**Основы статического временного анализа.**

**Часть 5: False Path Constraint.**

**Введение.**

В данной статье представлен временной анализ передачи сигналов между двумя тактовыми доменами. Показано несколько способов исключения путей из временного анализа. Рассмотрены инструменты Vivado, предназначенные для проверки корректности передачи данных между тактовыми доменами.

1. **Пересечение тактовых доменов.**

В предыдущих работах серии был представлен временной анализ передачи данных между двумя триггерами с общим тактовым сигналом. Теперь же будет рассмотрен случай, когда запускающий и защелкивающий триггеры имеют разные тактовые сигналы. Данная статья частично опирается на материал, рассмотренный ранее в [1]. Предполагается, что читатель уже знаком с такими понятиями, как ограничение на максимальное (*Setup*) и минимальное (*Hold*) время распространения сигнала, запас (*Slack*) и т.д.

Для начала введем несколько определений. Множество триггеров, которые тактируются одним и тем же сигналом, будем называть   
тактовым доменом (Clock Domain). Пересечение тактовых доменов   
(Clock Domain Cross – CDC) возникает в случае, когда при передаче данных запускающий триггер находится в одном домене, а защелкивающий – в другом.

Рассмотрим, как выполняется временной анализ при пересечении тактовых доменов. На рисунке 1 показан путь, на который нанесены задержки для данных и тактовых сигналов. Передающий триггер FF1 тактируется сигналом CLK\_1, а приёмный триггер FF2 – сигналом CLK\_2.

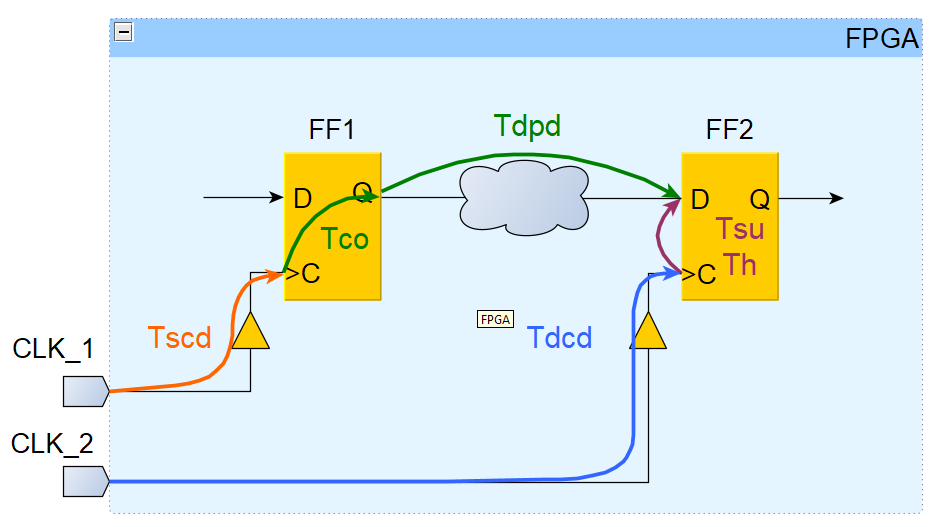


Рисунок 1. Путь с задержками для данных и тактовых сигналов.

Ниже даны определения задержек, представленных на рисунке 1.

* *Tscd* (***S****ource* ***C****lock* ***D****elay*) – задержка запускающего тактового сигнала от ножки CLK\_1 FPGA до тактового входа триггера FF1;
* *Tdcd* (***D****estination* ***C****lock* ***D****elay*) – задержка защелкивающего тактового сигнала от ножки CLK\_2 FPGA до тактового входа триггера FF2;
* *Tco* (***C****lock to* ***O****utput*) – интервал времени между приходом фронта на тактовый вход триггера и появлением данных на его выходе Q;
* *Tdpd* (***D****ata* ***P****ropagation* ***D****elay*) – задержка распространения данных по соединениям и через комбинационную логику;
* *Tsu* (***S****et****U****p time*) – время установки защелкивающего триггера;
* *Th* (***H****old time*) – время удержания защелкивающего триггера.

Для начала рассмотрим, каким образом выполняется анализ для проверки ограничения на максимальное время распространения (*Setup*). Пусть запускающий фронт появляется на ножке CLK\_1 FPGA в момент времени *Tclk\_1\_su*. Запишем уравнения для расчета фактического времени прибытия данных (см. рисунок 1):

* Время прибытия запускающего фронта к триггеру FF1  
  (***S****ource* ***С****lock* ***A****rrival time*):
* Время прибытия данных на вход триггера FF2 (***D****ata* ***A****rrival time*):

Далее, считая, что защелкивающий фронт появляется на ножке CLK\_2 FPGA в момент времени *Tclk\_2\_su*, получим уравнения для требуемого времени прибытия данных ко входу триггера FF2:

* Время прибытия защелкивающего фронта к триггеру FF2   
  (***D****estination* ***C****lock* ***A****rrival time*):
* Требуемое время прибытия данных (***D****ata* ***R****equired time*):

При анализе по *Setup* значение запаса рассчитывается с помощью следующего выражения:

Подставим в него найденные ранее результаты и получим:

где *ΔTsu* – интервал времени между появлением запускающего фронта на ножке CLK\_1 и защелкивающего фронта на ножке CLK\_2:

Теперь рассмотрим, как выполняется анализ для проверки ограничения на минимальное время распространения (*Hold*). Пусть запускающий фронт появляется на ножке CLK\_1 в момент времени *Tclk\_1\_h*. Напомним, что при анализе по *Hold* защелкивающим фронтом является тот, с помощью которого триггер FF2 принимает предыдущие данные [1]. Будем считать, что этот фронт появляется на ножке CLK\_2 в момент времени *Tclk\_2\_h*.

Для начала получим уравнения для фактического времени прибытия данных:

* Время прибытия запускающего фронта к триггеру FF1  
  (***S****ource* ***С****lock* ***A****rrival time*):
* Время прибытия данных на вход триггера FF2 (***D****ata* ***A****rrival time*):

Далее, запишем уравнения для требуемого времени прибытия данных:

* Время прибытия защелкивающего фронта к триггеру FF2  
  (***D****estination* ***C****lock* ***A****rrival time*):
* Требуемое время прибытия данных (***D****ata* ***R****equired time*):

Уравнение для расчета *Slack* при анализе по *Hold* имеет вид:

Учитывая предыдущие результаты, можем записать выражение для *Slack* в следующем виде:

где *ΔTh* – интервал времени между появлением запускающего и защелкивающего фронтов:

В уравнениях для *Slack*, полученных ранее, *ΔTsu* и *ΔTh* являются единственными слагаемыми, величины которых не известны временному анализатору после размещения и трассировки проекта. Эти значения должны рассчитываться исходя из временных соотношений между фронтами тактовых сигналов CLK\_1 и CLK\_2.

Для начала рассмотрим способ вычисления *ΔTsu* [2]. Как и всегда временной анализ проводится для самого пессимистичного случая. При анализе по *Setup* этому соответствует минимальное значение *ΔTsu*, так как данное слагаемое входит в уравнение для *Slack* c положительным знаком. В качестве примера будем считать, что период сигнала CLK\_1 равен 6 нс, а   
период CLK\_2 – 4 нс. Также пусть в нулевой момент времени формируются фронты обоих тактовых сигналов (см. рисунок 2).

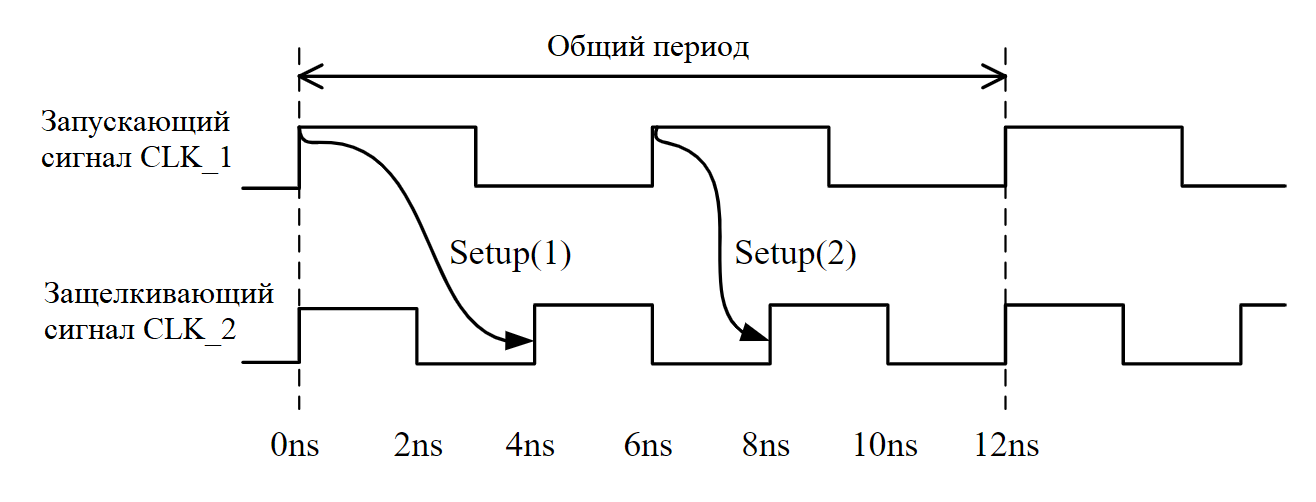


Рисунок 2. Временная диаграмма сигналов для анализа по *Setup*.

Для расчета *ΔTsu* временной анализатор поочередно выбирает один из фронтов сигнала CLK\_1 и рассматривает его в качестве запускающего. То есть, проверяются случаи, когда запускающий фронт формируется в момент времени 0 нс, 6 нс, 12 нс и так далее. Затем для каждого запускающего фронта CLK\_1 находится соответствующий защелкивающий фронт сигнала CLK\_2. При этом ищется ближайший фронт CLK\_2, который появляется строго после запускающего фронта CLK\_1.

Например, как видно из рисунка 2, первому фронту CLK\_1 (0 нс) соответствует второй фронт CLK\_2 (4 нс), а второму фронту CLK\_1 (6 нс) –третий фронт CLK\_2 (8 нс). Для каждой такой пары по формуле (1) рассчитывается значение *ΔTsu*. В нашем примере для первой пары   
фронтов получаем *ΔTsu* = 4 – 0 = 4 нс. Для второй пары находим, что  
 *ΔTsu* = 8 – 6 = 2 нс. После 12 нс временная диаграмма сигналов будет повторяться, поэтому для третьего фронта CLK\_1 опять получим, что   
*ΔTsu* = 4 нс (см. рисунок 2). Интервал времени, через который повторяются временные соотношения между двумя тактовыми сигналами, называют общим периодом.

После того, как временной анализатор найдет общий период и рассмотрит на нем все фронты CLK\_1, он останавливается и находит минимальное значение *ΔTsu*. В нашем примере это 2 нс. Именно эта величина в дальнейшем будет использоваться в расчете *Slack* при анализе по *Setup*. Отметим также, что возможна ситуация, при которой временной анализатор не сможет найти общий период в течение 1000 тактов сигнала CLK\_1. В этом случае он остановится и будет использовать минимальное значение *ΔTsu*, обнаруженное в течение этих 1000 тактов.

Теперь рассмотрим, как рассчитывается значение *ΔTh* для анализа по *Hold*. На предыдущем этапе при вычислении *ΔTsu*, анализатор нашел все пары фронтов CLK\_1 и CLK\_2, появляющиеся в течение общего периода. Для каждой такой пары должны быть проверены два условия [2]:

1. данные, которые передаются по запускающему фронту CLK\_1 не должны быть приняты фронтом CLK\_2, который является предыдущим по отношению к текущим защелкивающему фронту CLK\_2;
2. данные, которые передаются фронтом CLK\_1, следующим после запускающего, не должны быть приняты текущим защелкивающим фронтом CLK\_2.

Для примера изучим рисунок 3, на котором первая пара фронтов, полученная при анализе по *Setup*,обозначена как S1. Первое условие для S1 соответствует стрелке H1a. Данные, запущенные фронтом CLK\_1 в нулевой момент времени, не должны быть приняты фронтом CLK\_2, который появляется также в нулевой момент времени. Второму условию соответствует стрелка H1b. Следующий фронт CLK\_1 относительно пары S1 формируется через 6 нс. Данные, которые он запустит, не должны быть приняты защелкивающим фронтом из S1, который появится в момент времени 4 нс.

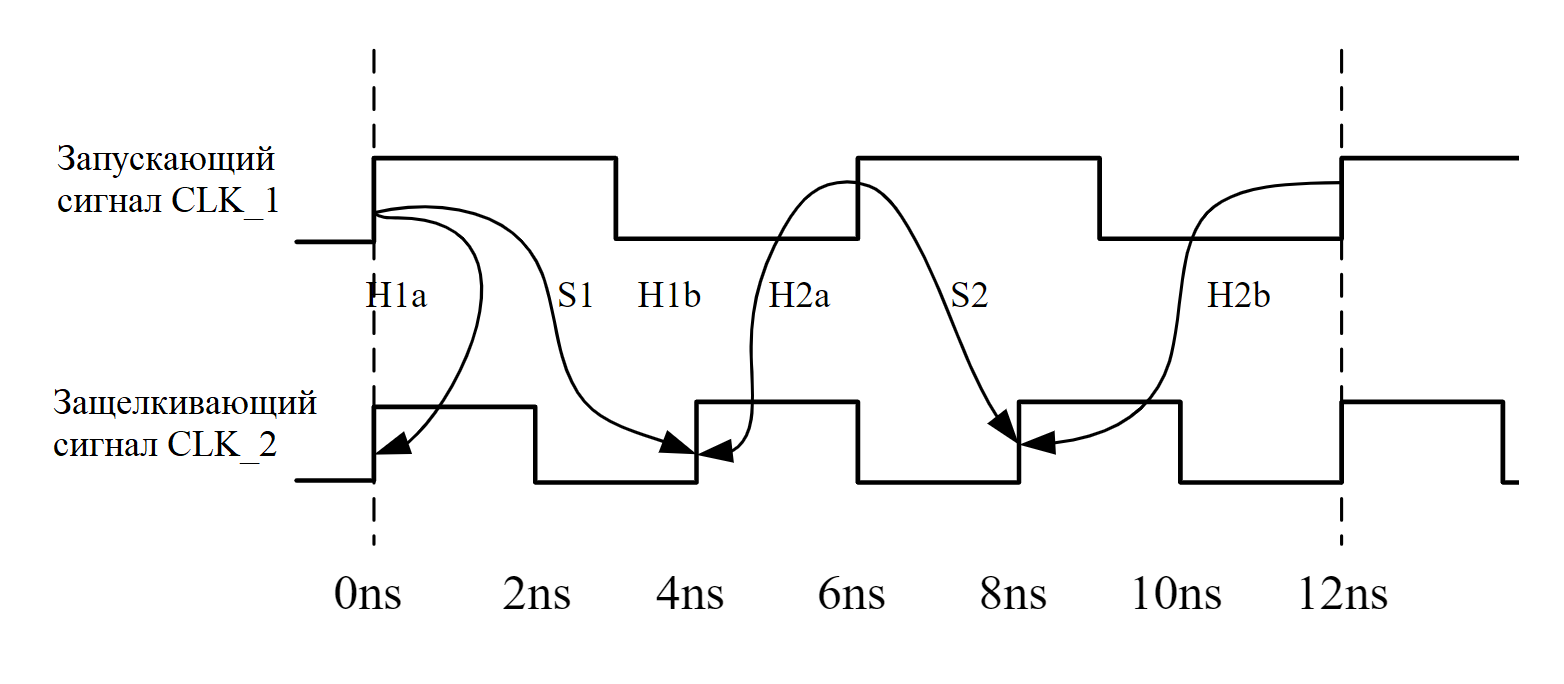


Рисунок 3. Временная диаграмма сигналов для анализа по *Hold*.

Аналогичным образом проверяются условия для второй пары фронтов S2, полученной при анализе по *Setup*. На рисунке 3 они обозначены стрелкам H2a (она же H1b) и H2b. На интервале времени общего периода для каждого условия по формуле (2) вычисляется значение *ΔTh*. В нашем примере для условия H1a величина *ΔTh* равна 0 – 0 = 0 нс. Для H2a и H1b получаем, что *ΔTh* = 4 – 6 = –2 нс, а для H2b – *ΔTh* = 8 – 12 = –4 нс. Самому пессимистичному случаю при анализе по *Hold* соответствует максимальное значение *ΔTh*, так как в уравнение для *Slack* это слагаемое входит с отрицательным знаком. Для нашего примера максимальное значение *ΔTh* равно нулю, что соответствует условию H1a (см. рисунок 3).

Чтобы временной анализатор мог провести свои расчеты, ему необходимо указать периоды и начальные фазы тактовых сигналов. Зная эти параметры, он вычислит общий период и найдет минимальное значение *ΔTsu* и максимальное значение *ΔTh.*

1. **Асинхронные тактовые сигналы.**

Как было показано выше, для проведения временного анализа при пересечении доменов необходимо знать точные временные соотношения между тактовыми сигналами. Однако, это возможно далеко не всегда. Например, если тактовые сигналы формируются из двух разных генераторов, то их начальные фазы и положения фронтов друг относительно друга неизвестны.

С точки зрения временных соотношений пары тактовых сигналов можно классифицировать следующим образом [3]:

* асинхронные – формируются разными генераторам;
* мезохронные – формируются из одного генератора, но в процессе распространения из-за различных факторов их временные соотношения становятся неизвестными;
* синхронные – формируются из одного генератора, и их временные соотношения точно известны.

Мезохронными, например, можно считать тактовые сигналы, которые формируются одним внешним генератором и поступают на разные тактовые ножки FPGA. При распространении по дорожкам платы в задержки этих сигналов может вноситься неопределенность, например, из-за неоднородности показателя диэлектрической проницаемости подложки. Это в свою очередь приводит к неопределенности положения фронтов тактовых сигналов друг относительно друга.

Так как для асинхронных и мезохронных сигналов невозможно точно рассчитать значения *ΔTsu* и *ΔTh*, временной анализ имеет смысл проводить только для синхронных тактовых сигналов. С точки зрения Vivado сигналы считаются синхронными, если они формируются внутри FPGA из одного источника, например, с помощью MMCM или PLL.

Напомним, для чего вообще проводится временной анализ. У триггеров есть время установки (*Tsu*) и время удержания (*Th*). Чтобы данные были корректно приняты, они должны быть стабильны на входе триггера в течение времени *Tsu* до появления защелкивающего фронта и в течение времени *Th* после него. Если эти условия не будут выполнены, то триггер может попасть в метастабильное состояние: некоторое промежуточное положение неустойчивого равновесия между логическим нулем и единицей. Рано или поздно триггер выйдет из метастабильности, однако, невозможно заранее предсказать, будет ли итоговое состояние логическим нулем или единицей. Также случайным является длительность временного интервала, в течение которого триггер будет находиться в метастабильном состоянии.

Временной анализ необходим, чтобы проверить выполнение ограничений для времени установки и удержания и убедиться, что защелкивающий триггер не попадет в метастабильное состояние. Для асинхронных и мезохронных тактовых сигналов невозможно корректно провести временной анализ и гарантировать отсутствие нарушений. Поэтому в таких случаях между двумя доменами нужно ставить дополнительные синхронизаторы, защищающие от метастабильности.

Рассмотрим пример передачи данных между двумя асинхронными тактовыми доменами на практике. Для простоты пусть каждый домен состоит всего из одного триггера. Схема проекта показана на рисунке 4. Описание на SystemVerilog представлено ниже:

module top\_1 (

    input  logic clk\_1,

    input  logic clk\_2,

    input  logic i\_data,

    output logic o\_data

);

    logic cdc\_data;

    // передающий тактовый домен

    always\_ff @(posedge clk\_1)

        cdc\_data <= i\_data;

    // принимающий тактовый домен

    always\_ff @(posedge clk\_2)

        o\_data <= cdc\_data;

endmodule

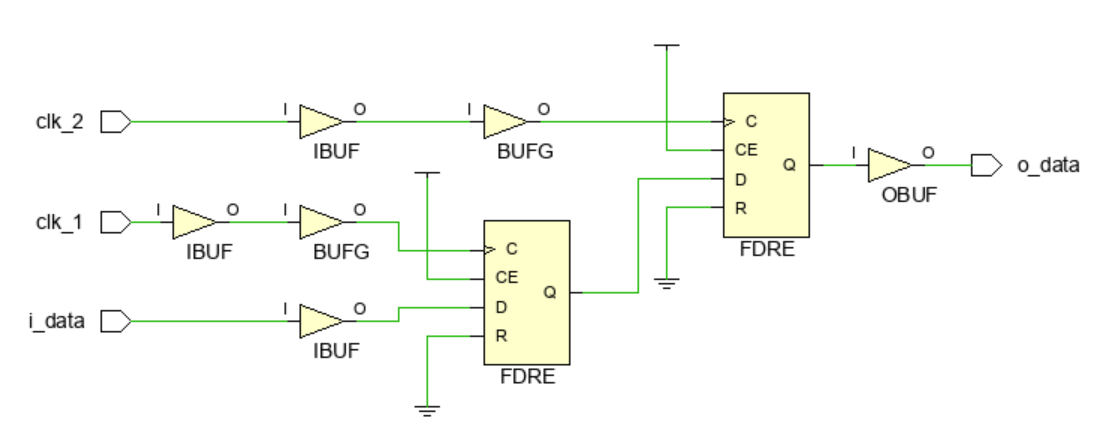


Рисунок 4. Схема проекта.

Чтобы Vivado смог провести временной анализ, ему необходимо указать периоды и начальные фазы тактовых сигналов. Пусть период запускающего тактового сигнала равен 10 нс, а защелкивающего – 5 нс. Занесем в xdc-файл следующие команды [1]:

# период запускающего тактового сигнала

create\_clock -period 10.0 -name clk\_1 [get\_ports clk\_1]

# период защелкивающего тактового сигнала

create\_clock -period 5.0 -name clk\_2 [get\_ports clk\_2]

По умолчанию при создании ограничений на период тактового сигнала считается, что его фронт формируется в нулевой момент времени. Чтобы это изменить, в команду *create\_clock* нужно добавить ключ *-waveform* [4].

Проверить корректность проведения временного анализа при пересечении тактовых доменом можно с помощью отчета, который называется *Clock Interaction*. Он представляет из себя таблицу, где по горизонтали отложены передающие тактовые сигналы, а по вертикали – принимающие. С помощью цвета указывается информация о временном анализе.

Для нашего примера отчет *Clock Interaction* будет иметь вид, представленный на рисунке 5. В проекте есть всего одни путь из домена clk\_1 в домен clk\_2. Тактовые сигналы формируются вне FPGA, поэтому Vivado считает их асинхронными. Это означает, что правильно провести временной анализ невозможно. В таблице клетка на пересечении clk\_1 и clk\_2 обозначена красным цветом.

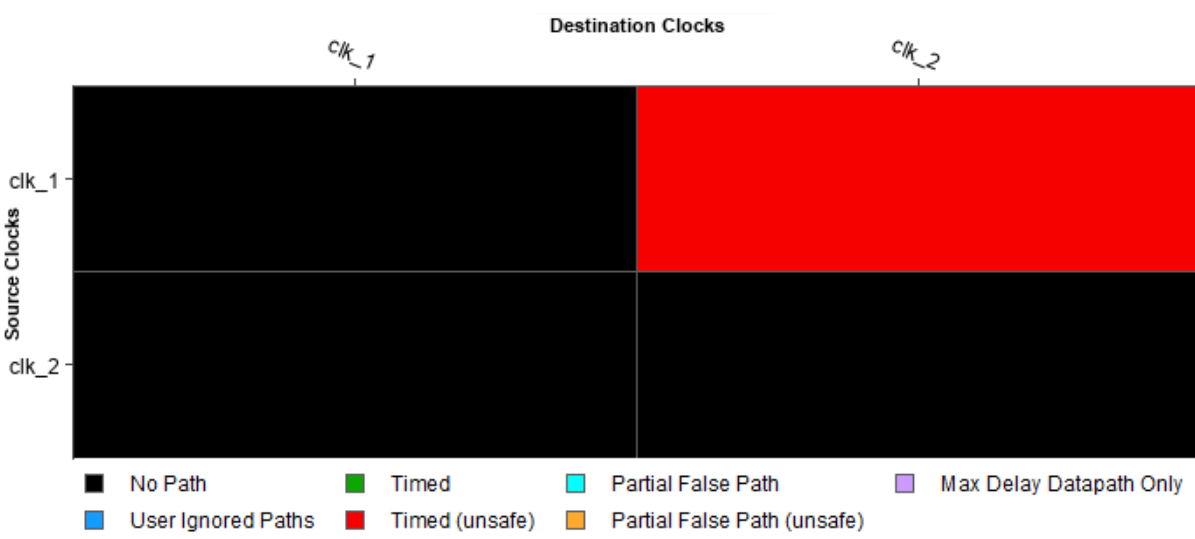


Рисунок 5. Таблица взаимодействия тактовых сигналов.

Данный отчет также можно получить в текстовой форме, если выполнить в TCL-консоли команду *report\_clock\_interaction*. Результат представлен ниже:

Clock Interaction Table

-----------------------

Clock-Pair Inter-Clock

From Clock To Clock Classification Constraints

------------ ------------ ------------------- --------------

clk\_1 clk\_2 No Common Clock Timed (unsafe)

Можно увидеть, что для пути между доменами временной анализ проводится (*Timed*), но его результатам не следует доверять (*unsafe*). Причиной является асинхронность тактовых сигналов, так как они формируются из разных источников (*No Common Clock*).

Теперь рассмотрим случай, когда тактовые сигналы синхронные. Для этого добавим в проект PLL. На рисунке 6 показаны настройки PLL, а на рисунке 7 – схема нового проекта. Описание проекта на SystemVerilog представлено ниже:

module top\_2 (

    input  logic i\_clk,

    input  logic i\_data,

    output logic o\_data

);

    logic clk\_1;

    logic clk\_2;

    logic cdc\_data;

    clk\_wiz\_0 pll

    (

        .i\_clk (i\_clk),

        .clk\_1 (clk\_1),

        .clk\_2 (clk\_2)

    );

    // передающий тактовый домен

    always\_ff @(posedge clk\_1)

        cdc\_data <= i\_data;

    // принимающий тактовый домен

    always\_ff @(posedge clk\_2)

        o\_data <= cdc\_data;

endmodule

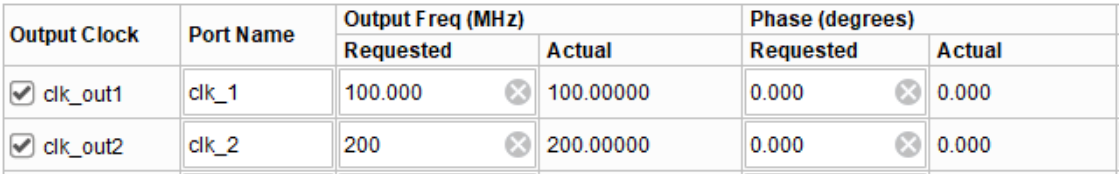


Рисунок 6. Настройки PLL.

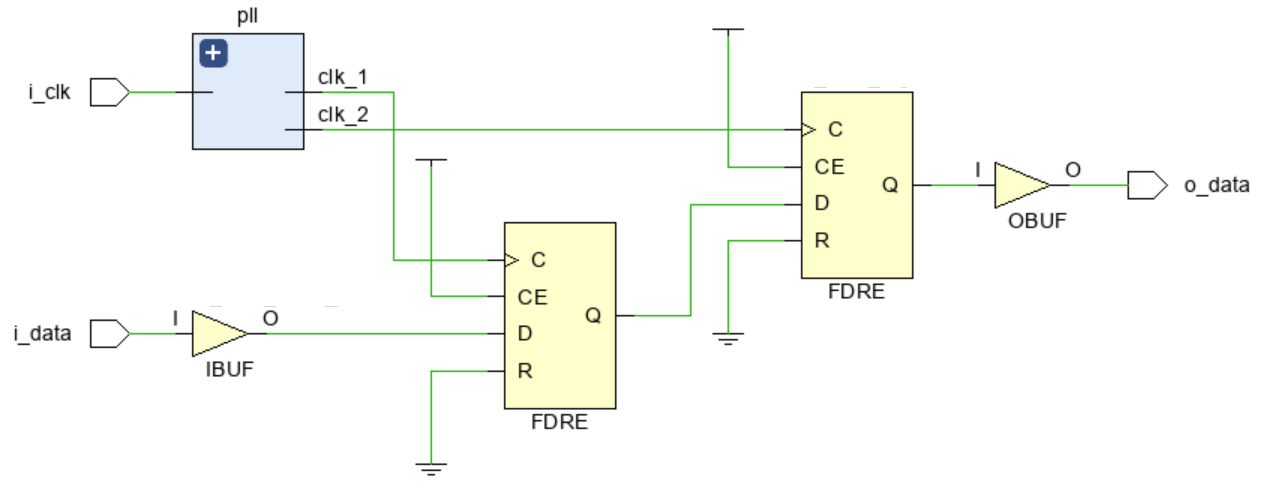


Рисунок 7. Схема проекта.

На ножку FPGA приходит тактовый сигнал i\_clk, из которого с помощью PLL формируются два сигнала clk\_1 и clk\_2. Добавим в xdc-файл следующие команды:

# период входного тактового сигнала

create\_clock -period 10.0 -name i\_clk [get\_ports i\_clk]

# объявление переменных, которые указывают места формирования

# тактовых сигналов

set pll\_input    [get\_pins pll/inst/mmcm\_adv\_inst/CLKIN1]

set pll\_output\_1 [get\_pins pll/inst/mmcm\_adv\_inst/CLKOUT0]

set pll\_output\_2 [get\_pins pll/inst/mmcm\_adv\_inst/CLKOUT1]

# обновление имен тактовых сигналов, сгенерированных с помощью PLL

create\_generated\_clock -name clk\_1 -source $pll\_input $pll\_output\_1

create\_generated\_clock -name clk\_2 -source $pll\_input $pll\_output\_2

Временному анализатору с помощью команды *create\_clock* необходимо указать период сигнала i\_clk. Зная его, а также учитывая настройки PLL, анализатор самостоятельно определит периоды сигналов clk\_1 и clk\_2. Некоторое неудобство может доставлять то, что для сигналов clk\_1 и clk\_2 анализатор сгенерирует очень длинные и трудночитаемые имена. Чтобы их изменить, можно с помощью команды *set* объявить переменные, указывающие места формирования тактовых сигналов. Далее, используя команду *create\_generated\_clock* с ключом *-name*, сигналам нужно присвоить более простые имена. В нашем случае просто clk\_1 и clk\_2.

На рисунке 8 представлен отчет *Clock Interaction*. Можно увидеть, что теперь ячейка, соответствующая пути из домена clk\_1 в домен clk\_2, выделена зеленым цветом.

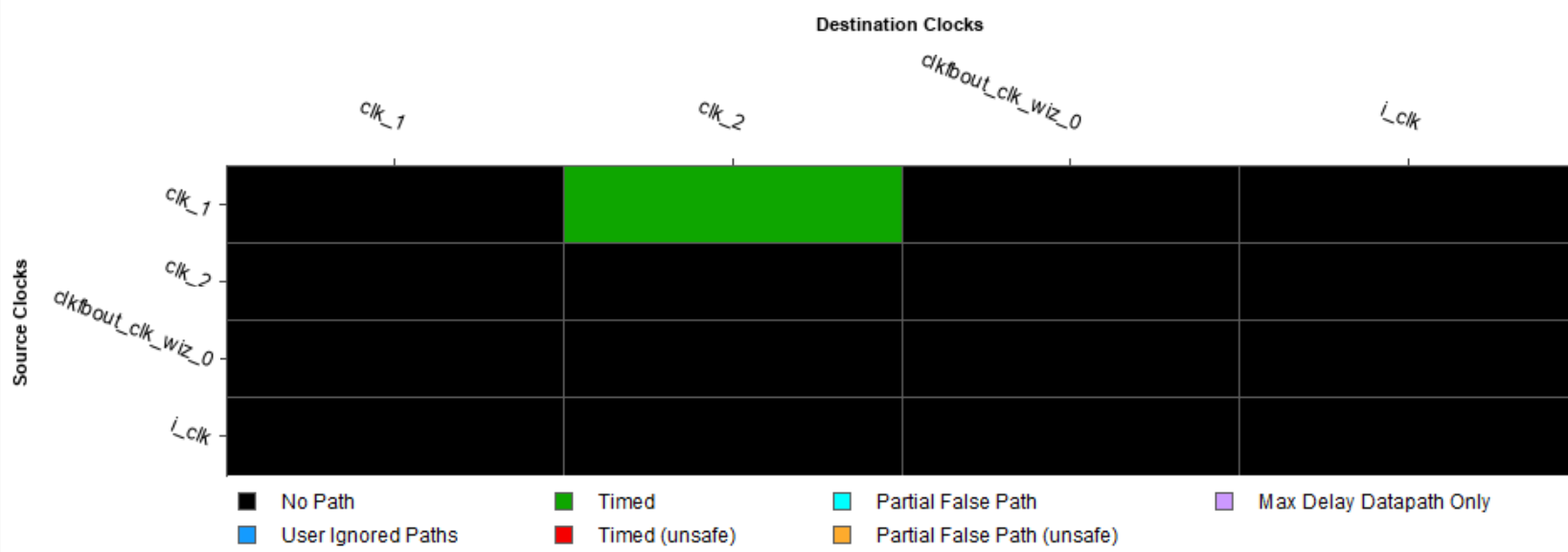


Рисунок 8. Таблица взаимодействия тактовых сигналов.

Ниже представлен отчет *Clock Interaction* в текстовом виде:

Clock Interaction Table

-----------------------

Clock-Pair Inter-Clock

From Clock To Clock Classification Constraints

------------ ------------ ------------------- -----------

clk\_1 clk\_2 Clean Timed

Временной анализ выполняется и теперь считается корректным (*Timed*), так как все временные соотношения между тактовыми сигналами точно известны (*Clean*).

1. **False Path Constrains.**

Иногда наличие в проекте асинхронных тактовых сигналов, для которых выполняется временной анализ, может привести к проблемам. Опять рассмотрим проект, который состоит всего из двух триггеров (см. рисунок 4). В xdc-файл добавим команды, указывающие, что периоды сигналов clk\_1 и clk\_2 равны 10 и 5 нс соответственно:

# период запускающего тактового сигнала

create\_clock -period 10.0 -name clk\_1 [get\_ports clk\_1]

# период защелкивающего тактового сигнала

create\_clock -period 5.0 -name clk\_2 [get\_ports clk\_2]

На рисунке 9 показан раздел *Summary* временного отчета для пути между тактовыми доменами при анализе по *Setup*. В строке *Requirements* можно увидеть, что запускающий фронт сигнала clk\_1 формируется в нулевой момент времени. Через 5 нс после это появляется защелкивающий фронт сигнала clk\_2. Общий период равен 10 нс. В течение этого времени наблюдается всего один фронт clk\_1. Таким образом анализатор находит, что значение *ΔTsu* равно 5 нс.

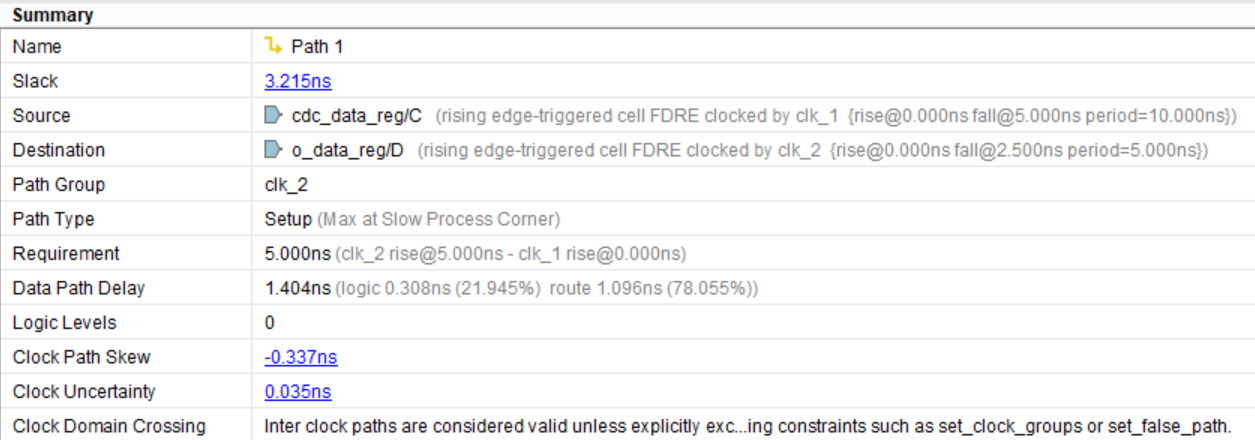


Рисунок 9. Раздел *Summary* временного отчета.

Хотя это и не имеет смысла, но анализатор все равно вычисляет все задержки и рассчитывает значение *Slack*. Величина запаса больше нуля и равна 3.215 нс, то есть временные ограничения как будто выполнены.

Далее, изменим период сигнала clk\_2 и сделаем его равным 33 нс. Файл с ограничениями имеет вид:

# период запускающего тактового сигнала

create\_clock -period 10.0 -name clk\_1 [get\_ports clk\_1]

# период защелкивающего тактового сигнала

create\_clock -period 33.3 -name clk\_2 [get\_ports clk\_2]

Раздел *Summary* временного отчета для нового периода clk\_2 показан на рисунке 10. Теперь *Slack* имеет отрицательное значение, то есть временные ограничения не выполнены. Давайте найдем причину.

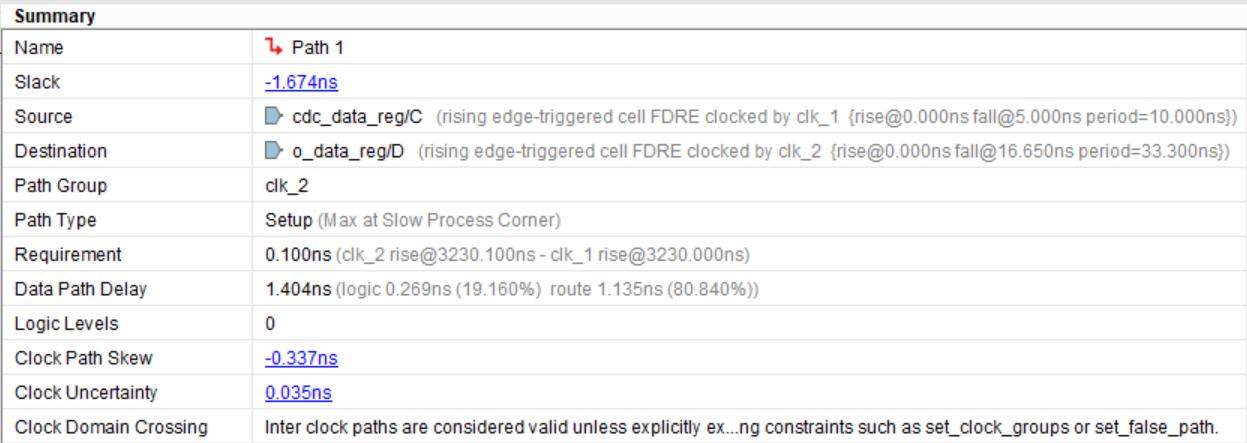


Рисунок 10. Раздел *Summary* временного отчета.

В строке *Requirements* указано минимальное значение *ΔTsu*,равное 0.1 нс. Это соответствует случаю, когда 323-ий запускающий фронт сигнала clk\_1 появляется в момент времени 3230 нс, а 97-ой защелкивающий фронт сигнала clk\_2 – в момент времени 3230.1 нс. Данные от одного триггера к другому должны быть переданы всего за 0.1 нс, что является трудновыполнимой задачей. В соответствии с расчетами данные опаздывают на  
целых -1.674 нс.

Мы знаем, что сигналы асинхронные, поэтому отчет не является корректным, и на это нарушение можно не обращать внимания. Однако, в процессе размещения и трассировки Vivado будет стремиться выполнить все ограничения, причем самые сложные, такие как *ΔTsu* = 0.1 нс, рассматриваются в первую очередь. Из-за этого возможна ситуация, когда Vivado приложит все усилия, чтобы выполнить временные ограничения для одного критического пути за счет наличия небольших нарушений для других путей. То есть, в проекте появятся нарушения, которых могло бы и не быть, если бы путь между асинхронными доменами не участвовал во временном анализе.

Чтобы исключить путь из временного анализа, можно воспользоваться командой *set\_false\_path*. Флаг *-from* указывает начало пути, а флаг *-to* – его конец. Напомним, что путь всегда начинается на тактовом входе запускающего триггера и заканчивается на одном из входов защелкивающего триггера. Для нашего примера в xdc-файл нужно добавить следующую команду:

# объявление false path через начало и конец пути

set\_false\_path -from [get\_pins cdc\_data\_reg/C]

-to [get\_pins o\_data\_reg/D]

С помощью команды *get\_pins* получаем тактовый вход запускающего (*cdc\_data\_reg/C*) и информационный вход защелкивающего (*o\_data\_reg/D*) триггеров.

Рассмотрим, как изменился раздел *Summary* временного отчета   
(см. рисунок 11). Можно увидеть, что теперь значение *Slack* и требуемое время передачи данных (*Requirement*) равны бесконечности. Также в отчете появилась дополнительная строка *Timing Exception*, указывающая, что путь был исключен из анализа с помощью команды *false path*.

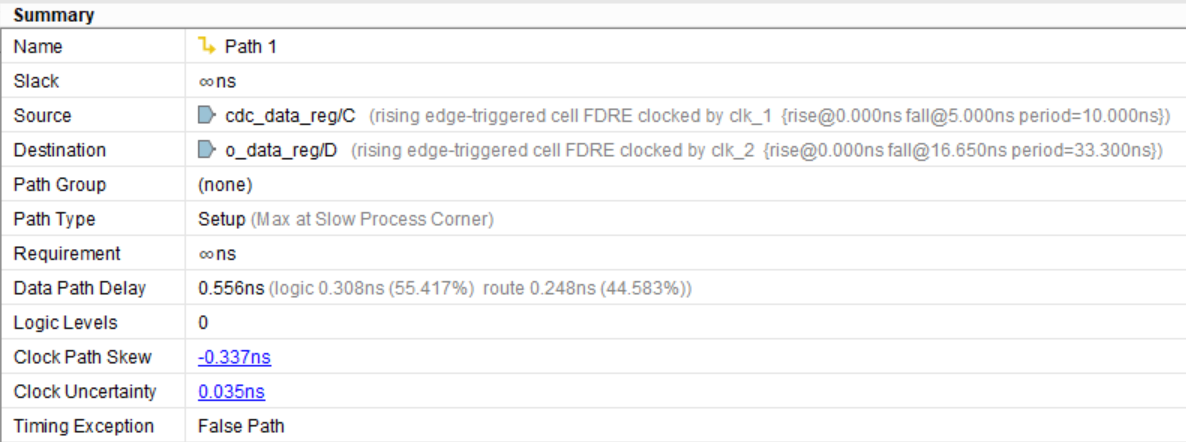


Рисунок 11. Раздел *Summary* временного отчета.

Теперь изучим, как это отразилось на отчете *Clock Interaction*   
(см. рисунок 12). Ячейка таблицы, соответствующая пути из домена clk\_1 в домен clk\_2, помечена синим цветом. Это означает, что разработчик проекта вручную исключил пути из временного анализа (*User Ignored Paths*).

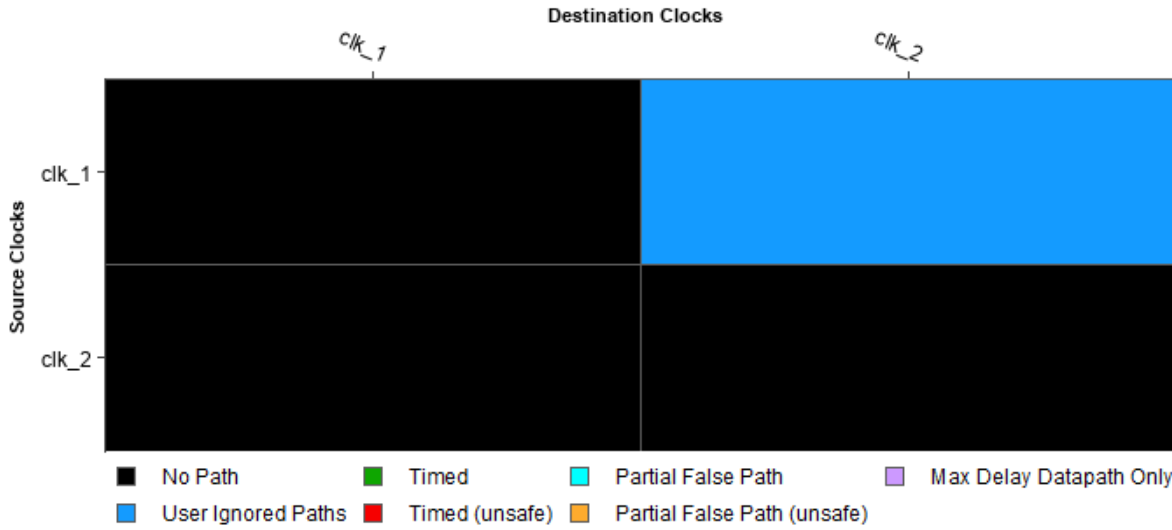


Рисунок 12. Таблица взаимодействия тактовых сигналов.

Ту же информацию можно получить в текстовой форме:

Clock Interaction Table

-----------------------

Clock-Pair Inter-Clock

From Clock To Clock Classification Constraints

------------ ------------ ------------------- -----------

clk\_1 clk\_2 Ignored False Path

Далее, рассмотрим еще несколько способов исключения пути из временного анализа. В качестве начала и конца пути для команды *set\_false\_path* можно просто задать запускающий и защелкивающий триггеры.

# объявление false path через запускающий и защелкивающий триггеры

set\_false\_path -from [get\_cells cdc\_data\_reg]

-to [get\_cells o\_data\_reg]

В этом случае будут исключены пути, идущие от входа C триггера *cdc\_data\_reg* до всех входов триггера *o\_data\_reg* (входы D, CE и другие). Также можно исключить сразу все пути, идущие из тактового домена clk\_1 в домен clk\_2. Для этого нужно указать тактовые сигналы с помощью команды *get\_clocks*:

# объявление false path через имена тактовых доменов

set\_false\_path -from [get\_clocks clk\_1] -to [get\_clocks clk\_2]

Еще один способ заключается в объявлении тактовых доменов асинхронными. Для этого нужно с помощью команды *set\_clock\_groups* определить группы тактовых сигналов и добавить для них флаг *-asynchronous*. В этом случае из анализа будут исключаться все пути из домена clk\_1 в домен clk\_2, а также из домен clk\_2 в домен clk\_1. Флаг *-name* задает имя, по которому можно в дальнейшем обращаться к асинхронным группам.

# объявление тактовых доменов асинхронными

set\_clock\_groups -name cdc\_async -group [get\_clocks clk\_1]

-group [get\_clocks clk\_2] -asynchronous

То, что домены являются асинхронными, также отражается в отчете *Clock Interaction*:

Clock Interaction Table

-----------------------

Clock-Pair Inter-Clock

From Clock To Clock Classification Constraints

------------ ------------ ------------------- -------------------

clk\_1 clk\_2 Ignored Asynchronous Groups

1. **Отчет о CDC.**

Применение *false path* и асинхронных групп может защитить от появления ложных нарушений, но не гарантирует, что между тактовыми доменами данные пересылаются корректно. Правильность передачи данных между доменами можно проверить с помощью отчета о CDC (Clock Domain Cross). Для этого в TCL-консоли Vivado нужно выполнить команду *report\_cdc*. Ее результат для примера из рисунка 4 представлен ниже:

CDC Report

Severity Source Clock Destination Clock Exceptions

-------- ------------ ----------------- -------------------

Critical clk\_1 clk\_2 Asynch Clock Groups

Endpoints Safe Unsafe Unknown No ASYNC\_REG

--------- ---- ------ ------- ------------

1 0 0 1 0

Из отчета следует, что установить корректность передачи данных из домена clk\_1 в домен clk\_2 невозможно (*Unknown* = 1). Это приводит к наличию в проекте критического замечания (*Severity* = *Critical*). Более подробно о причине замечания можно узнать в отчете *Report Methodology*(см. вкладку Vivado Flow Navigator). Один из его разделов представлен ниже:

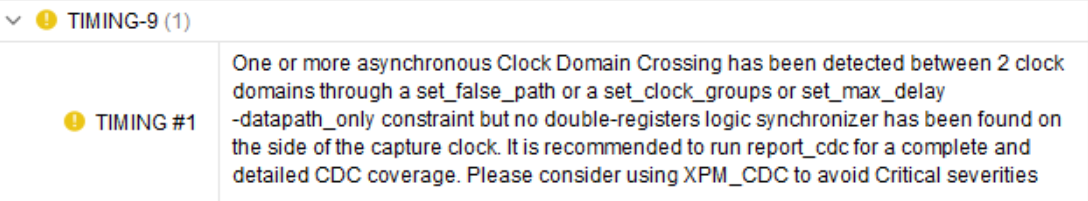


Рисунок 13. Отчет о проверке методологии проектирования.

В нашем проекте есть путь, который пересекает тактовые домены и исключен из временного анализа с помощью команды *set\_false\_path* или *set\_clock\_groups*. Чтобы гарантировать правильность передачи между асинхронными доменами, данные должны проходить через специальные синхронизаторы, например, через сдвоенные триггеры (double-register logic). При формировании отчета о CDC Vivado просматривает netlist и пытается найти между доменами синхронизаторы. В случае их отсутствия выводится соответствующее замечание. Все структуры в netlist, которые Vivado распознает как синхронизаторы, можно посмотреть в [2] в главе 2.

Чтобы получить синхронизатор на основе сдвоенных триггеров, добавим в домен clk\_2 еще одни триггер *sync\_data\_reg*. Схема обновленного проекта (см. рисунок 14) и его описание на SystemVerilog представлены ниже:

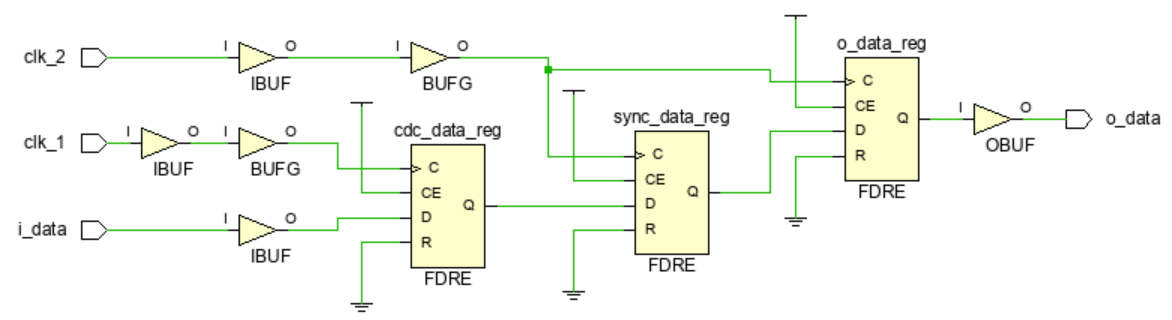


Рисунок 14. Схема проекта.

module top\_3 (

    input  logic clk\_1,

    input  logic clk\_2,

    input  logic i\_data,

    output logic o\_data

);

    logic cdc\_data, sync\_data;

    // передающий тактовый домен

    always\_ff @(posedge clk\_1)

        cdc\_data <= i\_data;

    // принимающий тактовый домен

    always\_ff @(posedge clk\_2) begin

        sync\_data <= cdc\_data;

        o\_data <= sync\_data;

    end

endmodule

Кратко разберем принцип работы синхронизатора. Так как домены clk\_1 и clk\_2 являются асинхронными, временные соотношения между их тактовыми сигналами неизвестны. Это означает, что при передаче данных для триггера *sync\_data\_reg* могут возникать нарушения времени установки и удержания, и он будет попадать в метастабильное состояние. В случае метастабильности сигнал на выходе *sync\_data\_reg* примет некоторый неопределенный уровень между нулем и единицей. Этот сигнал будет распространяться дальше по схеме и в итоге попадет на вход триггера *o\_data\_reg*.

Если теперь на триггер *o\_data\_reg* подать тактовый фронт, то он защелкнет неопределенный уровень и сам перейдет в метастабильное состояние. Однако, как было сказано ранее, метастабильность – это состояние неустойчивое. Рано или поздно триггер *sync\_data\_reg* из него выйдет, и на его выходе опять появится нормальный логический уровень. Если это произойдет до появления следующего тактового фронта, то триггер *o\_data\_reg* не увидит сигнал неопределенного уровня и не перейдет в метастабильное состояние.

Таким образом, триггер *sync\_data\_reg* может попасть в метастабильное состояние, но, если он быстро из него выйдет, то метастабильность не появится на выходе *o\_data\_reg* и будет локализована внутри синхронизатора.

Рассмотрим, как изменился отчет о CDC после добавления синхронизатора. Теперь данные между доменами передаются корректно  
 (*Safe* = 1), однако, к проекту по-прежнему есть замечания (*Severity* = *Warning*).

CDC Report

Severity Source Clock Destination Clock Exceptions

-------- ------------ ----------------- -------------------

Warning clk\_1 clk\_2 Asynch Clock Groups

Endpoints Safe Unsafe Unknown No ASYNC\_REG

--------- ---- ------ ------- ------------

1 1 0 0 1

1. **Свойство ASYNC\_REG.**

Чтобы понять причину оставшегося замечания, опять рассмотрим отчет *Report Methodology*. Его результаты представлены ниже:

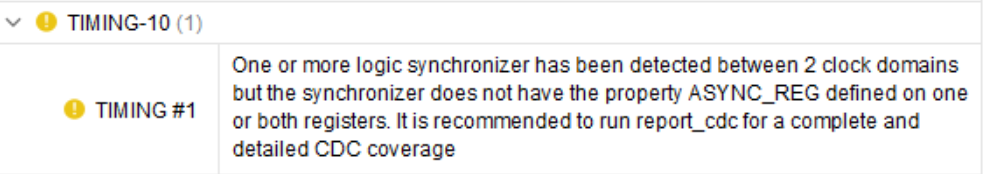


Рисунок 15. Отчет о проверке методологии проектирования.

Между тактовыми доменами данные передаются с помощью синхронизатора, но триггеры, входящие в его состав, не имеют свойства ASYNC\_REG. Эту же информацию можно получить из последнего отчета   
о CDC (*No ASYNC\_REG* = 1). Чтобы понять назначение этого свойства, вернемся к схеме проекта на рисунке 14.

Как было рассказано ранее, триггер *sync\_data\_reg* может становиться метастабильным. При этом, если он вернется в нормальное состояние до появления следующего тактового фронта, то триггер *o\_data\_reg* эту метастабильность не увидит. Однако, триггер *sync\_data\_reg* не просто должен успеть выйти из метастабильности, но и его выходной сигнал с нормальным логическим уровнем должен успеть распространиться до входа *o\_data\_reg*. Время выхода из метастабильного состояния – это величина случайная. Поэтому вероятность того, что на следующем фронте триггер *o\_data\_reg* увидит нормальный уровень сигнала, будет выше, если время передачи данных между триггерами будет как можно меньше.

Свойство ASYNC\_REG указывает Vivado, что триггер входит в состав синхронизатора. Чтобы уменьшить вероятность появления метастабильности на выходе синхронизатора, в процессе размещения и трассировки Vivado будет стараться сократить длину пути между триггерами со свойством ASYNC\_REG и располагать их как можно ближе друг к другу. Также свойство ASYNC\_REG запрещает проводить для таких триггеров некоторые виды оптимизации, например, retiming [5].

Добавим триггерам *sync\_data\_reg* и *o\_data\_reg* из предыдущего примера свойство ASYNC\_REG. Полное содержимое xdc-файла представлено ниже:

# период запускающего тактового сигнала

create\_clock -period 10.0 -name clk\_1 [get\_ports clk\_1]

# период защелкивающего тактового сигнала

create\_clock -period 33.3 -name clk\_2 [get\_ports clk\_2]

# объявление тактовых доменов асинхронными

set\_clock\_groups -name cdc\_async -group [get\_clocks clk\_1]

-group [get\_clocks clk\_2] -asynchronous

# объявление синхронизирующих триггеров

set\_property ASYNC\_REG true [get\_cells {sync\_data\_reg o\_data\_reg}]

Теперь триггеры, которые входят в состав синхронизатора, помечены свойством ASYNC\_REG, что указано в отчете о CDC как   
*No ASYNC\_REG* = 0. К проекту больше нет замечаний (*Severity* = *Info*).

CDC Report

Severity Source Clock Destination Clock Exceptions

-------- ------------ ----------------- -------------------

Info clk\_1 clk\_2 Asynch Clock Groups

Endpoints Safe Unsafe Unknown No ASYNC\_REG

--------- ---- ------ ------- ------------

1 1 0 0 0

В заключении отметим, что в Vivado Language Temples в разделе CDC есть готовые параметризируемые синхронизаторы. Модуль, описывающий синхронизатор из цепочки последовательно соединенных триггеров, называется *xpm\_cdc\_single*. Давайте попробуем его использовать. Ниже представлена схема проекта (см. рисунок 16) и его описание:

module top\_4 (

    input  logic clk\_1,

    input  logic clk\_2,

    input  logic i\_data,

    output logic o\_data

);

    logic cdc\_data, sync\_data;

    // передающий тактовый домен

    always\_ff @(posedge clk\_1)

        cdc\_data <= i\_data;

    // XPM синхронизатор

    xpm\_cdc\_single #(

        .DEST\_SYNC\_FF(2),

        .SRC\_INPUT\_REG(0)

    )

    xpm\_synchronizer (

        .src\_clk(clk\_1),

        .src\_in(cdc\_data),

        .dest\_clk(clk\_2),

        .dest\_out(sync\_data)

    );

    // принимающий тактовый домен

    always\_ff @(posedge clk\_2)

        o\_data <= sync\_data;

endmodule

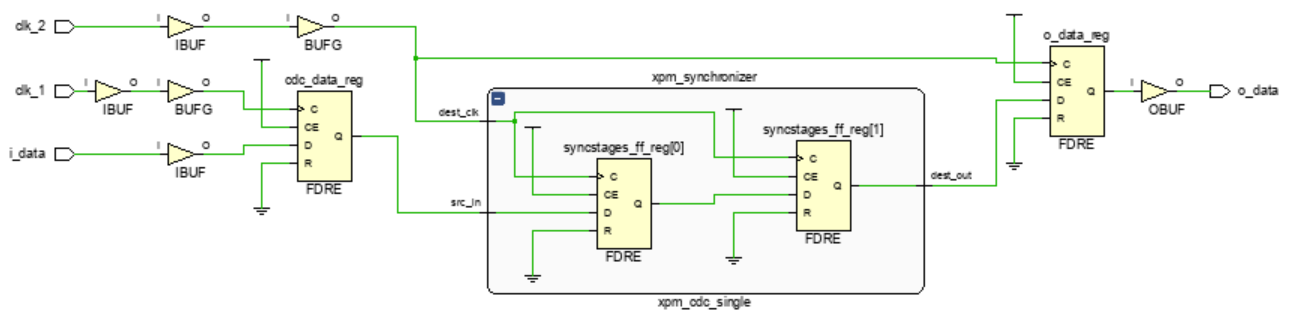


Рисунок 16. Схема проекта.

У синхронизатора есть несколько настроек. Так параметр SRC\_INPUT\_REG указывает требуется (1) или нет (0) поместить дополнительный триггер в передающий тактовый домен. С помощью параметра DEST\_SYNC\_FF можно задать количество триггеров в принимающем домене. Минимально возможное значение этого параметра равно двум.

В xdc-файл добавим только ограничения на периоды тактовых сигналов:

# период запускающего тактового сигнала

create\_clock -period 10.0 -name clk\_1 [get\_ports clk\_1]

# период защелкивающего тактового сигнала

create\_clock -period 33.3 -name clk\_2 [get\_ports clk\_2]

Из отчета о CDC можно увидеть, что при использовании готового синхронизатора путь между тактовыми доменами исключается из временного анализа (*Exceptions* = *False Path*), а триггерам синхронизатора автоматически присваивается свойство ASYNC\_REG.

CDC Report

Severity Source Clock Destination Clock Exceptions

-------- ------------ ----------------- -------------------

Info clk\_1 clk\_2 False Path

Endpoints Safe Unsafe Unknown No ASYNC\_REG

--------- ---- ------ ------- ------------

1 1 0 0 0

Использование синхронизаторов из Vivado Language Temples упрощает процесс создания проекта и существенно снижает количество ошибок при описании временных ограничений.

**Заключение.**

В статье представлен временной анализ при передаче данных между тактовыми доменами. Рассказано о возможных проблемах, возникающих   
из-за нарушения временных ограничений для асинхронных тактовых сигналов. Рассмотрены несколько способов исключения пути из временного анализа. Кратко описан принцип работы синхронизатора на основе сдвоенных триггеров.

**Ссылки.**

1. [Основы статического временного анализа. Часть 1: Period Constraint](https://fpga-systems.ru/static-timing-analysis-part-1-period-constraints)
2. [Vivado Design Analysis and Closure Techniques (UG 906)](https://docs.xilinx.com/v/u/2017.3-English/ug906-vivado-design-analysis)
3. [Xilinx Support Forum](https://support.xilinx.com/s/question/0D52E00006mibS0SAI/constraining-2-external-clocks-that-are-synchronous?language=en_US)
4. [Vivado Using Constraints (UG 903)](https://docs.xilinx.com/v/u/2013.1-English/ug903-vivado-using-constraints)
5. [Vivado Synthesis (UG 901)](https://docs.xilinx.com/v/u/2021.1-English/ug901-vivado-synthesis)