Лабораторная работа №4

Регистры и счётчики

Нам хотелось бы использовать цифровые устройства для обработки информации и выполнения вычислений. Но для осуществления этих возможностей нам недостаёт элемента памяти, который мог бы хранить промежуточные результаты. Ведь невозможно сделать калькулятор, если нет возможности сохранить вводимые числа и результат вычисления.

Элемент памяти — один из самых важных элементов цифровых устройств. Чтобы не делать ошибок при разработке цифровых устройств, необходимо понять место этого узла, его идею и инструменты языка Verilog, связанные с ним.

В качестве основных элементов памяти применяют **триггер** (flip-flop) и **защёлку** (latch). Разница между ними заключается в том, что первые изменяют своё состояние в зависимости от входных сигналов только в момент прихода импульса синхронизации, а вторые – в любой момент времени. Известно, что сигналы в цифровых схемах могут находиться в двух основных состояниях: логические «0» и «1», которым, как правило, соответствуют физические уровни напряжения в цепи распространения этого сигнала.

Следует понимать, что существует и промежуточное состояние, в котором напряжение в линии не достигло одного из устойчивых состояний, и такое состояние сигнала называют «метастабильным». Конкретные значения напряжений, при которых сигнал будет находиться в метастабильном состоянии, определяются технологией производства микросхем, напряжением питания, температурой и ещё множеством факторов, поэтому оценить нахождение сигнала в логических «0» или «1» можно только с некоторой вероятностью. Важно знать, что метастабильных состояний избежать нельзя, можно только минимизировать их влияние на поведение проектируемого устройства.

Защёлки

Первый элемент памяти, который мы рассмотрим – это **SR-защёлка**. В зависимости от выбранного базиса она будет иметь различную схему. Напомним, что **базисом** называют множество логических элементов, из которых построен функциональный узел.

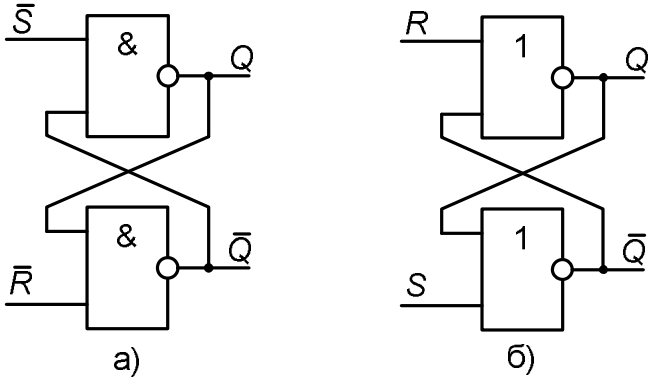

Рисунок 1 – а) SR-защёлка в базисе «И-НЕ»
б) SR-защёлка в базисе «ИЛИ-НЕ»

У SR-защёлки имеется два входа и два выхода. Входами являются сигналы S – «set» (установка в единицу) и R – «reset» (сброс). В зависимости от используемого базиса, полярность входных сигналов будет меняться. В базисе «И-НЕ», см. рисунок 1 а), установка и сброс происходят, когда сигналы S или R соответственно находятся в состоянии логического нуля, поэтому их обозначают как «инверсный сброс» и «инверсная установка», чтобы отразить этот факт. Выход защёлки Q - это тот бит данных, который

она хранит. Два выхода отличаются полярностью - один из них инвертирует хранимый бит. Отметим, что одновременное переключение сигналов S и R в логический ноль вызовет переключение сигналов Q и $\overline{Q}$ в состояние логического нуля, а затем, если один из сигналов R или S перейдёт в состояние логической единицы, то защёлка вернётся к нормальному функционированию. Если же R и S перейдут в состояние логической единицы одновременно, то состояние следующие состояние выхода Q защёлки не определено и возможны переключения из «0» в «1» и обратно. В связи с этой особенностью состояние $S = R = 0$ для защёлки в базисе «И-НЕ» называют запрещённым. Для SR-защёлки в базисе «ИЛИ-НЕ» (рисунок 1 б)) состояние $S = R = 1$ является запрещённым.

Ниже приведена временная диаграмма работы SR-защёлки в базисе «И-НЕ».

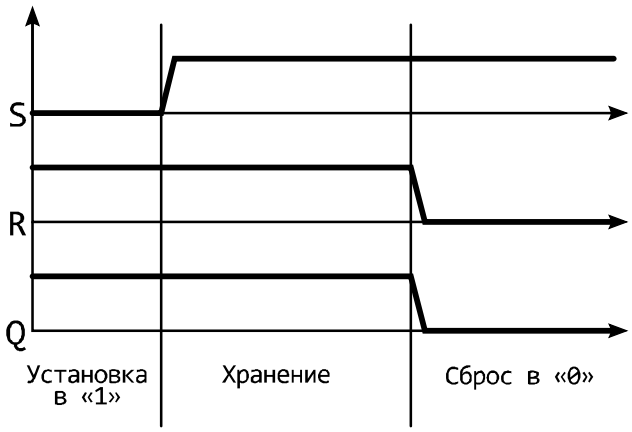

Рисунок 2 – Временная диаграмма работы асинхронного RS-триггера

Как будет выглядеть Verilog-описание этого триггера?

Для его реализации нам понадобятся два входа, два выхода и два элемента «И-НЕ», которые мы опишем с помощью логических операторов «&» и «~» - «И» и «НЕ», соответственно. Обратите внимание, что использован вентильный или структурный уровень абстракции.

```
module RS_trig(
input nR,
input nS,
output Q,
output nQ);

assign Q = ~(nS & nQ);
assign nQ = ~(nR & Q);

endmodule
```

SR-защёлки применяются в разного рода управляющих устройствах, где часто возникают ситуации, когда в качестве реакции на выполнение того или иного условия нужно выставлять «флаг», а при изменении этих условий сбрасывать его. Чаще всего элемент памяти нужен нам для хранения данных. Для того чтобы избавиться от необходимости проверять, что комбинация состояний входов S и R защёлки не является запрещённой, схема на рисунке 1 а) была усовершенствована и представляет собой **D-защёлку**, схема которой представлена на рисунке 3 а).

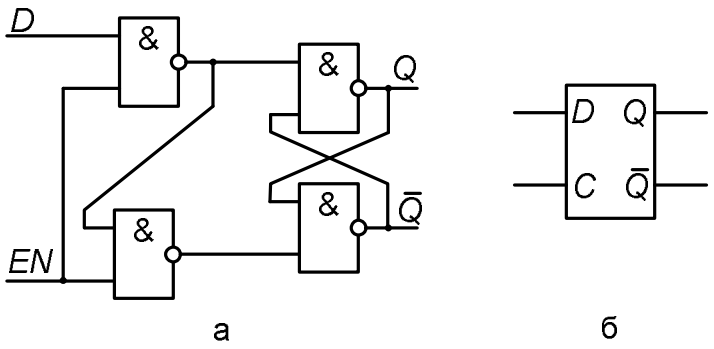

Рисунок 3 – D-защёлка в базисе «И-НЕ»:
а) внутренняя структура; б) графический примитив

D-защёлка работает следующим образом: при высоком уровне на входе *EN* - «enable» (разрешить работу) данные со входа *D* - «data» (данные) будут передаваться на выход защёлки, а при низком уровне на входе *EN* защелка будет сохранять на выходе последнее значение со входа *D*, которое было до переключения сигнала *EN*. Работа такой защелки проиллюстрирована временной диаграммой, изображённой на рисунке 4. В графическом примитиве, наиболее часто встречающемся в литературе, который изображён на рисунке 3 б), вход *EN* обозначен как *C* – «control».

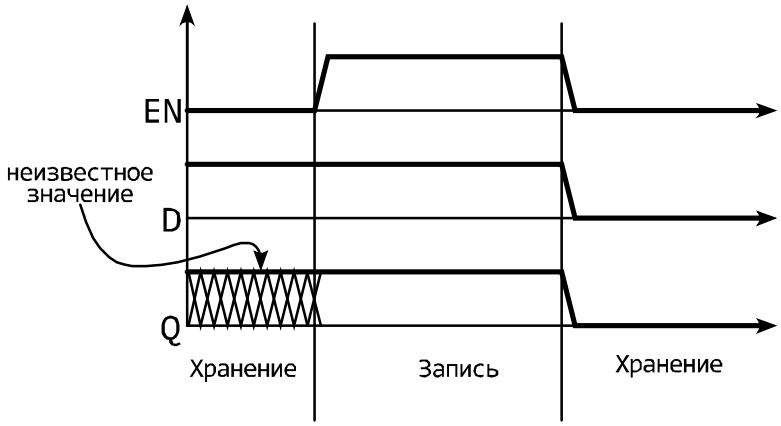

Рисунок 4 – Временная диаграмма работы D-защёлки

Как мы уже говорили, человеку гораздо удобнее использовать поведенческие описания на уровне регистровых пересылок. На этом уровне абстракции используются только синтезируемые конструкции, одновременно представляя наглядно потоки данных внутри модуля и уверенность в том, что после синтеза будет получен именно тот элемент, который задумывался. Такое описание легче понять разработчику, улучшается читаемость написанного кода и уменьшается вероятность ошибок при его написании и отладке.

Опишем D-защёлку на языке Verilog, используя поведенческий подход:

```
module d_latch_bhv(
input d,
input en,
output reg q);

always @(en, d) begin
  if (en) q <= d; // запись
  else q<= q; // хранение
end
endmodule
```

В данном примере использован блок **always**, в списке чувствительности которого находятся сигналы, влияющие на изменение значения в защёлке. В этом случае говорят, что изменение происходит «по уровню» сигналов в списке чувствительности. При изменении любого из этих сигналов значение выхода также будет изменено. Тем не менее, это изменение не произойдёт мгновенно, а займёт некоторое время.

Триггеры

Несмотря на то, что такая защёлка решает проблему с запрещёнными состояниями, проблема с метастабильностью выходного сигнала в такой защёлке тоже присутствует, в частности, если сигнал на входе *D* изменит своё состояние во время установки выхода защёлки в устойчивое состояние. Уменьшением риска возникновения метастабильного состояния на выходе защёлки является изменение значения, хранимого в защёлке «по фронту». В таком случае структура, хранящая один бит информации, будет называться **D-триггером** (d-flip-flop, dff), изображённая на рисунке 5 а). Этот триггер изменяет значение выхода по переходу управляющего сигнала *CLK* из «0» в «1» - то есть «переднему» фронту.

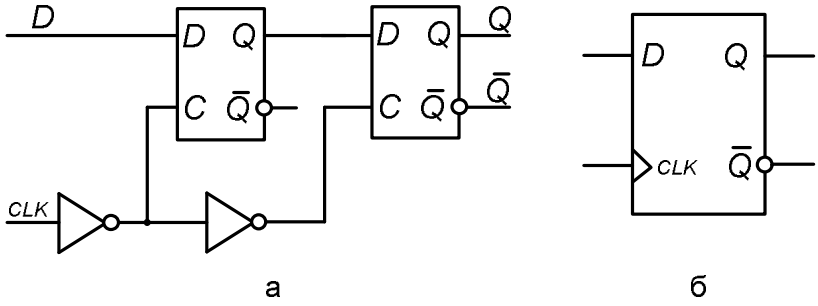

Рисунок 5 – D-триггер, состоящий из двух D-защёлок:
а) внутренняя структура; б) графический примитив

Англоязычное название подчёркивает принцип работы этого триггера:

В дальнейшем эта схема была усовершенствована за счёт добавления входов разрешения работы *EN*, установки в «1» *SET* и сброса в «0» *RST*. Графический примитив такого триггера выглядит следующим образом:

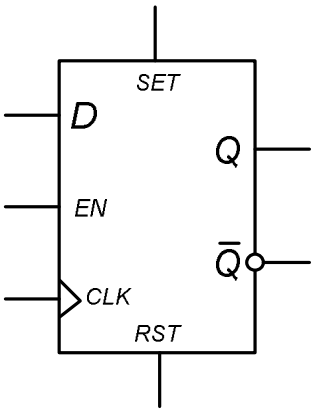

Рисунок 6 – D-триггер с входами разрешения работы, сброса в «0» и установки в «1»

Эта схема получила широчайшее применение в цифровой схемотехнике и называется D-триггер. Ниже приведена временная диаграмма работы D-триггера (см. рисунок 7). Чаще всего инверсный выход не используется, поэтому в схемах его обычно не изображают.

Отметим, что в различных микросхемах такой триггер может выглядеть гораздо сложнее.

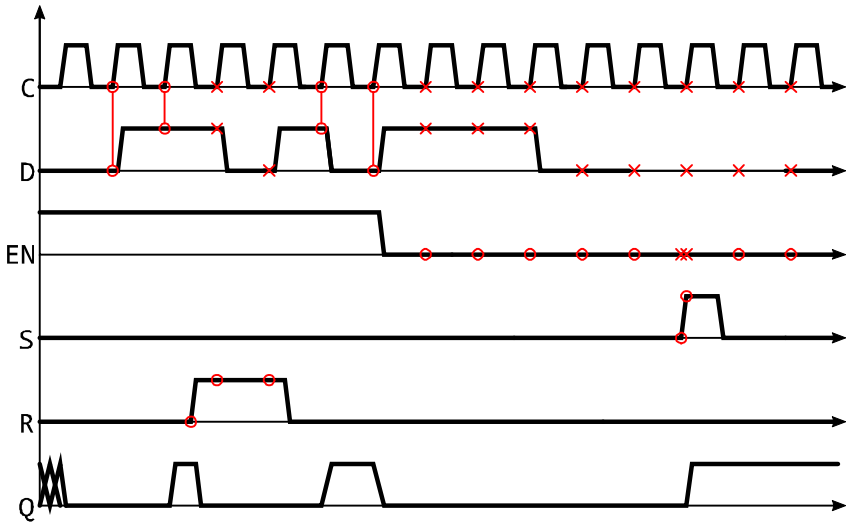

Рисунок 7 – временная диаграмма работы D-триггера

Заметим, что сигнал *CLK* называют «тактирующим» сигналом или «сигналом синхронизации». Такое название обусловлено тем, что для прогнозируемого поведения разрабатываемого устройства необходима слаженная работа всех его функциональных блоков. По аналогии с поездами, самолётами, синхронизация которых происходит с помощью часов, синхронизация работы функциональных блоков выполняется с помощью таких сигналов. Обычно в роли этого сигнала выступает сигнал, формируемый внешним источником (чаще всего кварцевым или MEMS-генератором) со стабильной частотой. Сами цифровые устройства, для работы которых необходим сигнал синхронизации, называют **синхронными**, при этом все прочие сигналы, формирование которых не зависит от сигнала синхронизации, называют **асинхронными**.

Сигнал синхронизации играет очень большую роль в цифровых устройствах. Прежде всего, он необходим для того, чтобы избежать непредсказуемого и нестабильного поведения триггеров в цифровых устройствах.

Опишем такой триггер на языке Verilog:

```
module dff_bhv(
input d,
input clk,
input set,
input rst,
input en,
output reg q);

always @(posedge clk or posedge set or posedge rst ) begin
  if (set) q <= 1'b1;
  else begin
    if (rst) q <= 1'b0;
    else begin
        if (en) q <= d;
        else q <= q;
    end
  end
end

endmodule
```

Обратите внимание, что в описании появилось новое ключевое слово **posedge**. Оно используется только в списке чувствительности блока **always** и означает событие перехода сигнала, имя которого стоит после этого ключевого слова, из состояния «0» в состояние «1».

Ключевое слово **posedge** было введено прежде всего для того, чтобы описывать схемы, содержащие триггеры. Ведь триггеры, как мы уже говорили, могут менять своё состояние только в момент положительного фронта («positive edge») сигнала синхронизации. Аналогичным ключевым словом для спадающего фронта, то есть для перехода из «1» в «0», является **negedge** («negative edge»).

Добавление в список чувствительности события **posedge** *rst* позволяет описать поведение триггера в момент асинхронного сброса: как только случается переход *rst* из «0» в «1» срабатывает блок **always** и проверка условия **if** (*rst*) дает положительный результат, значение триггера сбрасывается в «0». Аналогичным образом обрабатывается сигнал установки в «1» - *set* .

Регистры

Для хранения и обработки многоразрядных данных в цифровой схемотехнике используют группу из триггеров, которую называют **регистром**. Ниже представлено описание 8-разрядного регистра с параллельной загрузкой на D-триггерах с асинхронным сбросом на языке Verilog:

```
module register_bhv(
input [7:0] d,
input clk,
input rst,
input en,
output reg [7:0] q);

initial q <= 8'b0;

always @(posedge clk or posedge rst) begin
  if (rst) q <= 7'b0;
  else begin
     if (en) q <= d;
     else q<= q;
  end
end
endmodule
```

Элементы памяти позволяют нам сохранять информацию для дальнейшей обработки, хранить промежуточный или окончательный результат вычислений. На самом деле практически в любом устройстве, описанном на языке Verilog, используются конструкции, связанные с использованием групп триггеров, объявление которых выглядит одинаково, однако результат синтеза, как то: параллельный регистр, счётчик, сдвиговый регистр, будет определяться его использованием внутри блока **always.**

Важно, что переменные типа **reg** могут быть изменены только в пределах одного блока **always**. Переменные доступны для считывания и проверки в любом из блоков, но изменять их значение можно только в одном из них. Пример наиболее часто встречающейся ошибки при несоблюдении этого правила представлен в следующем Verilog-описании:

```
…
reg a;
reg b;

always @(posedge clk) begin
   if (in < 5) a <= in;
   else a<= a;
end

always @(posedge clk) begin
   if (n > 5) begin
      b <= in;
      a <= in – 5; //ошибка,
   end
   else b <= a;
end
…
```

Счётчики

Очень распространённой цифровой схемой на основе регистра является счётчик, который считает количество тактов, которое прошло с момента сброса его значения и предназначен для синхронизации каких-либо событий в устройствах, счёта времени и т.д.

Самый простой счётчик инкрементирует значение, хранящееся в регистре, на единицу каждый период сигнала синхронизации. Тем не менее, как будет показано далее, счётчик можно доработать таким образом, чтобы отсчитывались не такты, а какие-либо события. Такими событиями могут являться: нажатие кнопки, принятие пакета данных по интерфейсу, срабатывание датчика, выполнение какого-либо повторяющегося условия и другое.

Итак, для того чтобы создать счетчик, нам понадобится регистр, состоящий из D-триггеров, и сумматор. Причем сумматор будет складывать значение, хранящееся в регистре с константой (в нашем случае с единицей), а результат сложения будет поступать на вход регистра.

В результате получим следующую схему:

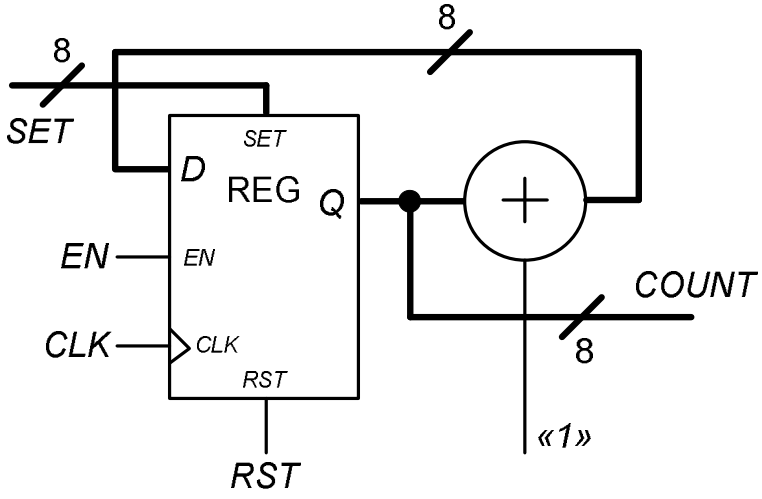

Рисунок 8 –Счётчик на восемь разрядов на основе D-триггеров

Заметим, что вход *SET* счётчика может использоваться для задания начального значения, причём не стоит пренебрегать этой возможностью, более того, любому счётчику, используемому в реальном устройстве во избежание его неправильной работы необходимо задать начальное значение, либо с помощью *SET* , либо с помощью *RST* .

На временной диаграмме ниже хорошо видно как работает счётчик:

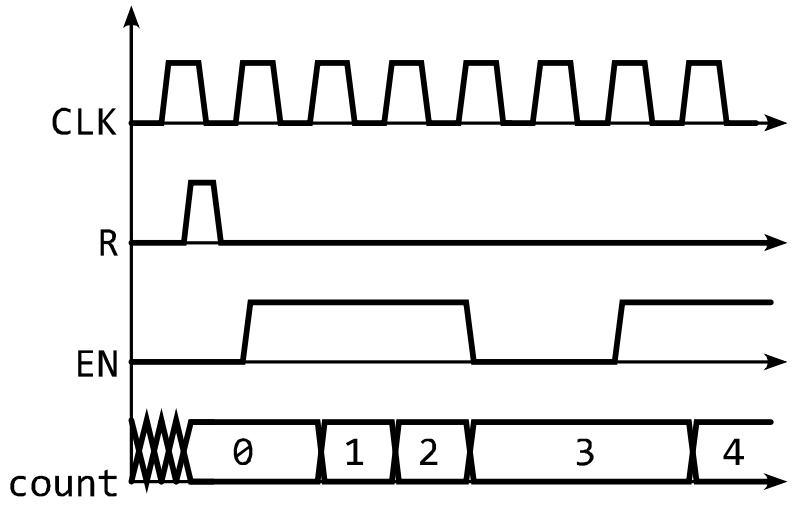

Рисунок 9 – Временная диаграмма работы 8-разрядного счётчика

Счётчик на восемь разрядов может считать до значения $2^8 - 1 = 255$ или до значения $8'b1111\_1111$.

Что же произойдёт со значением такого счётчика, если счёт не остановить до достижения этого значения? Ответ прост: счётчик переполнится и начнёт счёт заново, с нуля, поэтому следует всегда предусматривать возможность проверки на переполнение и/или своевременный сброс во избежание неправильной работы разрабатываемого устройства. Отметим, что счётчик может считать в обратную сторону, то есть выполнять декремент, вместо константы «1» используется «-1» такой счётчик называется **реверсивным**.

Создадим модуль, описывающий поведение такого счётчика на языке Verilog:

```
module counter_8bit(
input clk,
input en,
input rst,
output reg [7:0] counter);

initial counter = 8'b0;

always @(posedge clk or posedge rst) begin
  if (rst) counter <= 0;
  else begin
    if (en) counter <= counter + 1;
    else counter <= counter;
  end
end

endmodule
```

Обратите внимание на ключевое слово **initial,** которое используется для задания начального значения регистра. Причём, несмотря на то, что *SET* отсутствует в перечне портов разработанного модуля, для задания начального значения после синтеза будет использован именно он.

Для того чтобы можно было подсчитывать события, а не переходы сигнала синхронизации из «0» в «1» понадобится схема обработки такого события. Очевидно, что такие сигналы являются асинхронными по отношению к сигналу синхронизации и для предсказуемости работы разрабатываемого устройства требуется такие сигналы сделать синхронными. А для того, чтобы сформировать выходной сигнал длительностью в один период сигнала синхронизации требуется схема определения перехода из «0» в «1».

Ниже представлена схема, позволяющая реализовать обе функции:

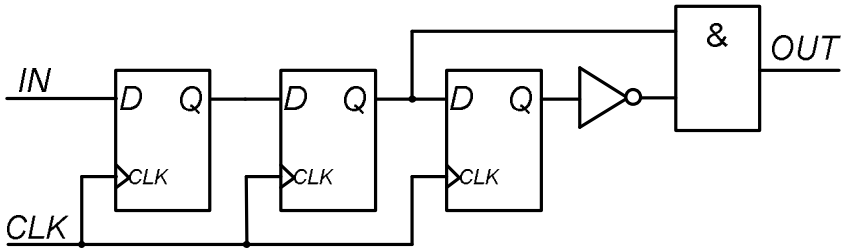

Рисунок 10 – Схема синхронизации и обнаружения фронта асинхронного сигнала

Временная диаграмма, иллюстрирующая работу такой схемы:

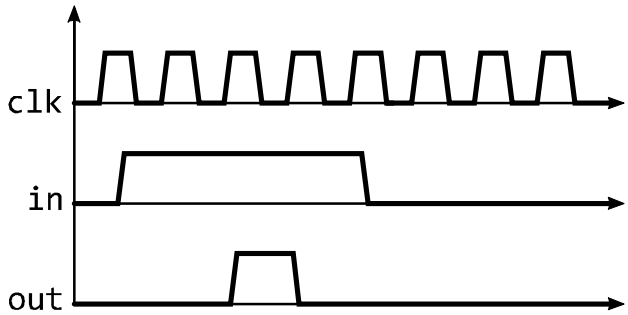

Рисунок 11 – Временная диаграмма работы схемы синхронизации и обнаружения переднего фронта асинхронного сигнала

В дальнейшем выход *OUT* такой схемы может быть использован как входной сигнал *EN* создаваемого счётчика. Естественно, что такая схема работает только тогда, когда входной сигнал *IN* изменяется с частотой меньшей, чем частота сигнала синхронизации, причём задержка от момента получения асинхронного сигнала до возникновения сигнала *OUT* составляет 2 периода сигнала синхронизации, что в некоторых случаях требует внимания и не всегда приемлемо. В случае, если входной сигнал *IN* зашумлён или в нём присутствуют колебания, вместо первого (слева) D-триггера может быть использован триггер Шмитта, если порт ввода/вывода микросхемы предусматривает такую возможность.

Сдвиговые регистры

Сдвиговым регистром называют устройство, в котором выполняется логический, арифметический или циклический сдвиг. Задача выполнения логического сдвига наиболее часто встречается в различных интерфейсах передачи данных в качестве преобразователя последовательного кода в параллельный. Сдвиговые регистры, реализующие циклический сдвиг, чаще всего используются в аппаратных реализациях алгоритмов шифрования.

Мы же рассмотрим наиболее часто встречающуюся задачу выполнения логического сдвига. У самого простого сдвигового регистра при сдвиге влево старший разряд «выпадает» из регистра, содержимое регистра сдвигается, как показано на рисунке 12:

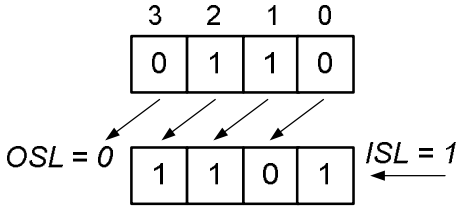

Рисунок 12 – Сдвиг влево в 4-разрядном сдвиговом регистре

Сигнал *ISL* («Input Shift Left»)- значение бита, который встает на место младшего разряда при сдвиге влево, а *OSL* («Output Shift Left») – старший разряд регистра, который является выходным битом при сдвиге влево. Аналогичным образом выполняется логический сдвиг вправо с использованием сигналов *ISR* («Input Shift Right») и *OSR* («Output Shift Right») соответственно:

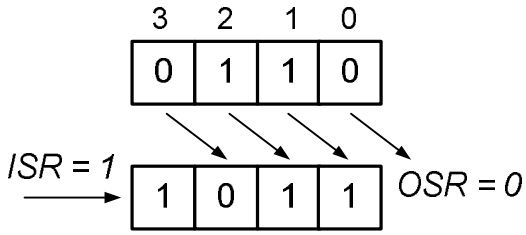

Рисунок 13 – Сдвиг вправо в 4-разрядном сдвиговом регистре

Графический примитив сдвигового регистра, осуществляющего сдвиг влево, будет выглядеть следующим образом:

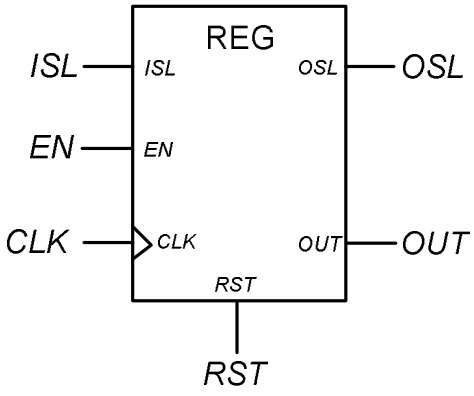

Рисунок 14 – Графический примитив сдвигового регистра, осуществляющего сдвиг влево

Теперь опишем поведение этого сдвигового регистра:

```
module shift_reg_left_4bit(
input clk,
input en,
input ISL,
input rst,
output reg OSL,
output reg [3:0] shift_rg);

initial shift_rg = 4'b0;

always @(posedge clk) begin
  if (rst) begin
    shift_rg <= 4'b0;
    OSL <= 1'b0;
  end else begin
     if (en) begin
       shift_rg <= {shift_rg[2:0], ISL};
       OSL <= shift_rg[3];
     end else begin
       OSL <= 1'b0;
       shift_rg <= shift_rg;
    end
  end
end

endmodule
```

Обратите внимание на синтаксис оператора объединения «{<…>, <…> }», который как раз реализует поведение сдвигового регистра согласно рисунку 12. Использование этого оператора предоставляет бóльшую гибкость по сравнению с оператором «<< 1», который выполняет логический сдвиг влево на один разряд, но не позволяет одновременно записать в регистр значение *ISL* . Сдвиг вправо описывается аналогичным образом.

Самостоятельно опишите модуль, реализующий следующий графический примитив:

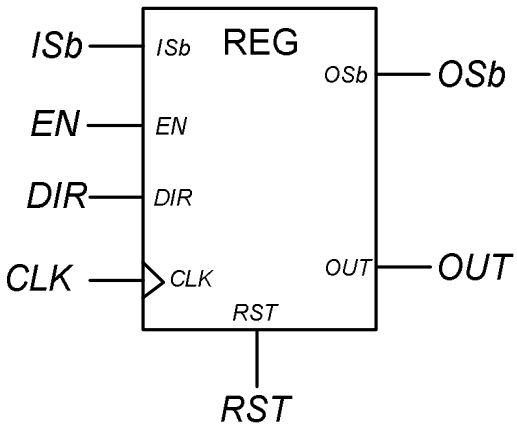

Рисунок 15 – Графический примитив сдвигового регистра с возможностью выбора направления сдвига

Эта схема, построенная на 4-разрядном регистре, позволяет выбирать направление логического сдвига с помощью сигнала *DIR*: «0» - сдвиг влево, «1» - сдвиг вправо, сигналы *ISb* и *OSb* - входной и выходной бит соответственно. Сброс *RST* - асинхронный. Проверьте работу разработанного модуля с помощью временных диаграмм.

На основе разработанной структуры можно реализовать остальные виды сдвигов. При циклическом сдвиге выходной бит при сдвиге подключается напрямую к сигналу «вдвигаемого» бита. Арифметический же подобен логическому сдвигу, только следует учитывать (не сдвигать) знак числа, который обычно располагается в старшем разряде.

Задание лабораторной работы:

1. Внимательно ознакомиться с работой всех устройств, рассмотренных в данных методических указаниях, скопировав код, проверить работу каждого устройства с помощью временных диаграмм.
2. Разработать сдвиговый регистр, реализующий примитив с рисунка 15.
3. Выполнить индивидуальное задание - описать на языке Verilog цифровое устройство, функционирующее согласно следующим принципам:
   1) Ввод информации происходит с переключателя SW[0] и кнопок KEY[1], KEY[0].
   2) Внешний источник сигнала синхронизации - CLOCK_50.
   3) Кнопка KEY[1] должна функционировать как общий асинхронный сброс устройства.
   4) При нажатии на KEY[0] устройство должно записывать данные с SW[0] в десятиразрядный сдвиговый регистр.
   5) Содержимое десятиразрядного регистра выводить на LEDR[9:0].
   6) При нажатии на KEY[0] увеличивать 7-разрядный счётчик нажатий на один, если произошло событие, указанное в индивидуальном задании студента.
   7) Содержимое счётчика выводить в десятичной форме на HEX0 и HEX1 используя модули, полученные во время выполнения лабораторной №3.

Индивидуальные задания:

№	Задание
1.	Число единиц в сдвиговом регистре чётно
2.	Количество единиц больше количества нулей
3.	Значение старших пяти битов больше значения младших пяти битов
4.	Сумма по модулю два старших пяти разрядов равно нулю
5.	Сумма младших 7 разрядов больше 4

Примечания:

1. Вариант выбирается студентом по модулю 6 в соответствии с номером в списке группы, то есть 6-ой вариант соответствует первому, 13-ый – третьему и т.д.
2. Для решения вопроса о чётности числа единиц нужно использовать оператор свёртки «^».
3. Выполнив описание модуля на языке Verilog необходимо построить временные диаграммы его работы с помощью САПР Altera Quartus II.
4. Подключить входы модуля к переключателям SW[9:0] учебного стенда, а выходы - к шинам HEX1, HEX0 и LEDR. Получить файл конфигурации ПЛИС и продемонстрировать работу проекта на учебном стенде.

Пример индивидуального задания

Событием является наличие 3 и более единиц на SW[9:0] в момент записи в регистр.

Решение индивидуального задания (фрагмент кода лабораторной работы)

```
…
reg sw_event = 1'b0;
reg [2:0] event_sync_reg = 3'b0;
wire synced_event;

// схема синхронизации и обнаружения фронта асинхронного сигнала
assign synced_event = event_sync_reg[1] & ~event_sync_reg[0];

always @(SW) begin
   if ((SW[0] + SW[1] + SW[2] + SW[3] + SW[4]
     + SW[5] + SW[6] + SW[7] + SW[8] + SW[9]) > 4'd3)  sw_event <= 1'b1;
   else sw_event <= 1'b0;
end

always @(posedge CLOCK_50) begin
  event_sync_reg[2] <= sw_event;
  event_sync_reg[1:0] <= event_sync_reg[2:1];
end
…
```

Вопросы к защите лабораторной работы

1. Какие элементы памяти вы изучили в данной лабораторной работе?
2. Чем отличается SR-защелка от D-защелки?
3. Какие входы могут быть у триггера? Перечислите все и назовите их функции.
4. Что такое сигнал синхронизации, как и где он формируется?
5. Какие блоки Вашего цифрового устройства синхронные? Какие - нет? Почему?
6. Какой фрагмент вашего Verilog-описания реализует вывод значения счетчика на семисегментный индикатор? Как называется эта цифровая схема?
7. Что такое сдвиговый регистр? Покажите его в Вашем проекте и продемонстрируйте его работу.
8. Что такое сдвиговый регистр? Объясните, как он работает.