ЛАБОРАТОРНАЯ РАБОТА №4: СОПРОЦЕССОР CORDIC-АЛГОРИТМА

1.1. Цель работы

Составить VHDL описание CORDIC-процессора, получить временные диаграммы его работы, оценить объем занимаемых ресурсов и быстродействие.

1.1.1. Теоретические сведения

1.1.1.1. 2-D CORDIC-алгоритм Вычислительный алгоритм CORDIC, предложенный Волдером и позднее развитый Вальтером, представляет собой алгоритм, оперирующий двумерными (2D) векторами для вычисления вращений, используя простые арифметические примитивы: сдвиг и сложение. Вращение вектора $\mathbf{x} = [x_1\, x_2]^T$ на угол φ может быть описан произведением данного вектора на соответствующую матрицу вращения $\mathbf{R}_2(\varphi)$:

$$\begin{bmatrix} y_1 \\ y_2 \end{bmatrix} = \underbrace{\begin{bmatrix} \cos\varphi & -\sin\varphi \\ \sin\varphi & \cos\varphi \end{bmatrix}}_{\mathbf{R}_2(\varphi))} \underbrace{\begin{bmatrix} x_1 \\ x_2 \end{bmatrix}}_{\mathbf{x}} \tag{1.1}$$

С другой стороны это выражение описывает произведение комплексного числа $(x_1 + jx_2)$ на комплексную экспоненту $e^{-j\varphi}$. CORDIC-алгоритм аппроксимирует матрицу вращения произведением N элементарных вращений $\mathbf{R}_2(i)$:

$$\mathbf{R}_2(\varphi) = \prod_{i=0}^{N-1} \mathbf{R}_2(i). \tag{1.2}$$

$$\begin{aligned} \mathbf{R}_2(i) &= \left(1 + \delta t_i^2\right)^{-1/2} \mathbf{U}_2(i), \\ \mathbf{U}_2(i) &= \begin{bmatrix} 1 & -\delta \cdot \sigma(i) t_i \\ \delta \cdot \sigma(i) t_i & 1 \end{bmatrix}, \end{aligned} \tag{1.3}$$

где $\mathbf{U}_2(i)$ — ненормализованная часть $\mathbf{R}_2(i)$, $\delta = 1$, если вращение в евклидовом пространстве, $\delta = -1$, если вращение осуществляется в псевдоевклидовом пространстве (гиперболическое вращение), $t_i = 2^{-i}$. Параметры N, $\sigma(i)$, i, подбираются таким образом, что результат произведения (1.2) аппроксимирует вращение на угол

$$\varphi \approx \sum_{i=0}^{N-1} \sigma(i) \cdot 2^{-i}, \tag{1.4}$$

где N —число вращений, а параметр $\sigma(i) \in \{-1, 1\}$ определяет направление вращения для оставшейся части угла.

CORDIC-алгоритм состоит из двух процессов: итерационного процесса вращения и процесса масштабирования. На каждой итерации входной вектор $\mathbf{x}$ поворачивается на угол 2^{-i}, но результат вращения не соответствует фиксированному радиусу — длина вектора увеличивается на величину $\left(1 + 2^{-2i}\right)^{1/2}$, т. е. необходима нормализация размера вектора, которая выполняется в процессе масштабирования. Таким образом, на итерации для соответствующего входного вектора $\mathbf{x} = [x_1(i)\ x_2(i)]^T$ выходной вектор $\mathbf{x}(i+1) = [x_1(i+1)\ x_2(i+1)]^T$ вычисляется следующим образом:

$$\mathbf{x}(i+1) = \mathbf{U}_2 \cdot \mathbf{x}(i), \tag{1.5}$$

$$\varphi(i+1) = \varphi(i) - \sigma(i)2^{-i}, \text{ для } i = \overline{0, N-1}, \tag{1.6}$$

где $\mathbf{U}_2$ представляет собой оператор микровращения на i-й итерации CORDIC алгоритма; $\varphi(i)$ — оставшийся угол после i-й итерации. Структура модуля CORDIC алгоритма (1.5) и (1.6) показана на рисунке 1.1.

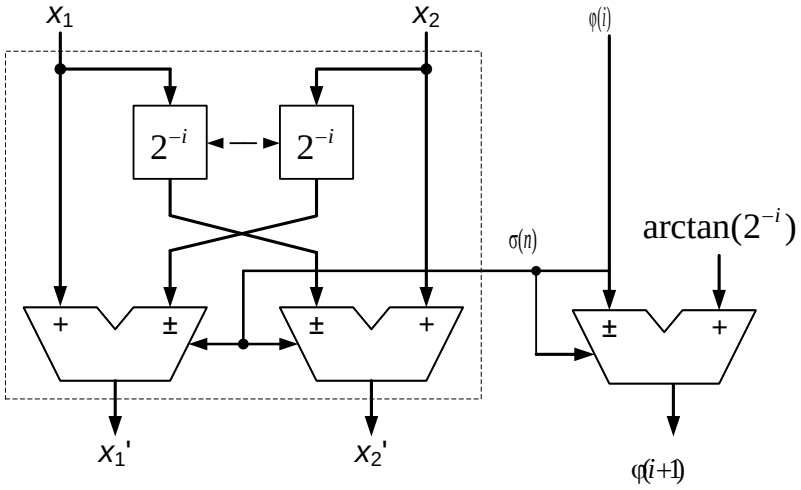

Рисунок 1.1. – Структура модуля CORDIC-алгоритма

Амплитуда результата нормализуется на этапе масштабирования после окончания итерационного этапа: выходной вектор $\mathbf{x}(N)$ делится на масштабный фактор $T = \prod_{i=0}^{N-1}\left(1 + 2^{-2i}\right)^{1/2}$. Итак, окончательный результат $\mathbf{x}_{out}$ CORDIC-алгоритма может быть представлен как

$$\mathbf{x}_{out}(N) = \frac{1}{T}\mathbf{x}(N) = \left(\prod_{i=0}^{N-1}\left(1 + 2^{-2i}\right)^{1/2}\right)^{-1} \cdot \mathbf{x}(N) \tag{1.7}$$

Таким образом, масштабирование требует дополнительной вычислительной операции, но, если масштабный фактор представить в виде

$$\frac{1}{T} = \sum_{i=0}^{S-1} \sigma(s) \cdot 2^{-i(s)}, \tag{1.8}$$

где $\sigma(s)$ и $0 \le i(s) \le N-1$, тогда процесс масштабирования может быть описан рекурсивным выражением: пусть $\mathbf{x}(0) = 0$ и $\mathbf{S} = \mathbf{x}_{out}(N)$, тогда

$$\mathbf{x}(s+1) = \mathbf{x}(s) + \sigma(s) \cdot 2^{-i(s)}\mathbf{x}(N) \tag{1.9}$$

В стандартной CORDIC-технике параметры алгоритма равны: $\sigma(i) \in \{-1, 1\}$, $i = \overline{0, N-1}$ и $N = B$, где B — число разрядов в двоичном представлении компонент векторов, т. е. сдвиг и знак оператора направления вращения для обоих процессов как итерационного, так и масштабирования определяются в режиме онлайн на основе входных данных. Выбор соответствующих параметров алгоритма определяет фиксированное число микровращений (итераций).

1.1.1.2. Структура вычислительного модуля сопроцессора CORDIC-алгоритма Вычислительный модуль CORDIC-алгоритма рассматриваемый в данном разделе, выполняет арифметические операции и операции сдвига над числами с фиксированной запятой формата `Q16.15` (правильные дроби с длиной слова 16 бит). Структурная схема АЛУ сопроцессора CORDIC-алгоритма показана на рис. 1.2.

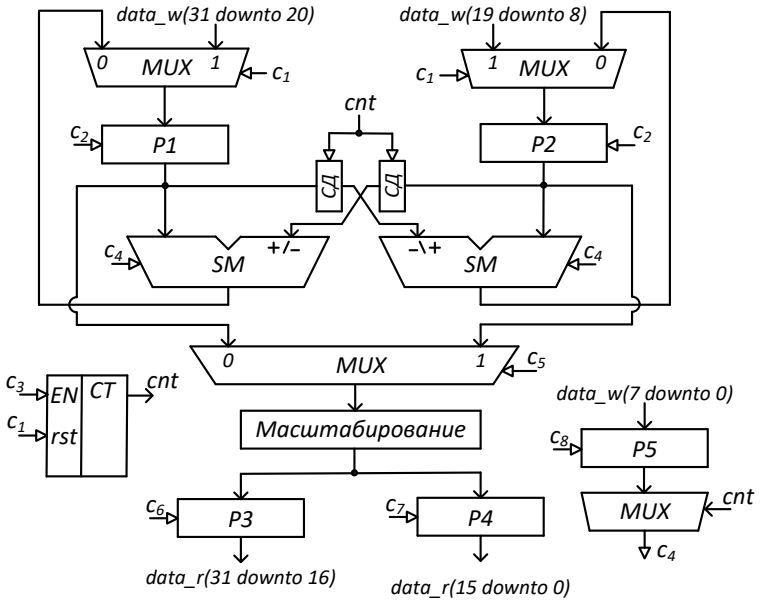

Рисунок 1.2. – АЛУ сопроцессора CORDIC

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, его интерфейс должен быть совместим с упрощенным интерфейсом шины данных для выполнения транзакций чтения и записи:

```
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;
```

С точки зрения RISC-процессора сопроцессор CORDIC-алгоритма является портом ввода-вывода отображенным на память. С точки зрения разработчика программного обеспечения работа с сопроцессором аналогична записи в ячейки физической памяти.

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

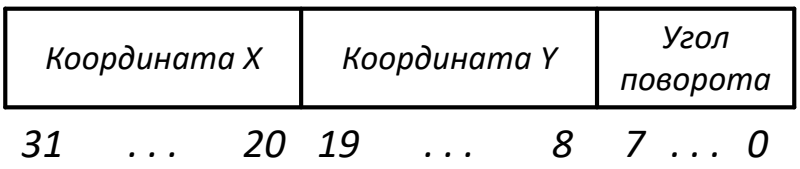

Рисунок 1.3. – Формат данных передаваемых в сопроцессор CORDIC-алгоритма

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 выходных результата

можно выполнить как аппросимацию множителя степенями двойки. На основании структурной схемы на рис.1.2 можно составить VHDL-описание АЛУ сопроцессора CORDIC (раздел 1.4.1).

1.1.1.3. Варианты заданий лабораторных работ В таблице 1.1 приведены варианты заданий.

Таблица 1.1. – Варианты заданий

Вариант	Устройство
1	Устройство вращения на основе CORDIC-алгоритма в евклидовом пространстве ($\delta = +1$) на фиксированный угол $\varphi = \pi/6$, разрядность слова $B = 12$, число итераций алгоритма $N = 6$
2	Устройство вращения на основе CORDIC-алгоритма в евклидовом пространстве ($\delta = +1$) на фиксированный угол $\varphi = 3\pi/7$, разрядность слова $B = 8$, число итераций алгоритма $N = 8$
3	Устройство вращения на основе CORDIC-алгоритма в евклидовом пространстве ($\delta = +1$) на фиксированный угол $\varphi = 3\pi/8$, разрядность слова $B = 6$, число итераций алгоритма $N = 6$
4	Устройство вращения на основе CORDIC-алгоритма в евклидовом пространстве ($\delta = +1$) на фиксированный угол $\varphi = 5\pi/9$, разрядность слова $B = 14$, число итераций алгоритма $N = 7$

1.2. Порядок выполнения работы

1. Изучить теоретические сведения по теме лабораторной работы.
2. Получить у преподавателя задание для выполнения практической части работы.
3. На языке VHDL составить структурное описание устройства в соответствии с вариантом.
4. Синтезировать устройство, проверить его работу. Оценить быстродействие, количество занимаемых в ПЛИС ресурсов и энергопотребление.
5. Показать результат работы устройства преподавателю.
6. Оформить и защитить отчет по лабораторной работе.

1.3. Содержание отчёта

1. Цель работы.
2. Схема синтезируемого устройства.

3. Описание устройства на VHDL с комментариями, листинг исходных текстов.
4. Временные диаграммы симуляции работы устройства.
5. Листинг отчета синтезатора о быстродействии, количестве занятых ресурсов.
6. Выводы по работе.

1.4. Задание повышенной сложности

Задание не является обязательным. Выполняется на выбор.

1. Реализовать на основе CORDIC-процессора генератор функций $\sin$ и $\cos$ с заданной угловой скоростью ω.
2. Параметры и состояние CORDIC-процессора задаются через регистровый файл с помощью интерфейса AXI-Lite.
3. CORDIC-процессор выполнен по полностью конвейерной схеме и реализует `axi_stream_master` и `axi_stream_slave` для соответствующих входов и выходов.

1.4.1. Реализация CORDIC-алгоритма в режиме вращения

VHDL-описание реализации сопроцессора для фиксированных углов вращения:

```
library ieee;
  use ieee.std_logic_1164.all;
  use ieee.numeric_std.all;

entity CORDIC_coprocessor is
    Port (
        address: in STD_LOGIC_VECTOR (31 downto 0);
        reset : in STD_LOGIC;
        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;


architecture rtl of CORDIC_coprocessor is

-- 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);


--.............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";


begin

--.............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;
```

```
79
80  --.............Управляющий автомат (Мили)
81  -- Память автомата
82  SYNC_PROC: process (clk)
83  begin
84  if rising_edge(clk) then
85      if (reset = '1') then
86          state <= a1;
87      else
88          state <= next_state;
89      end if;
90  end if;
91  end process;
92
93  --Декодирование выходных сигналов
94  OUTPUT_DECODE: process (state, we, address)
95  begin
96      case (state) is
97          when a1=>
98              -- Проверка условия x1
99              if address = x"80000800" and we = '1' and ff_busy = '0' then
100                 -- управление входными мультиплексорами и сброс счетчика cnt
101                 c1 <= '1';
102                 c2 <= '1'; -- запись данных в регистры P1/P2
103                 c8 <= '1'; -- запись данных в регистр угла поворота (P5)
104             else
105                 c1 <= '0'; c2 <= '0';
106                 c8 <= '0';
107             end if;
108             c3 <= '0';
109             -- Подключаем к блоку масштабирования регистр P1
110             c5 <= '0'; c6 <= '0';
111             c7 <= '0'; ff_busy_rst<='0';
112         when a2=>
113             -- управление входными мультиплексорами
114             c1 <= '0';
115             -- запись результата сложения/вычитания в регистры P1/P2
116             c2 <= '1';
117             c3 <= '0'; c5 <= '0';
118             c6 <= '0'; c7 <= '0';
119             c8 <= '0'; ff_busy_rst<='0';
120         when a3=>
121             c1 <= '0'; c2 <= '0';
122             c3 <= '1'; -- увеличиваем счетчик (cnt) на единицу
123             c5 <= '0'; c6 <= '0';
124             c7 <= '0'; c8 <= '0';
125             ff_busy_rst<='0';
126         when a4=>
127             if (cnt="111") then
128                 c6 <= '1'; -- запись результата в регистр P3
129             else
130                 c6 <= '0';
131             end if;
132             c1 <= '0'; c2 <= '0';
133             c3 <= '0'; c5 <= '0';
```

```
134             c7 <= '0'; c8 <= '0';
135             ff_busy_rst<='0';
136         when a5=>
137             c1 <= '0'; c2 <= '0';
138             c3 <= '0';
139             c5 <= '1'; -- подключаем регистр P2 к блоку масштабирования
140             c6 <= '0'; c8 <= '0';
141             c7 <= '1'; -- запись результата в регистр P4
142             ff_busy_rst<='1';
143         when others=>
144             c1 <= '0'; c2 <= '0';
145             c3 <= '0'; c5 <= '0';
146             c6 <= '0'; c7 <= '0';
147             c8 <= '0'; ff_busy_rst<='0';
148     end case;
149 end process;
150
151
152 NEXT_STATE_DECODE: process (state,we,address,cnt)
153 begin
154     next_state <= state; -- по-умолчанию остаемся в том же состоянии
155     case (state) is
156         when a1 =>
157             if address = x"80000800" and we = '1' and ff_busy = '0' then
158                 next_state <= a2;
159             end if;
160         when a2 => next_state <= a3;
161         when a3 => next_state <= a4;
162         when a4 =>
163             -- Если не выполнили все микроповороты - продолжаем.
164             if (cnt="111") then
165                 next_state <= a5;
166             else
167                 next_state <= a2;
168             end if;
169         when a5 => next_state <= a1;
170         when others =>
171             next_state <= a1;
172     end case;
173 end process;
174
175 -- выдача на шину либо сигнала занятости, либо выходных данных
176 data_r<= reg_out_X&reg_out_Y when ((address = x"80000800" ) and (we='0'))
177 else
178     (x"0000000"&"000"&ff_busy) when ((address = x"80000810" ) and (we='0'))
179 else "ZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZ";
180
181
182 ---------------- АЛУ сопроцессора CORDIC ----------------
183 --.............Мультиплексор для координаты X
184 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;
185 --.............Мультиплексор для координаты Y
186 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;
```

```
187
188 --.............Регистры X и Y (на схеме P1 и P2)
189 process (clk)
190 begin
191 if rising_edge(clk) then
192     if (c2='1') then
193         reg_X <= mux_X;
194         reg_Y <= mux_Y;
195     end if;
196 end if;
197 end process;
198
199 --.............Регистр угла поворота (на схеме P5)
200 process (clk)
201 begin
202 if rising_edge(clk) then
203     if (c8='1') then
204         reg_angle<= data_w(7 downto 0);
205     end if;
206 end if;
207 end process;
208
209 --.............Сдвигатель для регистра X
210 shf_X <=reg_X when cnt="000" else reg_X(15)&reg_X(15 downto 1) when cnt="001"
    ↪  else
211         reg_X(15)&reg_X(15)&reg_X(15 downto 2) when cnt="010" else
212         reg_X(15)&reg_X(15)&reg_X(15)&reg_X(15 downto 3) when cnt ="011" else
213         reg_X(15)&reg_X(15)&reg_X(15)&reg_X(15)&reg_X(15 downto 4) when cnt="100"
          ↪  else
214         reg_X(15)&reg_X(15)&reg_X(15)&reg_X(15)&reg_X(15)&reg_X(15 downto 5) when
          ↪  cnt="101" else
215         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
216
          ↪  reg_X(15)&reg_X(15)&reg_X(15)&reg_X(15)&reg_X(15)&reg_X(15)&reg_X(15)&reg
          ↪  downto 7);
217
218 --.............Сдвигатель для регистра Y
219 shf_Y <=reg_Y when cnt="000" else
220         reg_Y(15)&reg_Y(15 downto 1) when cnt="001" else
221         reg_Y(15)&reg_Y(15)&reg_Y(15 downto 2) when cnt="010" else
222         reg_Y(15)&reg_Y(15)&reg_Y(15)&reg_Y(15 downto 3) when cnt="011" else
223         reg_Y(15)&reg_Y(15)&reg_Y(15)&reg_Y(15)&reg_Y(15 downto 4) when cnt="100"
          ↪  else
224         reg_Y(15)&reg_Y(15)&reg_Y(15)&reg_Y(15)&reg_Y(15)&reg_Y(15 downto 5) when
          ↪  cnt="101" else
225         reg_Y(15)&reg_Y(15)&reg_Y(15)&reg_Y(15)&reg_Y(15)&reg_Y(15)&reg_Y(15
          ↪  downto 6) when cnt="110" else
226
          ↪  reg_Y(15)&reg_Y(15)&reg_Y(15)&reg_Y(15)&reg_Y(15)&reg_Y(15)&reg_Y(15)&reg
          ↪  downto 7);
227
228 --.............Сумматоры (на схеме SM)
229 process (shf_X, shf_Y, reg_X, reg_Y, c4)
230 begin
```

```
231     if c4 = '1' then -- 1 coded as 1; -1 coded as 0
232         add_X <= std_logic_vector(signed(reg_X) - signed(shf_Y));
233         add_Y <= std_logic_vector(signed(reg_Y) + signed(shf_X));
234     else
235         add_X <= std_logic_vector(signed(reg_X) + signed(shf_Y));
236         add_Y <= std_logic_vector(signed(reg_Y) - signed(shf_X));
237     end if;
238 end process;
239
240 --.............Мультиплексор выбора направления поворота
241 c4 <= reg_angle(7) when cnt="000" else
242       reg_angle(6) when cnt="001" else
243       reg_angle(5) when cnt="010" else
244       reg_angle(4) when cnt="011" else
245       reg_angle(3) when cnt="100" else
246       reg_angle(2) when cnt="101" else
247       reg_angle(1) when cnt="110" else
248       reg_angle(0);
249
250 --.............Мультиплексор для выполнения масштабирования
251 mux_sc <= reg_X when c5='0' else reg_Y;
252
253 --.............Блок масштабирования
254 -- sc = 2^-1 + 2^-3 - 2^-6 - 2^-9 - 2^-13
255 sc_tmp_1 <= mux_sc(15)&mux_sc(15 downto 1);
256 sc_tmp_3 <= mux_sc(15)&mux_sc(15)&mux_sc(15)&mux_sc(15 downto 3);
257 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);
258 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_s
    ↪  downto 9);
259 sc_tmp_13 <= mux_sc(15)&mux_sc(15)&mux_sc(15)&mux_sc(15)&mux_sc(15)&
260              mux_sc(15)&mux_sc(15)&mux_sc(15)&mux_sc(15)&mux_sc(15)&mux_sc(15)&
261              mux_sc(15)&mux_sc(15)&mux_sc(15 downto 13);
262
263 sc_output <= std_logic_vector(
264              signed(sc_tmp_1) + signed(sc_tmp_3) - signed(sc_tmp_6) -
                 ↪  signed(sc_tmp_9) - signed(sc_tmp_13));
265
266 --.............Выходные регистры для координат X и Y (на схеме P3 и P4)
267 process (clk)
268 begin
269 if rising_edge(clk) then
270     if (c6='1') then
271         reg_out_X <= sc_output;
272     end if;
273     if (c7='1') then
274         reg_out_Y <= sc_output;
275     end if;
276 end if;
277 end process;
278 --.............Счетчик циклов
279 process(clk)
280 begin
```

```
281     if rising_edge(clk) then
282         if (c1='1') then -- сброс счетчика
283             cnt<=(others=>'0');
284         elsif c3='1' then -- увеличение счетчика на единицу
285             cnt<= std_logic_vector(unsigned(cnt) + 1);
286         end if;
287     end if;
288 end process;
289 
290 end;
291 
292 ----------------------------------------------------------------------------------
293 --- Запуск в симуляторе Modelsim:
294 --- 1) Перейти в директорию с vhdl файлами
295 --- 2) vlib work
296 --- 3) vmap work
297 --- 4) vcom cordic.vhdl
298 --- 5) vsim -novopt work.cordic_module_tb
299 --- 6) do wave.do
300 --- 7) run 2000 ns
301 ---
302 --- Перезапуск и компиляция:
303 --- 1) vcom cordic.vhdl
304 --- 2) restart -f ; run 2000 ns
305 
306 library ieee;
307     use ieee.numeric_std.all;
308     use ieee.std_logic_1164.all;
309 
310 entity cordic_module_tb is
311 end;
312 
313 architecture cordic_module_tb_arch of cordic_module_tb is
314 
315     constant clk_period : time := 10 ns;  -- ! clk period
316 
317     component CORDIC_coprocessor is
318     Port (
319         address: in STD_LOGIC_VECTOR (31 downto 0);
320         reset  : in STD_LOGIC;
321         we     : in STD_LOGIC;
322         clk    : in STD_LOGIC;
323         data_w : in STD_LOGIC_VECTOR (31 downto 0);
324         data_r : out STD_LOGIC_VECTOR (31 downto 0)
325     );
326     end component CORDIC_coprocessor;
327 
328     signal clk : std_logic := '1';
329     signal reset : std_logic := '1';
330     signal we : std_logic;
331     signal data_w : STD_LOGIC_VECTOR (31 downto 0);
332     signal data_r : STD_LOGIC_VECTOR (31 downto 0);
333     signal address: STD_LOGIC_VECTOR (31 downto 0) := x"80000800"; -- адресс
        ↪ регистра данных
334 
```

```
335     signal test : STD_LOGIC_VECTOR(31 downto 0);
336 
337     alias X  : STD_LOGIC_VECTOR(11 downto 0)  is test (31 downto 20);
338     alias Y  : STD_LOGIC_VECTOR(11 downto 0)  is test (19 downto 8);
339     alias MU : STD_LOGIC_VECTOR( 7 downto 0 ) is test (7  downto 0);
340 
341 begin
342 
343     clk <= not clk after clk_period;
344     reset <= '0' after 2*clk_period;
345 
346     DUT: CORDIC_coprocessor
347         port map ( clk => clk, reset => reset, we => we, address => address,
            ↪  data_w => data_w, data_r => data_r );
348 
349         process
350         begin
351                 we <= '0';
352                 address <= x"80000800";
353                 X  <= "011111111111";
354                 Y  <= "000000000000";
355                 MU <= "10101101";
356             wait for 10*clk_period;
357                 data_w <= test;
358                 we <= '1';
359             wait for 2*clk_period;
360                  we <= '0';
361             wait for 200*clk_period;
362         end process;
363 
364 end;
```