Продолжение. Начало в № 3 `2008

Краткий курс HDL.
Часть 11. Асинхронные частоты, пересечение клоковых доменов и синхронизация

Иосиф КАРШЕНБОЙМ
iosifk@narod.ru

Выполнение канала передачи данных

Для синхронизации сигналов в проекте есть следующее правило: один сигнал должен синхронизироваться только в одном месте, то есть один сигнал не должен проходить более чем через один синхронизатор. Поскольку синхронизация требует от одного до двух тактовых циклов, то проектировщик не может достоверно предсказать, когда каждый из этих асинхронных сигналов проходит через домен синхрочастоты. Кроме того, синхронизация группы сигналов, которые уже были засинхронизированы и попадают в новый домен синхрочастоты, может измениться, потому что задержка может быть или в один тактовый цикл, или в два цикла, в зависимости от того, когда эти сигналы приходят к синхронизатору. Такая ситуация приводит к «условию гонки» между индивидуально синхронизированными сигналами. Это же условие гонки может быть отнесено и к группам сигналов, таких как данные, адреса, и к шинам управления, которые необходимо передавать вместе через домены синхрочастоты. Таким образом, нет необходимости использовать индивидуальные синхронизаторы для каждого сигнала в группе или для каждого бита шины данных либо шины адреса, поскольку в этих шинах каждый сигнал должен быть достоверным в новом домене синхрочастоты в одно и то же время. Один из способов решения проблемы синхронизации сигналов для шины состоит в том, чтобы использовать регистр временного хранения информации и дополнительные сигналы для установления связи. Такая схема включает в себя регистр, который защелкивает сигналы шины и схему, обеспечивающую процедуры установления связи (рис. 12).

Сигналы установления связи указывают схеме, приемнику данных, находящейся в новом домене синхрочастоты, момент времени, после которого она может читать защелкнутые с шины данные, и, соответственно, когда передающая схема может произвести следующую запись данных в регистр временного хранения информации. В таком проекте передающая схема хранит данные в регистре-защелке для временного хранения информации, и при защелкивании данных эта схема устанавливает сигнал запроса, свидетельствующий о том, что есть новые данные на передачу. Эти два действия могут произойти одновременно, потому что сигналу запроса необходим по крайней мере один тактовый цикл, прежде чем схема получения обнаружит его (это минимальная задержка синхронизации установления связи). Когда схема-приемник читает данные, то она устанавливает сигнал подтверждения. Такой режим работы использует полную процедуру установления связи и занимает много времени на то, чтобы завершить передачу. Проект, использующий полное установление связи при передаче данных, имеет большое окно по времени для схемы, приемника данных, во время которого приемник читает данные с шины, и, таким образом, этот проект не совсем эффективен. Но в том же самом проекте, для того чтобы ускорить передачу, можно вместо полного использовать только частичное установление связи.

При таком варианте шинной синхронизации можно синхронизировать только сигналы установления связи, но не саму шину сигнала. Она выходит из регистра временного хранения информации и должна оставаться устойчивой до окончания цикла чтения схемой-приемником. Необходимо отметить, что описанная шинная синхронизация, возможно, не будет работать в таком приложении, где передающая схема выдает данные слишком быстро, и схема-приемник не всегда успевает их обработать.

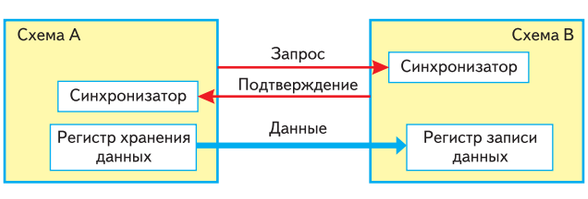

Рис. 12. Канал передачи данных, который содержит:
в схеме A — регистр временного хранения информации и дополнительные сигналы для установление связи;
в схеме B — регистр, который защелкивает сигналы шины и схему, обеспечивающую процедуры установления связи

Небольшое дополнение к предыдущему пункту

Довольно часто в телеконференции встречаются такие вопросы: «Я сделал описание небольшого статического автомата, он загружается в микросхему и работает. Но иногда почему-то «зависает» и находится в таком состоянии до тех пор, пока на автомат не будет подан сигнал «сброс», или надо перезагрузить микросхему. В чем тут дело?» После некоторых наводящих вопросов выясняется, что на вход автомата поступает асинхронный сигнал. На предложение засинхронизировать входной сигнал следует примерно такой ответ, что, мол, этот входной сигнал и так поступает на триггеры автомата и там синхронизируется. Давайте рассмотрим эту ситуацию.

Как было описано в предыдущих разделах, любой автомат состоит их регистра, представляющего из себя набор триггеров, и комбинационной логики. При этом физически триггеры, входящие в состав регистра автомата, разнесены в пространстве. Следовательно, любой сигнал, даже если он возник в какой-то одной точке кристалла, например, на входе микросхемы, приходит ко всем триггерам не одновременно из-за разной задержки во времени при прохождении сигналом неодинаковых путей по внутренним интерконнектам микросхемы. Таким образом, если входной сигнал не будет синхронизирован, то к одним триггерам такой сигнал может успеть прийти вовремя и не попасть в опасную зону **Окна Метастабильности**, а к другим триггерам сигнал придет позже и попадет в зону **Окна Метастабильности**. Следовательно, первые триггеры выполнят правильный переход, обусловленный входными сигналами, а вот вторые триггеры попадут в зону метастабильности, и их состояние будет не определено. Если ав-

томат выполнен в стиле “one hot”, то это значит, что для каждого состояния выделен отдельный триггер, и еще это значит, что благодаря избытку триггеров схема может иметь больше состояний, чем нужно разработчику. Но даже если же автомат и не выполнен в стиле «one hot», но в проекте не задействованы все состояния автомата, то в наиболее общем случае автомат может перейти в такое состояние, которое не было продекларировано разработчиком. Компилятор же, возможно, не сделал выхода из необъявленных состояний автомата. Поэтому автомат, попадая в такие состояния, и не выходит из них. Вот это как раз и есть та самая ситуация, когда проект «почему-то «зависает»»… Вывод прост: сигналы, подаваемые на автомат состояний, должны быть синхронизированы в том же домене синхрочастоты.

И еще одно небольшое дополнение к предыдущему пункту

В предыдущих разделах при описании проблем, возникающих при CDC, в основном все внимание было обращено только на временные соотношения сигналов. Но в реальной жизни, кроме проблем, связанных с временными соотношениями, есть еще и проблемы, связанные с тем, что сигналы имеют неидеальные фронты. Если периоды синхрочастот двух клоковых доменов сильно отличаются, то, может быть, так же сильно будут отличаться и фронты сигналов этих синхрочастот. А возможны также случаи, когда длительности фронтов будут отличаться настолько, что медленный фронт для домена быстрой синхрочастоты будет представлять собой пологий нарастающий или спадающий сигнал, длительность которого будет соизмерима с длительностью нескольких тактов быстрой синхрочастоты. Мало того, такой медленный фронт не всегда будет пологим. На нем могут быть любые выбросы, вызванные неидеальным согласованием или любыми другими причинами. Для примера можно сравнить тактовые частоты внутри и снаружи микросхемы. Внутренние тактовые частоты могут доходить до 500 МГц, в то время как внешние, по отношению к микросхеме, тактовые частоты обычно находятся в пределах 10–50 МГц. То есть частоты могут отличаться на один-два порядка. В таком случае даже если будет применен синхронизатор, то после него во входном сигнале могут быть просечки, соответствующие неидеальным фронтам медленного сигнала. Эти просечки для статического автомата, который тактируется на внутренней частоте микросхемы, будут восприняты как полезный сигнал. И, соответственно, автомат должен их отработать. Но поскольку последовательность перепадов во входном сигнале уже не будет подчиняться тому алгоритму, по которому должен работать автомат, то автомат также может попасть в тупиковое состояние. Особенно это необходимо иметь в виду в том случае, когда в качестве входного сигнала используется сигнал от каскада с выходом по открытому коллектору. Например, такая ситуация может возникнуть при разработке контроллера I²C. Частота на шине I²C — от 100 до 400 кГц, фронты пологие, и на них возможны колебания.

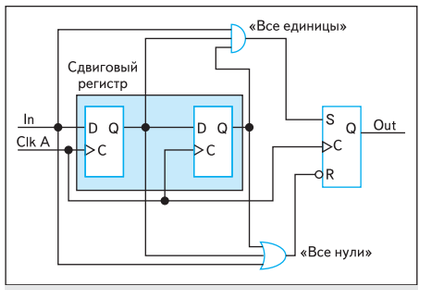

Рис. 13. Входной сигнал пропускается через цифровой фильтр

Поэтому есть два варианта действий в этом случае. Первый — аппаратный. После синхронизатора входной сигнал пропускается через простейший цифровой фильтр.

На рис. 13 показана схема такого фильтра. Цепочка триггеров, представляющая собой сдвиговый регистр, задерживает входной сигнал на то время, когда гарантированно должен закончиться переходный процесс. Исходя из длительности переходного процесса и зная длительность импульса помехи, можно подобрать число триггеров в сдвиговом регистре для задержки и/или подобрать более низкую частоту тактирования для триггеров. Под более низкой частотой тактирования здесь понимается то, что на триггеры будет выдаваться сигнал разрешения тактирования, и, таким образом, тактирование будет производиться не под каждый импульс синхрочастоты, а, например, под каждый 16-й импульс. Когда на выходах всех триггеров появится «1», то выходной триггер переключится в состояние «1». И, соответственно, только когда на выходах всех триггеров появится «0», то выходной триггер также переключится в состояние «0».

Второй вариант можно назвать «таймерный». В схему добавляется таймер, возможно, аппаратно-загружаемый константой или программно-загружаемый, и этот таймер осуществляет задержку входного сигнала. Такой вариант целесообразен в том случае, когда требуется задержать входной сигнал на довольно большое время.

Двунаправленная шина данных

В приведенных диаграммах обмена по протоколам установления связи не рассматривались особенности работы шины при двунаправленном обмене данными. Но сегодня тактовые частоты таковы, что при работе двунаправленной шины можно порекомендовать воспользоваться алгоритмами работы шины PCI. Для «мастера» шины в алгоритм обмена по приему данных добавлен дополнительный такт, называемый «переключение шины». «Мастер» шины, прежде чем выставлять сигнал чтения, переводит свои выходы в третье состояние и только затем выдает сигнал чтения. При таком алгоритме работы на шине не происходит столкновение данных.

Проект канала данных, имеющий дополнительные возможности

Зачастую данные при пересечении домена синхрочастоты должны «накопиться», поэтому в таком случае не работают проекты, использующие только один регистр для временного хранения информации. Например, передающая схема выдает данные в виде нескольких циклов передач по шине, непрерывно следующих друг за другом (burst), при этом может оказаться, что данные передаются слишком быстро для схемы приема, и она не успевает их читать. Другой вариант: если схема получения данных имеет разрядность меньшую, чем схема передачи данных. Тогда необходимо использовать буферный накопитель типа FIFO.

Существенно то, что разработчик применяет FIFO для выравнивания скоростей передачи и приема, а также выравнивания разрядности данных, или по обеим этим причинам. Для использования FIFO с целью выравнивания скоростей передачи и приема более быстрый порт на стороне передачи данных обрабатывает циклы записи данных в режиме burst, в то время как более медленный порт на стороне приема считывает данные на постоянной скорости. Однако и при различном типе доступа по разрядности и скорости средняя скорость передачи данных в FIFO и из него должна быть одинаковой, иначе FIFO переполняется или опустошается. Правда, надо заметить, что для некоторых приложений критично только переполнение FIFO, так как при этом данные могут потеряться. Как и в проекте с одним регистром, FIFO сохраняет данные в своих внутренних регистрах или своей внутренней памяти, в то время как его синхронизированные сигналы состояния определяют, когда данные могут быть записаны в FIFO, или когда из FIFO данные могут быть считаны. Порты FIFO, как по чтению, так и по записи, могут иметь различную синхрочастоту, соответствующую скорости работы схемы, подключенной к этому порту.

Регистры в FIFO используют синхрочастоту порта записи так же, как и регистр временного хранения информации использует синхрочастоту схемы, изменяющей содержание регистра. Синхронизация сигнала имеет место в логике указателя и более сложна, чем сигналы для установления связи.

Есть несколько способов для того, чтобы спроектировать логику указателя.

Первый метод состоит в том, что необходимо синхронизировать только стробы чтения и записи, а счетчики для указателей расположить в каждом домене синхрочастоты. Счетчики синхронны с соответствующими

портами, и эти счетчики показывают, какое число слов данных доступно по чтению и записи. Счетчик числа слов, доступных для чтения, показывает наличие и число достоверных слов данных для чтения. Счетчик числа слов, доступных по записи, показывает, сколько слов данных можно записать в FIFO.

Когда производится сброс схемы, то указатель сбрасывается в такое состояние, когда число слов, доступных по чтению, равно нулю, и число слов данных, доступных по записи, равно общему числу слов, которые может хранить данное FIFO. Это означает, что все FIFO свободны для того, чтобы записать в них новые данные.

Когда происходит операция чтения из FIFO, то строб чтения декрементирует указатель на чтение и, проходя через синхронизатор на стороне записи, инкрементирует указатель на запись. При операции записи в FIFO происходит обратное действие: строб записи декрементирует указатель на запись и, проходя через синхронизатор на стороне чтения, инкрементирует указатель на чтение.

В таком проекте для каждого порта необходимы только один импульс длительностью в один соответствующий период синхрочастоты и синхронизатор, работающий по приходящим импульсам для чтения и записи. Необходимость синхронизатора, работающего по приходящим импульсам, вызвана тем, что сигнал активного уровня выходит из своего, медленного домена синхрочастоты и приходит в более быстрый домен. Следовательно, он остается в более быстром домене в активном состоянии на большее количество тактовых циклов, чем этот же сигнал в медленном домене. Поскольку каждый счетчик изменяется всякий раз, когда происходит чтение или запись, то сигнал будет достоверным и активным, и тогда в более быстром домене синхрочастоты, если бы синхронизатор работал по уровню, а не по импульсу, могло бы произойти больше обнаружений фактов чтения или записи, чем это фактически произошло в более медленном домене синхрочастоты.

Синхронизатор, работающий по приходящему импульсу, переводит импульс длительностью в один период синхрочастоты в одном домене синхрочастоты в импульс длительностью в один период синхрочастоты нового домена синхрочастоты, и каждый импульс представляет только одно чтение или запись в FIFO.

Эта методика определения состояния FIFO дает пессимистическое состояние, как для чтения, так и для записи.

Сигнал состояния для порта записи показывает, что порт записи полон, когда FIFO уже заполнен. Если при этом произойдет чтение на стороне порта чтения, то сигнал состояния на стороне записи все еще будет показывать, что FIFO полный, даже после окончания импульсов чтения, только потому, что синхронизация задерживает строб, который должен прийти к счетчику записей.

Эта ситуация происходит и при состоянии FIFO, когда он пустой, и происходит запись в FIFO. Но импульс записи, проходя через синхронизатор, задерживается и попадает в порт чтения позже.

Другой вариант выполнения этого проекта позволяет обнаруживать состояния FIFO — «полный» или «пустой» — в нужное время. Если FIFO имеет только одну свободную ячейку для того, чтобы сохранить слово с входа записи, и строб записи переключился в активное состояние, то уже в этот момент FIFO должен установить состояние, указывающее, что он полон. Этот сценарий дает индикацию состояния FIFO, указывающего на то, что он заполнен, что происходит на один период синхрочастоты раньше. И это может дать схеме, пишущей в FIFO, достаточное количество времени, чтобы остановить следующую запись и не допустить переполнения FIFO. Эта ситуация справедлива и для порта чтения в FIFO. В этом случае, если в FIFO только одно слово данных, доступное по чтению, и строб чтения переключается, то уже можно выдавать состояние FIFO как «пустое», что даст дополнительное время для схемы чтения, чтобы предотвратить чтение из пустого FIFO. Такой способ работы с логикой указателей ограничивает возможность для доступа в FIFO. В таком FIFO нет возможности доступа на каждом тактовом цикле, даже и в медленном домене синхрочастоты. Преимущество этого варианта состоит в том, что для схем, обращающихся в FIFO, есть, по крайней мере, один тактовый цикл, чтобы оценить состояние FIFO.

Таким образом, в этом FIFO можно заполнить каждую ячейку хранения данными, не переполняя FIFO, то есть не записывая данные поверх достоверных данных. Точно так же FIFO может быть пустым, и при этом не производится чтение данных из пустого FIFO. Другое преимущество этого проекта состоит в том, что на каждой стороне FIFO можно читать соответствующий счетчик и определять, сколько ячеек хранения данных доступно. Проектировщик может использовать такие FIFO для схем, которые выполняют многократные чтение или запись данных, не вызывая переполнения или опустошения. Недостаток этого проекта состоит в том, что состояние определяют счетчики, а не прямые сравнения указателей на чтение и запись. Для большого FIFO, соответственно, разрядность этих счетчиков тоже будет большой. Кроме того, средние скорости передачи данных не могут быть выше, чем половина самой медленной синхрочастоты. Это вызвано тем, что при чтении или записи импульсы от более быстрого домена синхрочастоты должны пройти в домен более низкой синхрочастоты, и промежуток между ними должен длиться не менее чем два периода синхрочастоты, поскольку используется синхронизатор, работающий по импульсу.

Один из способов устранить некоторые из этих проблем состоит в том, чтобы использовать прямое сравнение указателей. В таком проекте FIFO производится сравнение указателей на чтение и запись, и это сравнение определяет состояние FIFO. Сравнение указателя в асинхронных проектах является более сложным вариантом, потому что каждый указатель существует в своем домене синхрочастоты и представляет собой шину с несколькими разрядами. А для синхронизации шины требуется, чтобы шина не изменялась до тех пор, пока идет обмен сигналами установления связи. В таком проекте FIFO при использовании синхронизации указателя невозможно получить высокий уровень быстродействия. Чтобы решить эту проблему, в FIFO для логики указателя вместо двоичного кода используется код Грея. Код Грея отличается от двоичного кода тем, что для каждого увеличения или уменьшения счетчика одновременно изменяется только один бит (табл. 3)

Таблица 3. Пример двоичного кода и кода Грея

Десятичный код	Двоичный код	Код Грея
0	0	0
1	1	1
2	10	11
3	11	10
4	100	110
5	101	111
6	110	101
7	111	100
8	1000	1100
9	1001	1101
10	1010	1111
11	1011	1110
12	1100	1010
13	1101	1011
14	1110	1001
15	1111	1000

Преобразование кода Грея в двоичный код

```
bin[0] = gray[3] ^ gray[2] ^ gray[1] ^ gray[0];
bin[1] = gray[3] ^ gray[2] ^ gray[1];
bin[2] = gray[3] ^ gray[2];
bin[3] = gray[3];
```

Пример 1. Преобразование 4-битового кода Грея в двоичный код

Самый простой способ выполнить преобразователь из кода Грея в двоичный код — это сформировать цикл **for** и сделать свертку по исключающему ИЛИ для каждого бита вектора кода Грея с переменным индексным диапазоном младшего бита, где каждый раз в цикле **for** индекс младшего бита увеличивается на единицу до тех пор, пока не останется простое назначение bin [MSB] = ^gray [MSB:MSB] (остался только 1-битовый MSB от вектора кода Грея), так, как показано в примере 1. Описание на языке Verilog, соответствующее алгоритму, приведенному в примере 1, показано в примере 2. К сожалению, Verilog не позволяет работать только с частью битов, используя переменный индексный диапазон, так, как это показано в при-

мере 2. И хотя этот проект концептуально правильный, он не будет компилироваться.

```
module gray2bin_bad (bin, gray);
parameter      SIZE = 4;
  output       [SIZE-1:0] bin;
  input        [SIZE-1:0] gray;
  reg          [SIZE-1:0] bin;
  integer                 i;
  // Syntax Error — variable index range
  always @(gray)
    for (i=0; i<SIZE; i=i+1)
      bin[i] = ^(gray[SIZE-1:i]);
endmodule
```

Пример 2. Концептуально правильное описание преобразователя кода Грея в двоичный код, но оно не работает

```
bin[0] = gray[3] ^ gray[2] ^ gray[1] ^ gray[0] ; // gray>>0
bin[1] = 1'b0    ^ gray[3] ^ gray[2] ^ gray[1] ; // gray>>1
bin[2] = 1'b0    ^ 1'b0    ^ gray[3] ^ gray[2] ; // gray>>2
bin[3] = 1'b0    ^ 1'b0    ^ 1'b0    ^ gray[3] ; // gray>>3
```

Пример 3. Преобразование 4-битового кода Грея в двоичный код. Второй способ

Другой способ создать преобразователь кода Грея в двоичный код состоит в использовании свертки по исключающему ИЛИ для каждого существенного бита кода с дополнением нулями, так, как показано в примере 3. Параметризированный и правильно работающий преобразователь кода Грея в двоичный код, работающий по этому алгоритму, показан в примере 4. Этот пример синтаксически правилен, он компилируется и работает.

```
module gray2bin (bin, gray);
  parameter SIZE = 4;
  output [SIZE-1:0] bin;
  input [SIZE-1:0] gray;
  reg [SIZE-1:0] bin;
  integer i;
  always @(gray)
    for (i=0; i<SIZE; i=i+1)
      bin[i] = ^(gray>>i);
endmodule
```

Пример 4. Параметризированный и правильно работающий преобразователь кода Грея в двоичный код

Преобразование двоичного кода в код Грея

Чтобы выполнить преобразование *n*-битового двоичного кода в код Грея, используют алгоритм, приведенный в примере 5. В этом примере показаны уравнения для преобразования 4-битового двоичного кода в код Грея. Для каждого бита кода Грея выполняют операцию исключающего ИЛИ над значениями входных двоичных данных, например, бит кода Грея 0 равен результату операции над двоичными битами 0 и 1. Бит кода Грея 1 равен результату операции над двоичными битами 1 и 2 и т. д. Старший бит кода Грея равен старшему биту двоичного кода.

```
gray[0] = bin[0] ^ bin[1];
gray[1] = bin[1] ^ bin[2];
gray[2] = bin[2] ^ bin[3];
gray[3] = bin[3];
```

Пример 5. Преобразование 4-битового двоичного кода в код Грея

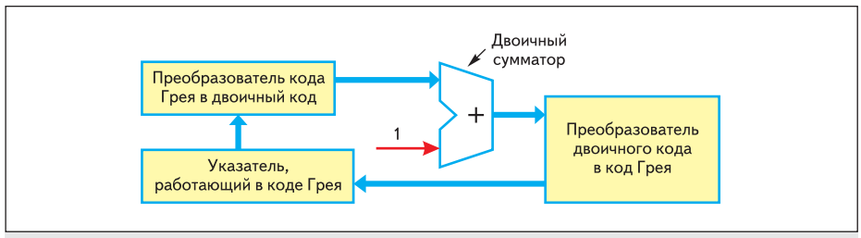

Рис. 14. В качестве счетчика в коде Грея целесообразнее применить двоичный сумматор с конвертерами в код Грея и из кода Грея на входе и выходе сумматора

Самый простой способ выполнить конвертер для преобразования двоичного кода в код Грея состоит в том, чтобы сделать простое непрерывное назначение, которое выполняет поразрядную операцию исключающего ИЛИ между двоичным вектором и сдвинутой вправо версией того же самого двоичного вектора, так, как показано в примере 6. Этот пример синтаксически правилен, он компилируется и работает.

```
module bin2gray (gray, bin);
    parameter SIZE = 4;
    output [SIZE-1:0] gray;
    input [SIZE-1:0] bin;
    assign gray = (bin>>1) ^ bin;
endmodule
```

Пример 6. Параметризированный преобразователь двоичного кода в код Грея

При использовании синхронизаторов на шинах, работающих в коде Грея, не возникает условия гонки сигналов между битами, потому что при изменении данных на шине каждый раз изменяется только один сигнал, даже притом, что все сигналы этой шины проходят через отдельные синхронизаторы. Поэтому для этого проекта используют указатели, представляющие собой счетчики, работающие в коде Грея. Применение двоичных указателей с последующим преобразованием их к коду Грея потребовало бы синхронизации значений указателя, причем эти сигналы уже не снимались бы с выхода триггеров счетчика, а выходили бы из логики, необходимой для преобразования. А это нарушает ограничение, связанное с пересечением домена синхрочастоты. Данное ограничение разрешает использовать для синхронизации только те сигналы, которые снимаются с выходов триггеров. Поэтому в таком случае в качестве счетчика в коде Грея целесообразнее применить двоичный сумматор с конвертерами в код Грея и из кода Грея на входе и выходе сумматора (рис. 14).

Преобразование в код Грея и обратно — это операция XOR, таким образом, необходимо к двоичному счетчику добавить только несколько логических элементов.

Счетчик в коде Грея

Код на Verilog для счетчика кода Грея представляет собой узел, в который входят преобразователь кода Грея в двоичный код, преобразователь двоичного кода в код Грея и двоичный сумматор, который инкрементирует значение данных между преобразованиями. Параметризованная модель на Verilog для счетчика, работающего в коде Грея, показана в примере 7.

```
module graycntr (gray, clk, inc, rst_n);
  parameter SIZE = 4;
  output [SIZE-1:0] gray;
  input clk, inc, rst_n;
  reg [SIZE-1:0] gnext, gray, bnext, bin;
  integer i;

  always @(posedge clk or negedge rst_n)
    if (!rst_n)
      gray <= 0;
    else
      gray <= gnext;
  always @(gray or inc)
  begin
      for (i=0; i<SIZE; i=i+1)
        bin[i] = ^(gray>>i);
      bnext = bin + inc;
      gnext = (bnext>>1) ^ bnext;
  end
endmodule
```

Пример 7. Параметризированная модель для счетчика, работающего в коде Грея

Для того чтобы сравнить значения указателей в коде Грея, можно использовать ту же самую методику, добавляя конвертеры между указателями и логикой двоичного сравнения.

При такой логике работы указателей FIFO работает очень быстро, и схемы, работающие с этим FIFO, могут выполнять операции чтения или записи в порядке поступления данных на каждом тактовом цикле. Однако доступ FIFO на каждом цикле означает, что сигналы состояния для FIFO должны представлять следующие состояния — «почти полный» и «почти пустой», так, чтобы схемы, обращающиеся в FIFO, имели время, чтобы обработать эти состояния и остановиться. Статус «почти полный» указывает на то, что один вход доступен для записи, а «почти пустой» указывает на то, что одно слово, хранящееся в FIFO, остается непрочитанным.

Эта ситуация описывает проект FIFO, который требует наименьшего количества возможных сигналов состояния работы FIFO. Но, если схемы, работающие с FIFO, используют доступ типа burst с известным минимальным размером циклов передач по шине, то проект нуждается в большем количестве индикаторов.

Эта методика для сигналов состояния FIFO дает пессимистическую картину сигналов состояния, как для чтения, так и для записи.

Сигнал состояния порта записи указывает на то, что FIFO заполнен, когда FIFO заполняется данными, и продолжает указывать, что FIFO заполнен, и после того, как будет произведено чтение в порте чтения, потому что синхронизация задерживает прохождение информации о состоянии указателя чтения на сторону записи для выполнения операции сравнения.

Эта ситуация также имеет место для состояния, когда FIFO будет пуст на стороне чтения, потому что синхронизация задерживает прохождение информации о состоянии указателя записи на сторону чтения для выполнения операции сравнения.

Выводы

Риск потери информации при передаче данных между схемами, находящимися в различных доменах синхрочастоты, значительно уменьшается в том случае, когда для разработки проекта применяются рассмотренные в этом разделе методики по обработке сигналов, пересекающих различные домены синхрочастоты. Синхронизацию необходимо использовать для того, чтобы предотвращать метастабильность триггеров, управляющихся сигналами, приходящими из других доменов синхрочастоты, и чтобы эта метастабильность не приводила к непредсказуемому поведению в схемах. Синхронизаторы, работающие по уровню, хорошо работают для сигналов, которые остаются в активном состоянии для нескольких тактовых циклов. В более медленном домене синхрочастоты необходимо использовать синхронизатор, работающий по фронту для сигналов уровня. Эти сигналы в новом, более быстром домене синхрочастоты будут преобразованы в импульсы. И для импульсов, пересекающих домены синхрочастоты, необходимо использовать синхронизаторы, работающие по импульсу. Помните, что, когда шина сигнала пересекает домены синхрочастоты, она должна «прибыть» в новый домен синхрочастоты в течение того же самого тактового цикла. Нет необходимости синхронизировать каждый сигнал шины отдельно. Для того чтобы синхронизировать все сигналы шины, необходимо использовать регистр для временного хранения информации и дополнительные сигналы, обеспечивающие процедуру установления связи. Сигналы, обеспечивающие процедуру установления связи, указывают, когда сигналы в регистре временного хранения информации будут достоверны и когда их можно обрабатывать. Использование сигналов для процедуры установления связи и регистра для временного хранения информации применяется для синхронизации шин данных, но не предусматривает прохождение более одного информационного слова одновременно к новому домену синхрочастоты.

Заключение

Автор надеется, что предложенный курс по программированию на HDL языке окажется вам полезен. Все замечания и предложения направляйте автору по почте **iosifk@narod.ru** или в редакцию журнала «Компоненты и технологии».

Литература

1. Cummings. C. E. Synthesis and Scripting Techniques for Designing Multi-Asynchronous Clock Designs. Synopsys Users Group Conference, San Jose, CA, **www.sunburst-esign.com/papers**, March 2001.
2. Clock Domain Crossing. Closing the Loop on Clock Domain Functional Implementation Problems. Technical paper. Cadence Design Systems, Inc. **www.cadence.com**
3. Metastability in Lattice Devices. Technical Note TN1055. July 2007.
4. Metastability in Altera Devices, ver. 4. Application Note 42. May 1999.
5. Metastability Considerations. Application Note (Version 1.0). XAPP077. January, 1997.
6. Metastability Characterization Report for Actel Flash FPGAs. **http://www.actel.com**
7. Johnson H. W., Graham M. High Speed Digital Design (A Handbook of Black Magic). Prentice-Hall, 1993.
8. Stein M. Crossing the abyss: asynchronous signals in a synchronous world. Paradigm Works. July 24, 2003. **www.edn.com**