**Основы статического временного анализа.**

**Часть 5: False Path Constraint.**

**Введение.**

В данной статье представлен временной анализ передачи сигналов между двумя тактовыми доменами. Рассмотрен способы исключения пути из временного анализа. Показаны инструменты Vivado, предназначенные для анализа правильности передачи данных между тактовыми доменами.

1. **Пересечение тактовых доменов.**

Данная статья частично опирается на материал, рассмотренный в предыдущих работах [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.

В предыдущих работах серии был представлен временной анализ передачи данных между двумя триггерами, которые тактировались одним и тем же тактовым сигналом. В данной статье будет рассмотрен случай, когда запускающий и защелкивающий фронты относятся к разным тактовым сигналам.

Множество триггеров и других последовательных устройств, которые тактируются одним и тем же сигналом называют тактовым доменом. Пересечением тактовых доменов (Clock Domain Cross - CDC) называется ситуация, когда передающий триггер находится в одном домене, а принимающий триггер – в другом.

Рассмотрим рисунок 1, на котором показан анализируемый путь, на который нанесены задержки сигналов. Передающий триггер FF1 тактируется сигналом CLK\_1, а приёмный триггер FF2 – сигналом CLK\_2.

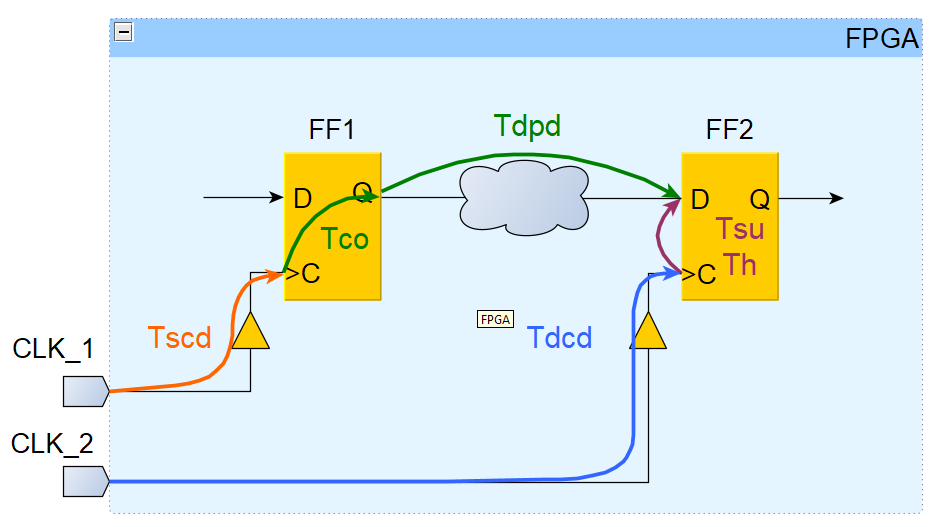


Рисунок 1. Путь с задержками для данных и тактового сигнала.

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 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 в момент времени *Tclk\_1\_su*, тогда, уравнения для расчета фактического времени прибытия данных можно записать в виде (см. рисунок 1):

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

Также считая, что защелкивающий фронт появляется на ножке CLK\_2 в момент времени *Tclk\_2\_su*, запишем 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:

Подставив в него, полученные ранее результаты, запишем уравнение для Slack в виде:

где *ΔTsu* – интервал времени между появлением запускающего фронта сигнала CLK\_1 и защелкивающего фронта сигнала CLK\_2 при анализе по *Setup*:

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*. При анализе по *Hold* в качестве защелкивающего фронта рассматривается фронт, предназначенный для приема триггером FF2 предыдущих данных. Будем считать, что этот фронт появляется на ножке CLK\_2 в момент времени *Tclk\_2\_h*. Для начала запишем 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 FF1(***D****ata* ***A****rrival time*):

Далее представлены уравнения для вычисления 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:

Используя полученные выше результаты, выражение для *Slack* можно представить в виде:

где *ΔTh* – интервал времени между появлением запускающего и защелкивающего фронта при анализе по *Hold*:

В полученных уравнениях для *Slack* единственные слагаемые, значения которых не известны временному анализатору после размещения и трассировки проекта – это *ΔTsu* и *ΔTh*. Эти слагаемые рассчитываются исходя из периодов и фазовых соотношений между тактовыми сигналами CLK\_1 и CLK\_2. Для начала рассмотрим, как анализатор Vivado вычисляет *ΔTsu* [2].

Как и всегда временной анализ проводится для самого пессимистичного случай. При анализе по *Setup* этому соответствует минимальное значение слагаемого *ΔTsu*, так как оно входит в уравнение для *Slack* c положительным знаком. В качестве примера рассмотрим случай, когда период сигнал CLK\_1 равен 6 нс, а период CLK\_2 – 4 нс. Также будем считать, что фронт обоих тактовых сигналов формируется в нулевой момент времени. Временные диаграммы сигналов представлены на рисунке 2.

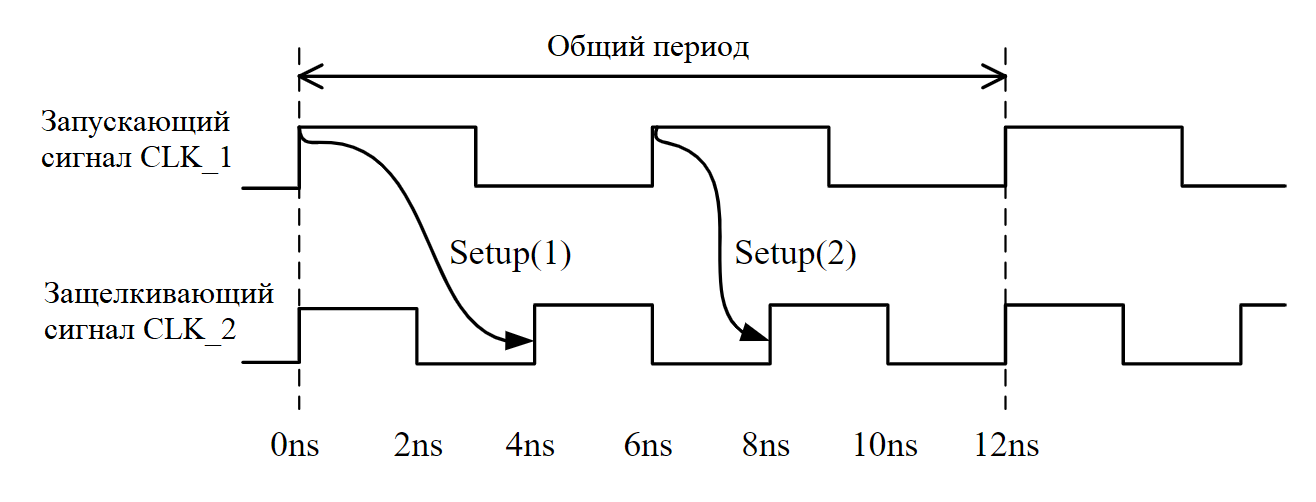


Рисунок 2. Временная диаграмма сигналов для анализа по *Setup*.

Временной анализатор поочередно рассматривает каждый фронт сигнала CLK\_1 в качестве запускающего фронта. То есть рассматриваются случаи, когда фронт CLK\_1 формируется в момент времени 0 нс, 6 нс, 12 нс и так далее. Далее каждому запускающего фронту CLK\_1 сопоставляется защелкивающий фронт сигнала CLK\_2. При этом ищется ближайший фронт CLK\_2, время формирования которого строго больше времени появления запускающего фронта.

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) соответствует третий фронт CLK\_2 (8 нс). Для каждой такой пары рассчитывается значение *ΔTsu*. В предыдущем примере для первого фронта CLK\_1 получим *ΔTsu* = 4 – 0 = 4 нс. Для второго фронта CLK\_1 находим, что *ΔTsu* = 8 – 6 = 2 нс.

Далее из рисунка 2 можно увидеть, что после момента времени 12 нс временная диаграмма сигналов будет повторяться, и для третьего фронта CLK\_1 опять получим *ΔTsu* = 4 нс. Интервал времени, через который будут повторяться фазовые 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*, которое в нашем примере равно 2 нс. Это значение в дальнейшем и будет использоваться в расчете *Slack* при анализе по *Setup*.

Также отметим, что возможно ситуация, при которой временной анализатор не обнаружит общего периода в течении 1000 тактов сигнала CLK\_1. В этом случае анализатор также остановится и будет использовать минимальное значение *ΔTsu*, обнаруженное в течении этих 1000 тактов.

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

* данные, которые передаются по запускающему фронту CLK\_1 не должны быть приняты ближайшим фронтом CLK\_2, появившимся раньше запускающего фронта;
* данные, которые передаются фронтом CLK\_1, следующим после запускающего, не должны быть приняты текущим защелкивающим фронтом CLK\_2.

Для примера рассмотрим рисунок 3. Первый пара фронтов полученная при анализе по *Setup* обозначена, как S1. Первое условие для S1 соответствует стрелке H1a, то есть данные, запушенные фронтом CLK\_1 в нулевой момент времени, не должны быть приняты фронтом CLK\_2, который также появляется в нулевой момент времени.

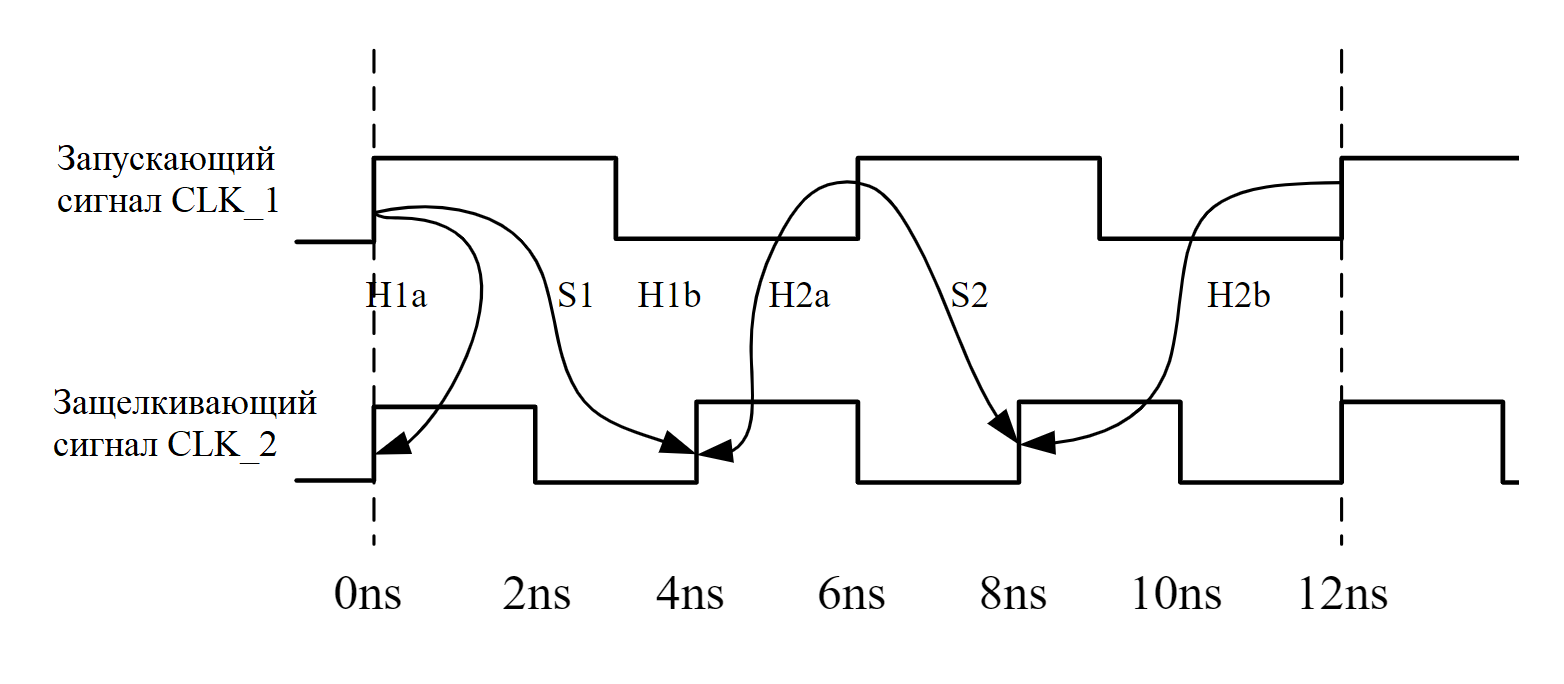


Рисунок 3. Временная диаграмма сигналов для анализа по *Hold*.

Второму условию соответствует стрелка H1b. Следующий фронт CLK\_1 для пары S1 появляется через 6 нс, и данные, которые он запустит, не должны быть приняты защелкивающим фронтом из S1, появляющимся через 4 нс.

Аналогичным образом условия проверяются для второй пары фронтов S2, что соответствует стрелка H2a (она же H1b) и H2b. Для каждой пары условий вычисляется значение *ΔTh*, например, для H1a она равно 0 – 0 = 0 нс. Для H2a и H1b получаем, что *ΔTh* = 4 – 6 = –2 нс, а для H2b – *ΔTh* = 8 – 12 = –4 нс.

Анализ по Hold также проводится для самого пессимистичного случая, которому соответствует максимальное значение *ΔTh*, так как в уравнение для *Slack* оно входит с отрицательным знаком. Для нашего примера получаем, что *ΔTh* = 0.

Таким образом, чтобы провести расчет временных ограничений анализатору необходимо указать периоды и начальные фазы тактовых сигналов. Зная эти параметры, анализатор вычислит общий период и найдет минимальной значение *ΔTsu* и максимальное значение *ΔTh.*

1. **Асинхронные тактовые сигналы.**

Как было показано выше, для проведения временного анализа при пересечении доменов необходимо знать точные фазовые 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 можно следующим образом классифицировать пары тактовых сигналов:

* асинхронные – формируются разными генераторам, причем генераторы не должны иметь общей внешней синхронизации.
* мезохронные – формируются из одного генератора, но из-за различных факторов в процессе распространении их фазовые соотношения становятся неизвестными.
* синхронные – формируются из одного генератора и их фазовые соотношения точно известны.

Мезохронными можно считать два тактовых сигнала, которые формируются одним внешним генератором и поступают на разные тактовые ножки FPGA. При распространении по дорожкам платы в фазы этих сигналов может вноситься неопределенность, например, из-за неоднородности показателя диеэлектрической проницаемости.

Временной анализ имеет смысл проводить только для синхронных тактовых сигналов. С точки зрения Vivado сигналы считаются синхронным, если они формируются внутри FPGA из одного источника, например, 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 попасть на вход триггера на *Tsu* раньше, чем появление защелкивающего фронта. Также они должны оставаться стабильными в течении времени *Th* после защелкивающего фронта. Если эти условия не будут выполнены, то триггер может попасть в метастабильное состояние: некоторое промежуточное состояние неустойчивого равновесия между логическим нулем и единицей. Из этого состояние триггер в итоге перейдет в устойчивое состояние, однако будет это логический ноль или единица заранее предсказать нельзя. Также случайным является длительность временного интервала, в течении которого триггер будет находиться в метастабильности.

Временной анализ проводится для того, чтобы убедиться, что ограничения, связанные с временем установки и удержания, выполняются и защелкивающий триггер никогда не попадет в метастабильное состояние. Так как для асинхронных и мезохронных тактовых сигналов временной анализ невозможно провести корректно, в этих случаях при передаче данных между двумя доменами нужно ставить дополнительные синхронизаторы, защищающие от метастабильности.

Рассмотрим практический пример, передачи данных между двумя асинхронными тактовыми доменами. Для простоты пусть каждый домен состоит всего из одного триггера. Схема проекта показана на рисунке 4. Описание на System Verilog 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

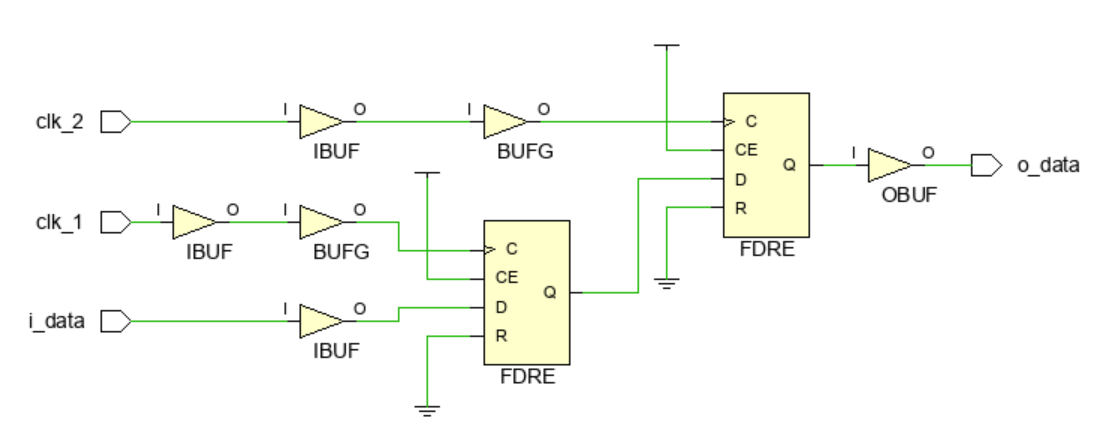


Рисунок 4. Схема проекта.

Чтобы Vivado мог 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 [2]:

# 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]

Каким образом в проекте осуществляется временной анализ при пересечении тактовых доменом, можно посмотреть 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 между двумя доменами.

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, что они являются асинхронными, и для них невозможно корректно провести временной анализ. Поэтому в таблице клетка на пересечении clk\_1 и clk\_2 обозначена красным цветом.

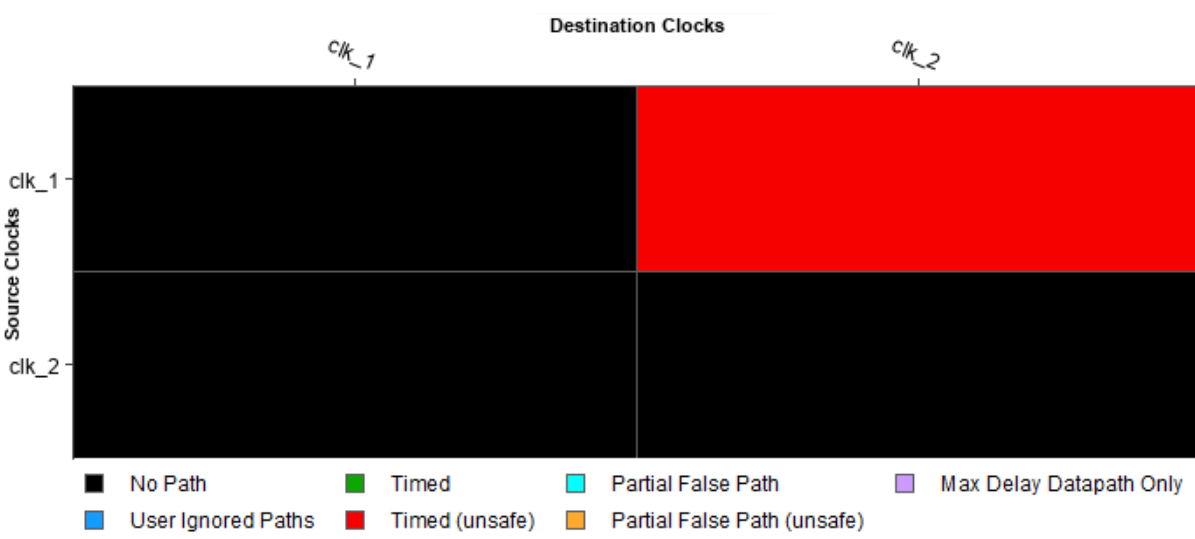


Рисунок 5. Таблица взаимодействия тактовых сигналов.

Данный отчет можно также 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 на System Verilog 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

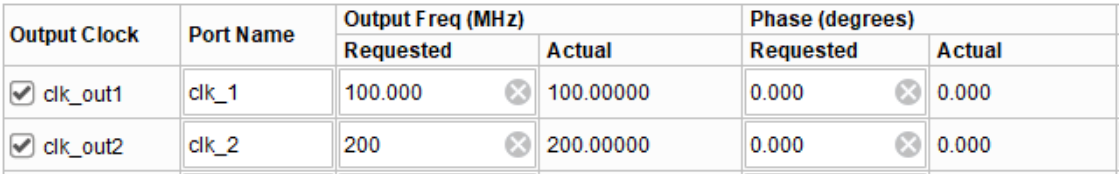


Рисунок 6. Настройки PLL.

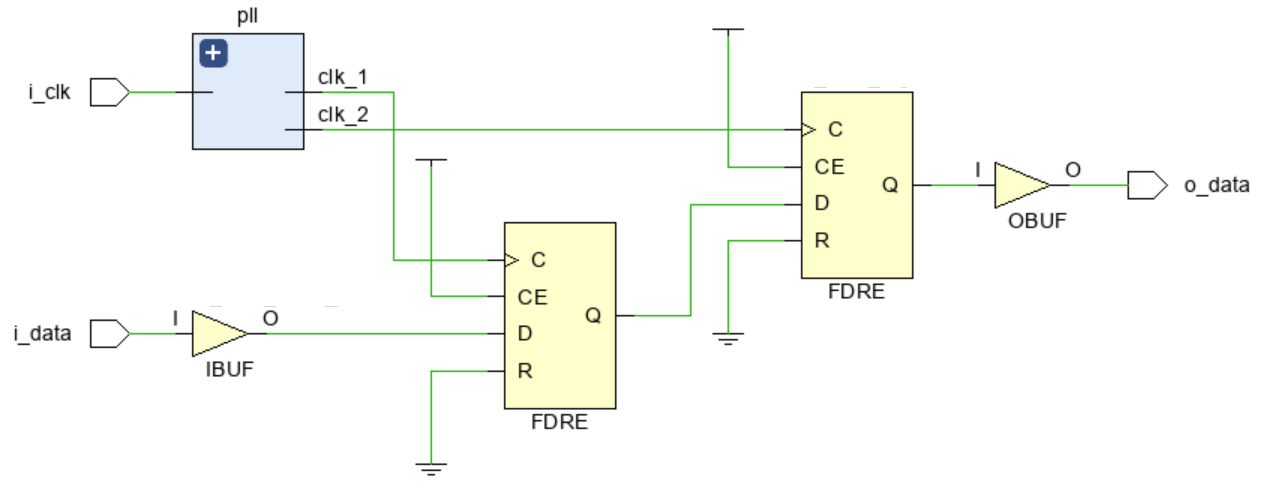


Рисунок 7. Схема проекта.

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 с ограничениями 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, сгенерированных с помощью PLL тактовых сигналов

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 необходимо указать период сигнала i\_clk с помощью команды *create\_clock*. 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. Однако эти имена можно изменить. Для этого с помощью команд *set* создаются переменные, указывающие места формирования тактовых сигналов. Далее, используя команды *create\_generated\_clock* и ключом *-name*, можно присвоить сигналам 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 передачи данных из домена clk\_1 в домен clk\_2, выделена зеленым цветом.

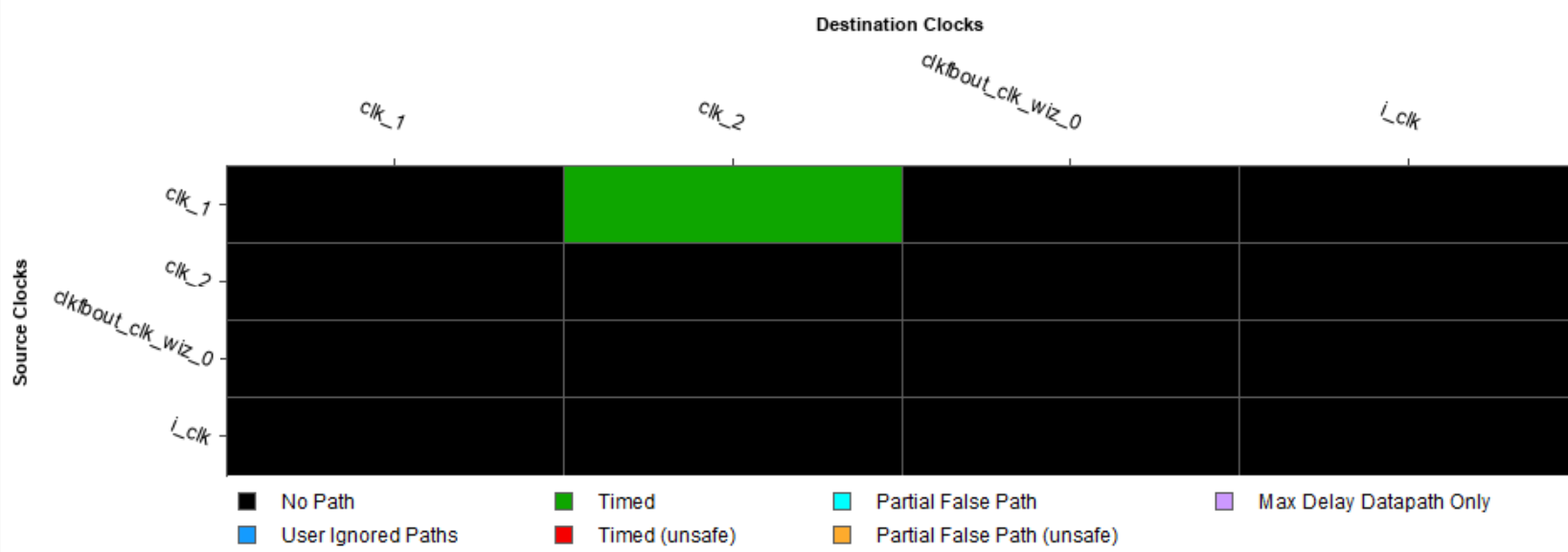


Рисунок 8. Таблица взаимодействия тактовых сигналов.

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 считается корректным (Timed), так как все фазовые соотношения между тактовыми сигналами точно известны и детерминированы (Clean).

1. **False Path Constrains.**

Иногда наличие в проекте асинхронных тактовых сигналов, для которых выполняется временной анализ, может привести к проблемам. Опять рассмотрим пример проекта, состоящего из двух триггеров (см. рисунок 4). В xdc-файл добавим команды, задающие период в 10 нс для сигнала clk\_1 и 5 нс для сигнала clk\_2:

# 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 для анализа оп *Setup*. В строке *Requirements* можно увидеть, что запускающий фронт сигнала clk\_1 появляется в нулевой момент времени. Через 5 нс после это появляется защелкивающий фронт сигнала clk\_2. Общий период равен 10 нс, в течении которого появится всего один фронт clk\_1. Таким образом анализатор рассчитал, что значение *ΔTsu* равно 5 нс.

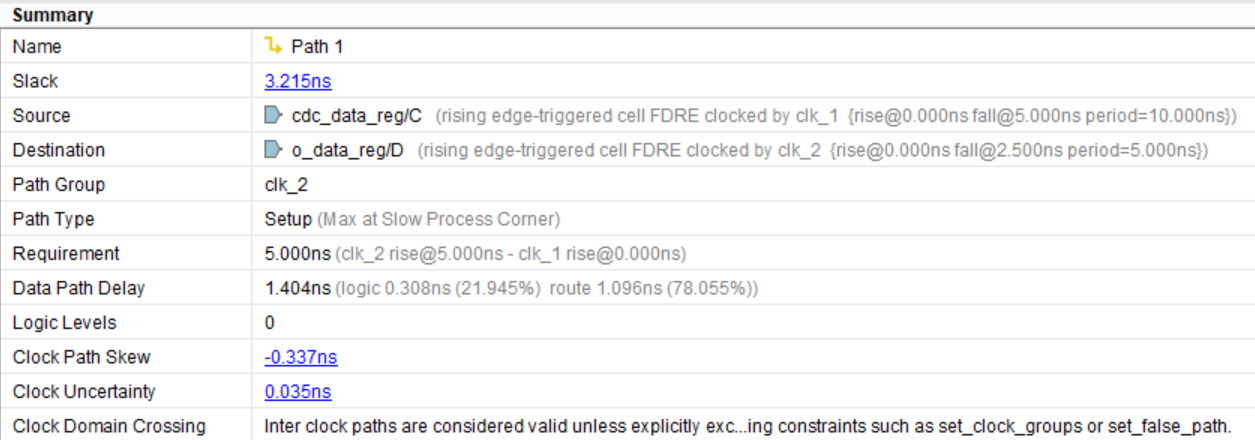


Рисунок 9. Раздел *Summary* временного отчета.

Хотя это и не имеет смысла, но анализатор все равно рассчитывает все задержки и вычисляет значение *Slack*. Оно больше нуль и равно 3.215 нс, то есть временные ограничения выполнены.

Теперь 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. Можно увидеть, что Slack имеет отрицательное значение, то есть временные ограничения не выполнены. Давайте найдем причину.

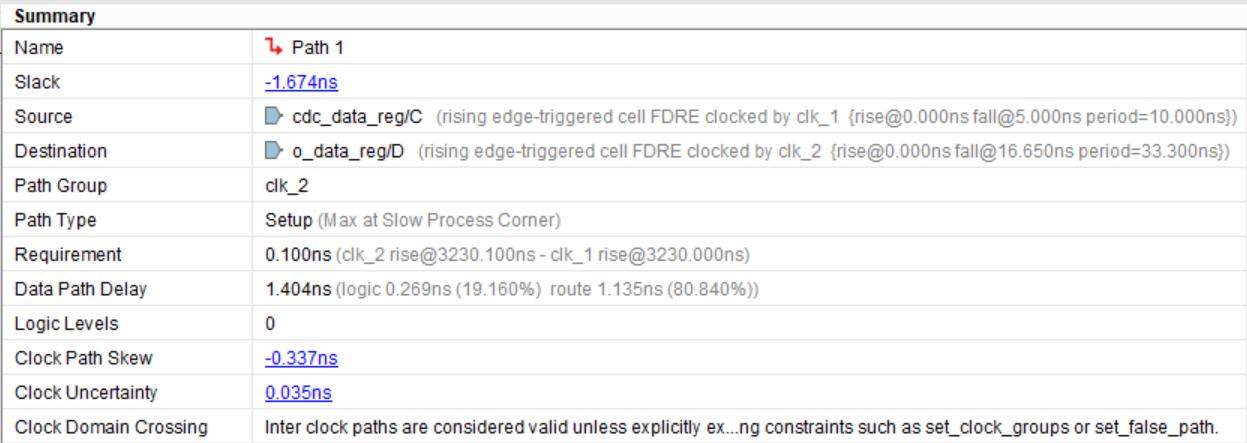


Рисунок 10. Раздел *Summary* временного отчета.

В строке *Requirements* указано минимальное значение *ΔTsu*,равное 0.1 нс. Это соответствует ситуации, 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 очень мало. 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 сложное, такие как *ΔTsu* = 0.1 нс, рассматриваются в первую очередь. Из-за этого возможно ситуация, когда Vivado приложит все усилия, чтобы выполнить временные ограничения для одного критического пути, за счет небольших нарушений для других путей. То есть в проекте появятся нарушения, которых могло и быть, если бы путь между асинхронными доменами не учувствовал 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. Тогда 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 триггера (*cdc\_data\_reg/C*) и информационный вход защелкивающего (*o\_data\_reg/D*).

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

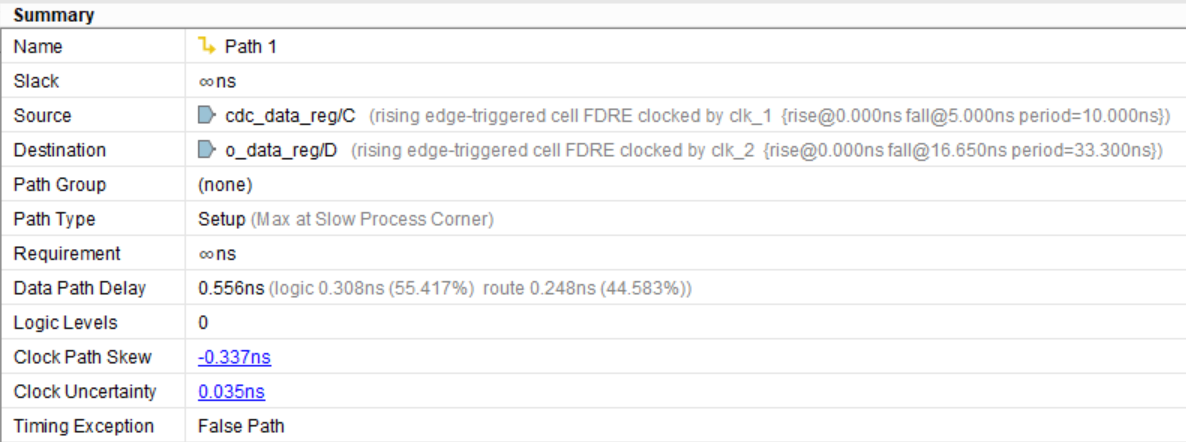


Рисунок 11. Раздел *Summary* временного отчета.

Также рассмотри отчет *Clock Interaction*, представленный на рисунке 12. Ячейка, соответствующая передачи данных из домена clk\_1 в домен clk\_2, обозначена синим цветом. Это означает, что разработчик вручную убрал пути из временного анализа (*User Ignored Paths*).

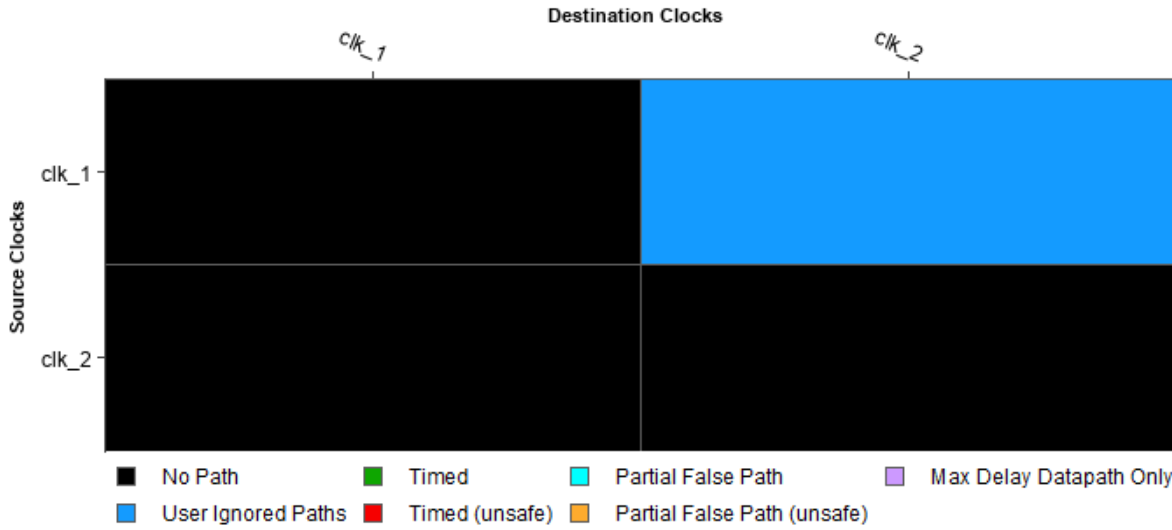


Рисунок 12. Таблица взаимодействия тактовых сигналов.

Ту же информацию из отчета *Clock Interaction* можно получить текстовой форме:

Clock Interaction Table

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

Clock-Pair Inter-Clock

From Clock To Clock Classification Constraints

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

clk\_1 clk\_2 Ignored False Path

Параметры для команды *set\_false\_path* можно задавать и в более упрощенном виде. Например, в качестве начала и конца пути можно просто указать запускающий и защелкивающий триггеры. В этом случае буду исключены все пути, идущие от входа C триггера *cdc\_data\_reg* до всех входов триггера *o\_data\_reg* (входы D, CE и другие).

# объявление false path через запускающий и защелкивающий триггеры

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

-to [get\_cells o\_data\_reg]

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 исключения путей из временного анализа заключается в объявлении тактовых доменов асинхронными. Для этого нужно с помощью команды *set\_clock\_groups* объявить группы тактовых сигналов и добавить флаг *-asynchronous*. В этом случае будут исключаться все пути из домена clk\_1 в домен clk\_2, а также из clk\_2 в 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 указывается 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. **Report CDC.**

Использование *false path* и асинхронных групп может защитить от ложных временных нарушений при передачи данных между тактовыми доменами. Однако, их использование не гарантирует, что данные в принципе будут передаваться корректно. В Vivado есть возможность проверить правильность передачи данных между доменами с помощью отчета о CDC (Clock Domain Cross). Для этого в TCL-консоли нужно выполнить команду *report\_cdc*. Ее результат 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

Можно увидеть, что при невозможно установить корректность передачи данных (*Unsafe* = 1) при передачи данных из домена clk\_1 в домен clk\_2. Это приводит к наличию в проекте критическому замечанию (*Severity* = *Critical*).

Более подробно о причине замечания можно узнать в отчете *Report Methodology* (см. вкладку Vivado Flow Navigator). Один из разделов отчета представлен ниже:

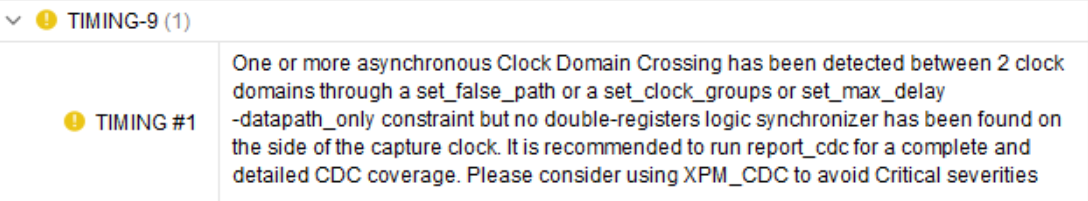


Рисунок 13. Отчет о проверки методологии проектирования.

В нашем проекте есть пути, которые пересекают тактовые домены и исключены из временного анализа с помощью команд *set\_false\_path* и *set\_clock\_groups*. В этом случае, чтобы гарантировать корректность передачи, данные должны проходить через специальные синхронизаторы, например, через сдвоенные триггеры (double-register logic). При формировании отчета о CDC Vivado, используя netlist, пытается найти синхронизаторы между асинхронными доменами, и в случае их отсутствия выводит замечание. Все структуры в netlist, которые Vivado распознает в качестве синхронизаторов, можно посмотреть в [1] глава 1.

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 System Verilog представлены ниже.

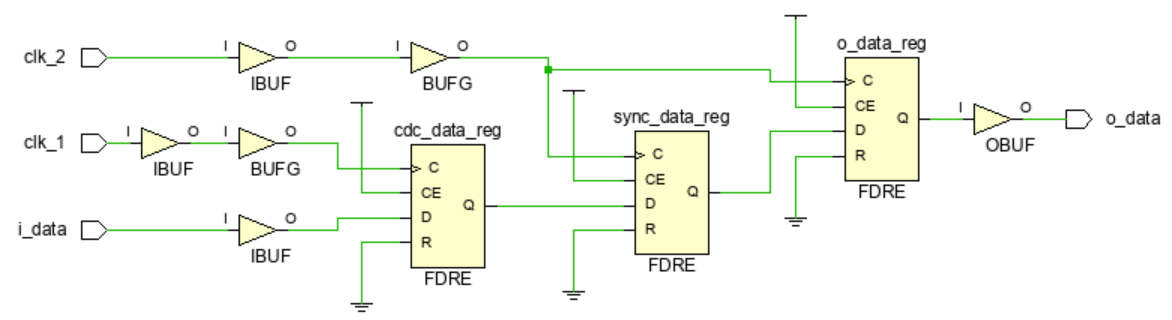


Рисунок 14. Схема проекта.

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

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 передачи данных могут для триггера *sync\_data\_reg* возникать нарушения времени установки и удержания, и он может попадать в метастабильное состояние. В случае метастабильности сигнал на выход *sync\_data\_reg* будет иметь некоторый неопределенный уровень между нулем и единицей. Этот сигнал будет распространяться дальше по схеме и в итоге попадет на вход триггера *o\_data\_reg*.

Теперь, если на триггер *o\_data\_reg* подать тактовый фронт, то он защелкнет неопределенный уровень сигнала и сам перейдет в метастабильное состояние. Однако, как было указано ранее, метастабильность – это состояние неустойчивое, и поэтому рано или поздно триггер *sync\_data\_reg* из него 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 попадет в метастабильное состояние.

Таким образом, триггер *sync\_data\_reg* может попадать в метастабильное состояние, но, если он из него выйдет до появления следующего фронта, то метастабильность не появится на выход *o\_data\_reg* и не распространится дальше по схеме. То есть, метастабильность будет локализована внутри синхронизатора.

Рассмотрим, как изменился отчет о CDC. Наличие синхронизатора означает, что данные между доменами передаются корректно (*Safe* = 1). К проекту по-прежнему есть замечания, но они уже не критические (*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.**

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

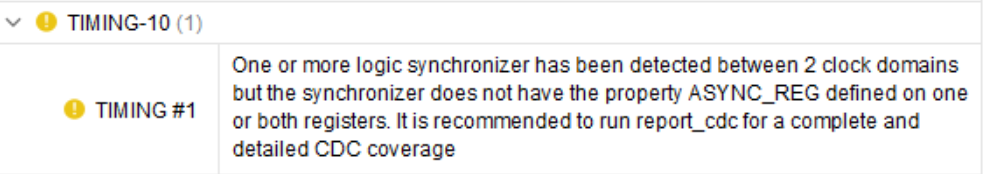


Рисунок 15. Отчет о проверки методологии проектирования.

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

# 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}]

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

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

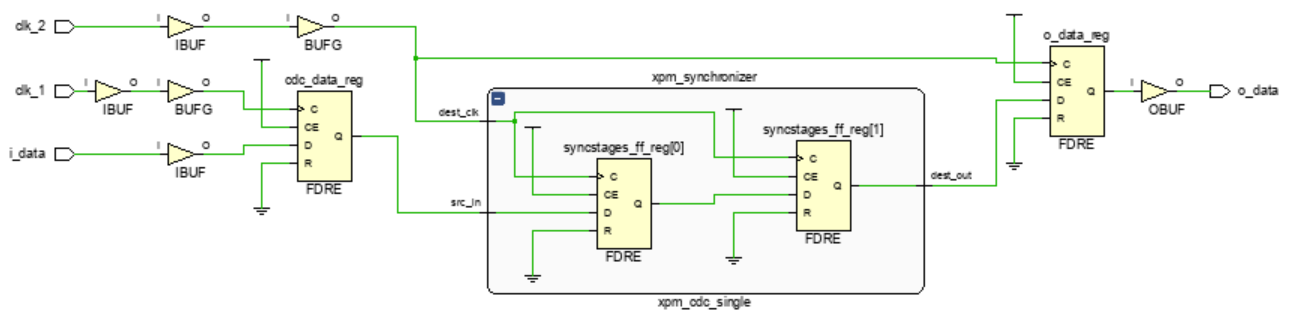


Рисунок 16. Схема проекта.

# 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]

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

**Заключение.**

**Ссылки.**