Лабораторная работа №2	22Б05	2024
Построение логических схем	Алькаев Равиль Юрисович	

Инструментарий и требования к работе: Работа выполняется в среде моделирования Logisim-evolution и Icarus Verilog 10 и новее (logisim-evolution v3.8.0 и iverilog (раздел "Инструменты для работы на Verilog")).

Описание работы

Работа состоит из трех частей: моделирования в Logisim, описания схемы и поведенческой модели на SystemVerilog. В заданиях задана одна и та же схема. Я все задания я выполнял с припиской normal.

Ссылка на репозиторий:

https://github.com/skkv-mkn/mkn-comp-arch-2023-circuit-alkaev

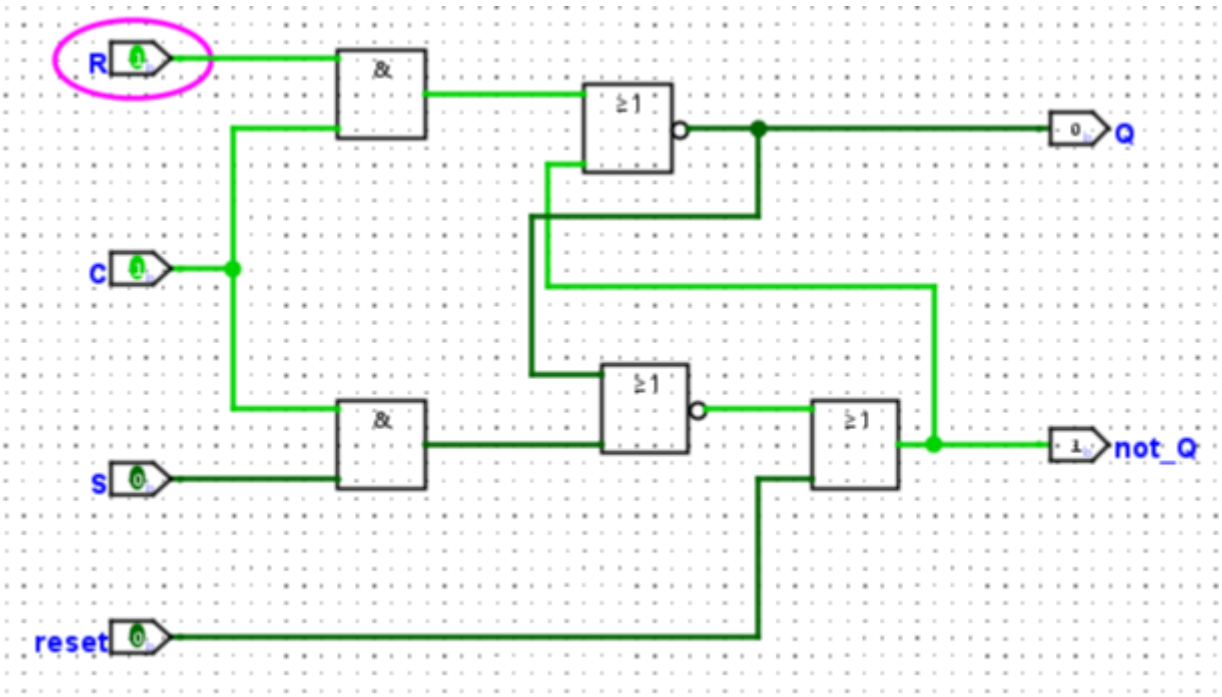
Logisim-evolution

Подсхема RS

Просто реализованный синхронный RS триггер с занулением (reset). Имеет 4 входа R, S, C, reset. Синхронизация по уровню. Когда C = 0 ничего не происходит. Когда C = 1 работает вот так:

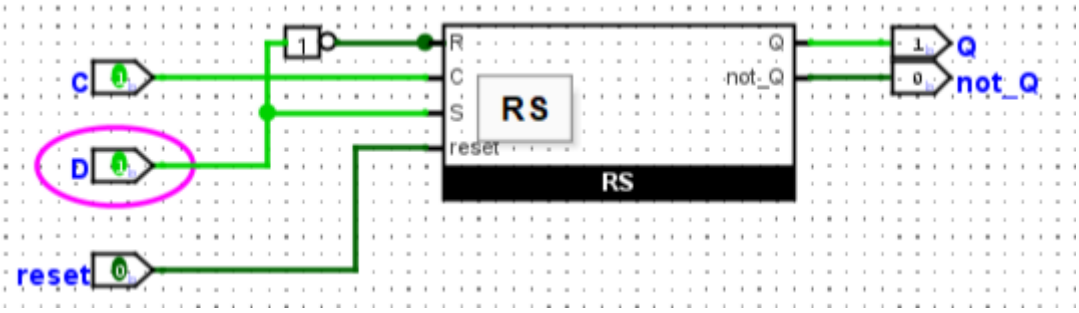
a	b	q
0	0	сохранение того значения, которое было в начале времён
0	1	0
1	0	1
1	1	не важно

.



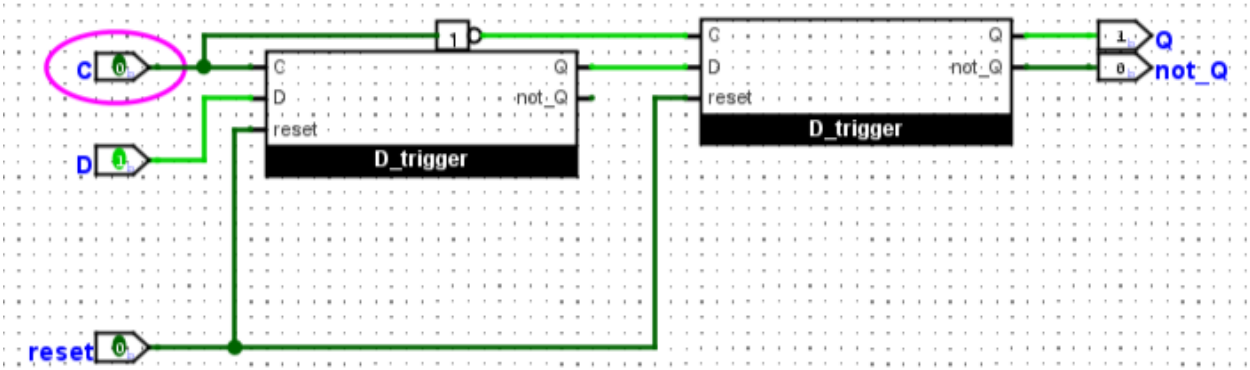
Подсхема D_trigger

Просто реализованный D-триггер. Он имеет 3 входа: C (синхронизация), D (записываемый бит) и reset(зануление). Когда C = 0, ничего не происходит, D-триггер просто продолжает хранить значение, которое в него когда-то записали. Когда C = 1, D-триггер сохраняет значение, которое ему подали на входе D



Подсхема D_trigger_front

Тот же самый D-триггер, но синхронизация работает по фронту. Пригодится для счетчика

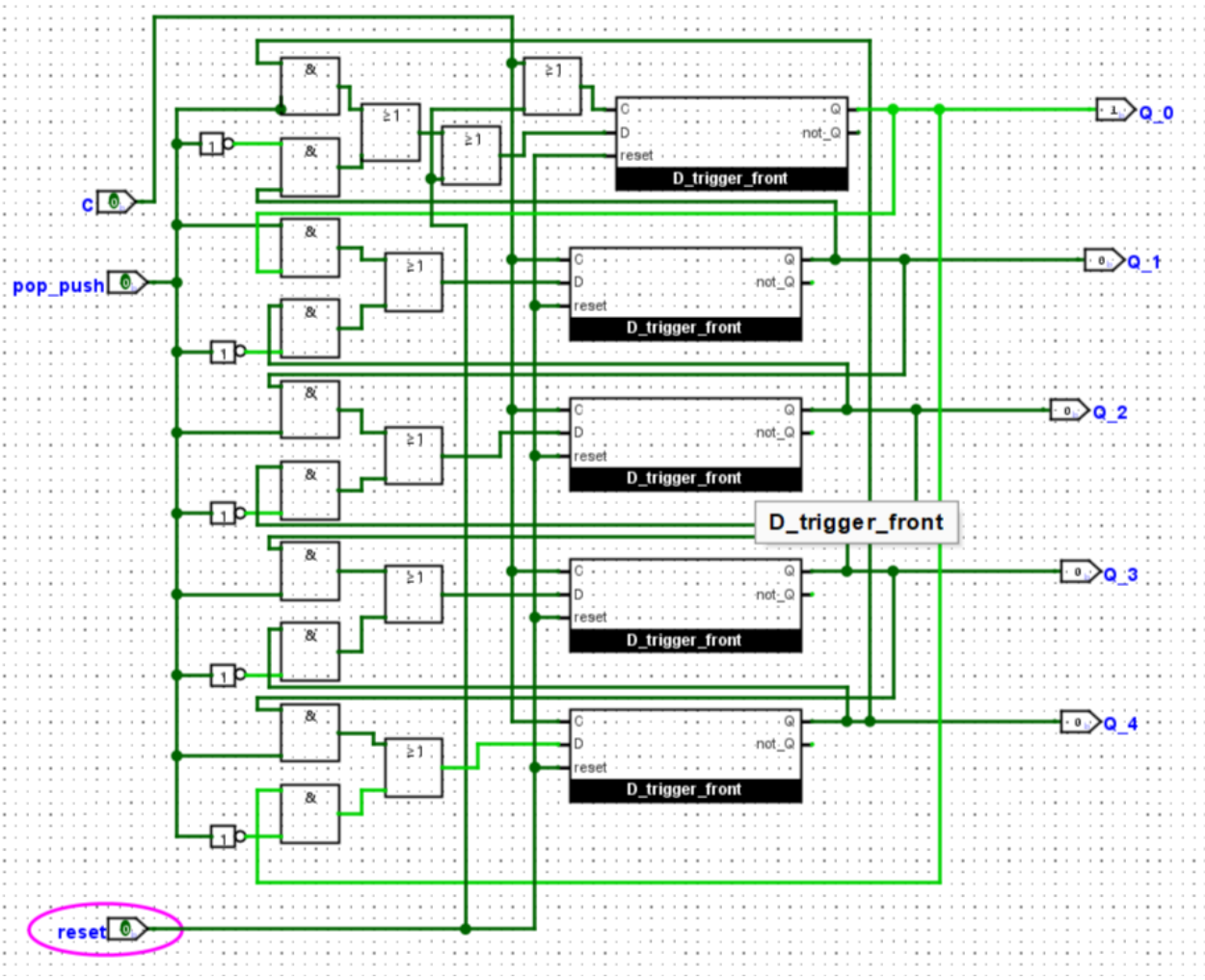


Подсхема counter_5

Эта схема дает нам понять на какой мы сейчас ячейке стека.

У нас два входа C(синхронизация) и pop_push. Синхронизация работает здесь по фронту.

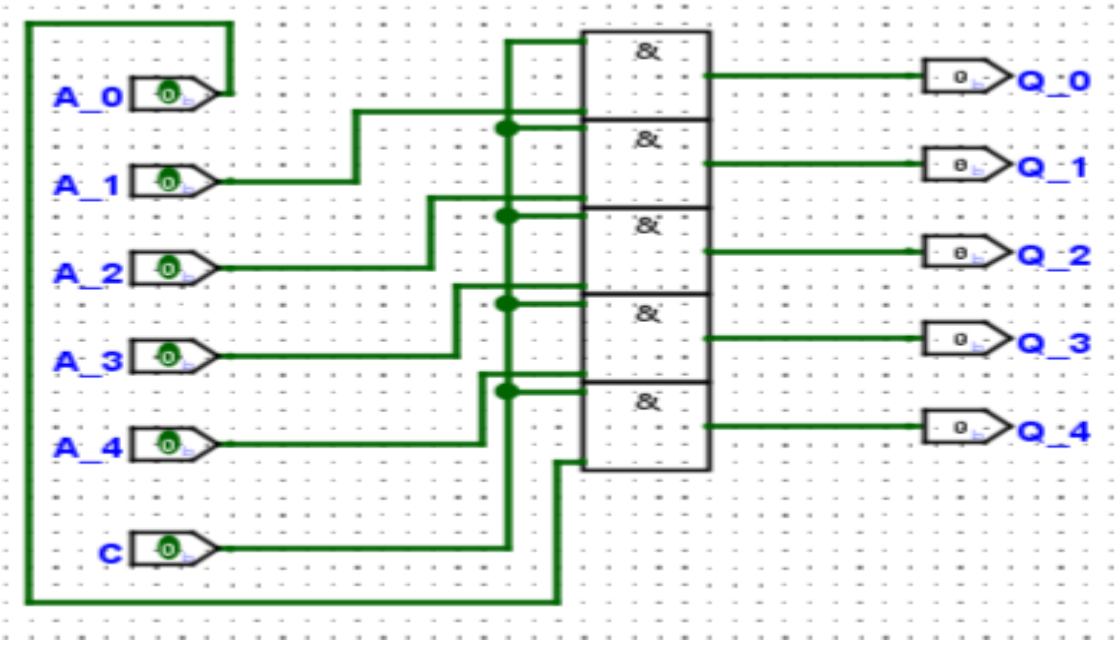
Если на входе pop_push подается 0, то это отвечает за pop и мы просто уменьшим счетчик на 1, если pop_push = 1 это отвечает за push и мы увеличим счетчик на 1. На выходе имеем нули во всех местах, кроме того куда указывает головка стека. В дальнейшем это будет выполнять роль синхронизаци



Подсхема Minus1_or_0

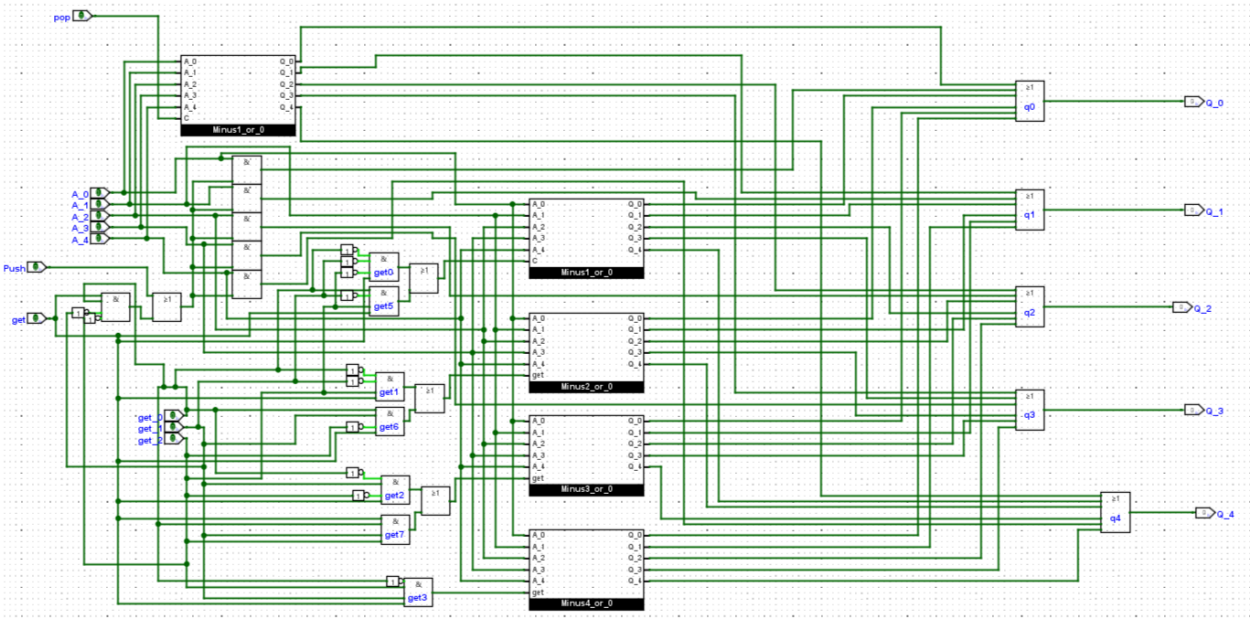
Имеется шесть входов. Все A_i кроме одного равны нулю. если C = 0, то на выходах получаем все нули, иначе смещаем все на 1.

То есть, если было C = 1, A_0 = 0, A_1 = 0, A_2 = 1 , A_3 = 0, A_4 = 0, то теперь все кроме Q_1 будут нулями. Аналогично работаю схемы Minus2_or_0, Minus3_or_0, Minus4_or_0, только смещение происходит не на 1, а на 2, 3 или 4 соответственно.



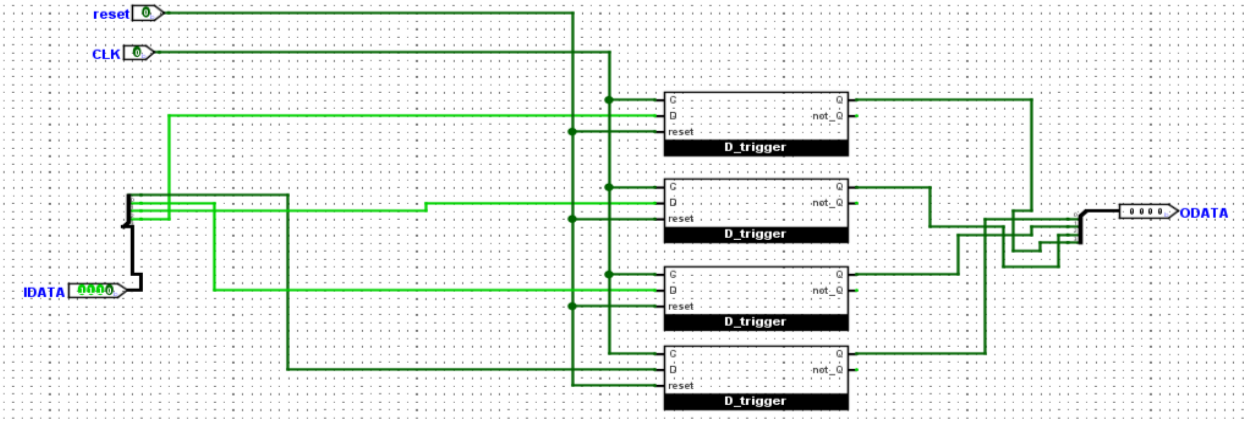
Подсхема OperationManager

Данная схема имеет 11 входов. A_0, A_1, A_2, A_3, A_4 передаются от счетчика, а pop, push, get в зависимости от команды которую мы должны выполнить, get_i индексы get. На выходе получаем номер ячейки стека с которой нам требуется провести операцию pop, push или get. Если на входе pop, get 0 или get 5, то пройдя через minus1_or_0 (можно было провести все три случая в один блок, но вначале у меня было немного по-другому а переделывать схему немного не приятно)) получим правильный индекс. Если у нас push или get 4, то не надо идти в подсхемы и сразу идем в суммы(через конъюнкции очевидно) получая на выходе правильный адрес.
При get 1 или get 6 -> Minus2_or_0
При get 2 или get 7 -> Minus3_or_0
При get 3 -> Minus4_or_0
(В одну подсхему идут одни остатки при деление на 5 индекса get) Теперь надо просуммировать соответствующие индексы и получим правильный адрес.



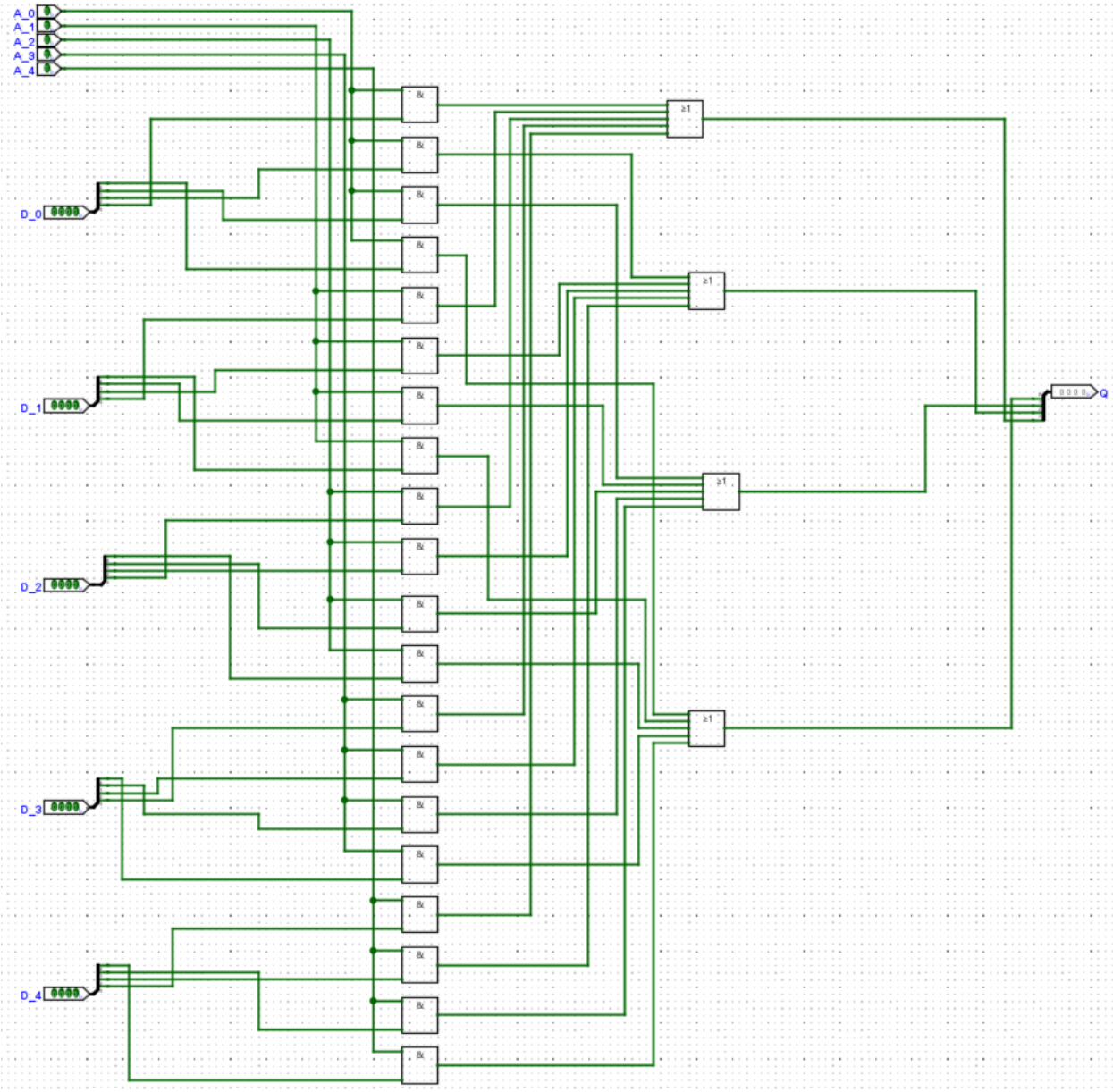
Подсхема cell_4

По сути это 4-битный D-trigger. Если C = 1 можем изменить значение, иначе ничего не произойдет



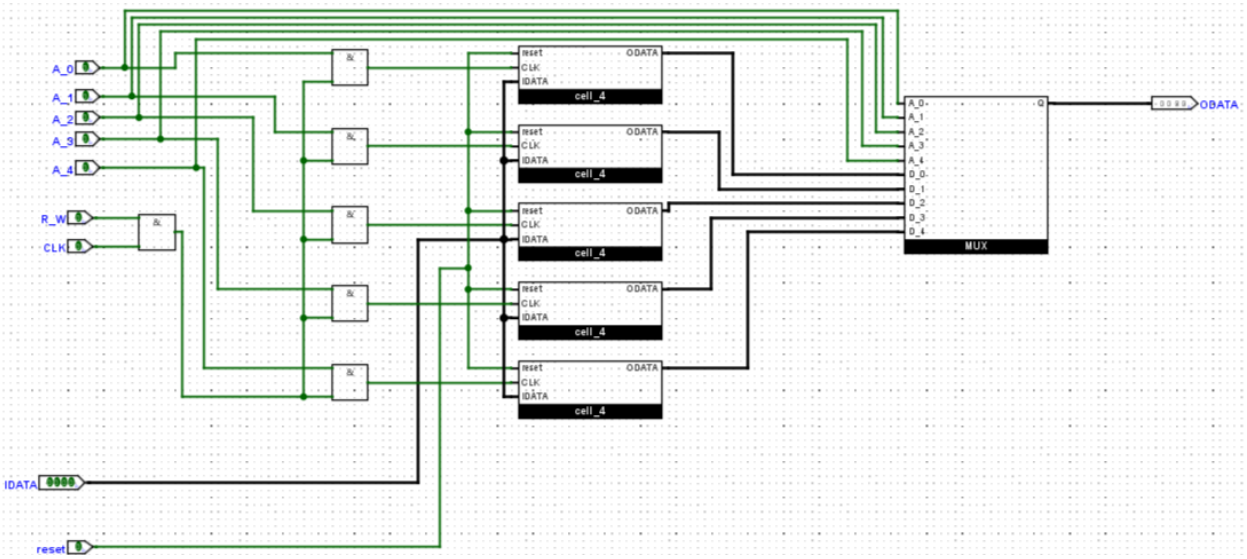
Подсхема MUX

Штука, которая читает биты из какой-то ячейки стека. На вход принимает следующие провода A_0, A_1, A_2, A_3, A_4 — они кодируют номер ячейки, из которой будем читать и D0, D1, D2 D3, D4 данные ячеек. С помощью легких конъюнкций будут выдаваться нужные данные.



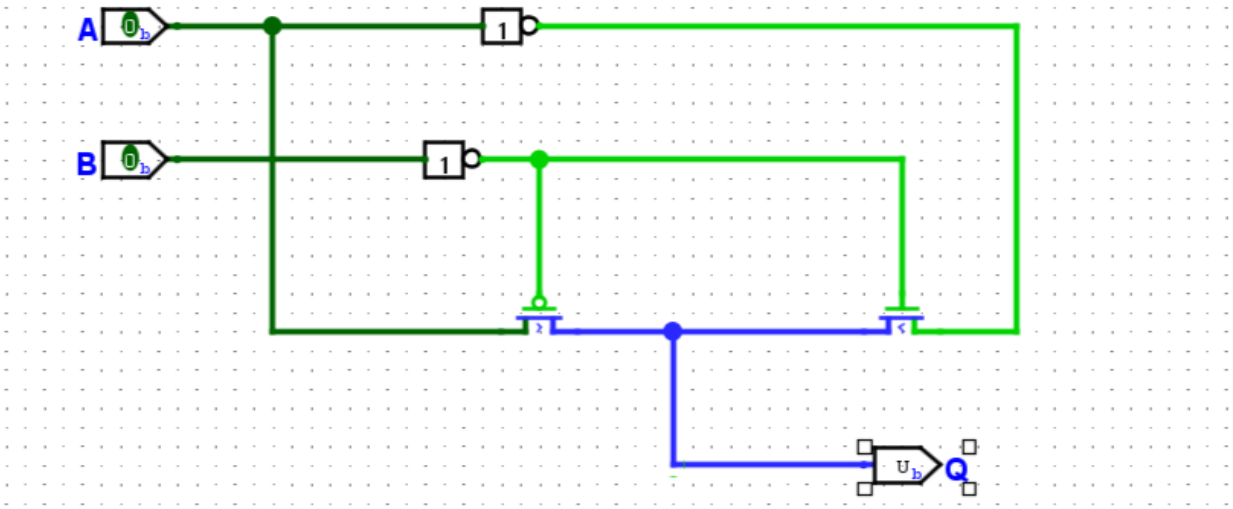
Подсхема MEM

Входы провода A_0, A_1, A_2, A_3, A_4 — код ячейки, с которой мы работаем, R_W — читаем мы (0) или пишем (1); C — синхронизация, IDATA - биты, которые мы можем захотеть записать в какую-нибудь ячейку, reset — если это равно 1, то вся память очищается (заполняется нулями). Выход ODATA содержимое ячейки, с которой мы работаем (если мы писали что-то в ячейку, то это ровно IDATA которые мы в неё записали). Используется 5 подсхем cell_4 (для пяти ячеек), если мы записываем, то есть если R_M = 1 и C = 1, значение 1 синхронизации попало только в нужную ячейку используются примитивные конъюнкции, MUX (для чтения).



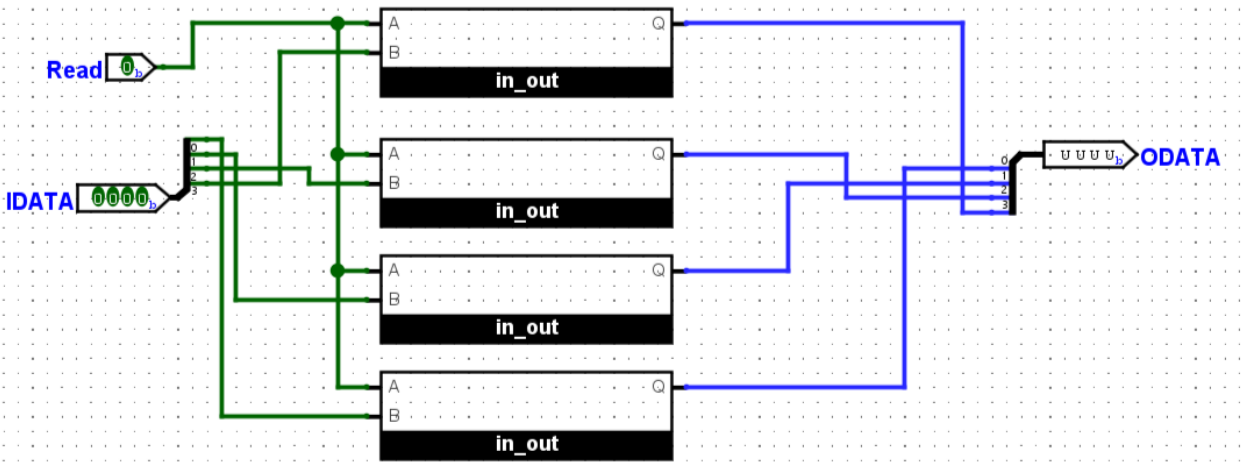
Подсхема in_out

Нам это понадобится для входо-выходов. Схема имеет два входа A и B и один выход Q. Когда A = 0, на выходе неизвестное (высокоимпедансное Z) значение. Когда A = 1, значение на выходе равно B. Состоит из двух отрицаний и транзисторов p-типа и n-типа.

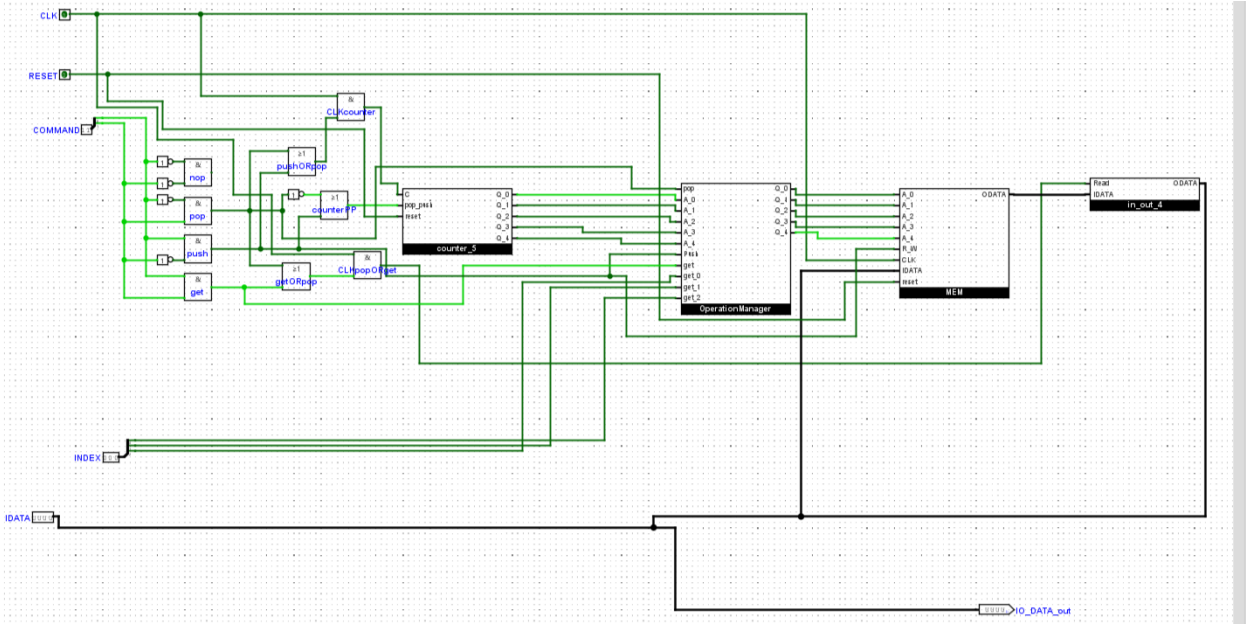


Подсхема in_out_4

Это просто объединение 4 схем in_out. Read мы посылаем в входы A, а биты из IDATA посылаем во входы под названием B.

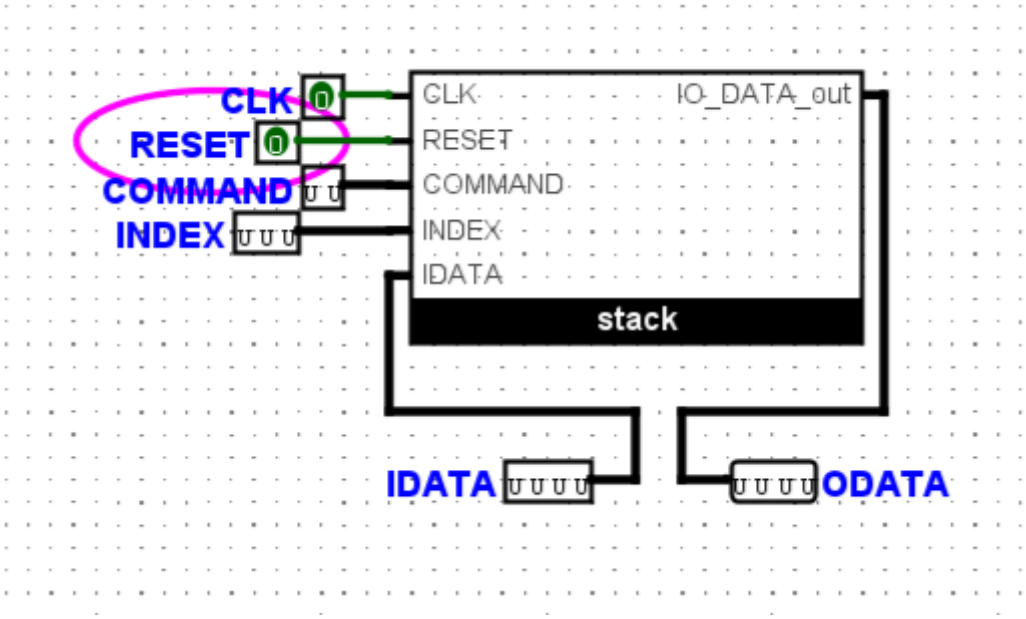


Подсхема stack



В этой части мы понимаем, какая команда должна выполняться, и после этого подаем нужные сигналы в counter_5, OperationManager и MEM. Например в counter_5 подаем в push_pop и синхронизацию как конъюнкцию CLK и дизъюнкции pop, push. В OperationManager мы подаем pop, push, get, текущий индекс головки стека и биты индекса get. В MEM подаем нужный нам индекс, который мы получили благодаря OperationManager, в R_W подаем push так, как запись только при пуше, иначе чтение. В IDATA подаем данные, reset в reset. Далее идем в in_out_4 в Read подаем CLKpopORget (так как нам нужно выводить только при pop или get).

Схема MAIN


CLK
RESET
COMMAND
INDEX
GLK
RESET
COMMAND
INDEX
IDATA
IO_DATA_out
stack
IDATA
ODATA

Структурная Модель на SystemVerilog

Чтобы написать структурную модель я просто описал все свои схемы и собрал их(поведение каждого модуля моделирует поведение соответствующей схемы). Тесты для каждого модуля X, называются X_tb и находятся в stack_structural_tb.sv

Модуль RS

```
module RS(output wire Q, input wire R, C, S, RESET);
   wire Rc, Sc, notQ, w1;
   and IN1(Rc, R, C);
   and IN2(Sc, S, C);
   nor NOR1(w1, Sc, Q);
   or OR1(notQ, w1, RESET);
   nor NOR2(Q, notQ, Rc);
endmodule
```

Модуль D_trigger

```
module D_trigger(output wire Q, input wire C, D, RESET);
   wire  notD;
   not ND(notD, D);
   RS RS1(.Q(Q), .R(notD), .C(C), .S(D), .RESET(RESET) );
endmodule
```

Модуль D_trigger_front

```
module D_trigger_front(output wire Q, input wire C, D, RESET);
   wire notC, w1;
   not NC(notC, C);
   D_trigger D_TR1(.Q(w1), .C(C), .D(D), .RESET(RESET));
   D_trigger D_TR2(.Q(Q), .C(notC), .D(w1), .RESET(RESET));
```

```
endmodule
```

Модуль counter_5

```
module counter_5(output wire Q0, Q1, Q2, Q3, Q4, input wire C, pop_push, RESET);

  wire w01, w02, w11, w12, w21, w22, w31, w32, w41, w42, W0, W1, W2, W3, W4, resandW0, resandC, notpop_push;

  not Npush_pop(notpop_push, pop_push);


  and W_01(w01, pop_push, Q4);

  and W_02(w02, notpop_push, Q1);

  or W_0(W0, w01, w02);

  or res_and_wo(resandW0, W0, RESET);

  or res_and_c(resandC, RESET, C);

  D_trigger_front D_TR0(.Q(Q0), .C(resandC), .D(resandW0), .RESET(RESET));


  and W_11(w11, pop_push, Q0);

  and W_12(w12, notpop_push, Q2);

  or W_1(W1, w11, w12);

  D_trigger_front D_TR1(.Q(Q1), .C(C), .D(W1), .RESET(RESET));


  and W_21(w21, pop_push, Q1);

  and W_22(w22, notpop_push, Q3);

  or W_2(W2, w21, w22);

  D_trigger_front D_TR2(.Q(Q2), .C(C), .D(W2), .RESET(RESET));


  and W_31(w31, pop_push, Q2);

  and W_32(w32, notpop_push, Q4);
```

```
   or W_3(W3, w31, w32);
   D_trigger_front D_TR3(.Q(Q3), .C(C), .D(W3), .RESET(RESET));

   and W_41(w41, pop_push, Q3);
   and W_42(w42, notpop_push, Q0);
   or W_4(W4, w41, w42);
   D_trigger_front D_TR4(.Q(Q4), .C(C), .D(W4), .RESET(RESET));
endmodule
```

Модуль Minus1_or_0

```
module Minus1_or_0(output wire Q0, Q1, Q2, Q3, Q4, input wire A0, A1, A2, A3, A4, C);
   and Q_0(Q0, C, A1 );
   and Q_1(Q1, C, A2 );
   and Q_2(Q2, C, A3 );
   and Q_3(Q3, C, A4 );
   and Q_4(Q4, C, A0 );
endmodule
```

Модуль Minus2_or_0

```
module Minus2_or_0(output wire Q0, Q1, Q2, Q3, Q4, input wire A0, A1, A2, A3, A4, C);
   and Q_0(Q0, C, A2 );
   and Q_1(Q1, C, A3 );
   and Q_2(Q2, C, A4 );
   and Q_3(Q3, C, A0 );
   and Q_4(Q4, C, A1 );
endmodule
```

Модуль Minus3_or_0

```
module Minus3_or_0(output wire Q0, Q1, Q2, Q3, Q4, input wire A0, A1, A2, A3, A4, C);
   and Q_0(Q0, C, A3 );
   and Q_1(Q1, C, A4 );
   and Q_2(Q2, C, A0 );
   and Q_3(Q3, C, A1 );
   and Q_4(Q4, C, A2 );
endmodule
```

Модуль Minus4_or_0

```
module Minus4_or_0(output wire Q0, Q1, Q2, Q3, Q4, input wire A0, A1, A2, A3, A4, C);
   and Q_0(Q0, C, A4 );
   and Q_1(Q1, C, A0 );
   and Q_2(Q2, C, A1 );
   and Q_3(Q3, C, A2 );
   and Q_4(Q4, C, A3 );
endmodule
```

Модуль OperationManager

```
module OperationManager(output wire Q0, Q1, Q2, Q3, Q4, input wire pop, push, get, get0, get1, get2,  A0, A1, A2, A3, A4 );
   wire w00, w01, w02, w03, w04, w10, w11, w12, w13, w14, w20, w21, w22, w23, w24, w30, w31, w32, w33, w34, w40, notget0, notget1, notget2,
   w41, w42, w43, w44, w50, w51, w52, w53, w54, pushOrget, C2, C3, C4, g0, g1, g2, g3, g4, g5, g6, g7;

   not Ng0(notget0, get0);
   not Ng1(notget1, get1);
   not Ng2(notget2, get2);
```

```
   and get_0(g0, notget0, notget1, notget2, get);
   and get_1(g1, notget0, notget1, get2, get);
   and get_2(g2, notget0, get1, notget2, get);
   and get_3(g3, notget0, get1, get2, get);
   and get_4(g4, get0, notget1, notget2, get);
   and get_5(g5, get0, notget1, get2, get);
   and get_6(g6, get0, get1, notget2, get);
   and get_7(g7, get0, get1, get2, get);

   or C_1(C1, g4, push);
   or C_2(C2, g0, g5);
   or c_3(C3, g1, g6);
   or C_4(C4, g2, g7);

   Minus1_or_0 M0(.A0(A0), .A1(A1), .A2(A2), .A3(A3), .A4(A4), .C(pop), .Q0(w00),
.Q1(w01), .Q2(w02), .Q3(w03), .Q4(w04));
   and AND0(w10, C1, A0);
   and AND1(w11, C1, A1);
   and AND2(w12, C1, A2);
   and AND3(w13, C1, A3);
   and AND4(w14, C1, A4);

   // нижние модули

   Minus1_or_0 M1(.A0(A0), .A1(A1), .A2(A2), .A3(A3), .A4(A4), .C(C2), .Q0(w20),
.Q1(w21), .Q2(w22), .Q3(w23), .Q4(w24));
   Minus2_or_0 M2(.A0(A0), .A1(A1), .A2(A2), .A3(A3), .A4(A4), .C(C3), .Q0(w30),
.Q1(w31), .Q2(w32), .Q3(w33), .Q4(w34));
```

```
   Minus3_or_0 M3(.A0(A0), .A1(A1), .A2(A2), .A3(A3), .A4(A4), .C(C4), .Q0(w40), .Q1(w41), .Q2(w42), .Q3(w43), .Q4(w44));

   Minus4_or_0 M4(.A0(A0), .A1(A1), .A2(A2), .A3(A3), .A4(A4), .C(g3), .Q0(w50), .Q1(w51), .Q2(w52), .Q3(w53), .Q4(w54));


   // выходы


   or Q_0(Q0, w00, w10, w20, w30, w40, w50);

   or Q_1(Q1, w01, w11, w21, w31, w41, w51);

   or Q_2(Q2, w02, w12, w22, w32, w42, w52);

   or Q_3(Q3, w03, w13, w23, w33, w43, w53);

   or Q_4(Q4, w04, w14, w24, w34, w44, w54);


endmodule
```

Модуль cell_4

```
module cell_4(output wire[3:0] ODATA, input wire RESET, CLK, input wire[3:0] IDATA);


   D_trigger D_TR0(.Q(ODATA[0]), .C(CLK), .RESET(RESET), .D(IDATA[0]));

   D_trigger D_TR1(.Q(ODATA[1]), .C(CLK), .RESET(RESET), .D(IDATA[1]));

   D_trigger D_TR2(.Q(ODATA[2]), .C(CLK), .RESET(RESET), .D(IDATA[2]));

   D_trigger D_TR3(.Q(ODATA[3]), .C(CLK), .RESET(RESET), .D(IDATA[3]));


endmodule
```

Модуль MUX

```
module MUX(output wire[3:0] DATA, input wire A0, A1, A2, A3, A4, input wire[3:0] D0, D1, D2, D3, D4);

wire w00, w01, w02, w03, w10, w11, w12, w13, w20, w21, w22, w23, w30, w31, w32, w33, w40, w41, w42, w43;
```

```
and AND00(w00, A0, D0[0]);
and AND01(w01, A0, D0[1]);
and AND02(w02, A0, D0[2]);
and AND03(w03, A0, D0[3]);

and AND10(w10, A1, D1[0]);
and AND11(w11, A1, D1[1]);
and AND12(w12, A1, D1[2]);
and AND13(w13, A1, D1[3]);

and AND20(w20, A2, D2[0]);
and AND21(w21, A2, D2[1]);
and AND22(w22, A2, D2[2]);
and AND23(w23, A2, D2[3]);

and AND30(w30, A3, D3[0]);
and AND31(w31, A3, D3[1]);
and AND32(w32, A3, D3[2]);
and AND33(w33, A3, D3[3]);

and AND40(w40, A4, D4[0]);
and AND41(w41, A4, D4[1]);
and AND42(w42, A4, D4[2]);
and AND43(w43, A4, D4[3]);

or OR0(DATA[0], w00, w10, w20, w30, w40);
```

```
  or OR1(DATA[1], w01, w11, w21, w31, w41);
  or OR2(DATA[2], w02, w12, w22, w32, w42);
  or OR3(DATA[3], w03, w13, w23, w33, w43);

endmodule
```

Модуль MEM

```
module MEM(output wire[3:0] ODATA, input wire A0, A1, A2, A3, A4, R_W, CLK, RESET, input wire[3:0] IDATA);
  wire RwAndCLK, CLK0, CLK1, CLK2, CLK3, CLK4;
  wire[3:0] D0, D1, D2, D3, D4;

  and read_and_clk(RwAndCLK, CLK, R_W);
  and AND0(CLK0, RwAndCLK, A0);
  and AND1(CLK1, RwAndCLK, A1);
  and AND2(CLK2, RwAndCLK, A2);
  and AND3(CLK3, RwAndCLK, A3);
  and AND4(CLK4, RwAndCLK, A4);

  cell_4 Cell0(.ODATA(D0), .RESET(RESET), .CLK(CLK0), .IDATA(IDATA));
  cell_4 Cell1(.ODATA(D1), .RESET(RESET), .CLK(CLK1), .IDATA(IDATA));
  cell_4 Cell2(.ODATA(D2), .RESET(RESET), .CLK(CLK2), .IDATA(IDATA));
  cell_4 Cell3(.ODATA(D3), .RESET(RESET), .CLK(CLK3), .IDATA(IDATA));
  cell_4 Cell4(.ODATA(D4), .RESET(RESET), .CLK(CLK4), .IDATA(IDATA));

  MUX MUX1(.DATA(ODATA), .A0(A0), .A1(A1), .A2(A2), .A3(A3), .A4(A4), .D0(D0), .D1(D1), .D2(D2), .D3(D3), .D4(D4));
endmodule
```

Модуль in_out

```
module in_out(output wire Q, input wire A, B);
   wire notA;
   not not_A(notA, A);
   nmos p_tr(Q, (B), A);
   pmos n_tr(Q, (B), notA);
endmodule
```

Модуль in_out_4

```
module in_out_4(output wire[3:0] ODATA, input wire READ, input wire[3:0] IDATA);

   in_out INOUT0(.Q(ODATA[0]), .A(READ), .B(IDATA[0]) );
   in_out INOUT1(.Q(ODATA[1]), .A(READ), .B(IDATA[1]) );
   in_out INOUT2(.Q(ODATA[2]), .A(READ), .B(IDATA[2]) );
   in_out INOUT3(.Q(ODATA[3]), .A(READ), .B(IDATA[3]) );

endmodule
```

Модуль stack

```
module stack(output wire[3:0] IODATA_out, input wire RESET, CLK, input wire[1:0] COMMAND, input wire[2:0] INDEX, input wire[3:0] IDATA);
   wire notOP1, notOP2, nop, push, pop, get, getORpop, pushORpop, CLKcounter, CLKpopORget, notpop, counterPP,
   Q0, Q1, Q2, Q3, Q4, W0, W1, W2, W3, W4;
   wire[3:0] OUTMEM;
   not not_OP1(notOP1, COMMAND[0]);
   not not_OP2(notOP2, COMMAND[1]);
   and NOP(nop, notOP1, notOP2);
   and PUSH(push, notOP1, COMMAND[1]);
   and POP(pop, COMMAND[0], notOP2);
```

```
   not NOT_POP(notpop, pop);
   and GET(get, COMMAND[0], COMMAND[1]);
   or POP_OR_GET(getORpop, pop, get);
   or PUSH_OR_POP(pushORpop, push, pop);
   and CLK_COUNTER(CLKcounter, pushORpop, CLK);
   and CLK_POP_OR_GET(CLKpopORget, getORpop, CLK);
   or COUNTER_PP(counterPP, notpop, push);


   counter_5 C5(.C(CLKcounter), .pop_push(counterPP), .RESET(RESET), .Q0(Q0),
.Q1(Q1), .Q2(Q2), .Q3(Q3), .Q4(Q4));
   OperationManager M5(.pop(pop), .A0(Q0), .A1(Q1), .A2(Q2), .A3(Q3), .A4(Q4),
   .push(push), .get(get), .get0(INDEX[0]), .get1(INDEX[1]), .get2(INDEX[2]),
.Q0(W0), .Q1(W1), .Q2(W2), .Q3(W3), .Q4(W4) );
   MEM MEM0(.A0(W0), .A1(W1), .A2(W2), .A3(W3), .A4(W4), .R_W(push),
.CLK(CLK), .RESET(RESET), .IDATA(IDATA), .ODATA(OUTMEM));
   in_out_4 INOUT40(.READ(CLKpopORget), .IDATA(OUTMEM),
.ODATA(IODATA_out));
endmodule
```

Модуль stack_structural_normal (Main)

```
module stack_structural_normal(
   input wire RESET,
   input wire CLK,
   input wire[1:0] COMMAND,
   input wire[2:0] INDEX,
   inout wire[3:0] IO_DATA
   );
   stack ST0(.IODATA_out(IO_DATA), .RESET(RESET), .CLK(CLK),
.COMMAND(COMMAND), .INDEX(INDEX), .IDATA(IO_DATA));
endmodule
```

Поведенческая модель на Verilog

```
module stack(output reg[3:0] IODATA_out, input wire RESET, CLK, input wire[1:0] COMMAND, input wire[2:0] INDEX, input wire[3:0] IDATA);
int Counter = 0;
int INDEX_GET;
reg[3:0] cell0;
reg[3:0] cell1;
reg[3:0] cell2;
reg[3:0] cell3;
reg[3:0] cell4;
always @(*) begin
  case ({RESET})
    1'b0: begin
      case ({CLK})
        1'b0: begin
          IODATA_out = 4'bZZZZ;
        end
        1'b1: begin
          case ({COMMAND})
            2'b00: begin // nop
              IODATA_out = 4'bZZZZ;
            end
            2'b01: begin // push
              if (Counter % 5 == 0) begin
              cell0 = IDATA;
              end else if (Counter % 5 == 1) begin
              cell1 = IDATA;
```

```
    end else if  (Counter % 5 == 2) begin
    cell2 = IDATA;
    end else if  (Counter % 5 == 3) begin
    cell3 = IDATA;
    end else begin
    cell4 = IDATA;
    end
    Counter = (Counter + 1) % 5;
    IODATA_out = 4'bZZZZ;
end
2'b10: begin // pop
    Counter = (Counter + 4) % 5;
    if (Counter % 5 == 0) begin
    IODATA_out = cell0;
    end else if (Counter % 5 == 1) begin
    IODATA_out = cell1;
    end else if  (Counter % 5 == 2) begin
    IODATA_out = cell2;
    end else if  (Counter % 5 == 3) begin
    IODATA_out = cell3;
    end else  begin
    IODATA_out = cell4;
    end
end
2'b11: begin // get
    INDEX_GET = (Counter + 9 - INDEX) % 5;
    if (INDEX_GET % 5 == 0) begin
```

```
                    IODATA_out = cell0;
                    end else if (INDEX_GET % 5 == 1) begin
                    IODATA_out = cell1;
                    end else if  (INDEX_GET % 5 == 2) begin
                    IODATA_out = cell2;
                    end else if  (INDEX_GET % 5 == 3) begin
                    IODATA_out = cell3;
                    end else  begin
                    IODATA_out = cell4;
                    end
                  end
                endcase
              end
            endcase
          end
          1'b1: begin
            Counter = 0;
            IODATA_out = 4'bZZZZ;
            cell0 = 0;
            cell1 = 0;
            cell2 = 0;
            cell3 = 0;
            cell4 = 0;
          end
        endcase
    end
endmodule
```

```
module stack_behaviour_normal(
   input wire RESET,
   input wire CLK,
   input wire[1:0] COMMAND,
   input wire[2:0] INDEX,
   inout wire[3:0] IO_DATA
   );
   stack stack(.IODATA_out(IO_DATA), .RESET(RESET), .CLK(CLK),
.COMMAND(COMMAND), .INDEX(INDEX), .IDATA(IO_DATA));

endmodule
```

Разберем программу по подробнее

Для начала разберемся с введенными переменами cell0, cell1, cell2, cell3, cell4 отвечают за ячейки стека. Counter показывает текущее индекс головки стека. INDEX_GET введется далее.

10-11 строчка разбираем, когда состояние reset = 0;

```
case ({RESET})
        1'b0: begin
```

68-77 строка разбираем, когда состояние reset = 1 (обнуляем все)

```
1'b1: begin
            Counter = 0;
            IODATA_out = 4'bZZZZ;
            cell0 = 0;
            cell1 = 0;
            cell2 = 0;
            cell3 = 0;
            cell4 = 0;
```

```
        end
```

12-13 строчка разбираем случай с выключенной синхронизацией

```
        case ({CLK})
            1'b0: begin
```

16 строчка синхронизация включена

```
1'b1: begin
```

17 строчка, разбираем разные команды

```
case ({COMMAND})
```

18-19 команда nop (ничего интересного)

```
2'b00: begin // nop
                IODATA_out = 4'bZZZZ;
```

21-34 строчки команда push

```
2'b01: begin // push
                if (Counter % 5 == 0) begin
                cell0 = IDATA;
                end else if (Counter % 5 == 1) begin
                cell1 = IDATA;
                end else if  (Counter % 5 == 2) begin
                cell2 = IDATA;
                end else if  (Counter % 5 == 3) begin
                cell3 = IDATA;
                end else begin
                cell4 = IDATA;
                end
                Counter = (Counter + 1) % 5;
                IODATA_out = 4'bZZZZ;
```

Смотрим куда показывает счетчик и в ячейку с этим номером вставляем данные, после этого номер счетчика необходимо увеличить на 1

37-48 строки команда pop

```
2'b10: begin // pop
                    Counter = (Counter + 4) % 5;
                    if (Counter % 5 == 0) begin
                    IODATA_out = cell0;
                    end else if (Counter % 5 == 1) begin
                    IODATA_out = cell1;
                    end else if (Counter % 5 == 2) begin
                    IODATA_out = cell2;
                    end else if  (Counter % 5 == 3) begin
                    IODATA_out = cell3;
                    end else  begin
                    IODATA_out = cell4;
                    end
                  end
```

Смотрим куда указывает счетчик, после этого уменьшаем значение счетчика на 1(это будет индекс последнего элемента), прибавив 4(одно и тоже так как у нас все по модулю 5). Выводим удаленное значение.

50-61 строка команда get

```
2'b11: begin // get
                    INDEX_GET = (Counter + 9 - INDEX) % 5;
                    if (INDEX_GET % 5 == 0) begin
                    IODATA_out = cell0;
                    end else if (INDEX_GET % 5 == 1) begin
                    IODATA_out = cell1;
                    end else if (INDEX_GET % 5 == 2) begin
                    IODATA_out = cell2;
                    end else if (INDEX_GET % 5 == 3) begin
```

```
                IODATA_out = cell3;
                end else  begin
                IODATA_out = cell4;
                end
            end
```

INDEX_GET показывает на ячейку значение которой нужно будет вывести для этого от Counter(то куда показывает головка стека) нужно вычесть (INDEX+1) это число до 8, и прибавим 10 чтобы не работать с отрицательными числами. Теперь просто нужно вывести значение этого числа.

В файле stack_behaviour_tb.sv напишем, команду monitor, чтобы посмотреть как все работает на тестах.

time 100: RESET=0, CLK=0, COMMAND=00, INDEX=000, IO_DATA=xxxx

time 101: RESET=1, CLK=0, COMMAND=00, INDEX=000, IO_DATA=xxxx

time 102: RESET=0, CLK=0, COMMAND=00, INDEX=000, IO_DATA=xxxx

time 103: RESET=0, CLK=0, COMMAND=00, INDEX=000, IO_DATA=0000

time 104: RESET=0, CLK=1, COMMAND=00, INDEX=000, IO_DATA=0000

time 105: RESET=0, CLK=0, COMMAND=00, INDEX=000, IO_DATA=0000

time 106: RESET=0, CLK=0, COMMAND=01, INDEX=000, IO_DATA=0001

time 107: RESET=0, CLK=1, COMMAND=01, INDEX=000, IO_DATA=0001

time 108: RESET=0, CLK=0, COMMAND=01, INDEX=000, IO_DATA=0001

time 109: RESET=0, CLK=0, COMMAND=01, INDEX=000, IO_DATA=0010

time 110: RESET=0, CLK=1, COMMAND=01, INDEX=000, IO_DATA=0010

time 111: RESET=0, CLK=0, COMMAND=01, INDEX=000, IO_DATA=0010

time 112: RESET=0, CLK=0, COMMAND=01, INDEX=000, IO_DATA=0011

time 113: RESET=0, CLK=1, COMMAND=01, INDEX=000, IO_DATA=0011

time 114: RESET=0, CLK=0, COMMAND=01, INDEX=000, IO_DATA=0011

time 115: RESET=0, CLK=0, COMMAND=01, INDEX=000, IO_DATA=0100

time 116: RESET=0, CLK=1, COMMAND=01, INDEX=000, IO_DATA=0100

time 117: RESET=0, CLK=0, COMMAND=01, INDEX=000, IO_DATA=0100

time 118: RESET=0, CLK=0, COMMAND=01, INDEX=000, IO_DATA=0101

time 119: RESET=0, CLK=1, COMMAND=01, INDEX=000, IO_DATA=0101

time 120: RESET=0, CLK=0, COMMAND=01, INDEX=000, IO_DATA=0101

time 121: RESET=0, CLK=0, COMMAND=01, INDEX=000, IO_DATA=0110

time 122: RESET=0, CLK=1, COMMAND=01, INDEX=000, IO_DATA=0110

time 123: RESET=0, CLK=0, COMMAND=01, INDEX=000, IO_DATA=0110

time 124: RESET=0, CLK=0, COMMAND=10, INDEX=000, IO_DATA=zzzz

time 125: RESET=0, CLK=1, COMMAND=10, INDEX=000, IO_DATA=0110

time 127: RESET=0, CLK=0, COMMAND=10, INDEX=000, IO_DATA=zzzz

time 129: RESET=0, CLK=1, COMMAND=10, INDEX=000, IO_DATA=0101

time 131: RESET=0, CLK=0, COMMAND=10, INDEX=000, IO_DATA=zzzz

time 133: RESET=0, CLK=1, COMMAND=10, INDEX=000, IO_DATA=0100

time 135: RESET=0, CLK=0, COMMAND=10, INDEX=000, IO_DATA=zzzz

time 137: RESET=0, CLK=1, COMMAND=10, INDEX=000, IO_DATA=0011

time 139: RESET=0, CLK=0, COMMAND=10, INDEX=000, IO_DATA=zzzz

time 141: RESET=0, CLK=1, COMMAND=10, INDEX=000, IO_DATA=0010

time 143: RESET=0, CLK=0, COMMAND=10, INDEX=000, IO_DATA=zzzz

time 144: RESET=0, CLK=0, COMMAND=01, INDEX=000, IO_DATA=1111

time 145: RESET=0, CLK=1, COMMAND=01, INDEX=000, IO_DATA=1111

time 146: RESET=0, CLK=0, COMMAND=01, INDEX=000, IO_DATA=1111

time 147: RESET=0, CLK=0, COMMAND=01, INDEX=000, IO_DATA=1110

time 148: RESET=0, CLK=1, COMMAND=01, INDEX=000, IO_DATA=1110

time 149: RESET=0, CLK=0, COMMAND=01, INDEX=000, IO_DATA=1110

time 150: RESET=0, CLK=0, COMMAND=01, INDEX=000, IO_DATA=1100

time 151: RESET=0, CLK=1, COMMAND=01, INDEX=000, IO_DATA=1100

time 152: RESET=0, CLK=0, COMMAND=01, INDEX=000, IO_DATA=1100

time 153: RESET=0, CLK=0, COMMAND=01, INDEX=000, IO_DATA=1000

time 154: RESET=0, CLK=1, COMMAND=01, INDEX=000, IO_DATA=1000

time 155: RESET=0, CLK=0, COMMAND=01, INDEX=000, IO_DATA=1000

time 156: RESET=0, CLK=0, COMMAND=01, INDEX=000, IO_DATA=0000

time 157: RESET=0, CLK=1, COMMAND=01, INDEX=000, IO_DATA=0000

time 158: RESET=0, CLK=0, COMMAND=01, INDEX=000, IO_DATA=0000

time 159: RESET=0, CLK=0, COMMAND=11, INDEX=000, IO_DATA=zzzz

time 160: RESET=0, CLK=1, COMMAND=11, INDEX=000, IO_DATA=0000

time 162: RESET=0, CLK=0, COMMAND=11, INDEX=000, IO_DATA=zzzz

time 163: RESET=0, CLK=1, COMMAND=11, INDEX=001, IO_DATA=1000

time 165: RESET=0, CLK=0, COMMAND=11, INDEX=001, IO_DATA=zzzz

time 166: RESET=0, CLK=1, COMMAND=11, INDEX=010, IO_DATA=1100

time 168: RESET=0, CLK=0, COMMAND=11, INDEX=010, IO_DATA=zzzz

time 169: RESET=0, CLK=1, COMMAND=11, INDEX=011, IO_DATA=1110

time 171: RESET=0, CLK=0, COMMAND=11, INDEX=011, IO_DATA=zzzz

time 172: RESET=0, CLK=1, COMMAND=11, INDEX=100, IO_DATA=1111

time 174: RESET=0, CLK=0, COMMAND=11, INDEX=100, IO_DATA=zzzz

time 175: RESET=0, CLK=0, COMMAND=10, INDEX=100, IO_DATA=zzzz

time 176: RESET=0, CLK=1, COMMAND=10, INDEX=100, IO_DATA=0000

time 178: RESET=0, CLK=0, COMMAND=10, INDEX=100, IO_DATA=zzzz

time 180: RESET=0, CLK=1, COMMAND=10, INDEX=100, IO_DATA=1000

time 182: RESET=0, CLK=0, COMMAND=10, INDEX=100, IO_DATA=zzzz

time 184: RESET=0, CLK=1, COMMAND=10, INDEX=100, IO_DATA=1100

time 186: RESET=0, CLK=0, COMMAND=10, INDEX=100, IO_DATA=zzzz

time 188: RESET=0, CLK=1, COMMAND=10, INDEX=100, IO_DATA=1110

time 190: RESET=0, CLK=0, COMMAND=10, INDEX=100, IO_DATA=zzzz

time 192: RESET=0, CLK=1, COMMAND=10, INDEX=100, IO_DATA=1111

time 194: RESET=0, CLK=0, COMMAND=10, INDEX=100, IO_DATA=zzzz

Как можно увидеть все работает просто отлично).

На этом и закончим)