Продолжение. Начало в № 3 `2008

Краткий курс HDL. Часть 10. Несколько слов о «сбросе»

Иосиф КАРШЕНБОЙМ
iosifk@narod.ru

Сигнал «сброс» используется для того, чтобы...

… привести аппаратуру в исходное состояние. И применяется данный сигнал как в одном отдельно взятом схемном «кирпичике» — триггере, так и в более крупных блоках и узлах аппаратуры: в микросхемах, модулях и в устройствах в целом.

В этом разделе будут рассмотрены некоторые аспекты применения сигнала сброса в различных узлах схемы, а также обработка сигнала сброса для работы в синхронном и асинхронном режимах. Методика привязки асинхронного сигнала сброса может быть распространена и на привязку любых других асинхронных сигналов.

Синхронный сброс — это сигнал, который переводит триггер в состояние «сброс» только на активном фронте синхрочастоты. Сброс может быть добавлен к триггеру как часть комбинационной логики, генерирующей входные данные на входе D у триггера (рис. 1). В этом случае для того, чтобы моделировать сброс, необходимо добавить условие **if/else**. И в секции **if** описать условие для сброса, а в секции **else** — всю остальную комбинационную логику, необходимую для работы триггера. Если этот стиль не соблюден строго, то могут возникнуть две проблемы. Первая — в некоторых симуляторах, основанных на логических уравнениях, логика, которую синтезирует симулятор, может при некоторых соотношениях сигналов на входе D заблокировать прохождение сигнала сброса к входу триггера. Хотя данная проблема может возникать только при моделировании, но, тем не менее, лучше с ней не сталкиваться. Вторая проблема в том, что сигнал сброса добавляется к комбинационной логике на входе D, и из-за увеличения этой логики увеличится и время прохождения данных через всю цепь. Если же говорить о FPGA, то такое выполнение сброса приведет к дополнительному увеличению числа интерконнектов.

Этот стиль синхронного сброса можно использовать с любой логикой и с любой библиотекой. В примерах 1, 2 показано выполнение синхронного сброса как часть комбинационной логики на входе загрузки данных — D для счетчика с загрузкой данных.

Необходимо также отметить, что в некоторых библиотеках, которые поставляют изготовители микросхем, могут присутствовать компоненты с отдельно выполненным входом синхронного сброса. Если таковые компоненты имеются, то использование их синхронных входов для сброса предпочтительнее, чем объединение сброса с комбинационной логикой на входах загрузки данных.

```
module ctr8sr (q, co, d, ld, rst_n, clk);
      output [7:0] q;
      output co;
      input [7:0] d;
      input ld, rst_n, clk;
      reg [7:0] q;
      reg co;
   always @(posedge clk)
      if (!rst_n)
         {co,q} <= 9'b0; // sync reset
      else if (ld)
         {co,q} <= d; // sync load
      else
         {co,q} <= q + 1'b1; // sync increment
endmodule
```

Пример 1. Код на Verilog для загружаемого счетчика с синхронным сбросом

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;
entity ctr8sr is
port (
      clk    : in std_logic;
      rst_n : in std_logic;
      d      : in std_logic;
      ld     : in std_logic;
      q      : out std_logic_vector(7 downto 0);
      co     : out std_logic);
end ctr8sr;
architecture rtl of ctr8sr is
   signal count : std_logic_vector(8 downto 0);
begin
   co <= count(8);
   q <= count(7 downto 0);

   process (clk)
   begin
      if (clk'event and clk = '1') then
         if (rst_n = '0') then
            count <= (others => '0'); -- sync reset
         elsif (ld = '1') then
            count <= '0' & d; -- sync load
         else
            count <= count + 1; -- sync increment
            end if;
         end if;
      end process;
end rtl;
```

Пример 2. Код на VHDL для загружаемого счетчика с синхронным сбросом

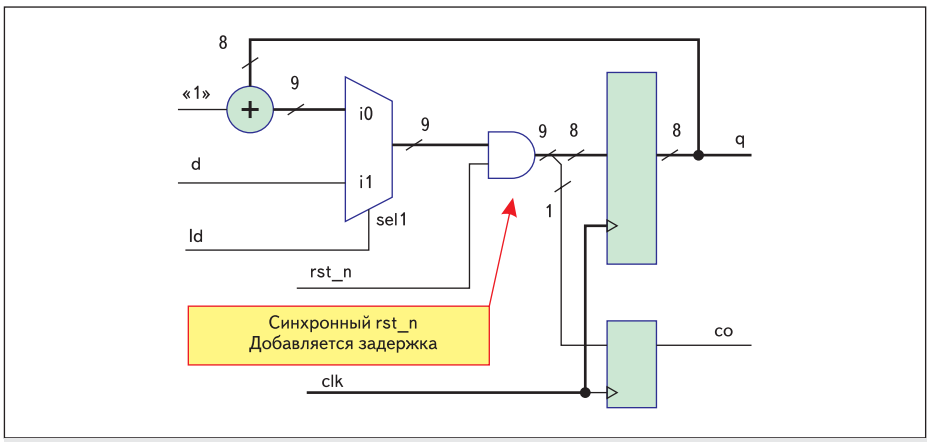

Рис. 1. Загружаемый счетчик с синхронным сбросом

Стиль кодирования и схема триггера с синхронным сбросом

В данном разделе еще раз приводится код для Verilog (пример 3) и код для VHDL (пример 4), показывающие правильный способ моделирования синхронного сброса в триггере. Отметьте, что сброс не входит в список чувствительности. Как мы знаем, именно по изменению тех сигналов, которые находятся в списке чувствительности, симулятор и определяет новое значение переменных. Поэто-

```
module sync_resetFFstyle (q, d, clk, rst_n);
                output q;
                input d, clk, rst_n;
                reg q;
                always @(posedge clk)
                if (!rst_n) q <= 1'b0;
                else q <= d;
endmodule
```

Пример 3. Правильный способ моделирования триггера с синхронным сбросом на языке Verilog

```
library ieee;
use ieee.std_logic_1164.all;
entity syncresetFFstyle is
port (
      clk    : in std_logic;
      rst_n : in std_logic;
      d      : in std_logic;
      q      : out std_logic);
end syncresetFFstyle;
architecture rtl of syncresetFFstyle is
   begin
   process (clk)
   begin
      if (clk'event and clk = '1') then
         if (rst_n = '0') then
            q <= '0';
         else
            q <= d;
         end if;
      end if;
   end process;
end rtl;
```

Пример 4. Правильный способ моделирования триггера с синхронным сбросом на языке VHDL

му для языка Verilog исключение сброса из списка чувствительности показывает то, что этот сброс является синхронным, так как теперь симулятор производит новые назначения переменным только при изменении сигнала синхрочастоты, а не по изменениям сигнала сброса. Для VHDL исключение сброса из списка чувствительности и проверка сброса после возникновения события, описанного в утверждении **"if clk'event and clk = 1"** делают сброс синхронным. Также отметьте, что сбросу уделяют самый высокий среди всех назначений приоритет в обслуживании, и для этого используют стиль кодирования **if-else**.

Для триггеров, спроектированных с синхронным сбросом, когда сигнал сброса добавляется к данным и поступает на вход D, «Синопсис» имеет директиву компилятора **sync_set_reset**, которую проектировщик может использовать для того, чтобы указать компилятору на применение триггеров с синхронными сбросами.

Преимущества синхронных сбросов

Что касается сокращения логики в случае применения синхронного сброса, то есть числа вентилей, задействованных для обработки этого сигнала, то для каждого проекта это действие должно быть рассмотрено отдельно. Самое существенное здесь то, что синхронный сброс намного упрощает работу симулятора, при этом, естественно, сигнал сброса привязывают к временной сетке симулятора. Синхронный сброс позволяет получить на 100% синхронную схему. Таким образом, синхронный сброс — это такой сброс, который может произойти только на активном фронте синхроимпульса. Как и во всех других синхронных схемах, синхрочастота работает как фильтр для подавления очень коротких сбойных импульсов сброса. Но если эти сбои происходят около активного фронта синхроимпульса, то такие импульсы могут привести триггер в метастабильное состояние.

В случае применения в проекте нескольких синхрочастот можно использовать несколько сигналов для синхронного сброса, так, чтобы каждый сигнал в своем клоковом домене полностью бы удовлетворял всем требованиям по **setup/hold**.

Недостатки синхронных сбросов

В случае применения синхронного сброса разработчик аппаратуры должен быть уверен, что исходный сигнал, приходящий на вход «сброс», не нуждается в расширителе импульса для того, чтобы гарантировать достаточную продолжительность импульса сброса. Длительность приходящего импульса должна быть не меньше, чем период синхрочастоты, и импульс сброса должен присутствовать в течение активного фронта синхрочастоты.

Еще одна проблема может заключаться в том, что при разработке комбинационной логики часть сигналов разработчик имеет право указать как сигналы, находящиеся в неопределенном состоянии — X. И тогда такие сигналы могут замаскировать сигнал сброса.

Асинхронный сброс

Это наиболее широко применяемый способ для установки цифровых устройств в исходное состояние. Однако асинхронные сбросы могут быть очень опасными. Самая большая проблема состоит в том, как подать на схему и как снять с нее асинхронный сигнал. Более подробно это будет рассмотрено в следующих разделах. Как правило, дополнительные входы асинхронного сброса и установки входят в состав триггеров. Вход сброса обычно имеет активный низкий уровень.

Стиль кодирования и схема примера

В примерах 5, 6 показаны коды для Verilog и VHDL. В этих примерах показаны правильные способы моделирования асинхронного сброса в триггере. В этом случае сигнал сброса уже входит в список чувствительности. Для языка Verilog добавление сигнала сброса к списку чувствительности заставляет симулятор сделать новое назначение переменным при изменении сигнала сброса, что и делает этот сигнал асинхронным. Для того чтобы правильно выполнить моделирование для асинхронного триггера в Verilog, список чувствительности должен быть только активным на переднем фронте асинхронного сигнала сброса. Следовательно, в примере 5 блок процедуры **always** будет обновляться по переднему фронту сигнала сброса, а условный оператор **if** проверит правильный уровень этого сигнала.

Итак, теперь у нас в списке чувствительности будет не менее двух сигналов, запускаемых по переднему фронту. Здесь надо сделать уточнение. Ведущий производитель программных средств, фирма «Синопсис» в своих программных инструментах вводит следующее ограничение: если хотя бы один из сигналов, присутствующих в списке чувствительности, будет чувствителен к фронту, то все сигналы в списке чувствительности должны быть чувствительными только к фронту. Другие стили кодирования могут привести к ошибке. Более того, только синхрочастота и сигналы сброса могут быть в списке чувствительности. Если бы другие сигналы были включены в этот список чувствительности, то симуляционная модель не была бы воспринята как правильная модель для триггера, и «Синопсис» выдал бы сообщение об ошибке.

Для VHDL, включая сигнал сброса в список чувствительности и проверяя сброс перед утверждением **"if clk'event and clk = 1"**, разработчик делает сброс асинхронным. Также необходимо отметить, что назначение по сигналу сброса имеет более высокий приоритет, чем любое другое назначение (включая синхрочастоту), и для этого используют стиль кодирования **if/else**. Поскольку список чувствительности и стиль кодирования триггера в VHDL отличаются от аналогичного в Verilog, то дополнительные сигналы могут быть включены в список чувствительности, и они не будут иметь негативных последствий для моделирования и синтеза (пример 6). Однако рекомендуется, чтобы в списке чувствительности были только те сигналы, ко-

```
module async_resetFFstyle (q, d, clk, rst_n);
   output q;
   input d, clk, rst_n;
   reg q;
   // Verilog-2001: permits comma-separation
   // @(posedge clk, negedge rst_n)
   always @(posedge clk or negedge rst_n)
   if (!rst_n)
      q <= 1'b0;
   else
      q <= d;
endmodule
```

Пример 5. Правильный способ моделирования триггера с асинхронным сбросом на языке Verilog

```
library ieee;
use ieee.std_logic_1164.all;
entity asyncresetFFstyle is
   port (
      clk    : in std_logic;
      rst_n : in std_logic;
      d      : in std_logic;
      q      : out std_logic);
end asyncresetFFstyle;

architecture rtl of asyncresetFFstyle is
begin
   process (clk, rst_n)
      begin
      if (rst_n = '0') then
         q <= '0';
      elsif (clk'event and clk = '1') then
         q <= d;
      end if;
   end process;
end rtl;
```

Пример 6. Правильный способ моделирования триггера с асинхронным сбросом на языке VHDL

торые могут непосредственно изменить выход триггера. А именно только синхрочастота и асинхронный сброс. Все другие сигналы замедлят моделирование и будут игнорироваться при синтезе.

Моделирование триггеров с асинхронным сбросом и асинхронной установкой в Verilog

Необходимо сделать еще одно дополнительное замечание относительно моделирования асинхронных сбросов в Verilog. Симуляционная модель триггера, который имеет и асинхронный сброс, и асинхронную установку в Verilog, не может быть создана правильно без небольшой помощи проектировщика. Вообще, обычно синхронные проекты не имеют триггеров с асинхронными сбросом и установкой, но в ряде случаев такой триггер может потребоваться. В примере 6 показан образец для кодирования такого триггера, и он используется, чтобы исправить Verilog RTL-модель, где и сброс, и установка могут быть выданы одновременно, и при этом сброс снимается первым.

Для тех редких проектов, где должны быть оба сигнала — и сброс, и установка, и в том случае, если устанавливаются одновременно оба сигнала, а затем сигнал сброса будет снят первым, для устранения описанной проблемы моделирования необходимо в модели триггера использовать дополнительный корректирующий код, который будет включен в пределах директив **translate_off/translate_on**. Этот код использует директивы **translate_off/translate_on** и позволяет перевести выход триггера в правильное для этого условия значение при помощи директивы force. Конечно, лучше, если есть возможность избежать применения триггера, использующего и асинхронный сброс, и асинхронную установку. В примере 7 показан код, в котором проблема с двумя асинхронными сигналами решена, и этот код моделируется правильно и гарантирует соответствие между результатами моделирования до и после синтеза.

```
// Good DFF with asynchronous set and reset and self-
      // correcting set-reset assignment
      module dff3_aras (q, d, clk, rst_n, set_n);
   output q;
   input d, clk, rst_n, set_n;
   reg q;

   always @(posedge clk or negedge rst_n or negedge set_n)
      if (!rst_n) q <= 0; // asynchronous reset
      else if (!set_n) q <= 1; // asynchronous set
      else q <= d;
   // synopsys translate_off
   always @(rst_n or set_n)
   if (rst_n && !set_n) force q = 1;
   else release q;
      // synopsys translate_on
Endmodule
```

Пример 7. Код в Verilog для моделирования и синтеза асинхронных сигналов установки/сброса

Преимущества асинхронных сбросов

Самое большое преимущество при использовании асинхронных сбросов состоит в том, что пока библиотека примитивов, поставляемая разработчиками микросхем FPGA или кремниевым ателье, имеет триггеры с асинхронными входами, цепи данных будут отделены от цепей сброса. В проектах, которые выполняются на предельных частотах и имеют критические времена для прохождения данных, нет возможности добавлять логику и, соответственно, дополнительные задержки в тракте передачи данных из-за этой логики, чтобы обработать синхронные сбросы. Конечно, этот аргумент не работает, если библиотека примитивов уже имеет триггеры с синхронными входами сброса, и проектировщик может заставить компилятор использовать эти входы. Используя асинхронный сброс, проектировщик не добавит к тракту данных дополнительную логику, обрабатывающую сигнал сброса (рис. 2). В примерах 8, 9 показан код, обрабатывающий асинхронные сбросы, которые гарантированно не будут добавлены к тракту передачи данных.

Другое преимущество асинхронных сбросов состоит в том, что схема может выполнить сброс в исходное состояние даже без наличия сигнала синхрочастоты. Таким образом, при использовании стиля кодирования для асинхронных сбросов, описанных в этом разделе, синтезатор автоматически компилирует проект. Значит, нет вообще никакой необходимости добавлять дополнительные команды для синтеза, чтобы заставить программный инструмент синтезировать триггер с асинхронным входом сброса.

```
module ctr8ar (q, co, d, ld, rst_n, clk);
   output [7:0] q;
   output co;
   input [7:0] d;
   input ld, rst_n, clk;
   reg [7:0] q;
   reg co;
   always @(posedge clk or negedge rst_n)
      if (!rst_n)
         {co,q} <= 9'b0; // async reset
      else if (ld)
         {co,q} <= d;  // sync load
      else
         {co,q} <= q + 1'b1; // sync increment
Endmodule
```

Пример 8. Verilog-код для загружаемого счетчика с асинхронным сбросом

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;
entity ctr8ar is
   port (
      clk    : in std_logic;
      rst_n : in std_logic;
      d      : in std_logic;
      ld     : in std_logic;
      q      : out std_logic_vector(7 downto 0);
      co     : out std_logic);
end ctr8ar;
architecture rtl of ctr8ar is
   signal count : std_logic_vector(8 downto 0);
   begin
      co <= count(8);
      q <= count(7 downto 0);
   process (clk)
   begin
      if (rst_n = '0') then
         count <= (others => '0'); -- sync reset
      elsif (clk'event and clk = '1') then
         if (ld = '1') then
            count <= '0' & d; -- sync load
         else
            count <= count + 1; -- sync increment
         end if;
      end if;
   end process;
end rtl;
```

Пример 9. VHDL-код для загружаемого счетчика с асинхронным сбросом

Недостатки асинхронных сбросов

Существует много причин, почему асинхронные сбросы могут привести к неправильному функционированию проекта.

Для микросхем, имеющих цепи JTAG, обычно делают отдельную цепь асинхронного сброса, которая связана с внутренней логикой микросхемы. И во время граничного сканирования цепи сброса отключаются от ядра микросхемы. Также необходимо учитывать, что при использовании асинхронных сбросов статический временной анализ схемы выполнить труднее. При выдаче сигнала

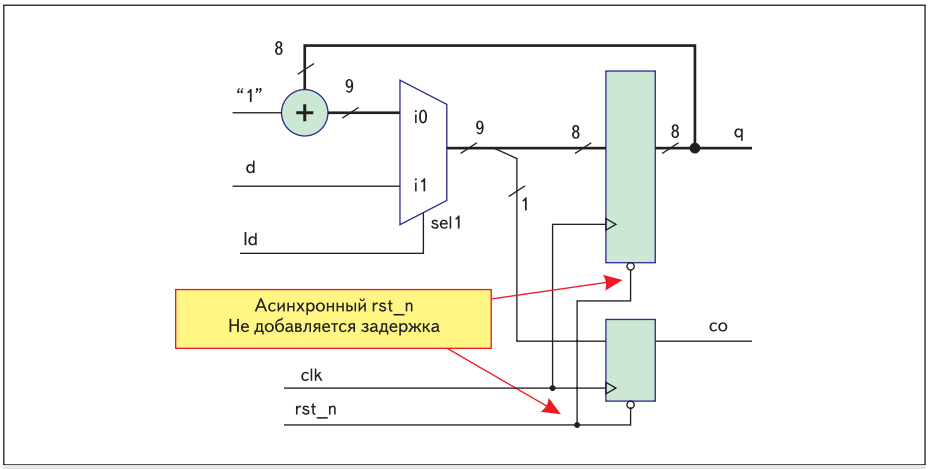

Рис. 2. Загружаемый счетчик с асинхронным сбросом

сброса необходимо гарантировать, что и для синхронных, и для асинхронных сбросов сигнал сброса будет активен, как минимум, в пределах одного периода синхрочастоты. Поскольку сигнал сброса должен быть подан из одной точки ко многим триггерам, то он имеет большое разветвление (fanout), и под сигнал сброса обычно задействуется одна из глобальных цепей внутри кристалла. Временной анализ для такой цепи сброса должен быть выполнен после того, как произведено размещение проекта пользователя на кристалле. Это необходимо для того, чтобы гарантировать, что все требования по синхронизации соответствуют заданным. И самая большая проблема с асинхронными сбросами состоит именно в том, что они являются асинхронными, и во время активного состояния сигнала, и в момент снятия активного состояния сигнала сброса. Причем, если установка активного состояния сигнала не представляет трудностей, то момент снятия активного состояния сигнала сброса — это действительно проблема для разработчика. Рассмотрим это более подробно.

Предположим, что у нас в проекте есть импульсы синхрочастоты. Теперь представим, что мы подаем сигнал сброса. Если асинхронный сброс выдан во время фронта сигнала синхрочастоты или перед этим активным фронтом синхроимпульса триггера, но время перед фронтом синхрочастоты меньше, чем время предустановки, то выход триггера мог войти в метастабильное состояние (рис. 3). Но уже при истечении времени, необходимого для перехода в состояние сброса, триггер перейдет в это состояние. А вот при снятии сигнала сброса ситуация может быть совсем другая. При снятии сигнала сброса во время фронта сигнала синхрочастоты или перед этим активным фронтом синхроимпульса триггера, если время перед фронтом синхрочастоты меньше, чем время предустановки, то выход триггера также может перейти в метастабильное состояние. И, таким образом, состояние схемы станет неопределенным.

Еще одна проблема асинхронного сброса состоит в том, что вход асинхронного сброса может быть чувствителен к ложным сбросам, которые могут возникнуть из-за шума или сбоев на плате, либо сбоев в системном сбросе. Хотя необходимо заметить, что подобная проблема существует и для синхронных сбросов в том случае, если сбойные импульсы сброса происходят около фронта синхроимпульса: триггеры также могут перейти в метастабильное состояние.

Конечно, изготовители FPGA принимают все меры для того, чтобы триггеры микросхемы устойчиво переходили в состояние сброса. И все же многие инженеры используют асинхронный сброс, не принимая никаких мер для надежного срабатывания сброса, так как они думают, что никаких проблем в этом месте нет. Они проверяют сброс в среде моделирования, где все работает прекрасно, но иногда в «железе» их проект не приходит в исходное состояние. При этом они не рассматривают ситуацию, что сигнал сброса в реальной системе мог заставить микросхему входить в метастабильное неизвестное состояние и, таким образом, не производить правильный сброс. Необходимо более строго связать установление и снятие сигнала сброса, чтобы препятствовать микросхеме входить в метастабильное неизвестное состояние. То же самое можно сказать и о синхронном сбросе: он должен выдаваться и сниматься в соответствии с заданными временными характеристиками на предустановку и удержание сигнала.

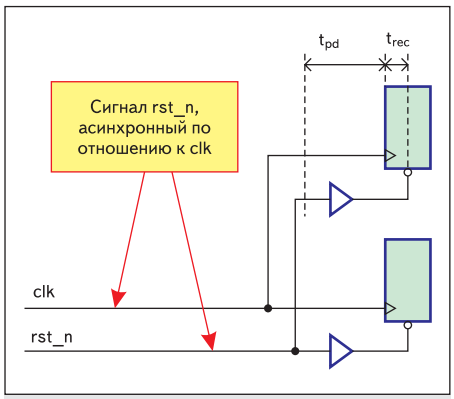

Рис. 3. Асинхронная проблема времени восстановления удаления сброса

Время восстановления сигнала «сброс»

Время восстановления по сигналу сброса представляет собой время между тем моментом времени, когда снимается сигнал сброса, и временем, когда появляется активный фронт сигнала синхрочастоты . Стандарт Verilog-2001 (см. [1] в КиТ № 3 '2008) имеет три встроенных команды, для моделирования и проверки времени восстановления и снятия сигнала: $recovery, $removal и $recrem (последняя команда — это комбинация команд восстановления и удаления сигнала). Нарушение времени восстановления сигнала сброса может вызвать проблемы целостности сигналов или проблемы метастабильности данных, защелкнутых в регистрах.

Неравномерность прохождения сигнала «сброс» по кристаллу

Когда снятие сигнала сброса является асинхронным по отношению к активному фронту синхроимпульса, то небольшие различия в задержках распространения, как самого сигнала сброса, так и сигнала синхрочастоты, могут заставить некоторые регистры или триггеры выходить из состояния сброса раньше других.

Синхронизатор сброса

Рекомендация: каждая ASIC-микросхема, имеющая асинхронный сброс, должна включать в себя схему синхронизатора сброса.

В реально работающем устройстве невозможно получить правильно функционирующий сброс без специального узла — синхронизатора сброса, даже если сигнал сброса работает при моделировании. Логика работы синхронизатора сброса показана на рис. 4.

Внешний сигнал «сброс» — это сигнал, получаемый на входе кристалла, — pad_rst_n, он обрабатывается в два этапа. Сигнал сброса поступает на входы Reset двух специальных триггеров, включенных каскадно. На первом этапе приходящий извне сигнал сбрасывает асинхронно эти два триггера. Триггеры, в свою очередь, управляют главной цепью сброса, проходящей через дерево буферов сброса к остальной части триггеров в проекте. И, таким образом, при подаче асинхронного сброса все триггеры в проекте будут асинхронно сброшены в исходное состояние.

При снятии сигнала сброса первый из триггеров, на вход D которого подана единица, перейдет в высокий уровень. На следующем фронте синхрочастоты и второй триггер га-

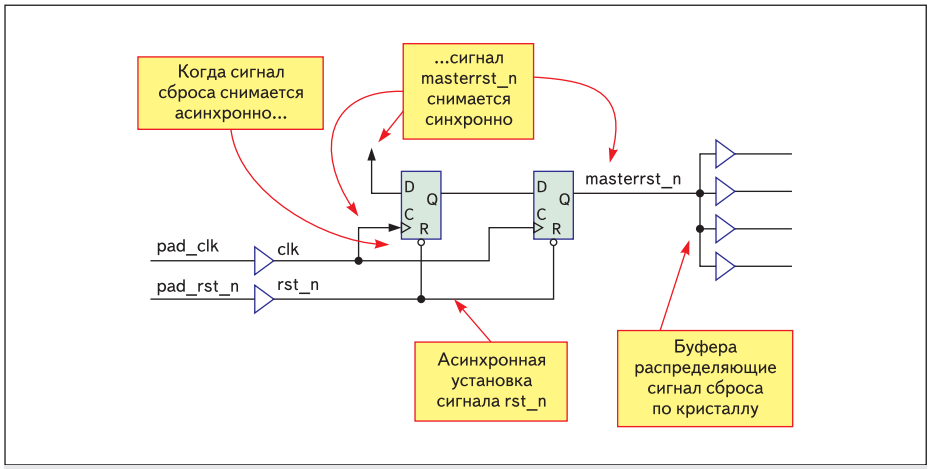

Рис. 4. Блок-схема синхронизатора сброса

```
module async_resetFFstyle2 (rst_n, clk, asyncrst_n);
   output rst_n;
   input clk, asyncrst_n;
   reg rst_n, rff1;

always @(posedge clk or negedge asyncrst_n)
   if (!asyncrst_n) {rst_n,rff1} <= 2'b0;
   else {rst_n,rff1} <= {rff1,1'b1};

endmodule
```

Пример 10. Пример кода для схемы синхронизатора сброса на языке Verilog

```
library ieee;
      use ieee.std_logic_1164.all;
      entity asyncresetFFstyle is
   port (
         clk          : in std_logic;
         asyncrst_n   : in std_logic;
         rst_n        : out std_logic);
end asyncresetFFstyle;

architecture rtl of asyncresetFFstyle is
   signal rff1 : std_logic;
begin
      process (clk, asyncrst_n)
      begin
            if (asyncrst_n = '0') then
               rff1 <= '0';
               rst_n <= '0';
            elsif (clk'event and clk = '1') then
               rff1 <= '1';
               rst_n <= rff1;
            end if;
         end process;
end rtl;
```

Пример 11. Пример кода для схемы синхронизатора сброса на языке VHDL

рантированно перейдет в состояние единицы. Второй триггер используется для того, чтобы устранить метастабильность, которая могла бы быть вызвана сигналом сброса в том случае, когда снятие сигнала асинхронного сброса происходит слишком близко к активному фронту синхроимпульса. Таким образом, после снятия сигнала сброса потребуется два активных фронта синхрочастоты для того, чтобы синхронизировать снятие внешнего сигнала «сброс». Коды для схемы синхронизатора сброса показаны в примере 10 и 11.

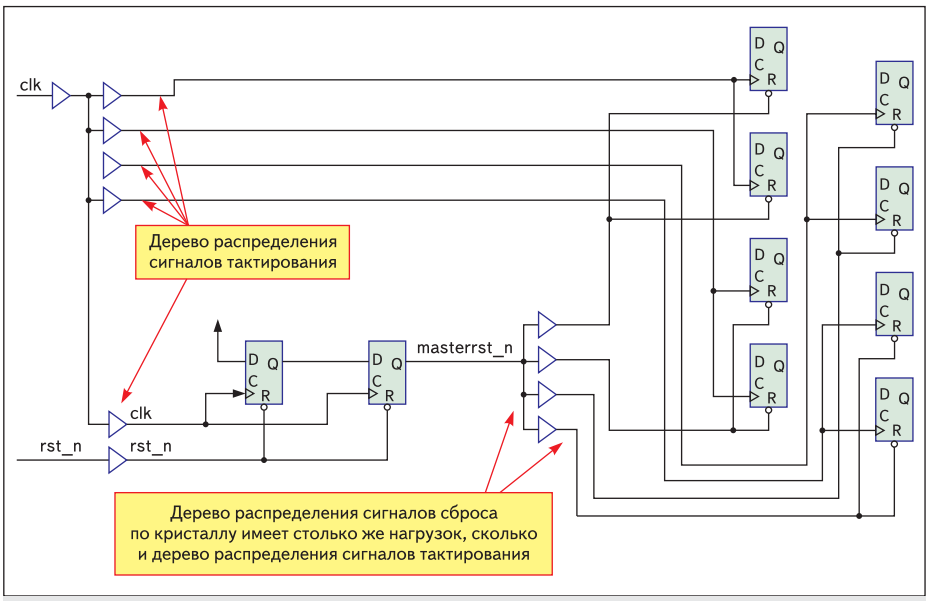

Рис. 5. Дерево распределения сигнала сброса

Дерево распределения сигнала «сброс»

Дерево распределения сигнала сброса требует почти такого же большого внимания, как и дерево распределения сигнала синхрочастоты, потому что в любом проекте входов для подключения сигналов сбросов практически столько же, сколько и для подключения входов синхрочастоты, как это показано на рис. 5. Требования синхронизации для дерева сбросов одинаковы и для синхронного, и для асинхронного стилей сброса.

Но есть одно важное различие между деревом распределения синхрочастоты и деревом распределения сброса: необходимо свести к минимуму перекос между распределенными по кристаллу сигналами сброса. В отличие от сигналов синхрочастоты, перекос между сигналами сброса не является критическим до тех пор, пока задержка, связанная с любым сигналом сброса, достаточно мала по сравнению с периодом синхрочастоты, и для всех регистров и триггеров выдерживаются требования по времени снятия сигнала сброса.

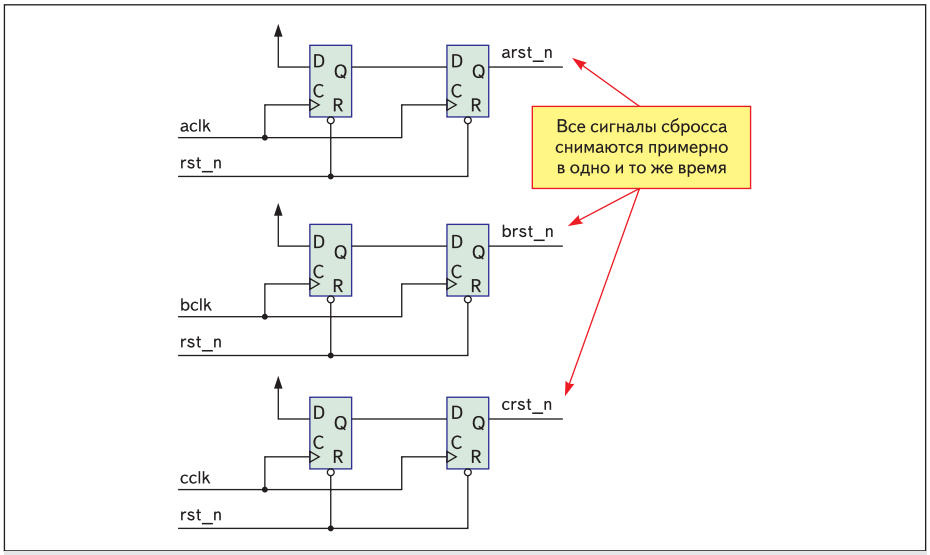

Рис. 6. Все сигналы сброса снимаются в одно и то же время

Некоординированное снятие сброса

Мы рассмотрели проблему по обработке сигнала сброса, а конкретно — наличие только одного такого сигнала в проекте. Но в реальности, кроме этого случая, есть и другие. Например, в проекте есть несколько тактовых частот. Тогда, выполняя привязку сигнала сброса к каждой из синхрочастот (рис. 6), мы получим неизбежное рассогласование снятия сигналов сброса в разных доменах синхрочастот.

Снятие сброса в определенном порядке

Необходимо также рассмотреть, что произойдет в проекте при снятии сигнала сброса в разные промежутки времени. Мало того, в проекте пользователя может быть определен порядок, в котором необходимо снимать сигналы сброса с разных частей схемы.

На рис. 7 показано, как можно организовать снятие сигналов сброса в определенном порядке. Видно, что сигналы arst_n, brst_n и crst_n устанавливаются в состояние «сброс» одновременно, а снимаются из этого состояния один за другим.

Заключение

Используя асинхронный сброс, разработчик может гарантированно привести схему в состояние сброса. Но, хотя асинхронный сброс — это безопасный путь, который на-

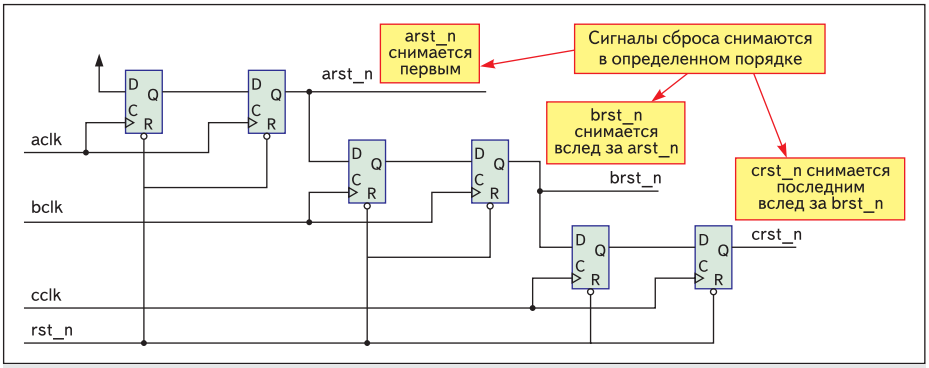

Рис. 7. Сигналы сброса снимаются в определенном порядке

дежно приводит схему в состояние сброса, снятие сигнала асинхронного сброса может вызвать существенные проблемы в том случае, если проект не сделан должным образом.

Правильный способ проектирования схемы с асинхронными сбросами состоит в том, чтобы применять логику синхронизатора для сброса. Это позволяет снимать асинхронный сброс синхронно с тактовой частотой и обеспечивает безопасное восстановление нормального функционирования схемы.

В следующем разделе мы рассмотрим вопросы, связанные с асинхронными частотами.

Литература

1. Cummings C. E., Mills D. Synchronous Resets? Asynchronous Resets? I am so confused! How will I ever know which to use? www.sunburst-design.com
2. Get Smart About Reset: Think Local, Not Global. WP272 (v1.0) January 8, 2008. www.xilinx.com