Построение узла синтезатора синхросигналов различной частоты

в логическом проекте ПЛИС серии Spartan-3E фирмы Xilinx

Николай БОРИСЕНКО
fpga-mechanic@rambler.ru

В современных цифровых устройствах массово применяются интегральные микросхемы класса «система на кристалле» (СнК). Современные ПЛИС по уровню интеграции и набору встроенных аппаратных ядер позволяют проектировать на их основе системы на программируемом кристалле, содержащие процессорные ядра, блоки памяти, периферийные модули и каналы интерфейсов ввода/вывода. Для синхронизации проектов такой сложности в объеме кристалла необходимо формировать множество тактовых сигналов, обладающих различными параметрами. В статье рассмотрен пример построения узла синтезатора синхросигналов из одной опорной частоты, построенного в базисе примитивов ПЛИС серии Spartan-3E фирмы Xilinx.

При проектировании современных цифровых устройств зачастую приходится решать задачу преобразования тактовых частот. Это связано с тем, что различные функциональные узлы, блоки и интерфейсы должны функционировать на фиксированных, не кратных между собой частотах. Например, в составе системы на кристалле процессорное ядро может работать на частоте 1833 МГц, интерфейс памяти — на 667 МГц, каналы PCI express — на 100 МГц, шина PCI — на 66 МГц, контроллеры интерфейсов Ethernet и USB — на 125 и 48 МГц соответственно. Все тактовые частоты должны быть сформированы на основе одной опорной частоты, заданной кварцевым генератором. Эту функцию выполняет специальный аппаратный узел — синтезатор синхросигналов.

В общем случае синтезатор синхросигналов преобразует входную опорную частоту в ряд выходных частот путем умножения и деления опорной частоты на натуральные или дробные положительные коэффициенты. В большинстве устройств коэффициенты деления и умножения жестко заданы текущей конфигурацией системы и не могут изменяться во время штатной работы. Тем не менее бывают исключения, примером которых служит технология Cool-and-Quiet фирмы AMD, реализованная в микропроцессорах архитектуры K8 и K10. (Динамическое управление тактовой частотой позволяет контролировать потребление энергии и нагрев кристалла микропроцессора.).

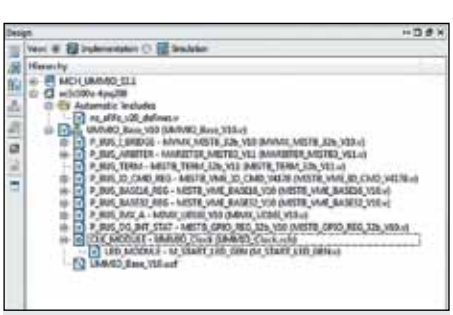

Рис. 1. Иерархия проекта ПЛИС XC3S500E

В ПЛИС для построения синтезаторов синхросигналов предназначены специальные ресурсы кристалла, имеющие различные возможности и названия у разных производителей элементной базы. В сериях Spartan-3/3E и Spartan-6 фирмы Xilinx реализованы примитивы DCM (Digital Clock Manager), позволяющие преобразовывать синхросигналы путем настройки параметров частоты, фазы и скважности [1, 2, 3, 7].

Предлагается рассмотреть проект для ПЛИС XC3S500E, в котором в качестве основного синхросигнала использована тактовая частота 64 МГц, полученная из опорной частоты 50 МГц. Иерархия проекта показана на рис. 1.

В верхнем модуле UMMIO_Base_V10 описаны соединение вложенных модулей, а также синхронизаторы цепей начальной установки. Не все вложенные модули, отображенные на рис. 1, имеют отношение к предмету статьи. Эти модули определяют пользовательский набор функциональных блоков и узлов, требующих синхронизации различными тактовыми частотами. Этот пример взят из специализированного проекта, реализующего заданные функции в ПЛИС.

Верхний модуль иерархии проекта представлен моделью на языке Verilog (подключение не описанных в статье модулей из кода удалено):

```
`timescale 1ns / 1ps
//////////////////////////////////////////
// Engineer: FPGA-Mechanic
//
// Create Date: 13:52:22 03/22/2013
// Design Name: UMM IO XC3S500E
// Module Name: UMMIO_Base_V10 - Top Module
// Project Name: UMMIO_3S500E_V1
// Target Devices: XC3S500E-PQ208
// Tool versions: Xilinx DS 14.4
//
// Revision: 1.0 (22.03.2013)
// Revision 1.0 - File Created
//////////////////////////////////////////
module UMMIO_Base_V10(
    inout SPRT3_CLK_IN,
    output L_8,
    inout AB_NRST_BUF,
    inout VMX_RST,
    // Other Ports
    // ...
    input CONF_DONE
    );

// Internal signals declaration:
    // Inputs-System
    wire CLK; // 64MHz
    wire ASYNCH_RST; // Internal Asynch. Reset
    // Local-System
    reg RST, RST_B; //Synch. Reset 64MHz;
//----------------------------------------
```

```
//----------------------------------------------

//----------------------------------------------
   // 64MHz Reset Synchronizer
 always @ (posedge CLK, posedge ASYNCH_RST)
   if(ASYNCH_RST)
      begin
         RST <= 1;
         RST_B <= 1;
      end
   else
      begin
         RST_B <= 0;
         RST <= RST_B;
      end
//----------------------------------------------
 UMMIO_Clock CLK_MODULE (
 .SPRT3_CLK_IN(SPRT3_CLK_IN),
 .L_8(L_8),
 .AB_NRST_BUF(AB_NRST_BUF),
 .VMX_RST(VMX_RST),
 .CONF_DONE(CONF_DONE),
 .ASYNCH_RST(ASYNCH_RST),
 .CLK_50M(),
 .CLK_100M(),
 .CLK_80M(),
 .CLK_160M(),
 .CLK_75M(CLK_VMX),
 .CLK_32M(),
 .CLK_64M(CLK),
 .CLK_16M(),
 .CLK_48M(),
 .CLK_33M(),
 .CLK_66M(),
 .CLK_16M7(),
 .CLK_83M(),
 .RST_50M()
 );
//----------------------------------------------

//----------------------------------------------

//----------------------------------------------
endmodule
```

Рассмотрим модуль UMMIO_Clock, выделенный на рис. 1 пунктиром. Именно этот модуль является синтезатором синхросигналов, описанным в схемотехническом редакторе. Схема модуля состоит из пяти листов, они показаны на рис. 2.

Ввод сигнала опорной частоты осуществляется через входную цепь SPRT3_CLK_IN, соединяющую двунаправленный маркер с резистором PULLDOWN и входным буфером IBUFG. Опорная частота составляет 50 МГц и формируется внешним кварцевым генератором. Входной буфер ретранслирует опорную частоту на внутренний сигнал CLKI_50M.

Синтезатор синхросигналов имеет три входа асинхронной начальной установки: AB_NRST_BUF, VMX_RST и CONF_DONE. Все три входа имеют активный низкий уровень сигнала. Для установки проекта ПЛИС в исходное состояние (сброса) достаточно подачи низкого уровня хотя бы на один из трех перечисленных входов. Первые два входа используются в проекте для ввода сигналов сброса с двух системных шин, а сигнал CONF_DONE предназначен для ожидания загрузки конфигурации другой ПЛИС FPGA (сигнал CONF_DONE для ПЛИС семейств Cyclone — Cyclone-V фирмы Altera либо сигнал DONE для ПЛИС архитектуры FPGA фирмы Xilinx). Логический вентиль OR3B3 формирует из трех сигналов AB_NRST_BUF, VMX_RST и CONF_DONE общий внутренний сигнал асинхронного сброса ASYNCH_RST_INPUT, имеющий активный высокий уровень.

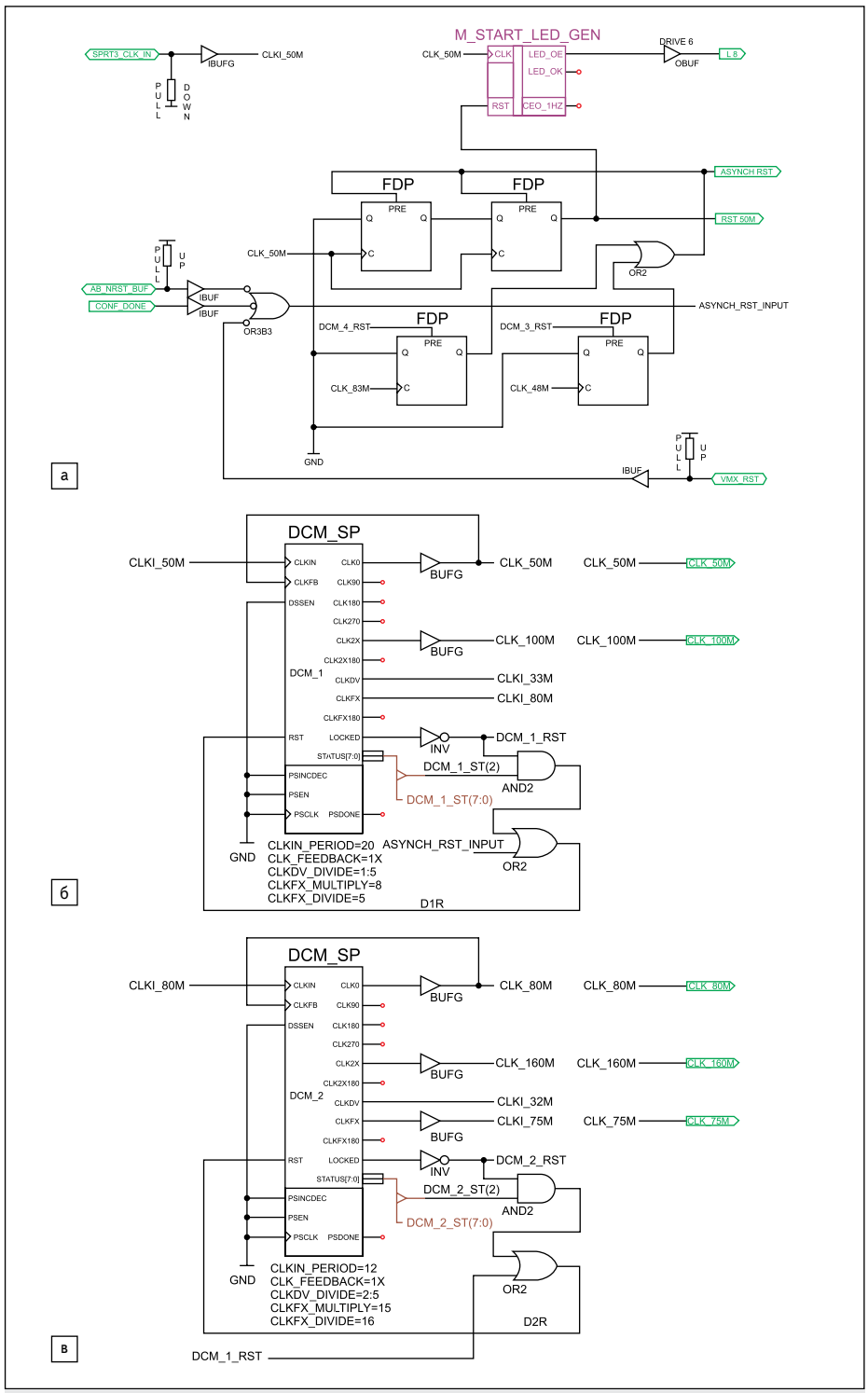

Рис. 2. Схема синтезатора синхросигналов: а) лист 1; б) лист 2; в) лист 3

Принятый с вывода ПЛИС через входной буфер IBUFG сигнал опорной частоты CLKI_50M подается на вход CLKIN примитива DCM_SP, имеющего имя экземпляра DCM_1 (рис. 2б). Примитив DCM_SP имеет атрибут CLKIN_PERIOD, задающий период входной частоты в наносекундах. Для DCM_1 входная частота составляет 50 МГц, соответственно, атрибуту CLKIN_PERIOD присвоено значение 20.

К рассмотрению остальных цепей первого листа (рис. 2) предлагаем вернуться после изучения примеров синтеза тактовых частот с использованием примитивов DCM_SP.

Примитив DCM состоит из двух функциональных узлов: схемы автоподстройки частоты DLL (Delay Locked Loop) и цифрового синтезатора частоты DFS (Digital Frequency Synthesizer) [1–3].

Узел DLL формирует тактовые сигналы на выходах CLK0, CLK90, CLK180, CLK270,

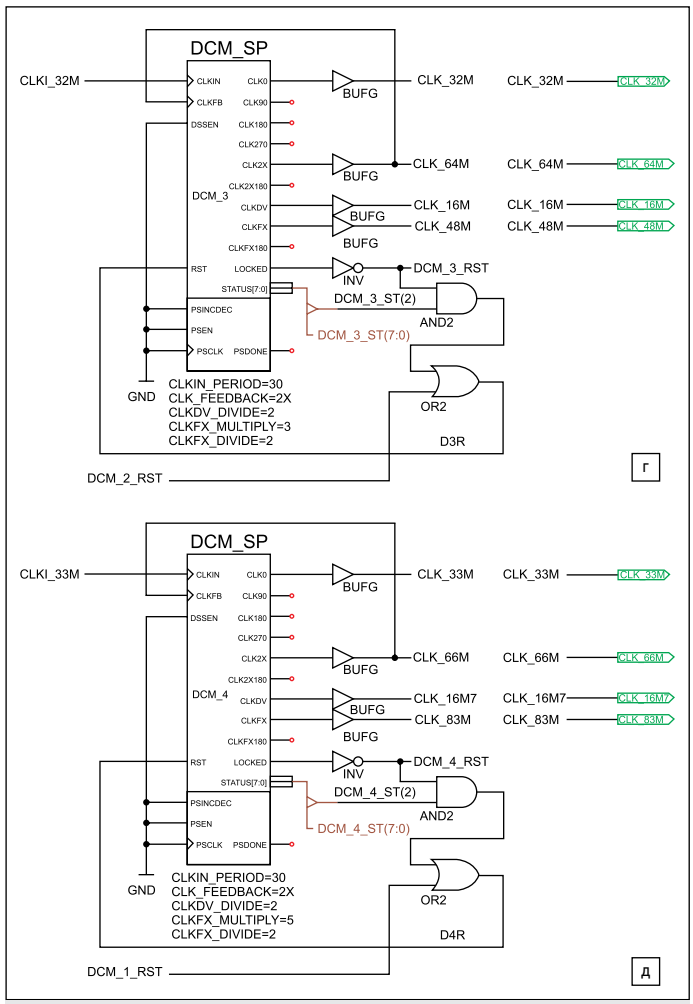

Рис. 2. Схема синтезатора синхросигналов: г) лист 4; д) лист 5

CLK2X, CLK2X180 и CLKDV. При этом узел DLL имеет вход обратной связи CLKFB, необходимый для коррекции фаз выходных синхросигналов. На выходах CLK0, CLK90, CLK180 и CLK270 формируются синхросигналы с частотой, равной входной частоте на входе CLKIN, но различающиеся сдвигом фаз на 0°, 90°, 180° и 270° соответственно. На выходах CLK2X и CLK2X180 формируются противофазные синхросигналы с удвоенной частотой относительно входной частоты CLKIN. Выход CLKDV позволяет формировать синхросигнал, полученный путем деления частоты входного сигнала CLKIN на коэффициенты: 1,5, 2, 2,5, 3, 3,5, 4, 4,5, 5, 5,5, 6, 6,5, 7, 7,5, 8, 9, 10, 11, 12, 13, 14, 15 и 16. Выбор коэффициента деления определяется атрибутом CLKDV_DIVIDE.

На вход обратной связи могут подаваться выходные сигналы DLL, имеющие частоту, равную входной частоте, и сигналы с удвоенной частотой. За выбор частоты в цепи обратной связи примитивов DCM отвечает атрибут CLK_FEEDBACK. В первом случае данный атрибут устанавливается в значение 1X, а во втором — в значение 2X. Как показано на рис. 2б, на вход обратной связи CLKFB примитива DCM_1 подан сигнал CLK_50M, сформированный выходом CLK0, вследствие чего атрибуту CLK_FEEDBACK задано значение 1X. Буфер BUFG соответствует ресурсам глобальной сети синхронизации кристалла, обеспечивающим распространение сигнала в объеме ПЛИС с минимальными задержками.

С выхода CLK2X примитива DCM_1 снимается синхросигнал CLK_100M с частотой 100 МГц, а с выхода CLKDV — сигнал CLKI_33M с частотой 33,333 МГц, полученной делением входной частоты на коэффициент 1,5 (атрибут CLKDV_DIVIDE установлен в 1,5).

Узел DFS позволяет синтезировать на выходах CLKFX и CLKFX180 противофазные синхросигналы с частотой, полученной путем умножения входной частоты CLKIN на коэффициент, заданный атрибутом CLKFX_MULTIPLY, и последующего деления на коэффициент, заданный атрибутом CLKFX_DIVIDE. Таким образом, выходная частота синтезатора определяется выражением:

$$f_{CLKFX} = f_{CLKIN}(CLKFX\_MULTIPLY/CLKFX\_DIVIDE).$$

Коэффициент умножения определяется натуральным атрибутом CLKFX_MULTIPLY, допускающим установку значений от 2 до 32. Коэффициент деления задается натуральным атрибутом CLKFX_DIVIDE из диапазона от 1 до 32. При установке значений атрибутов CLKFX_MULTIPLY и CLKFX_DIVIDE следует учитывать технологические ограничения выходной частоты, зависящие от серии ПЛИС и степпинга ядра. Например, в серии Spartan-3E для кристаллов со степпингом 0 выходные частоты синтезатора были ограничены диапазоном [5:90] МГц в низкочастотном режиме и диапазоном [220:307] МГц в высокочастотном режиме. В более поздних кристаллах со степпингом 1 выходные частоты CLKFX и CLKFX180 имеют общий диапазон ограничения [5:311] МГц. Подробно частотные ограничения примитивов DCM для современных серий ПЛИС Spartan фирмы Xilinx приведены в таблице.

Таблица. Частотные ограничения примитивов DCM

Серия ПЛИС	Частота, МГц				
	CLKIN	CLK0	CLK2X	CLKDV	CLKFX
Spartan-3	[18:167] [48:280]	[18:167] [48:280]	[36:334]	[1.125:110] [3:185]	[18:210] [210:307]
Spartan-3E Stepping-0	[5:90]	[5:90]	[10:180]	[0.3125:60]	[5:90] [220:307]
Spartan-3E Stepping-1	[5:240]	[5:240]	[10:311]	[0.3125:160]	[5:311]
Spartan-6	[5:250]	[5:250]	[10:334]	[0.3125:166]	[5:333]

Приведенные значения не полностью характеризуют ограничения частот примитивов DCM. Например, ограничения для серии Spartan-6 соответствуют классу быстродействия — 2. Максимальные частоты сигналов на выходах CLK90 и CLK270 ниже максимальных выходных частот CLK0 и CLK180. За более подробной информацией при построении синтезатора тактовых частот следует обращаться к документации [1, 2, 4].

Рис. 3. Плата с инженерным образцом ПЛИС Spartan-3E

В рассматриваемом проекте для ПЛИС Spartan-3E (рис. 1) установлен инженерный образец ПЛИС XC3S500E-4PQ208, выпущенный в 2005 году и не имеющий степпинга ядра (рис. 3). Для инженерных образцов действуют ограничения, аналогичные степпингу 0 ядра. По этим причинам для примитива DCM_1 была выбрана выходная частота CLKFX, составляющая 80 МГц (диапазон частот от 5 до 90 МГц). Такая частота получена путем задания атрибутов CLKFX_MULTIPLY = 8 и CLKFX_DIVIDE = 5 (рис. 2а).

Элемент DCM_1 формирует два промежуточных сигнала — CLKI_33M и CLKI_80M — с частотами 33 и 80 МГц соответственно. Эти сигналы используются для передачи частоты на другие примитивы DCM.

На рис. 2в показан третий лист схемы синтезатора частот. Примитив DCM_2 получает опорную частоту 80 МГц от примитива DCM_1 по локальной цепи CLKI_80M. Для элемента DCM_2 атрибут CLKIN_PERIOD установлен в значение 12, наиболее близкое к периоду входной частоты (12,5 нс). Обратная связь использует глобальный буфер синхросигнала CLK_80M, формируемого выходом CLK0. По этой причине атрибут CLK_FEEDBACK установлен в значение 1X.

Путем деления входной частоты на коэффициент 2,5 на выходе CLKDV получен локальный сигнал CLKI_32M с частотой 32 МГц. Для этого атрибуту CLKDV_DIVIDE задано значение 2,5.

На выходе CLK2X элемента DCM_2 формируется синхросигнал CLK160M, имеющий частоту 160 МГц.

Узел DFS элемента DCM_2 настроен на генерацию частоты 75 МГц, поступающей с выхода CLKFX на глобальную цепь CLK_75M. Коэффициенты умножения и деления для синтеза частоты 75 МГц из опорной частоты 80 МГц составляют 15 и 16 соответственно. Таким образом, атрибуту CLKFX_MULTIPLY задано значение 15, а CLKFX_DIVIDE — значение 16.

Примитив с именем DCM_3, показанный на четвертом листе (рис. 2г), получает входную опорную частоту 32 МГц на входе CLKIN с локального сигнала CLKI_32M, сформированного элементом DCM_2. Атрибут CLKIN_PERIOD установлен в значение 30, соответствующее частоте 33,333 МГц.

В отличие от DCM_1 и DCM_2 примитив DCM_3 в цепи обратной связи по входу CLKFB использует удвоенную частоту 64 МГц, полученную на выходе CLK2X. По этой причине атрибут CLK_FEEDBACK установлен в значение 2X.

На выходе CLKDV элемента DCM_3 формируется частота 16 МГц, полученная делением входной частоты на коэффициент CLKDV_DIVIDE = 2.

Синтезатор DFS формирует на выходе CLKFX частоту 48 МГц, полученную путем умножения на 3 и деления на 2. Для этого атрибуту CLKFX_MULTIPLY задано значение 3, а CLKFX_DIVIDE — значение 2.

Примитив с именем DCM_4, показанный на пятом листе (рис. 2д), получает входную опорную частоту 33 МГц на входе CLKIN с локального сигнала CLKI_33M, сформированного элементом DCM_1. Атрибут CLKIN_PERIOD установлен в значение 30, соответствующее входной частоте.

Аналогично DCM_3, примитив DCM_4 в цепи обратной связи по входу CLKFB использует удвоенную частоту 66,667 МГц, полученную на выходе CLK2X. По этой причине атрибут CLK_FEEDBACK установлен в значение 2X.

На выходе CLKDV элемента DCM_4 формируется частота 16,667 МГц, полученная делением входной частоты на коэффициент CLKDV_DIVIDE = 2.

Синтезатор DFS формирует на выходе CLKFX частоту 83,333 МГц, полученную путем умножения несущей частоты 33,333 МГц на 5 и деления на 2. Для этого атрибуту CLKFX_MULTIPLY задано значение 5, а CLKFX_DIVIDE — значение 2.

Судя по описанию схемы синтезатора частот, использование четырех примитивов DCM в составе ПЛИС XC3S500E позволяет из одной опорной частоты 50 МГц получить множество синхросигналов с разными, не кратными между собой частотами. При построении схем синтеза тактовых частот в объеме ПЛИС следует учитывать жесткие

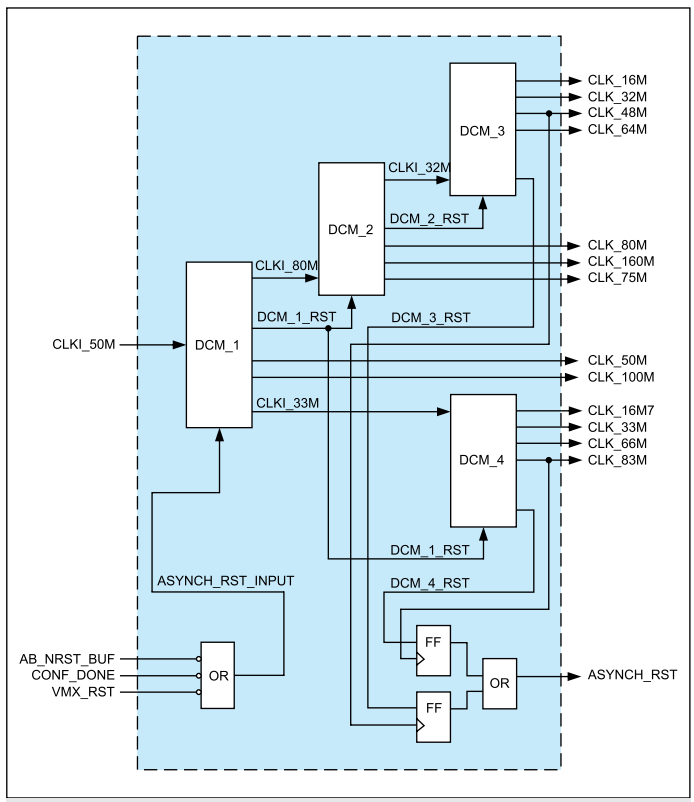

Рис. 4. Каскадное соединение примитивов DCM_SP

ограничения частот на входах и выходах технологических примитивов DCM. Все частотные диапазоны описаны в разделах “Switching Characteristics — Digital Clock Manager (DCM) Timing” документации на выбранную серию ПЛИС.

Помимо описанных входов и выходов примитивов DCM_SP, в рассматриваемом синтезаторе частот используется вход асинхронного сброса RST, выход STATUS [2] и выход LOCKED. Подача высокого уровня на вход RST приводит к сбросу узлов DLL и DFS, сопровождающемуся прекращением генерации выходных частот. Высокий уровень на выходе STATUS [2] отражает остановку работы узла DFS (DFS output CLKFX is stopped). Высокий уровень на выходе LOCKED сигнализирует об устойчивой работе примитива DCM, а также об установившихся стабильных выходных сигналах. Автоподстройка частоты в DCM происходит с некоторой временной задержкой относительно установки стабильного сигнала опорной частоты на входе CLKIN. В связи с этим, согласно рекомендациям из [3] (Figure 2-11: Spartan FPGA DCM DFS Lock Logic), на всех четырех листах схемы синтезатора частот, содержащих примитивы DCM_SP, реализованы рекомендуемые цепи обратной связи.

В рамках подключения одного примитива DCM входной сигнал сброса подается на логический вентиль OR2, а выходной сигнал сброса снимается с инверсии выхода LOCKED. Выходной сигнал необходим для начальной установки всех функциональных блоков и узлов, использующих синхросигнал, сгенерированный на выходе примитива DCM.

Так как в рассмотренном примере примитивы соединены каскадом, в котором элементы DCM_2 и DCM_4 используют выходной сигнал DCM_1, а DCM_3 в качестве опорной частоты использует выходной сигнал DCM_2, сигналы сброса должны быть переданы по каскаду в той же последовательности. Каскадное соединение примитивов DCM отражено на рис. 4. В последнюю очередь сигналы сброса будут деактивированы на выходах элементов DCM_3 и DCM_4. Поэтому инверсные сигналы с выходов LOCKED этих элементов используются для генерации выходного сигнала ASYNCH_RST. Низкий уровень на выходе ASYNCH_RST свидетельствует об отсутствии входного сигнала сброса и стабильности всех выходных частот синтезатора синхросигналов.

Модуль M_START_LED_GEN представляет собой узел управления дискретным светодиодом, подключенным к выводу ПЛИС. Свечению соответствует низкий логический сигнал на выходе LED_OE. Модуль M_START_LED_GEN синхронизируется от частоты 50 МГц и выдает на выходе пять импульсов с частотой 1 Гц. В исходном состоянии светодиод светится, потом мигает четыре раза, а на пятом такте гаснет окончательно. Сброс модуля путем установки высокого уровня на входе RST повторяет описанный процесс. Этот функциональный узел служит визуальным индикатором загрузки проекта в ПЛИС и наличия стабильных тактовых частот на выходах синтезатора.

Синтезируемый код узла управления диагностическим светодиодом описан на языке Verilog в файле *M_START_LED_GEN.v* следующим образом:

```
`timescale 1ns / 1ps
//////////////////////////////////////
// Create Date: 13:58:21 10/12/2012
// Design Name: VME Adapter
// Module Name: M_START_LED_GEN
// Project Name: EA4178
// Target Devices: FPGA
// Tool versions: Xilinx 14.1
// Description: 50 MHz to 1 Hz Divider
// With Diagnostic LED Start Blink
//////////////////////////////////////
module M_START_LED_GEN(
   input CLK,
   input RST,
   output reg LED_OE,
   output CEO_1HZ,
   output reg LED_OK
   );

// Internal signals declaration:
 reg [9:0] CNTR_10_A, CNTR_10_B;
 reg [5:0] CNTR_6_C;
 reg CE_1HZ;
 reg [3:0] CNTR_FSM;
//---------------------------------
// Primary 1:1000 Divider:
 always @ (posedge CLK, posedge RST)
   if(RST)
      CNTR_10_A <= 10'h000;
   else if(CNTR_10_A == 10'd999)
      CNTR_10_A <= 10'h000;
   else
      CNTR_10_A <= CNTR_10_A + 1;
//---------------------------------
// Secondary 1:1000 Divider:
 always @ (posedge CLK, posedge RST)
   if(RST)
      CNTR_10_B <= 10'h000;
   else if(CNTR_10_A == 10'd999)
      if(CNTR_10_B == 10'd999)
         CNTR_10_B <= 10'h000;
      else
         CNTR_10_B <= CNTR_10_B + 1;
//---------------------------------
// Third 1:50 Divider:
 always @ (posedge CLK, posedge RST)
   if(RST)
      begin
         CNTR_6_C <= 6'h00;
         CE_1HZ <= 0;
      end
   else if(CNTR_10_A == 10'd999 && CNTR_10_B == 10'd999)
      if(CNTR_6_C == 6'd49)
         begin
            CNTR_6_C <= 6'h00;
            CE_1HZ <= 1;
         end
      else
         CNTR_6_C <= CNTR_6_C + 1;
   else CE_1HZ <= 0;
 assign CEO_1HZ = CE_1HZ;
//---------------------------------
 always @ (posedge CLK, posedge RST)
   if(RST)
      begin
         CNTR_FSM <= 4'h0;
         LED_OE <= 1;
         LED_OK <= 0;
      end
   else if(CE_1HZ)
      begin
         case(CNTR_FSM)

            4'd0 :
            begin
               LED_OE <= 0;
               CNTR_FSM <= 4'd1;
            end

            4'd1 :
            begin
               LED_OE <= 1;
               CNTR_FSM <= 4'd2;
            end

            4'd2 :
            begin
               LED_OE <= 0;
               CNTR_FSM <= 4'd3;
            end

            4'd3 :
            begin
               LED_OE <= 1;
               CNTR_FSM <= 4'd4;
            end

            4'd4 :
            begin
               LED_OE <= 0;
               CNTR_FSM <= 4'd5;
            end

            4'd5 :
            begin
               LED_OE <= 1;
               CNTR_FSM <= 4'd6;
            end

            4'd6 :
            begin
               LED_OE <= 0;
               CNTR_FSM <= 4'd7;
            end

            4'd7 :
            begin
               LED_OE <= 1;
               CNTR_FSM <= 4'd8;
            end

            4'd8 :
            begin
               LED_OE <= 0;
               CNTR_FSM <= 4'd9;
            end

            default:
            begin
               LED_OK <= 1;
               LED_OE <= 0;
               CNTR_FSM <= 4'd9;
            end

         endcase
//---------------------------
endmodule
```

Для подключения модуля управления светодиодом к схемотехническому модулю UMMIO_Clock.sch, который показан на рис. 2, был создан файл описания символа *УГО M_START_LED_GEN.sym*.

Ниже приведено текстовое описание символа:

```
<?xml version="1.0" encoding="UTF-8"?>
<symbol version="7" name="M_START_LED_GEN">
   <symboltype>BLOCK</symboltype>
   <timestamp>2012-10-16T10:2:16</timestamp>
   <attr value="M_START_LED_GEN" name="VeriModel" />
   <pin polarity="Input" x="-192" y="-32" name="CLK" />
   <pin polarity="Output" x="192" y="-32" name="LED_OE" />
   <pin polarity="Input" x="-192" y="160" name="RST" />
    <pin polarity="Output" x="192" y="32" name="LED_OK" />
   <pin polarity="Output" x="192" y="160" name="CEO_1HZ" />
   <graph>
      <attrtext style="alignment:BCENTER;fontsize:56;fontname:Arial" attrname="SymbolName" x="0" y="-72" type="symbol" />
      <attrtext style="alignment:RIGHT;fontsize:24;fontname:Arial" attrname="PinName" x="120" y="-32" type="pin LED_OE" />
      <line x2="-192" y1="-32" y2="-32" x1="-128" />
      <line x2="192" y1="-32" y2="-32" x1="128" />
      <rect width="256" x="-128" y="-64" height="256" />
      <line x2="-112" y1="-48" y2="-32" x1="-128" />
      <line x2="-112" y1="-16" y2="-32" x1="-128" />
      <attrtext style="fontsize:24;fontname:Arial" attrname="PinName" x="-104" y="-32" type="pin CLK" />
      <attrtext style="fontsize:24;fontname:Arial" attrname="PinName" x="-120" y="160" type="pin RST" />
      <line x2="-192" y1="160" y2="160" x1="-128" />
      <attrtext style="alignment:RIGHT;fontsize:24;fontname:Arial" attrname="PinName" x="120" y="32" type="pin LED_OK" />
      <line x2="192" y1="32" y2="32" x1="128" />
      <attrtext style="alignment:RIGHT;fontsize:24;fontname:Arial" attrname="PinName" x="120" y="160" type="pin CEO_1HZ" />
      <line x2="192" y1="160" y2="160" x1="128" />
      <line x2="0" y1="-64" y2="192" x1="0" />
      <line x2="-48" y1="-64" y2="192" x1="-48" />
      <line x2="-48" y1="0" y2="0" x1="-128" />
      <line x2="-48" y1="128" y2="128" x1="-128" />
      <line x2="128" y1="128" y2="128" x1="0" />
   </graph>
</symbol>
```

Соответствующее описанию условное графическое изображение выделено фиолетовым цветом вверху на рис. 2а.

Все схемотехнические модули, созданные в САПР серии Xilinx ISE, транслируются в аналогичные им низкоуровневые модели на выбранном в свойствах проекта языке описания аппаратуры (Preferred Language: Verilog или VHDL). Созданная САПР на основе схемотехнического описания синтезируемой модели сохраняется в одноименном файле с расширением **.vf* — для языка Verilog и **.vhf* — для языка VHDL. Для схемотехнического модуля, листы которого изображены на рис. 2, в САПР Xilinx Design Suite был сгенерирован файл *UMMIO_Clock.vf*, содержащий низкоуровневое описание схемы на языке Verilog:

```
//////////////////////////////////////
// Copyright (c) 1995-2012 Xilinx, Inc. All rights reserved.
//////////////////////////////////////
//   ___  ____
//  /   /\/   /
// /___/  \  / Vendor: Xilinx
// \   \   \/  Version:14.4
//  \   \       Application:sch2hdl
//  /   /       Filename:UMMIO_Clock.vf
// /___/   /\   Timestamp:03/26/2013 13:10:38
// \   \  /  \
//  \___\/\___\
//
//Command: sch2hdl -intstyle ise -family spartan3e –verilog
// E:/Logic_CAD/MCH_PRJ/Xilinx/MCH_UMMIO_S3.1/UMMIO_Clock.vf
// -w E:/Logic_CAD/MCH_PRJ/Xilinx/MCH_UMMIO_S3.1/UMMIO_Clock.sch
//Design Name: UMMIO_Clock
//Device: spartan3e
//Purpose:
// This verilog netlist is translated from an ECS schematic.
// It can be synthesized and simulated,
// but it should not be modified.
`timescale 1ns / 1ps

module UMMIO_Clock(CONF_DONE, ASYNCH_RST, ASYNCH_RST_INPUT, CLK_16M, CLK_16M7, CLK_32M, CLK_33M, CLK_48M, CLK_50M, CLK_64M, CLK_66M, CLK_75M, CLK_80M, CLK_83M, CLK_100M, CLK_160M, L_8, RST_50M, AB_NRST_BUF, SPRT3_CLK_IN, VMX_RST);

    input CONF_DONE;
   output ASYNCH_RST;
   output ASYNCH_RST_INPUT;
   output CLK_16M;
   output CLK_16M7;
   output CLK_32M;
   output CLK_33M;
   output CLK_48M;
   output CLK_50M;
   output CLK_64M;
   output CLK_66M;
   output CLK_75M;
   output CLK_80M;
   output CLK_83M;
   output CLK_100M;
```

```
output CLK_160M;
output L_8;
output RST_50M;
   inout AB_NRST_BUF;
   inout SPRT3_CLK_IN;
   inout VMX_RST;

wire BUFG_VMX;
wire BUFG_16M;
wire BUFG_16M7;
wire BUFG_32M;
wire BUFG_33M;
wire BUFG_48M;
wire BUFG_50M;
wire BUFG_64M;
wire BUFG_66M;
wire BUFG_80M;
wire BUFG_83M;
wire BUFG_100M;
wire BUFG_160M;
wire CLKI_32M;
wire CLKI_33M;
wire CLKI_50M;
wire CLKI_80M;
wire DCM_1_RST;
wire [7:0] DCM_1_ST;
wire DCM_2_RST;
wire [7:0] DCM_2_ST;
wire DCM_3_RST;
wire [7:0] DCM_3_ST;
wire DCM_4_RST;
wire [7:0] DCM_4_ST;
wire D1R;
wire D2R;
wire D3R;
wire D4R;
wire XLXN_8;
wire XLXN_15;
wire XLXN_23;
wire XLXN_125;
wire XLXN_148;
wire XLXN_166;
wire XLXN_217;
wire XLXN_218;
wire XLXN_230;
wire XLXN_256;
wire XLXN_257;
wire XLXN_260;
wire XLXN_276;
wire XLXN_277;
wire XLXN_280;
wire XLXN_314;
wire XLXN_318;
wire XLXN_361;
wire XLXN_362;
wire XLXN_364;
wire ASYNCH_RST_INPUT_DUMMY;
wire CLK_66M_DUMMY;
wire CLK_48M_DUMMY;
wire RST_50M_DUMMY;
wire CLK_80M_DUMMY;
wire CLK_50M_DUMMY;
wire ASYNCH_RST_DUMMY;
wire CLK_64M_DUMMY;
wire CLK_83M_DUMMY;

   assign ASYNCH_RST = ASYNCH_RST_DUMMY;
   assign ASYNCH_RST_INPUT = ASYNCH_RST_INPUT_DUMMY;
   assign CLK_48M = CLK_48M_DUMMY;
   assign CLK_50M = CLK_50M_DUMMY;
   assign CLK_64M = CLK_64M_DUMMY;
   assign CLK_66M = CLK_66M_DUMMY;
   assign CLK_80M = CLK_80M_DUMMY;
   assign CLK_83M = CLK_83M_DUMMY;
   assign RST_50M = RST_50M_DUMMY;
   DCM_SP #( .CLKFX_MULTIPLY(8), .CLKFX_DIVIDE(5),
.CLKIN_PERIOD(20.0),
      .CLK_FEEDBACK("1X"), .CLKDV_DIVIDE(1.5), .CLKIN_
DIVIDE_BY_2("FALSE"),
      .CLKOUT_PHASE_SHIFT("NONE"), .DESKEW_
ADJUST("SYSTEM_SYNCHRONOUS"),
      .DFS_FREQUENCY_MODE("LOW"), .DLL_FREQUENCY_
MODE("LOW"),
      .DSS_MODE("NONE"), .DUTY_CYCLE_
CORRECTION("TRUE"), .FACTORY_JF(16'hC080))
DCM_1 (.CLKFB(CLK_50M_DUMMY), .CLKIN(CLKI_50M),
.DSSEN(XLXN_125), .PSCLK(XLXN_125), .PSEN(XLXN_125),
.PSINCDEC(XLXN_125), .RST(D1R), .CLKDV(CLKI_33M),
.CLKFX(CLKI_80M), .CLKFX180(), .CLK0(BUFG_50M),
.CLK2X(BUFG_100M), .CLK2X180(), .CLK90(), .CLK180(), .CLK270(),
.LOCKED(XLXN_148), .PSDONE(), .STATUS(DCM_1_ST[7:0]));
   DCM_SP #( .CLK_FEEDBACK("1X"), .CLKIN_PERIOD(12.0),
.CLKDV_DIVIDE(2.5),
      .CLKFX_DIVIDE(16), .CLKFX_MULTIPLY(15), .CLKIN_
DIVIDE_BY_2("FALSE"),
      .CLKOUT_PHASE_SHIFT("NONE"), .DESKEW_
ADJUST("SYSTEM_SYNCHRONOUS"),
      .DFS_FREQUENCY_MODE("LOW"), .DLL_FREQUENCY_
MODE("LOW"),
      .DSS_MODE("NONE"), .DUTY_CYCLE_
CORRECTION("TRUE"), .PHASE_SHIFT(0),
      .STARTUP_WAIT("FALSE"), .FACTORY_JF(16'hC080))
DCM_2 (.CLKFB(CLK_80M_DUMMY), .CLKIN(CLKI_80M),
.DSSEN(XLXN_217), .PSCLK(XLXN_217), .PSEN(XLXN_217),
.PSINCDEC(XLXN_217), .RST(D2R), .CLKDV(CLKI_32M),
.CLKFX(BUFG_VMX), .CLKFX180(), .CLK0(BUFG_80M),
.CLK2X(BUFG_160M), .CLK2X180(), .CLK90(), .CLK180(), .CLK270(),
.LOCKED(XLXN_218), .PSDONE(), .STATUS(DCM_2_ST[7:0]));
   DCM_SP #( .CLK_FEEDBACK("2X"), .CLKIN_PERIOD(30.0),
.CLKDV_DIVIDE(2.0),
      .CLKFX_DIVIDE(2), .CLKFX_MULTIPLY(3), .CLKIN_
DIVIDE_BY_2("FALSE"),
      .CLKOUT_PHASE_SHIFT("NONE"), .DESKEW_
ADJUST("SYSTEM_SYNCHRONOUS"),
      .DFS_FREQUENCY_MODE("LOW"), .DLL_FREQUENCY_
MODE("LOW"),
      .DSS_MODE("NONE"), .DUTY_CYCLE_
CORRECTION("TRUE"), .PHASE_SHIFT(0),
      .STARTUP_WAIT("FALSE"), .FACTORY_JF(16'hC080))
DCM_3 (.CLKFB(CLK_64M_DUMMY), .CLKIN(CLKI_32M),
.DSSEN(XLXN_256), .PSCLK(XLXN_256), .PSEN(XLXN_256),
.PSINCDEC(XLXN_256), .RST(D3R), .CLKDV(BUFG_16M),
.CLKFX(BUFG_48M), .CLKFX180(), .CLK0(BUFG_32M),
.CLK2X(BUFG_64M), .CLK2X180(), .CLK90(), .CLK180(), .CLK270(),
.LOCKED(XLXN_257), .PSDONE(), .STATUS(DCM_3_ST[7:0]));
   DCM_SP #( .CLK_FEEDBACK("2X"), .CLKIN_PERIOD(30.0),
.CLKDV_DIVIDE(2.0),
      .CLKFX_DIVIDE(2), .CLKFX_MULTIPLY(5), .CLKIN_
DIVIDE_BY_2("FALSE"),
      .CLKOUT_PHASE_SHIFT("NONE"), .DESKEW_
ADJUST("SYSTEM_SYNCHRONOUS"),
      .DFS_FREQUENCY_MODE("LOW"), .DLL_FREQUENCY_
MODE("LOW"),
      .DSS_MODE("NONE"), .DUTY_CYCLE_
CORRECTION("TRUE"), .PHASE_SHIFT(0),
      .STARTUP_WAIT("FALSE"), .FACTORY_JF(16'hC080))
DCM_4 (.CLKFB(CLK_66M_DUMMY), .CLKIN(CLKI_33M),
.DSSEN(XLXN_276), .PSCLK(XLXN_276), .PSEN(XLXN_276),
.PSINCDEC(XLXN_276), .RST(D4R), .CLKDV(BUFG_16M7),
.CLKFX(BUFG_83M), .CLKFX180(), .CLK0(BUFG_33M),
.CLK2X(BUFG_66M), .CLK2X180(), .CLK90(), .CLK180(), .CLK270(),
.LOCKED(XLXN_277), .PSDONE(), .STATUS(DCM_4_ST[7:0]));
   M_START_LED_GEN LED_MODULE (.CLK(CLK_50M_
DUMMY), .RST(RST_50M_DUMMY), .CEO_1HZ(), .LED_
OE(XLXN_8), .LED_OK());
   (* IOSTANDARD = "DEFAULT" *) (* IBUF_DELAY_VALUE =
"0" *)
   IBUFG XLXI_2 (.I(SPRT3_CLK_IN), .O(CLKI_50M));
   (* DRIVE = "6" *) (* IOSTANDARD = "DEFAULT" *) (* SLEW
= "SLOW" *)
   OBUF XLXI_4 (.I(XLXN_8), .O(L_8));
   (* IOSTANDARD = "DEFAULT" *) (* IBUF_DELAY_VALUE =
"0" *) (* IFD_DELAY_VALUE = "AUTO" *)
   IBUF XLXI_8 (.I(AB_NRST_BUF), .O(XLXN_23));
   FDP #( .INIT(1'b1) ) XLXI_10 (.C(CLK_50M_DUMMY),
.D(XLXN_361), .PRE(ASYNCH_RST_DUMMY), .Q(XLXN_15));
   FDP #( .INIT(1'b1) ) XLXI_11 (.C(CLK_50M_DUMMY),
.D(XLXN_15), .PRE(ASYNCH_RST_DUMMY), .Q(RST_50M_
DUMMY));
   PULLUP XLXI_12 (.O(AB_NRST_BUF));
   GND XLXI_13 (.G(XLXN_361));
   (* IOSTANDARD = "DEFAULT" *) (* IBUF_DELAY_VALUE =
"0" *) (* IFD_DELAY_VALUE = "AUTO" *)
   IBUF XLXI_14 (.I(VMX_RST), .O(XLXN_314));
   OR3B3 XLXI_15 (.I0(XLXN_314), .I1(XLXN_318),
.I2(XLXN_23), .O(ASYNCH_RST_INPUT_DUMMY));
   PULLUP XLXI_16 (.O(VMX_RST));
   GND XLXI_28 (.G(XLXN_125));
   INV XLXI_73 (.I(XLXN_148), .O(DCM_1_RST));
   OR2 XLXI_74 (.I0(ASYNCH_RST_INPUT_DUMMY),
.I1(XLXN_166), .O(D1R));
   AND2 XLXI_75 (.I0(DCM_1_ST[2]), .I1(DCM_1_RST),
.O(XLXN_166));
   BUFG XLXI_95 (.I(BUFG_100M), .O(CLK_100M));
   BUFG XLXI_99 (.I(BUFG_50M), .O(CLK_50M_DUMMY));
   GND XLXI_104 (.G(XLXN_217));
   BUFG XLXI_105 (.I(BUFG_VMX), .O(CLK_75M));
   BUFG XLXI_106 (.I(BUFG_160M), .O(CLK_160M));
   INV XLXI_108 (.I(XLXN_218), .O(DCM_2_RST));
   AND2 XLXI_112 (.I0(DCM_2_ST[2]), .I1(DCM_2_RST),
.O(XLXN_230));
   OR2 XLXI_113 (.I0(DCM_1_RST), .I1(XLXN_230), .O(D2R));
   BUFG XLXI_114 (.I(BUFG_80M), .O(CLK_80M_DUMMY));
   GND XLXI_116 (.G(XLXN_256));
   BUFG XLXI_117 (.I(BUFG_48M), .O(CLK_48M_DUMMY));
   BUFG XLXI_118 (.I(BUFG_64M), .O(CLK_64M_DUMMY));
   BUFG XLXI_119 (.I(BUFG_16M), .O(CLK_16M));
   INV XLXI_120 (.I(XLXN_257), .O(DCM_3_RST));
   AND2 XLXI_121 (.I0(DCM_3_ST[2]), .I1(DCM_3_RST),
.O(XLXN_260));
   OR2 XLXI_122 (.I0(DCM_2_RST), .I1(XLXN_260), .O(D3R));
   BUFG XLXI_123 (.I(BUFG_32M), .O(CLK_32M));
   GND XLXI_125 (.G(XLXN_276));
   BUFG XLXI_126 (.I(BUFG_83M), .O(CLK_83M_DUMMY));
   BUFG XLXI_127 (.I(BUFG_66M), .O(CLK_66M_DUMMY));
   BUFG XLXI_128 (.I(BUFG_16M7), .O(CLK_16M7));
   INV XLXI_129 (.I(XLXN_277), .O(DCM_4_RST));
   AND2 XLXI_130 (.I0(DCM_4_ST[2]), .I1(DCM_4_RST),
.O(XLXN_280));
   OR2 XLXI_131 (.I0(DCM_1_RST), .I1(XLXN_280), .O(D4R));
   BUFG XLXI_132 (.I(BUFG_33M), .O(CLK_33M));
   OR2 XLXI_142 (.I0(XLXN_364), .I1(XLXN_362), .O(ASYNCH_
RST_DUMMY));
   PULLDOWN XLXI_151 (.O(SPRT3_CLK_IN));
   (* IOSTANDARD = "DEFAULT" *) (* IBUF_DELAY_VALUE =
"0" *) (* IFD_DELAY_VALUE = "AUTO" *)
   IBUF XLXI_155 (.I(CONF_DONE), .O(XLXN_318));
   FDP #( .INIT(1'b1) ) XLXI_156 (.C(CLK_83M_DUMMY),
.D(XLXN_361), .PRE(DCM_4_RST), .Q(XLXN_362));
   FDP #( .INIT(1'b1) ) XLXI_157 (.C(CLK_48M_DUMMY),
.D(XLXN_361), .PRE(DCM_3_RST), .Q(XLXN_364));
endmodule
```

Судя по этому коду, все схемотехнические элементы представлены на языке Verilog примитивами из библиотеки фирмы Xilinx. При этом сохранены все названия цепей, использованные в схемотехническом файле.

В настоящее время схемотехническое описание сложных цифровых устройств в объеме кристалла практически не используется. В статье рассмотрен схемотехнический модуль для большей наглядности. При построении моделей синтезаторов частот в новых проектах рекомендуется использовать описание на языках VHDL и Verilog. Файл *UMMIO_Clock.vf* можно взять за основу для построения иных конфигураций синтезаторов тактовых частот. В частности, автору без каких-либо существенных проблем удалось использовать этот низкоуровневый код в проектах ПЛИС серии Spartan-6. (Кроме изменения названия модуля, следует изменить название файла, исправив расширение на **.v*, либо просто скопировать Verilog-код в новый файл.) Подробное описание примитивов DCM серии Spartan-6 и их варианты подключения в объеме проекта ПЛИС описаны в статье [5].

В седьмом поколении ПЛИС FPGA фирмы Xilinx синтезаторы частот строятся на других технологических примитивах. Вариант построения синтезатора частоты для проекта ПЛИС XC7A100T серии Artix-7 был рассмотрен в статье [6].

Выводы

Примитивы Digital Clock Manager, реализованные в ПЛИС серий Spartan-3 — Spartan-6 фирмы Xilinx, позволяют синтезировать тактовые частоты в широких пределах, получая на выходе сигнал, частота которого не кратна входной опорной частоте. Например, можно получить частоты 48, 64 и 120 МГц из опорной частоты 66,667 МГц, использовав каскадное соединение нескольких DCM. При подборе атрибутов конфигурации определенного примитива DCM следует руководствоваться ограничениями, указанными в документации на серию ПЛИС.

Список связей схемотехнического описания синтезатора тактовых частот, сгенерированный средствами САПР в виде низкоуровневой синтезируемой модели на языке Verilog, можно использовать в качестве

основы при построении синтезаторов частот для других проектов.

Литература

1. Spartan-3 FPGA Family Data Sheet. DS099. Xilinx, Inc. 2003–2013.
2. Spartan-3E FPGA Family Data Sheet. DS312. Xilinx, Inc. 2005–2013.
3. Spartan-6 FPGA Clocking Resources User Guide. UG382 (v1.9) Xilinx, Inc. Dec. 20, 2013.
4. Spartan-6 FPGA Data Sheet: DC and Switching Characteristics. DS162 (v3.0) Xilinx, Inc. Oct. 17, 2011.
5. Зотов В. Разработка узлов синхронизации цифровых устройств и встраиваемых микропроцессорных систем, реализуемых на базе ПЛИС фирмы Xilinx серии Spartan-6 // Компоненты и технологии. 2013. № 4.
6. Борисенко Н. Модель узла управления динамическим 7-сегментным индикатором с подавлением дребезга контактов кнопок в объеме ПЛИС Xilinx Artix-7 для отладочной платы Digilent Nexys 4 // Компоненты и технологии. 2014. № 1.
7. Agarwal S., Kumar A. Phase-locked loops in an IC-based clock distribution system., Cypress Semiconductor Bangalore. Aug. 4, 2013.