

СХЕМЫ С ТАКОВЫМ СИГНАЛОМ И СОСТОЯНИЕМ



ШКОЛА СИНТЕЗА
ЦИФРОВЫХ СХЕМ

ПРИ ПАРТНЕРСТВЕ

Занятие №3

19 октября 2024





Александр Силантьев

Руководитель лаборатории НИЛ ЭСК Университета МИЭТ

Окончил МИЭТ в 2014 году

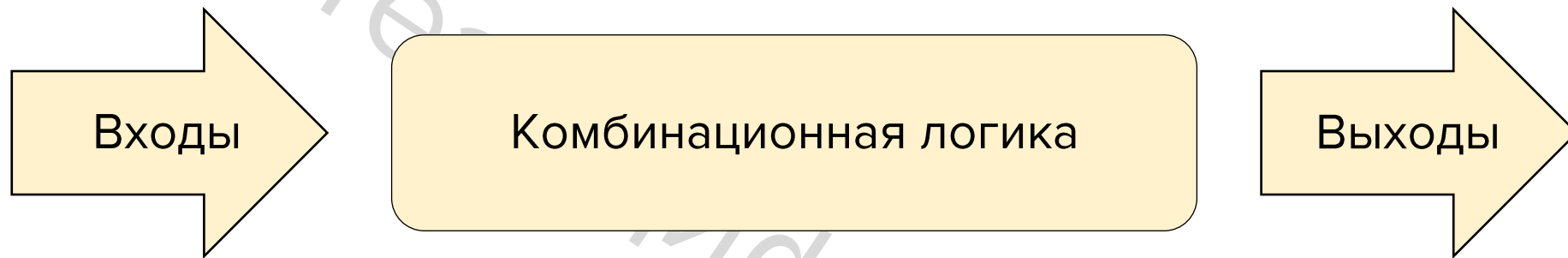
13-летний опыт инженерной деятельности в области проектирования IP-ядер и СнК

Старший преподаватель института МПСУ МИЭТ

Организатор первого в России инженерного хакатона по микроэлектронике и системам на кристалле

С 2014 года организатор семинаров, школ и олимпиад по популяризации электроники среди студентов и школьников

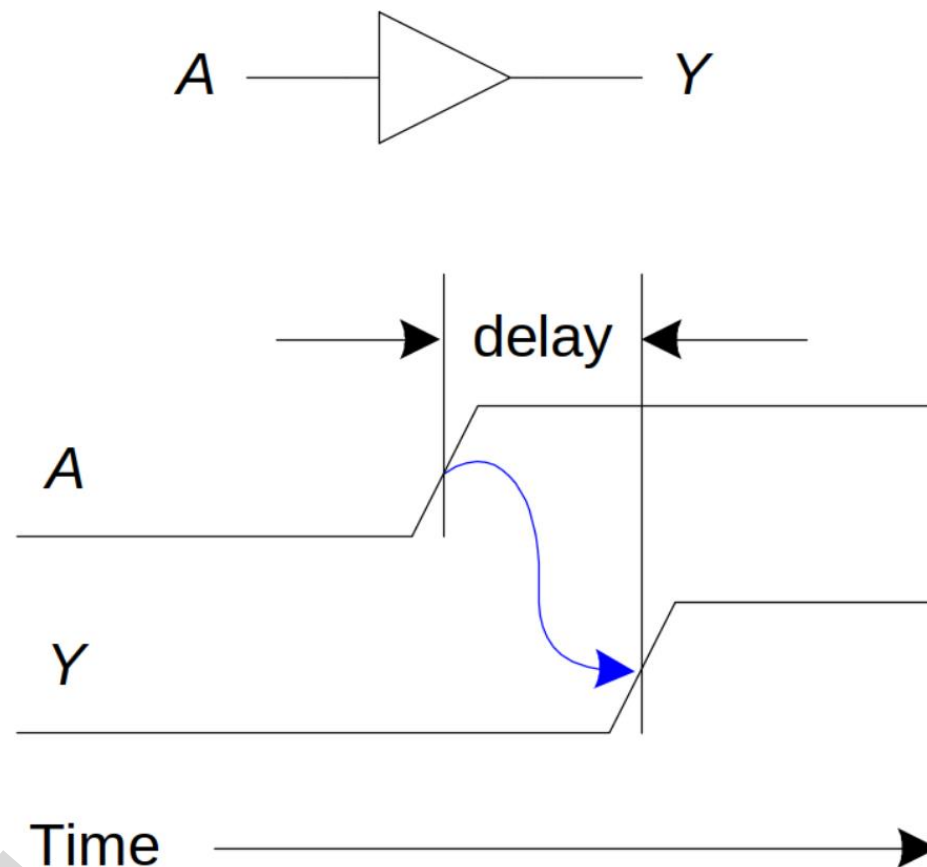
Комбинационная логика



Проблема вычислений в комбинационной логике.

Вычисления начинаются при изменении входных сигналов логики

Как понять, когда результат на выходе комбинационной логики будет готов для дальнейших вычислений?



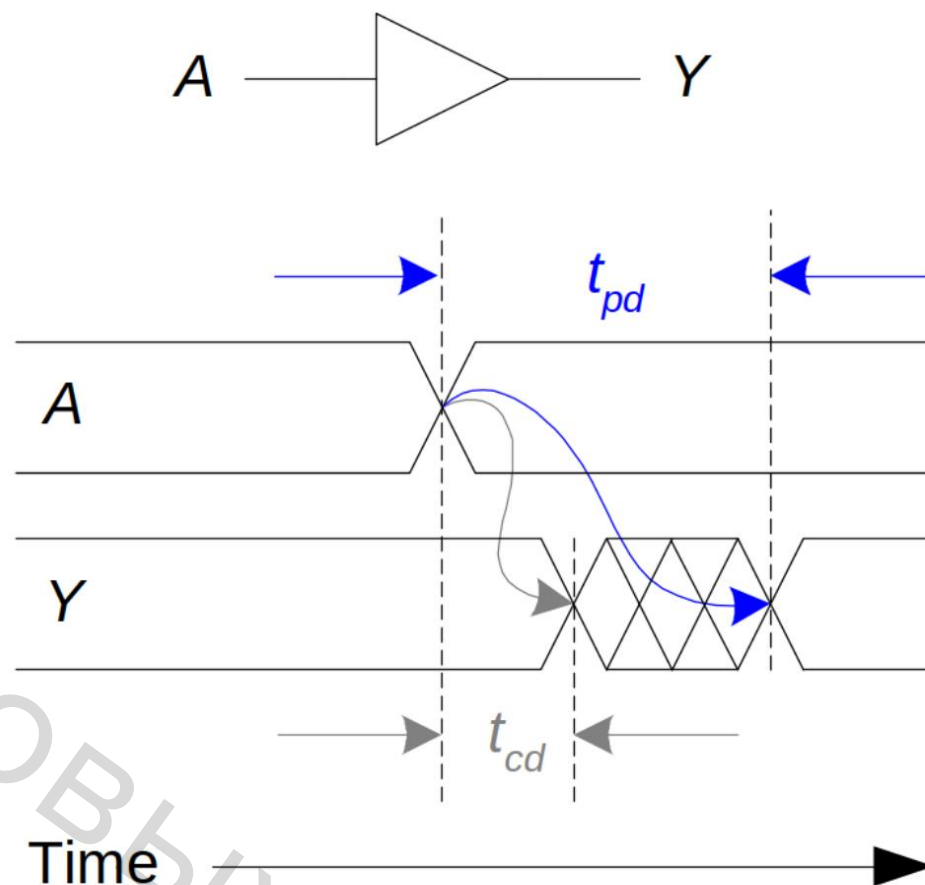
Contamination and propagation delays.

Contamination delay

Входы изменились, но на выходе результат пока что нестабильный

Propagation delay

Стабильный результат на выходе комбинационной схемы.

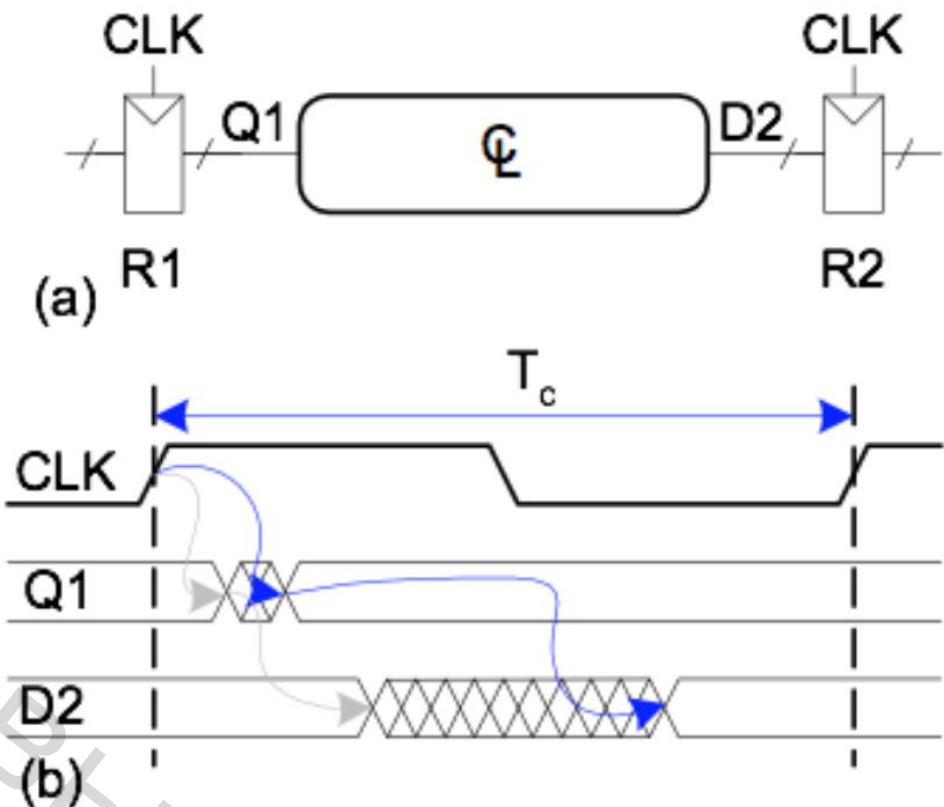


Использование тактового сигнала

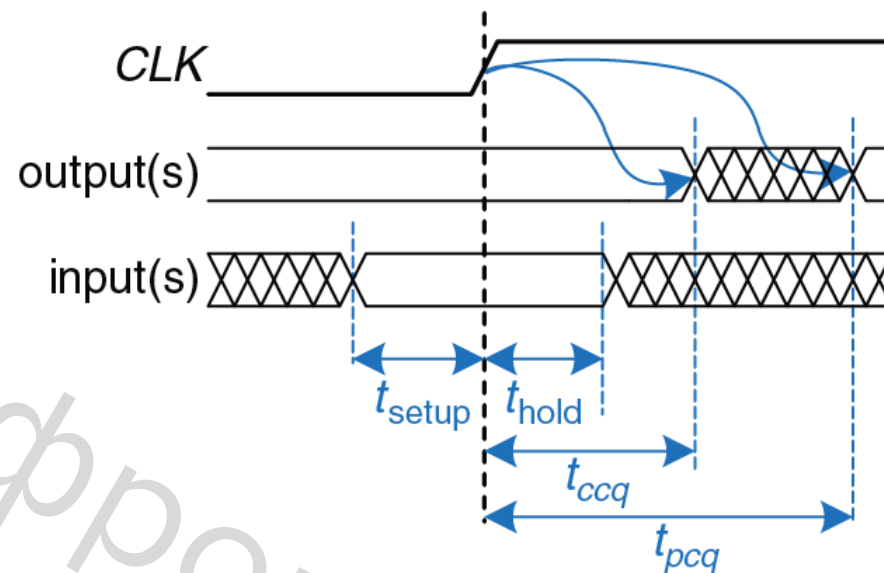
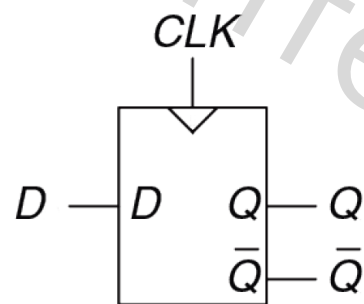
Перед завершением вычислений выходные данные могут содержать случайные значения

Как логике узнать, когда результаты готовы и могут использоваться на следующем этапе вычислений?

Вычисления можно синхронизировать с помощью специального сигнала – сигнала тактирования

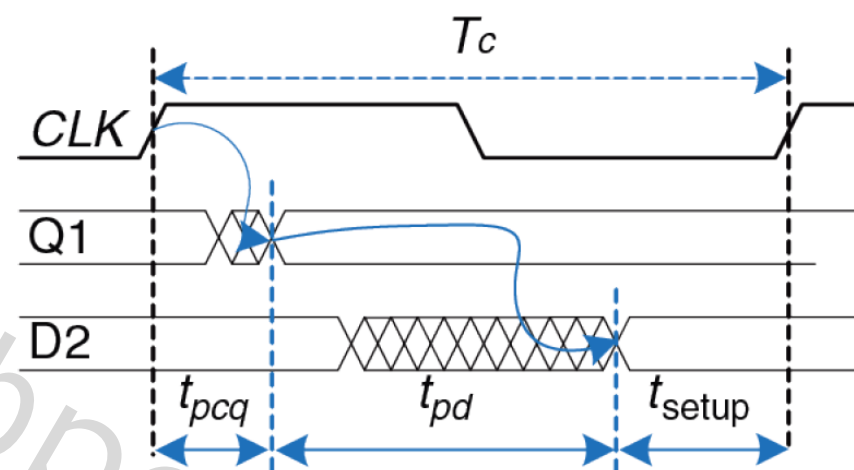
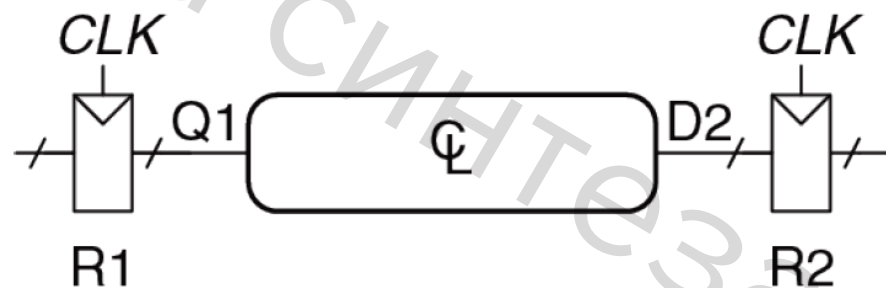


Временные характеристики



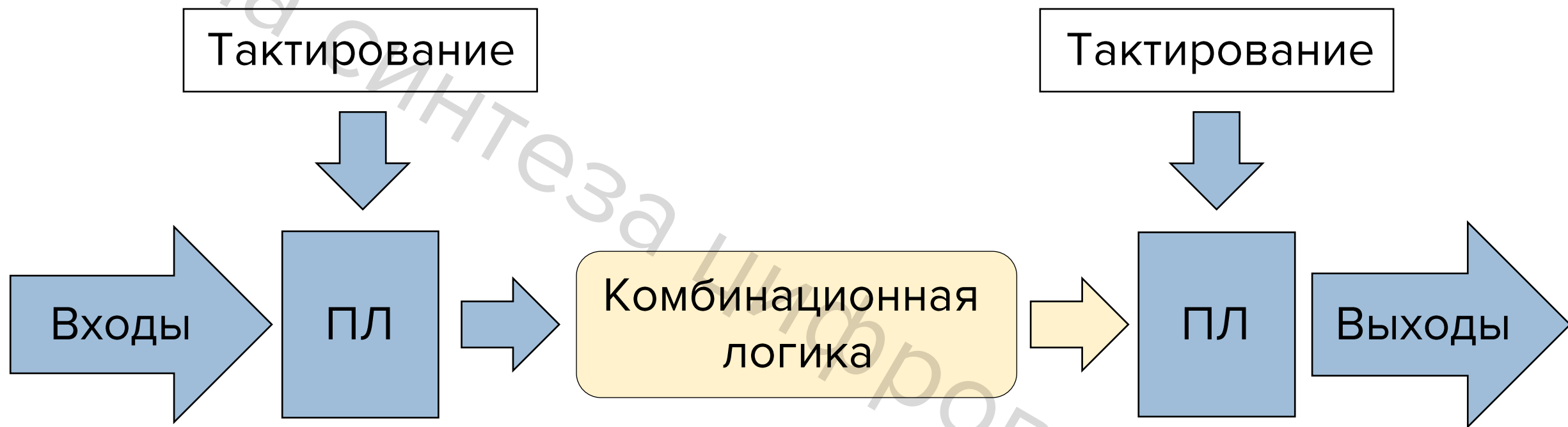
картинка из книги Харрис & Харрис «Цифровая схемотехника и архитектура компьютера»

Временные характеристики

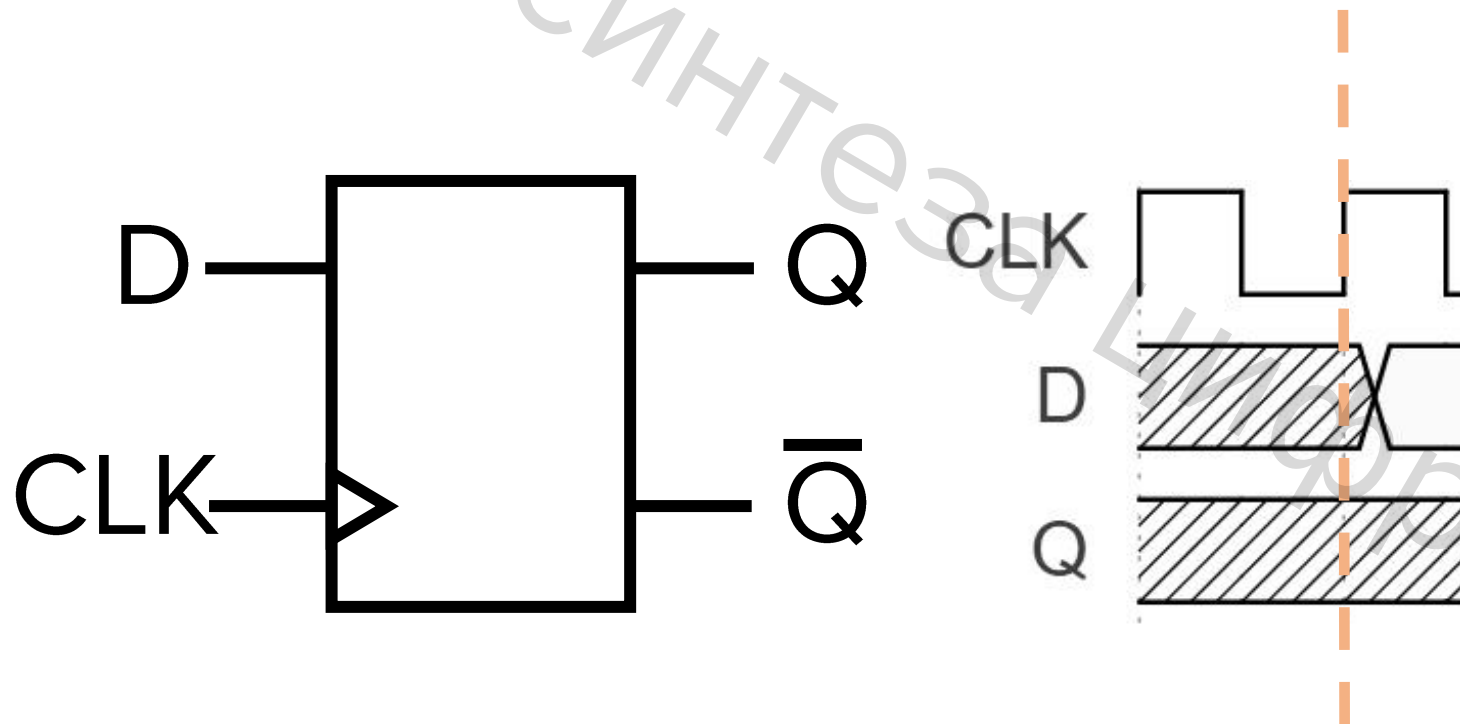


книга Харрис & Харрис «Цифровая схемотехника и архитектура компьютера»

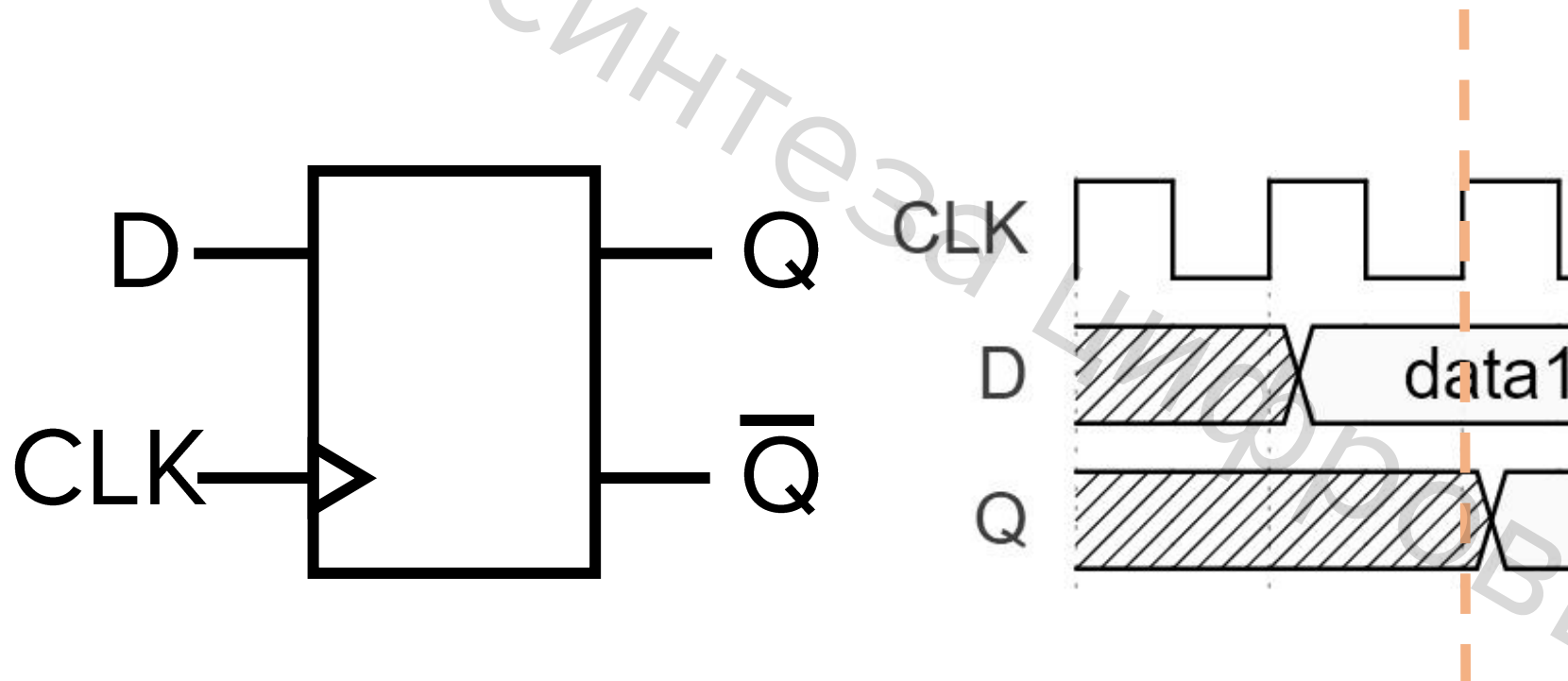
Последовательная логика



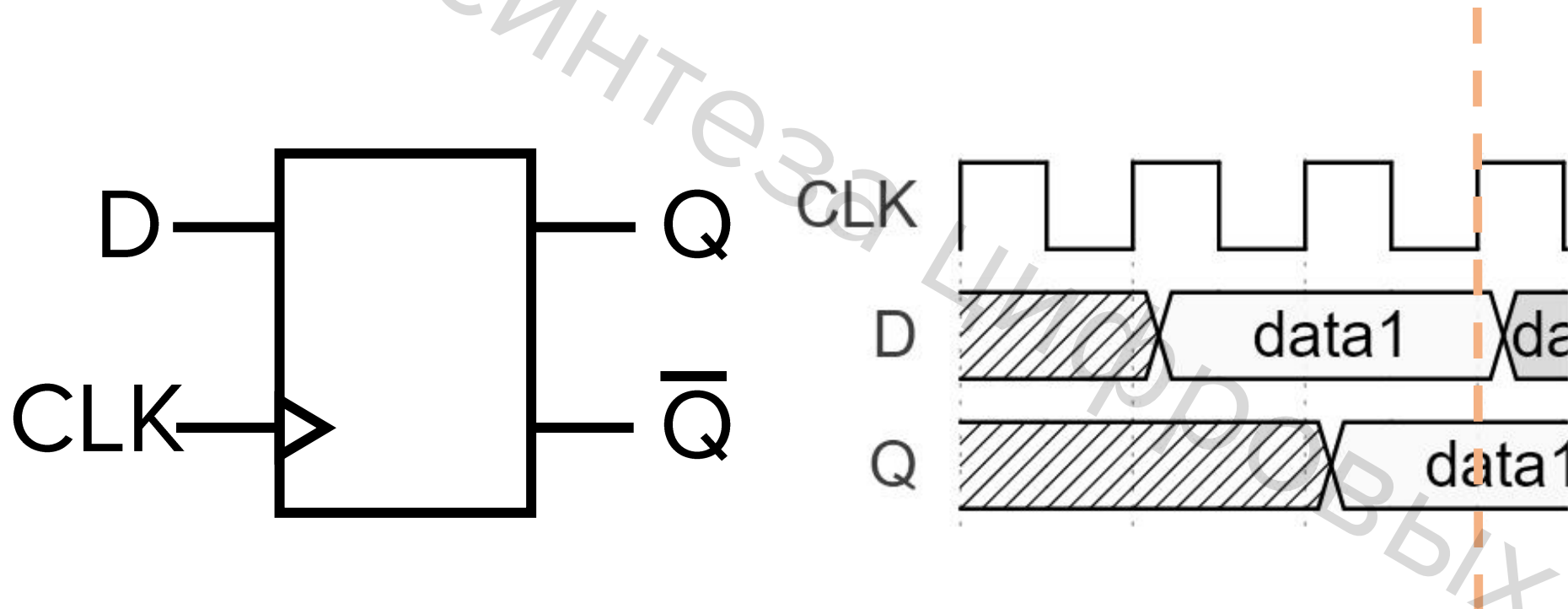
D-триггер



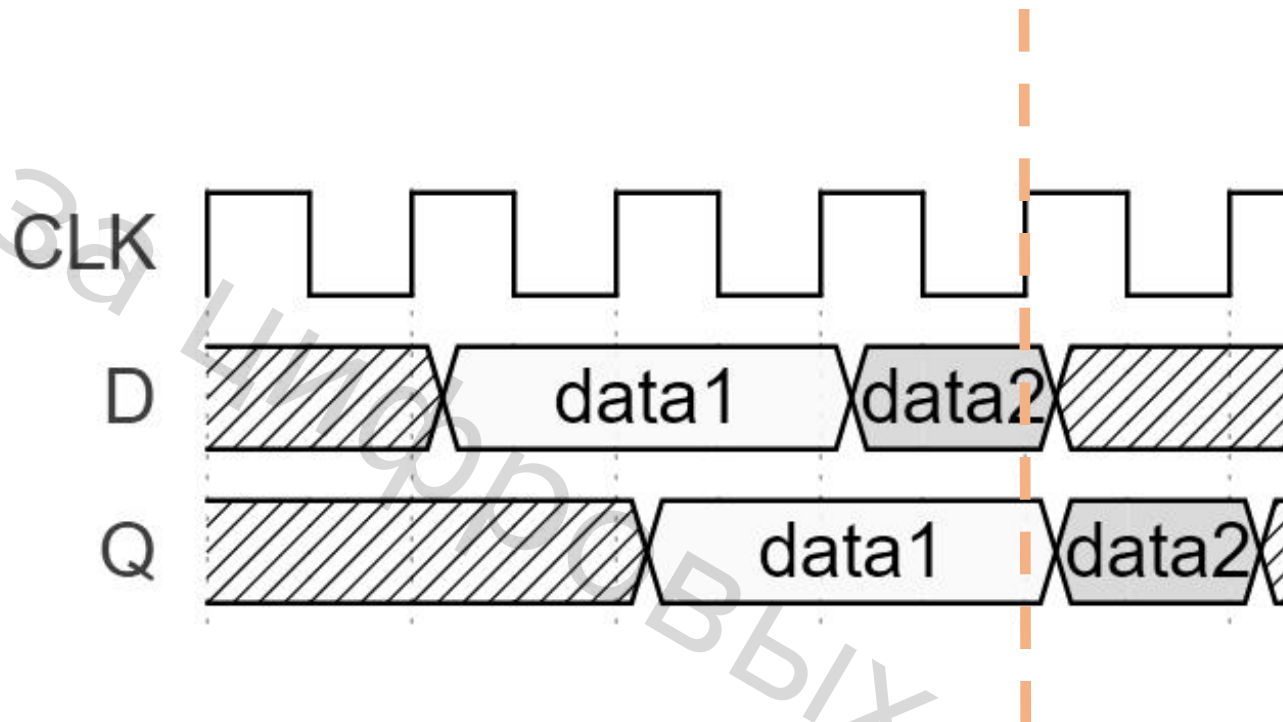
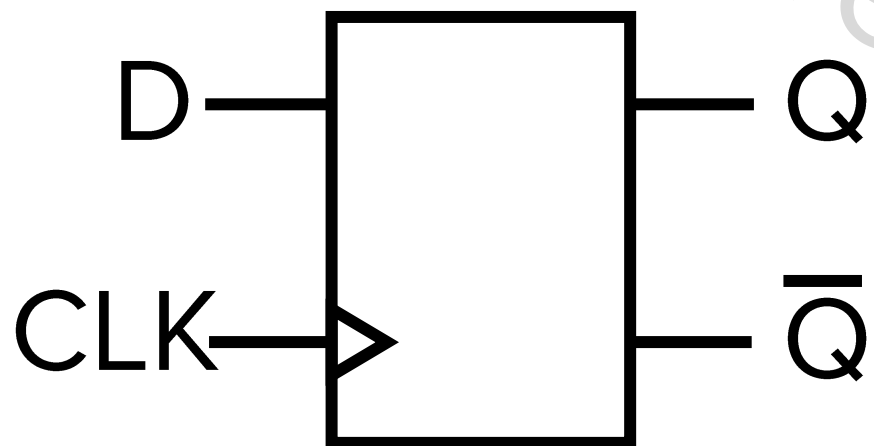
D-триггер



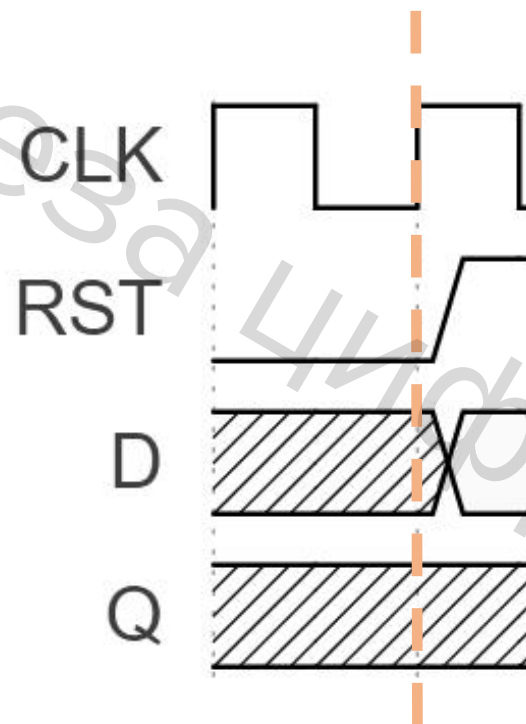
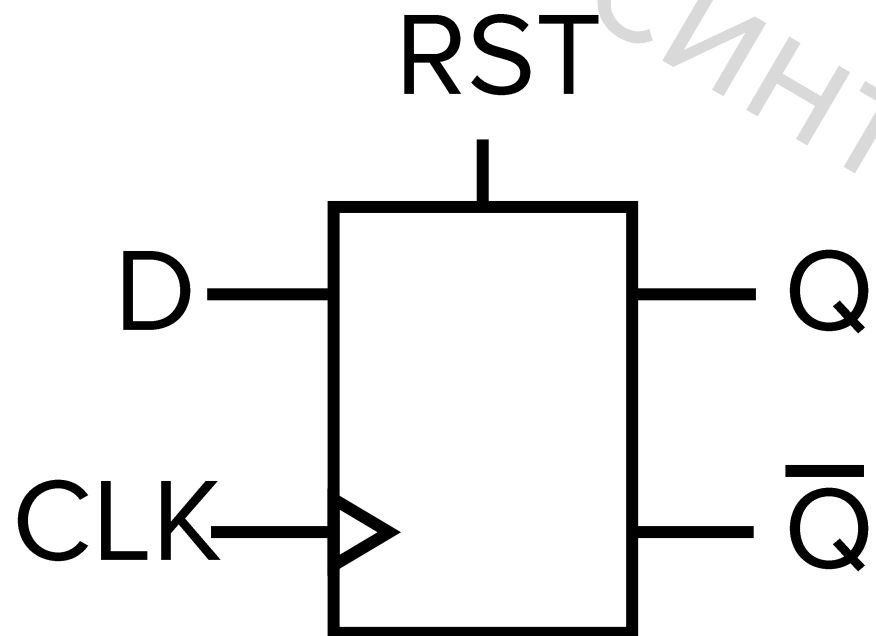
D-триггер



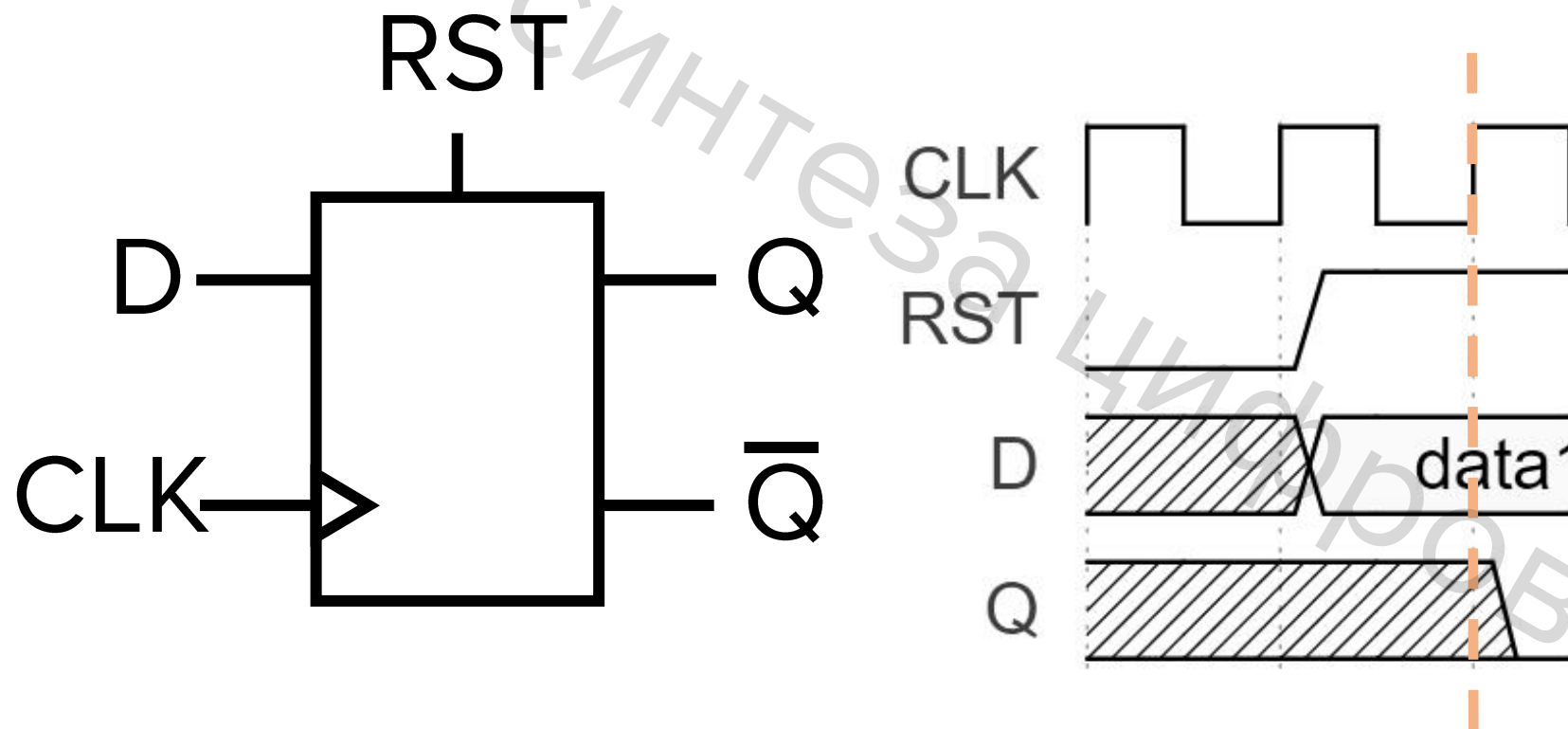
D-триггер



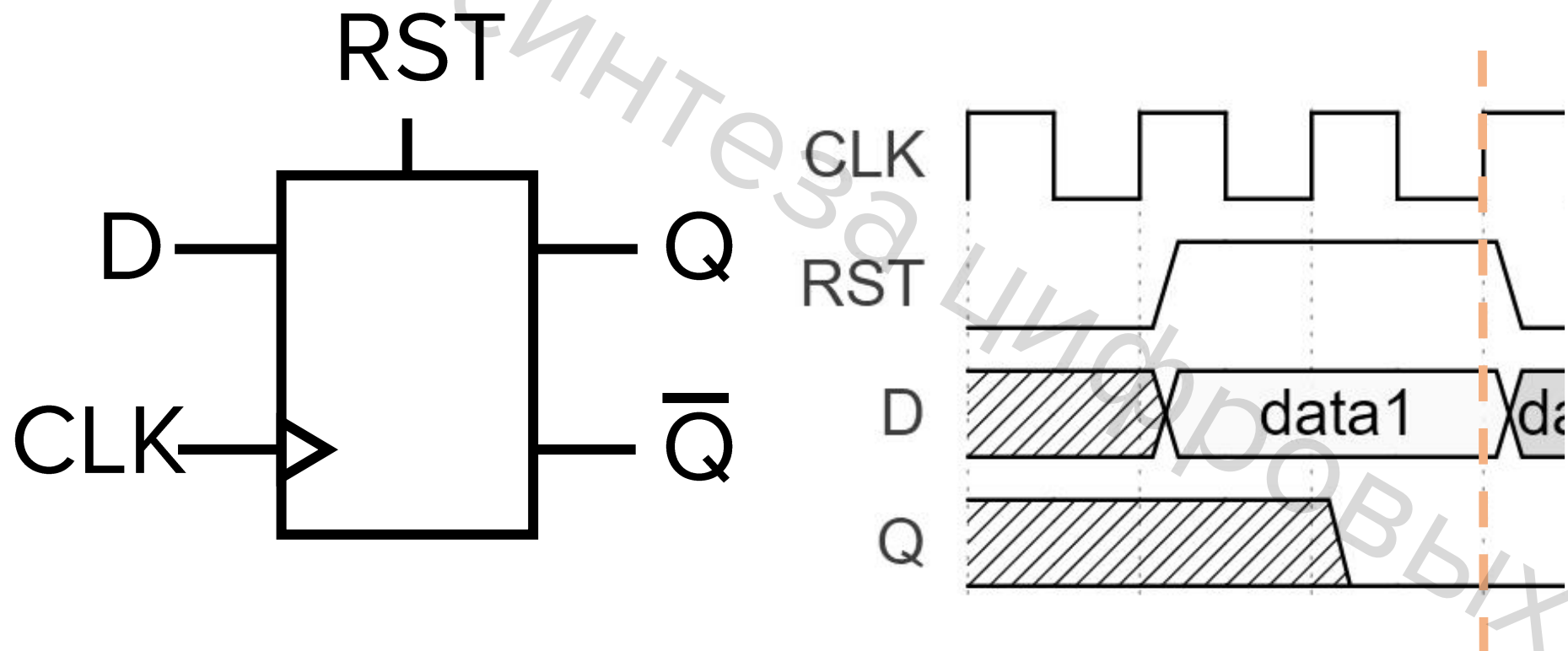
D-триггер. Сигнал сброса



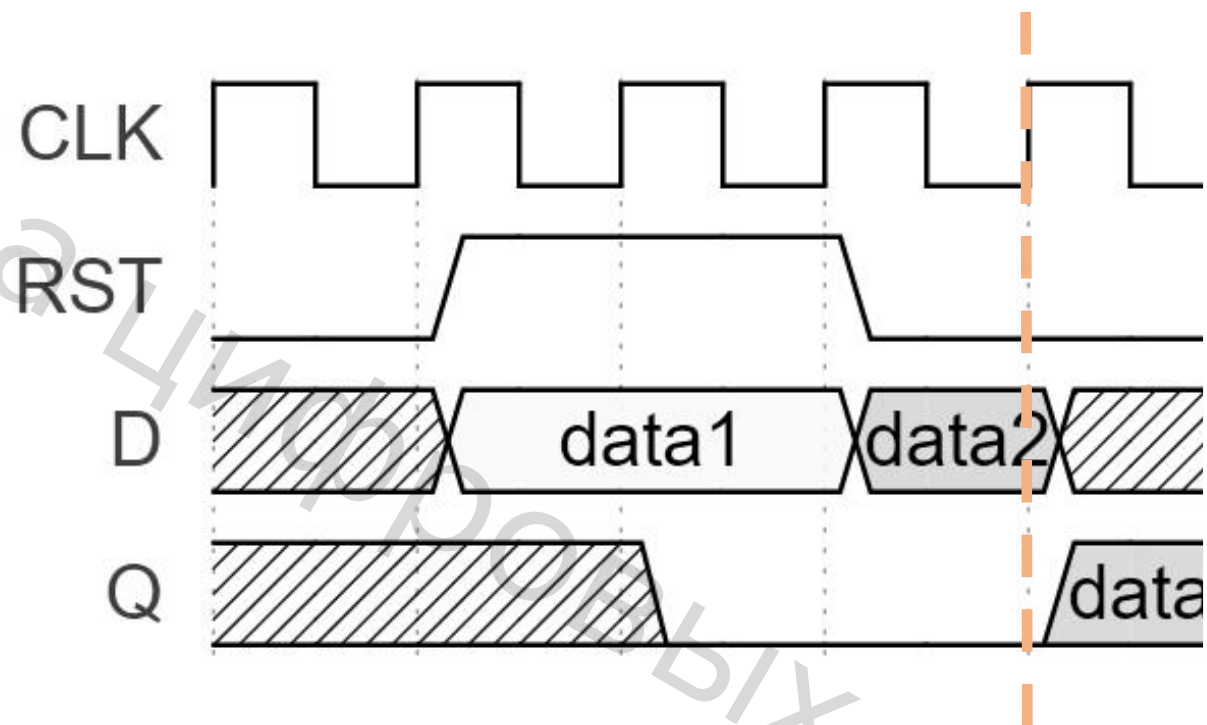
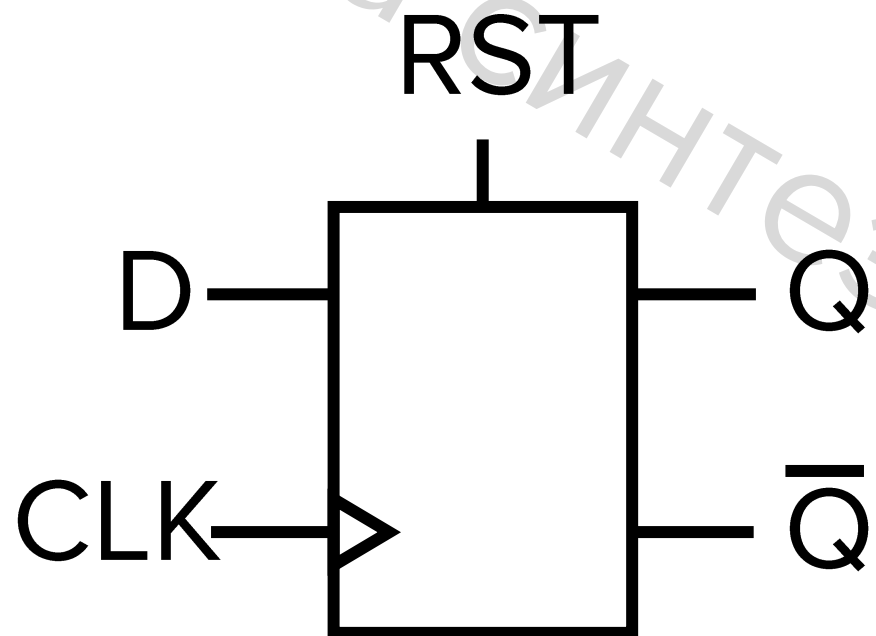
D-триггер. Сигнал сброса



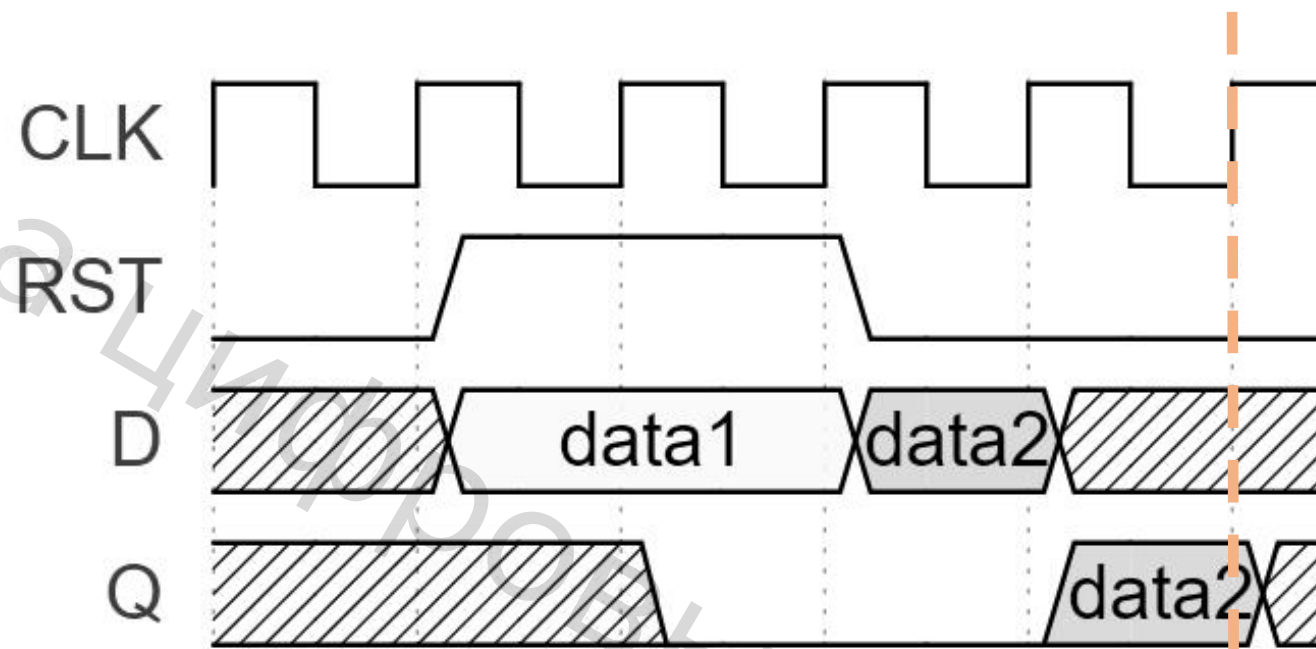
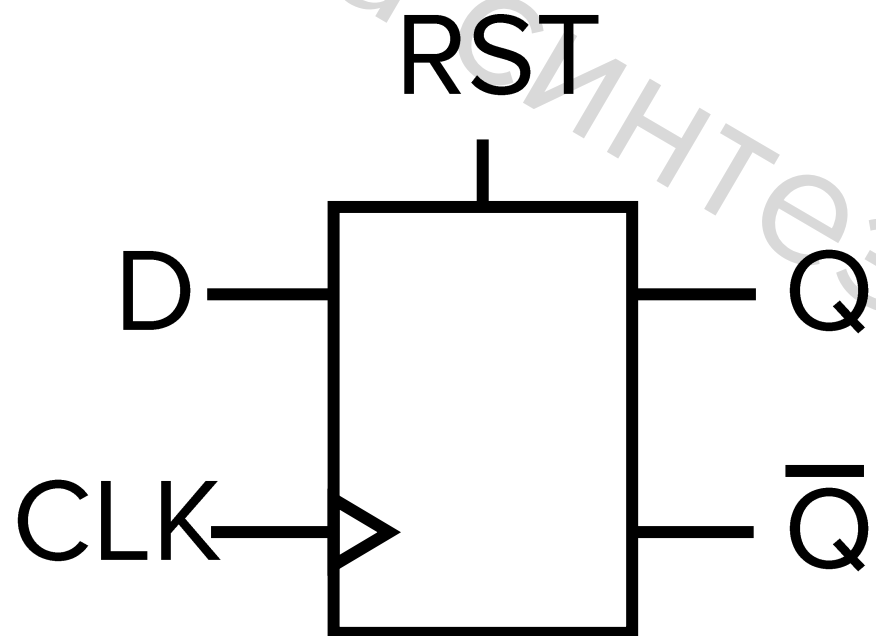
D-триггер. Сигнал сброса



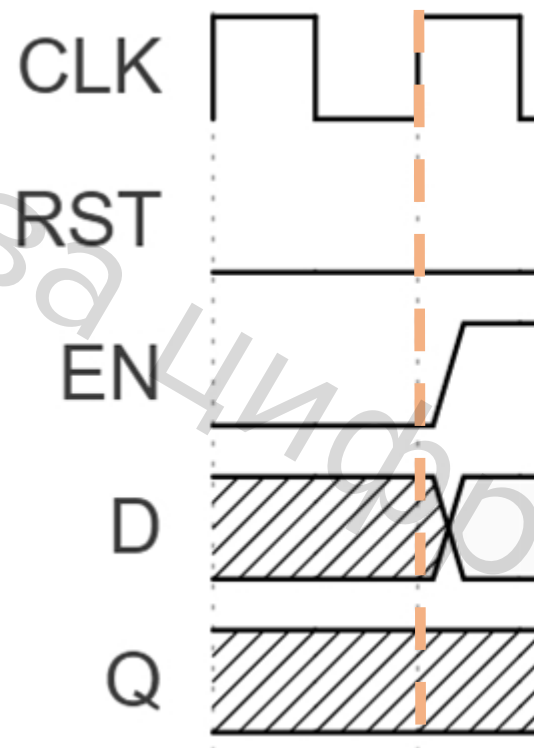
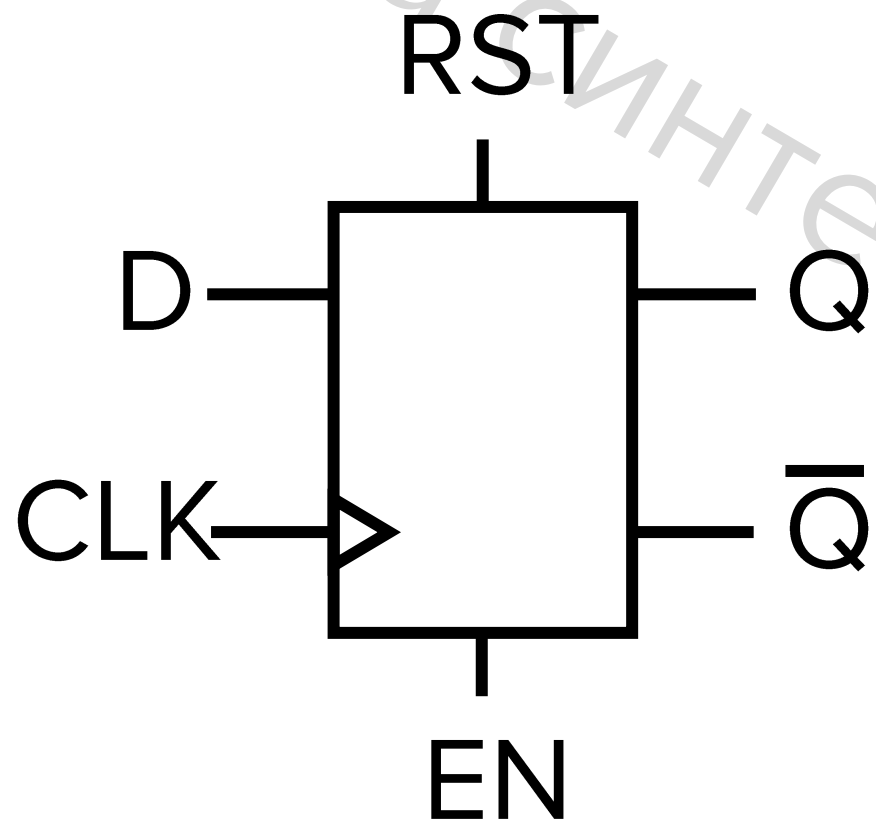
D-триггер. Сигнал сброса



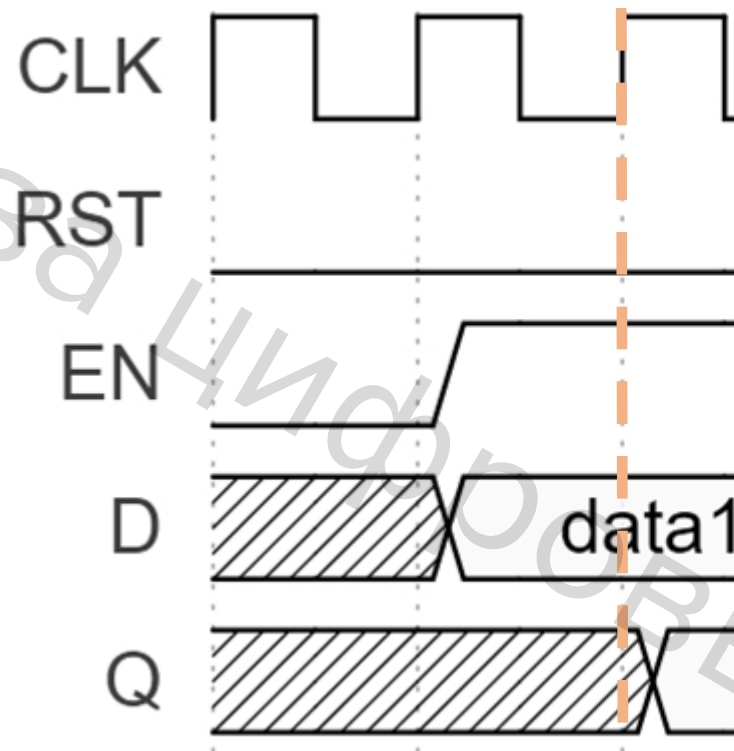
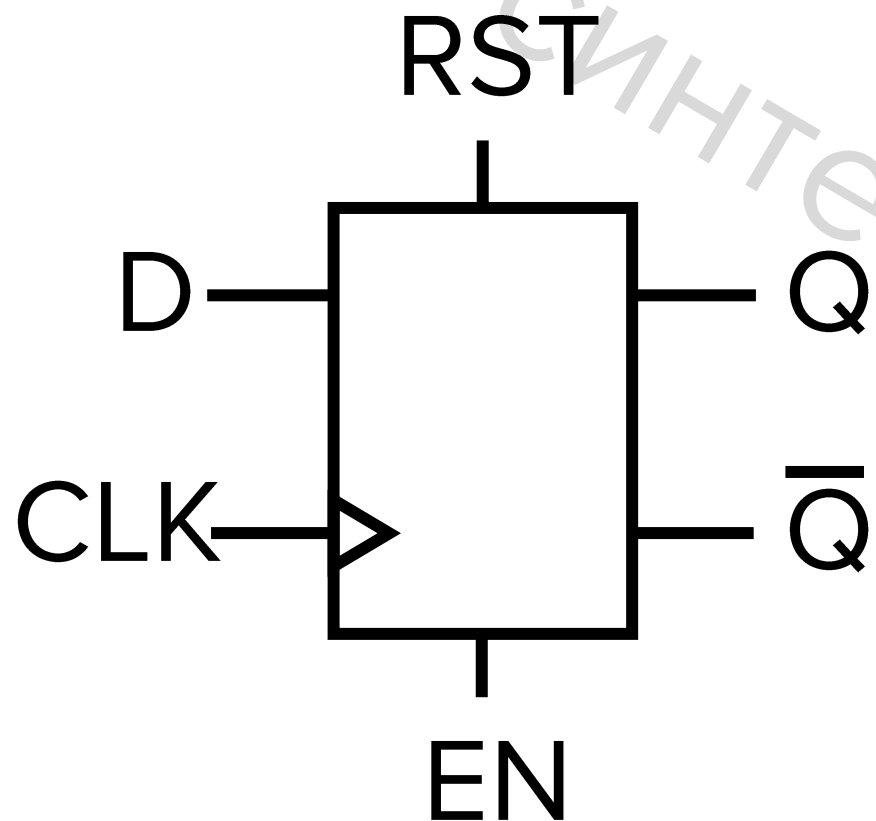
D-триггер. Сигнал сброса



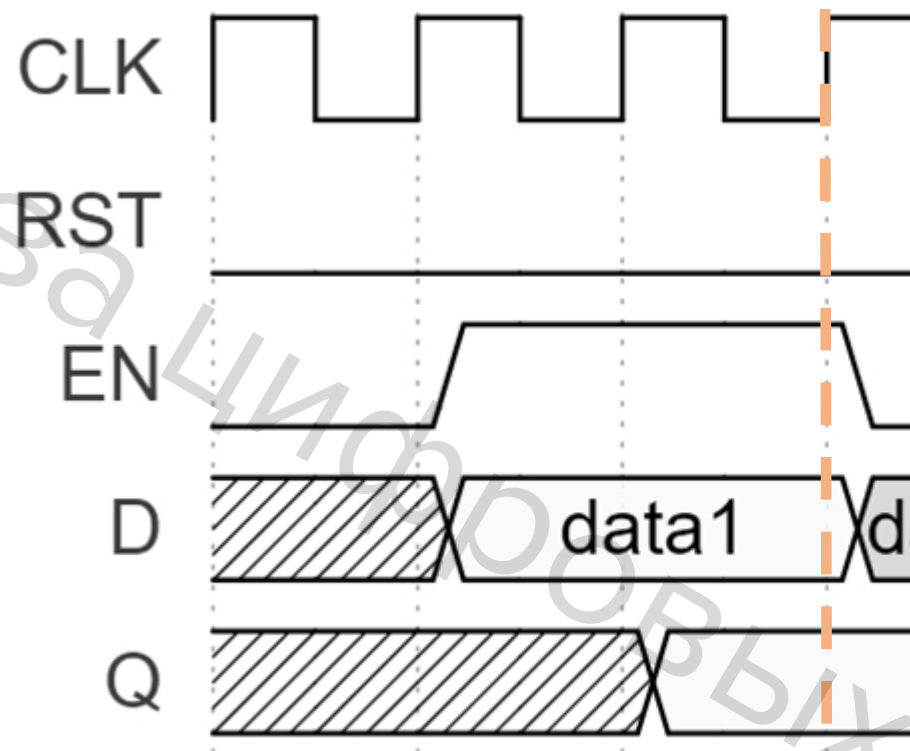
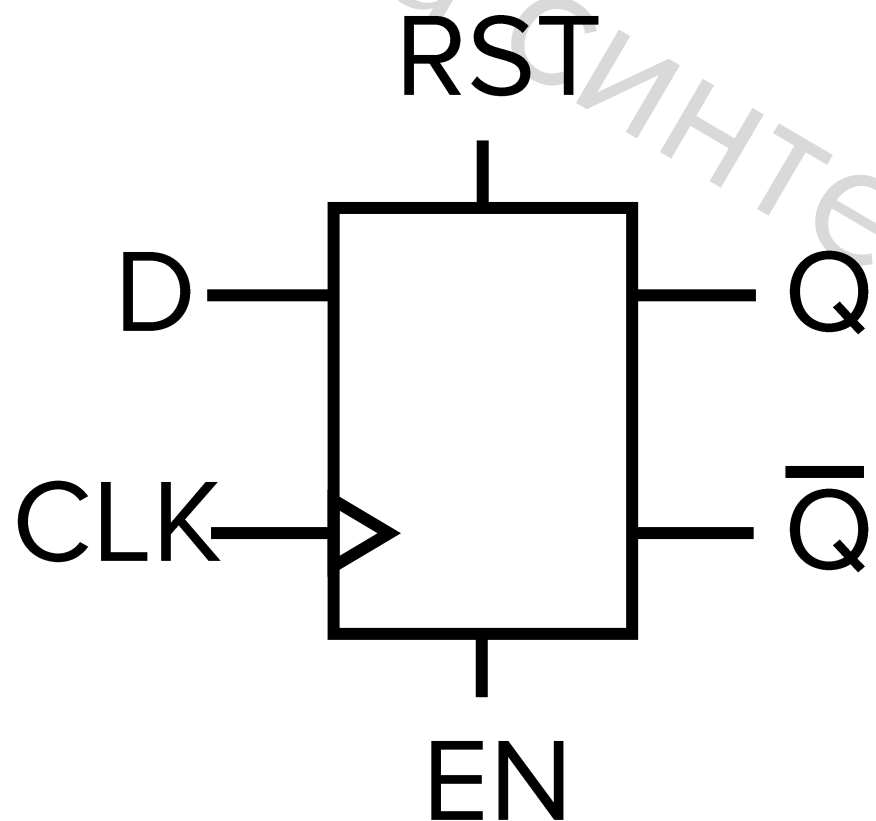
D-триггер. Сигнал разрешения



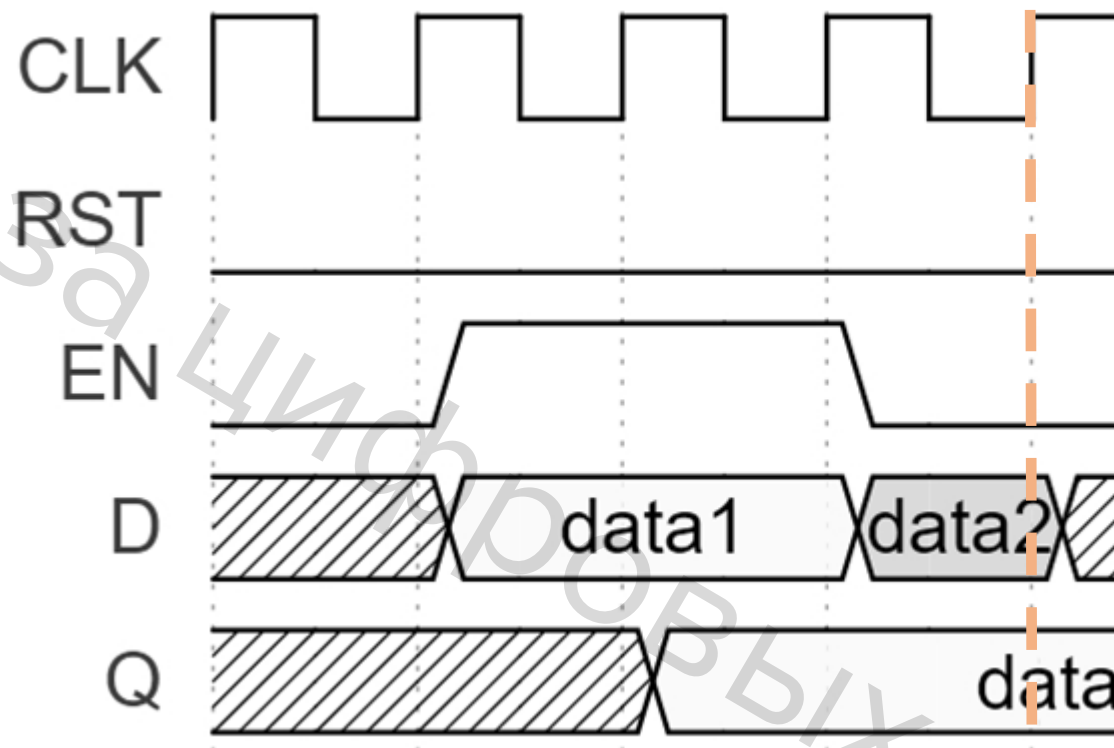
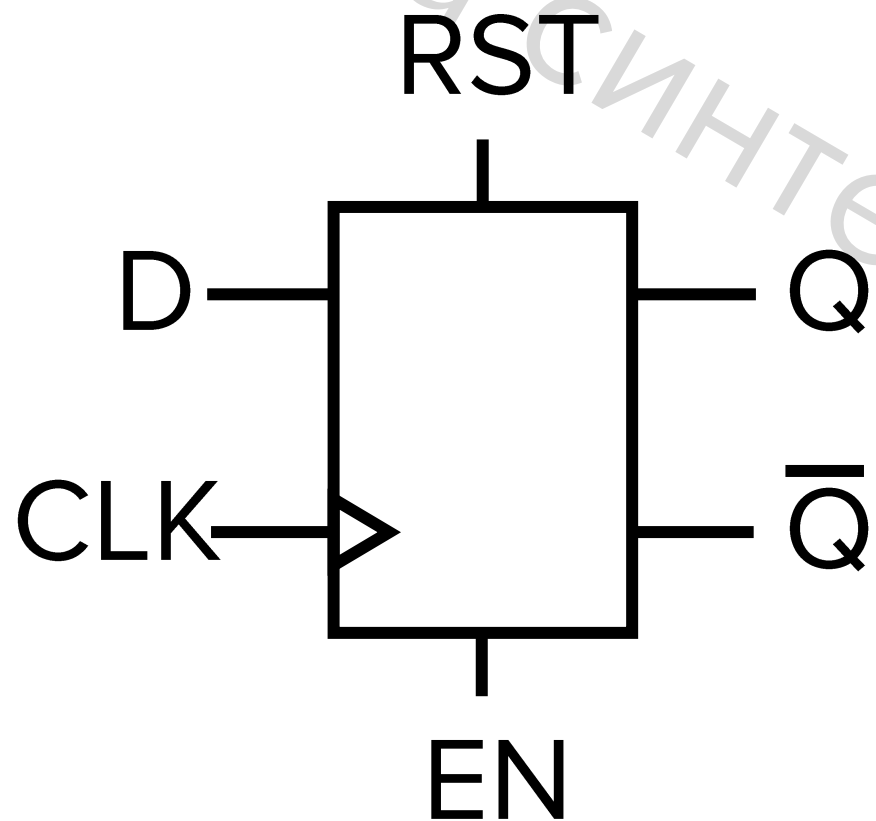
D-триггер. Сигнал разрешения



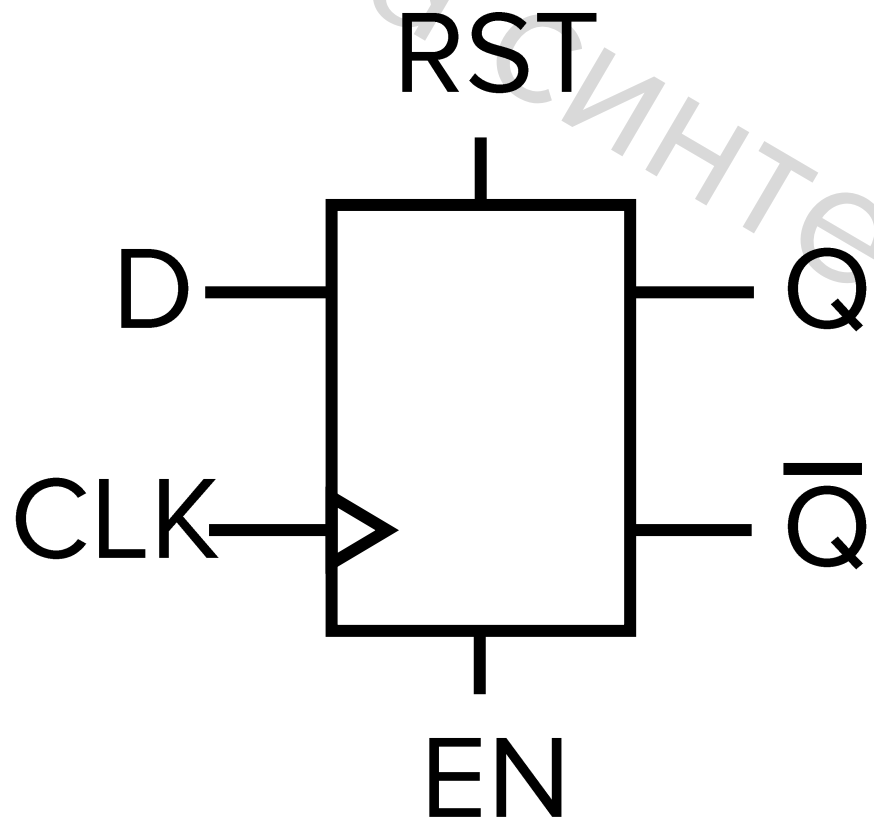
D-триггер. Сигнал разрешения



D-триггер. Сигнал разрешения

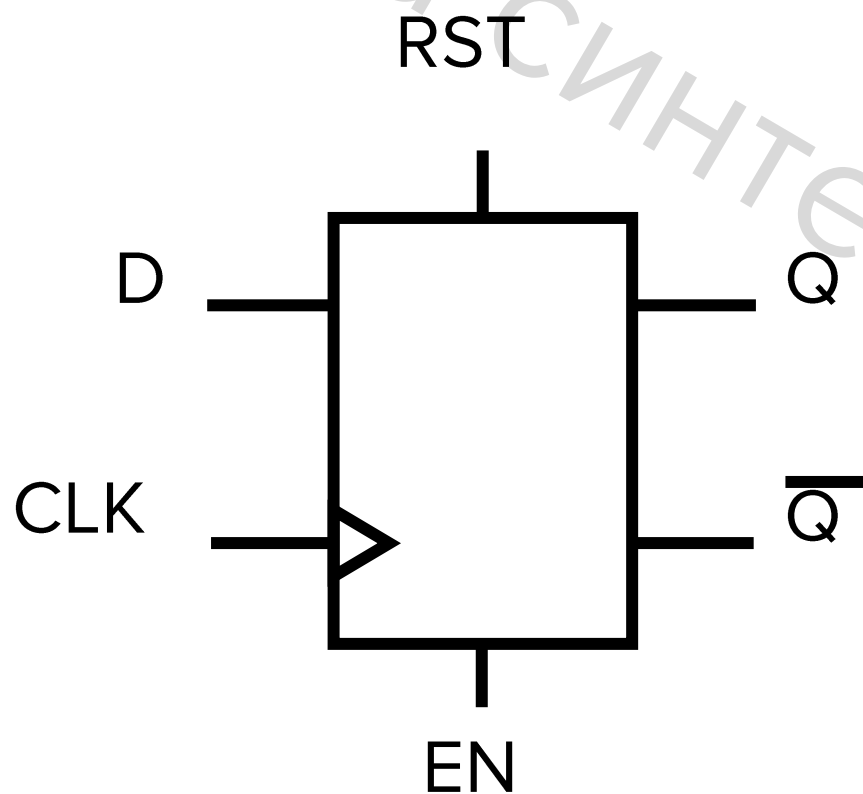


D-триггер. Синхронный сброс



```
module my_reg
(
    input clk_i,
    input rst_i,
    input d_i,
    output q_o
);
    logic out;
    always_ff @(posedge clk_i) begin
        if(rst_i) out <= 1'b0; else out <= d_i;
    end
    assign q_o = out;
endmodule
```

D-триггер. Асинхронный сброс



```
module my_reg  
(  
    input clk_i,  
    input rst_i,  
    input d_i,  
    output q_o  
);
```

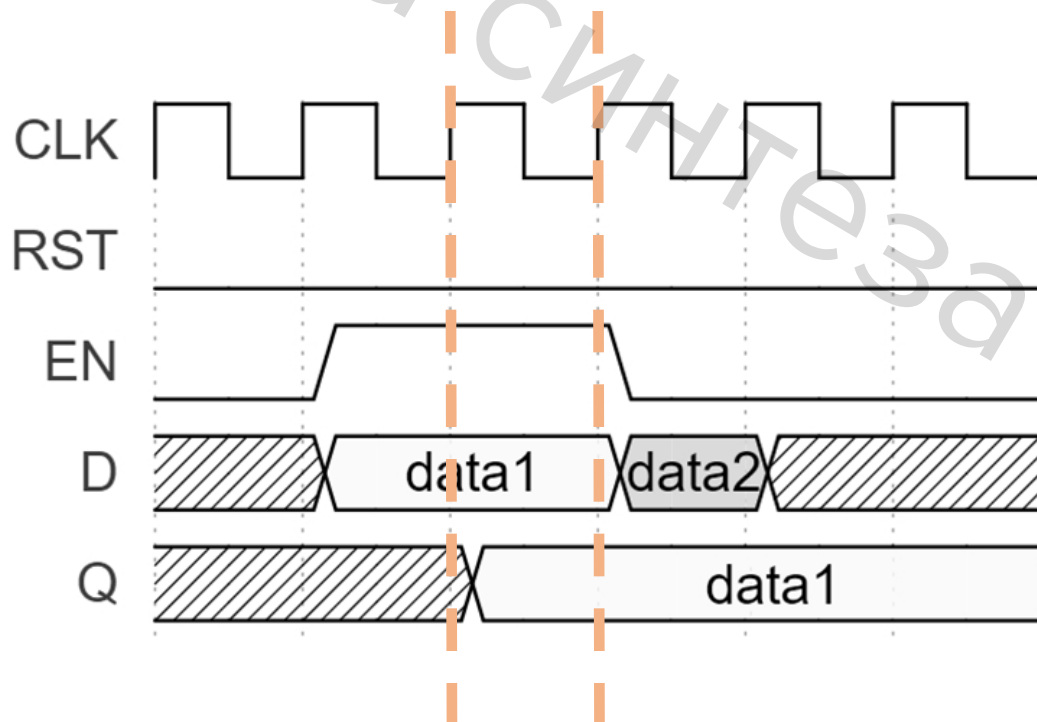
```
    logic out;  
    always_ff @(posedge clk_i or posedge rst_i)  
    begin  
        if(rst_i) out <= 1'b0; else out <= d_i;  
    end  
    assign q_o = out;  
endmodule
```

Указание условия
асинхронного сброса

Правила наименования сигнала сброса

- Сброс может обозначаться в сокращенном или полном виде: **reset** или **rst**
- Префикс в названии сброса указывает на его тип: **arst** — асинхронный сброс, **rst** — синхронный
- Постфикс указывает на активный уровень сброса (то значение, при котором триггер находится в состоянии сброса): **arstn** — асинхронный сброс с активным уровнем 0, **arst** — асинхронный сброс с активным уровнем 1

D-триггер. EN



```
module my_reg
```

```
(
```

```
    input clk_i,
```

```
    input rst_i,
```

```
    input en_i,
```

```
    input d_i,
```

```
    output q_o
```

```
);
```

```
    logic out;
```

```
    always_ff @(posedge clk_i or posedge rst_i)
```

```
    begin
```

```
        if(rst_i) out <= 1'b0; else
```

```
        if(en_i) out <= d_i;
```

```
    end
```

```
    assign q_o = out;
```

```
endmodule
```

Запись в триггер,
если EN = 1

Правила описания триггеров для начинающих

- Для описания триггеров используйте **неблокирующее** присваивание `<=`
- **Сброс обязателен для всех сигналов контроля**. Он необязателен для сигналов данных, если с данными передается специальный бит (**valid**) который говорит, содержат ли данные корректную информацию. При этом сам сигнал **valid** нужно использовать с сбросом, так как он является контрольным.
- **Нельзя** делать присваивание в один триггер в разных блоках **always**
- Описание сброса самое **приоритетное** действие в **always** блоке

Конструкция Generate

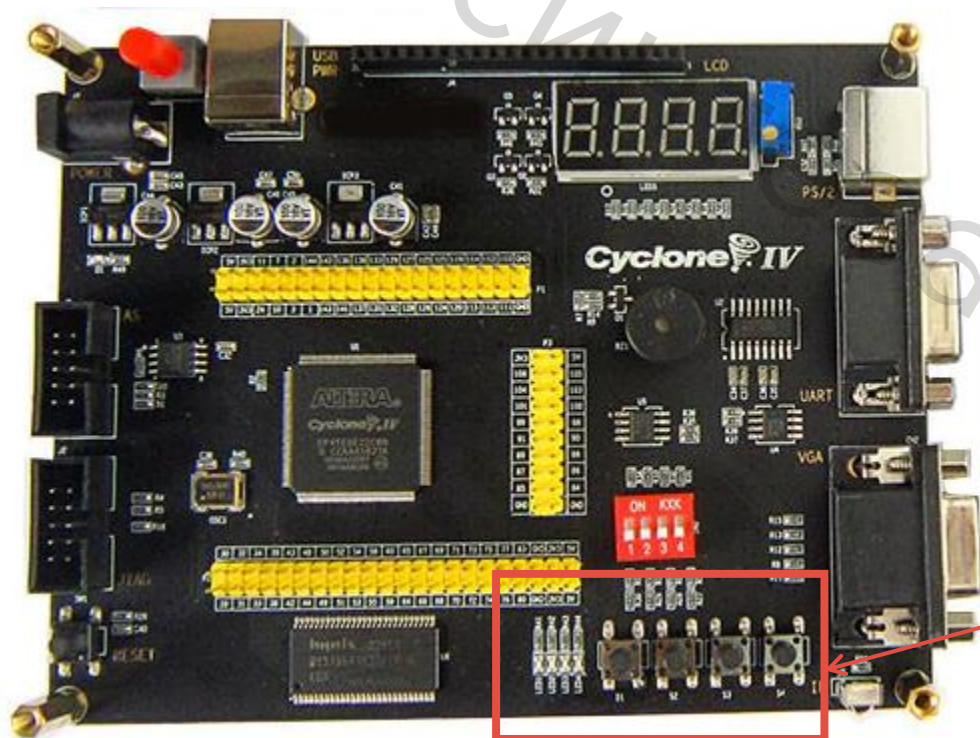
```
module add_gen (  
    input [2:0] A,  
    input [2:0] B,  
    output [2:0] S  
);  
  
genvar i;  
generate  
    for (i=0; i<3; i=i+1) begin : newgen  
        adder new (  
            .a(A[i]),  
            .b(B[i]),  
            .s(S[i])  
        );  
    end  
endgenerate  
  
endmodule
```

≡

```
adder new0 (  
    .a(A[0]),  
    .b(B[0]),  
    .s(S[0])  
);  
  
adder new1 (  
    .a(A[1]),  
    .b(B[1]),  
    .s(S[1])  
);  
  
adder new2 (  
    .a(A[2]),  
    .b(B[2]),  
    .s(S[2])  
);
```

i==0

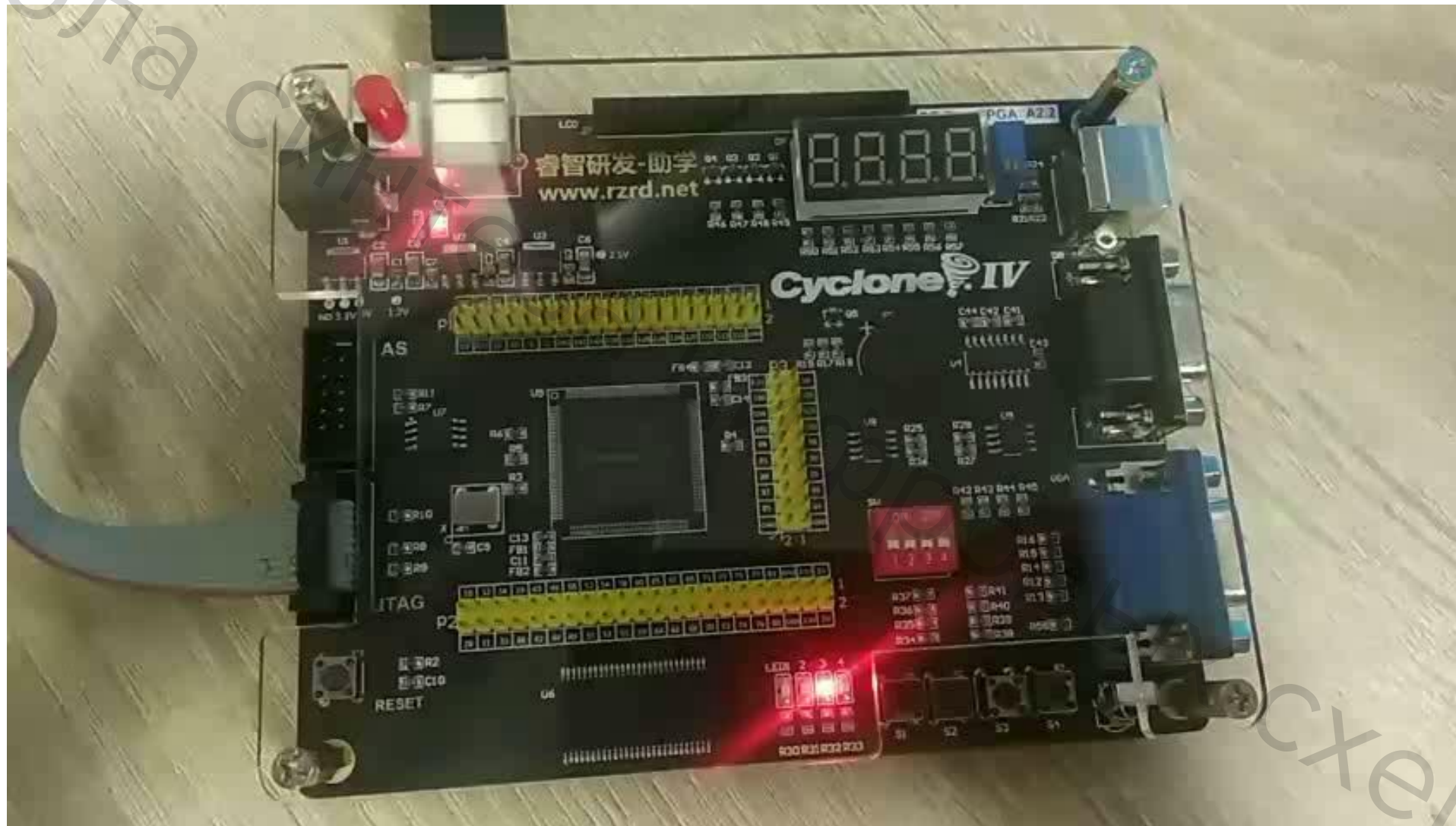
Упражнение со счетчиком 1_06_binary_counter



Вывод значений счетчика на **свето**диоды.
Использование **кноп**ок для изменения направления счета

ПОСЛЕДОВАТЕЛЬНАЯ ЛОГИКА НА ПЛИС. СХЕМЫ С ТАКОВЫМ СИГНАЛОМ И СОСТОЯНИЕМ

Упражнение со счетчиком 1_06_binary_counter




Упражнение со счетчиком 1_06_binary_counter

```
// Exercise 1: Free running counter.  
// How do you change the speed of LED blinking?  
// Try different bit slices to display.
```

```
localparam w_cnt = $clog2 (clk_mhz * 1000 * 1000);
```

```
logic [w_cnt - 1:0] cnt;
```



Объявление
многоразрядного
счетчика `cnt`

Упражнение со счетчиком 1_06_binary_counter

```
// Exercise 1: Free running counter.  
// How do you change the speed of LED blinking?  
// Try different bit slices to display.
```

```
localparam w_cnt = $clog2 (clk_mhz * 1000 * 1000);
```

```
logic [w_cnt - 1:0] cnt;
```

Объявление
многоразрядного
счетчика `cnt`

Вычисление параметра
разрядности счетчика
от текущей тактовой
частоты

Упражнение со счетчиком 1_06_binary_counter

```
// Exercise 1: Free running counter.  
// How do you change the speed of LED blinking?  
// Try different bit slices to display.
```

```
localparam w_cnt = $clog2 (clk_mhz * 1000 * 1000);
```

```
logic [w_cnt - 1:0] cnt;
```

Объявление
многоразрядного
счетчика `cnt`

Вычисление параметра
разрядности счетчика
от текущей тактовой
частоты

`$clog2` – встроенная системная функция вычисления логарифма по основанию 2.

`$clog2` – встроенная системная функция вычисления логарифма по основанию 2, с округлением вверх. Результат `$clog2` равен количеству бит, в которые можно вместить числа от 0 до N-1, где N - аргумент `$clog2`

`$clog2` можно использовать в синтезируемом коде только тогда, когда ее параметр - это **константное выражение**, которое состоит из **числовых констант**, а также **parameter**, **localparam** и **`define** выражений, которые сводятся к константам.

Все системные функции в SystemVerilog начинаются с `$`

Упражнение со счетчиком 1_06_binary_counter

```
always_ff @ (posedge clk or posedge rst) ← Асинхронный сброс
    if (rst)
        cnt <= '0;
    else
        cnt <= cnt + 1'd1;

assign led = cnt [$left (cnt) -: w_led];
```

Упражнение со счетчиком 1_06_binary_counter

```
always_ff @ (posedge clk or posedge rst)
    if (rst)
        cnt <= '0;
    else
        cnt <= cnt + 1'd1;

assign led = cnt [$left (cnt) -: w_led];
```

← Активный уровень сброса 1

Упражнение со счетчиком 1_06_binary_counter

```
always_ff @ (posedge clk or posedge rst)
    if (rst)
        cnt <= '0;
    else
        cnt <= cnt + 1'd1;

assign led = cnt [$left (cnt) -: w_led];
```

Начальное значение счетчика 0

Упражнение со счетчиком 1_06_binary_counter

```
. always_ff @ (posedge clk or posedge rst)
    if (rst)
        cnt <= '0;
    else
        cnt <= cnt + 1'd1;

assign led = cnt [$left (cnt) -: w_led];
```

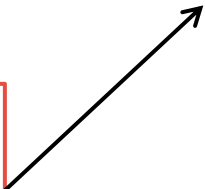
Инкрементация счетчика каждый такт на 1, если нет сигнала активного сброса. При достижении максимального значения переполняется и переходит в значение 0. Если счетчик 4 разрядный и равен 4'b1111, то следующее значение 0

Упражнение со счетчиком 1_06_binary_counter

```
always_ff @ (posedge clk or posedge rst)
    if (rst)
        cnt <= '0;
    else
        cnt <= cnt + 1'd1;

assign led = cnt [$left (cnt) -: w_led];
```

На светодиоды выводится диапазон разрядов счетчика. Чем больше индекс разряда, тем с меньшей частотой мигает светодиод



Упражнение со счетчиком 1_06_binary_counter

```
always_ff @ (posedge clk or posedge rst)
    if (rst)
        cnt <= '0;
    else
        cnt <= cnt + 1'd1;

assign led = cnt [$left (cnt) -: w_led];
```

На светодиоды выводится диапазон разрядов счетчика. Чем больше индекс разряда, тем с меньшей частотой мигает светодиод.

`$left(cnt)` возвращает номер старшего индекса в `cnt`. Если `cnt` 8-разрядный, то вернется 7. Если `$left (cnt) = 7` а `w_led = 3` то `[$left (cnt) -: w_led]` эквивалентно `[7:5]`

Упражнение со счетчиком 1_06_binary_counter

```
logic [31:0] a_vect;  
logic [0:31] b_vect;  
logic [63:0] dword;  
integer sel;  
  
a_vect[0  +:8]      // ==a_vect[7:0]  
a_vect[15 -:8]     // ==a_vect[15:8]  
  
b_vect[0  +:8]      // ==b_vect[0:7]  
b_vect[15 -:8]     // ==b_vect[8:15]  
  
dword[8*sel +:8]    // variable part-select with  
fixed width
```

$w[x+:y]$ эквивалентно $w[x:(x+y-1)]$
 $w[x -:y]$ эквивалентно $w[x:(x-y+1)]$

`$left(cnt)` возвращает номер старшего Индекса в `cnt`. Если `cnt` 8-разрядный, То вернется 7.
Если `$left(cnt) = 7`, а `w_led = 3`, то `[$left(cnt) -: w_led]` эквивалентно `[7:5]`

Упражнение со счетчиком 1_06_binary_counter

```

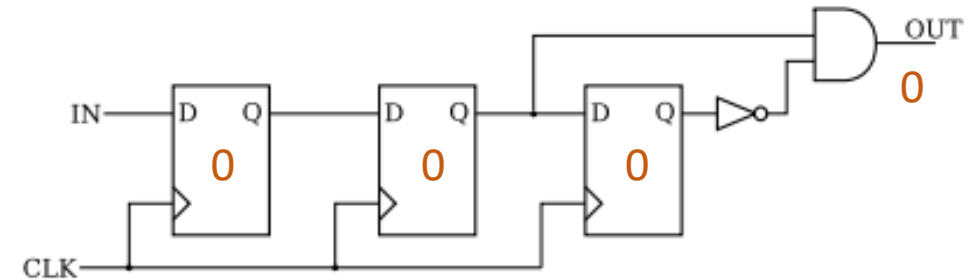
wire any_key = | key;

logic any_key_r;

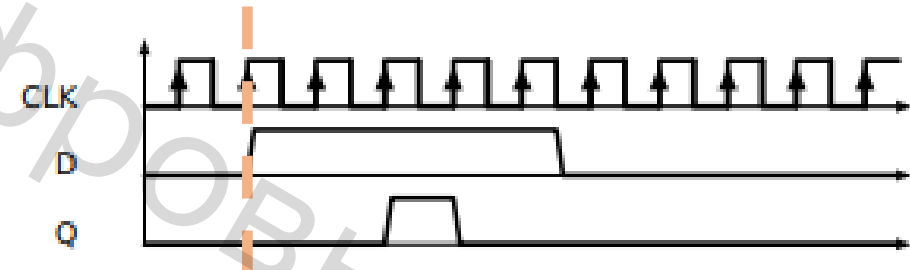
always_ff @ (posedge clk or posedge rst)
    if (rst)
        any_key_r <= '0;
    else
        any_key_r <= any_key;

wire any_key_pressed = ~ any_key & any_key_r;

```



Пример подобной схемы с тремя регистрами.
И временная диаграмма отражающая работу



Упражнение со счетчиком 1_06_binary_counter

```

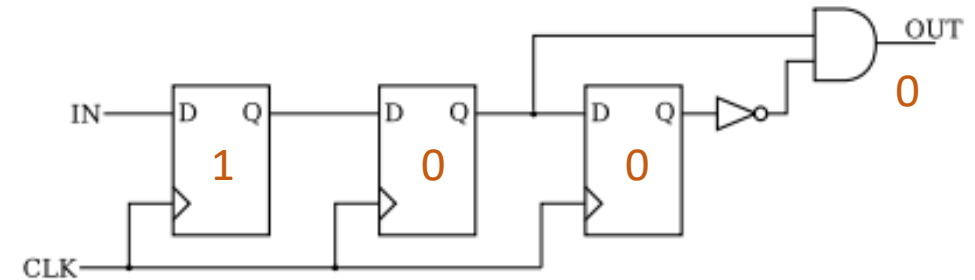
wire any_key = | key;

logic any_key_r;

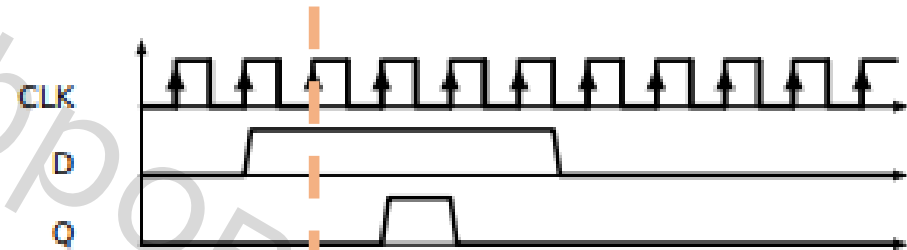
always_ff @ (posedge clk or posedge rst)
    if (rst)
        any_key_r <= '0;
    else
        any_key_r <= any_key;

wire any_key_pressed = ~ any_key & any_key_r;

```



Пример **подобной** схемы с тремя регистрами.
И временная диаграмма отражающая работу



Упражнение со счетчиком 1_06_binary_counter

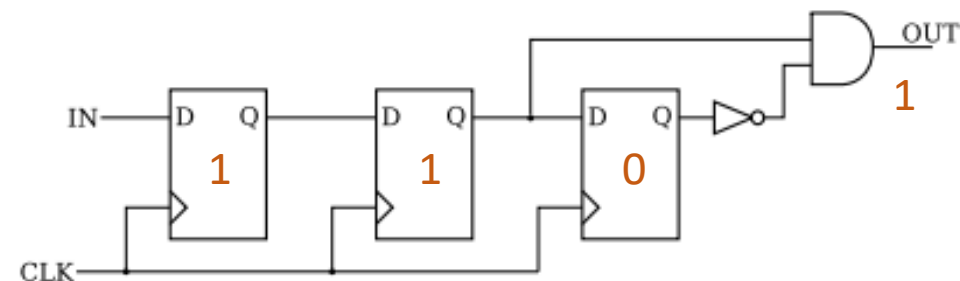
```

wire any_key = | key;

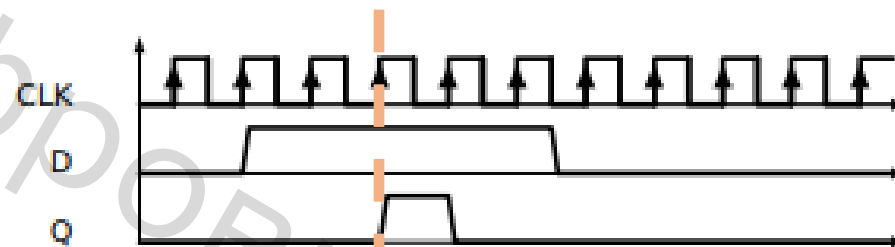
logic any_key_r;

always_ff @ (posedge clk or posedge rst)
    if (rst)
        any_key_r <= '0;
    else
        any_key_r <= any_key;

wire any_key_pressed = ~ any_key & any_key_r;
  
```



Пример **подобной** схемы с тремя регистрами.
И временная диаграмма отражающая работу



Превращение нажатия на любую кнопку
в однократный строб для однократной
обработки нажатия на кнопки

Упражнение со счетчиком 1_06_binary_counter

```
logic [w_led - 1:0] cnt;  
  
always_ff @ (posedge clk or posedge rst)  
    if (rst)  
        cnt <= '0;  
    else if (any_key_pressed)  
        cnt <= cnt + 1'd1;  
  
assign led = w_led' (cnt);
```

Счетчик инкрементируется на 1, если в момент положительного фронта `clk` `any_key_pressed` равен 1
В итоге каждое новое нажатие на кнопку увеличивает счетчик на 1

Упражнение со счетчиком 1_06_binary_counter

Упражнение 1: Свободно инкрементирующийся счетчик.

Измените скорость мигания светодиода. Попробуйте разные диапазоны бит для отображения.

Закомментируйте решение первого упражнения, перед решением второго

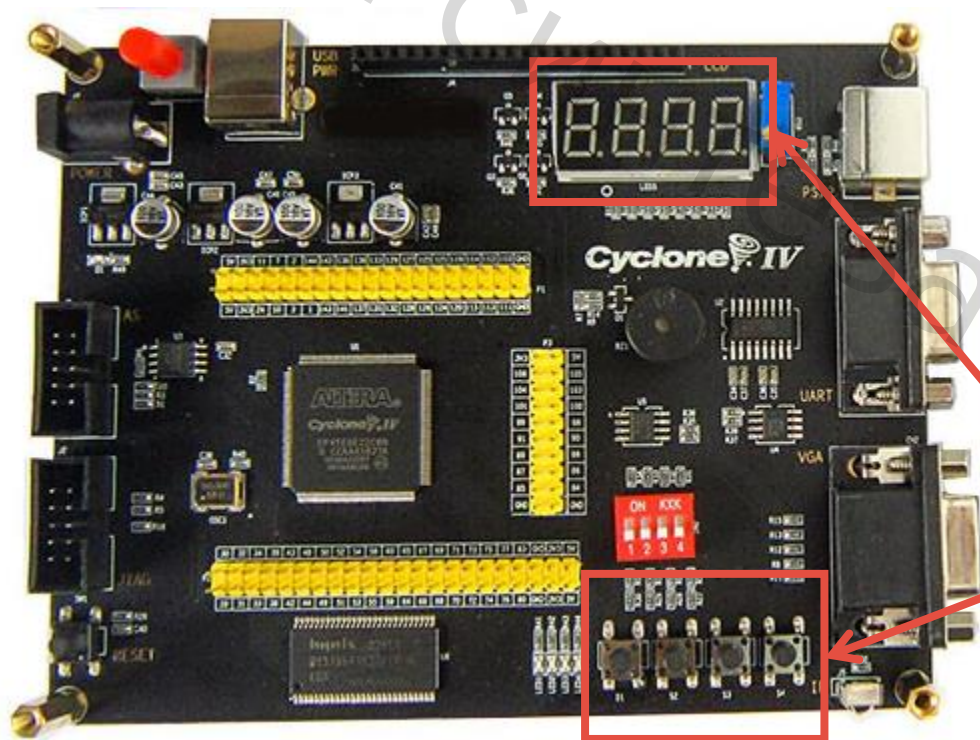
Упражнение 2: Счетчик, управляемый кнопкой.

Раскомментируйте и синтезируйте код. Нажмите кнопку, чтобы увидеть увеличение счетчика.

Измените дизайн:

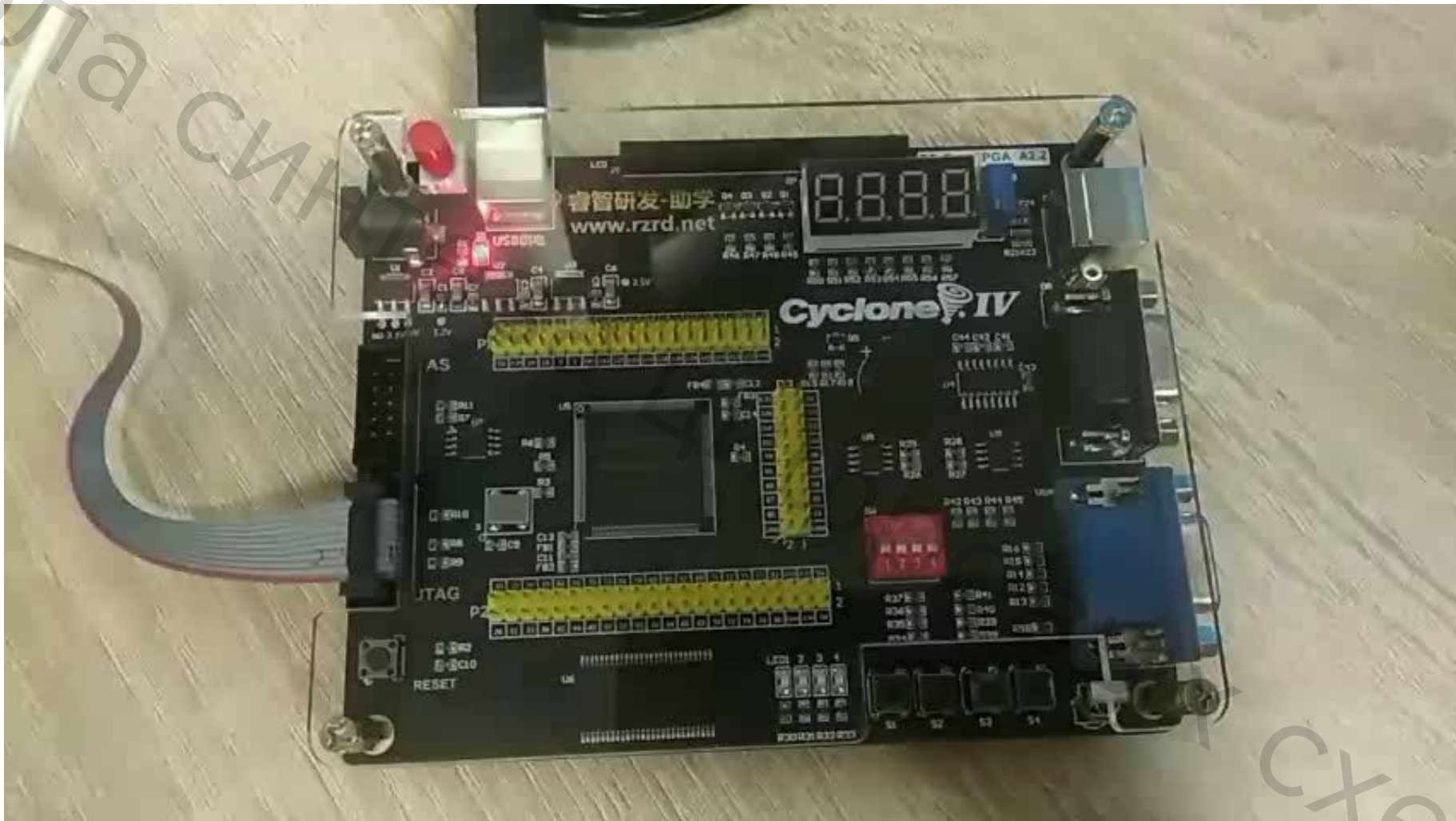
1. Одна кнопка используется для увеличения, другая для уменьшения значения счетчика.
2. Сделайте два счетчика, управляемые разными кнопками и отобразите их в разных группах светодиодов.

Упражнение со сдвиговым регистром 1_07_shift_register

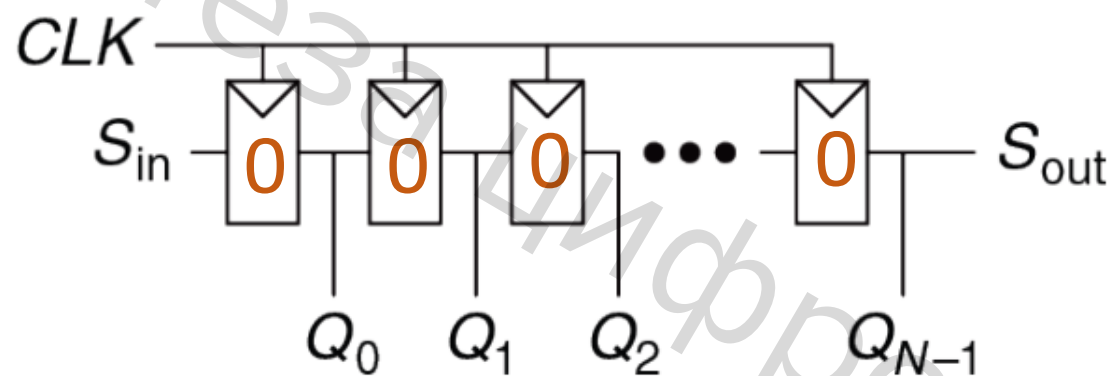


Управление мерцанием **светодиодов** и **семисегментного** индикатора при помощи последовательной логики и воздействий на кнопки управления на плате.

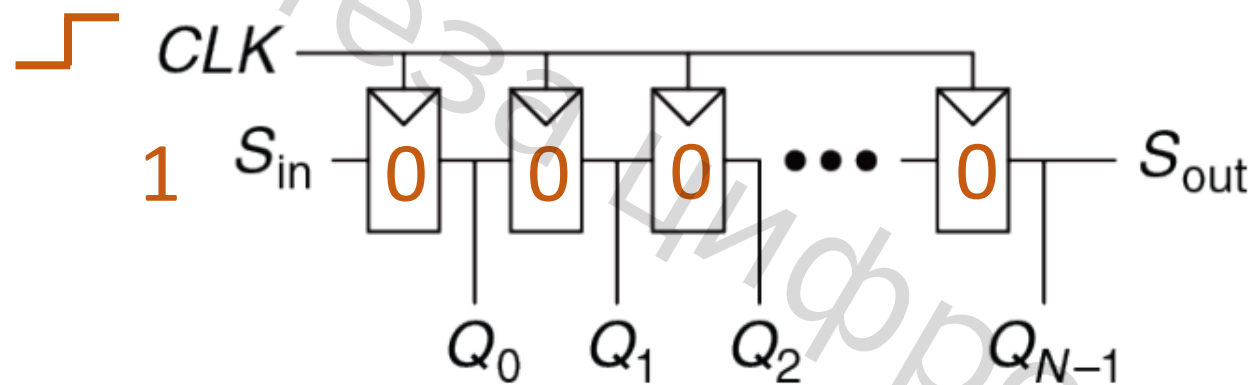
Упражнение со сдвиговым регистром 1_07_shift_register



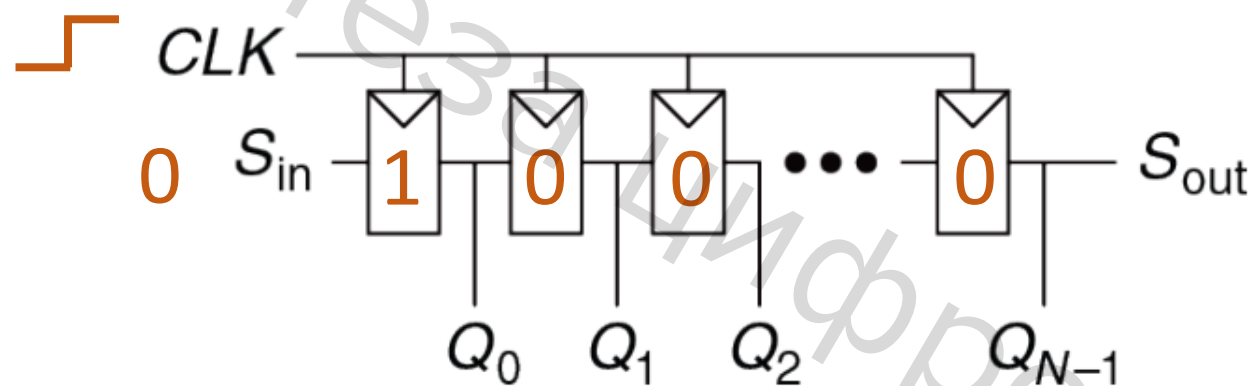
Сдвиговый регистр



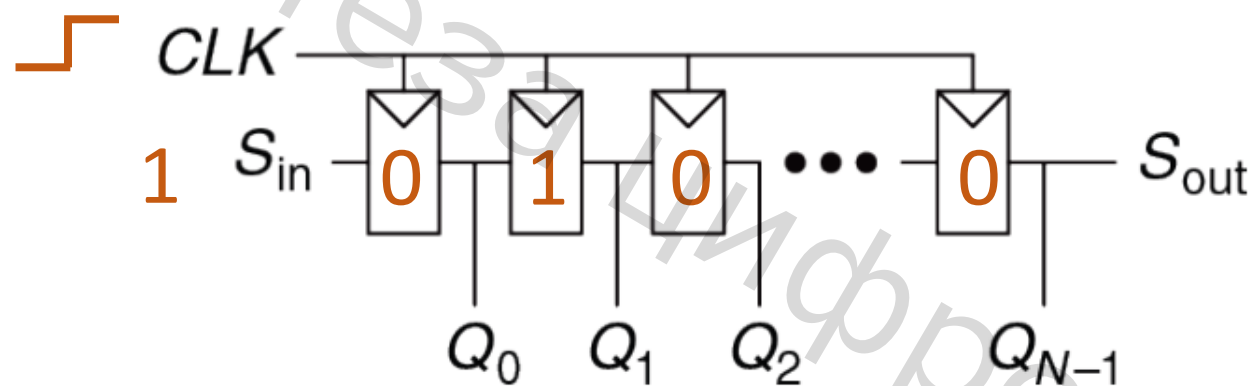
Сдвиговый регистр



Сдвиговый регистр



Сдвиговый регистр



Упражнение со сдвиговым регистром 1_07_shift_register

```
logic [31:0] cnt;  
  
always_ff @ (posedge clk or posedge rst)  
    if (rst)  
        cnt <= '0;  
    else  
        cnt <= cnt + 1'd1;  
  
wire enable = (cnt [22:0] == '0);
```

Генерация сигнала **enable** который равен 1 один раз в 2^{23} тактов. Изменив присвариваемый диапазон, можно изменить частоту генерации сигнала **enable**.
Данный сигнал необходим для удобной визуализации работы сдвигового регистра в примере

Упражнение со сдвиговым регистром 1_07_shift_register

```
wire button_on = | key;
logic [w_led - 1:0] shift_reg;

always_ff @ (posedge clk or posedge rst)
    if (rst)
        shift_reg <= '1;
    else if (enable)
        shift_reg <= { button_on, shift_reg [w_led - 1:1] };

assign led = shift_reg;
```

Нажатие на любую кнопку дает `button_on = 1`

Упражнение со сдвиговым регистром 1_07_shift_register

```
wire button_on = | key;

logic [w_led - 1:0] shift_reg;

always_ff @ (posedge clk or posedge rst)
    if (rst)
        shift_reg <= '1;
    else if (enable)
        shift_reg <= { button_on, shift_reg [w_led - 1:1] };

assign led = shift_reg;
```

Объявление сдвигового регистра разрядности той же, что и количество **led** на вашей плате. Если мы хотим, чтобы по сдвиговому регистру двигали не однобитные значения то его надо объявить двумерным массивом регистров

Упражнение со сдвиговым регистром 1_07_shift_register

```
wire button_on = | key;

logic [w_led - 1:0] shift_reg;

always_ff @ (posedge clk or posedge rst)
    if (rst)
        shift_reg <= '1;
    else if (enable)
        shift_reg <= { button_on, shift_reg [w_led - 1:1] };

assign led = shift_reg;
```

Реализация сдвига вправо на 1 разряд на такт с помощью операции конкатенации. Причем **сдвиг** будет осуществляться только, когда **enable = 1**

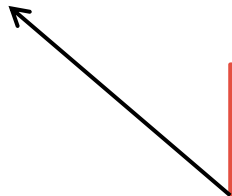
Упражнение со сдвиговым регистром 1_07_shift_register

```
wire button_on = | key;

logic [w_led - 1:0] shift_reg;

always_ff @ (posedge clk or posedge rst)
    if (rst)
        shift_reg <= '1;
    else if (enable)
        shift_reg <= { button_on, shift_reg [w_led - 1:1] };

assign led = shift_reg;
```



Вывод значения сдвигового регистра на светодиоды. В итоге на **led** при нажатии на кнопку будет видно движение огоньков по сдвиговому регистру


Упражнение со сдвиговым регистром 1_07_shift_register

```
wire button_on = | key;

logic [w_led - 1:0] shift_reg;

always_ff @ (posedge clk or posedge rst)
    if (rst)
        shift_reg <= '1;
    else if (enable)
        shift_reg <= { button_on, shift_reg [w_led - 1:1] };

assign led = shift_reg;
```

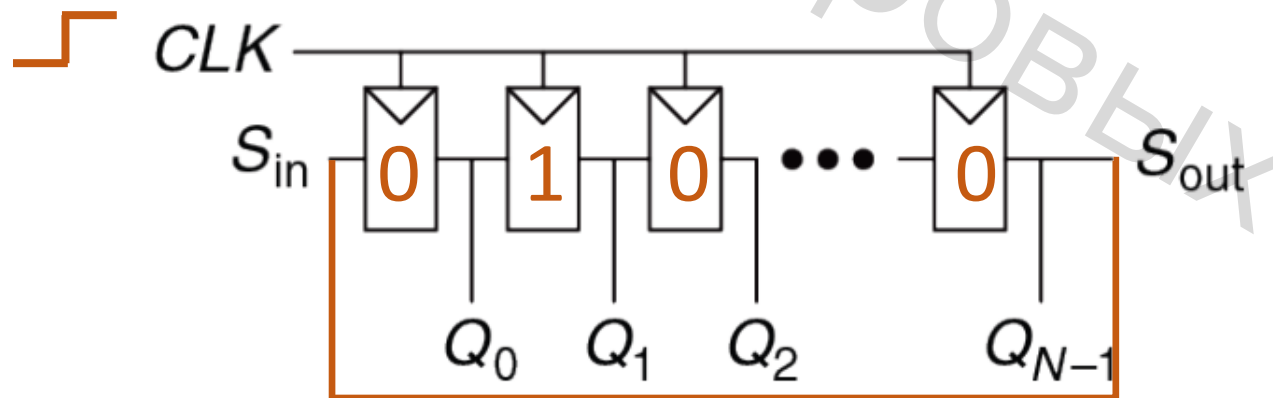


Для реализации циклического сдвига надо каждый такт
вдвигать не `button_on`, а выталкиваемый разряд `shift_reg`

Упражнение со сдвиговым регистром 1_07_shift_register

```
always_ff @ (posedge clk or posedge rst)
  if (rst)
    shift_reg <= '1;
  else if (enable)
    shift_reg <= { button_on, shift_reg [w_led - 1:1] };
```

Для реализации циклического сдвига надо каждый такт
вдвигать не `button_on`, а выталкиваемый разряд `shift_reg`



Упражнение 1_07_shift_register

Упражнение 1:

Заставьте огоньки двигаться в противоположном направлении на светодиодах

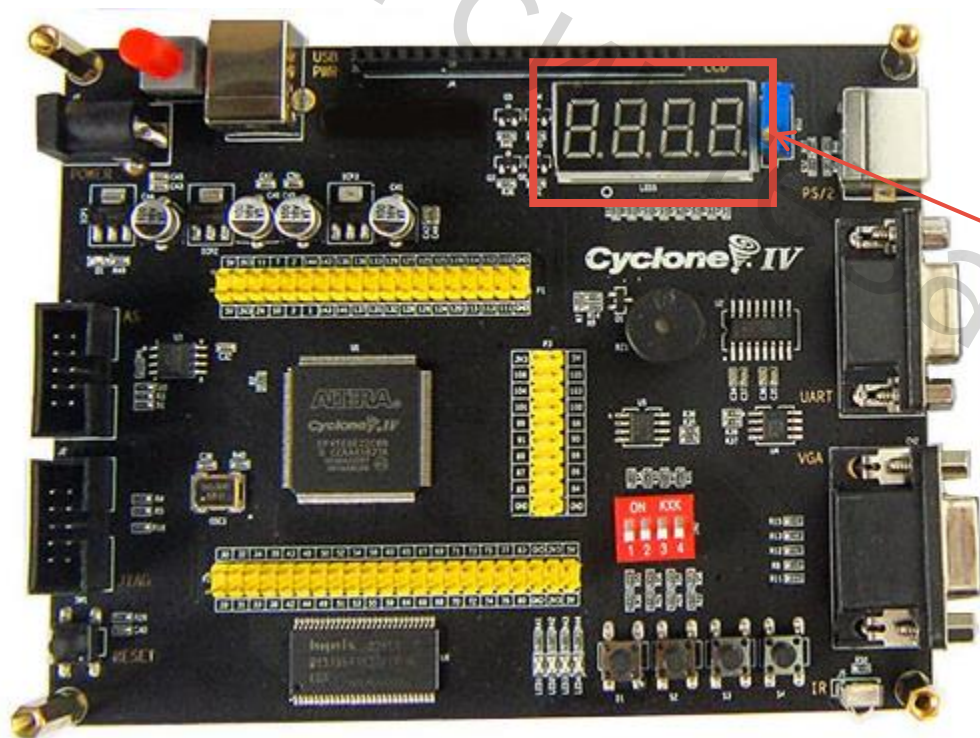
Упражнение 2:

Заставьте один огонек двигаться по кругу на светодиодах.
Используйте другую кнопку, чтобы сбросить движущийся огонек обратно в режим отсутствия

Упражнение 3:

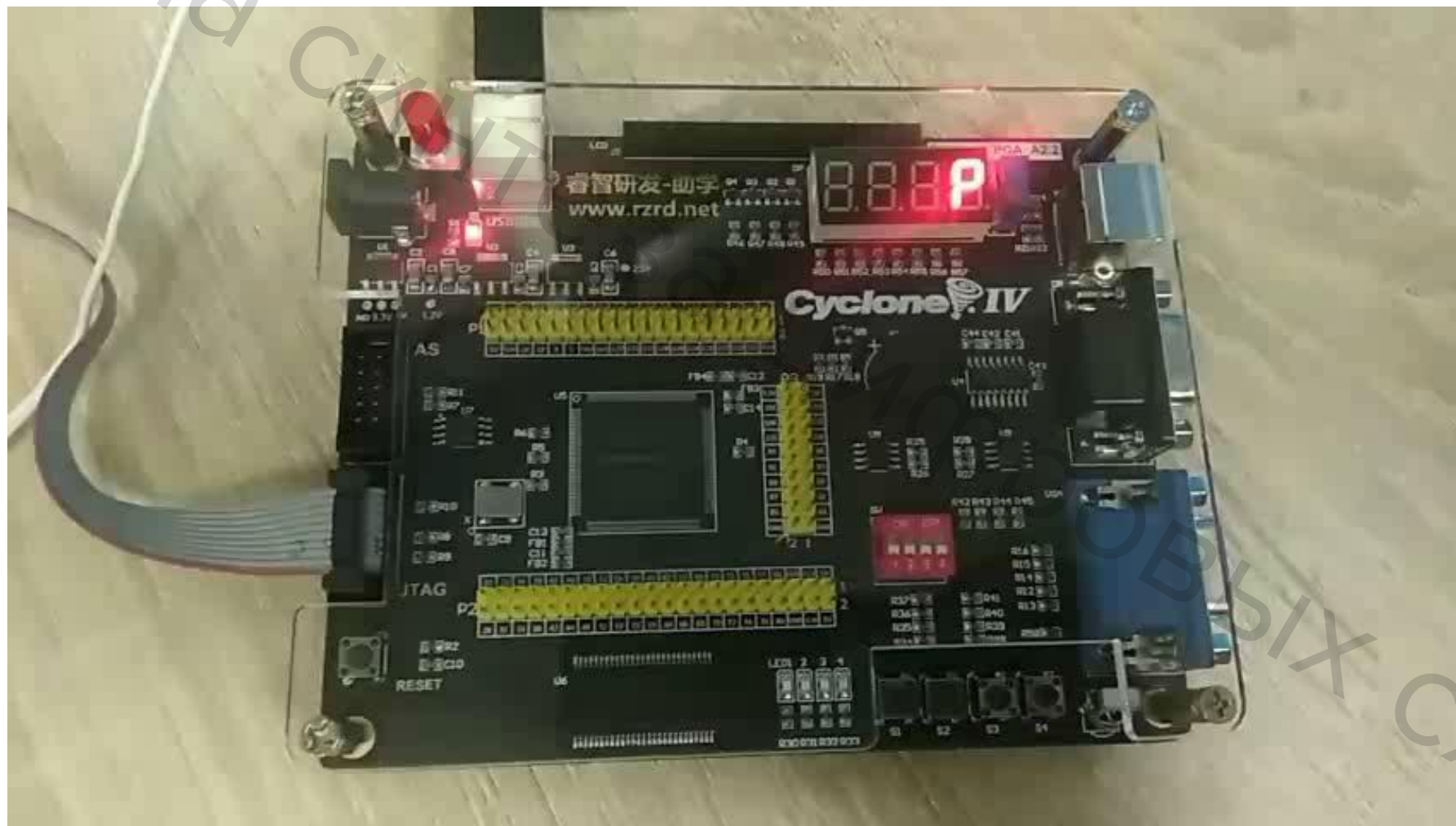
Отобразите состояние сдвигового регистра на семисегментном дисплее, перемещая огонек по нему

Упражнение: вывод слова на семисегментный индикатор 1_08_7segment_word

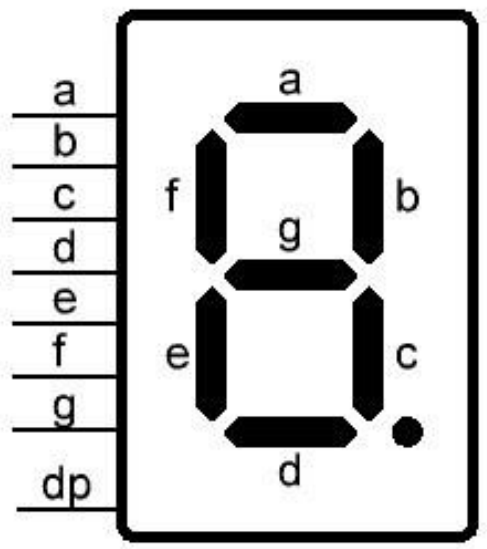


Вывод слова на **семисегментный** индикатор
при помощи последовательной логики.

Упражнение: вывод слова на семисегментный индикатор 1_08_7segment_word

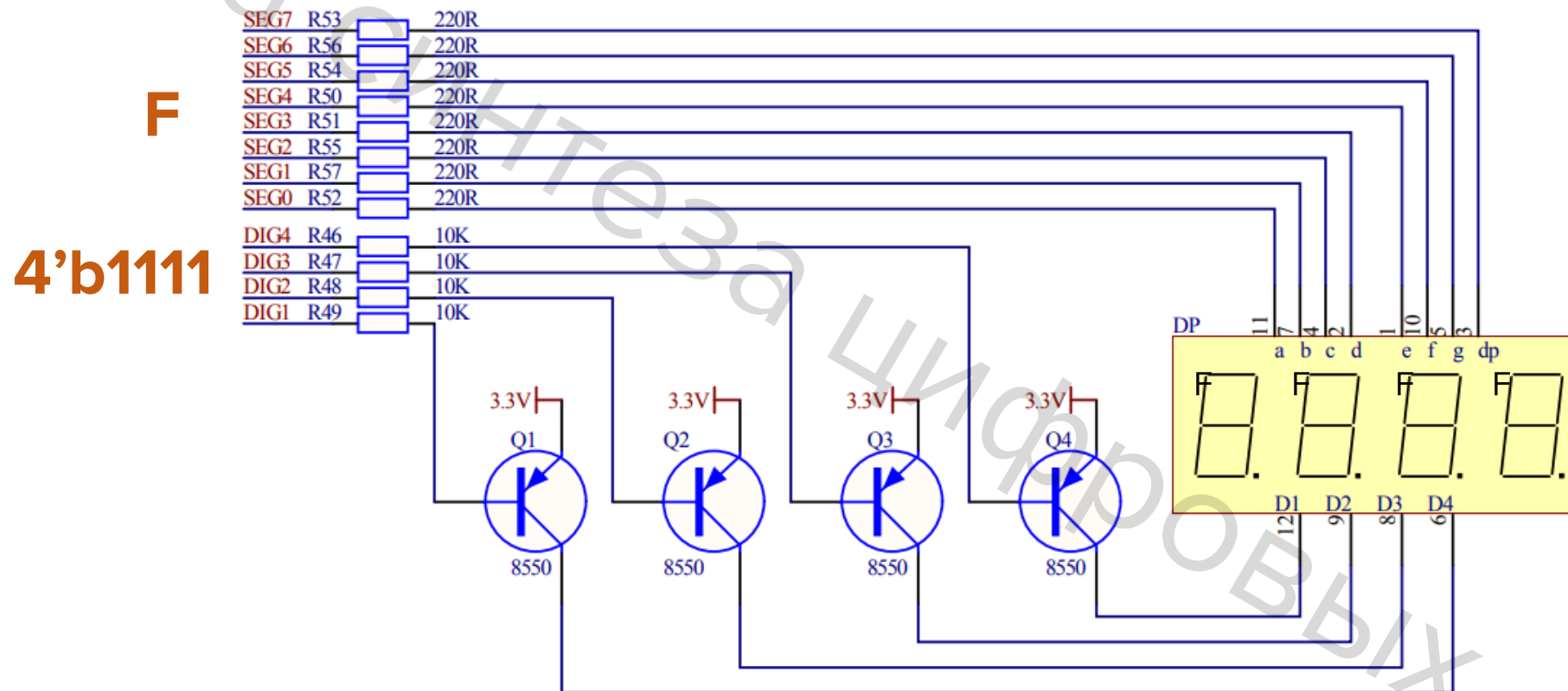


Семисегментный индикатор

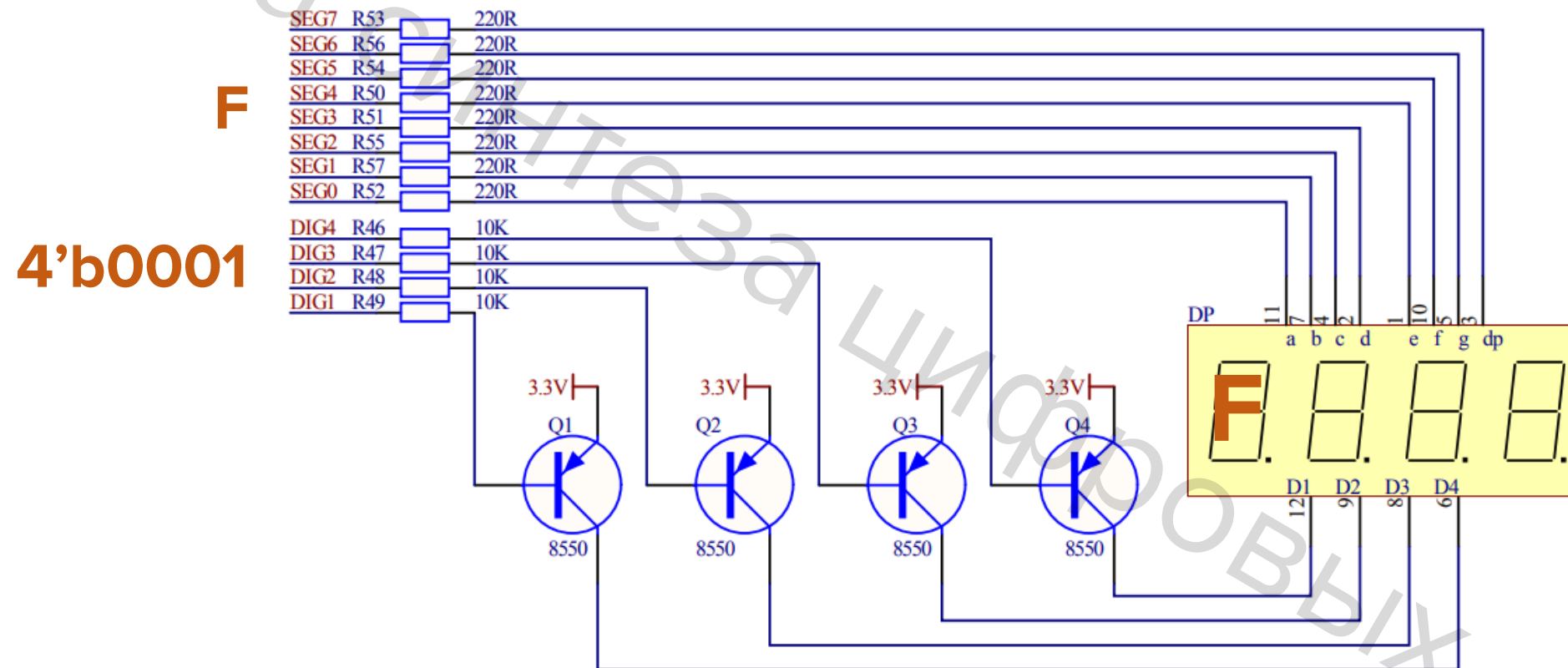


Dec	4-bit шина				7-сегментный индикатор						
	3	2	1	0	a	b	c	d	e	f	g
0	0	0	0	0	1	1	1	1	1	1	0
1	0	0	0	1	0	1	1	0	0	0	0
2	0	0	1	0	1	1	0	1	1	0	1
3	0	0	1	1	1	1	1	1	0	0	1
4	0	1	0	0	0	1	1	0	0	1	1
5	0	1	0	1	1	0	1	1	0	1	1
6	0	1	1	0	1	0	1	1	1	1	1
7	0	1	1	1	1	1	1	0	0	0	0
8	1	0	0	0	1	1	1	1	1	1	1
9	1	0	0	1	1	1	1	1	0	1	1

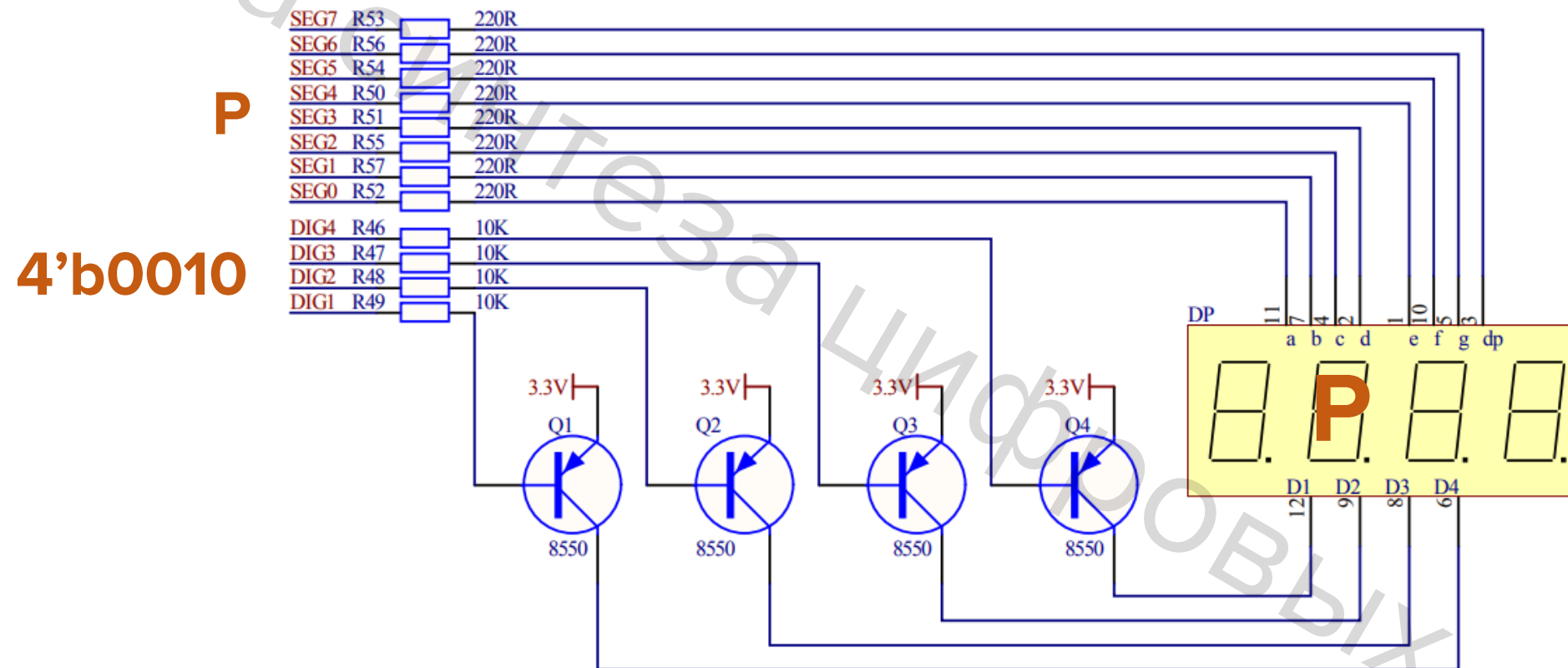
Семисегментный индикатор. Динамическая индикация



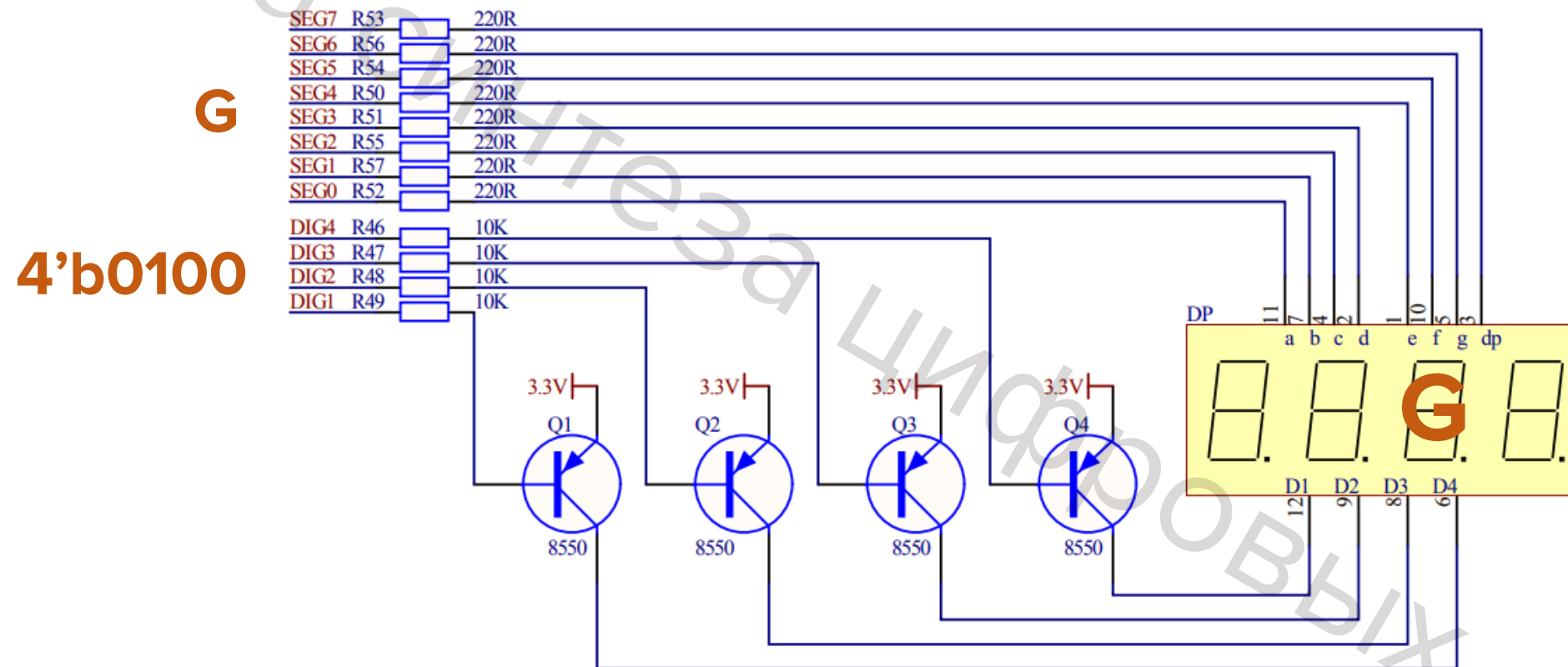
Семисегментный индикатор. Динамическая индикация



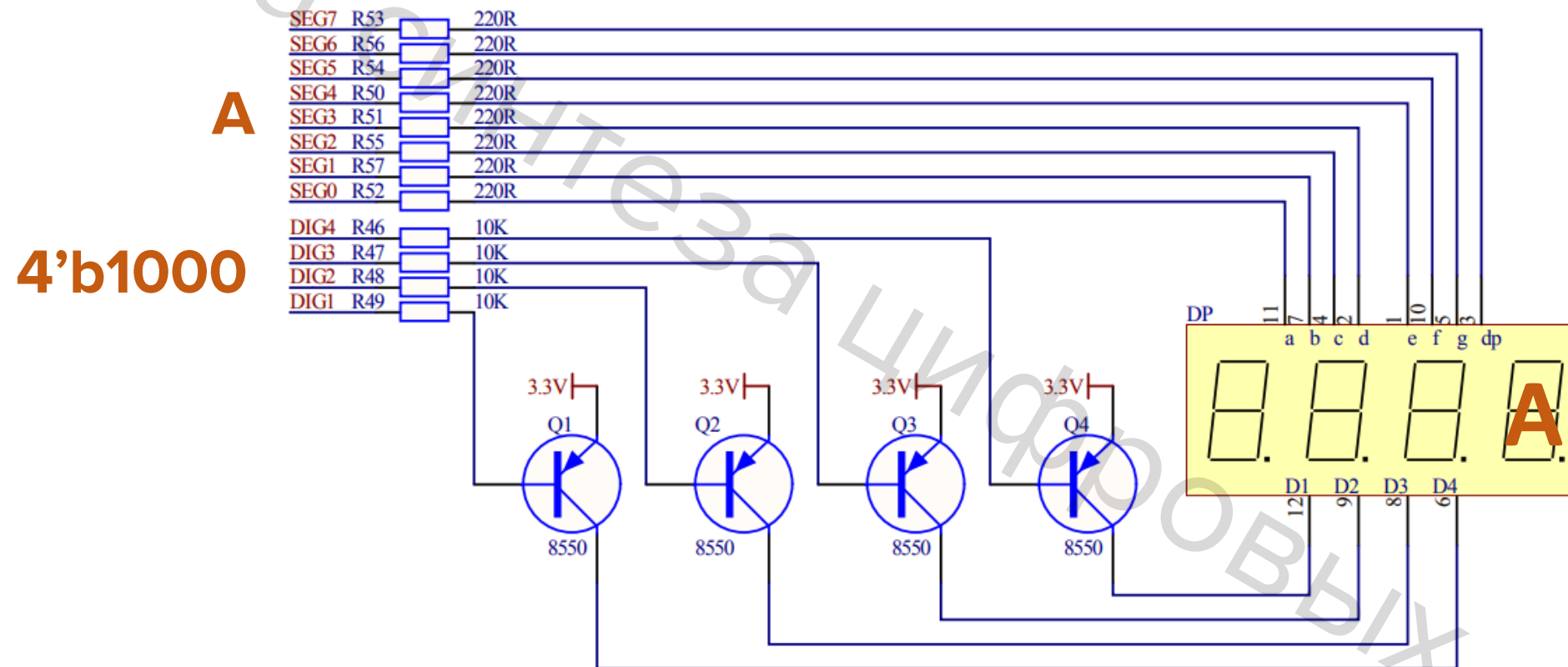
Семисегментный индикатор. Динамическая индикация



Семисегментный индикатор. Динамическая индикация



Семисегментный индикатор. Динамическая индикация



Упражнение: вывод слова на семисегментный индикатор

1_08_7segment_word

```
logic [31:0] cnt;  
  
always_ff @ (posedge clk or posedge rst)  
    if (rst)  
        cnt <= '0;  
    else  
        cnt <= cnt + 1'd1;  
  
wire enable = (cnt [22:0] == '0);
```

Генерация сигнала **enable** который равен 1 один раз в 2^{23} тактов. Изменив свариваемый диапазон, можно изменить частоту генерации сигнала **enable**. Данный сигнал необходим для удобной визуализации работы сдвигового регистра в примере

Упражнение: вывод слова на семисегментный индикатор

1_08_7segment_word

```
logic [w_digit:0] shift_reg;  
  
always_ff @ (posedge clk or posedge rst)  
    if (rst)  
        shift_reg <= w_digit' (1);  
    else if (enable)  
        shift_reg <= { shift_reg [0], shift_reg [w_digit - 1:1] };  
  
assign led = w_led' (shift_reg);
```

Циклический сдвиговый регистр, в котором по кругу бежит 1

0001 → 1000 → 0100 → 0010 → 0001

Упражнение: вывод слова на семисегментный индикатор

1_08_7segment_word

```
typedef enum bit [7:0]
{
    F      = 8'b1000_1110,
    P      = 8'b1100_1110,
    G      = 8'b1011_1100,
    A      = 8'b1110_1110,
    space  = 8'b0000_0000
}
seven_seg_encoding_e;
```

```
seven_seg_encoding_e letter;

always_comb
    case (4' (shift_reg))
        4'b1000: letter = F;
        4'b0100: letter = P;
        4'b0010: letter = G;
        4'b0001: letter = A;
        default: letter = space;
    endcase

assign abcdefgh = letter;
assign digit    = shift_reg;
```

Упражнение 1_08_7segment_word

Упражнение 1:

Увеличьте частоту сигнала **enable** до уровня, когда ваши глаза будут видеть буквы как сплошное слово без моргания. Каков порог такой частоты?

Упражнение 2:

Выведите свое имя или другое слово на дисплей.

Упражнение 3:

Закомментируйте строку **"default"** из оператора **"case"** в блоке **"always"** и заново синтезируйте пример. Получаете ли вы какие-либо предупреждения или ошибки? Попробуйте объяснить почему.

Упражнение 1_09_hex_counter

```
// Exercise 1. Synthesize the counter controlled by two keys.  
// When one key is in pressed position - the frequency increases,  
// when another key is in pressed position - the frequency decreases.  
// Change the period increment / decrement and see what happens.
```

```
logic [31:0] period;
```

```
localparam min_period = clk_mhz * 1000 * 1000 / 30,  
            max_period = clk_mhz * 1000 * 1000 * 3;
```


Упражнение 1_09 _hex_counter

```
always_ff @ (posedge clk or posedge rst)
    if (rst)
        period <= 32' ((min_period + max_period) / 2);
    else if (key [0] & period != max_period)
        period <= period + 32'h1;
    else if (key [1] & period != min_period)
        period <= period - 32'h1;
```

Регистр, хранящий в себе настраиваемое значение счета `cnt_1` (в некотором виде скорость счета). Обратите внимание что `key[0]` приоритетнее `key[1]`

А так же в условиях инкремента и декремента проверяется, что изменение счетчика не приведет к переполнению. Например, если `period = 0` и пришло нажатие на `key[1]` то останется `0`, а не будет значение `'b1111`

Упражнение 1_09_hex_counter

```
logic [31:0] cnt_1;  
  
always_ff @ (posedge clk or posedge rst)  
    if (rst)  
        cnt_1 <= '0;  
    else if (cnt_1 == '0)  
        cnt_1 <= period - 1'b1;  
    else  
        cnt_1 <= cnt_1 - 1'd1;
```

Счетчик, отсчитывающий время в тактах от 0 до `period - 1`. Изменяя значение `period` кнопками, можно изменить частоту, с которой `cnt_1` обнуляется

Упражнение 1_09_hex_counter

```
logic [31:0] cnt_2;  
  
always_ff @ (posedge clk or posedge rst)  
    if (rst)  
        cnt_2 <= '0;  
    else if (cnt_1 == '0)  
        cnt_2 <= cnt_2 + 1'd1;  
  
assign led = cnt_2;
```

Счетчик, инкрементирующийся каждый раз, когда `cnt_1` равен 0. Выведен на `led` в двоичном виде. Чем меньше `period`, тем чаще равен нулю `cnt_1`. Тем быстрее меняется значение `cnt_2`

Упражнение 1_09_hex_counter

```

seven_seg_encoding_e letter;

always_comb
    case (4' (shift_reg))
        4'b1000: letter = F;
        4'b0100: letter = P;
        4'b0010: letter = G;
        4'b0001: letter = A;
        default: letter = space;
    endcase

assign abcdefgh = letter;
assign digit     = shift_reg;

```



```

// 4 bits per hexadecimal digit
localparam w_display_number = w_digit * 4;

seven_segment_display # (w_digit) i_7segment
(
    .clk      ( clk          ),
    .rst      ( rst          ),
    .number   ( w_display_number' (cnt_2) ),
    .dots     ( w_digit' (0)   ),
    .abcdefgh ( abcdefgh     ),
    .digit    ( digit        )
);

```

Реализуем дешифратор для семисегментного индикатора в виде отдельного модуля

Упражнение 1_09_hex_counter

```
// 4 bits per hexadecimal digit
localparam w_display_number = w_digit * 4;

seven_segment_display # (w_digit) i_7segment
(
    .clk      ( clk                ),
    .rst      ( rst                ),
    .number   ( w_display_number' (cnt_2) ), ←
    .dots     ( w_digit' (0)       ),
    .abcdefg ( abcdefgh           ),
    .digit    ( digit             )
);
```

Счетчик `cnt_2` является 32-битным и будет выводиться на семисегментные индикаторы в виде 4 битный 16-ричных чисел. Каждый на отдельном индикаторе

Упражнение 1_09_hex_counter

Упражнение 1.

Синтезируйте счетчик, управляемый двумя кнопками.

Когда одна кнопка находится в нажатом положении - частота увеличивается, когда другая кнопка находится в нажатом положении - частота уменьшается.

Измените шаг увеличения / уменьшения периода и посмотрите, что произойдет.

Упражнение 2:

1. Удвоить частоту, когда одна кнопка нажимается и отжимается.
2. Уменьшите частоту вдвое, когда нажата и отжата вторая кнопка.

Покажите решение куратору кластера.

Автор (разработчик материала) лекции – Силантьев Александр Михайлович

Использование материалов и записи лекции и/или их частей без предварительного согласия не допускается.

По вопросам использования материалов и записи лекции в коммерческих целях необходимо направить обращение в ООО «КНС ГРУПП» (YADRO) по адресу электронной почты synthesis@yadro.com.

По вопросам некоммерческого использования материалов и записи лекции обращение может быть направлено в ООО «КНС ГРУПП» (YADRO) по адресу электронной почты synthesis@yadro.com, либо на адрес электронной почты автора Силантьева А.М. silantieva@org.miet.ru. Такое обращение обязательно должно содержать описание цели использования.