Министерство образования Республики Беларусь
Учреждение образования
«Белорусский государственный университет
информатики и радиоэлектроники»

Кафедра электронных вычислительных средств

А. А. Петровский, М. И. Вашкевич, М. М. Родионов

ПРОЕКТИРОВАНИЕ ЭВС С ДИНАМИЧЕСКИ РЕКОНФИГУРИРУЕМОЙ АРХИТЕКТУРОЙ

Методическое пособие
для студентов специальности 1-40 02 02
«Электронные вычислительные средства»
дневной формы обучения

Минск БГУИР 2011

УДК 004.383.3-027.31(076.6)
ББК 32.973.26-02я73
П31

Р е ц е н з е н т:
заведующий кафедрой ИИТ учреждения образования
«Белорусский государственный университет
информатики и радиоэлектроники»,
доктор технических наук, профессор В. В. Голенков

Петровский, А. А.

П31 Проектирование ЭВС с динамически реконфигурируемой архитектурой : метод. пособие для студ. спец. 1-40 02 02 «Электронные вычислительные средства» днев. формы обуч. / А. А. Петровский, М. И. Вашкевич, М. М. Родионов. – Минск : БГУИР, 2011. – 40 с. : ил.

ISBN 978-985-488-628-2.

Описываются сопроцессорные конфигурации для построения эффективных вычислительных средств для систем реального времени. Рассмотрено проектирование вычислительного модуля на базе RISC-процессора Plasma с архитектурой MIPS. В качестве примера показана разработка сопроцессора, реализующего CORDIC алгоритм, предназначенного для совместной работы с процессором Plasma. Показаны все этапы разработки сопроцессора: от анализа алгоритма до VHDL-описания устройства. Рассмотрены вопросы реализации алгоритмов цифровой обработки сигналов на основе арифметики с фиксированной запятой переменного формата.

УДК 004.383.3-027.31(076.6)
ББК 32.973.26-02я73

ISBN 978-985-488-628-2

Содержание

1. Проектирование проблемно-ориентированных вычислительных средств

1.1. Сопроцессорные конфигурации

При построении высокопроизводительных вычислительных систем часто используется принцип специализации. Суть его заключается в разработке вспомогательных специализированных процессоров, ориентированных на конкретные прикладные области. Такие процессоры работают под управлением центрального процессора и разделяют с ним основную память. Специализация позволяет достичь высокого быстродействия вспомогательных процессоров и повысить эффективную производительность системы благодаря параллельной работе нескольких процессоров.

В сопроцессорной конфигурации вспомогательный процессор (сопроцессор) подключается к системной шине параллельно с центральным процессором (ЦП). Сопроцессор не имеет своей отдельной программы и не может считывать команды из памяти, но может обращаться к ней для записи и считывания данных, запрашивая для этого шину у ЦП. Сопроцессор не может выполнять команды ЦП, но свои команды выполняет очень быстро (по сравнению с их программной реализацией). Такое «разделение труда» между сопроцессором и ЦП позволяет достичь очень высокой производительности.

Остановимся на достоинствах и недостатках сопроцессорной конфигурации. Её альтернативой является разработка ЦП со всеми функциональными возможностями сопроцессора – такая задача оказывается довольно сложной и ведет к увеличению длительности проектирования, усложнению системы и уменьшению её надежности. Поэтому ЦП и сопроцессор в отдельности разработать проще, а при умелом программировании оба процессора могут работать параллельно. Наличие двух и более процессоров обеспечивает разработчикам дополнительную гибкость – сопроцессоры применяются только там, где они необходимы.

1.2. Вычислительная система на основе RISC-процессора

На рис. 1.1 приведена система с несколькими сопроцессорами. В качестве центрального процессора используется RISC-процессор Plasma с архитектурой MIPS. Plasma представляет собой 32-разрядный процессор, VHDL-описание которого свободно распространяется через сеть интернет (http://opencores.org). Блок-схема процессора показана на рис. 1.2. В состав процессора входят аппаратный умножитель/делитель, блок 32-разрядных регистров, устройство формирования адреса следующей команды, устройство шинного интерфейса, мультиплексор шины, АЛУ, сдвигатель и блок управления (логика управления).

На базе микропроцессора Plasma можно построить сложную вычислительную систему с подключаемыми сопроцессорами.

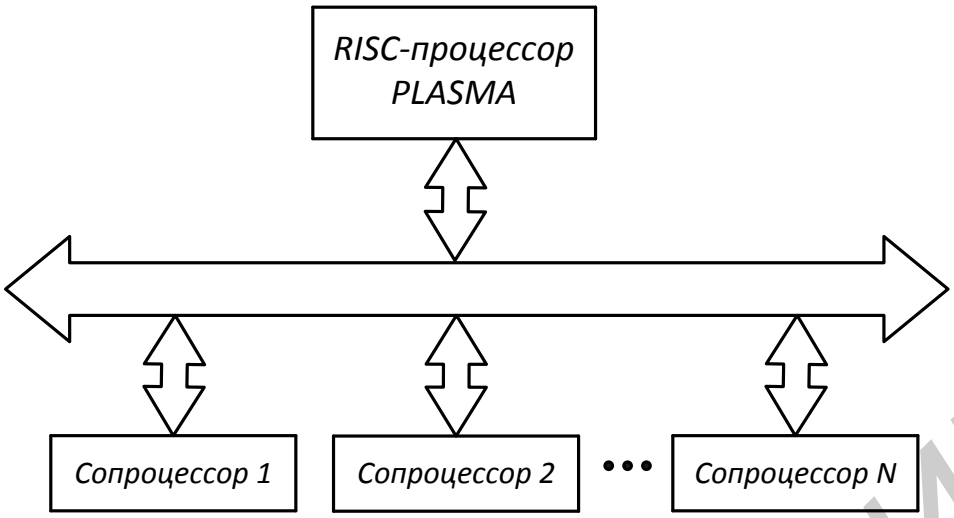


Рис. 1.1. Сопроцессорная конфигурация

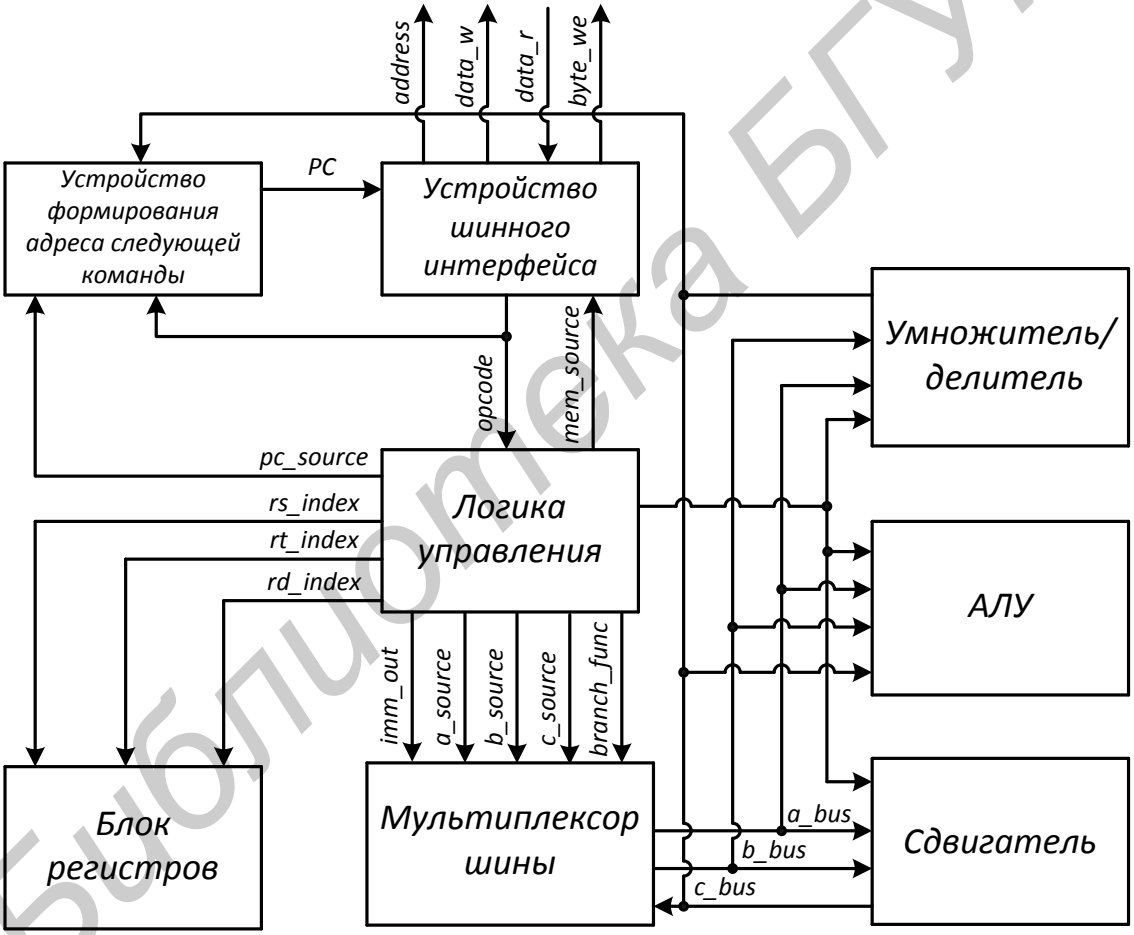


Рис. 1.2. Блок-схема процессора Plasma

В следующих разделах будут рассмотрены вопросы проектирования сопроцессоров, подключаемых к шинному интерфейсу процессора Plasma. Ниже приведена декларация входных и выходных сигналов ядра процессора Plasma.

```
entity plasma_cpu is
port(clk       : in std_logic;
```

```
    reset_in  : in std_logic;
    intr_in   : in std_logic;
    address_next : out std_logic_vector(31 downto 2);
    byte_we_next : out std_logic_vector(03 downto 0);
    address      : out std_logic_vector(31 downto 2);
    byte_we      : out std_logic_vector(03 downto 0);
    data_w       : out std_logic_vector(31 downto 0);
    data_r       : in  std_logic_vector(31 downto 0);
    mem_pause    : in  std_logic);
end;
```

В таблице дано описание входных и выходных портов процессора Plasma.

Название	Описание
clk	Вход для сигнала тактирования
reset_in	Вход сброса в начальное состояние
intr_in	Вход прерывания
address_next	Адрес для следующего такта
byte_we_next	Сигналы разрешения записи байтов для следующего такта
address	Шина адреса
byte_we	Сигналы разрешения записи байтов
data_w	Шина данных по записи
data_r	Шина данных по чтению
mem_pause	Сигнал запрещения обращения к памяти

1.3. Состав регистров процессора Plasma

Так как процессор Plasma имеет архитектуру MIPS в его состав входят 32 регистра общего назначения. Первый регистр ($0) содержит значение 0 и не может быть изменен. Последний регистр ($31/$ra) содержит адрес возврата из подпрограммы. Полная информация о регистрах процессора с архитектурой MIPS дана в таблице.

Название	Номер регистра	Назначение
1	2	3
$zero	0	Значение константы 0
$at	1	Временный регистр ассемблера
$v0-$v1	2-3	Регистры, используемые для передачи значений, возвращаемых подпрограммой
$a0-$a3	4-7	Регистры для передачи параметров в подпрограмму
$t0-$t9	8-15	Регистры общего назначения. Их содержимое может изменяться при вызове подпрограмм

Окончание таблицы

1	2	3
`$s0-$s7`	16-23	Регистры, используемые для сохранения контекста, при вызове подпрограмм
`$t8-$t9`	24-25	Регистры общего назначения, содержимое которых не теряется при вызове подпрограммы
`$k0-$k1`	26-27	Зарезервированы для описания прерывания
`$gp`	28	Глобальный указатель
`$sp`	29	Указатель стека
`$fp`	30	Указатель фрейма
`$ra`	31	Адрес возврата из подпрограммы

Архитектура MIPS предусматривает только двухоперандные команды, при этом один операнд обязательно находится в регистре, а для второго адрес указывается в команде.

Все команды MIPS делятся на три типа. К R-типу относят команды, при выполнении которых используются только регистры процессора. Команды, в которых имеется непосредственный операнд, относятся к I-типу. Все команды переходов относят к J-типу. Форматы команд MIPS приведены на рис. 1.3.

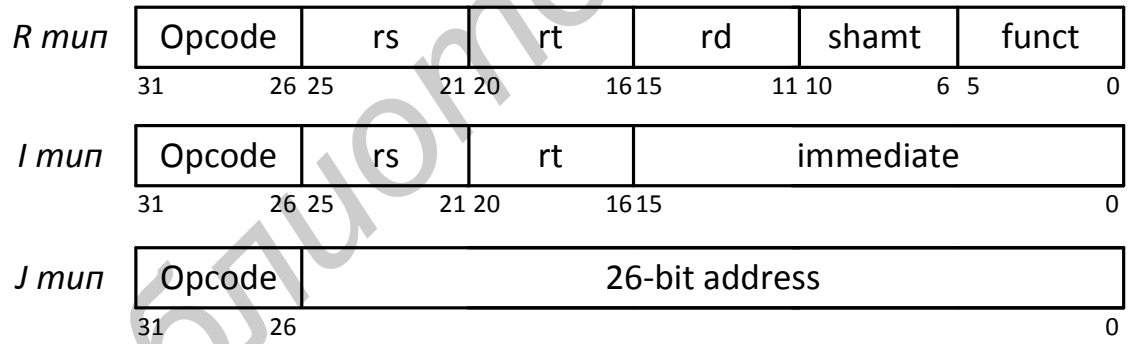


Рис. 1.3. Форматы команд MIPS

В начале каждой команды имеется поле, задающее код операции (Opcode), далее в зависимости от типа команды следуют поля, задающие номера регистров (rs/rt/rd), непосредственный операнд (immediate) или адрес перехода.

2. Реализация CORDIC-алгоритма

При построении вычислительной системы разработчику всегда приходится решать, каким будет в ней соотношение программных и аппаратных средств. В последние несколько десятилетий за счет стремительного развития элементной базы наблюдается переход от программной к аппаратурной реализации многих функций. Это прежде всего относится к вычислению математических функций и преобразований. Часто в виде отдельных аппаратурных модулей реализуют генераторы тригонометрических функций, генераторы случайных чисел, модули, выполняющие поворот вектора на заданный угол, устройства, реализующие дискретное преобразование Фурье. Аппаратурная реализация данных функций по сравнению с программной имеет ряд преимуществ в части быстродействия, пропускной способности и простоты программирования.

В данном разделе рассматривается итерационный метод вычисления элементарных функций, который в отечественной литературе носит название «*цифра за цифрой*», а в зарубежной известен как CORDIC (COordinate Rotation Digital Computer). Чаще всего CORDIC-алгоритм используют для вычисления поворота вектора на плоскости, однако, известно также множество аппаратурных реализаций более сложных преобразований, таких, как дискретное преобразование Фурье, Хартли, в основе которых лежит CORDIC-алгоритм.

2.1. Описание CORDIC-алгоритма

Начнем изучение CORDIC алгоритма с задачи поворота вектора (либо точки) на заданный угол α. Как известно, вектор в двумерном пространстве задается парой координат (x_1, y_1). Поворот вектора на угол α можно записать в матричном виде следующим образом:

$$\begin{matrix} x_2 \\ y_2 \end{matrix} = \mathbf{R}(\alpha) \times \begin{matrix} x_1 \\ y_1 \end{matrix}, \qquad (2.1)$$

где $\mathbf{R}(\alpha)$ матрица поворота, которая задается как

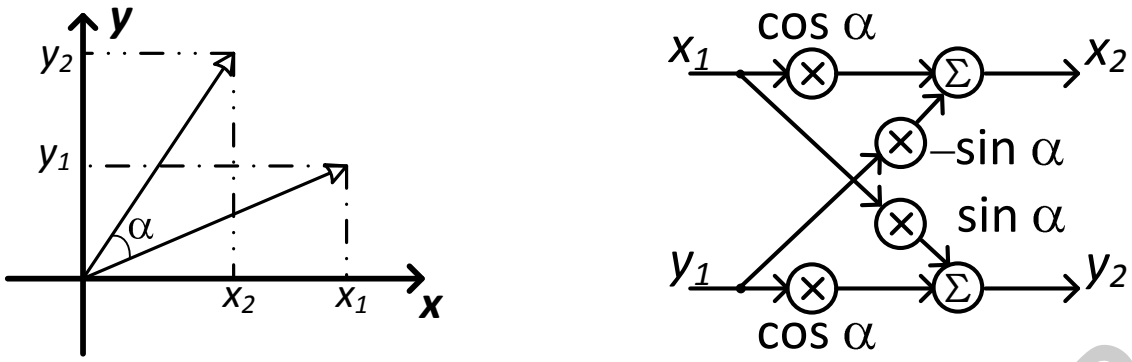
$$\mathbf{R}(\alpha) = \begin{matrix} \cos\alpha & -\sin\alpha \\ \sin\alpha & \cos\alpha \end{matrix}. \qquad (2.2)$$

Прямая реализация выражения (2.1) потребует выполнения четырех операций умножения и двух операций сложения, в то время как использование CORDIC-алгоритма позволяет выполнить операцию поворота вектора без использования операций умножения. Преобразуем матрицу поворота следующим образом:

$$\mathbf{R}(\alpha) = \cos\alpha \times \begin{matrix} 1 & -\operatorname{tg}\alpha \\ \operatorname{tg}\alpha & 1 \end{matrix}, \qquad (2.3)$$

далее, используя соотношение $\cos\alpha = 1/\sqrt{1+(\operatorname{tg}\alpha)^2}$, получим

$$\mathbf{R}(\alpha) = \frac{1}{\sqrt{1+(\operatorname{tg}\alpha)^2}} \times \begin{matrix} 1 & -\operatorname{tg}\alpha \\ \operatorname{tg}\alpha & 1 \end{matrix}. \qquad (2.4)$$



а *б*

Рис. 2.1. Поворот вектора на угол α:
а – поворот вектора на плоскости; *б* – прямая реализация поворота

Отметим одно свойство матриц поворота, которое понадобится для дальнейших выкладок:

$$\mathbf{R}(\alpha+\beta) = \mathbf{R}(\alpha) \times \mathbf{R}(\beta). \tag{2.5}$$

Основная идея CORDIC-алгоритма заключается в том, чтобы исходный угол поворота α представить в виде линейной комбинации заранее заданных (базовых) углов θ_i:

$$\alpha \approx \sum_{i=0}^{N_A-1} \mu(i)\theta_i, \tag{2.6}$$

где N_A – количество используемых базовых углов поворота; $\mu(i) \in \{1, -1\}$ – последовательность, определяющая поворот.

Базовые углы θ_i задаются следующим образом:

$$\theta_i \triangleq \operatorname{arctg} 2^{-i}. \tag{2.7}$$

Далее, используя (2.5) и (2.6), получаем

$$\begin{aligned}\mathbf{R}(\alpha) &\approx \mathbf{R}\left(\sum_{i=0}^{N_A-1} \mu(i)\theta_i\right) = \\ &= \mathbf{R}(\mu(0)\theta_0) \times \mathbf{R}(\mu(1)\theta_1) \ldots \mathbf{R}(\mu(N_A-1)\theta_{N_A-1}) = \\ &= \prod_{i=0}^{N_A-1} \mathbf{R}(\mu(i)\theta_i).\end{aligned} \tag{2.8}$$

Используя (2.8), выражение (2.4) переписывается в виде

$$\mathbf{R}(\alpha) \approx \prod_{i=0}^{N_A-1} \mathbf{R}(\mu(i)\theta_i) = \prod_{i=0}^{N_A-1} \frac{1}{\sqrt{1+2^{-2i}}} \times \begin{pmatrix} 1 & -\mu(i)2^{-i} \\ \mu(i)2^{-i} & 1 \end{pmatrix}. \tag{2.9}$$

При выводе последнего выражения учитывалось, что $\mu(i)^2 = 1$ и то, что $\operatorname{tg}(\mu(i)\theta_i) = \mu(i)2^{-i}$. Перепишем (2.9) в следующем виде:

$$\mathbf{R}\ \alpha\ \approx K_{N_A} \times \prod_{i=0}^{N_A-1} \begin{matrix} 1 & -\mu\ i\ 2^{-i} \\ \mu\ i\ 2^{-i} & 1 \end{matrix}, \qquad (2.10)$$

где

$$K_{N_A} = \prod_{i=0}^{N_A-1} \frac{1}{\sqrt{1+2^{-2i}}}. \qquad (2.11)$$

Таким образом, задача поворота вектора сводится к выполнению последовательных операций сдвига и сложения с последующим масштабированием выходных данных на коэффициент K_{N_A}. Важно отметить, что как только выбрано множество базовых углов, коэффициент масштабирования K_{N_A} оказывается фиксированным и постоянным для всех вращений. Ниже приведен пример использования CORDIC-алгоритма.

Пример. *Необходимо выполнить поворот вектора на угол* $\alpha = \frac{\pi}{6} = 0{,}523598$ *радиан, используя CORDIC-алгоритм (*$N_A = 8$*). Вначале составим таблицу базовых углов:*

Индекс	**Базовый угол**	**Значение в радианах**
$i = 0$	$\theta_0 = \text{arctg}(2^{-0})$	0,785398
$i = 1$	$\theta_1 = \text{arctg}(2^{-1})$	0,463647
$i = 2$	$\theta_2 = \text{arctg}(2^{-2})$	0,244978
$i = 3$	$\theta_3 = \text{arctg}(2^{-3})$	0,124354
$i = 4$	$\theta_4 = \text{arctg}(2^{-4})$	0,062419
$i = 5$	$\theta_5 = \text{arctg}(2^{-5})$	0.031240
$i = 6$	$\theta_6 = \text{arctg}(2^{-6})$	0,015623
$i = 7$	$\theta_7 = \text{arctg}(2^{-7})$	0,007812

Также для $N_A = 8$ *по формуле* (2.11) *рассчитаем масштабирующий множитель:*

$$K_{N_A} = \prod_{i=0}^{7} \frac{1}{\sqrt{1+2^{-2i}}} = 0{,}607259.$$

Далее определим последовательность $\mu(i)$*, определяющую поворот.*

$$\alpha \approx \theta_0 - \theta_1 + \theta_2 - \theta_3 + \theta_4 + \theta_5 - \theta_6 + \theta_7 = 0{,}528221.$$

Таким образом, $\mu\ i\ = [1\quad -1\quad 1\quad -1\quad 1\quad 1\quad -1\quad 1]$*, при этом ошибка аппроксимации угла* α *равна*

$$\epsilon = \alpha - \sum_{i=0}^{7} \mu\ i\ \theta_i \approx -0{,}004623 \text{ рад.}$$

Ниже показана последовательность микроповоротов, которая имеет место

при использовании CORDIC-алгоритма. Пусть исходный вектор имеет координаты $x, y = (1; 0)$.

Номер итерации	**Микроповорот**
$i = 0$	$\begin{matrix}1\\1\end{matrix} = \begin{matrix}1 & -2^{-0}\\2^{-0} & 1\end{matrix} \times \begin{matrix}1\\0\end{matrix}$
$i = 1$	$\begin{matrix}3/2\\1/2\end{matrix} = \begin{matrix}1 & 2^{-1}\\-2^{-1} & 1\end{matrix} \times \begin{matrix}1\\1\end{matrix}$
$i = 2$	$\begin{matrix}11/2^3\\7/2^3\end{matrix} = \begin{matrix}1 & -2^{-2}\\2^{-2} & 1\end{matrix} \times \begin{matrix}3/2\\1/2\end{matrix}$
$i = 3$	$\begin{matrix}95/2^6\\45/2^6\end{matrix} = \begin{matrix}1 & 2^{-3}\\-2^{-3} & 1\end{matrix} \times \begin{matrix}11/2^3\\7/2^3\end{matrix}$
$i = 4$	$\begin{matrix}1475/2^{10}\\815/2^{10}\end{matrix} = \begin{matrix}1 & -2^{-4}\\2^{-4} & 1\end{matrix} \times \begin{matrix}95/2^6\\45/2^6\end{matrix}$
$i = 5$	$\begin{matrix}46385/2^{15}\\27555/2^{15}\end{matrix} = \begin{matrix}1 & -2^{-5}\\2^{-5} & 1\end{matrix} \times \begin{matrix}1475/2^{10}\\815/2^{10}\end{matrix}$
$i = 6$	$\begin{matrix}2971395/2^{21}\\1717135/2^{21}\end{matrix} = \begin{matrix}1 & 2^{-6}\\-2^{-6} & 1\end{matrix} \times \begin{matrix}46385/2^{15}\\27555/2^{15}\end{matrix}$
$i = 7$	$\begin{matrix}378621425/2^{28}\\222764675/2^{28}\end{matrix} = \begin{matrix}1 & -2^{-7}\\2^{-7} & 1\end{matrix} \times \begin{matrix}2971395/2^{21}\\1717135/2^{21}\end{matrix}$
Масштабирование	$\begin{matrix}0{,}856524\\0{,}503942\end{matrix} = 0{,}607259 \times \begin{matrix}1{,}410475\\0{,}829863\end{matrix}$

Реализацию умножения на константу (масштабирование) можно выполнить с использованием четырех сложений, если представить коэффициент масштабирования как

$$0{,}607259 \approx 2^{-1} + 2^{-3} - 2^{-6} - 2^{-9} - 2^{-13}.$$

Таким образом, $(0{,}856524; 0{,}503942)$ *– это координаты вектора* $(1; 0)$*, повернутого на* $\pi/6$ *радиан, найденные посредством CORDIC-алгоритма.*

2.2. Сопроцессор для реализации CORDIC-алгоритма

2.2.1. Общие сведения о проектировании процессоров

В составе любого процессора можно выделить как минимум две составные части – арифметическо-логическое устройство (АЛУ) и устройство управления (УУ).

АЛУ является частью процессора, назначением которой является выполнение арифметических и логических операций над двоичными числами, а также операций сдвигов двоичных чисел. В наиболее простых процессорах АЛУ выполняет лишь операции сложения и вычитания чисел с фиксированной

запятой, а также логические операции и операции сдвига. Умножение и деление чисел в таких процессорах реализуется программно в виде последовательности повторяющихся команд сложения и сдвига. В более производительных процессорах применяют АЛУ, которые аппаратно реализуют операции умножения и деления чисел с фиксированной запятой. Такая реализация операций приводит к уменьшению времени их выполнения по сравнению с программной реализацией данных операций.

Различают параллельные и последовательные АЛУ. В *параллельных* АЛУ все разряды чисел обрабатываются одновременно (в одном машинном такте). Для сложения чисел параллельное АЛУ содержит многоразрядный сумматор. В *последовательных* АЛУ разряды чисел обрабатываются последовательно во времени; сложение чисел в последовательном АЛУ выполняется с помощью входящего в его состав одноразрядного сумматора разряд за разрядом, начиная с младших.

Выполнение операций в АЛУ сводится к выполнению последовательности микроопераций под действием управляющих сигналов $y_1, y_2, \dots y_N$, которые вырабатывает устройство управления УУ (рис. 2.2).

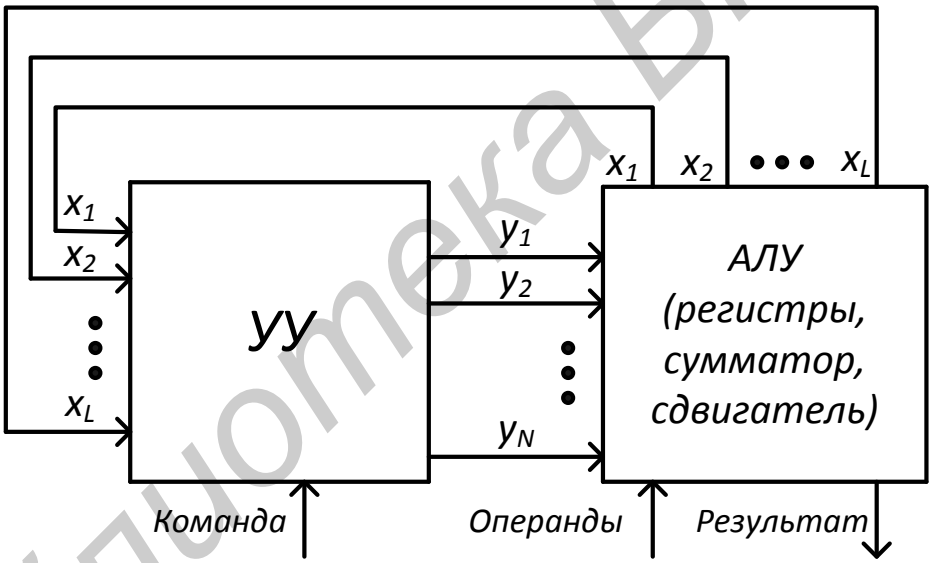


Рис. 2.2. Организация процессора

АЛУ вместе с УУ составляют процессор (либо сопроцессор). Выполняемая операция задается кодом операции команды, поступающей в УУ. Последовательность выполнения микроопераций в АЛУ для реализации конкретной машинной операции (умножения, деления, CORDIC-алгоритма и др.) определяется алгоритмом выполнения этой операции. Алгоритмы выполнения операций в АЛУ чаще всего задаются в форме *микропрограмм*, называемых также *граф-схемами алгоритмов* (ГСА).

ГСА – это ориентированный связный граф, содержащий вершины четырех типов: начальную, конечную, операторную и условную (рис. 2.3).

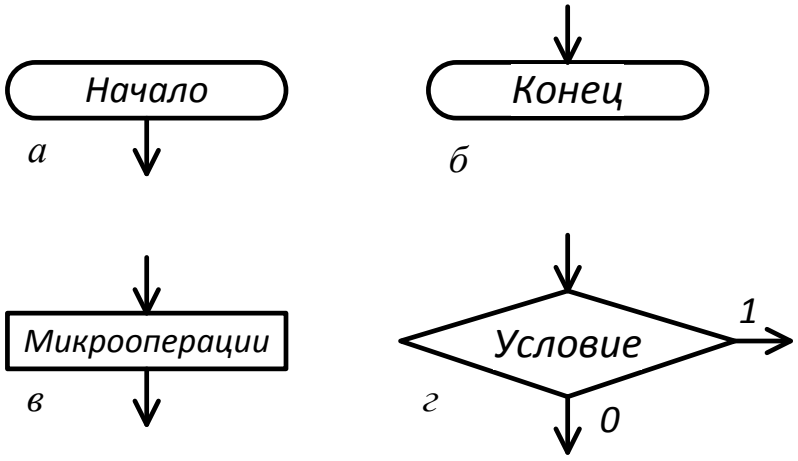


Рис. 2.3. Типы вершин ГСА:
а – начальная; *б* – конечная; *в* – операторная; *г* – условная

При составлении ГСА необходимо руководствоваться следующими правилами:

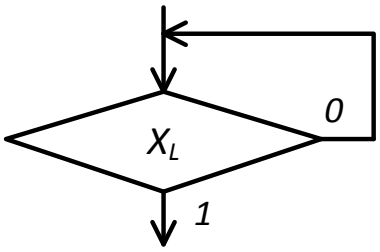


Рис. 2.4. Ждущая условная вершина

1) ГСА должна содержать одну начальную, одну конечную и конечное множество операторных и условных вершин;

2) входы и выходы различных вершин соединяются дугами, направленными от выхода ко входу;

3) каждый выход с одним входом;

4) для любой вершины ГСА существует по крайней мере один путь из этой вершины, проходящей через операторные и условные вершины в направлении соединяющих их дуг;

5) в каждой операторной вершине записывается множество микроопераций, составляющих микрокоманду Y_t, являющуюся подмножеством микроопераций $Y = \{y_1, y_2, \dots, y_N\}$; пустая микрокоманда обозначается символом Y_0;

6) в каждой условной вершине записывается один из элементов множества логических условий $X = \{x_1, x_2, \dots, x_L\}$;

7) разрешается соединять один из выходов условной вершины с её входом, что недопустимо для операторной вершины; такая вершина называется *ждущей условной вершиной* (рис. 2.4); с её помощью можно описывать ожидание в работе дискретных устройств.

2.2.2. Основные понятия о синтезе устройств управления (УУ)

Устройства управления характеризуются некоторым числом внутренних состояний. В каждый момент времени оно может находиться в одном из этих

состояний. Под действием входных сигналов УУ переходит из одного состояния в другое. Новое состояние, в которое устанавливается автомат, зависит от комбинации действующих на его входах сигналов и предшествующего состояния, в котором находился автомат. Выходные сигналы автомата определяются либо только его внутренним состоянием, либо внутренним состоянием и комбинацией входных сигналов.

Функционирование автомата можно описать, используя три множества:

– входных сигналов $x_1, x_2, \dots, x_L$;

– внутренних состояний $a_1, a_2, \dots, a_K$;

– выходных сигналов $y_1, y_2, \dots y_N$.

Одно из состояний (a_0) является начальным. В него устанавливается автомат перед началом его работы.

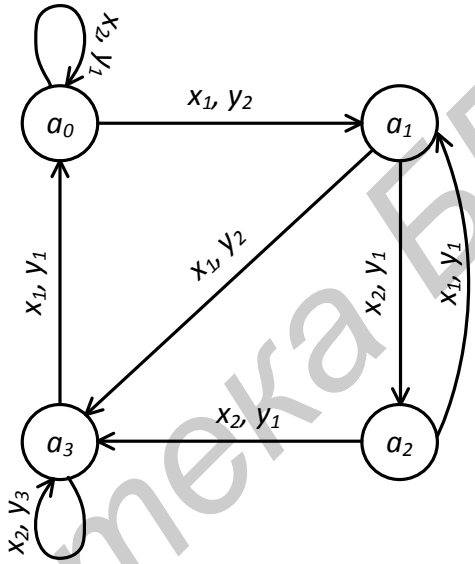


Рис. 2.5. Граф автомата

Работа автомата описывается:

1) функцией переходов f, которая определяет состояние автомата $a\ t+1$ в момент времени $t+1$ в зависимости от его состояния $a(t)$ и значения входного сигнала $x(t)$, в момент времени t: $a\ t+1\ = f[a\ t\ , x(t)]$;

2) функцией выходов φ, определяющей зависимость выходного сигнала автомата $y\ t$ от состояния автомата $a\ t$ и входного сигнала $x\ t$:

$$y\ t\ = \varphi[a\ t\ , x(t)].$$

Автомат с функцией выходов, определенной таким образом, называется *автоматом Мили*; Другой тип автомата – *автомат Мура*. Особенность автомата Мура состоит в том, что в нем

Входной сигнал	*Состояние*			
	a_0	a_1	a_2	a_3
x_1	a_1/y_2	a_3/y_2	a_1/y_1	a_0/y_1
x_1	a_0/y_1	a_2/y_1	a_3/y_1	a_3/y_3

Рис. 2.6. Таблица переходов автомата

выходной сигнал зависит от внутреннего состояния $a(t)$ и не зависит от входного сигнала. Функции переходов и выходов автомата Мура имеют вид:

$$a\ t+1\ = f\ a\ t\ , x\ t\ \ , \qquad y(t) = \varphi\ a\ t\ \ .$$

Функции переходов и выходов автомата могут быть заданы в форме таблиц переходов и выходов либо с помощью графов. В практических разработках граф автомата строится на основе ГСА.

2.2.3. Структура АЛУ сопроцессора CORDIC-алгоритма

АЛУ, рассматриваемое в данном разделе, выполняет арифметические операции и операции сдвига над числами с фиксированной запятой, формат которых представлен на рис. 2.7.

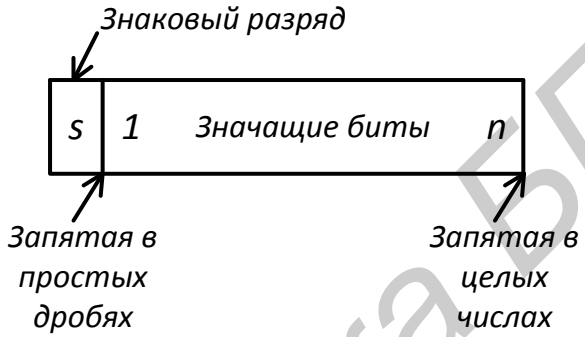


Рис. 2.7. Формат числа с фиксированной запятой

Структурная схема АЛУ сопроцессора CORDIC-алгоритма показана на рис. 2.8.

В состав АЛУ входят пять регистров: P1 и P2 – хранят промежуточные значения координат вектора, в регистры P3 и P4 записываются результаты вычисления CORDIC-алгоритма после масштабирования, P5 хранит закодированную последовательность, задающую угол поворота. Кроме регистров в АЛУ находятся два сумматора (SM), два сдвигателя (СД), а также четыре мультиплексора (MUX) и схема масштабирования.

Поскольку сопроцессор CORDIC-алгоритма предназначен для совместной работы с RISC-процессором Plasma, его интерфейс должен быть совместим с указанным процессором:

```
entity CORDIC_coprocessor is
    Port ( address: in  STD_LOGIC_VECTOR (31 downto 0);
           we     : in  STD_LOGIC;
           clk    : in  STD_LOGIC;
           data_w : in  STD_LOGIC_VECTOR (31 downto 0);
           data_r : out  STD_LOGIC_VECTOR (31 downto 0));
end CORDIC_coprocessor;
```

Со стороны RISC-процессора сопроцессор CORDIC-алгоритма является портом ввода/вывода, отображенным на адресное пространство памяти. В нача-

ле работы RISC-процессор должен обратиться к регистру состояния сопроцессора CORDIC по адресу $80000810h$, если в младшем прочитанном разряде находится «1», то сопроцессор занят, иначе можно начать работу с сопроцессором.

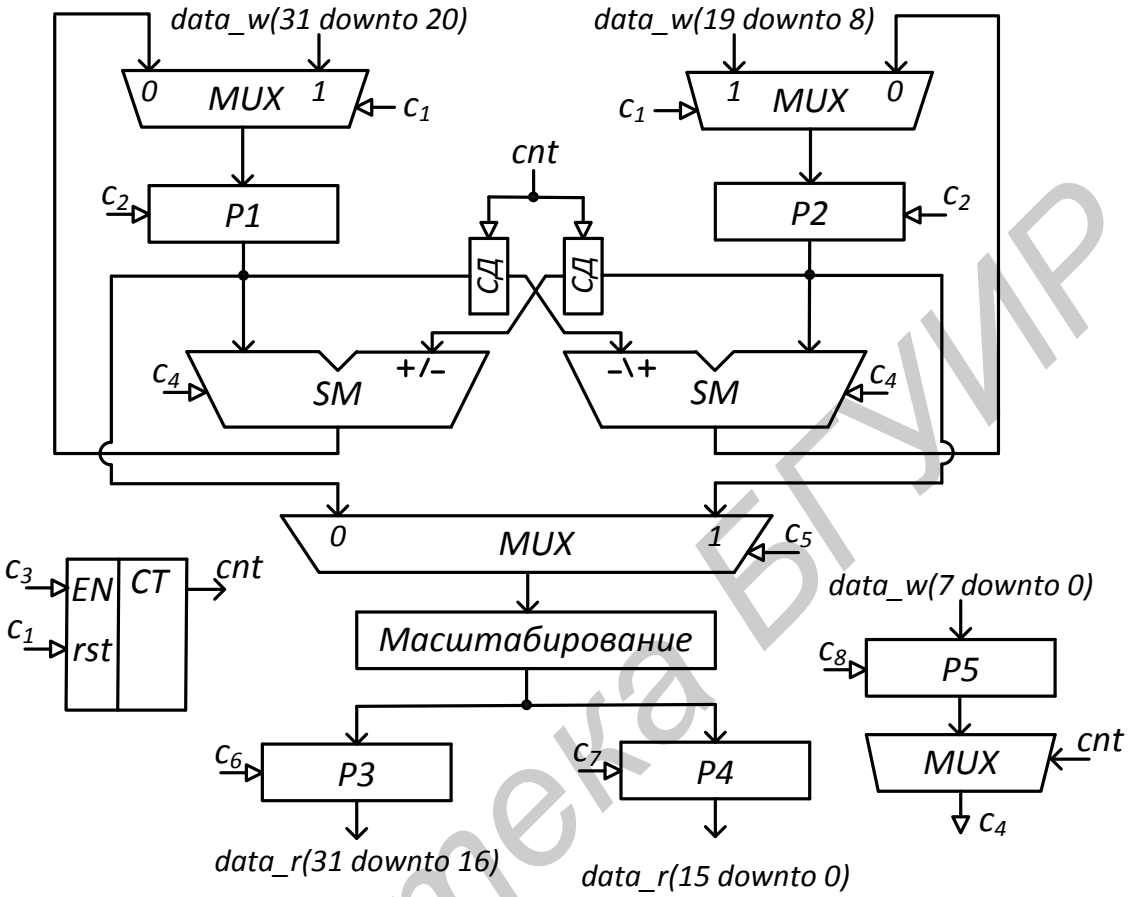


Рис. 2.8. АЛУ сопроцессора CORDIC

Для запуска сопроцессора необходимо записать в регистр данных сопроцессора по адресу $80000800h$ входную информацию в следующем виде:

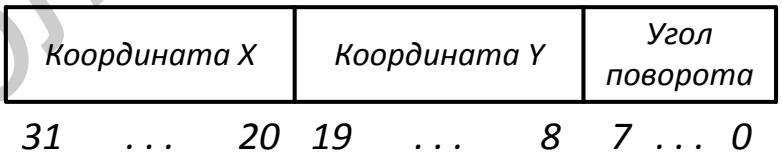


Рис. 2.9. Формат данных, передаваемых
в сопроцессор CORDIC-алгоритма

Предполагается, что координаты передаются в формате с фиксированной запятой (в дополнительном коде) в виде дроби, при этом под знак отводится один бит, а под дробную часть 11 бит. Как известно, последовательность, определяющая угол поворота, содержит либо «1», либо «–1». В данном случае «1» кодируется единицей, а «–1» кодируется нулем.

Так как для кодирования угла отводится только 8 бит, то, следовательно, используется 8 базовых углов в алгоритме CORDIC. Из этого следует, что масштабирование выходного результата можно выполнить так, как это показано в примере в разд. 2.1.

На основании структурной схемы на рис. 2.8 можно составить VHDL-описание АЛУ сопроцессора CORDIC.

```
--        Описание сигналов
-- Выходные сигналы мультиплексоров (MUX)
signal mux_X, mux_Y : std_logic_vector(15 downto 0);
-- Выходные сигналы сумматоров (SM)
signal add_X, add_Y : std_logic_vector(15 downto 0);
-- Выходные регистров P1 и P2
signal reg_X, reg_Y : std_logic_vector(15 downto 0);
-- Выходные сигналы сдвигателей (СД)
signal shf_X, shf_Y : std_logic_vector(15 downto 0);
-- Выходной сигнал мультиплексора для блока масштабирования
signal mux_sc       : std_logic_vector(15 downto 0);
-- Промежуточные сигналы для блока масштабирования
signal sc_tmp_1     : std_logic_vector(15 downto 0);
signal sc_tmp_3     : std_logic_vector(15 downto 0);
signal sc_tmp_6     : std_logic_vector(15 downto 0);
signal sc_tmp_9     : std_logic_vector(15 downto 0);
signal sc_tmp_13    : std_logic_vector(15 downto 0);
-- Выход блока масштабирования
signal sc_output    : std_logic_vector(15 downto 0);
-- Выходные регистры координат X и Y
signal reg_out_X    : std_logic_vector(15 downto 0);
signal reg_out_Y    : std_logic_vector(15 downto 0);
signal reg_angle    : std_logic_vector(7  downto 0);
...
---------------- АЛУ сопроцессора CORDIC ----------------
--.............Мультиплексор для координаты X
mux_X <= data_w(31)&data_w(31)&data_w(31)&data_w(31)&data_w(31 downto 20)
          when c1='1' else add_X;

--.............Мультиплексор для координаты Y
mux_Y <= data_w(19)&data_w(19)&data_w(19)&data_w(19)&data_w(19 downto 8)
          when c1='1' else add_Y;

--.............Регистры X и Y (на схеме P1 и P2)
process (clk)
begin
     if rising_edge(clk) then
          if (c2='1') then
               reg_X <= mux_X;
               reg_Y <= mux_Y;
          end if;
     end if;
end process;

--.............Регистр угла поворота (на схеме P5)
process (clk)
begin
```

```
     if rising_edge(clk) then
          if (c8='1') then
               reg_angle<= data_w(7 downto 0);
          end if;
     end if;
end process;

--.............Сдвигатель для регистра X
shf_X <=reg_X  when cnt="000" else
     reg_X(15)&reg_X(15 downto 1) when cnt="001" else
     reg_X(15)&reg_X(15)&reg_X(15 downto 2) when cnt="010" else
     reg_X(15)&reg_X(15)&reg_X(15)&reg_X(15 downto 3) when cnt ="011"
else
     reg_X(15)&reg_X(15)&reg_X(15)&reg_X(15)&reg_X(15 downto 4) when
cnt="100" else
     reg_X(15)&reg_X(15)&reg_X(15)&reg_X(15)&reg_X(15)&reg_X(15 downto
5) when cnt="101" else
     reg_X(15)&reg_X(15)&reg_X(15)&reg_X(15)&reg_X(15)&reg_X(15)&reg_X(
15 downto 6) when cnt="110" else
     reg_X(15)&reg_X(15)&reg_X(15)&reg_X(15)&reg_X(15)&reg_X(15)&reg_X(
15)&reg_X(15 downto 7);

--.............Сдвигатель для регистра Y
shf_Y <=reg_Y when cnt="000" else
     reg_Y(15)&reg_Y(15 downto 1) when cnt="001" else
     reg_Y(15)&reg_Y(15)&reg_Y(15 downto 2) when cnt="010" else
     reg_Y(15)&reg_Y(15)&reg_Y(15)&reg_Y(15 downto 3)when cnt="011" else
     reg_Y(15)&reg_Y(15)&reg_Y(15)&reg_Y(15)&reg_Y(15 downto 4) when
cnt="100" else
     reg_Y(15)&reg_Y(15)&reg_Y(15)&reg_Y(15)&reg_Y(15)&reg_Y(15 downto
5) when cnt="101" else
     reg_Y(15)&reg_Y(15)&reg_Y(15)&reg_Y(15)&reg_Y(15)&reg_Y(15)&reg_Y(1
5 downto 6) when cnt="110" else
     reg_Y(15)&reg_Y(15)&reg_Y(15)&reg_Y(15)&reg_Y(15)&reg_Y(15)&reg_Y(1
5)&reg_Y(15 downto 7);

--.............Сумматоры (на схеме SM)
process (shf_X, shf_Y, reg_X, reg_Y, c4)
begin
   if c4 = '1' then                --    1 coded as 1; -1 coded as 0
      add_X <= reg_X - shf_Y;
      add_Y <= reg_Y + shf_X;
   else
      add_X <= reg_X + shf_Y;
      add_Y <= reg_Y - shf_X;
   end if;
end process;

--.............Мультиплексор выбора направления поворота
y4 <= reg_angle(7) when cnt="000" else
     reg_angle(6) when cnt="001" else
     reg_angle(5) when cnt="010" else
     reg_angle(4) when cnt="011" else
     reg_angle(3) when cnt="100" else
     reg_angle(2) when cnt="101" else
     reg_angle(1) when cnt="110" else
```

```
     reg_angle(0);

--.............Мультиплексор для выполнения масштабирования
mux_sc <= reg_X when c5='0' else
                 reg_Y;

--.............Блок масштабирования
--  sc = 2^-1 + 2^-3 - 2^-6 - 2^-9 - 2^-13
sc_tmp_1 <= mux_sc(15)&mux_sc(15 downto 1);
sc_tmp_3 <= mux_sc(15)&mux_sc(15)&mux_sc(15)&mux_sc(15 downto 3);
sc_tmp_6 <= mux_sc(15)&mux_sc(15)&mux_sc(15)&mux_sc(15)&mux_sc(15)&
mux_sc(15)&mux_sc(15 downto 6);
sc_tmp_9 <= mux_sc(15)&mux_sc(15)&mux_sc(15)&mux_sc(15)&mux_sc(15)&
mux_sc(15)&mux_sc(15)&mux_sc(15)&mux_sc(15)&mux_sc(15 downto 9);
sc_tmp_13 <= mux_sc(15)&mux_sc(15)&mux_sc(15)&mux_sc(15)&mux_sc(15)&
mux_sc(15)&mux_sc(15)&mux_sc(15)&mux_sc(15)&mux_sc(15)&mux_sc(15)&
mux_sc(15)&mux_sc(15)&mux_sc(15 downto 13);

sc_output <= sc_tmp_1 + sc_tmp_3 - sc_tmp_6 - sc_tmp_9 - sc_tmp_13;

--.............Выходные регистры для координат X и Y (на схеме P3 и P4)
process (clk)
begin
     if rising_edge(clk) then
          if (c6='1') then
               reg_out_X <= sc_output;
          end if;
          if (c7='1') then
               reg_out_Y <= sc_output;
          end if;
     end if;
end process;

--.............Счетчик циклов
process(clk)
begin
     if rising_edge(clk) then
          if (c1='1') then           -- сброс счетчика
               cnt<=(others=>'0');
          elsif c3='1' then          -- увеличение счетчика на единицу
               cnt<=cnt+1;
          end if;
     end if;
end process;
```

2.2.4. Управляющее устройство сопроцессора CORDIC-алгоритма

Исходной информацией для построения управляющего устройства сопроцессора служит микропрограмма выполнения CORDIC-алгоритма в разрабатываемом сопроцессоре (рис. 2.10).

Микрооперации и логические условия в микропрограмме означают следующее:

Микрооперации

1) P1 = data_w(31…20) — загрузка координаты X в регистр P1;

2) P2 = data_w(19…8) — загрузка координаты Y в регистр P2;

3) P3 = data_w(7…0) — загрузка последовательности, определяющей поворот в регистр P3;

4) cnt = 0 — сброс счетчика;

5) P1 = P1 ± (P2>>cnt) — сложение/вычитание регистра P1 со сдвинутым на cnt разрядов вправо содержимым регистра P2, запись результата в регистор P1;

6) P2 = P2 ± (P1>>cnt) — сложение/вычитание регистра P2 со сдвинутым на cnt разрядов вправо содержимым регистра P1, запись результата в регистор P2;

7) cnt = cnt +1 — увеличение содержимого счетчика на единицу;

8) P3 = scale (P1) — выполнение операции масштабирования для координаты X;

9) P4 = scale (P2) — выполнение операции масштабирования для координаты Y.

Логические условия

1) получение запроса на обработку;

2) cnt==7 — содержимое счетчика равно 7;

3) P5[cnt]==0 — значение бита с номером cnt в регистре P5.

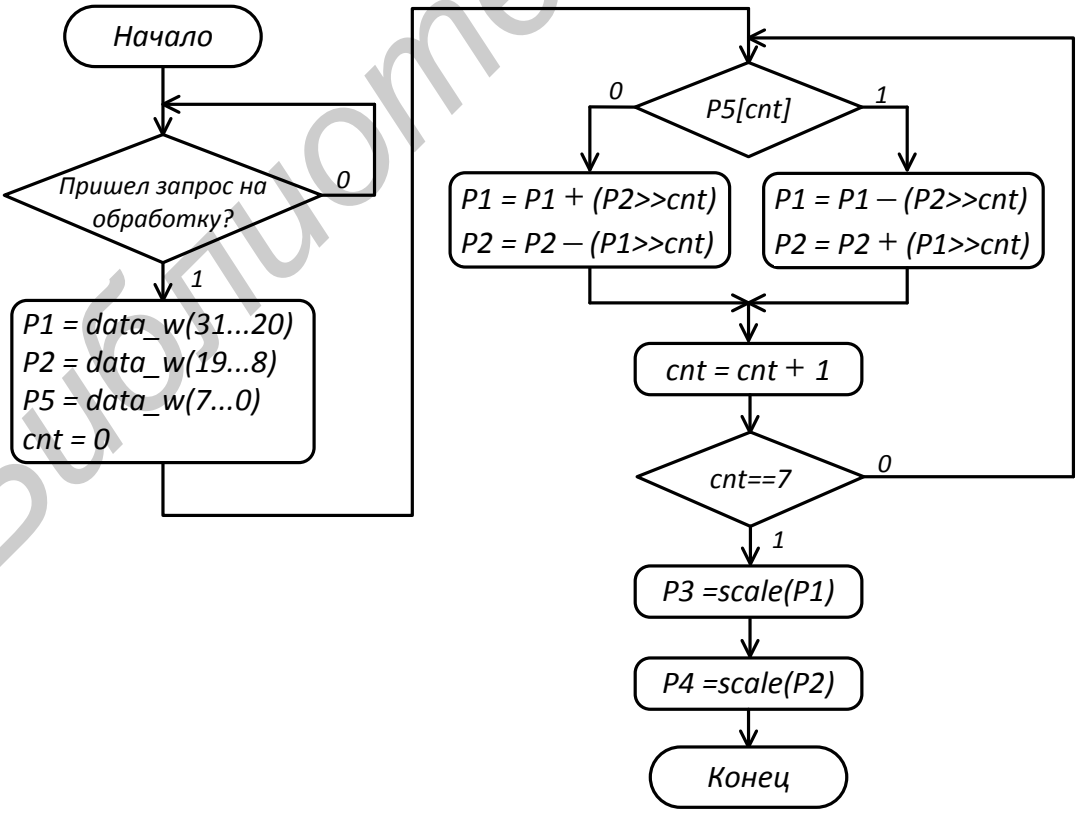


Рис. 2.10. Микропрограмма работы сопроцессора CORDIC-алгоритма

Далее заменим в микропрограмме на рис. 2.10 логические условия символами $x_1, \ldots, x_L$, а микрооперации символами $y_1, \ldots, y_N$.

Логические условия

x_1: *Получение запроса на обработку.*
x_2: *P5[cnt]*
x_3: *cnt==7.*

Микрооперации

y_1: *P1 = data_w(31…20)*
y_2: *P2 = data_w(19…8)*
y_3: *P5 = data_w(7…0)*
y_4: *cnt = 0*
y_5: *P1 = P1 + (P2>>cnt)*
y_6: *P1 = P1 – (P2>>cnt)*
y_7: *P2 = P2 + (P1>>cnt)*
y_8: *P2 = P2 – (P1>>cnt)*
y_9: *cnt = cnt +1*
y_{10}: *P3 = scale (P1)*
y_{11}: *P4 = scale (P2)*

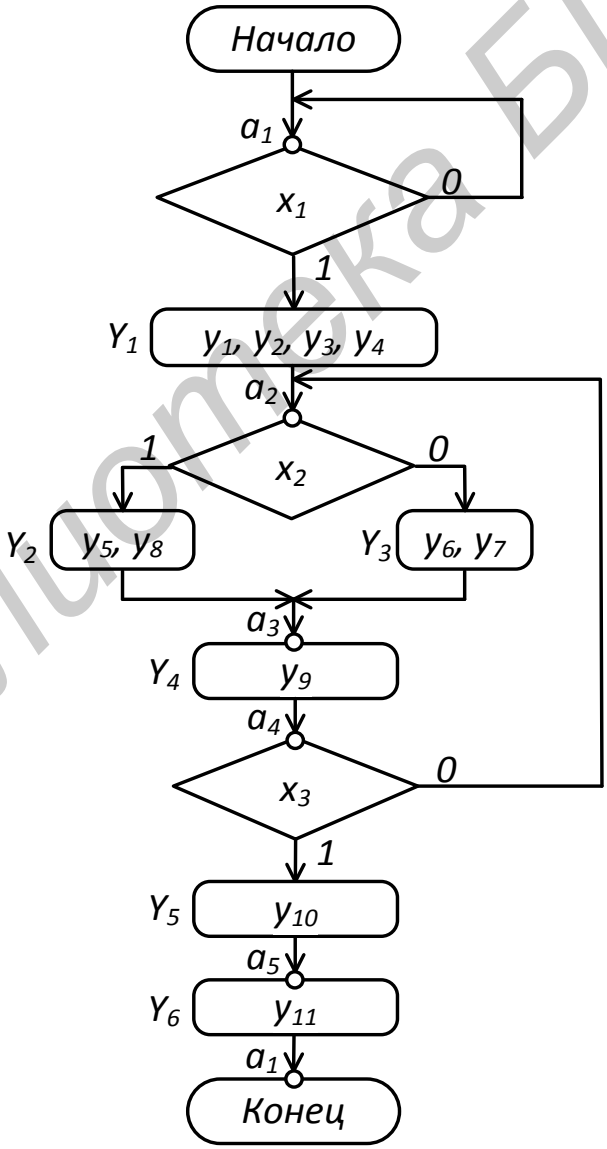


Рис. 2.11. Граф-схема алгоритма

После замены в микропрограмме логических условий и микроопераций получим граф-схему алгоритма (ГСА), изображенную на рис. 2.11.

В каждой условной вершине ГСА записывается один из элементов логических условий $X = \{x_1, \dots, x_L\}$, а в каждой операторной вершине записывается *оператор* (микрокоманда) Y_t – подмножество множества микроопераций $Y = \{y_1, \dots, y_N\}$.

Рассмотрим синтез микропрограммного автомата (МПА) Мили. На первом шаге в ГСА выделяются состояния МПА по следующим правилам:

1) вход вершины, следующей за начальной, а также вход конечной вершины отмечаются символом a_1;

2) входы вершин, следующих за операторными, отмечаются символами $a_2, a_3, \dots, a_M$.

Во втором правиле не оговаривается, вход какой вершины отмечается символом a_m. Это могут быть как условные вершины (a_1, a_3, a_5 на рис. 2.11), так и операторные вершины (a_2, a_4, a_6). Важно только, чтобы отмечаемая вершина следовала за операторной вершиной.

На втором шаге синтеза строится таблица переходов МПА (табл. 2.1). Таблица включает пять столбцов: a_m и a_s – исходное состояние и состояние перехода; X_h и Y_t – входной и выходной сигналы на переходе автомата из состояния a_m в состояние a_s соответственно.

На третьем шаге синтеза для каждой микрооперации определяется набор управляющих сигналов, который обеспечивает ее выполнение. Например, для выполнения микрооперации y_1 (загрузка данных в регистр P1) необходимо установить c_1 в единицу (управление мультиплексором) и c_2 также в единицу (разрешение записи в регистр P1). В табл. 2.2 приведены наборы управляющих сигналов для всех микроопераций.

Таблица 2.1

Переходы МПА

a_m	a_s	X_h	Y_t
a_1	a_1	x_1	–
	a_2	x_1	y_1, y_2, y_3, y_4 Y_1
a_2	a_3	x_2	y_5, y_8 Y_3
		x_2	y_6, y_7 Y_2
a_3	a_4	1	y_9 Y_4
a_4	a_2	x_3	–
	a_5	x_3	y_{10} Y_5
a_5	a_1	1	y_{11} Y_6

Таблица 2.2

Управляющие сигналы

Микрооперация	**Значение**	**Управляющие сигналы**						
		c_1	c_2	c_3	c_5	c_6	c_7	c_8
y_1	*P1 = data_w(31…20)*	1	1	0	0	0	0	0
y_2	*P2 = data_w(19…8)*	1	1	0	0	0	0	0
y_3	*P5 = data_w(7…0)*	0	0	0	0	0	0	1
y_4	*cnt = 0*	1	0	0	0	0	0	0
y_5, y_6	*P1 = P1 ± (P2>>cnt)*	0	1	0	0	0	0	0
y_7, y_8	*P1 = P1 ± (P2>>cnt)*	0	1	0	0	0	0	0
y_9	*cnt = cnt +1*	0	0	1	0	0	0	0
y_{10}	*P3 = scale (P1)*	0	0	0	0	1	0	0
y_{11}	*P4 = scale (P2)*	0	0	0	1	0	1	0

Имея в распоряжении таблицу переходов МПА и таблицу управляющих сигналов, можно составить VHDL-описание устройства управления сопроцессором CORDIC-алгоритма.

```
--.............ДЕКЛАРАЦИЯ СИГНАЛОВ
-- Состояния автомата управления
constant a1   : STD_LOGIC_VECTOR (2 downto 0):= "000";
constant a2   : STD_LOGIC_VECTOR (2 downto 0):= "001";
constant a3   : STD_LOGIC_VECTOR (2 downto 0):= "010";
constant a4   : STD_LOGIC_VECTOR (2 downto 0):= "011";
constant a5   : STD_LOGIC_VECTOR (2 downto 0):= "111";

-- Управляющие сигналы
signal c1  : std_logic:='0';    -- управление мультиплексорами
signal c2  : std_logic:='0';    -- управление регистрами
signal c3  : std_logic:='0';    -- сигнал управления счетчиком (cnt++)
signal c4  : std_logic:='0';    -- управление сумматорами/вычитателями
signal c5  : std_logic:='0';    -- управление мультиплексором
signal c8  : std_logic:='0';    -- управление регистром, хранящим угол.
signal c6  : std_logic:='0';    -- управление записью в регистр P3
signal c7  : std_logic:='0';    -- управление записью в регистр P4
signal ff_busy  : std_logic:='0';     -- флаг занятости
signal ff_busy_rst  : std_logic:='0'; -- сброс флага занятости
signal cnt : std_logic_vector (2 downto 0):="000";
signal state, next_state : std_logic_vector(2 downto 0):="000";

--.............ОПИСАНИЕ АРХИТЕКТУРЫ
--.............Флаг занятости
busy: process(clk)
begin
     if rising_edge(clk) then
          if (c1 = '1') then
               ff_busy <= '1';       -- установка флага занятости
          elsif (ff_busy_rst='1') then
               ff_busy <= '0';
          end if;
```

```
     end if;
end process;
--.............Управляющий автомат (Мили)
-- Память автомата
SYNC_PROC: process (clk)
begin
     if rising_edge(clk) then
          if (reset = '1') then
               state <= a1;
          else
               state <= next_state;
          end if;
     end if;
end process;

--Декодирование выходных сигналов
OUTPUT_DECODE: process (state,we,address)
begin
case (state) is
     when a1=>
     -- Проверка условия x1
     if address = x"80000800" and we = '1' and ff_busy = '0' then
     -- управление входными мультиплексорами и сброс счетчика cnt
          c1 <= '1';
          c2 <= '1';      -- запись данных в регистры P1/P2
          c8 <= '1';      -- запись данных в регистр угла поворота (P5)
     else
          c1 <= '0';      c2 <= '0';
          c8 <= '0';
     end if;
     c3 <= '0';
     -- Подключаем к блоку масштабирования регистр P1
     c5 <= '0';      c6 <= '0';
     c7 <= '0';      ff_busy_rst<='0';

     when a2=>
     -- управление входными мультиплексорами
     c1 <= '0';
     -- запись результата сложения/вычитания в регистры P1/P2
     c2 <= '1';
     c3 <= '0'; c5 <= '0';
     c6 <= '0'; c7 <= '0';
     c8 <= '0'; ff_busy_rst<='0';

     when a3=>
     c1 <= '0'; c2 <= '0';
     c3 <= '1';                  -- увеличиваем счетчик (cnt) на единицу
     c5 <= '0'; c6 <= '0';
     c7 <= '0'; c8 <= '0';
     ff_busy_rst<='0';

     when a4=>
     if (cnt="111") then
          c6 <= '1';             -- запись результата в регистр P3
     else
          c6 <= '0';
```

```
      end if;
      c1 <= '0'; c2 <= '0';
      c3 <= '0'; c5 <= '0';
      c7 <= '0'; c8 <= '0';
      ff_busy_rst<='0';

      when a5=>
      c1 <= '0'; c2 <= '0';
      c3 <= '0';
      c5 <= '1';       -- подключаем регистр P2 к блоку масштабирования
      c6 <= '0';       c8 <= '0';
      c7 <= '1';       -- запись результата в регистр P4
      ff_busy_rst<='1';

      when others=>
      c1 <= '0'; c2 <= '0';
      c3 <= '0'; c5 <= '0';
      c6 <= '0'; c7 <= '0';
      c8 <= '0'; ff_busy_rst<='0';
end case;
end process;

NEXT_STATE_DECODE: process (state,we,address,cnt)
begin
      next_state <= state;  -- по-умолчанию остаемся в том же состоянии
      case (state) is
            when a1 =>
            if address = x"80000800" and we = '1' and ff_busy = '0' then
                  next_state <= a2;
            end if;

            when a2 =>       next_state <= a3;

            when a3 =>       next_state <= a4;


            when a4 =>
            -- Если не выполнили все микроповороты - продолжаем.
            if (cnt="111") then
                  next_state <= a5;
            else
                  next_state <= a2;
            end if;

            when a5 =>       next_state <= a1;

            when others =>
                  next_state <= a1;
            end case;
end process;
-- выдача на шину либо сигнала занятости, либо выходных данных
data_r<= reg_out_X&reg_out_Y when ((address = x"80000800" ) and (we='0'))
else
      (x"0000000"&"000"&ff_busy) when ((address = x"80000810" ) and
(we='0'))
 else "ZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZ";
```

3. Реализация алгоритмов ЦОС на основе арифметики с фиксированной запятой переменного формата

При проектировании портативных цифровых систем, работающих в реальном масштабе времени, особое внимание уделяется таким показателям, как скорость вычислений, аппаратные затраты, сложность алгоритмов выполнения арифметических операций. По этой причине большинство устройств такого класса выполняются на основе арифметики с фиксированной запятой, которая позволяет получить хорошие результаты по перечисленным показателям.

Процесс реализации алгоритма на основе арифметики с фиксированной запятой в общем случае может быть представлен в виде граф-схемы, изображенной на рис. 3.1.

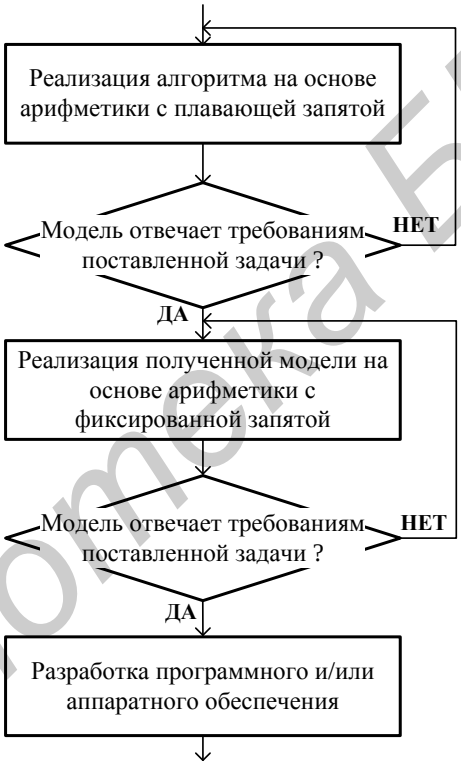


Рис. 3.1. Этапы проектирования цифровых устройств на основе арифметики с фиксированной запятой

Как видно из рисунка, первый этап проектирования заключается в построении модели, использующей арифметику с плавающей запятой. Данный этап позволяет разработчику исследовать структуру и особенности самого алгоритма без учета эффектов квантования коэффициентов и данных, и если модель удовлетворяет требованиям ТЗ, можно смело переходить на следующую ступень – реализацию алгоритма на основе арифметики с фиксированной запятой. Здесь перед разработчиком возникает ряд вопросов касательно выбора формата данных, разрядности вычислений, схемы округления результата. От успешного нахождения ответов на данные вопросы во многом зависит положи-

тельный результат проектирования программного и/или аппаратного обеспечения цифрового устройства.

В данном разделе рассмотрен подход к реализации алгоритмов ЦОС на основе арифметики с фиксированной запятой переменного формата. В соответствии с этим подходом производится анализ алгоритма в каждом узле и последующее масштабирование данных с целью сохранения максимального динамического диапазона. Использование масштабирования приводит к тому, что на различных участках алгоритма данные имеют неодинаковый формат. Данное обстоятельство определяет некоторые особенности организации вычислений, о чем изложено ниже. В заключение раздела будет рассмотрен пример реализации сопроцессора для организации вычислений в алгоритмах на арифметике с переменным форматом.

3.1. Представление чисел в арифметике с фиксированной запятой переменного формата

Любое число, представленное в арифметике с фиксированной запятой (ФЗ) переменного формата, задается в виде выражения:

$$a = ma \cdot 2^{exp_a}, \quad \text{где } ma = -1^{s} + \sum_{i=0}^{wl-2} a_i \cdot 2^{i-wl+1}. \qquad (3.1)$$

Здесь ma – само значение числа, представленное в дополнительном коде, интерпретируемое как дробное в диапазоне $-1, 1$; a_i – это значения соответствующих битов числа, равные 0 либо 1; s – знаковый бит; wl – разрядность слова (рис. 3.2); exp_a – это порядок масштабирующего множителя 2^{exp_a}, который в общем случае для переменной $a \in [a_{\min} a_{\max}]$ можно определить как

$$exp_a = \log_2 \max(\, a_{\min} \,,\, a_{\max} \,) \,. \qquad (3.2)$$

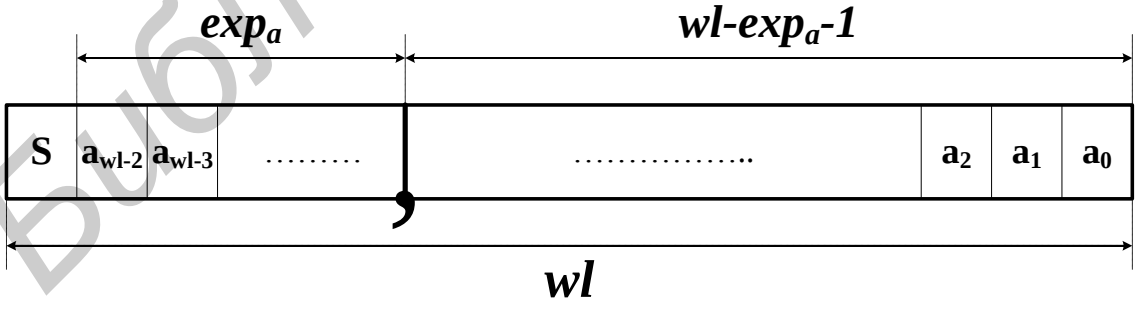


Рис. 3.2. Представление числа в арифметике с фиксированной запятой

Таким образом, в каждом узле алгоритма можно определить конкретный формат значений переменных и констант, после чего можно переходить к реализации заданных вычислений.

3.2. Выполнение математических операций в арифметике переменного формата

Пусть на определенном участке алгоритма необходимо произвести сложение двух входных переменных, заданных в следующих форматах: $a \ \ wl, exp_a$, $b \ \ wl, exp_b$, где wl – длина слова, а exp_a, exp_b – число битов, отведенных под целую часть, при этом пусть $exp_b \geq exp_a$, тогда результат суммы с учетом (3.1) можно записать как

$$c = a + b = mc \cdot 2^{exp_c} = \ ma \cdot 2^{exp_a - exp_b} + mb \ \cdot 2^{exp_b}. \qquad (3.3)$$

В данной записи умножение на $2^{exp_a - exp_b}$ соответствует арифметическому сдвигу вправо для «выравнивания» битов с одинаковыми весами в операндах ma и mb (рис. 3.3). Для большинства цифровых систем длина слова wl является фиксированной, поэтому младшие $exp_b - exp_a$ бит числа a в результате арифметического сдвига отбрасываются и не участвуют в операции сложения.

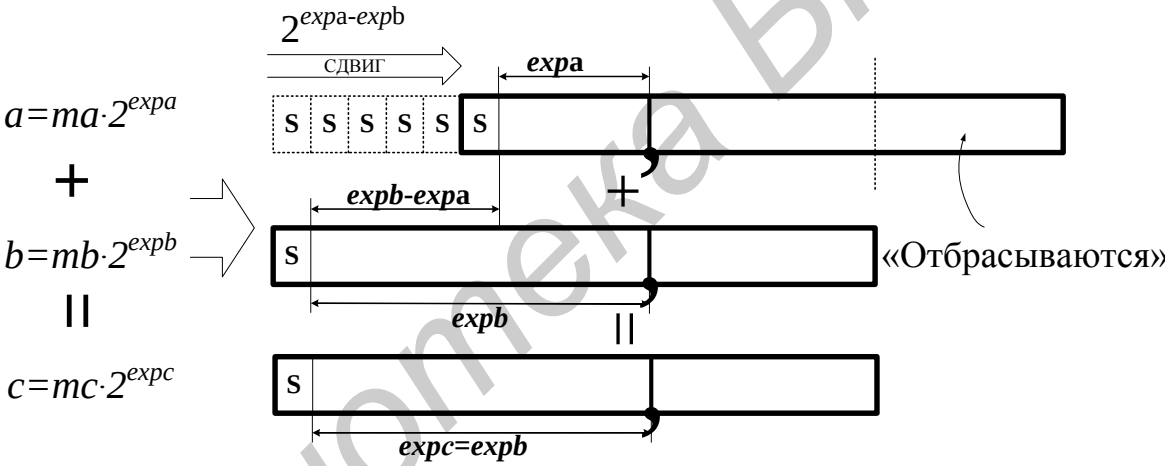


Рис. 3.3. Сложение чисел, заданных в разных форматах

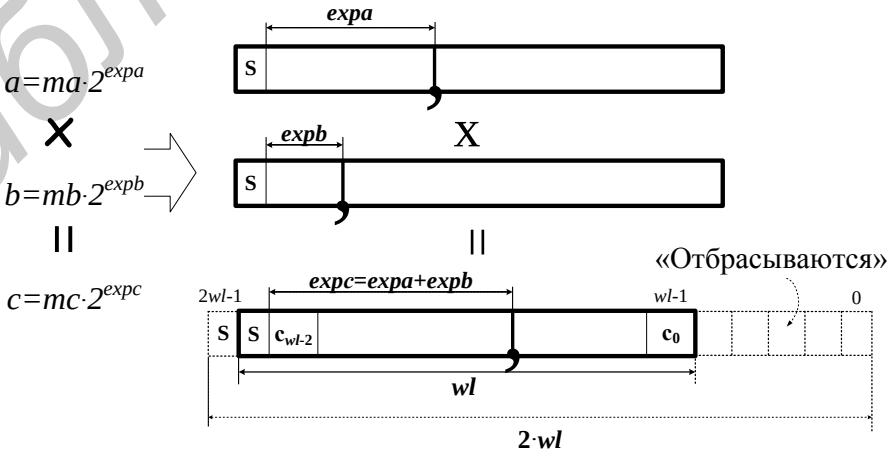


Рис. 3.4. Умножение чисел, заданных в разных форматах

Теперь рассмотрим особенности умножения чисел, представленных в различных форматах. В соответствии с (3.1) результат произведения двух операндов заданных в различных форматах определяется как

$$c = mc \cdot 2^{exp_c} = ma \cdot mb \cdot 2^{exp_a + exp_b}. \quad (3.4)$$

Так как значения $ma, mb \in -1,1$, то операцию $ma \cdot mb$ можно выполнять по правилам умножения дробных чисел, представленных в дополнительном коде. Тогда результат умножения будет иметь длину слова, равную $2 \cdot wl$, при этом первый значащий бит сместится на одну позицию вправо. Таким образом, корректное значение mc для выбранной длины слова wl будет расположено в разрядах, начиная с $wl - 1$ и по $2wl - 2$ (рис. 3.4).

Для лучшего усвоения материала рассмотрим простой пример.

Пример 1. *Дан следующий участок алгоритма:*

$$c = a + b,$$
$$e = c \cdot d,$$

где $a \in [-0{,}25\ 0{,}75]$, $b \in [-1{,}25\ 0{,}5]$*– переменные,* $d = 1{,}5$ *– константа, разрядность слов 4 бита, вычисления выполняются в дополнительном коде. Необходимо реализовать данный алгоритм на основе арифметики переменного формата.*

<u>Решение</u>. Рассчитаем значения степеней масштабирующих множителей для переменных a, b, c, e*:*

$$exp_a = \log_2(\max\ a\ ) = \log_2(0.75) = 0,$$
$$exp_b = \log_2(\max\ b\ ) = \log_2(1.25) = 1,$$
$$exp_c = \max(exp_a, exp_b) = 1,$$
$$exp_e = exp_c + exp_d = 1 + \log_2(1.5) = 2.$$

Из рассчитанных значений видим, что для корректного выполнения операции сложения необходимо осуществить арифметический сдвиг вправо переменной a*, так как ее значение степени масштабирующего множителя меньше, чем у переменной* b *. В итоге с учетом требований к формату представления данных получаем схему для вычисления алгоритма, изображенную на рис. 3.5.*

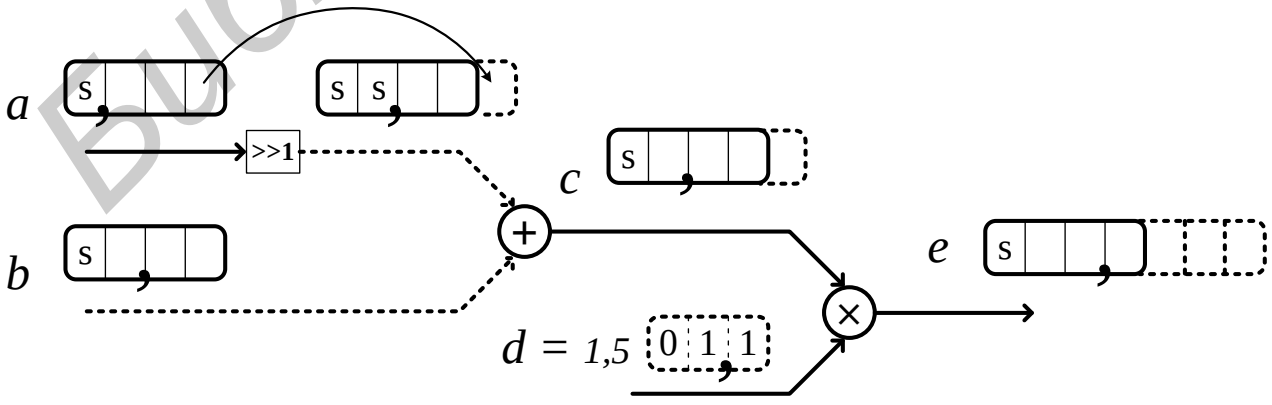


Рис. 3.5. Схема реализации алгоритма на арифметике с фиксированной запятой переменного формата

Как видно из рисунка, при использовании фиксированной разрядности чисел возникают ошибки вычислений при арифметическом сдвиге и усечении (либо округлении) после операции умножения (пунктирная линия на рис. 3.5), поэтому при реализации алгоритмов, как правило, вводится дополнительный анализ точности вычислений и, если необходимо, увеличивается разрядность промежуточных и выходных операндов.

Второй пример, приведенный ниже, уже будет относиться к конкретной задаче по реализации на основе арифметики с ФЗ двухканального банка фильтров.

Пример 2. *Пусть необходимо реализовать на основе арифметике с ФЗ двухканальный банк фильтров, сепарирующий входной сигнал на НЧ- и ВЧ- составляющие с последующей децимацией по основанию два в каждом канале*[1]. *Исходные данные: в качестве базисных используются вейвлет-фильтры семейства Добеши Db2 (4 коэффициента); значения входного сигнала представляются как дробные числа в формате с фиксированной запятой в дополнительном коде, разрядность 16 битов (Q15); реализация банка должна быть выполнена с использованием схемы на лестничных структурах (в зарубежной литературе lifting scheme).*

Решение. В рамках данного пособия не будут рассматриваться теоретические выкладки, касающиеся схемы банка фильтров на лестничных структурах. Поэтому перейдем непосредственно к рассмотрению алгоритма. На рис. 3.6 приведена блок-схема проектируемого двухканального банка с использованием терминов -преобразования.

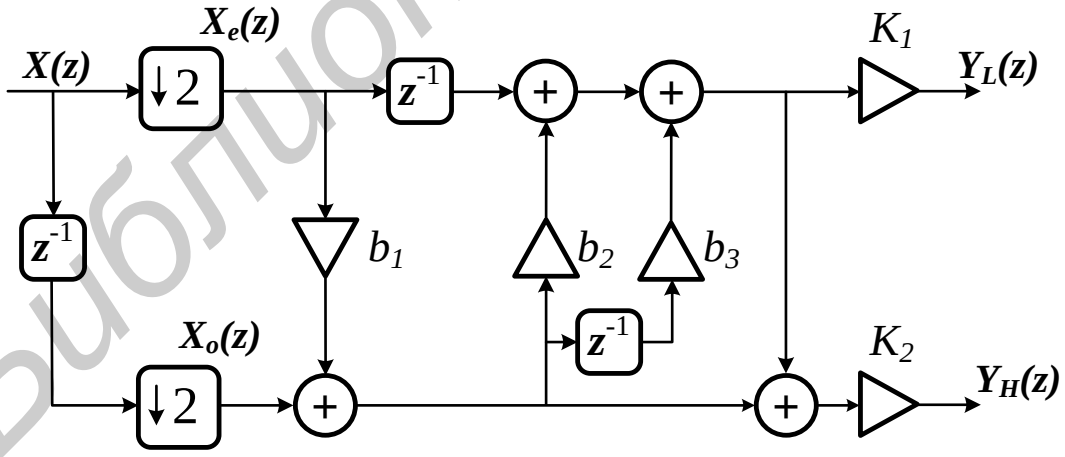


Рис. 3.6. Блок-схема двухканального банка фильтров анализа на лестничных структурах для материнской вейвлет-функции Добеши Db2

[1] Двухканальный банк фильтров применяется в различных приложениях многоскоростной обработки сигналов. В частности данный блок является базовой декомпозицией в алгоритмах быстрого вейвлет-преобразования и пакетного дискретного вейвлет-преобразования.

На рисунке введены следующие обозначения: $X\ z$ *– представление в* - *области входного сигнала;* $X_e\ z$, $X_o\ z$ *– представления последовательностей, состоящих соответственно из четных (even) и нечетных (odd) отсчетов входной последовательности;* b_1, b_2, b_3, K_1, K_2 *– постоянные коэффициенты, на которые осуществляется умножение входных и промежуточных значений в соответствии со схемой алгоритма;* $Y_L\ z$, $Y_H(z)$ *– представления выходных НЧ- и ВЧ-компонент входного сигнала.*

Значения постоянных коэффициентов лестничной структуры для вейвлет-фильтров Добеши Db2 можно получить, используя среду MATLAB, при помощи следующего кода:

```
lsdb2 = liftwave('db2');
```

в результате выполнения которого в переменную `lsdb2` *будет занесена структура данных, хранящая информацию о схеме двухканального банка фильтров на лестничных структурах. Таким образом, имеем следующие значения коэффициентов:* $b_1 = -1{,}7321,\ b_2 = -0{,}0670,\ b_3 = 0{,}4330,\ K_1 = 1{,}9319,\ K_2 = 0{,}5176$. *При данных значениях схема имеет на выходе в полосе пропускания коэффициент усиления* $\overline{2}$.

Теперь для выполнения арифметических вычислений необходимо осуществить квантование (перевод из формата с плавающей запятой в формат с фиксированной запятой) коэффициентов. Для этого используем следующее общее выражение:

$$b = m_b \cdot 2^{exp_b}, \qquad m_b \in [0{,}5;\ 1), \qquad exp_b \in \mathbb{Z}. \tag{3.5}$$

В сущности, данное выражение подразумевает, что при переводе из формата с плавающей запятой в формат с фиксированной запятой значению m_b *присваивается мантисса исходного коэффициента, заданного в формате с плавающей запятой, а параметру* exp_b *сопоставляется порядок этого значения. Данный способ позволяет сохранить наибольшую точность при переводе числа при выбранной длине слова.*

В итоге получаем следующие значения для коэффициентов с точностью представления 16 битов: $m_{b1} = 9126_{16}$, $m_{b2} = BB68_{16}$, $m_{b3} = 6EDA_{16}$, $m_{K1} = 7BA4_{16}$, $m_{K2} = 4241_{16}$, *и соответственно значения* $exp_{b1} = 1$, $exp_{b2} = -3$, $exp_{b3} = -1$, $exp_{K1} = 1$, $exp_{K2} = 0$.

В заключение остается определить формат представления данных в узлах алгоритма по аналогии с предыдущим примером и в соответствии с полученными результатами пересчитать значения для арифметических сдвигов перед операциями сложения.

Таким образом, для рассматриваемого алгоритма получаем схему реализации на арифметике с фиксированной запятой переменного формата, представленную на рис. 3.7.

Как видно из приведенной схемы, значения порядков масштабирующих

множителей выходных значений равняются 1. Данный результат связан с тем, что выходной коэффициент усиления в полосе пропускания равен для каждого канала $\overline{2}$*, что в свою очередь и потребовало 1 бит для задания целой части числа.*

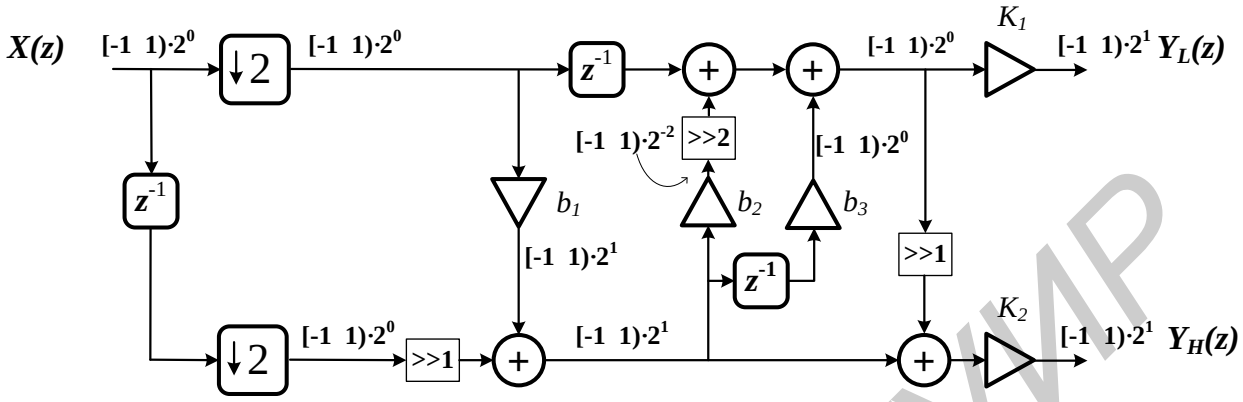


Рис. 3.7. Блок-схема реализации двухканального банка фильтров на основе арифметики переменного формата

3.3. Структура сопроцессора для вычислений в арифметике переменного формата

В качестве примера рассмотрим реализацию сопроцессорного блока для RISC-процессора PLASMA, выполняющего в арифметике переменного формата заданный набор математических операций.

Структурно данный блок состоит из шести регистров данных, блока управления, а также двух умножителей, двух сумматоров, четырех мультиплексоров и четырех устройств арифметического сдвига вправо (рис. 3.8). В общем виде выражение, вычисляемое сопроцессором, имеет вид

$$Y = (O_1 \cdot 2^{-sh1} + O_2 \cdot 2^{-sh2}) \cdot 2^{-sh3} + O_3 \cdot 2^{-sh4}. \qquad (3.6)$$

Здесь $sh1, sh2, sh3, sh4$ значения арифметических сдвигов вправо операндов, принимающие значения от 0 до 7; O_1, O_2, O_3 – операнды, определяемые кодом операции.

Выполнение заданных вычислений осуществляется путем записи исходных данных и управляющей команды в регистры, отображаемые на память процессора PLASMA. Вычисленное значение считывается из регистра результата.

С точки зрения программиста, принцип работы сопроцессора заключается в следующем. Для задания типа выполняемой операции необходимо записать в 32-разрядный регистр управления по адресу 30000000_{16} определенный двоичный код. Формат регистра приведен на рис. 3.9.

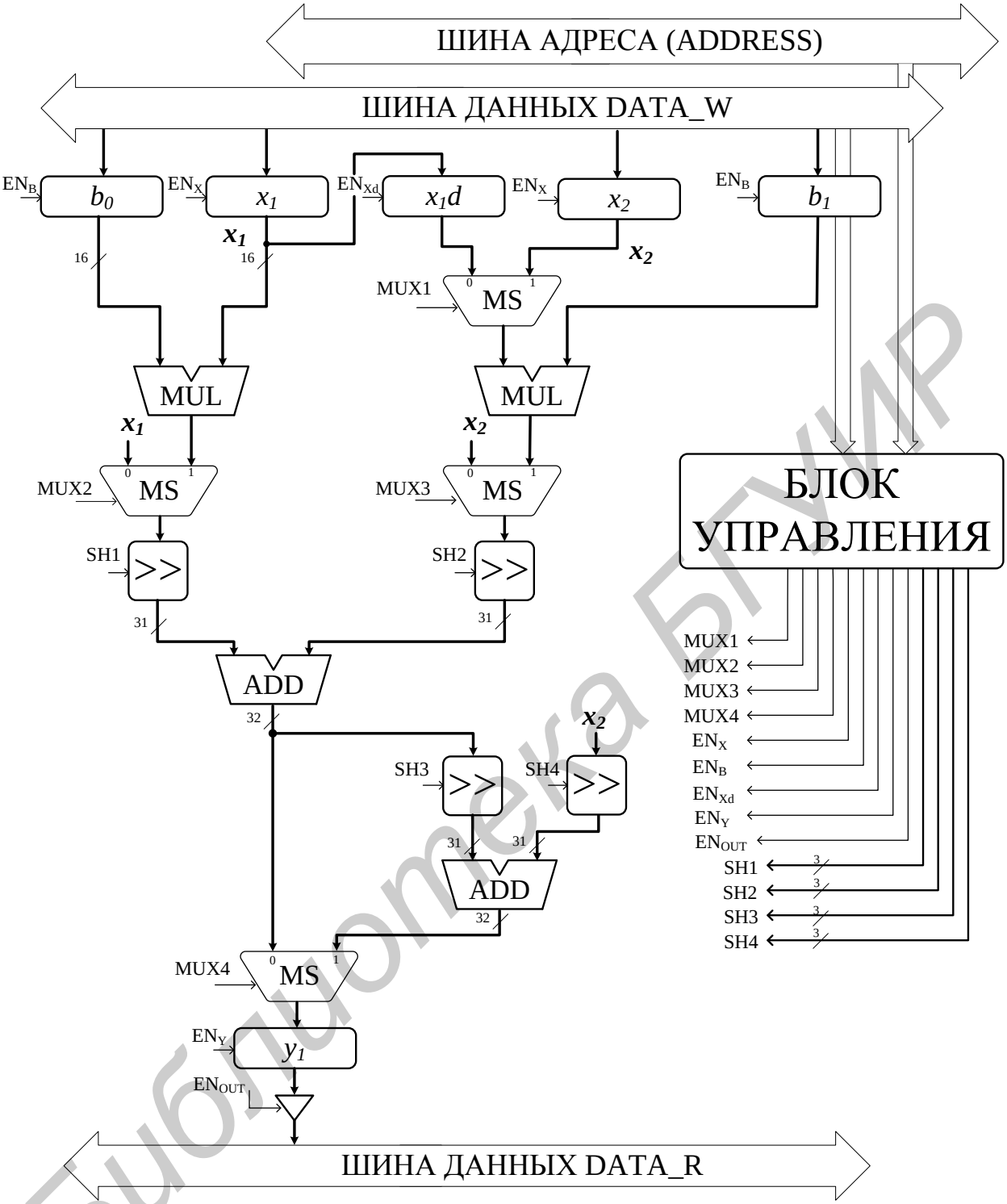
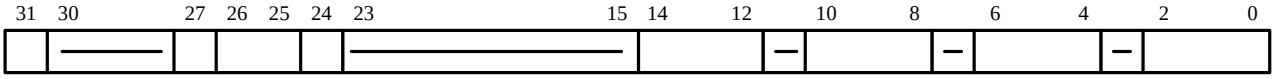


Рис. 3.8. Блок-схема сопроцессора

Программист задает операнды O_1, O_2, O_3 и значения арифметических сдвигов вправо перед подачей данных на сумматоры. Также при записи в старший 31-й бит логической единицы осуществляется операция сброса всех регистров сопроцессора.



Номера битов: Значения полей:

2-0: Значение арифметического сдвига для операнда O_1

6-4: Значение арифметического сдвига для операнда O_2

10-8: Значение арифметического сдвига для выхода сумматора O_1+O_2

14-12: Значение арифметического сдвига для операнда O_3

24: Бит выбора операнда O_1: '0' - x_1 '1' - $x_1 \cdot b_0$

26-25: Поле выбора операнда O_2: '00' либо '10' - x_2; '01' - $b_1 \cdot x_1 d$; '11' - $b_1 \cdot x_2$

27: Бит выбора операнда O_3: '0' - нет; '1' - x_2

31: Бит сброса регистров данных **R**: '1' -сброс, '0' - рабочий режим

3,7,11,15-23,28-30: Не используются

Рис. 3.9. Поля регистра управления

Например, если необходимо выполнить вычисление по следующей формуле:

$$y\ n\ = (b_0 x_1\ n\ + (b_1 x_1\ n-1\ ) \cdot 2^{-3}) \cdot 2^{-1} + x_2[n],$$

то в регистр управления заносится следующее значение: $0b000130_{16}$.

Далее необходимо передать сопроцессору 16-разрядные значения коэффициентов b_0 и b_1. Для этого программист должен в старшую часть 32-разрядной переменной поместить значение b_1, а в младшую часть – значение b_0, после чего сохранить переменную по адресу 30000008_{16}.

После конфигурирования сопроцессора выполнение алгоритма сводится к последовательности из двух операций: записи переменной, содержащей пару 16-разрядных значений $x_1[n]$ и $x_2\ n$ по адресу 30000004_{16}, и чтения результата вычислений по адресу $3000000C_{16}$. При этом значение $x_1[n]$ заносится в старшую часть переменной, $x_2\ n$ – в младшую часть.

Операция умножения в сопроцессоре реализована в соответствии со следующими замечаниями: входные операнды представлены как дробные знаковые числа в дополнительном коде в формате Q15, а результат умножения в формате Q30. Для реализации блока умножения написан следующий VHDL-код:

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_SIGNED.ALL;--библиотека работы со знаковыми числами
--Полный знаковый умножитель дробных чисел (разрядность входных данных N)
entity MULT_FS is
     Generic(N :natural:=16);
    Port ( A : in  STD_LOGIC_VECTOR (N-1 downto 0);
           B : in  STD_LOGIC_VECTOR (N-1 downto 0);
           S : out  STD_LOGIC_VECTOR (2*N-2 downto 0));
```

```
end MULT_FS;

architecture Behavioral of MULT_FS is
signal result: STD_LOGIC_VECTOR (2*N-1 downto 0);
begin
result<=A*B;
S<=result(2*N-2 downto 0);
end Behavioral;
```

Сумматоры сопроцессора выполняют операцию сложения чисел разрядностью 31 бит, при этом результат имеет разрядность 32 бита, что позволяет предотвратить переполнение. В связи с этим необходимо учитывать, что вес старшего значащего бита результата суммы будет на единицу больше веса старшего значащего бита входных операндов. Ниже дан VHDL-код сумматора:

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_SIGNED.ALL; --библиотека работы со знаковыми числами
--Полный N-разрядный сумматор в дополнительном коде
--Разрядность результата N+1
entity ADD_FULL is
     Generic(N: natural :=32);
    Port ( X0 : in  STD_LOGIC_VECTOR (N-1 downto 0);
           X1 : in  STD_LOGIC_VECTOR (N-1 downto 0);
           Y : out  STD_LOGIC_VECTOR (N downto 0));
end ADD_FULL;

architecture Behavioral of ADD_FULL is
begin
Y<=(X0(N-1) & X0) + (X1(N-1) & X1);
end Behavioral;
```

Для реализации сдвигающего устройства в сопроцессоре используется мультиплексор, с разрядностью данных 31 бит. На информационные входы мультиплексора по адресам с 0 по 7 подаются соответственно входное значение и значения, сдвинутые вправо на 1, 2, 3, 4, 5, 6 и 7 битов с расширением знакового бита. Таким образом, значение, подаваемое на адресные входы мультиплексора, определяет величину арифметического сдвига в данном блоке. VHDL-описание сдвигающего устройства приведено ниже.

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
--Блок арифметического сдвига вправо: максимальное значение сдвига 7
--Разрядность входных и выходных данных определяется параметром N
entity ARITH_RIGHT_SHIFT is
     Generic(N: natural:=16);
    Port ( X : in  STD_LOGIC_VECTOR (N-1 downto 0);
           SH : in  STD_LOGIC_VECTOR (2 downto 0);
           Y : out  STD_LOGIC_VECTOR (N-1 downto 0));
```

```
end ARITH_RIGHT_SHIFT;

architecture Behavioral of ARITH_RIGHT_SHIFT is

begin
sh_proc: process(SH,X)
begin
   case SH is
      when "000" => Y<=X;
      when "001" => Y<=X(N-1)& X(N-1 downto 1);
      when "010" => Y<=X(N-1)& X(N-1)&X(N-1 downto 2);
      when "011" => Y<=X(N-1)& X(N-1)&X(N-1)&X(N-1 downto 3);
      when "100" => Y<=X(N-1)& X(N-1)&X(N-1)&X(N-1)&X(N-1 downto 4);
      when "101" =>
      Y<=X(N-1)&X(N-1)&X(N-1)&X(N-1)&X(N-1)&X(N-1 downto 5);
      when "110" =>
      Y<=X(N-1)&X(N-1)&X(N-1)&X(N-1)&X(N-1)&X(N-1)&X(N-1 downto 6);
      when "111" =>
Y<=X(N-1)&X(N-1)&X(N-1)&X(N-1)&X(N-1)&X(N-1)&X(N-1)&X(N-1 downto 7);
      when others => Y<=X;
end case;
end process;
end Behavioral;
```

Блок управления сопроцессора (рис. 3.10) выполняет две основные функции:

1) дешифрирование адреса на шине ADDRESS и формирование сигналов разрешения записи с шины DATA_W в регистры сопроцессора, а также формирование сигнала разрешения выдачи результата вычислений на шину DATA_R;

2) запись команды в регистр управления с последующим декодированием.

Первая функция реализована на основе комбинационной схемы, входами которой являются биты шины адреса и сигнал WE, определяющий тип команды (запись/чтение). На выходах данной схемы при обращении к регистрам x_1, x_2, b_0, b_1, y или регистру управления формируются следующие сигналы: EN_X – разрешение записи в регистры x_1, x_2; EN_B – разрешение записи в регистры b_0, b_1; en_cntrl – разрешение записи в регистр управления; EN_OUT – разрешение выдачи данных из регистра y на шину DATA_R.

Управляющие сигналы для внутренней схемы сопроцессора формируются с выходов регистра управления (рис. 3.10). Биты полей, задающих типы операндов O_1, O_2, O_3, (сигналы MUX1, MUX2, MUX3, MUX4 на рис. 3.10) подаются на адресные входы четырех мультиплексоров (рис. 3.8) для реализации нужной схемы вычислений. Поля, определяющие величины арифметических сдвигов операндов (сигналы SH1, SH2, SH3, SH4), подаются на управляющие входы соответствующих блоков. Сигнал сброса RESET_REGS формируется как функция логического «ИЛИ» бита 31 регистры управления и системного сигнала сброса RESET. Для фиксации результата вычислений в регистр y используется сигнал разрешения EN_Y, который формируется как задержанный на один

такт сигнал EN_X при помощи D-триггера. Таким образом, результат вычислений всегда будет фиксироваться в следующем такте после записи данных в регистры x_1, x_2.

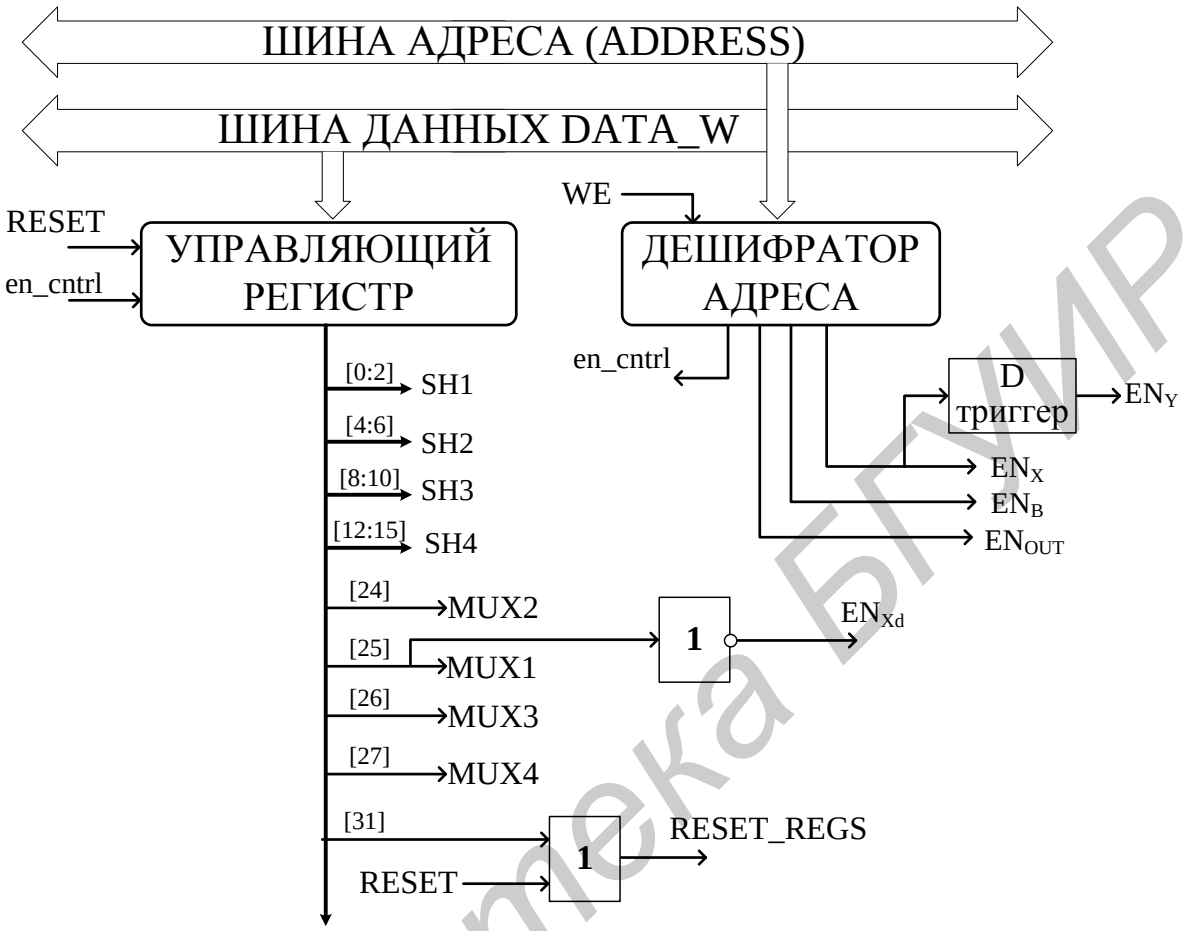


Рис. 3.10. Функциональная схема блока управления сопроцессора

Ниже приведено VHDL-описание блока управления сопроцессора.

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity CONTROL_UNIT is
    Port ( CLK : in  STD_LOGIC;
           RESET: in STD_LOGIC;
           WE: in STD_LOGIC;
           DATA_W : in  STD_LOGIC_VECTOR (31 downto 0);
           ADDRESS : in  STD_LOGIC_VECTOR (31 downto 0);
           MUX1 : out  STD_LOGIC:='0';
           MUX2 : out  STD_LOGIC:='0';
           MUX3 : out  STD_LOGIC:='0';
           MUX4 : out  STD_LOGIC:='0';
           EN_X : out  STD_LOGIC:='0';
           EN_B : out  STD_LOGIC:='0';
```

```
           EN_Y:  out  STD_LOGIC:='0';
           EN_OUT: out STD_LOGIC:='0';
           RESET_REGS:out STD_LOGIC:='0';
           SH1: out STD_LOGIC_VECTOR(2 downto 0):="000";
           SH2: out STD_LOGIC_VECTOR(2 downto 0):="000";
           SH3: out STD_LOGIC_VECTOR(2 downto 0):="000";
           SH4: out STD_LOGIC_VECTOR(2 downto 0):="000");
end CONTROL_UNIT;

architecture Behavioral of CONTROL_UNIT is
--Сигналы дешифратора адреса
signal ds_x,ds_b,ds_out,ds_cntrl,en_cntrl: STD_LOGIC:='0';
begin
--Дешифратор адреса
inst_addr_ds:process(ADDRESS)
begin
   case ADDRESS is
      --Адрес регистра управления
when x"30000000" => ds_x<='0'; ds_b<='0'; ds_out<='0'; ds_cntrl<='1';
      --Адрес регистров x1 x2
when x"30000004" => ds_x<='1'; ds_b<='0'; ds_out<='0'; ds_cntrl<='0';
      --Адрес регистров b0 b1
when x"30000008" => ds_x<='0'; ds_b<='1'; ds_out<='0'; ds_cntrl<='0';
      --Адрес регистра результата
when x"3000000C" => ds_x<='0'; ds_b<='0'; ds_out<='1'; ds_cntrl<='0';
when others => ds_x<='0'; ds_b<='0'; ds_out<='0'; ds_cntrl<='0';
end case;
end process;
--Формирование сигналов разрешения записи в регистры сопроцессора
EN_X<=ds_x and (not WE); EN_B<=ds_b and (not WE);
EN_OUT<=ds_out and WE;
en_cntrl<=ds_cntrl and (not WE);

--Управляющий регистр
inst_cntrl_reg: process(CLK)
begin
if rising_edge(CLK) then
     if RESET='1' then
           SH1<="000"; SH2<="000";
           SH3<="000"; SH4<="000";
           MUX1<='0';  MUX2<='0';
           MUX3<='0';  MUX4<='0';
           RESET_REGS<='0'; EN_Y<='0';
     else
           if en_cntrl='1' then
                 SH1<=DATA_W(2 downto 0);  SH2<=DATA_W(6 downto 4);
                 SH3<=DATA_W(10 downto 8); SH4<=DATA_W(14 downto 12);
                 MUX1<=DATA_W(24);  MUX2<=DATA_W(25);
                 MUX3<=DATA_W(26);  MUX4<=DATA_W(27);
                 RESET_REGS<=DATA_W(31) or RESET;
           end if;
           EN_Y<=ds_x and (not WE);
     end if;
end if;
end process;
end Behavioral;
```

Литература

1. Байков, В. Д. Аппаратурная реализация элементарных функций в ЦВМ / В. Д. Байков, В. Б. Смолов. – Л. : Изд-во Ленингр. ун-та, 1975. – 96 с.

2. Кун, С. Матричные процессоры на СБИС / С. Кун. – М. : Мир, 1991. – 672 с.

3. Байков, В. Д. Специализированные процессоры: Итерационные алгоритмы и структуры / В. Д. Байков, В. Б. Смолов – М. : Радио и связь, 1985. – 288 с.

4. Путков, В. П. Электронные вычислительные устройства : учеб. пособие для радиотех. спец. вузов / В. П. Путков, И. И. Обросов, С. В. Бекетов. – Минск : Выш. шк., 1981. – 333 с.

5. Калабеков, Б. А. Основы автоматики и вычислительной техники : учебник для техникумов связи / Б. А. Калабеков, И. А. Мамзелов. – М. : Связь, 1980. – 296 с.

6. Айнспрук, Н. Электроника СБИС. Проектирование микроструктур / Н. Айнспрук ; пер. с англ. – М. : Мир, 1989. – 256 с.

7. Влах, И. Машинные методы анализа и проектирования электронных схем / И. Влах, К. Сингхал ; пер. с англ. – М. : Радио и связь, 1988. – 560 с.

8. Хамахер, К. Организация ЭВМ / К. Хамахер, З. Вранешич, С. Заки ; пер. с англ. – 5-е изд. – СПб. : Питер, 2003. – 848 с.

Св. план 2010, поз. 13

Учебное издание

Петровский Александр Александрович
Вашкевич Максим Иосифович
Родионов Максим Михайлович

ПРОЕКТИРОВАНИЕ ЭВС С ДИНАМИЧЕСКИ РЕКОНФИГУРИРУЕМОЙ АРХИТЕКТУРОЙ

Методическое пособие
для студентов специальности 1-40 02 02
«Электронные вычислительные средства»
дневной формы обучения

Редактор Е. Н. Батурчик
Корректор А. В. Тюхай

Подписано в печать 07.06.2011.	Формат 60x84 1/16.	Бумага офсетная.
Гарнитура «Таймс».	Отпечатано на ризографе.	Усл. печ. л. 2,44.
Уч.- изд. л. 2,5.	Тираж 100 экз.	Заказ 542.

Издатель и полиграфическое исполнение: учреждение образования
«Белорусский государственный университет информатики и радиоэлектроники»
ЛИ №02330/0494371 от 16.03.2009. ЛП №02330/0494175 от 03.04.2009.
220013, Минск, П. Бровки, 6