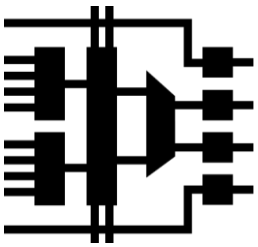

FPGA-SYSTEMS.RU

сообщество FPGA разработчиков

ПОДГОТОВИЛИ:

- Перевод: Михаил Коробков
- Иллюстрации: Николай Терновой

Асинхронный и синхронный сброс

Методы проектирования.

Часть II

Раздел 4

2021

Оглавление

Контакты автора

Коробков Михаил

e-mail:

KeisN13@fpga-systems.ru

телефон:

+7-929-955-68-75

Контакты комьюнити

сайт:

fpga-systems.ru

e-mail:

admin@fpga-systems.ru

Reset-Привет!

Перед вами вторая часть перевода одной из фундаментальных статей о проектировании и применении синхронного и асинхронного сбросов, которая появилась в далеком 2003 году за авторством Clifford E. Cummings, Don Mills и Steve Golson под названием Asynchronous & Synchronous Reset Design Techniques - Part Deux.

Перевод подготовлен для информационно-образовательного портала FPGA / ПЛИС разработчиков FPGA-Systems.ru

Исходные коды из статьи выложены на github

Приятного прочтения.

Дата	Версия	Автор	Изменения
24.05.2021	1.0	KeisN13	Начальный релиз документа

***О найденных опечатках и замечаниях просим сообщить admin@fpga-systems.ru**

Аннотация

См. здесь

1.0 Введение

См. здесь

2.0 Предназначение сброса

См. здесь

3.0 Общие примечания по стилю кодирования триггеров

См. здесь

4.0 Синхронный сброс

По мере проведения исследований для этой статьи был собран и рассмотрен сборник статей ESNUG и SOLV-IT. Около 80% собранных статей были посвящены вопросам синхронного сброса. Во многих статьях из SNUG утверждалось что-то вроде: “Мы все знаем, что лучший способ выполнить сброс в ASIC – это строго использовать синхронные сбросы”, или, возможно, “асинхронные сбросы плохи, и их следует избегать”. Тем не менее, было представлено мало доказательств, подтверждающих эти заявления. Использование синхронных или асинхронных сбросов имеет как преимущества, так и недостатки. Дизайнер / разработчик должен использовать подход, соответствующий определенному дизайну / проекту.

Синхронные сбросы основаны на предположении, что сигнал сброса будет влиять на состояние триггера (сбрасывать его) только при активном фронте тактового сигнала. Сброс может быть применен к триггеру как часть комбинационной логики, формирующей сигнал на D-входе триггера. Если это так, то стиль кодирования для описания сброса должен быть описан с помощью приоритетного **if**/**else** со сбросом в условии **if** и всей другой комбинационной логикой в ветке **else**. Если этот стиль не соблюдается строго, могут возникнуть две возможные проблемы. Во-первых, в некоторых симуляторах, основанных на логических уравнениях, логика может блокировать сброс и он не достигнет триггера. Однако, это проблема моделирования, а не аппаратная проблема, но помните, что одна из главных целей сброса – привести ASIC

в известное состояние для выполнения моделирования. Во-вторых, сброс может “запаздывать” относительно тактового сигнала из-за высокого ветвления цепей сброса. Даже если сброс будет буферизован с помощью специального буфера (буфера сброса), разумно ограничить объем логики, которая должна быть сброшена. Этот стиль описания синхронного сброса можно использовать с любой логикой или библиотекой. В примере 3 показана реализация синхронного сброса как части счетчика с загрузкой данных с функцией переноса (carry out).

```
module ctr8sr (
    output reg [7:0] q,
    output reg co,
    input [7:0] d,
    input ld, clk, rst_n);

    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 3а - Код Verilog-2001 для загружаемого счетчика с синхронным сбросом (git)

```
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 3б - код VHDL для загружаемого счетчика с синхронным сбросом (git)

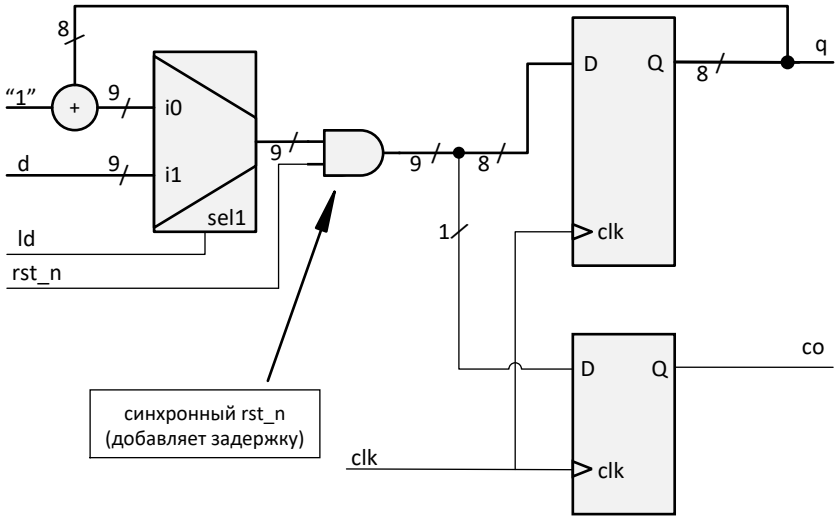

Рисунок 3 - Загружаемый счетчик с синхронным сбросом

4.1 Стиль кодирования и пример схемы

Код Verilog в примере 4а и код VHDL в примере 4б показывают правильный способ описания триггеров с синхронным сбросом. Обратите внимание, что сброс не является частью списка чувствительности. Для Verilog исключение сброса из списка чувствительности – это то, что делает сброс синхронным. Для VHDL удаление сброса из списка чувствительности и проверка сброса после оператора "**if clk'event и clk = 1**" делает сброс синхронным. Также обратите внимание, что сброс имеет приоритет над любым другим присвоением при использовании стиля кодирования **if-else**.

```
module sync_resetFFstyle (
    output reg q,
    input d, clk, rst_n);

    always @(posedge clk)
        if (!rst_n) q <= 1'b0;
        else q <= d;
endmodule
```

Пример 4 а - Правильный способ описания триггера с синхронным сбросом на Verilog-2001 (git)

```vhdl
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 4б - Правильный способ описания триггера с синхронным сбросом на VHDL (git)

Одна из проблем с синхронными сбросами заключается в том, что синтезатор не может просто взять и отличить сигнал сброса от любого другого сигнала данных. Рассмотрим код из примера 3, который приводит к схеме на рис. 3. В качестве альтернативы синтезатор мог бы создать схему, показанную на рис. 4.

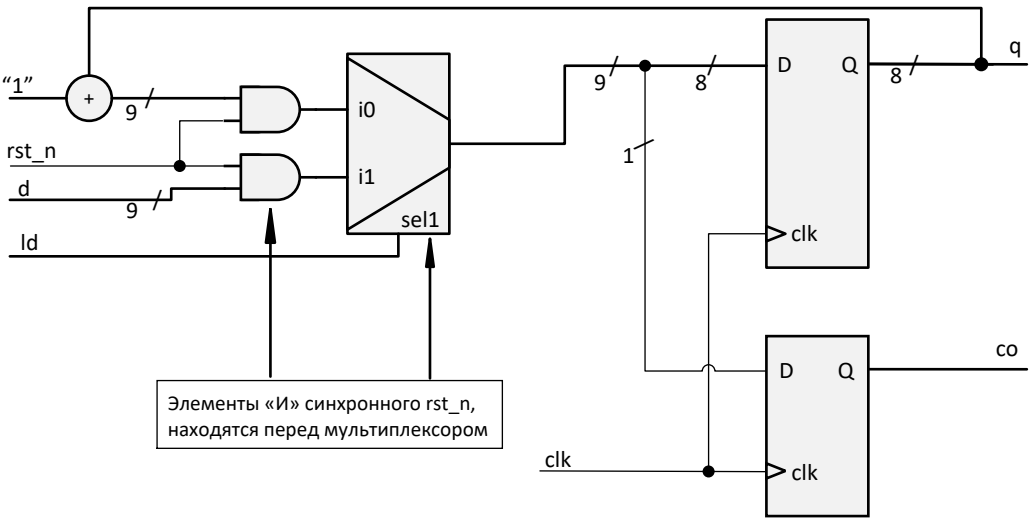

Рисунок 4 - Альтернативная схема загружаемого счетчика с синхронным сбросом

Эта схема функционально идентична схеме на рисунке 3. Единственная разница заключается в том, что элементы «И» сброса находятся вне мультиплексора. Теперь рассмотрим, что происходит в начале моделирования на уровне логических элементов.

Оба входа мультиплексора могут быть принудительно установлены в 0 при удержании **rst_n** в низком уровне, однако, если значение сигнала **ld** неизвестно (X), а модель мультиплексора пессимистична[1], то триггеры останутся в неизвестном состоянии (X), а не будут сброшены. Обратите внимание, что это проблема возникает только во время моделирования! Фактическая схема будет работать правильно и сбросит триггеры в 0.

Synopsys предоставляет директиву компилятора **sync_set_reset**, которая сообщает синтезатору, что данный сигнал является синхронным сбросом (или сигналом установки (set)). Синтезатор “подтянет” этот сигнал как можно ближе к триггеру, чтобы предотвратить возникновение этой проблемы. Директива может быть использована путем добавления следующей строки где-то внутри модуля:

// synopsys sync_set_reset "rst_n"

В общем, мы рекомендуем использовать атрибуты и директивы Synopsys только тогда, когда они необходимы и имеют значение; однако директива **sync_set_reset** не влияет на логическое поведение проекта, она влияет только на его функциональную реализацию. Опытный инженер предпочел бы избежать повторного синтеза на поздней стадии разработки проект и добавил бы директиву **sync_set_reset** ко всему RTL-коду на ранней стадии. Поскольку объявление ранее упомянутой директивы требуется только один раз, рекомендуется добавлять ее в каждый модуль с синхронными сбросами.

В качестве альтернативы решение можно использовать переменную синтеза **hdlin_ff_always_sync_set_reset**, значение которой следует установить в **true**, что даст тот же результат, не требуя вставки каких-либо директив в самом коде.

Несколько лет назад другой участник SNUG рекомендовал добавить переменную **compile_preserve_sync_resets = "true"** [15]. Хотя эта переменная могла быть полезна несколько лет назад, она была удалена Synopsys, начиная с версии 3.4 b [38].

1. Прим. пер.: подробнее про X-пессимизм:
   - https://sutherland-hdl.com/papers/2013-DVCon_In-love-with-my-X_paper.pdf
   - https://www.eetimes.com/x-pessimism-a-realistic-approach-is-needed/
   - https://www.techdesignforums.com/practice/technique/catch-x-propagation-issues-rtl

4.2 Преимущества синхронного сброса

Логика синхронного сброса будет синтезироваться в более компактные триггеры, особенно если сброс совмещается с логикой, генерирующей D-вход. Но в таком случае увеличивается число комбинационных логических элементов, поэтому общая экономия может быть не столь значительной. Однако, если дизайн микросхемы достаточно плотный, то экономия площади в один или два логических вентиля на триггер может повлиять на возможность размещения ASIC на подложке. Однако, при современных технологических нормах и больших размерах подложки экономия одного или двух логических вентилей на триггер, как правило, не играет важной роли и не будет существенным фактором того, уместится ли дизайн на подложке.

Использование синхронных сбросов обычно гарантирует, что схема на 100% синхронна.

Использование синхронных сбросов гарантирует, что сброс может произойти только на активном фронте тактового сигнала. Тактовый сигнал работает как фильтр при небольших глитчах сброса; однако, если эти сбои происходят вблизи активного фронта тактового сигнала, триггер может перейти в метастабильное состояние. Такое поведение ничем не отличается от любого другого ввода данных в триггер; любой сигнал, нарушающий требования по времени установления (setup), может привести к метастабильности.

В некоторых проектах сброс должен быть вызван набором внутренних условий. Для таких проектов рекомендуется использование синхронного сброса, поскольку он будет фильтровать глитчи, вызванные логикой формирования сброса (*прим. пер. эффектом гонок в комбинационных схемах*) между тактовыми сигналами

Согласно Руководству по методологии повторного использования (Reuse Methodology Manual - RMM)[32], с синхронными сбросами может быть проще работать при использовании симуляторов на основе циклов. По этой причине синхронные сбросы рекомендуются в разделе 3.2.4 (2-е издание, раздел 3.2.3 в 1-м издании) RMM. Мы считаем, что использование асинхронных сбросов с хорошим стилем кодирования тестбенчей, где сброс изменяется только на фронтах тактовых импульсов, устраняет любые преимущества в простоте моделирования или скорости, приписываемые

синхронным сбросам RMM. Примечание: сомнительно, что стиль сброса имеет большое значение, как в легкости, так и в скорости моделирования.

4.2 Недостатки синхронного сброса

Не все библиотеки ASIC имеют триггеры со встроенными синхронными сбросами. Однако, поскольку синхронный сброс – это просто еще один вход данных, вам действительно не нужен специальный триггер. Логика сброса может быть легко синтезирована вне самого триггера.

Для синхронных сбросов может потребоваться «удлинитель» импульсов, чтобы гарантировать достаточную длительность импульса сброса для обеспечения его наличия во время активного фронта тактового сигнала [16]. Это вопрос, который важно учитывать при работе с несколькими тактовыми доменами. Можно использовать небольшой счетчик, который будет обеспечивать достаточную длительность сигнала сброса.

Дизайнер должен уметь работать с проблемами симулятора. Потенциальная проблема существует, если сброс генерируется комбинационной логикой в ASIC или если сброс должен проходить через множество уровней локальной комбинационной логики. Во время моделирования, в зависимости от того, как генерируется сброс или как данные подаются на функциональный блок, сброс может быть замаскирован неопределенным состоянием X. Большое количество SNUG статей посвящено этому вопросу. Большинство симуляторов не смогут разрешить некоторые условия X-логики и, следовательно, заблокируют синхронный сброс [7] [8] [9] [10] [11] [12] [13] [14] [15] [34]. Обратите внимание, что это также может быть проблемой и с асинхронными сбросами. Проблема не столько в том, какой тип сброса вы используете, сколько в том, легко ли сигнал сброса управляется внешним выводом.

По своей природе синхронный сброс требует тактового сигнала для сброса схемы. Для одних проектов использование синхронного сброса является приемлемым, но для некоторых дизайнов может стать большой проблемой. Например, если у вас есть схема управления тактовой частотой, позволяющей включать и отключать тактовый сигнал для экономии энергопотребления (*прим. пер. т.н. gated clock или clock gating*), тактовый сигнал может быть отключен одновременно со сбросом. В этой ситуации будет работать только асинхронный сброс.

Требование к наличию тактового сигнала для осуществления сброса, является существенным, если ASIC/FPGA имеет внутреннюю tristate шину. Чтобы предотвратить конфликт на внутренней шине tristate при включении микросхемы, микросхема должна иметь асинхронный сброс питания (см. рис. 5). Можно использовать синхронный сброс, однако вы должны напрямую перевести сигнал разрешения tristate (сигнал oe) в неактивное состояние с помощью сигнала сброса (см. рис. 6). Этот синхронный метод имеет преимущество при более простом временном анализе для пути reset-to-HiZ.

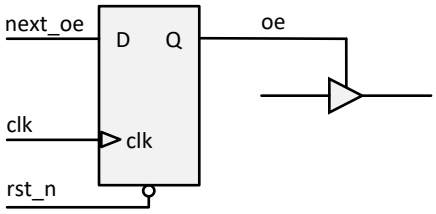

Рисунок 5 – Асинхронный сброс для сигнала разрешения выхода

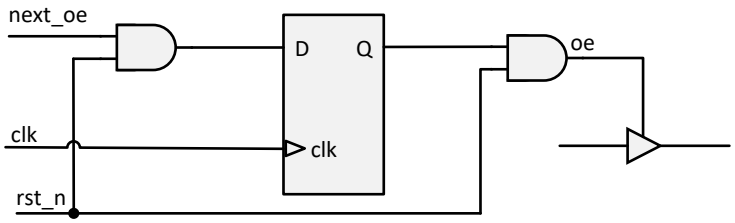

Рисунок 6 – Синхронный сброс для сигнала разрешения выхода

5.0 Асинхронный сброс

Продолжение

Ссылки

1. Оригинал статьи: Clifford E. Cummings, Don Mills, Steve Golson Asynchronous & Synchronous Reset Design Techniques - Part Deux