описания на языках VHDL и Verilog, то для верификации проектов могут использоваться моделировщики, ориентированные на эти языки. В первую очередь, это хорошо зарекомендовавшие себя моделировщики VHDL- и Verilog-описаний фирмы Model Technology, хотя, конечно, могут использоваться и другие моделировщики, например, VCS и VSS фирмы Synopsys, VerilogXL фирмы Cadence, SpeedWave VHDL Analyzer фирмы Innoveda и др.

Ускорения проектирования можно достичь, ориентируясь на использование отладочных плат для выбранного семейства БИС (в нашем случае E5). Подобные платы выпускают фирмы Triscend и XESS. Применение отладочных плат целесообразно не только на заключительных стадиях проектирования в качестве прототипа будущего проекта, но и на начальных этапах для ознакомления с особенностями проектирования данного типа SOPC.

4.4.2. Разработка аппаратной части БИС SOPC

Этап 1. ТЗ на проектирование аппаратной части БИС

Независимо от формы представления, ТЗ на проектирование аппаратной части кристалла E5 будет очевидно содержать следующие ключевые сведения:

- объем буферного ОЗУ, выделяемого из общего объема памяти, соответствует 256 двенадцатиразрядным словам;
- запись в ОЗУ осуществляется блоками по запускающему сигналу `Start`, формируемым внешней средой;
- начало записи блока данных в ОЗУ и ее завершение сопровождается соответствующими изменениями сигнала `Busy`;
- о завершении записи МП-ядро информируется выставлением сигнала `Int`;
- аппаратура SOPC управляет ADC, выставляя сигнал `nConvStr`;
- сигнал `nEOC` выставляется ADC после завершения процедуры преобразования;
- чтение данных из ADC осуществляется аппаратурой SOPC путем выставления сигналов `nCS` и `nRD` в сторону ADC;

- аппаратура SOPC обеспечивает передачу двенадцатиразрядных данных в ОЗУ МП побайтно в два приема, формируя и анализируя необходимые сигналы управления контроллера прямого доступа (DMA);
- запись очередных данных в ОЗУ обеспечивается передачей сначала восьми старших, а затем четырех младших битов данных.

Ресурсы CSL БИС E5 использованы для реализации функции трех входных восьмиразрядных портов (`Reg_A`, `Reg_B` и `Reg_C`) и одного выходного восьмиразрядного порта (`Reg_D`).

Этап 2. Разработка общей структуры аппаратной части проекта

Перечисленные выше пункты ТЗ определяют основные блоки проектируемой системы БИС и их взаимодействие. Блочная схема устройства приведена на рис. 4.18. На рисунке опущены соединения БИС SOPC с загрузочным ПЗУ и схемой тактового генератора. Элементы структуры должны опираться на ресурсы программируемой логики кристалла E5 (CSL-логики). Функциональное назначение блоков следует из их названий. Схема укрупненно отображает следующие процессы.

- Блок `Cnt_ADC` отвечает за управление отдельными циклами преобразования и запись одиночных данных от аналого-цифрового преобразователя в промежуточном регистре. Блок обеспечивает прием блока данных заданной длины и формирует сигналы, требуемые для записи принимаемых от АЦП данных в ОЗУ МП.
- Блок `Cnt_Cmp` обеспечивает процедуру сравнения сигналов, поступающих в порты МП `A` и `B`, и формирует необходимые внешние сигналы управления "больше", "меньше" или "равно". Выбор рабочего канала определяется входным сигналом `B/C`.

Следующим шагом проектировщика является обоснованный выбор средств, при помощи которых будут проектироваться отдельные фрагменты проекта. Как правило, специфические характеристики, свойственные проектируемым фрагментам, позволяют определить рациональность применения того или иного проектного средства. Для проекта, выбранного в качестве примера, в аппаратной части E5 можно выбрать ряд блоков, реализация которых целесообразна с привлечением различных средств.

- Для реализации блока управления приемом информации от аналого-цифрового преобразователя (блок `Cnt_ADC`) будем ориентироваться на возможности, предоставляемые компилятором Synplify фирмы Synplicity. Окончательным результатом работы компилятора при этом будет импортируемый модуль на языке EDIF. Поскольку в качестве входной информации для Synplify должен использоваться текстовый файл на языке VHDL, то для подготовки текстового описания этого блока будут приме-

няться программные пакеты фирмы Mentor Graphic. Представляется следующая последовательность работ: графический ввод и составление тестовой процедуры Test-Bench с использованием САПР HDL Designer, а затем верификация полученного VHDL-описания с помощью САПР Model Technology.

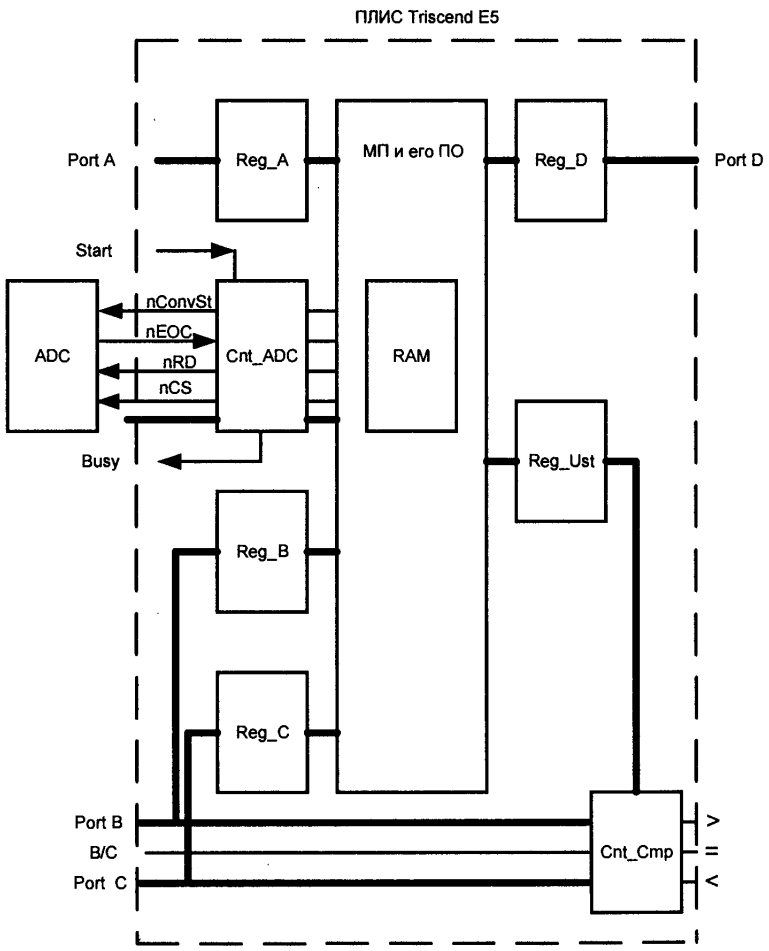


Рис. 4.18. Блок-схема устройства, реализуемого программируемой CSL-логикой кристалла E5

- Для реализации блока сравнения двух восьмиразрядных кодов с регистром уставок (блок `Cnt_Cmp`) будем ориентироваться на возможности, предоставляемые графическим редактором САПР OrCAD. Как и для предыдущего фрагмента, окончательным результатом при этом будет импортируемый модуль на языке EDIF.
- Для реализации проекта на верхнем уровне иерархии будем ориентироваться на возможности, предоставляемые структурным редактором САПР FastChip. На этом уровне обеспечивается подключение импортируемых описаний блоков `Cnt_ADC` и `Cnt_Cmp` как к внешним выводам кристалла, так и к внутренней шине встроенного МП. Кроме того, на этом уровне обеспечивается подключение к шине МП портов ввода/вывода цифровой информации: выходного регистра `Reg_D`, входного регистра `Reg_C`, входных портов `Reg_A` и `Reg_B` и регистра уставок `Reg_Ust`.

В соответствии с принятым распределением функций проектируемого устройства последовательно рассмотрим проектирование блоков `Cnt_ADC`, `Cnt_Cmp` и общей схемы.

Этап 3. Проектирование основных компонентов блока *Cnt_ADC* — блока управления записью данных от АЦП

Функционально блок `Cnt_ADC` решает следующую задачу — обеспечивает управление процедурой получения блока данных от АЦП и передачу ее в память МП, опираясь на ресурсы контроллера ПДП, встроенного в микропроцессорную часть кристалла. Решение этой основной задачи сопровождается решением двух вложенных подзадач:

- формированием необходимой последовательности сигналов, управляющих работой АЦП, для получения 12-разрядного кода результата с обеспечением его временного сохранения;
- формированием необходимой последовательности сигналов, управляющих передачей сохраненных данных в память МП.

Общая структура блока `Cnt_ADC` приведена на рис. 4.19 (для удобства рассмотрения дополнительно на рисунке изображен и внешний аналого-цифровой преобразователь `ADC`). Ключевым элементом схемы является автомат управления `Avt_ADC`. Операционная часть фрагмента содержит два регистра `Reg_Buf1` и `Reg_Buf2` для промежуточного хранения 8- и 4-разрядных частей данных от АЦП, мультиплексор `MUX`, счетчик `Count`, отвечающий за длину принимаемого блока данных.

Триггеры `T1` и `T2` введены в схему устройства для синхронизации сигналов асинхронно работающих блоков — АЦП и автомата. Функциональное назначение управляющих сигналов очевидно. Сигналы, квитирующие обмен

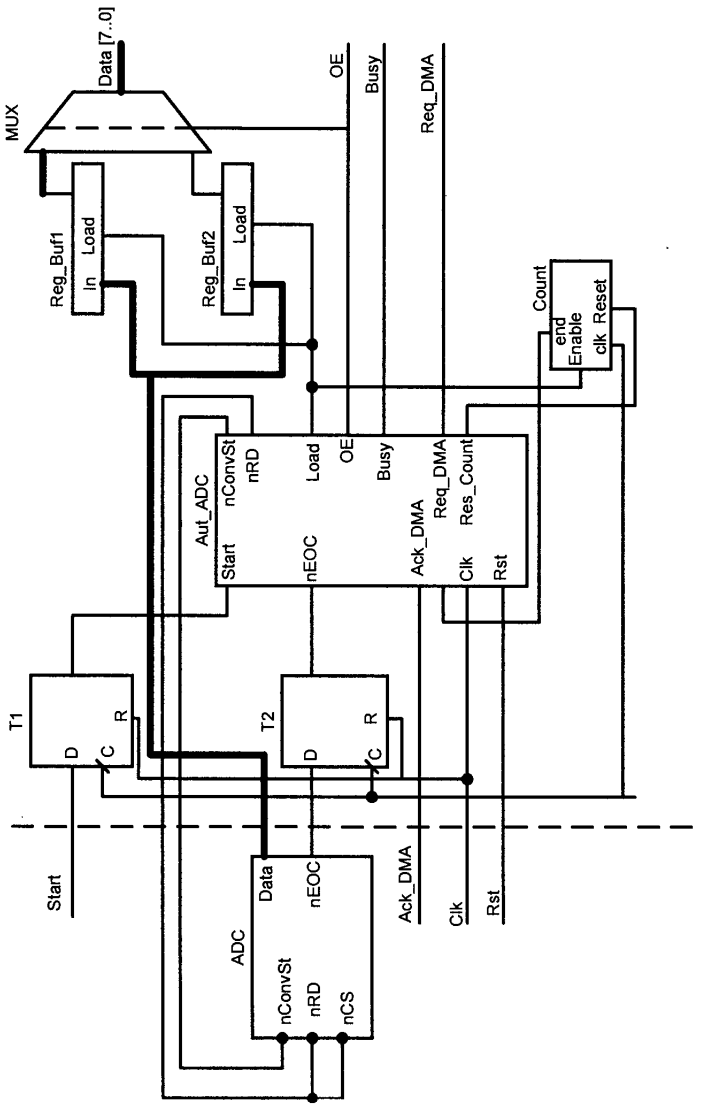


Рис. 4.19. Структура блока Cnt_ADC

блока данных, — Start, Busy. Сигналы, квитирующие управление одиночным циклом преобразования АЦП, — nConvSt, nEOC. Сигналы, обеспечивающие передачу данных от АЦП, — nCS, nRD и сигнал Load, обеспечивающий запись данных Data в регистры промежуточного хранения Reg_Buf1 и Reg_Buf2. Сигнал Load используется одновременно для наращивания значения счетчиком Count числа принятых слов. Начальный сброс счетчика осуществляется сигналом Res_Count. Состояние сигнала End_Count указывает заполнение буфера памяти.

Разбиение передачи 12-разрядных кодов на две посылки потребовало разбиения регистра на две части (8 и 4 разряда) и введения сигнала OE, управляющего съемом информации из этих частей. Мультиплексор MUX обеспечивает подключение выхода выбранного промежуточного регистра (Reg_Buf) к МП для передачи данных в память. Для одиночной передачи данных по каналу прямого доступа к памяти (DMA) служит система квитирующих сигналов Req_DMA и Ack_DMA.

Автомат синхронный — тактовый сигнал Clk, сигнал асинхронного сброса — Rst.

Проектирование управляющего автомата *Avt_ADC*

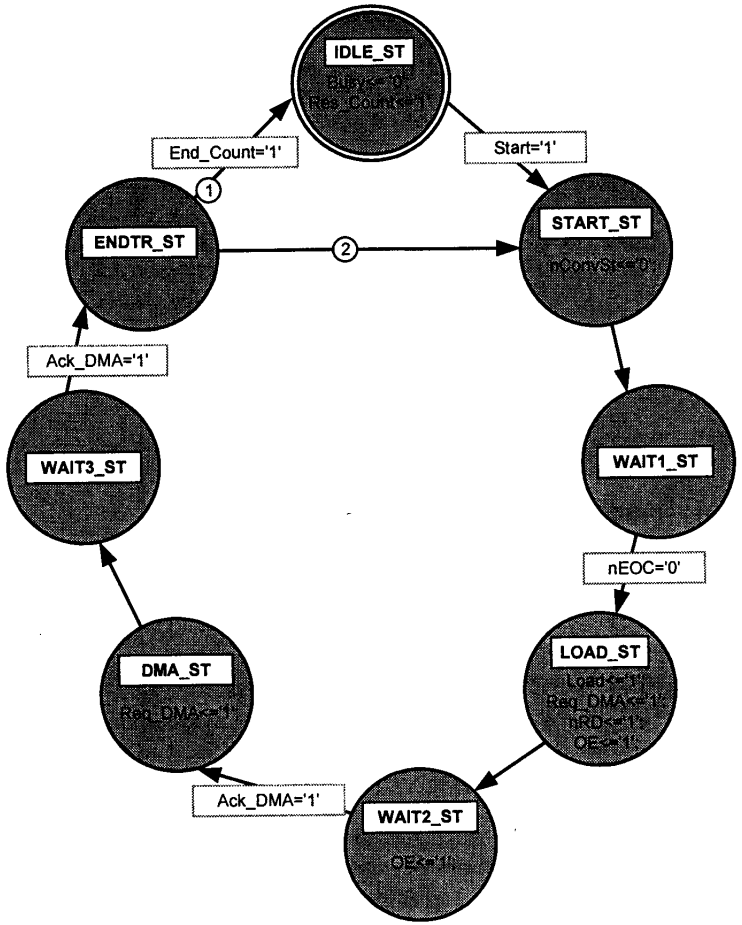
Возможный алгоритм работы автомата Avt_ADC блока Cnt_ADC, отвечающий протоколу, необходимому для правильной работы БИС AD7892 фирмы Analog Devices, может иметь вид, соответствующий схеме переходов автомата, приведенной на рис. 4.20. Схема переходов при помощи графического редактора пакета HDL Designer Series фирмы Mentor Graphics была занесена в соответствующий диаграммный файл.

Перейдем к описанию поведения автомата Avt_ADC, управляющего считыванием данных из АЦП во временный регистр Reg_Buf и записью этих данных в память МП, и поддерживающего для этих обменов требуемое взаимодействие квитирующих сигналов.

Основу алгоритма образует циклическая последовательность смены состояний, определяющая выполнение 128 циклов обрабатываемого блока.

Исходно (по сигналу Rst и по завершении цикла записи блока данных) автомат находится в состоянии IDLE_ST. В этом состоянии автомат формирует сигнал Busy, сигнализирующий о готовности к приему блока данных от АЦП, и сигнал Res_Count, обнуляющий счетчик числа циклов записи Count, и продолжает находиться в состоянии IDLE_ST до появления сигнала Start, который приводит к переходу автомата в последующее состояние START_ST. В этом состоянии формируется сигнал запуска процедуры преобразования: сигнал nConvSt устанавливается в 0.

В состоянии WAIT1_ST автомат остается до тех пор, пока не поступит сигнал готовности данных от ADC — сигнал nEOC не станет равным нулю.



Signals Status

SIGNAL	SCOPE	DEFAULT	RESET	STATUS
Busy	OUT	'0'		Comb
nConvSt	OUT	'1'		Comb
Load	OUT	'0'		Comb
OE	OUT	'0'		Comb
Req_DMA	OUT	'0'		Comb
nRD	OUT	'0'		Comb
Res_Count	OUT	'0'		Comb

Рис. 4.20. Граф-схема переходов автомата управления Avt_ADC

В состоянии LOAD_ST формируются сигнал квитирования nRD для АЦП, сигнал запуска записи данных Load от ADC в промежуточный регистр и сигнал запроса DMA_Req к контроллеру ПДП. Одновременно с запросом подготавливается передача в МП старшего байта данных из регистра, для чего формируется сигнал OE.

В состоянии WAIT2_ST формируется сигнал OE. В этом состоянии автомат остается до тех пор, пока не появится сигнал Ack_DMA, подтверждающий готовность МП к приему данных по каналу ПДП. По этому сигналу выходные данные Reg_Buf1, находящиеся на выходе мультиплексора, переписываются в ОЗУ.

В состоянии DMA_ST формируются новый сигнал запроса Req_DMA к контроллеру ПДП и подготавливается передача в МП младших 4-х разрядов данных из регистра Reg_Buf2, для чего снимается сигнал OE.

Автомат безусловно переходит из состояния DMA_ST в состояние WAIT3_ST. В этом состоянии он остается до тех пор, пока не появится сигнал Ack_DMA, подтверждающий готовность МП к приему данных по каналу ПДП.

В состоянии ENDTR_ST анализируется сигнал End_Count. Если сигнал End_Count равен 1, что соответствует завершающему 128 циклу приема оцифрованных данных от АЦП, то автомат переводится в начальное состояние IDLE_ST, в противном случае автомат переходит в состояние START_ST, запускающее новый цикл преобразования.

Пояснения к синтаксису VHDL-программы устройства управления

Для автомата нашего примера с помощью программы пакета HDL Designer Series фирмы Mentor Graphics была выполнена трансляция диаграммы. Получен вариант, ориентированный на возможности языка высокого уровня описания аппаратуры VHDL. Фрагмент кода, соответствующий заданному автомату, приведен (с несущественными изменениями и сокращениями относительно оригинала) в листинге 4.10.

Листинг 4.10

```
-- hds header_start
-- hds header_end
LIBRARY ieee;
USE ieee.std_logic_1164.all;
USE ieee.std_logic_ariht.all;

ENTITY Avt_ADC IS
    PORT(Clk, Start, Rst, nEOC, Ack_DMA, End_Count : in std_logic;
nConvSt, Busy, Load, Req_DMA, OE, nRD, Res_Count: out std_logic);
END Avt_ADC;
```

```
-- hds interface_end
LIBRARY ieee;
USE ieee.std_logic_1164.all;
USE ieee.std_logic_ariht.all;

ARCHITECTURE fsm of Avt_ADC IS
    -- Architecture Declarations
    TYPE state_values IS (IDLE_ST, START_ST, WAIT1_ST, LOAD_ST,
WAIT2_ST, DMA_ST, WAIT3_ST, ENDTR_ST,);
    SIGNAL current_state, next_state: state_values;
BEGIN

cloced:  process (Clk, Rst)
 BEGIN
        IF Rst = '1' THEN
          current_state <= IDLE_ST;
             -- Reset Values
        ELSIF rising_edge(Clk) then
            current_state <= next_state;
            -- Default Assignment To Internals
        END IF;
 END PROCESS cloced;

NEXTSTATE: PROCESS (current_state, Start, nEOC, Ack_DMA, End_Count)
    BEGIN
        -- Defaults Assignment
        nConvSt <= '1';
        Busy <= '1';
        Load <= '0';
        Req_DMA <= '0';
        OE <= '0';
        nRD <= '0';
        Res_Count <= '0';
        -- Combined Actions
        CASE current_state IS
            WHEN IDLE_ST =>
                Busy <= '0';
                Res_count <= '1';
                IF Start = '1' THEN
                    next_state <= START_ST;
                END IF;
            WHEN START_ST =>
                nConvSt <= '0';
                next_state <= WAIT1_ST;
            WHEN WAIT1_ST =>
```



```
                IF nEOC = '0' THEN
                    next_state <= LOAD_ST;
                END IF;
            WHEN LOAD_ST =>
                Load <= '1';
                nRD <= '0';
                Req_DMA <= '1';
                OE <= '1';
                next_state <= WAIT2_ST;
            WHEN WAIT2_ST =>
                OE <= '1';
                IF Ack_DMA = '1' THEN
                    next_state <= DMA_ST;
                END IF;
            WHEN DMA_ST =>
                Req_DMA <= '1';
                next_state <= WAIT3_ST;
            WHEN WAIT3_ST =>
                IF Ack_DMA = '1' THEN
                    next_state <= ENDTR_ST;
                END IF;
            WHEN ENDTR_ST =>
                IF End_Count = '1' THEN
                    next_state <= IDLE_ST;
                ELSE
                    next_state <= START_ST;
                END IF;
            WHEN OTHERS =>
                    next_state <= IDLE_ST;
            END CASE;
        END PROCESS nextstate;
        -- Concurrent Statements
END fsm;
```

Разделы проектного модуля типичны для языка VHDL. В самом начале указывается используемая в проекте библиотека (`IEEE`). В заголовочном разделе `ENTITY` перечислены имена и типы всех сигналов: входных внешних управляющих — тактового `Clk`, начала работы с блоком `Start`, начального сброса `Rst`, окончания блока данных `End_Count`, запроса на запись байта данных `Ack_DMA`, флага готовности АЦП `nEOC` и выходных управляющих — запуска АЦП `nConvSt`, квитирования АЦП `nRD` и `OE`, управления счетчиком адреса `ResAdr`, управления режимом записи буферного регистра `Load`, запроса ПДП `Req_DMA` и управления внешним выходным сигналом `Busy`.

Следующий раздел — `ARCHITECTURE` — представляет собой описание архитектуры или поведения (в нашем случае поведения) блока, интерфейс которого

был описан в ENTITY. Как и в обычных языках, в начале раздела дается описание типов и объявление переменных, используемых при описании действий, выполняемых в разделе ARCHITECTURE. В данном автомате определен перечислимый тип данных STATE_VALUES со всем списком допустимых значений (они, естественно, совпадают с именами, введенными в граф-схеме переходов). Далее в тексте объявлены два сигнала (Signal): current_state и next_state введенного типа STATE_VALUES. Введение двух сигналов связано с необходимостью определения текущего и следующего состояний автомата при переходе от одного состояния к другому.

Главная часть архитектурного тела содержит два оператора процесса. Первый процесс по имени Clocked запускается на исполнение каждый раз, когда происходит изменение любого входного сигнала Clk или Rst. При составлении программы автомата учитывалась необходимость его установки в исходное состояние при подаче сигнала сброса: выражение IF (Rst='1') THEN current_state<=IDLE_ST;. Однако его основное действие — назначение автомату нового состояния — происходит только по переднему фронту сигнала Clk. Использование для тактирования автомата переднего фронта синхронизирующего сигнала: предложение ELSIF (rising_edge (Clk)) THEN current_state<=next_state; END IF; служит для синхронизации выбранных библиотечных операционных узлов и обеспечит стабильность входных управляющих сигналов в моменты тактирования.

Поведение управляющего автомата в тексте программы задано вторым процессом по имени nextstate. Процесс запускается каждый раз, когда изменяется состояние автомата (current_state) или изменяется какой-либо входной сигнал. Содержимое этого процесса и определяет поведение управляющего автомата.

В начале процесса выходные сигналы устанавливаются в начальные состояния, соответствующие значениям по умолчанию.

Последовательность смены состояний конечных автоматов в языке VHDL удобно описывать посредством оператора выбора CASE, используя в качестве ключа выбора варианта переменную состояния автомата в текущий момент времени (current_state). Внутри каждого варианта определяется состояние перехода и значения выходных сигналов, формируемых в соответствии с входными условиями. Состояние перехода из текущего состояния в следующее осуществляется с помощью оператора назначения переменной next_state нового значения. В тех случаях, когда переход из текущего состояния зависит от внешних сигналов, этот оператор назначения входит в состав условного оператора, логическое выражение которого совпадает с последовательностью условий, встречающихся на соответствующих путях переходов на схеме алгоритма. Аналогично определяются и выходные сигналы, вырабатываемые на переходах и задающие исполняемые в других блоках операции.

Компиляция из графической формы в текстовую осуществляется многими САПР. Уже упоминавшаяся выше программа StateCAD, например, учитывает, для компилятора какой фирмы предполагается использовать описание автомата (соответствующим образом выбирая используемые синтаксические конструкции). Аналогичные соображения должны приниматься во внимание и при ручном написании программ. Это ограничение возникает из-за того, что набор допустимых синтаксических конструкций языка для различных фирм существенно отличается от стандартного.

Тестирование программного описания автомата

Для проверки правильности составления и работоспособности автоматов необходимо проведение тестовых испытаний. Возможны разные подходы к организации тестирования. Наибольшее распространение получили следующие варианты.

- В первом случае программа САПР, создающая текстовое описание автомата (такая как, например, программа StateCAD), проверяет наличие комбинаций входных сигналов, соответствующих неопределенному направлению перехода автомата, и поэтому требует формирования полного перечня входных сигналов, однозначно определяющих направления переходов.
- Второй подход состоит в том, что используемая САПР может обеспечивать так называемый *режим анимации*. В этом режиме САПР последовательно проходит по всем состояниям автомата. Переход в очередное активное состояние осуществляется путем установки желаемых значений входных сигналов. Таким образом, в интерактивном режиме разработчик может проверить все нужные состояния, переходы и значения выходных переменных, меняя на каждом шаге значения входных переменных.

В общем случае проверка работоспособности автомата (созданного любым способом) может осуществляться на основании традиционных методов создания тестирующих программ (Test-Bench). Современные САПР упрощают этот подход, предлагая автоматическое создание тестового блока Test-Bench. Возможности автоматизации изменяются в достаточно широких пределах: от автоматического формирования только интерфейса блока Test-Bench до автоматического формирования последовательности входных сигналов, обеспечивающей перебор всех возможных состояний автомата. Для рассматриваемого примера проектирования было целесообразно ориентироваться на возможности пакета HDL Designer в части формирования тестирующих программ Test-Bench. Поэтому пакет был запущен на создание Test-Bench и автоматически сформировал ENTITY для тестирующей программы. Для заполнения архитектурного тела можно использовать любые редакторы, входящие в состав HDL Designer, например графический редактор описания потоков Flow Chart или редактор таблиц истинности Truth Table.

"Подводные камни" могут ждать проектировщиков не только при организации переходов. Особое внимание при проектировании автоматов следует уделять вопросам формирования выходных сигналов. Выходные сигналы могут назначаться как выход простой комбинационной схемы или как выход триггерной схемы, тактируемой синхросигналом. В результате каждый из вариантов оформления имеет свои особенности. Наиболее важным положительным свойством триггерных выходов является гарантированное отсутствие паразитных просечек (рисков), обусловленных ярусной организацией логических схем. В отличие от триггерных, комбинационные выходы обеспечивают изменение их состояния сразу после изменения входных сигналов (асинхронное изменение), что для ряда практических приложений может представляться существенным свойством.

Отмеченное выше положение делает целесообразным организацию дополнительных исследований, например, вариантов тестирования, основанных на моделировании поведения автомата при его воплощении в базисе логических ячеек, близких по структуре к будущей реализации. Для ячеек CSL SOPC типа E5 фирмы Triscend сходные логические ячейки реализованы, например, в ПЛИС фирмы Altera типа ACEX. Простота организации модельных экспериментов в САПР MAX+PLUS II делает допустимым и целесообразным проверку поведения автомата при реализации его модели в БИС типа ACEX. Конечно, проведение таких экспериментов потребует обработки VHDL-описания автомата, например, компилятором Synplify фирмы Synplicity и далее передачу EDIF-файла в САПР MAX+PLUS II для компиляции в БИС ACEX. Предварительная обработка VHDL-описания пакетом Synplify целесообразна не столько из соображений достижения лучших результатов компиляции, сколько для получения возможности произвести анализ скомпилированной САПР структуры на регистровом и вентильном уровнях.

Разработка общей структуры блока *Cnt_ADC*

Описание блока `Cnt_ADC` может, как и в большинстве других проектов, ориентироваться на различные возможности и средства описания, предоставляемые САПР HDL Designer. Допустим как поведенческий, так и структурный подход. Наиболее компактным и быстрым способом спецификации блока является создание его текстового описания. Для демонстрации возможностей САПР HDL Designer воспользуемся структурным редактором САПР и будем ориентироваться на использование элементов стандартных библиотек пакета. Структурная схема блока `Cnt_ADC` приведена на рис. 4.21. При ее составлении использовались стандартные элементы библиотеки HDL Designer ModuleWare:

- два одиночных D-триггера `adff`;
- четырехразрядный банк триггеров `dff`;

- восьмиразрядный банк триггеров `dff`;
- восьмиразрядный счетчик `cntr`;
- один восьмиразрядный мультиплексор `Omux2`;
- четырехразрядная константа нуля.

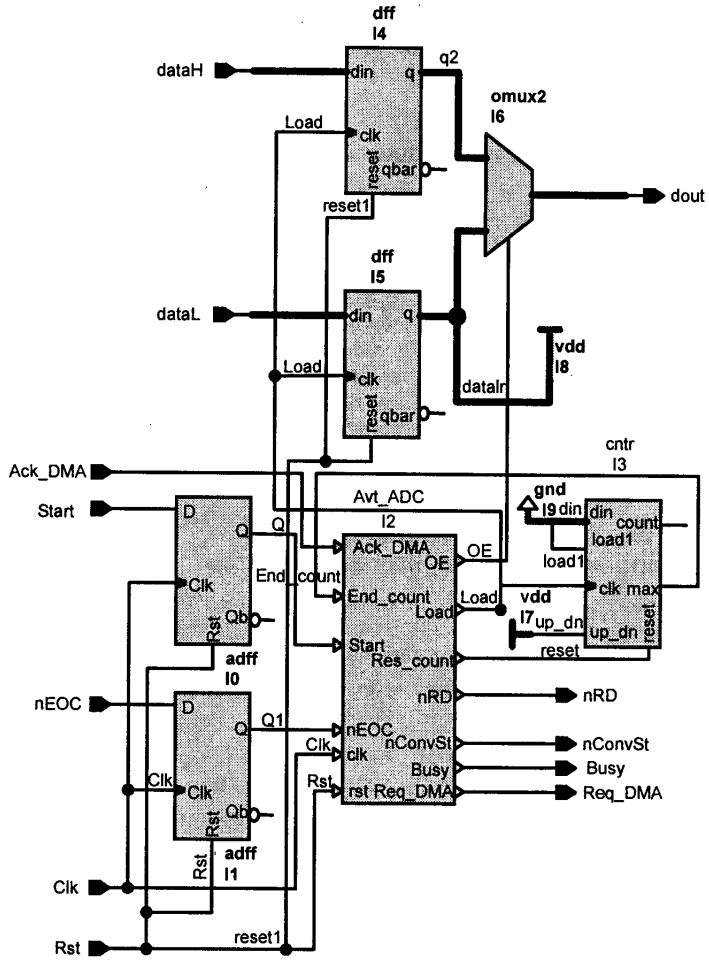


Рис. 4.21. Структурная схема блока управления `Cnt_ADC`

Необходимость подключения ко входу мультиплексора четырехразрядной константы нуля диктуется требованием выравнивания разрядностей, записываемых по каналу данных ПДП.

В состав блока включен и ранее разработанный и протестированный модуль автомата `Avt_ADC`.

На заключительном этапе структурная схема блока `Cnt_ADC` транслировалась на язык VHDL. Далее использовался компилятор Synplify фирмы Synplicity для получения EDIF-файла, который будет в дальнейшем импортироваться в САПР FastChip Triscend для создания требуемой аппаратной части разрабатываемой системы.

Этап 4. Проектирование блока *Cnt_Cmp*

Спецификация блока `Cnt_Cmp` выполнялась на базе возможностей, предоставляемых графическим редактором Capture OrCAD фирмы Cadence. Схема блока приведена на рис. 4.22. Основные элементы схемы: двухвходовые мультиплексоры `MUX2` и схема сравнения `COMP8` выбраны из библиотеки примитивов фирмы Triscend. Для этого библиотека была специально подключена к САПР OrCAD. После составления схемы требуется проверка ее правильности, для чего должны использоваться стандартные средства пакета OrCAD. На заключительном этапе разработки осуществляется трансляция списка соединений примитивов схемы в конфигурацию логических ячеек SOPC фирмы Triscend. В результат получается EDIF-файл, который также будет в дальнейшем импортироваться в САПР FastChip Triscend для создания требуемой аппаратной части разрабатываемой системы.

Этап 5. Проектирование общей схемы — сборка проекта из отдельных фрагментов

Нетрудно видеть, что на верхнем уровне иерархии для реализации рассматриваемого устройства (рис. 4.23) из библиотеки элементов САПР FastChip можно использовать следующий набор библиотечных настраиваемых модулей:

- Три группы восьмиразрядных элементов (в результате образующих традиционные входные порты МП-ядра):
  - входные контакты — `Inp_A`, `Inp_B`, `Inp_C`;
  - входные буферные регистры RD — `Reg_A`, `Reg_B`, `Reg_C`;
  - адресные селекторы — `AdrSel_A`, `AdrSel_B`, `AdrSel_C`.
- Две группы восьмиразрядных элементов (в результате образующих традиционные выходные порты МП-ядра):
  - выходные буферные каскады WR — `WR_D`, `WR_Ust`;
  - адресные селекторы — `AdrSel_D`, `AdrSel_UST`;
  - выходные регистры — `Reg_D`, `Reg_Ust`.

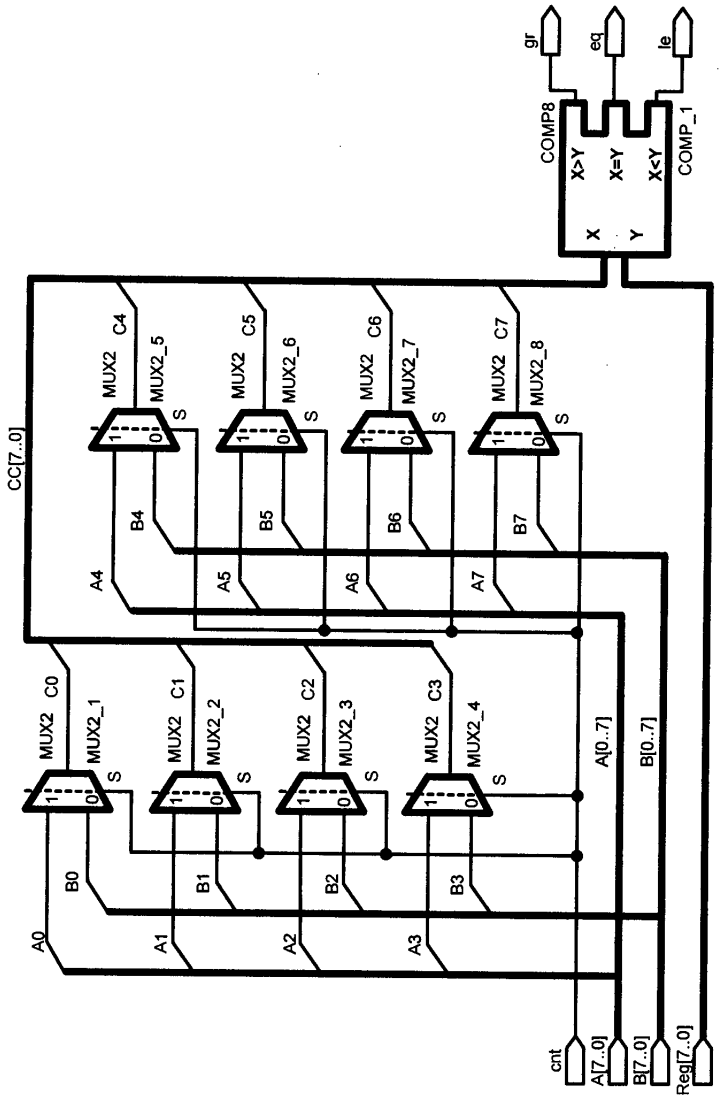


Рис. 4.22. Структурная схема блока `Cnt_Cmp`

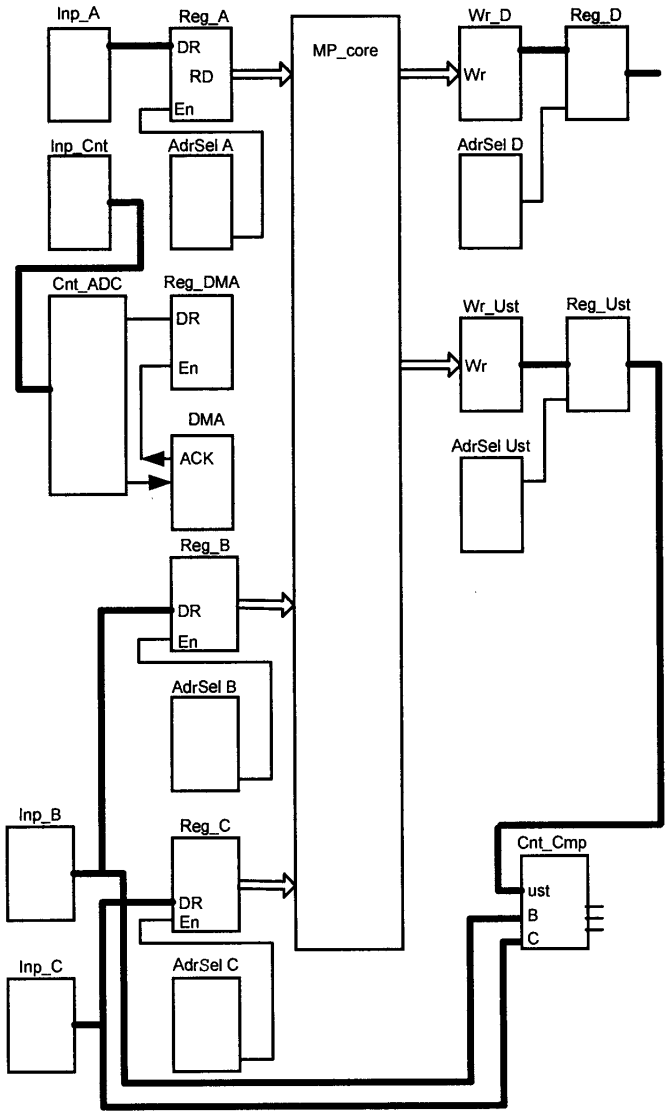


Рис. 4.23. Структура конфигурируемых аппаратных ресурсов проектируемой системы

- Для организации ввода данных в ОЗУ по линии ПДП требуется использование входного буфера `Reg_DMA` и связанного с ним селектора адреса DMA.

Кроме библиотечных модулей в схему необходимо импортировать файлы EDIF, определяющие конструкцию блоков `Cnt_ADC` и `Cnt_Cmp`. Для подключения управляющих сигналов к входам автомата `Cnt_ADC` используется модуль входных контактов — `Inp_Cnt`.

Понятие параметризированных модулей соответствует возможности настроить выбранный библиотечный элемент на определенный режим функционирования, на определенную разрядность данных, их полярность и т. д.

Построение проекта в САПР Triscend FastChip состоит в образовании требуемой схемы из используемых в проекте модулей путем описания соединения модулей между собой. Модули будут соединены друг с другом, если к их выводам приписать цепи (net) с одинаковыми именами. Поскольку схема соединений непосредственно не отображается на экране, разработчик должен следить за правильностью соединений.

После определения структуры аппаратной части проекта (ресурсов CSL) разработчик задает *расположение внешних контактов* аппаратной части системы по периметру кристалла.

Следующий шаг проектной процедуры связан с *проверкой корректности* построения конфигурируемой части проекта. САПР проверяет правила создания проекта и выдает предупреждения или сообщения об ошибках. Проектировщику сообщаются имена неподключенных цепей или цепей, имеющих несколько источников сигналов, и т. д.

После проверки корректности САПР может быть запущена на выполнение *процедуры распределения логических ресурсов* по кристаллу и образования системы их соединений (в терминологии фирмы Triscend процедура носит название Bind, связывание). Результатом этой процедуры является создание файла конфигурации программируемой логики кристалла. Теперь САПР готова в любой момент времени загрузить созданную конфигурацию в кристалл. Если предполагается отладка только автономно работающих (без процессора) аппаратных ресурсов, то можно приступать к работам, описанным в *разд. 4.4.5*. Встроенные в кристалл E5 средства внутрикристальной отладки создают уникальные возможности для настройки аппаратных решений.

4.4.3. Настройка предопределенных ресурсов кристалла

Поскольку мало вероятно, что проектировщик выбрал для реализации своего проекта кристалл SOPC, но собирается опираться в проекте только на аппаратные ресурсы кристалла, то, как правило, далее разработчик присту-

пает к действиям, связанным с использованием встроенного в кристалл SOPC процессора и его периферийных узлов. Эти действия предполагают разработку той части программного обеспечения микропроцессорного ядра, которая обеспечит правильную настройку начальных параметров периферийных блоков, а также тех блоков и аппаратных связей, которыми процессор связан с программируемой логикой (разработанной проектировщиком аппаратурой). В рассматриваемом примере необходима инициализация двух блоков. Это контроллер прямого доступа к памяти и контроллер прерываний. Задание параметров предопределенных (Hardcore) ресурсов кристалла в САПР FastChip осуществляется в интерактивном режиме путем выбора пиктограмм периферийных устройств и заполнения соответствующих опций выпадающего меню. За начальную настройку процессорных ресурсов в САПР FastChip отвечает так называемый заголовочный файл.

Создание заголовочного файла

После спецификации всех аппаратных ресурсов САПР Triscend FastChip выполняет программу вторичной инициализации (Secondary initialization), которая формирует заголовочный файл (Tailored Header File format). Основная особенность формата этого файла состоит в том, что он содержит назначение логических адресов для символьных имен объектов, использованных в проекте. Это является одним из удобств, предлагаемых системной интеграцией. Имена всех пользовательских модулей области CSL кристалла, связанных с интерфейсной шиной (CSI Bus) микропроцессора при помощи селекторов адреса, становятся автоматически доступными программному обеспечению микроконтроллера и могут использоваться на последующих этапах проектирования программной части проекта, включая компиляцию, верификацию и отладку.

4.4.4. Разработка программного обеспечения

Разработка программного обеспечения для кристаллов E5 практически ничем, кроме отмеченных выше свойств расширенной версии заголовочного файла, не отличается от стандартного процесса разработки программного обеспечения. Эта разработка должна осуществляться с помощью внешних относительно Triscend FastChip компиляторов, поскольку САПР не содержит интегрированных в нее подобных средств. В качестве средств разработки программной части проекта может использоваться большинство существующих компиляторов (таких как Franklin, IAR, Tasking и т. д.), однако более полную интеграцию с пакетом Triscend FastChip могут обеспечить инструментальные средства разработки программного обеспечения фирм Keil Software или Archimedes Software. Именно поэтому фирма Triscend рекомендует пользоваться указанными средствами.

Инструментальные средства разработки программного обеспечения современных САПР (такие как продукция фирмы Keil Software) позволяют не

только разрабатывать прикладные программы на языке ассемблера или C, но и, что очень важно, поддерживают все стадии их разработки. Для фирмы Keil Software — это интегрированная среда разработки mVision и высокоуровневый отладчик-симулятор dScope. Интегрированная среда разработки осуществляет настройку всех программ пакета и управление всеми стадиями разработки, включая вызов специализированного текстового редактора (цветовое выделение синтаксиса и диалоговое исправление ошибок), вызов менеджера проектов (простая интеграция различных файлов в проект) и вызов отладчика.

Как правило, разработка программного обеспечения микроконтроллеров состоит из написания трех основных частей: заголовочного файла (Header File), основной программы (Main Function) и подпрограмм обработки прерываний (Interrupt Service Routine, ISR).

В рассматриваемом примере вопросы разработки программного обеспечения микропроцессорной системы рассматриваться не будут. Новой разработки практически не требуется, поскольку модернизация ранее существующей системы почти не изменила функционирования элементов, связанных с программным обеспечением МП. В своей основе программное обеспечение берется от старой разработки. Исключение составляют фрагменты, связанные с новым объектом — контроллером прямого доступа к памяти. Программная реализация режима ПДП в Triscend E5 осложнена тем, что контроллер прямого доступа поддерживает доступ к расширенному относительно стандартного (MSC-51) 64-килобайтного пространства данных. Однако в результате при помощи контроллера ПДП можно, например, получить доступ к внешней относительно кристалла памяти команд или данных.

Компиляция и создание объектного кода

После создания всех модулей программной части проекта выполняется его компиляция. Время создания объектного кода программного обеспечения наступает после того, как объектный код отдельных модулей программы синтаксически и логически отлажен. Для Keil Software этому соответствует вызов директивы создания объектного кода. Созданный в результате компоновки файл может далее впрямую использоваться САПР Triscend FastChip.

Разработка на этом этапе завершается компиляцией кода программ и созданием результирующего файла <project>.hex. Для обеспечения работоспособности создаваемого программного обеспечения целесообразна его отладка в рамках симулирующих программ.

Кодовая симуляция и отладка

Современные инструментальные комплексы, предназначенные для проектирования программного обеспечения микропроцессоров и микроконтроллеров, включают в свой состав специальные средства для интерактивного

процесса отладки. Рассматриваемый в качестве примера комплекс программ фирмы Keil Software не является исключением. Высокоуровневый отладчик-симулятор dScope, входящий в состав комплекса, позволяет производить отладку проектов на языке ассемблера, языке С или в смешанных форматах. С его помощью путем моделирования поведения окружающей среды и предопределенных ресурсов процессора (таймеров, параллельных или последовательных каналов ввода/вывода и т. д.) можно производить отладку программного обеспечения в пошаговом режиме или режиме работы с точками останова. Удобство отладки обеспечивается возможностью легкой организации контроля за состоянием всех объектов программы в различных окнах наблюдений. Кроме того, возможна также оценка производительности и эффективности кодового представления программного обеспечения.

4.4.5. Монтирование ресурсов SOPC в кристалл и комплексная отладка проекта

Следующим этапом работ является создание загрузочного файла. Загрузочный файл содержит информацию об аппаратной и о программной частях системы, загружаемых в кристалл SOPC. Список цепей, соединяющих модули CSL, и информация, настраивающая вентили на требуемый режим функционирования, берутся из файла конфигурации кристалла. А содержимое памяти программ микропроцессорного ядра — из выходного файла пакета Keil Software.

Загрузка проекта

Как и большинство других БИС ПЛ, семейство кристаллов E5 фирмы Triscend позволяет загружать проект в реальный кристалл различными способами. Загрузка в оперативную память БИС E5 может выполняться из внешней последовательной памяти типа EPROM или из внешней параллельной памяти типа SRAM, Flash Memory или EPROM. Загрузка проекта в постоянную память обычно является финальным действием в проектной процедуре. Наличие же возможности загружать проект в ОЗУ реальных микросхем непосредственно из персонального компьютера (с помощью JTAG-интерфейса) позволяет построить более эффективную процедуру верификации проекта как при отладке прототипа, так и при натурных экспериментах на конечной продукции.

Натурная отладка проекта

Обычно совместная отладка функционирования аппаратных и программных частей системы возможна только после создания опытного образца системы. Для отладки кристаллов типа SOPC могут использоваться как традиционные средства и методы (рассмотренные в предыдущих разделах), так и методы,

учитывающие возможность перепрограммирования БИС ПЛ, т. е. структуры, обрамляющей микропроцессор или микроконтроллер. Более того, могут быть разработаны специальные тестовые конфигурации (нередко опирающиеся на тестовые варианты ПО МК), которыми следует пользоваться не только на этапе проверки результатов проектирования, но и на этапах изготовления, выпуска или контроля промышленной продукции.

Весь комплекс средств, связанных с проектированием программируемой логики класса реконфигурируемых систем на кристалле (CSoC), начиная от архитектуры кристаллов и кончая ресурсами САПР, поддерживающих создание реальной аппаратуры на всех этапах разработки, позволяет существенно уменьшить время проектирования и ускорить выпуск конечной продукции на рынок.

Для ускорения разработок проектов фирма Triscend, как и большинство других фирм производителей БИС ПЛ, выпускает специальные отладочные средства, например плату Triscend Evaluation Board, содержащую в своей основе кристалл TE520S40.

Другой отладочной платой является демонстрационная плата myCSoCKit, разработанная фирмой XESS Inc., USA и содержащая в своей основе кристалл TE505S16.

Как уже отмечалось, кристаллы E5 фирмы Triscend поддерживают расширенный вариант JTAG-интерфейса, обеспечивающего загрузку программной и аппаратной частей проекта внутрь кристалла. Допустимы два варианта комплексной отладки проекта. Один состоит в использовании сначала прототипных плат и лишь затем в переходе к отладке в конечной системе, другой сразу ориентируется на отладку в выпускаемой продукции.

Заложенные в кристалл E5 уникальные возможности внутрикристальной отладки позволяют САПР FastChip работать не только в режиме редактирования и компиляции проекта, но и в режиме отладки реальной аппаратуры (Real-Time In-System Debug). На эти же возможности могут опираться и типовые инструментальные средства отладки микропроцессорных систем (например, уже упоминавшийся пакет фирмы Keil Software). Каждый из вариантов имеет свои достоинства и целесообразные области применения.

Удобство и эффективность комплексной отладки аппаратно-программных средств в САПР FastChip достигается за счет возможности устанавливать точки останова в отлаживаемых программах путем занесения информации в специальные встроенные в кристаллы E5 аппаратные средства (Hardware Breakpoint Unit). Переход САПР FastChip в режим отладки (Debug) сопровождается открытием в основном меню дополнительного окна наблюдения за отлаживаемыми объектами (Debug Watch).

Отладка аппаратуры непосредственно в FastChip имеет определенные ограничения по наглядности и доступности к объектам по их символическим именам. Поэтому отладку больших объемов программного обеспечения бо-

лее рационально производить, опираясь на режим отладки реальной аппаратуры отладчика пакета Keil Software. Особенно удобен такой режим при отладке программного обеспечения, разработанного на языке С. Установка точек останова либо в исходном тексте программы, либо и в дизассемблерной форме этого же текста существенно проще, чем установка абсолютных значений точек останова в FastChip.

Вариант отладки проекта непосредственно в целевой системе имеет целый ряд преимуществ по сравнению с отладкой при помощи моделирующих программ и даже с прототипными системами. САПР позволяет загружать отлаживаемый вариант программной и конфигурационной памяти в реальную БИС сразу целевой системы и отлаживать проект с требуемыми временными параметрами при наличии всего целевого окружения (Software и Hardware). Информация о состоянии отдельных модулей проекта и величинах, выбранных для наблюдения параметров, возможность модифицировать величины адресуемых элементов проекта, потактовое или пошаговое выполнение программы — все это позволяет существенно упростить процедуру поиска обычно трудно локализуемых допущенных в проекте ошибок. Ряд особенностей отладочного режима работы FastChip делает его похожим на отладку с помощью внутрисхемных эмуляторов. Например, очень удобной оказывается возможность наблюдения за значениями контролируемых объектов отлаживаемой системы с циклическим обновлением результатов.

Завершение проектирования БИС SOPC, включающей аппаратные ресурсы, требуемые для функционирования микропроцессорной системы и отвечающей требованиям технического задания на разработку микропроцессорной системы, позволяет приступить к выполнению заключительных этапов проектирования, связанных с конструкторской разработкой всей системы целиком в форме конечного продукта.

4.4.6. Разработка конструкции устройства

Исходной информацией для заключительного этапа проектирования, а именно разработки конструкторско-технологической документации на печатную плату, содержащую проектируемую микропроцессорную систему, являются не только данные о межсоединениях стандартных элементов системы, но и сведения о размещении сигналов по контактам ввода/вывода БИС SOPC.

Ввиду того, что память БИС E5 организована в форме SRAM, требуется решение о способе загрузки памяти конфигурации и памяти МП. Кристалл E5 допускает использование различных видов БИС ПЗУ, хранящих загружаемую после включения питания память конфигурации и кода. Возможны две основные формы загрузки: последовательная и параллельная. Загрузка из последовательного ПЗУ требует значительных временных затрат, но зато

большее количество контактов БИС SOPC может быть использовано в качестве пользовательских контактов ввода/вывода.

Другим вопросом, требующим решения, является вопрос о способе формирования тактовой частоты. Кристалл E5 допускает применение для простых систем (в которых не требуются повышенные характеристики стабильности и точности частоты) встроенных ресурсов, в противном случае должны использоваться наружные элементы типа осциллятора или кварцевого резонатора.

В отличие от стандартных элементов, у которых функциональное распределение входных и выходных сигналов заранее фиксировано и не может изменяться, БИС ПЛ дают возможность проектировщику задавать собственное распределение контактов для большинства контактов. Если по соображениям топологии межсоединений элементов на печатной плате желательно определенное распределение номеров контактов БИС ПЛ, то требуется поручить компилятору САПР выполнить компиляцию проекта с заданным распределением номеров контактов для всех сигналов. Подобная возможность БИС ПЛ, с одной стороны, позволяет получать очень эффективные результаты трассировки межсоединений, а с другой — выполнять конструкторскую разработку проекта до завершения проектных работ.