

Лабораторная работа №2

Регистры и счётчики

Функции цифровых устройств, естественно, не сводятся к реализации разнообразных ФАЛ. Нам хотелось бы использовать цифровые устройства для обработки информации, 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.

Первый элемент памяти, который мы рассмотрим — это **защелка** (англ. latch).

Защелка является основой всех элементов памяти. Она состоит из двух элементов И-НЕ (или из двух элементов ИЛИ-НЕ, в зависимости от базиса, выбранного при проектировании), соединённых по следующей схеме:

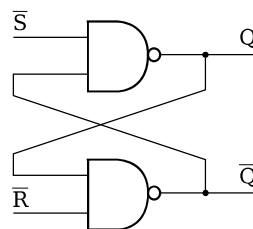


Рис. 2.1: Структура RS-защелки

У защелки два входа и два выхода. Входами являются сигналы «сброс» и «установка в единицу» или по-английски «reset»

и «set». В зависимости от элементов, из которых состоит защелка, полярность входных сигналов будет меняться. В базисе И-НЕ сброс и установка происходят, когда соответственно сигналы R или S находятся в нуле, поэтому их обозначают как «не-сброс» и «не-установка», чтобы отразить этот факт. Выход защелки — это тот бит данных, который она хранит. Два выхода отличаются полярностью — один из них инвертирует хранимый бит. Ниже приведена таблица со всеми возможными комбинациями входных сигналов и временная диаграмма работы защелки.

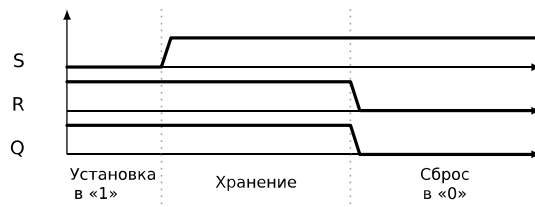


Рис. 2.2: Временная диаграмма работы RS-защелки

Опишем защелку на языке Verilog, опираясь на её структуру, которую мы рассмотрели выше. Нам понадобятся два входа, два выхода и два элемента И-НЕ, которые мы опишем с помощью операций И (оператор `&`) и НЕ (оператор `~`).

```

1 module latch_struct(
2   input nR,
3   input nS,
4   output Q,
5   output nQ);
6
7   assign Q = ~(nS & nQ);
8   assign nQ = ~(nR & Q);
9
10  endmodule;
```

Листинг 2.1: Описание RS-защелки на языке Verilog HDL

Элемент памяти нам, прежде всего, нужен для хранения данных. Для того, чтобы защелкой стало удобнее пользоваться, немного изменим схему подключения управляющих сигнала-

ЛОВ.

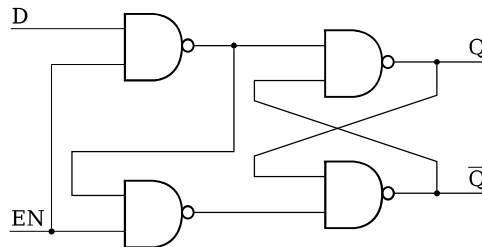


Рис. 2.3: Структура D-защелки

Защелка теперь будет работать следующим образом: при высоком уровне на входе «разрешить работу» («enable») данные со входа «данные» («data») будут проходить через защелку на выход, при низком уровне на входе «разрешить работу» защелка будет сохранять на выходе последнее значение со входа «данные», которое было до переключения сигнала «разрешить работу». Работа такой защелки показана на временной диаграмме ниже.

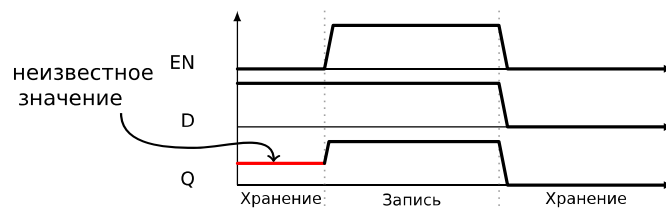


Рис. 2.4: Временная диаграмма работы D-защелки

Как мы уже говорили, использовать структурные описания не всегда удобно. В большинстве случаев использовать поведенческое описание намного эффективнее. Поведенческое описание часто формулируется гораздо лаконичнее, и, так как его легче понять человеку, улучшается читаемость кода и уменьшается вероятность ошибок при его написании.

```
1 module d-latch_behav(
2   input d,
3   input en,
4   output reg q);
```

```

5
6 always @(en, d) begin
7   if (en) q <= d;
8 end
9
10 endmodule;

```

Листинг 2.2: Поведенческое описание D-защелки на языке Verilog HDL

Если добавить к этой схеме еще две защелки, то можно привязать изменение «содержимого» защелки к переходу управляющего сигнала из «0» в «1», то получим следующую структуру:

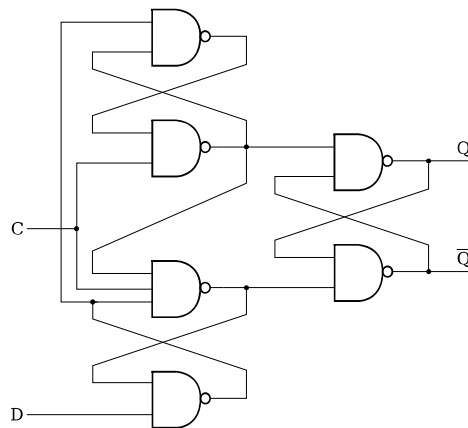


Рис. 2.5: Структура D-триггера

Эту схему можно немного доработать, введя управляющие сигналы сброса, установки в единицу и разрешения работы. Упрощенно такая схема изображается следующим образом.

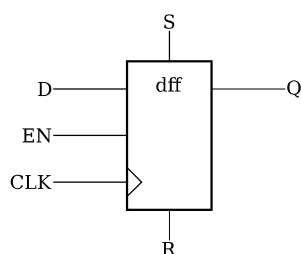


Рис. 2.6: Графическое обозначение D-триггера

Эта схема получила широчайшее применение в цифровой схемотехнике и называется d-триггер (от слова «data» — данные). Ниже приведена временная диаграмма работы d-триггера.

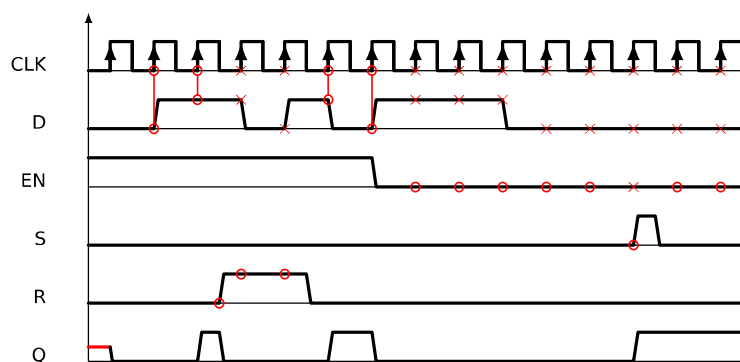


Рис. 2.7: Пример работы регистра

Заметим, что сигнал S называют «тактирующим» сигналом или «сигналом синхронизации». Обычно в роли этого сигнала выступает сигнал от внешнего источника (чаще всего кварцевого резонатора) со стабильной частотой. А сами цифровые устройства, для работы которых необходим сигнал синхронизации, называют синхронными.

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.

```

1 module d-flipflop_behav(
2   input d,
3   input clk,
4   input rst,
5   input en,
6   output reg q);
7
8   always @(posedge clk or posedge rst) begin
9     if (rst) q <= 0;
10    else if (en) q <= d;
11  end
12
13 endmodule;

```

Листинг 2.3: Описание D-триггера на языке Verilog HDL

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 (англ. *positive edge*) сигнала синхронизации.

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, триггер сбрасывается в «0».

Если объединить несколько триггеров в группу, то получится то, что в цифровой схемотехнике называют «регистр».

```

1 module register_behav(
2   input [7:0] d,
3   input clk,
4   input rst,
5   input en,
6   output reg [7:0] q);
7

```

```

8  always @(posedge clk or posedge rst) begin
9      if (rst) q <= 0;
10     else if (en) q <= d;
11 end
12
13 endmodule;

```

Листинг 2.4: Описание регистра на языке Verilog HDL

Элементы памяти позволяют нам сохранять информацию для дальнейшей обработки или хранить готовый результат вычисления, хранить промежуточные результаты.

Запомните описание регистра. Оно используется при проектировании практически любого цифрового устройства с помощью Verilog.

Необходимо отметить важную концепцию языка Verilog. **Переменные типа reg могут быть изменены только в пределах одного блока always. Переменные доступны для проверки в любом из блоков, но изменять их значение можно только в одном из них.**

```

1  reg a;
2  reg b;
3
4  always @(posedge clk) begin
5      if (in < 5) a <= in;
6  end
7
8  always @(posedge clk) begin
9      if (n > 5) begin
10         b <= in;
11         a <= in - 5; //ошибка!!!
12     end
13     else b <= a;
14 end

```

Листинг 2.5: Пример присвоения значения переменной a разных блоках always на языке Verilog HDL

Одной из простейших, и в тоже время широко распространённой, цифровой схемой на основе регистров является счёт-

Счётчик считает количество тактов, которое прошло с момента его обнуления.

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

The diagram shows a feedback loop where the output of a counter is added to a constant value (represented by a circle with a plus sign) and the result is stored in a register. The register's output is then fed back into the counter. The register has an enable input (EN) and a clock input (CLK). The counter has a clock input (CLK) and an output (count). The feedback path includes an adder and a constant value of 1.

На временной диаграмме ниже хорошо видно как работает счётчик:



Опишем поведение такого счётчика на Verilog.

```
1 module counter_8bit(  
2   input clk,  
3   input en,  
4   input rst,  
5   output reg [7:0] counter);  
6  
7   always @(posedge clk or posedge rst) begin  
8     if (rst) counter <= 0;  
9     else if (en) counter <= counter + 1;  
10  end  
11  
12 endmodule;
```

Листинг 2.6: Описание восьмибитного счетчика на языке Verilog HDL

Для того чтобы можно было подсчитывать события, а не переходы сигнала синхронизации из «0» в «1» понадобится ввести еще одну схему. Её смысл и назначение заключается в следующем: нам необходимо из асинхронного события получить синхронный сигнал единичной длительности. Тогда, подавая такой сигнал на вход enable счётчика, мы сможем считать количество произошедших событий.

Ниже представлена схема, позволяющая сделать это:

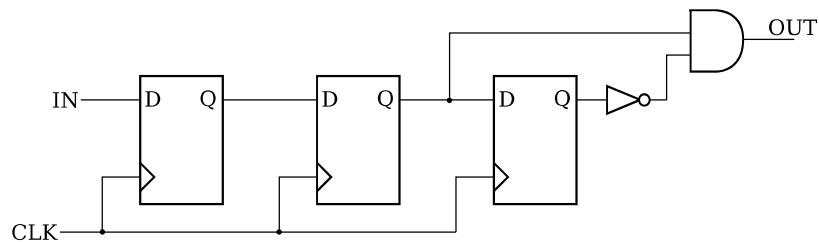


Рис. 2.10: Структура регистра на D-триггерах

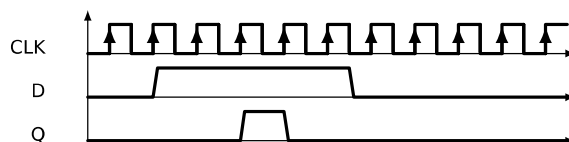


Рис. 2.11: Пример работы регистра

Естественно такая схема работает только тогда, когда входной сигнал изменяется с частотой меньшей, чем частота синхронизации.

Сигнал out в таком случае подключается к входу enable счётчика.

2.1 Задание лабораторной работы

Описать на языке Verilog цифровое устройство, функционирующее согласно следующим принципам:

1. Ввод информации происходит с переключателей SW[9:0] и кнопок KEY[0], KEY[1]. Внешний источник сигнала синхронизации: CLK50;
2. KEY[1] должна функционировать как общий асинхронный сброс устройства;
3. При нажатии на KEY[0] записывать данные с SW[9:0] в десятиразрядный регистр;
4. Содержимое десятиразрядного регистра выводить на LEDR[9:0];
5. При нажатии на KEY[0] увеличивать 8-ми разрядный счётчик нажатий на 1, если произошло событие, указанное в индивидуальном задании студента;
6. Содержимое счётчика выводить в шестнадцатеричной форме на HEX0 и HEX1 (цифры с 0 до 9 и буквы A, B, C, D, E, F)

Выполнив описание модуля на языке Verilog необходимо построить временные диаграммы его работы с помощью САПР Altera Quartus.

Привязать входы модуля к переключателям SW, отладочной платы, а выход к шине HEX0[6:0], получить прошивку для

ПЛИС и продемонстрировать её работу.

2.2 Пример индивидуального задания

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

```
1 reg sw_event;  
2 always @(SW) begin  
3     if ((SW[0] + SW[1] + SW[2] + SW[3]  
4         + SW[4] + SW[5] + SW[6] + SW[7]  
5         + SW[8] + SW[9]) > 4'd3) sw_event <= 1'b1;  
6     else sw_event <= 1'b0;  
7 end  
8  
9 reg [2:0] event_sync_reg;  
10 wire synced_event;  
11 assign synced_event = event_sync_reg[1]  
12                      & ~event_sync_reg[0];  
13  
14 always @(posedge CLK50) begin  
15     event_sync_reg[2] <= sw_event;  
16     event_sync_reg[1:0] <= event_sync_reg[2:1];  
17 end
```

Листинг 2.7: Решение индивидуального задания
(фрагмент кода лабораторной работы)

2.3 Вопросы к защите лабораторной работы

1. Какие элементы памяти вы изучили в данной лабораторной работе?
2. Чем отличается RS-защелка от D-защелки?
3. Какие входы могут быть у триггера? Перечислите все и назовите их функции.

4. Какие блоки вашего цифрового устройства синхронные? Какие нет? Почему?
5. Какой фрагмент вашего кода описывает вывод значения счетчика на семи сегментный индикатор? Как называется эта цифровая схема?
6. Продемонстрируйте код реализующий индивидуальное задание.
7. Покажите в коде лабораторной код счётчика.
8. Что такое сигнал синхронизации?