Синтезируемое VHDL-описание автомата управления динамическим сдвигом фазы примитивы MMCM для ПЛИС 7-й серии фирмы Xilinx

В статье предложено синтезируемое описание модулей управления динамическим сдвигом фазы в блоках управления тактовой частотой в ПЛИС 7-й серии фирмы Xilinx. Приведены текстовые описания модулей и осциллограммы натурных испытаний, а также необходимые формулы расчета выполняемого сдвига фазы.

Михаил КОРОБКОВ
Korobkov. M.A@yandex.ru

Введение

Седьмая серия ПЛИС фирмы Xilinx представлена тремя семействами микросхем: Artix, Kintex и Virtex. Каждое из семейств 7-й серии имеет в своем составе специализированный модуль управления тактовой частотой и синхронизацией — Clock Management Tile (CMT). В самых больших ПЛИС 7-й серии таких модулей может быть до 24. Каждый из них содержит по одному блоку MMCM (Mixed-Mode Clock Manager) и PLL (Phase-Locked Loop) [1]. Блок MMCM имеет широкие возможности управления тактовыми частотами, а также их динамическими изменениями, что позволяет подстраивать тактовую частоту «на лету».

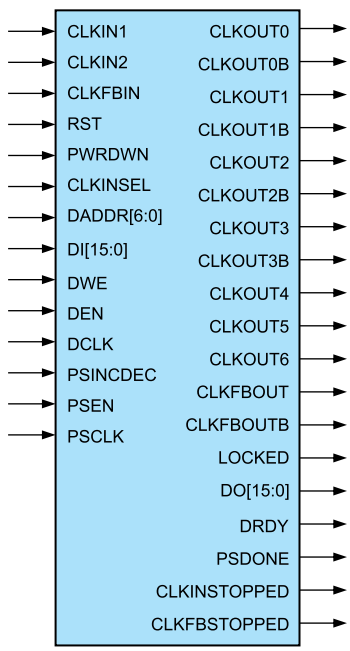

Рис. 1. Символьное изображение примитивы MMCME2_ADV

Динамический сдвиг фазы находит свое применение при учете времени окончания переходного процесса в различного рода системах, содержащих коммутирующие элементы. Поскольку коммутирующее устройство имеет некоторый переходный процесс, то в системе с динамической коммутацией, например в многоканальных схемах при использовании одного АЦП и нескольких коммутаторов, важно взять выборку сигнала именно в области закончившегося переходного процесса для получения достоверных данных. Использовать подстройку фазового сдвига «на лету» удобно при первой настройке изделия или на этапе отладки проекта в «железе». Можно вручную подстроить фазовый сдвиг тактового сигнала АЦП или модуля управления коммутацией, при котором будет взята чистая выборка, а затем фазовый сдвиг внести в MMCM как постоянный.

Описание примитивы MMCME2_ADV

Блок MMCM для пользователя представлен двумя примитивами MMCME2_BASE и MMCME2_ADV. В отличие от базовой версии MMCME2_BASE расширенная примитива MMCME2_ADV содержит дополнительные порты для выбора источника опорной частоты, доступ к порту динамической реконфигурации (DRP, Dynamic Reconfiguration Port), а также порты управления динамическим сдвигом фазы.

Таблица 1. Порты примитивы MMCME2_ADV

Описание	Порты
Тактовый вход	CLKIN1, CLKIN2, CLKFBIN, DCLK, PSCLK
Сигналы контроля и входных данных	RST, CLKINSEL, DWE, DEN, DADDR, DI, PSINCDEC, PSEN
Тактовый выход	CLKOUT0 — CLKOUT6, CLKOUT0B — CLKOUT3B, CLKFBOUT, CLKFBOUTB
Сигналы состояний и выходные данные	LOCKED, DO, DRDY, PSDONE, CLKINSTOPPED, CLKFBSTOPPED
Контроль вкл./откл.	PWRDWN

Символьное изображение примитивы MMCME2_ADV представлено на рис. 1, а функциональное назначение портов приведено в таблице 1.

Примитивы блока CMT в ПЛИС 7-й серии являются цифро-аналоговыми блоками, имеющими три режима работы: устранение искажений тактового сигнала, синтез частоты и уменьшение джиттера. Рабочая частота генератора, управляемого напряжением (VCO), определяется следующим выражением:

$$F_{VCO} = F_{CLKIN}(M/D), \qquad (1)$$

а выходная частота рассчитывается как:

$$F_{вых} = F_{CLKIN}(M/(D \times O)), \qquad (2)$$

где F_{CLKIN} — входная тактовая частота; M, D и O — значения счетчиков, показанных на рис. 2. Значение M соответствует параметру CLKFBOUT_MULT_F, значение D соответствует параметру DIVCLK_DIVIDE, а значение O соответствует параметру CLKOUT_DIVIDE в текстовом описании примитивы.

Семь O счетчиков могут быть независимо запрограммированы. Единственное ограничение накладывает рабочая частота ГУН, которая одинакова для всех выходных счетчиков, поскольку всеми счетчиками управляет только один ГУН.

Ограничения, накладываемые на параметры FCLKIN и FVCO для микросхем 7-й серии, приведены в соответствующих спецификациях на микросхемы [2].

Значения, которые могут принимать счетчики, а также другие некоторые атрибуты и тип их значений приведены в таблице 2.

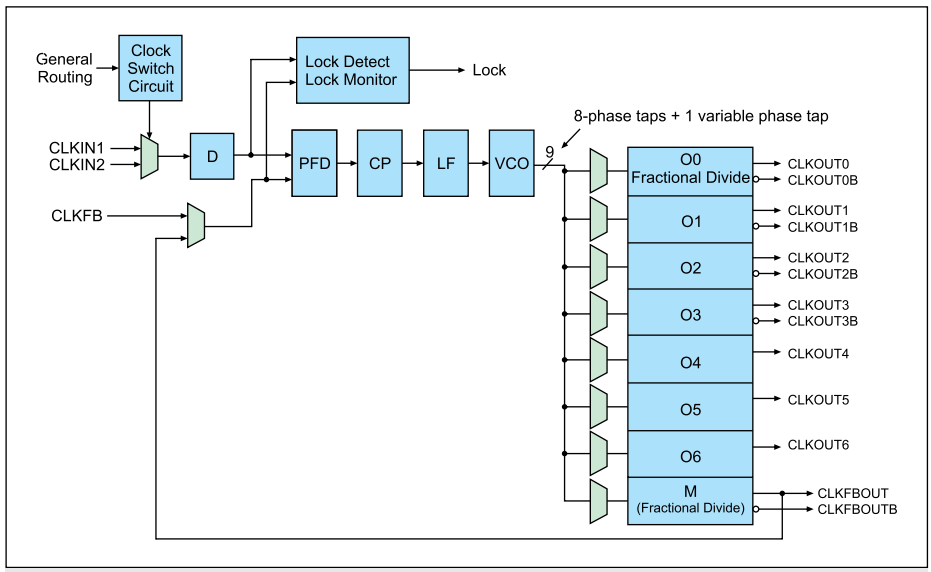

Рис. 2. Блок диаграмма блока MMCM

Таблица 2. Некоторые атрибуты блока MMCM

Атрибут	Тип	Диапазон значений	Значение по умолчанию	Описание
CLKOUT[1:6]_DIVIDE	Integer	1–128	1	Выходной делитель или счетчик O
CLKOUT[0]_DIVIDE_F	Integer или Real	1–128, шаг 0,125	1	
CLKFBOUT_MULT_F	Integer или Real	2–64, шаг 0,125	5	Делитель в цепи обратной связи, счетчик M
DIVCLK_DIVIDE	Integer	1–106	1	Делитель входной тактовой частоты, счетчик D
CLKIN1_PERIOD	Real	0,938–100	0.000	Период входной тактовой частоты, вход CLKIN1
CLKOUT0_USE_FINE_PS	Логический	FALSE, TRUE	FALSE	Разрешение динамического сдвига фазы. Параметр CLKOUT0_DIVIDE должен быть целочисленным
CLKOUT[1:6]_USE_FINE_P	Логический	FALSE, TRUE	FALSE	Разрешение динамического сдвига фазы

Описание интерфейса управления сдвигом фазы

Примитива MMCME2_ADV имеет три входа и один выход для осуществления динамического сдвига фазы. Каждый выход CLKOUT и делитель CLKFBOUT могут быть индивидуально выбраны для выполнения фазового сдвига. За выбор выходов, фаза которых будет сдвинута, отвечают атрибуты CLKOUT [0:6]_USE_FINE_PS и CLKFBOUT_USE_FINE_PS.

Контроль выполнения фазового сдвига осуществляется через порты PSEN, PSINDEC, PSCLK и PSDONE, временная диаграмма состояний которых приведена на рис. 3. Начальная фаза определяется атрибутом CLKOUT_PHASE. Часто начальный фазовый сдвиг не выбирается и остается равным 0°.

Фаза выхода MMCM инкрементируется/декрементируется в соответствии с состояниями управляющих сигналов PSEN, PSINCDEC, PSCLK и PSDONE относительно начального или предыдущего фазового сдвига. Сигналы PSEN, PSINCDEC и PSDONE синхронны с сигналом PSCLK. Когда PSEN активен один период PSCLK, инициализируется увеличение/уменьшение фазового сдвига. Когда PSINCDEC в «1», инициализируется положительный фазовый сдвиг, когда PSINCDEC в «0», инициализируется отрицательный фазовый сдвиг. Каждая команда инкремента фазового сдвига добавляет фазовый сдвиг выходу MMCM, равный 1/56 части периода ГУН. Аналогично, при выполнении команды декремента, происходит уменьшение фазы на 1/56 часть периода VCO. Сигнал PSEN должен быть активен в течение одного периода PSCLK. После завершения фазового сдвига сигнал PSDONE будет установлен в «1» ровно один период PSCLK. Время, в течение которого происходит выполнение команды, фиксировано и составляет 12 периодов PSCLK. После того как фазовый сдвиг инициализирован установкой PSEN, выход MMCM выполняет инкремент/декремент фазы относительно ее текущего значения. О завершении выполнения фазового сдвига сигнализирует сигнал PSDONE, значение которого становится «1». После того как PSDONE установился в «1», может быть инициализирован следующий фазовый сдвиг. Для фазового сдвига нет максимального значения и переполнения. Полный сдвиг (360°) всегда может быть инициализирован для любой частоты. Когда достигается конец периода, фазовый сдвиг переходит в 0°.

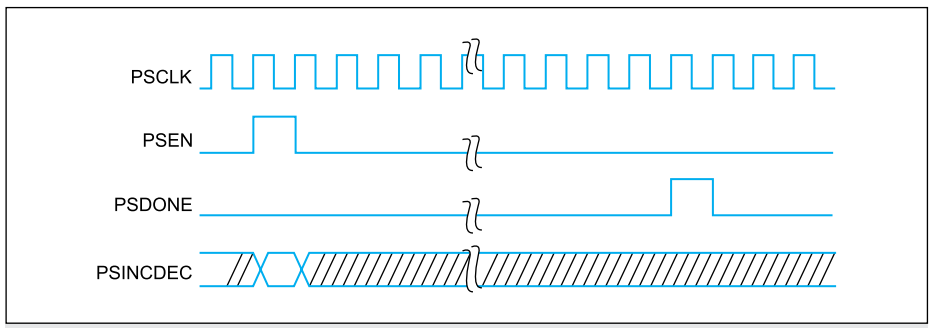

Рис. 3. Временная диаграмма интерфейса фазового сдвига

Описание программного модуля

Проект динамического управления содержит два модуля: модуль примитивы MMCME2_ADV и автомат управления. Листинги модулей приведены в листингах 1 и 2 соответственно.

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

library unisim;
use unisim.vcomponents.all;

entity Mmcm is
   Port ( ClkIn : in STD_LOGIC;
          ClkOut0 : out STD_LOGIC;
          ClkOut1 : out std_logic;
          ClkOut2 : out std_logic;
          ClkOut3 : out std_logic;
          ClkOut4 : out std_logic;
          ClkOut5 : out std_logic;
          ClkOut6 : out std_logic;
          MmcmLocked : out std_logic;
          Psen : in STD_LOGIC;
          PsIncDec : in STD_LOGIC;
          PsClk : in STD_LOGIC;
          PsDone : out STD_LOGIC);
end Mmcm;

architecture Behavioral of Mmcm is

 signal clkin1 : std_logic;
 signal clkfbout : std_logic;
 signal clkfbout_buf : std_logic;
 signal clkfboutb_unused : std_logic;
 signal clkout0_Sig : std_logic;
 signal clkout0b_unused : std_logic;
 signal clkout1_Sig : std_logic;
 signal clkout1b_unused : std_logic;
 signal clkout2_Sig : std_logic;
 signal clkout2b_unused : std_logic;
 signal clkout3_Sig : std_logic;
 signal clkout3b_unused : std_logic;
 signal clkout4_Sig : std_logic;
 signal clkout5_Sig : std_logic;
 signal clkout6_Sig : std_logic;

 -- Выходы динамического реконфигурирования не используются
 signal do_unused : std_logic_vector(15 downto 0);
 signal drdy_unused : std_logic;

 -- Статусные сигналы не используются
 signal clkfbstopped_unused : std_logic;
 signal clkinstopped_unused : std_logic;

begin

   clkin1_buf : BUFG
   port map
         (O => clkin1,
          I => ClkIn);
```

```
Inst_Mmcme2Adv : MMCME2_ADV
generic map
   (BANDWIDTH => "OPTIMIZED",
   CLKOUT4_CASCADE => FALSE,
   COMPENSATION => "ZHOLD",
   STARTUP_WAIT => FALSE,
   DIVCLK_DIVIDE => 2, --D
   --
   CLKFBOUT_MULT_F => 10.00, --M
   CLKFBOUT_PHASE => 0.000,
   CLKFBOUT_USE_FINE_PS => False,
   --
   CLKOUT0_DIVIDE_F => 1.00, --O0
   CLKOUT0_PHASE => 0.000,
   CLKOUT0_DUTY_CYCLE => 0.500,
   CLKOUT0_USE_FINE_PS => False,
   --
   CLKOUT1_DIVIDE => 40, --O1
   CLKOUT1_PHASE => 0.000,
   CLKOUT1_DUTY_CYCLE => 0.500,
   CLKOUT1_USE_FINE_PS => FALSE,
   --
   CLKOUT2_DIVIDE => 40, --O2
   CLKOUT2_PHASE => 0.000,
   CLKOUT2_DUTY_CYCLE => 0.500,
   CLKOUT2_USE_FINE_PS => True,
   --
   CLKOUT3_DIVIDE => 1, --O3
   CLKOUT3_PHASE => 0.000,
   CLKOUT3_DUTY_CYCLE => 0.500,
   CLKOUT3_USE_FINE_PS => True,
   --
   CLKOUT4_DIVIDE => 1, --O4
   CLKOUT4_PHASE => 0.000,
   CLKOUT4_DUTY_CYCLE => 0.500,
   CLKOUT4_USE_FINE_PS => FALSE,
   --
   CLKOUT5_DIVIDE => 1, --O5
   CLKOUT5_PHASE => 45.000,
   CLKOUT5_DUTY_CYCLE => 0.500,
   CLKOUT5_USE_FINE_PS => FALSE,
   --
   CLKOUT6_DIVIDE => 1, --O6
   CLKOUT6_PHASE => 0.000,
   CLKOUT6_DUTY_CYCLE => 0.500,
   CLKOUT6_USE_FINE_PS => FALSE,
   --
   CLKIN1_PERIOD => 5.0, --Период основного тактового сигнала
   REF_JITTER1 => 0.010)

port map

   (
   -- Выходные порты
   CLKFBOUT => clkfbout,
   CLKFBOUTB => clkfboutb_unused,
   CLKOUT0 => clkout0_Sig,
   CLKOUT0B => clkout0b_unused,
   CLKOUT1 => clkout1_Sig,
   CLKOUT1B => clkout1b_unused,
   CLKOUT2 => clkout2_Sig,
   CLKOUT2B => clkout2b_unused,
   CLKOUT3 => clkout3_Sig,
   CLKOUT3B => clkout3b_unused,
   CLKOUT4 => clkout4_Sig,
   CLKOUT5 => clkout5_Sig,
   CLKOUT6 => clkout6_Sig,
   -- Входные порты
   CLKFBIN => clkfbout_buf,
   CLKIN1 => clkin1,
   CLKIN2 => '0',
   -- Выбор входного тактового порта
   CLKINSEL => '1',
   -- Порты динамической регонфигурации
   DADDR => (others => '0'),
   DCLK => '0',
   DEN => '0',
   DI => (others => '0'),
   DO => do_unused,
   DRDY => drdy_unused,
   DWE => '0',
   -- Порты динамического сдвига фазы
   PSCLK => PsClk,
   PSEN => PsEn,
   PSINCDEC => PsIncDec,
   PSDONE => PsDone,
   -- Другие контрольные и статусные порты
   LOCKED => MmcmLocked,
   CLKINSTOPPED => clkinstopped_unused,
   CLKFBSTOPPED => clkfbstopped_unused,
   PWRDWN => '0',
   RST => '0');

clkf_buf : BUFG
port map
      (O => clkfbout_buf,
      I => clkfbout);

   clkout0_buf : BUFG
   port map
         (O => clkout0,
         I => clkout0_Sig);

   clkout1_buf : BUFG
   port map
         (O => clkout1,
         I => clkout1_Sig);

   clkout2_buf : BUFG
   port map
         (O => clkout2,
         I => clkout2_Sig);

   clkout3_buf : BUFG
   port map
         (O => clkout3,
         I => clkout3_Sig);

   clkout4_buf : BUFG
   port map
         (O => clkout4,
         I => clkout4_Sig);

   clkout5_buf : BUFG
   port map
         (O => clkout5,
         I => clkout5_Sig);

   clkout6_buf : BUFG
   port map
         (O => clkout6,
         I => clkout6_Sig);

end Behavioral;
```

Листинг 1. Текстовое описание блока Mmcm

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

entity PhaseShiftInterface is
   generic(MultPhaseShift : natural := 10); --Выполнить фазовый
сдвиг MultPhaseShift раз
   Port( Up : in std_logic; --Увеличить фазу
      Down : in std_logic; --Уменьшить фазу
      Reset : in std_logic; --Синхронный сброс
      Clk : in std_logic; --Тактовая частота
      PhaseShiftDone : out std_logic; --Окончание выполнения сдви-
га MultPhaseShift раз
      --
      MmcmLocked : in std_logic; --Готовность Mmcm к работе
      PsEn : out std_logic; --Разрешить фазовый сдвиг
      PsClk : out std_logic; --Тактовая частота сигналов контроля
      PsIncDec : out std_logic; --Инкремент/Декремент фазы
      PsDone : in std_logic); --Фазовый сдвиг выполнен
end PhaseShiftInterface;

architecture Behavioral of PhaseShiftInterface is

   signal PsEn_R, PsEn_N : std_logic;
   signal PsIncDec_R, PsIncDec_N : std_logic;
   signal Count_R, Count_N : natural range 0 to MultPhaseShift - 1;
   signal ActUp_R, ActUp_N : std_logic;
   signal ActDown_R, ActDown_N : std_logic;
   signal PhaseShiftDone_R, PhaseShiftDone_N : std_logic;

   type State_Type is (S0, S1, S2, S3, S4, S5, S6);
   signal State_R, State_N : State_Type;

begin

   PsClk_Process: process(Clk)
      begin
         PsClk <= Clk;
   end process;

   FsmStateR_Process: process(Clk)
      begin
         if rising_edge(Clk) then
            if Reset = '1' then
               State_R <= S0;
            else
               State_R <= State_N;
            end if;
         end if;
      end process;

   FsmSync_Process: process(Clk)
      begin
         if rising_edge(Clk) then
            PsEn_R <= PsEn_N;
            PsIncDec_R <= PsIncDec_N;
            ActUp_R <= ActUp_N;
            ActDown_R <= ActDown_N;
            Count_R <= Count_N;
            PhaseShiftDone_R <= PhaseShiftDone_N;
         end if;
      end process;

   FsmComb_Process: process(Up, Down, PsEn_R, PsIncDec_R,
PsDone, PhaseShiftDone_R,
      State_R, ActUp_R, ActDown_R, Count_R, MmcmLocked)
      begin
         PsEn_N <= PsEn_R;
         PsINcDec_N <= PsINcDec_R;
         State_N <= State_R;
         ActUp_N <= ActUp_R;
         ActDown_N <= ActDown_R;
         Count_N <= Count_R;
      PhaseShiftDone_N <= PhaseShiftDone_R;

   case State_R is
      when S0 => if MmcmLocked = '1' then
         if Up = '1' then
            PsIncDec_N <= '1';
            PsEn_N <= '1';
            ActUp_N <= '1';
            State_N <= S1;
         elsif Down = '1' then
            PsIncDec_N <= '0';
            PsEn_N <= '1';
            ActDown_N <= '1';
            State_N <= S1;
         else
            State_N <= S0;
            PsEn_N <= '0';
            PsIncDec_N <= '0';
            ActUp_N <= '0';
            ActDown_N <= '0';
            Count_N <= 0;
            PhaseShiftDone_N <= '0';
         end if;
      end if;
      when S1 => PsEn_N <= '0';
         PsIncDec_N <= '0';
         State_N <= S2;

      when S2 => if PsDone = '1' then
         State_N <= S3;
      end if;

      when S3 => if Count_R = MultPhaseShift - 1 then
            State_N <= S0;
            PhaseShiftDone_N <= '1';
         else
            if ActUp_R = '1' then
               PsEn_N <= '1';
               PsIncDec_N <= '1';
               State_N <= S1;
         end if;
            if ActDown_R = '1' then
               PsEn_N <= '1';
               PsIncDec_N <= '0';
               State_N <= S1;
            end if;
            Count_N <= Count_R + 1;
      end if;

      when others => State_N <= S0;

      end case;
   end process;

   PsEn <= PsEn_R;
   PsIncDec <= PsIncDec_R;
   PhaseShiftDone <= PhaseShiftDone_R;

end Behavioral;
```

Листинг 2. Текстовое описание модуля PhaseShiftInterface

Динамический сдвиг фазы в модуле Mmcm осуществляется для выхода ClkOut2, поскольку соответствующий параметр CLKOUT2_USE_FINE_PS имеет значение True. При входной частоте 200 МГц выходная частота будет:

$$F_{ClkOut2} = F_{CLKIN}(M/(D\times O)) = \\ = 200\times10^6(10/(2\times40)) = 25\ \text{МГц}. \quad (3)$$

Для демонстрации выполнения фазового сдвига выход ClkOut1 также настроен на час-

тоту 25 МГц, но режим динамического фазового сдвига в нем отключен. ClkOut1 будет использован в качестве опорного для измерения фазового сдвига.

Блок PhaseShiftInterface отвечает за формирование и контроль управляющих и статусных сигналов, изображенных на рис. 3. Параметр MultPhaseShift определяет, сколько раз необходимо выполнить фазовый сдвиг, после поступления команды на соответствующий вход Up — инкрементировать фазу или Down — декрементировать фазу. Выход PhaseShiftDone является статусным и устанавливается в «1» на 1 такт после выполнения фазового сдвига MultPhaseShift раз.

Входы Clk и Reset отвечают соответственно за тактовую частоту и сброс.

Оба модуля Mmcm и PhaseShiftInterface объединены в один модуль верхнего уровня DinamicPhaseShift, текстовая версия которого приведена в листинге 3.

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

entity DinamicPhaseShift is
 generic(MultPhaseShift : natural := 559); --Выполнить фазовый
сдвиг MultPhaseShift раз
   Port( Up : in std_logic;
         Down : in std_logic;
         Reset : in std_logic;
         Clk : in std_logic;
         PhaseShiftDone : out std_logic;
         MmcmLocked : out std_logic;
         ClkOut0 : out std_logic;
         ClkOut1 : out std_logic;
         ClkOut2 : out std_logic;
         ClkOut3 : out std_logic;
         ClkOut4 : out std_logic;
         ClkOut5 : out std_logic;
         ClkOut6 : out std_logic
   );
end DinamicPhaseShift;

architecture Behavioral of DinamicPhaseShift is

   COMPONENT PhaseShiftInterface
      generic(MultPhaseShift : natural := 10);
   PORT(
      Up : in std_logic;
      Down : in std_logic;
      Reset : in std_logic;
      Clk : in std_logic;
      MmcmLocked : in std_logic;
      PsDone : in std_logic;
      PhaseShiftDone : out std_logic;
      PsEn : out std_logic;
      PsClk : out std_logic;
      PsIncDec : out std_logic
      );
   END COMPONENT;

   COMPONENT Mmcm
   PORT(
      ClkIn : in std_logic;
      Psen : in std_logic;
      PsIncDec : in std_logic;
      PsClk : in std_logic;
      ClkOut0 : out std_logic;
      ClkOut1 : out std_logic;
      ClkOut2 : out std_logic;
      ClkOut3 : out std_logic;
      ClkOut4 : out std_logic;
      ClkOut5 : out std_logic;
      ClkOut6 : out std_logic;
      MmcmLocked : out std_logic;
      PsDone : out std_logic
      );
   END COMPONENT;

   signal MmcmLocked_Sig : std_logic;
   signal PsEn_Sig : std_logic;
   signal PsClk_Sig : std_logic;
   signal PsDone_Sig : std_logic;
   signal PsIncDec_Sig : std_logic;

begin
```

```
   Inst_PhaseShiftInterface: PhaseShiftInterface
      GENERIC MAP(MultPhaseShift => MultPhaseShift)
      PORT MAP(
      Up => Up,
      Down => Down,
      Reset => Reset,
      Clk => Clk,
      PhaseShiftDone => PhaseShiftDone,
      MmcmLocked => MmcmLocked_Sig,
      PsEn => PsEn_Sig,
      PsClk => PsClk_Sig,
      PsIncDec => PsIncDec_Sig,
      PsDone => PsDone_Sig
   );

   Inst_Mmcm: Mmcm PORT MAP(
      ClkIn => Clk,
      ClkOut0 => ClkOut0,
      ClkOut1 => ClkOut1,
      ClkOut2 => ClkOut2,
      ClkOut3 => ClkOut3,
      ClkOut4 => ClkOut4,
      ClkOut5 => ClkOut5,
      ClkOut6 => ClkOut6,
      MmcmLocked => MmcmLocked_Sig,
      Psen => Psen_Sig,
      PsIncDec => PsIncDec_Sig,
      PsClk => PsClk_Sig,
      PsDone => PsDone_Sig
   );

   MmcmLocked <= MmcmLocked_Sig;

end Behavioral;
```

Листинг 3. Текстовое описание модуля DinamicPhaseShift

Проверка модуля

Проверка описанного модуля выполнялась на отладочном комплекте KC705 фирмы Xilinx. При проверке осуществлялся фазовый сдвиг на 90°, при этом значение параметра MultPhaseShift равнялось 559, поскольку один фазовый сдвиг приводил к смещению сигнала на:

$$\Delta\varphi = (360°/(56\times O)) = 0{,}161°, \qquad (4)$$

$$\text{MultPhaseShift} = 90°/(\Delta\varphi) \approx 559. \qquad (5)$$

Инкремент/декремент фазы осуществлялся по нажатии кнопок, поэтому для устранения их дребезга добавлены дополнительные модули Debounce, которые также дополнительно формировали по нажатии кнопки импульс длительностью в один такт (листинг 4).

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

entity Debounce is
   Port( Inp : in std_logic;
         Clk : in std_logic;
         Reset : in std_logic;
         Outp : out std_logic);
end Debounce;

architecture Behavioral of Debounce is

 constant Waiting : natural := 4*10**7;
 signal Count_N, Count_R : integer range 0 to Waiting;
 signal start : std_logic;

 type StateType is (S0, S1, S2, S3);
 signal State_R, State_N : StateType;
 signal Outp_R, Outp_N : std_logic;

begin

 process(Clk)
   begin
      if rising_edge(Clk) then
         if Reset = '1' then
            State_R <= S0;
         else
            Outp_R <= Outp_N;
            State_R <= State_N;
            Count_R <= Count_N;
         end if;
      end if;
   end process;

 process(Inp, Outp_R, Count_R, State_R)
   begin

   State_N <= State_R;
   Count_N <= Count_R;
   Outp_N <= Outp_R;

   case State_R is
   when S0 => Outp_N <= '0';
      Count_N <= 0;
      if Inp = '1' then
         State_N <= S1;
      end if;

   when S1 => if Count_R = Waiting then
         State_N <= S2;
         Outp_N <= '1';
         Count_N <= 0;
      else
         Count_N <= Count_R + 1;
         State_N <= S1;
      end if;

   when S2 => Outp_N <= '0';
      if Inp = '0' then
         State_N <= S3;
      end if;

   when S3 => if Count_R = Waiting then
         State_N <= S0;
         Outp_N <= '0';
         Count_N <= 0;
      else
         Count_N <= Count_R + 1;
         State_N <= S3;
      end if;
      end case;
   end process;

   Outp <= Outp_R;

end Behavioral;
```

Листинг 4. Текстовое описание модуля Debounce

Описание проекта верхнего уровня Buf_DinamicPhaseShift для проверки работоспособности описанных модулей приведено в листинге 5.

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

Library UNISIM;
use UNISIM.vcomponents.all;

entity Buf_DinamicPhaseShift is
   port(
         SysClkP : in std_logic;
         SysClkN : in std_logic;
         Reset : in std_logic;
         Up : in std_logic;
         Down : in std_logic;
         ClkOut1 : out std_logic;
         ClkOut2 : out std_logic;
         PhaseShiftDone : out std_logic
   );
end Buf_DinamicPhaseShift;

architecture Behavioral of Buf_DinamicPhaseShift is

   COMPONENT DinamicPhaseShift
   PORT(
         Up : IN std_logic;
         Down : IN std_logic;
         Reset : IN std_logic;
         Clk : IN std_logic;
         PhaseShiftDone : OUT std_logic;
         MmcmLocked : OUT std_logic;
         ClkOut0 : OUT std_logic;
         ClkOut1 : OUT std_logic;
         ClkOut2 : OUT std_logic;
         ClkOut3 : OUT std_logic;
```

```
        ClkOut4 : OUT std_logic;
        ClkOut5 : OUT std_logic;
        ClkOut6 : OUT std_logic
    );
    END COMPONENT;

    COMPONENT Debounce
    PORT(
        Inp : IN std_logic;
        Reset : IN std_logic;
        Clk : IN std_logic;
        Outp : OUT std_logic
      );
    END COMPONENT;

      signal Clk_Buf : std_logic;
      signal Reset_Buf : std_logic;
      signal Up_Buf : std_logic;
      signal Down_Buf : std_logic;
      signal ClkOut1_Buf : std_logic;
      signal ClkOut2_Buf : std_logic;
      signal PhaseShiftDone_Buf : std_logic;
      signal UpDebounce_Sig : std_logic;
      signal DownDebounce_Sig : std_logic;

begin

    Inst_DinamicPhaseShift: DinamicPhaseShift PORT MAP(
      Up => UpDebounce_Sig,
      Down => DownDebounce_Sig,
      Reset => Reset_Buf,
      Clk => Clk_Buf,
      PhaseShiftDone => PhaseShiftDone_Buf,
      MmcmLocked => Open,
      ClkOut0 => Open,
      ClkOut1 => ClkOut1_Buf,
      ClkOut2 => ClkOut2_Buf,
      ClkOut3 => Open,
      ClkOut4 => Open,
      ClkOut5 => Open,
      ClkOut6 => Open
    );

    Inst_DebounceUp: Debounce PORT MAP(
      Inp => Up_Buf,
      Clk => Clk_Buf,
      Reset => Reset_Buf,
      Outp => UpDebounce_Sig
    );

    Inst_DebounceDown: Debounce PORT MAP(
      Inp => Down_Buf,
      Clk => Clk_Buf,
      Reset => Reset_Buf,
      Outp => DownDebounce_Sig
    );

    Inst_SysClk_Ibufgds : IBUFGDS
      generic map (
        DIFF_TERM => FALSE,
        IBUF_LOW_PWR => TRUE,
        IOSTANDARD => "DEFAULT")
      port map (
        O => Clk_Buf,
        I => SysClkP,
        IB => SysClkN
      );

    Inst_Reset_Ibuf : IBUF
      generic map (
        IBUF_LOW_PWR => TRUE,
        IOSTANDARD => "DEFAULT")
      port map (
        O => Reset_Buf,
        I => Reset
      );

    Inst_Up_Ibuf : IBUF
      generic map (
        IBUF_LOW_PWR => TRUE,
        IOSTANDARD => "DEFAULT")
      port map (
        O => Up_Buf,
        I => Up
      );

    Inst_Down_Ibuf : IBUF
      generic map (
        IBUF_LOW_PWR => TRUE,
        IOSTANDARD => "DEFAULT")
      port map (
        O => Down_Buf,
        I => Down
      );

    Inst_ClkOut1_Obuf : OBUF
      generic map (
```

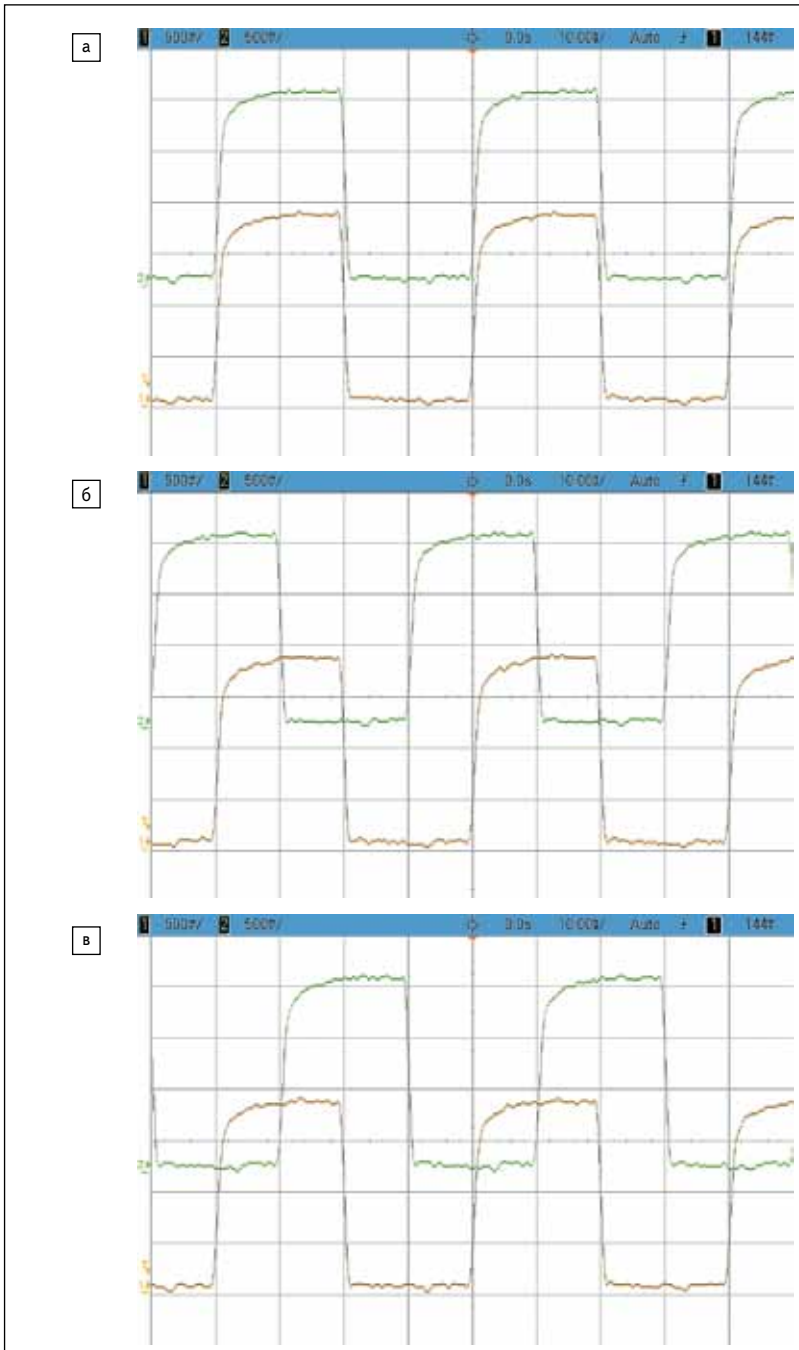

Рис. 4. Результат испытаний модуля динамического сдвига фазы: а) исходное положение; б) сдвиг фазы вверх; в) сдвиг фазы вниз

```
        DRIVE => 12,
        IOSTANDARD => "DEFAULT",
        SLEW => "SLOW")
      port map (
        O => ClkOut1,
        I => ClkOut1_Buf
      );

    Inst_ClkOut2_Obuf : OBUF
      generic map (
        DRIVE => 12,
        IOSTANDARD => "DEFAULT",
        SLEW => "SLOW")
      port map (
        O => ClkOut2,
        I => ClkOut2_Buf
      );

    Inst_PhaseShiftDone_Obuf : OBUF
      generic map (
        DRIVE => 12,
        IOSTANDARD => "DEFAULT",
        SLEW => "SLOW")
      port map (
        O => PhaseShiftDone,
        I => PhaseShiftDone_Buf
      );
end Behavioral;
```

Листинг 5. Текстовое описание модуля Buf_DinamicPhaseShift

Результаты испытаний приведены на рис. 4, где показаны снятые осциллограммы. Желтым отмечен опорный сигнал, зеленым — сигнал, для которого был выполнен фазовый сдвиг на 90°.

Заключение

В статье был предложен вариант исполнения модуля динамического сдвига фазы для блока тактовой синхронизации MMCM, являющегося частью блока управления тактовой частотой CMT для ПЛИС 7-й серии фирмы Xilinx. Представлены текстовые описания соответствующих модулей и приведены результаты натурных испытаний предлагаемого решения. По запросу автор может выслать по электронной почте описание всех модулей.

Литература

1. 7 Series FPGAs Clocking Resources. UG472 (v1.9), Xilinx, Inc. Aug. 8, 2013.
2. www.xilinx.com/support/documentation/7_series.htm#156339