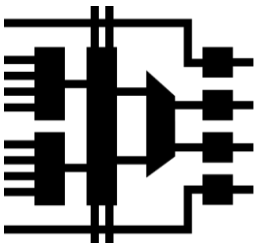

FPGA-SYSTEMS.RU

сообщество FPGA разработчиков

ПОДГОТОВИЛ: KeisN13

Асинхронный и синхронный сброс

Методы проектирования.

Часть II

Разделы 1, 2, 3

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 первая 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
21.05.2021	1.0	KeisN13	Начальный релиз документа

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

Аннотация

В этой статье будут рассмотрены плюсы и минусы синхронных и асинхронных сбросов. Мы рассмотрим использование каждого типа сброса, за которыми последуют рекомендации по правильному использованию каждого из них.

1.0 Введение

Тема дизайна сброса удивительно сложна и плохо освещена. Инженерные школы, как правило, не дают полной картины по детализации подводных камней неправильного проектирования сброса. Основываясь на нашем отраслевом и консалтинговом опыте, мы собрали наше текущее понимание вопросов, связанных с дизайном сброса, и для этой статьи добавили опыт нашего коллеги Стива Голсона, который проделал очень инновационную работу по дизайну сброса. Мы приветствуем любые отзывы коллег, связанные с этим важным вопросом.

Мы представили наш первый доклад по проблемам и методам сброса на конференции SNUG в марте 2002 года[4] и впоследствии получили многочисленные отзывы по электронной почте и вопросы, связанные с проблемами проектирования сброса.

Очевидно, мы не смогли адекватно объяснить все проблемы, связанные со схемой синхронизатора асинхронного сброса, потому что во многих из полученных нами электронных писем нас спрашивали, есть ли проблемы с метастабильностью, связанные с описанной схемой. Ответ на этот вопрос заключается в том, что нет никаких проблем с метастабильностью, связанных с этой схемой, и технический анализ и объяснение теперь подробно описаны в разделе 7.1 этой статьи.

Вопрос о том, следует ли использовать синхронные или асинхронные сбросы в дизайне, стал почти религиозным вопросом, и сильные сторонники утверждают, что их техника сброса - единственный способ правильно подойти к этому вопросу

В первой статье Дон и Клифф поддержали и рекомендовали использовать асинхронные сбросы в проектах и изложили наши причины выбора этого метода. С помощью Стива Голсона мы провели дополнительный анализ по этому вопросу и стали более нейтральны в отношении правильного выбора реализации сброса.

Очевидно, что существуют явные преимущества и недостатки использования синхронных или асинхронных сбросов, и любой метод может быть эффективно использован в реальных проектах. При выборе стиля сброса очень важно учитывать вопросы, связанные с выбранным стилем, чтобы принять обоснованное дизайнерское решение.

В этой статье представлены обновленные методы и соображения, связанные как с синхронным, так и с асинхронным сбросом. Эта версия документа включает обновленные порты модулей, описанные в стиле ANSI Verilog-2001 во всех примерах Verilog.

Первая версия статьи включала интересный метод синхронизации сброса нескольких ASIC в высокоскоростных приложениях. Данный материал был удален из этой статьи поэтому заинтересованным читателям стоит ознакомиться с первой версией, если эта тема представляет интерес.

2.0 Предназначение сброса

В любом случае, зачем беспокоиться об этих раздражающих маленьких сбросах? Зачем посвящать целую статью такой тривиальной теме? Любой, кто использовал компьютер с ОС, знает, что аппаратный сброс очень удобен. Он вернет компьютер в известное рабочее состояние (по крайней мере, временно), применив сброс системы к каждому чипу в системе, который имеет или требует сброса.

Для отдельных ASIC основной целью сброса является принудительное приведение дизайна ASIC (поведенческого, RTL или структурного) в известное состояние. После того, как ASIC спроектирован, необходимость в его сбросе определяется системой, применением ASIC и конструкцией ASIC. Например, многие ASIC, применяемые для передачи данных, предназначены для синхронизации со входным потоком данных, обработки данных и их последующей выдачи. Если синхронизация в какой либо момент теряется, ASIC проходит процедуру повторного получения сигнала синхронизации. Если этот тип ASIC спроектирован правильно, так что все неиспользуемые состояния указывают на состояние “начать синхронизацию”, он может нормально функционировать в системе без сброса. Сброс системы потребовался бы при включении питания для такого ASIC, если бы конечные автоматы в ASIC

использовали преимущество логического сокращения “все равно” (don’t care) на этапе синтеза.

Мы считаем, что, как правило, каждый триггер в ASIC должен быть сброшен независимо от того, требуется это системой или нет. В некоторых случаях, когда конвейерные триггеры (триггеры сдвигового регистра) используются в высокоскоростных приложениях, сброс может быть исключен из некоторых триггеров для достижения более высокой производительности. Такой вариант реализации требует заданного количества импульсов тактового сигнала в течение активного периода сброса, чтобы перевести ASIC в известное состояние.

Многие вопросы проектирования должны быть рассмотрены перед выбором стратегии сброса для проектирования ASIC. Например, следует ли использовать синхронные или асинхронные сбросы, будет ли каждый триггер получать сброс, как будет размещено и буферизовано дерево сброса, как проверить синхронизацию дерева сброса, как функционально протестировать сброс с помощью векторов тестового сканирования и как применить сброс через несколько тактовых логических доменов.

3.0 Общие примечания по стилю кодирования триггеров

3.1 Триггеры с синхронным сбросом с подключёнными триггерами без сброса

Каждый процедурный блок Verilog или процесс VHDL должен описывать только один тип триггера. Другими словами, дизайнер / разработчик не должен смешивать триггеры со сбросом с последующими триггерами (триггерами без сброса) в одном и том же процедурном блоке или процессе[14]. Последовательные триггеры - это триггеры, которые представляют собой простые регистры сдвига данных.

В коде Verilog примера 1a и коде VHDL примера 1б первый триггер используется для захвата данных, а затем его выход подключается к следующему триггеру. Первый этап этой конструкции – сброс, выполняемый синхронно. Второй этап описывает приёмный триггер, который не сбрасывается, но поскольку два триггера были описаны в одном и том же процедурном блоке/процессе, сигнал сброса **rst_n** будет использоваться в качестве источника данных для второго триггера. Этот стиль кодирования будет генерировать постороннюю логику, как показано на рис.1.

```
module badFFstyle (
    output reg  q2,
    input       d, clk, rst_n);

    reg         q1;

    always @(posedge clk)
        if (!rst_n) q1 <= 1'b0;
        else begin
            q1 <= d;
            q2 <= q1;
        end
endmodule
```

Пример 1а – Плохой стиль описания отличающихся триггеров на Verilog

```
library ieee;
use ieee.std_logic_1164.all;
entity badFFstyle is
    port (
        clk : in std_logic;
        rst_n : in std_logic;
        d : in std_logic;
        q2 : out std_logic);
end badFFstyle;

architecture rtl of badFFstyle is
    signal q1 : std_logic;
begin
    process (clk)
    begin
        if (clk'event and clk = '1') then
            if (rst_n = '0') then
                q1 <= '0';
            else
                q1 <= d;
                q2 <= q1;
            end if;
        end if;
    end process;
end rtl;
```

Пример 1б – Плохой стиль описания отличающихся триггеров на VHDL

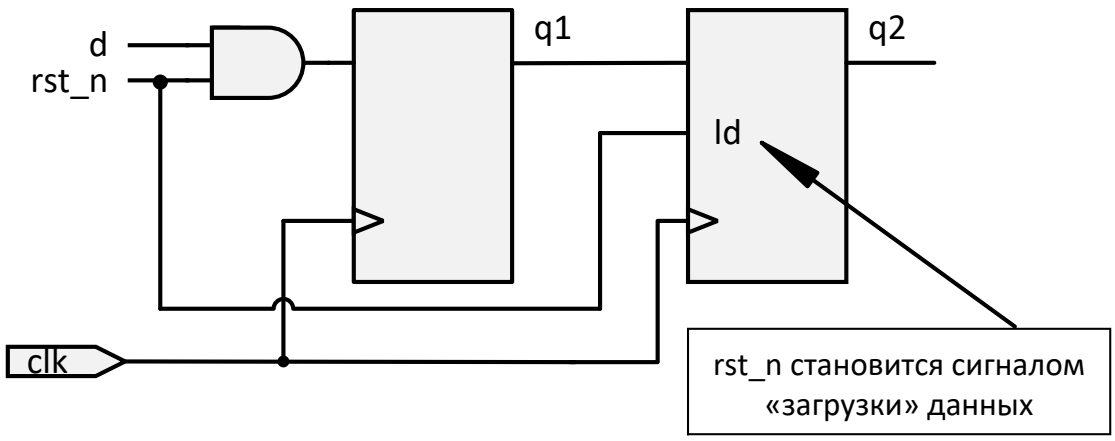

Рисунок 1 - Плохой стиль кодирования приводит к триггеру с лишней загрузкой

Правильный способ описания принимающего триггера – использовать два процедурных блока Verilog, как показано в примере 2а, или два процесса VHDL, как показано в примере 2б. Эти стили кодирования будут генерировать логику, показанную на рис. 2

```
module goodFFstyle (
    output reg q2,
    input d, clk, rst_n);
    reg q1;

    always @(posedge clk)
        if (!rst_n) q1 <= 1'b0;
        else q1 <= d;

    always @(posedge clk)
        q2 <= q1;
endmodule
```

Пример 2а – Хороший стиль описания отличающихся триггеров на Verilog-2001

```
library ieee;
use ieee.std_logic_1164.all;
entity goodFFstyle is
    port (
        clk : in std_logic;
        rst_n : in std_logic;
        d : in std_logic;
        q2 : out std_logic);
end goodFFstyle;

architecture rtl of goodFFstyle is
    signal q1 : std_logic;
begin
    process (clk)
    begin
        if (clk'event and clk = '1') then
            if (rst_n = '0') then
                q1 <= '0';
            else
                q1 <= d;
            end if;
        end if;
    end process;

    process (clk)
    begin
        if (clk'event and clk = '1') then
            q2 <= q1;
        end if;
    end process;
end rtl;
```

Пример 2б – Хороший стиль описания отличающихся триггеров на VHDL

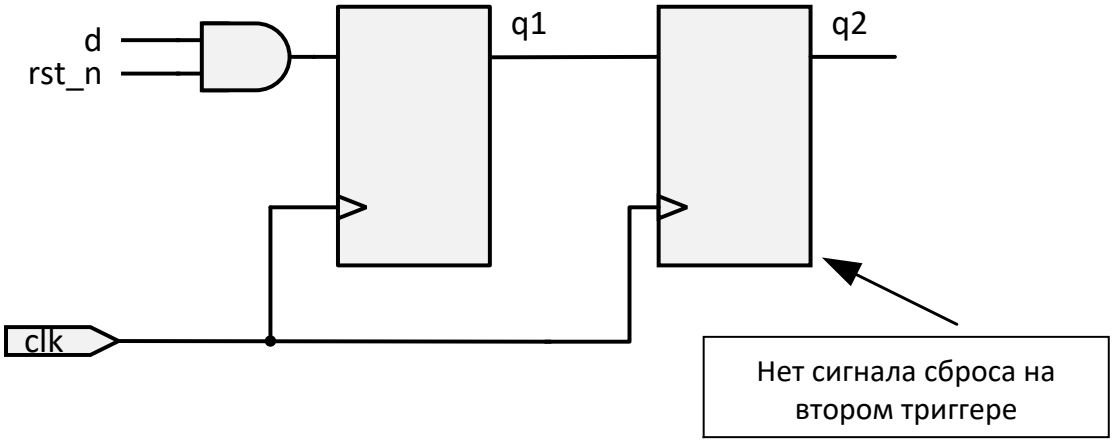

Рисунок 2 – Два разных триггера, один с синхронным сбросом, второй без

Следует отметить, что посторонняя логика, генерируемая кодом в примере 1а и примере 1б, является только результатом использования синхронного сброса. Если бы использовался подход асинхронного сброса, то оба стиля кодирования синтезировались бы в одном и том же дизайне без какой-либо дополнительной комбинационной логики. Генерация различных стилей триггеров в значительной степени зависит от списков чувствительности и операторов **if-else**, которые используются в коде HDL. Более подробная информация о списке чувствительности и стилях кодирования **if-else** приведена в разделе 4.1

3.2 Стили описания триггера

Не следует описывать каждый триггер в своем собственном процедурном блоке/процессе. С точки зрения стиля, все триггеры, выполняющие одну функцию или даже группу функций, должны быть описаны с использованием одного процедурного блока/процесса. Для описания больших последовательных блоков в рамках данного модуля/архитектуры следует использовать несколько процедурных блоков/процессов. Исключение из этого правила составляют последовательные триггеры, как описано в разделе 3.1, где для корректного описания требуется несколько процедурных блоков/процессов.

3.3 Рекомендации по оператору присваивания

В Verilog все назначения, выполняемые внутри блока always, описывающего триггер (последовательная логика), должны выполняться с неблокирующими операторами назначения[3]. Аналогично, для VHDL триггеры должны быть описаны с использованием оператора присваивания сигналов.

4.0 Синхронный сброс

О синхронном сбросе читайте во второй части

Мотивировать автора перевода	Поддержать проект FPGA-Systems.ru
Оставить комментарий/отзыв	

Ссылки

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