Synopsys Design Constraint — язык задания временных ограничений на примере Altera TimeQuest. Часть 4

Денис ШЕХАЛЕВ
shdv@micran.ru

В предыдущих частях статьи мы рассмотрели все аспекты задания временных ограничений для проектов: задание частот, их соотношений, портов ввода/вывода. Но информация о временных ограничениях будет не полной без рассмотрения так называемых исключений временных ограничений.

Зачем нужны исключения задания временных ограничений

Задание исключений временных ограничений представляет собой изменение правил временного анализа. Эти исключения могут быть применены как ко всему проекту, так и к какой-либо конкретной цепи. Исключения применяются в том случае, когда нужно задать специфические условия функционирования логической схемы, или в случае, когда TimeQuest, пользуясь информацией, извлекаемой из списка соединений (netlist) и файла временных ограничений, проводит временной анализ, не совпадающий с логикой функционирования логической схемы. Рассмотрим использование различных исключений временных ограничений подробнее. Но сразу оговоримся, что в целях экономии места ***sdc***-файлы для проектов будут содержать только те команды, которые нужны для демонстрации того или иного использования исключений.

Использование ложных путей для устранения временных нарушений

С этим видом исключений временных ограничений мы уже знакомы: на примерах, приведенных в предыдущих частях статьи, при задании ложных путей (***false_path***) или эксклюзивных групп тактовых частот (***set_clock_groups-exclusive***). Этими командами мы исключали те или иные пути из анализа и оптимизации. Это позволяло проводить временной анализ проекта, соответствующий его функциональности, и облегчить условия работы Quartus. Рассмотрим еще одно полезное применение ложных путей.

Иногда возникает ситуация, когда использовать PLL для выполнения временных ограничений выходных интерфейсов нельзя. А задержки на LUT не дают результата. В этом случае может быть использована техника вывода данных и генерации выходного сигнала тактовой частоты на одинаковых DDR-ячейках ввода/вывода. Для этого сигнал тактовой частоты формируется как сигнал данных со значениями 0 и 1. Это позволяет максимально выровнять задержки сигналов данных и тактовой частоты. А при использовании техники инверсии тактовой частоты этот метод дает возможность выполнить временные ограничения по ***tsu/th*** выходного интерфейса. Рассмотрим вывод «пилы» на ЦАП с помощью DDR-ячеек:

```
module ddio (input iclk, output oclk, output logic [7 : 0] data);

   logic [7 : 0] cnt;
   always_ff @(posedge iclk) begin
      cnt <= cnt + 1'b1;
   end

   generate
      genvar i;
      for (i = 0; i < 8; i++) begin : gen
         altddio data_ddio (
            .datain_h ( cnt[i] ),
            .datain_l ( cnt[i] ),
            .outclock ( iclk ),
            .dataout ( data[i] ));
      end
   endgenerate

   altddio clk_ddio (
      .datain_h ( 1'b0 ),
      .datain_l ( 1'b1 ),
      .outclock ( iclk ),
      .dataout ( oclk ));

endmodule
```

altddio — это выходной однобитный буфер вывода, полученный с помощью MegaWizard. Предположим, что ***tsu/th*** ЦАП равны 2 нс/2 нс соответственно. Начальный ***sdc***-файл для этого проекта будет такой:

```
set_time_format -unit ns -decimal_places 3
derive_clock_uncertainty

create_clock -period 180MHz -name {iclk} [get_ports {iclk}]
create_generated_clock -name {oclk} -source [get_ports {iclk}] -invert
[get_ports {oclk}]

set_clock_groups -exclusive -group {iclk oclk}

set_output_delay -clock [get_clocks {oclk}] -max 2.0 [get_ports
{data[*]}]
set_output_delay -clock [get_clocks {oclk}] -min -2.0 [get_ports
{data[*]}]
```

Собираем проект, запускаем временной анализ и видим неожиданную картину (рис. 43). Невероятно, но факт: вместо одной пары параметров ***tsu/th*** TimeQuest находит две пары. Подробный анализ выполнения

Inputs to Outputs (Setup)

	Slack	From Node	To Node	Launch Clock	Latch Clock
1	-0.697	iclk	data[2]	iclk	oclk
2	-0.604	iclk	data[7]	iclk	oclk
3	-0.580	iclk	data[1]	iclk	oclk
4	-0.580	iclk	data[6]	iclk	oclk
5	-0.578	iclk	data[0]	iclk	oclk
6	-0.578	iclk	data[4]	iclk	oclk
7	-0.578	iclk	data[5]	iclk	oclk
8	-0.571	iclk	data[3]	iclk	oclk
9	2.035	iclk	data[2]	iclk	oclk
10	2.127	iclk	data[7]	iclk	oclk
11	2.155	iclk	data[1]	iclk	oclk
12	2.155	iclk	data[6]	iclk	oclk
13	2.157	iclk	data[0]	iclk	oclk
14	2.157	iclk	data[4]	iclk	oclk
15	2.157	iclk	data[5]	iclk	oclk
16	2.165	iclk	data[3]	iclk	oclk

Inputs to Outputs (Hold)

	Slack	From Node	To Node	Launch Clock	Latch Clock
1	-1.328	iclk	data[3]	iclk	oclk
2	-1.320	iclk	data[0]	iclk	oclk
3	-1.320	iclk	data[4]	iclk	oclk
4	-1.320	iclk	data[5]	iclk	oclk
5	-1.318	iclk	data[1]	iclk	oclk
6	-1.318	iclk	data[6]	iclk	oclk
7	-1.280	iclk	data[7]	iclk	oclk
8	-1.175	iclk	data[2]	iclk	oclk
9	1.417	iclk	data[3]	iclk	oclk
10	1.423	iclk	data[0]	iclk	oclk
11	1.423	iclk	data[4]	iclk	oclk
12	1.423	iclk	data[5]	iclk	oclk
13	1.425	iclk	data[1]	iclk	oclk
14	1.425	iclk	data[6]	iclk	oclk
15	1.459	iclk	data[7]	iclk	oclk
16	1.565	iclk	data[2]	iclk	oclk

Рис. 43. Результат временного анализа проекта ddio с нарушениями временных ограничений

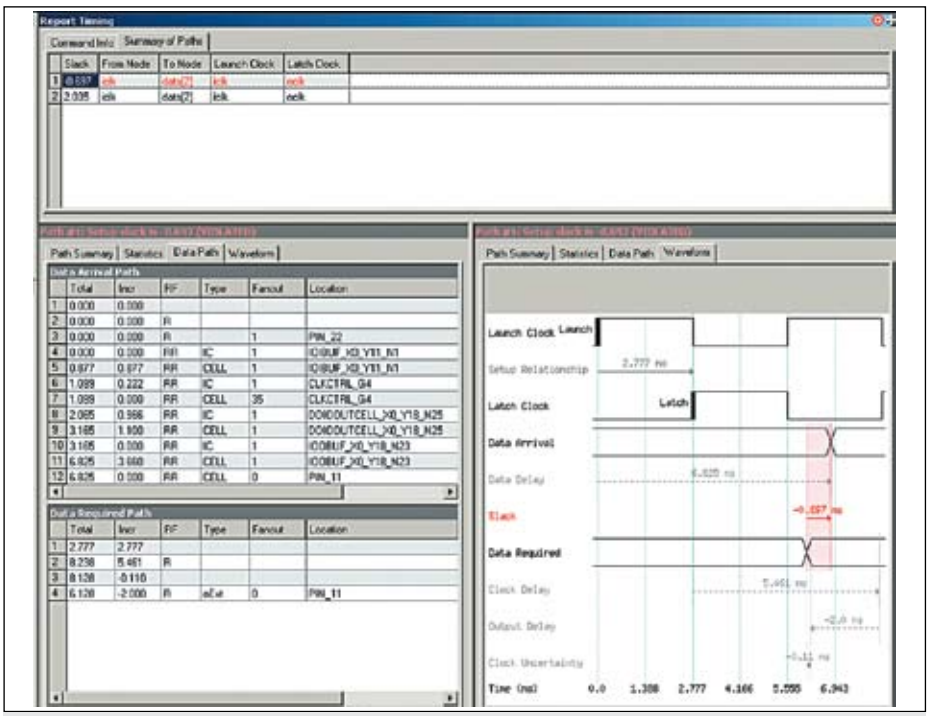

Рис. 44. Подробный результат временного анализа порта data [2] проекта ddio

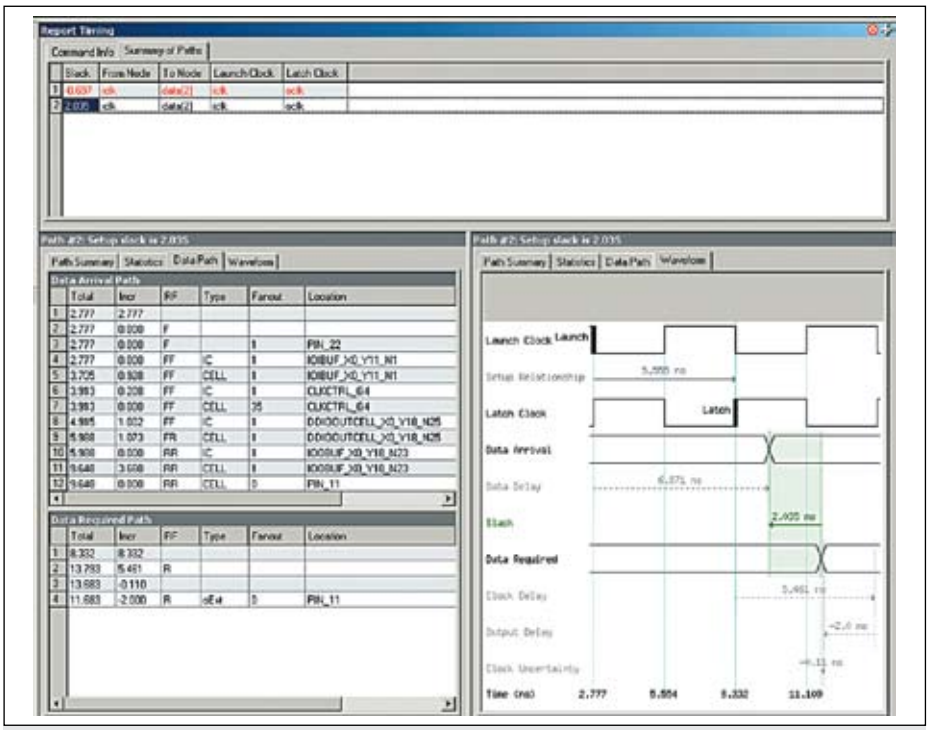

Рис. 45. Подробный результат временного анализа порта data [2] проекта ddio

Inputs to Outputs (Setup)

	Slack	From Node	To Node	Launch Clock	Latch Clock
1	0.529	iclk	data[2]	iclk	oclk
2	0.571	iclk	data[1]	iclk	oclk
3	0.571	iclk	data[6]	iclk	oclk
4	0.573	iclk	data[0]	iclk	oclk
5	0.573	iclk	data[4]	iclk	oclk
6	0.573	iclk	data[5]	iclk	oclk
7	0.580	iclk	data[3]	iclk	oclk
8	0.622	iclk	data[7]	iclk	oclk

Inputs to Outputs (Hold)

	Slack	From Node	To Node	Launch Clock	Latch Clock
1	0.383	iclk	data[7]	iclk	oclk
2	0.412	iclk	data[3]	iclk	oclk
3	0.418	iclk	data[0]	iclk	oclk
4	0.418	iclk	data[4]	iclk	oclk
5	0.418	iclk	data[5]	iclk	oclk
6	0.420	iclk	data[1]	iclk	oclk
7	0.420	iclk	data[6]	iclk	oclk
8	0.489	iclk	data[2]	iclk	oclk

Рис. 48. Результат временного анализа проекта ddio без нарушений временных ограничений

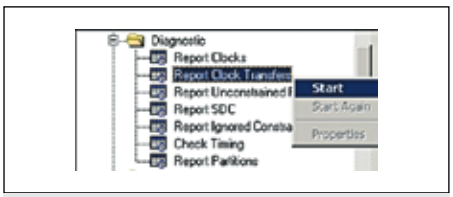

Рис. 46. Диалоговое окно выполнения команды диагностики Report Clock Transfers

Setup Transfers

	From Clock	To Clock	RR Paths	FR Paths	RF Paths	FF Paths
1	iclk	iclk	52	0	0	0
2	iclk	oclk	8	8	0	0

Рис. 47. Отчет о выполнении команды диагностики Report Clock Transfers

временных ограничений (рис. 44, 45) еще больше вводит в тупик. Для того чтобы разобраться, в чем же причина такого поведения TimeQuest, нужно воспользоваться средствами по диагностике проводимого временного анализа. Выполним команду ***Report Clock Transfers*** (рис. 46). В результате видно (рис. 47), что TimeQuest обнаружил передачу данных с восходящего фронта ***iclk*** до восходящего фронта ***oclk*** (RR) и от нисходящего фронта ***iclk*** до восходящего фронта ***oclk*** (FR). Все это TimeQuest сделал самостоятельно, на основе информации, извлеченной из нетлиста, а так как он ничего не знает о нашем проекте, то проводит анализ для каждой пары фронтов тактовых сигналов.

Нужно каким-то способом указать TimeQuest, что одна пара фронтов лишняя. Это можно сделать, назначив ложный путь на цепи, источниками которых являются регистры, тактируемые нисходящим фронтом частоты ***iclk***, а приемниками — регистры, тактируемые восходящим фронтом частоты ***oclk***. Добавляем в наш ***sdc***-файл следующие строки:

```
set_false_path -setup -fall_from [get_clocks iclk] -rise_to [get_clocks oclk]
set_false_path -hold -fall_from [get_clocks iclk] -rise_to [get_clocks oclk]
```

Собираем проект, запускаем анализ, смотрим (рис. 48). Видим, что для анализа используется только одна пара фронтов тактовых частот и все временные ограничения выполнены.

Использование мультицикловых путей для устранения временных нарушений

Мультицикловые пути (multicycle paths) представляют собой еще один вид исключений временных ограничений. В отличие от ложных путей, которые просто удаляют цепи из временного анализа и оптимизации, мультицикловые пути позволяют более гибко варьировать временными параметрами цепей. Рассмотрим, в чем заключается эта гибкость. На рис. 49–51 изображены временные диаграммы одноциклового и различных

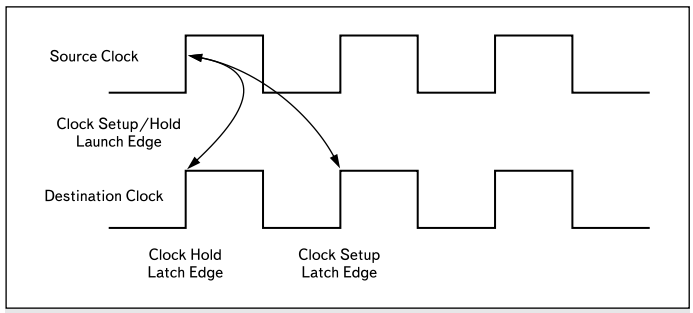

Рис. 49. Фронты тактового сигнала, используемые для временного анализа одноцикловых синхронных путей

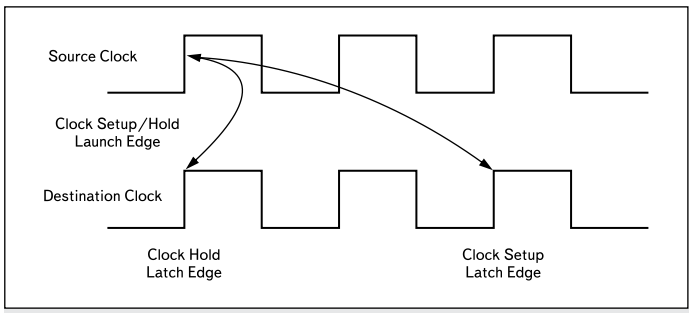

Рис. 50. Фронты тактового сигнала, используемые для временного анализа двухцикловых синхронных путей

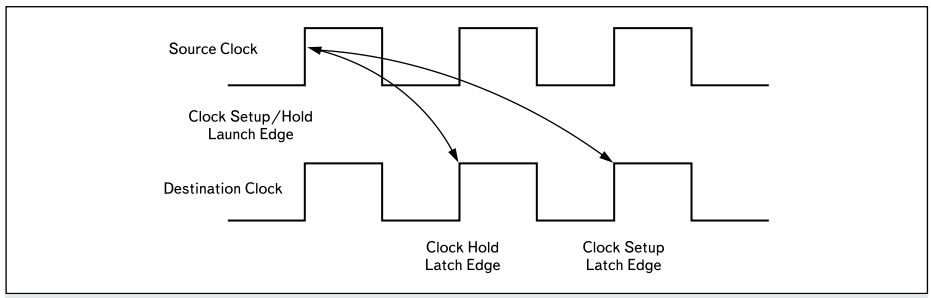

Рис. 51. Фронты тактового сигнала, используемые для временного анализа одноцикловых синхронных путей с задержкой

вариантов мультициклового анализа. Судя по диаграммам, при мультицикловых путях фронты тактового сигнала источника и приемника, используемые для анализа, могут быть в различных временных соотношениях между собой. Например, на рис. 50 изображен случай, когда приемник читает данные источника с частотой в два раза меньше тактовой частоты. А на рис. 51 показана ситуация, когда приемник читает данные источника со сдвигом (задержкой) в 1 такт.

Для задания мультицикловых путей в ***sdc***-файле используются следующие команды:

```
set_multicycle_path -from [<names>] -to [<names>] -setup <-end/begin> <setup_edge>
set_multicycle_path -from [<names>] -to [<names>] -hold <-end/begin> <hold_edge_offset>
```

Поясним значения ключей, используемых в командах:

- ***from/to*** — это имена источника и приемника цепи, на которую накладывается мультицикл. Как вы понимаете, это могут быть только регистры (в комбинационных цепях никаких мультициклов быть не может). Вместо имен цепей могут использоваться логические имена тактовых частот источника/приемника.
- ***setup/hold*** — описывает, к какому именно временному ограничению относится указываемая в команде величина.
- ***start/end*** — описывает, к какому именно сигналу тактовой частоты (источника или приемника соответственно) относится задаваемая величина.
- ***setup_edge*** — номер ***Latch Setup Clock Edge***, относительно которого будет проводиться анализ по ***tsu*** (рис. 50).
- ***hold_edge_offset*** — смещение фронта ***Latch Hold Clock Edge***, относительно которого будет проводиться анализ по ***th*** (рис. 50).

Как видите, если значение по ***tsu*** представляет собой новое значение фронта тактовой частоты, то значение по ***th*** — это смещение относительно фронта тактовой частоты, заданного для ***tsu***. Дело в том, что TimeQuest определяет ***Latch Hold Clock Edge*** по формуле:

```
hold_edge = setup_edge - 1 - hold_edge_offset
```

По умолчанию ***hold_edge_offset*** равен нулю. Об этом нужно помнить при задании мультицикловых путей. Для одноцикловых путей значения фронтов тактовой частоты по ***tsu/th*** равны 1/0 соответственно. Для двухцикловых путей — 2/1 и т. д. Возникает логичный вопрос: зачем вводить отдельную команду для задания ограничений по ***th***? Это было нужно для возможности задания одноцикловых и мультицикловых путей с задержкой. Это очень интересная техника, позволяющая «чудесным» образом выполнять заданные ограничения в высокочастотных интерфейсах, но начать автор предлагает с рассмотрения примера, демонстрирующего основное назначение мультицикловых путей — оптимизацию проектов по быстродействию.

Использование мультицикловых путей для оптимизации проектов

Рассмотрим простой проект, небольшую часть системы ЦОС — цифровой фильтр, с реализацией на цепи сумматоров:

```
module fir
  #(parameter int pDAT_W = 16)
  (input logic iclk, ival, input logic [pDAT_W-1 : 0] idat,
    output logic oval, output logic [pDAT_W-1 : 0] odat
    ) ;

  localparam int cCOE_W = 14;
  localparam int P = 23;
  localparam int cRES_W = pDAT_W + cCOE_W;

  typedef logic signed [pDAT_W-1 : 0] dat_t ;
  typedef logic signed [cCOE_W-1 : 0] coe_t ;
  typedef logic signed [cRES_W-1 : 0] res_t ;

  localparam coe_t coe [0:P-1] = '{
    -1, -2, 4, 15, -24, -76, 142, 246, -657, -501, 3331,
    6225,
    3331, -501, -657, 246, 142, -76, -24, 15, 4, -2, -1
  };

  dat_t resample_in = '0;

  res_t mul_result [0: P-1] /*synthesis multstyle = "logic"*/;
  res_t acc [0: P-1] ;

  always_ff @(posedge iclk) begin
    int i;
    if (ival) begin
      resample_in <= idat;
      // mull + 1
      for (i = 0; i < P; i++) begin
        mul_result[i] <= resample_in * coe[P-1-i];
      end
      // add + 64 tap shift reg
      for (i = 0; i < P; i++) begin
        if (i == 0)
          acc [i] <= mul_result[i];
        else
          acc [i] <= mul_result[i] + acc[i-1];
      end
    end
    oval <= ival;
  end

  assign odat = acc[$high(acc)][cRES_W-1 : cRES_W-pDAT_W];

endmodule
```

Как видите, это обычный фильтр 23-го порядка, с симметричной характеристикой. Теперь представим себе, что в нашем проекте есть тактовая частота 250 МГц (***iclk***) и символьная частота 125 МГц (***ival***). Начальный ***sdc***-файл для этого проекта будет такой:

```
set_time_format -unit ns -decimal_places 3
derive_clock_uncertainty
create_clock -period 250MHz -name {iclk} [get_ports {iclk}]
```

Собираем проект, запускаем временной анализ и видим большое количество нарушений заданных временных ограничений (рис. 52). Помимо этого, TimeQuest информирует нас о том, что тактовая частота данного проекта составляет всего 123 МГц (рис. 53).

Рис. 52. Отчет о нарушениях временных ограничений проекта fir

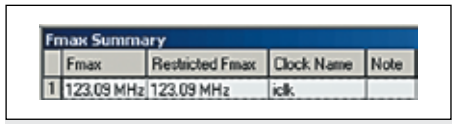

Рис. 53. Отчет о максимальной тактовой частоте проекта fir

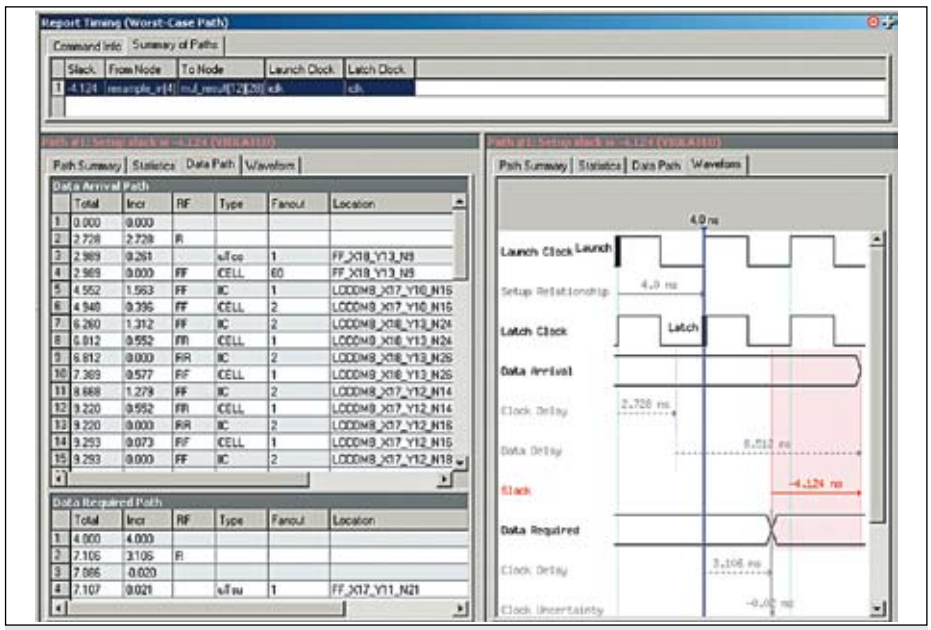

Рис. 54. Подробный отчет о временном анализе путей проекта fir

Давайте посмотрим, как именно он это измерил. Смотрим критический путь по одному из сигналов (рис. 54) и видим, что TimeQuest использует метод одноциклового анализа. То есть он делает анализ, исходя из предположения, что данные изменяются каждый такт. В этом случае символьная частота будет 250 МГц (сигнал ***ival*** всегда стоит в единице). Но ведь у нас символьная частота всего 125 МГц. Как указать это TimeQuest?

Тут на помощь и приходят команды задания мультицикловых путей. Укажем TimeQuest, что пути данных в этом проекте двухцикловые. Такими путями в проекте ***fir*** являются пути между регистрами ***resample_in → mul_result, mul_result → acc, acc → acc***. Добавим в ***sdc***-файл команды задания мультициклов для этих цепей:

```
set_multicycle_path -from [get_registers {resample_in[*]}] -to [get_registers {mul_result[*]}] -setup -end 2
set_multicycle_path -from [get_registers {resample_in[*]}] -to [get_registers {mul_result[*]}] -hold -end 1
set_multicycle_path -from [get_registers {mul_result[*] acc[*]}] -to [get_registers {acc[*]}] -setup -end 2
set_multicycle_path -from [get_registers {mul_result[*] acc[*]}] -to [get_registers {acc[*]}] -hold -end 1
```

Заново собираем проект, запускаем временной анализ и видим, что максимальная тактовая частота стала больше требуемой (рис. 55) и анализ проводится по правилам, соответствующим заложенной функциональности (рис. 56).

В этом примере мы не изменяли архитектуру проекта, не перебирали различные настройки Quartus: мы всего лишь правильно описали требования к временной функциональности проекта. Определение мест, временные требования к которым могут быть снижены, положительно сказывается на количестве используемых ресурсов ПЛИС и на времени сборки проекта. Но определение всех таких мест в большом проекте может быть нетривиальной задачей.

Один из вариантов оптимизации этой задачи — это задание маски имени. То есть все регистры должны содержать в имени уникальный префикс/суффикс, по которому их можно будет однозначно идентифицировать. Но такое решение накладывает ограничение на пространство имен при разработке проекта и плохо подходит к уже существующему проекту.

Другой способ более красивый. Судя по проекту ***fir***, на все регистры-источники/приемники приходит сигнал разрешения. Следовательно, для задания ограничений на весь проект можно описать мультицикл так:

```
set_multicycle_path 2 -to [get_fanouts [get_ports {ival}] -through [get_pins -hier *|*ena*]] -setup
set_multicycle_path 1 -to [get_fanouts [get_ports {ival}] -through [get_pins -hier *|*ena*]] -hold
```

То есть, используя функцию ***get_fanouts***, мы назначаем мультицикловые пути на все регистры, на которые приходит сигнал разрешения тактирования (***ival***). Использование такого определения мультицикла дает точно такой же результат, что и использование

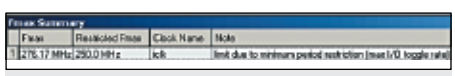

Рис. 55. Отчет о максимальной тактовой частоте проекта fir после задания мультицикловых путей

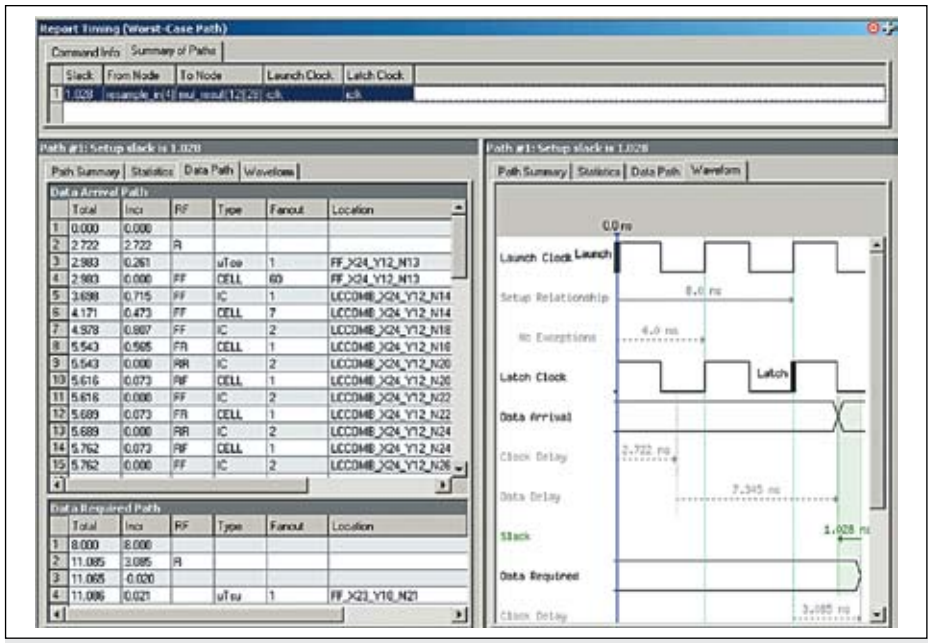

Рис. 56. Подробный отчет о временном анализе путей проекта fir после задания мультицикловых путей

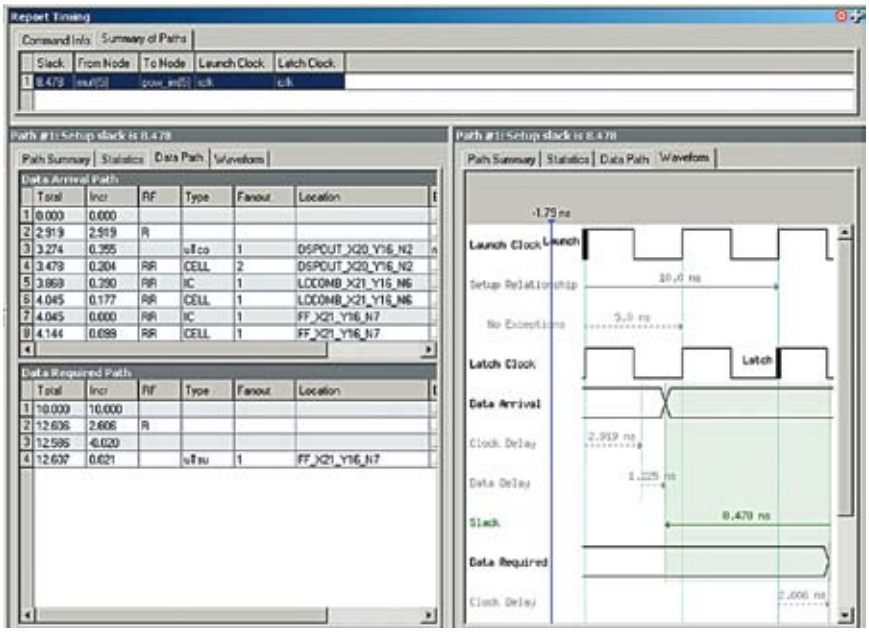

Рис. 57. Подробный отчет о временном анализе путей проекта multicycle

определения через явное задание имен регистров. Но этот метод таит в себе большую опасность. Рассмотрим модуль расчета мощности комплексного сигнала:

```
module multicycle
   #(parameter int pDAT_W = 16)
   (input logic iclk, ival, input logic [pDAT_W-1 : 0] idat_re, idat_im,
      output logic oval, output logic [pDAT_W-1 : 0] power
      ) ;

   localparam int cRES_W = pDAT_W + pDAT_W;

   logic signed [pDAT_W-1 : 0] dat_re;
   logic signed [pDAT_W-1 : 0] dat_im;

   logic signed [pDAT_W-1 : 0] dat2mult;
   logic signed [cRES_W-1 : 0] mult;

   logic signed [cRES_W-1 : 0] pow_re;
   logic signed [cRES_W-1 : 0] pow_im;
   logic signed [cRES_W-1 : 0] pow;

   always_ff @(posedge iclk) begin
      if (ival) begin
         dat_re <= idat_re;
         dat_im <= idat_im;
      end

      dat2mult <= ival ? dat_re : dat_im;
      mult <= dat2mult * dat2mult;

      if (ival)
         pow_re <= mult;
      if (~ival)
         pow_im <= mult;

      if (ival) begin
         pow <= pow_re + pow_im;
      end

      oval <= ival;
   end

   assign power = pow[cRES_W-1 : cRES_W-pDAT_W];

endmodule
```

По коду проекта ***multicycle*** видно, что пути ***mult → pow_re/pow_im*** и ***pow_im → pow*** не мультицикловые (!!!), тогда как TimeQuest считает наоборот (рис. 57).

Еще одна особенность использования мультицикловых путей в подобных проектах заключается в том, что они требуют равномерности поступления сигналов разрешения. Для примера вернемся к проекту ***fir***. Мультицикловые пути будут соответствовать действительности только в том случае, если символьная частота (сигналы на порту ***ival***) будет распределена равномерно. Если же это не выполняется (например, это происходит в цепях восстановления тактовой частоты на основе интерполяторов и NCO), то применять мультициклы нельзя, или нужно задавать не среднее количество тактов, а их минимальное количество между сигналами разрешения.

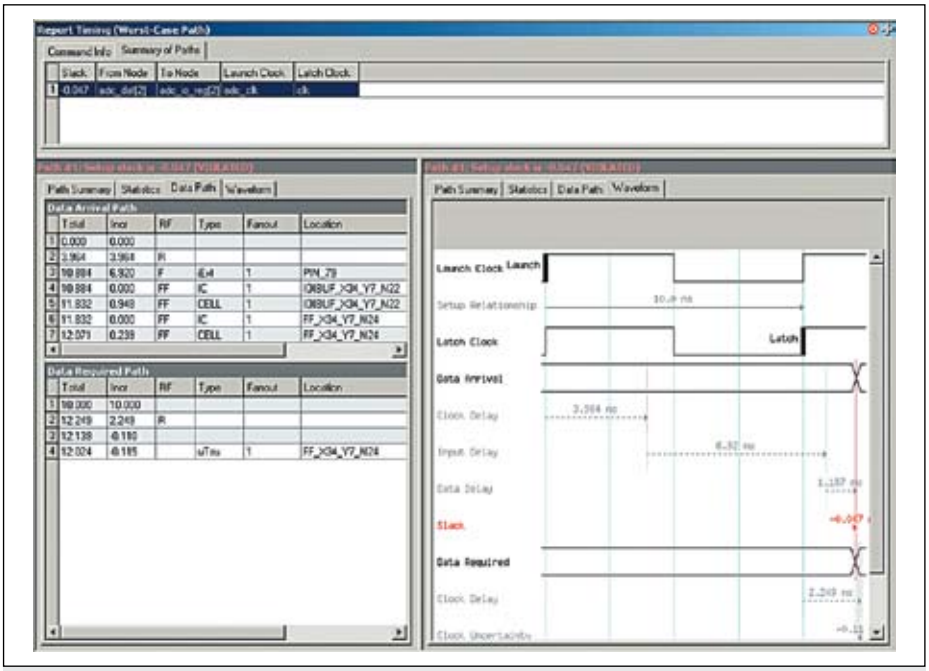

Рис. 58. Подробный отчет о нарушении временных ограничений проекта adc

В качестве последнего напутствия автор хочет сказать, что сломать проект при неправильном использовании мультицикловых путей очень легко. Если вы не уверены в своих силах, то используйте их только тогда, когда это вам действительно нужно. При этом уделите особое внимание стыкам мультицикловых и одноцикловых путей, во избежание появления трудно диагностируемых ошибок.

Использование мультицикловых путей для задания временных ограничений в интерфейсах

Мы рассмотрели использование мультицикловых путей для оптимизации проектов, но это не единственная область их применения. С помощью мультицикловых путей можно эффективно исправлять различные нарушения временных ограничений в интерфейсах. Рассмотрим это на различных примерах.

Устранение временных нарушений в Source Synchronous Input

В третьей части статьи мы рассматривали проект ***Source Synchronous Input*** интерфейса:

```
module adc (input clk, input [9 : 0] adc_dat, output adc_clk, output
logic [9 : 0] data);

   logic [9 : 0] adc_io_reg;
   logic [9 : 0] adc_reg;

   always_ff @(posedge clk) begin
      {adc_reg, adc_io_reg} <= {adc_io_reg, adc_dat};
   end

   assign adc_clk = clk;

   always_ff @(posedge clk) begin
      data <= adc_reg;
   end

endmodule
```

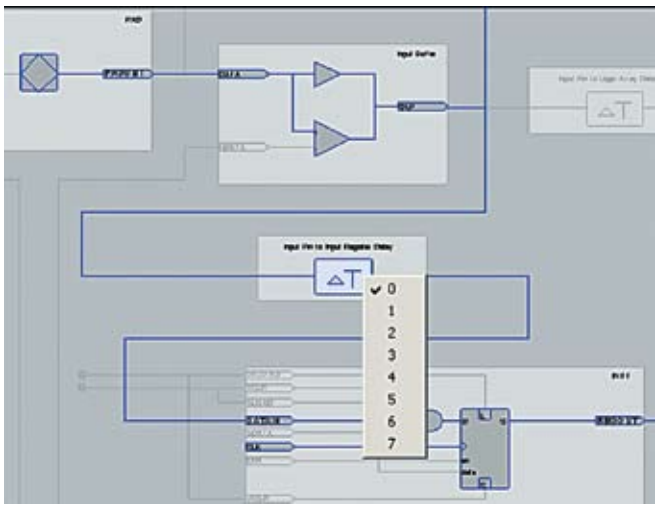

Рис. 59. Буфер ввода/вывода ПЛИС проекта adc

В том проекте АЦП работал на частоте 50 МГц и тактировался от ПЛИС. Увеличим частоту тактирования до 100 МГц. Тогда ***sdc***-файл будет следующий:

```
set_time_format -unit ns -decimal_places 3
derive_clock_uncertainty

create_clock -period 100MHz -name {clk} [get_ports {clk}]
create_generated_clock -name {adc_clk} -source [get_ports {clk}]
[get_ports {adc_clk}]

set clkA_delay_max [expr 30.0*0.007]
set clkA_delay_min [expr 20.0*0.007]

set bdA_delay_max [expr 30.0*0.007]
set bdA_delay_min [expr 20.0*0.007]

set Tco_max 6.5
set Tco_min 2.5

set usedTsu [expr $clkA_delay_max + $Tco_max + $bdA_delay_max]
set usedTh [expr $clkA_delay_min + $Tco_min + $bdA_delay_min]

set_input_delay -clock {adc_clk} -max $usedTsu [get_ports {adc_dat[*]}]
set_input_delay -clock {adc_clk} -min $usedTh [get_ports {adc_dat[*]}]
```

Собираем проект, запускаем временной анализ и видим нарушения во входном интерфейсе (рис. 58). При анализе нарушения видно, что задержка по данным слишком большая. Есть ли резервы ее уменьшить? Смотрим, что Quartus сделал в буфере ввода/вывода (рис. 59). Он сделал все, чтобы уменьшить задержку. Но этого оказалось недостаточно.

Для устранения нарушения можно использовать PLL, как мы делали в прошлой статье. Но давайте сделаем это без PLL. В качестве первого шага сдвинем точку временного анализа на 1 такт частоты (задержим сигнал на 1 период). Добавляем в ***sdc***-файл следующую строку:

```
set_multicycle_path -end -from {get_clocks {adc_clk}} -to {get_clocks
{clk}} -setup 2
```

Автор хочет отметить, что он добавил всего одну команду вместо двух. Потому что значение смещения в команде по ***th*** по умолчанию и есть нужное нам значение (0). Собираем проект, запускаем анализ и снова видим нарушения в интерфейсе (рис. 60). В этот раз задержка сигнала слишком маленькая, а Quartus сделал все (рис. 61), чтобы ее увеличить, но этого не хватило.

Для устранения нарушения нужно выбрать такое решение, которое даст значение задержки между этими двумя вариантами. Это можно сделать с помощью инверсии сигнала тактовой частоты. В коде модуля ***adc*** делаем инверсию выходного сигнала тактовой частоты:

```
assign adc_clk = ~clk;
```

И описываем это изменение в ***sdc***-файле:

```
create_generated_clock -name {adc_clk} -invert -source [get_ports
{clk}] [get_ports {adc_clk}]
```

Нарушений временных ограничений больше нет (рис. 62), и не потребовалось использовать PLL.

Устранение временных нарушений в Source Synchronous Input/Output

Рассмотрим еще пример того, как могут не совпадать взгляды TimeQuest и разработчика на временной анализ проекта. Этот пример мне прислал пользователь ***vadimuzzz*** с форума www.electronix.ru. К сожалению,

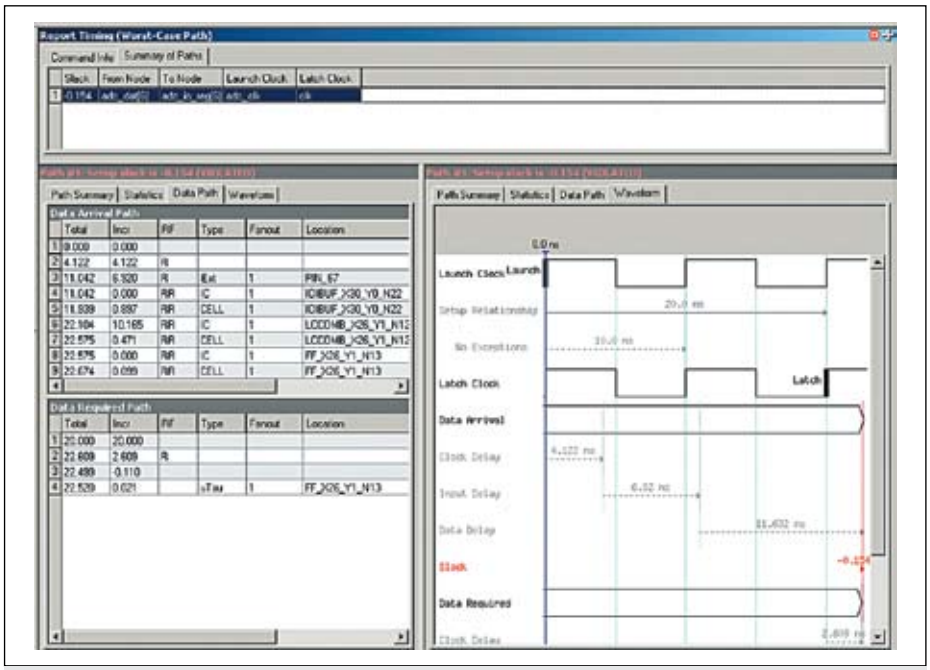

Рис. 60. Подробный отчет о нарушении проекта adc после добавления мультицикла

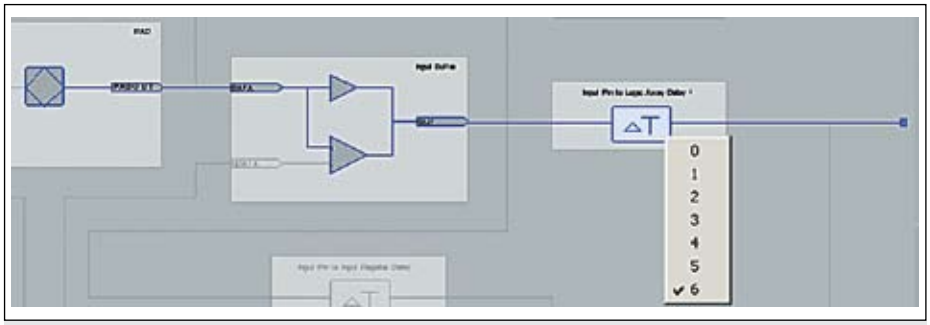

Рис. 61. Ячейка ввода/вывода ПЛИС проекта adc после добавления мультицикла

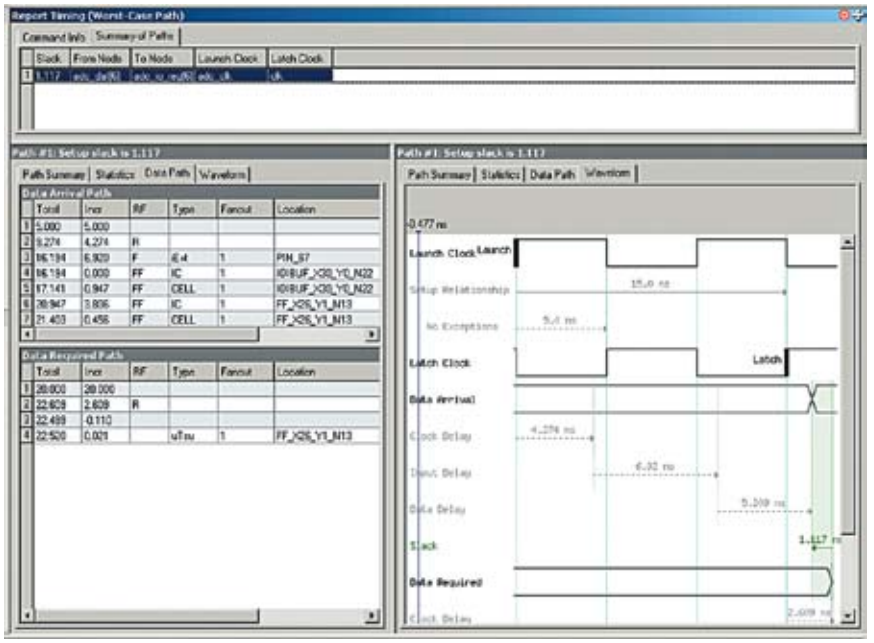

Рис. 62. Подробный отчет о нарушении проекта adc после добавления мультицикла и инверсии сигнала тактовой частоты

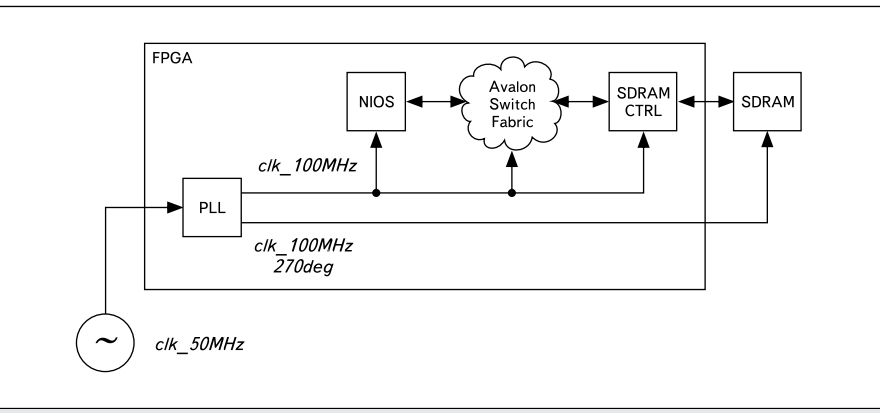

Рис. 63. Функциональная схема проекта с NIOS и SDRAM

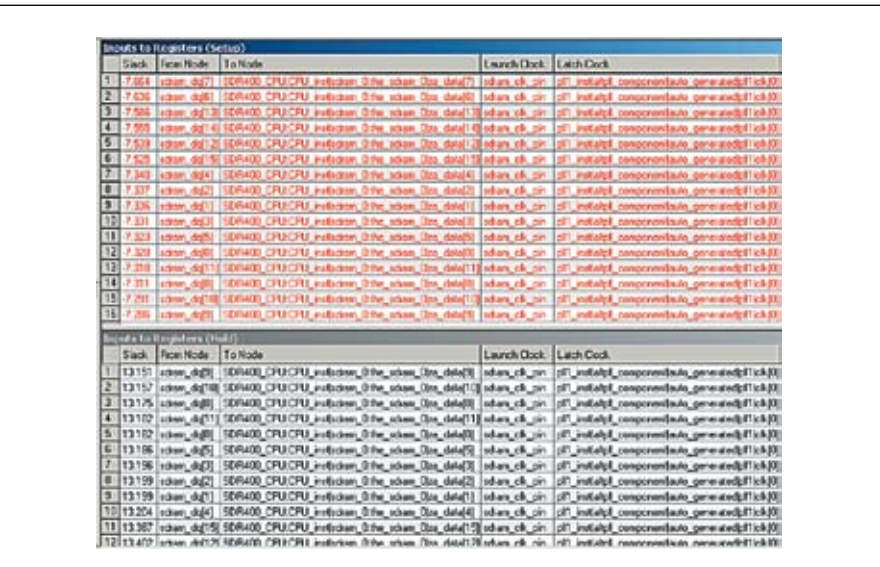

Рис. 64. Отчет о временном анализе проекта с NIOS

размер статьи не позволяет привести код проекта, ограничимся только его функциональной схемой (рис. 63). Как видно, для устранения нарушений временных ограничений SDRAM тактируется сигналом тактовой частоты, сдвинутым по фазе на 270°.

Интересующая нас часть файла временных ограничений этого проекта была такой:

```
create_clock -name {sys_clk} -period 50MHz [get_ports {sys_clk}]

derive_pll_clocks -create_base_clocks
derive_clock_uncertainty

set main_clk pll1_inst|altpll_component|auto_generated|pll1|clk[0]
set sdram_clk pll1_inst|altpll_component|auto_generated|pll1|clk[1]

create_generated_clock -name sdram_clk_pin -source $sdram_clk
[get_ports {sdram_clk}]

set_clock_groups -exclusive -group [list $main_clk $sdram_clk
sdram_clk_pin]

#**************************************************************
# board delays
#**************************************************************
set CLK_BD_MIN [expr 0.415 - 0.1]
set CLK_BD_MAX [expr 0.415 + 0.1]

set DATA_BD_MIN [expr 0.415 - 0.1]
set DATA_BD_MAX [expr 0.415 - 0.1]
#**************************************************************
# sdram timings
#**************************************************************
set Tac         5.5
set tOH         2.5

#**************************************************************
# Set Input Delay
#**************************************************************
set_input_delay -max -clock sdram_clk_pin [expr $CLK_BD_MAX +
$tAC + $DATA_BD_MAX] [get_ports {sdram_dq[*]}]
set_input_delay -min -clock sdram_clk_pin [expr $CLK_BD_MIN +
$tOH + $DATA_BD_MIN] [get_ports {sdram_dq[*]}]
```

Как видите, все было задано правильно. Но при временном анализе TimeQuest обнаружил нарушения временных ограничений (рис. 64), возникающие при чтении данных из SDRAM (при передаче информации с тактовой частоты ***sdram_clk_pin*** на тактовую частоту ***main_clk***). На первый взгляд подробный отчет о временном анализе (рис. 65) вызывает подозрения в адекватности TimeQuest, но вместо этого внимательно проанализируем результат. Видно, что TimeQuest выбрал не ту пару ***Launch/Latch Setup Edge*** для анализа. Поэтому для корректного анализа ему нужно указать те пары фронтов тактовых сигналов, которые нам нужны. Для этого добавляем в ***sdc***-файл следующую строку:

```
set_multicycle_path -from [get_clocks {sdram_clk_pin}] -to [get_
clocks $main_clk] -setup -end 2
```

Таким образом, временной анализ стал соответствовать функциональности проекта (рис. 66), и нарушений временных ограничений нет.

Использование мультицикловых путей для асинхронных интерфейсов ввода/вывода

В качестве последнего примера рассмотрим использование мультицикловых путей в асинхронных интерфейсах. Рассмотрим простой

контроллер асинхронной статической памяти (SRAM) со временем доступа 10 нс, работающий на тактовой частоте 100 МГц:

```
module async_ctrl
(
   input clk, reset,
   input sys_write, sys_read,
   input [7 : 0] sys_addr, sys_wdat,
   output logic [7 : 0] sys_rdat,
   output logic sys_ready,
   output logic ce_n, oe_n, we_n,
   output logic [7 : 0] addr,
   inout wire [7 : 0] data
);

   enum {cIDLE_STATE, cWRITE_STATE, cREAD_STATE, cWAIT_
STATE} state, next_state;

   always_ff @(posedge clk) begin
      if (reset)
         state <= cIDLE_STATE;
      else
         state <= next_state;
   end

   always_comb begin
      next_state = cIDLE_STATE;
      case (state)
         cIDLE_STATE : begin
            if ((sys_write ^ sys_read) & ~sys_ready)
               next_state = sys_write ? cWRITE_STATE : cREAD_STATE;
            else
               next_state = cIDLE_STATE;
         end
         cWRITE_STATE : next_state = cWAIT_STATE;
         cREAD_STATE : next_state = cWAIT_STATE;
         cWAIT_STATE : next_state = cIDLE_STATE;
      endcase
   end

   always_ff @(posedge clk) begin : ram_ctrl_path
      if (reset) begin
         {ce_n, oe_n, we_n} <= 3'b111;
      end
      else begin
         case (next_state)
            cIDLE_STATE : {ce_n, oe_n, we_n} <= 3'b111;
            cWRITE_STATE : {ce_n, oe_n, we_n} <= 3'b010;
            cREAD_STATE : {ce_n, oe_n, we_n} <= 3'b001;
         endcase
      end
   end

   logic [7 : 0] wdat;

   always_ff @(posedge clk) begin : ram_addr_data_path
      case (next_state)
         cWRITE_STATE : {addr, wdat} <= {sys_addr, sys_wdat};
         cREAD_STATE : {addr, wdat} <= {sys_addr, 8'hz};
      endcase
   end

   assign data = wdat;

   wire read_done = (state == cWAIT_STATE & ~oe_n);
   wire write_done = (next_state == cWAIT_STATE & oe_n);

   always_ff @(posedge clk) begin : sys_ctrl_path
      if (reset)
         sys_ready <= 1'b0;
      else
         sys_ready <= write_done | read_done;
   end

   always_ff @(posedge clk) begin : sys_data_path
      if (read_done)
         sys_rdat <= data;
   end

endmodule
```

На основе третьей части статьи мы знаем, что есть два способа задания временных ограничений для таких проектов:

- Объявить пути до всех портов ввода/вывода как ложные и выдержать нужные задержки с помощью состояний конечного автомата (КА). Этот метод прост, но чреват уменьшением возможной производительности системы из-за лишних задержек.

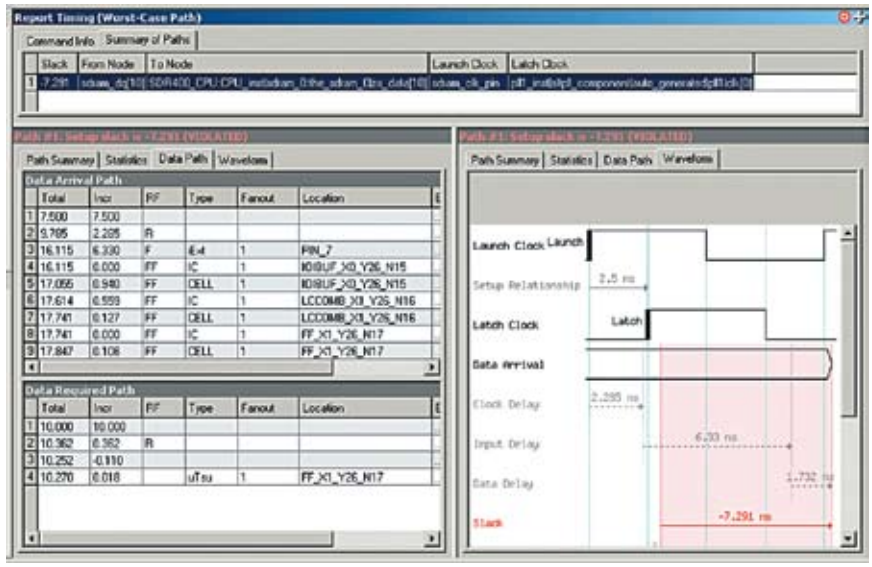

Рис. 65. Подробный отчет о временном анализе проекта NIOS

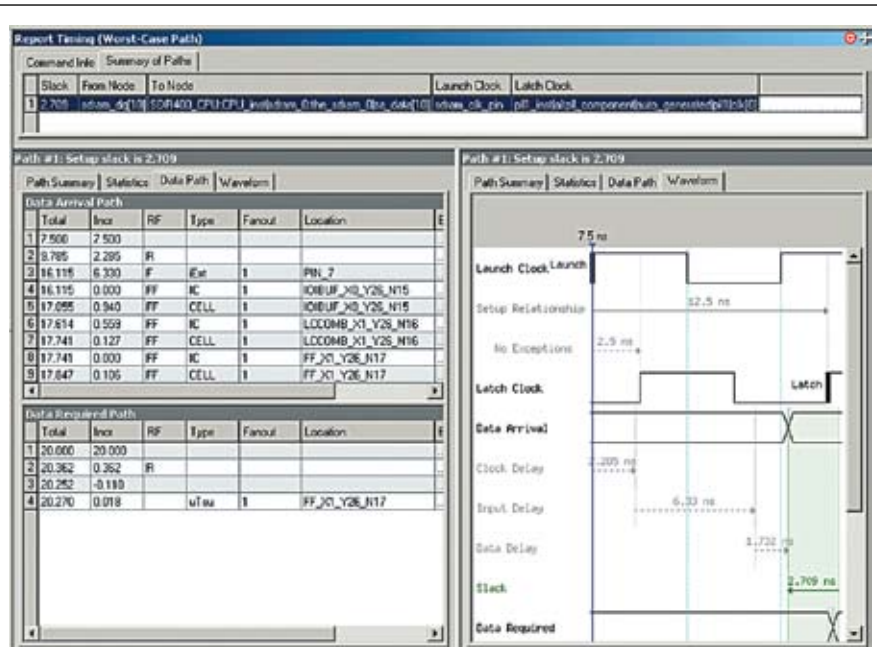

Рис. 66. Подробный отчет о временном анализе проекта NIOS после задания правильных фронтов тактовых сигналов

- Использовать возможности задания ограничений для синхронных интерфейсов. Этот метод более сложный, потому что нужно отобразить сигналы асинхронного интерфейса в сигналы, подходящие для синхронного анализа.

Первый метод тривиален, и на нем останавливаться не будем. Кроме того, КА у нас уже есть, не будем его трогать. Начальный файл временных ограничений для проекта ***async_ctrl*** будет таким:

```
derive_clock_uncertainty
create_clock -period 100MHz -name {clk} [get_ports {clk}]

# set output constraints (remember to use FAST_OUTPUT_
REGISTER_HERE)
set_false_path -from [all_clocks] -to [get_ports {ce_n oe_n we_n
addr[*] data[*]}]

create_generated_clock -name ce_clk -source [get_ports {clk}] [get_
registers {ce_n~reg0}]

set_input_delay -clock {ce_clk} -reference_pin [get_ports {ce_n}] 10
[get_ports data[*]]
```

В качестве сигнала «тактовой» частоты для асинхронного интерфейса выбран сигнал разрешения работы (***ce***). Потому что в данном контроллере этот сигнал используется при любых транзакциях к памяти. Задержка чтения из памяти составляет 10 нс, задержками сигнала на плате пренебрегаем. Собираем проект, запускаем анализ и видим нарушение временных огра-

ничений (рис. 67). При анализе видно, что задержка по данным слишком большая, что приводит к нарушениям. Так и должно быть для памяти со временем доступа 10 нс и задержками сигналов в портах ввода/вывода ПЛИС 3–5 нс. Задержка в два такта частоты есть в КА проекта ***async_ctrl***, и для правильного задания временных ограничений эту задержку нужно указать и в ***sdc***-файле:

```
set_multicycle_path -from [get_clocks {ce_clk}] -to [get_clocks {clk}]
-setup -end 2
```

Собираем проект и видим (рис. 68), что нарушений временных ограничений нет, и временной анализ проводится в соответствии с логикой работы проекта.

Как видите, не всегда действия TimeQuest совпадают с требованиями разработчика и логикой работы проекта. Поэтому нужно обязательно овладеть навыками использования исключений временных ограничений. А то когда-нибудь случится так, что вы не сможете «объяснить» TimeQuest, что вы от него хотите.

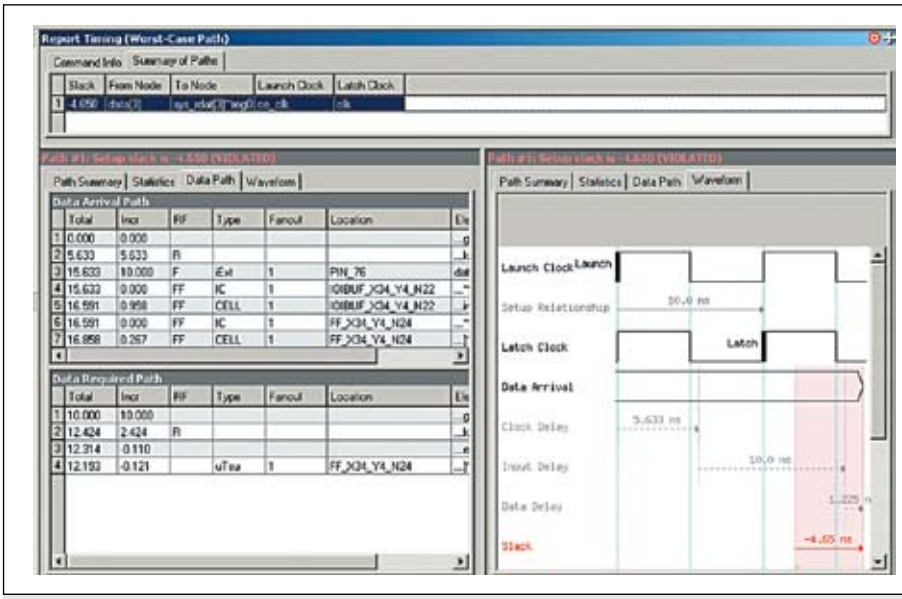

Рис. 67. Подробный отчет о временном анализе проекта async_ctrl

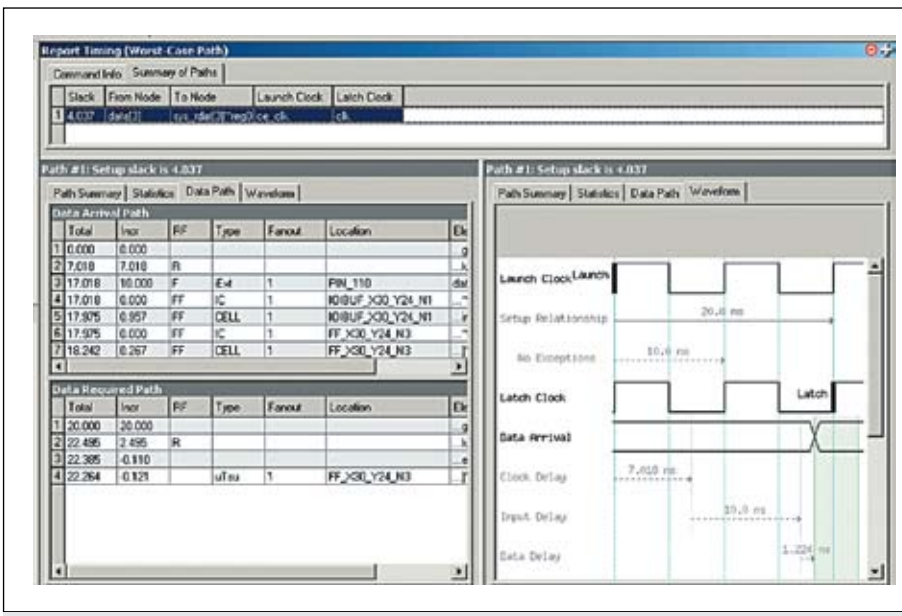

Рис. 68. Подробный отчет о временном анализе проекта async_ctrl после задания мультицикловых путей

Заключение

Мы закончили рассмотрение основ задания временных ограничений. Формат статьи не позволяет проанализировать все возможные проекты, но большинство из них может быть сведено к рассмотренным типовым.

Надеюсь, что у читателя изменилось отношение к временным ограничениям, TimeQuest и SDC в лучшую сторону. Овладение этими инструментами позволит вам увеличить качество ваших проектов. Но помните главное: правильное задание временных ограничений требует от разработчика понимания их физического смысла, внимательности и аккуратности. Испортить проект неправильно заданными ограничениями легко, а понять в этом случае, почему проект работает неправильно, гораздо сложнее.

Автор будет рад получить ваши отзывы и замечания о цикле статей, а также предложения по написанию дополнений к нему с учетом ваших пожеланий.

Литература

1. Quartus II Handbook Version 9.0 — http://www.altera.com/literature/lit-qts.jsp
2. SDC and TimeQuest API Reference Manual — http://www.altera.com/literature/manual/mnl_sdctmq.pdf
3. Quartus II TimeQuest Timing Analyzer Cookbook — http://www.altera.com/literature/manual/mnl_timequest_cookbook.pdf
4. Applying Multicycle Exceptions in the TimeQuest Timing Analyzer — http://www.altera.com/literature/an/an481.pdf
5. Шехалев Д. Synopsys Design Constraint — язык задания временных ограничений на примере Altera Time Quest. Части 1–3//Компоненты и технологии. 2010. № 9–11.
6. http://embedders.org/blog/des00