Бобовский Кирилл	M3138	Построение логических схем

https://github.com/skkv-itmo/itmo-comp-arch-2023-circuit-WannaSleep43

Logisim Version v3.8.0	Icarus Verilog version 12.0 (stable) (v12_0-dirty)

Logisim

Схема RS-Триггера:

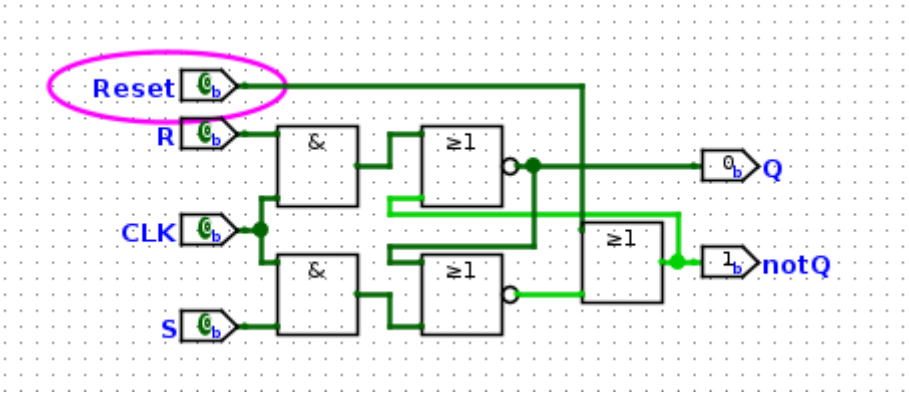
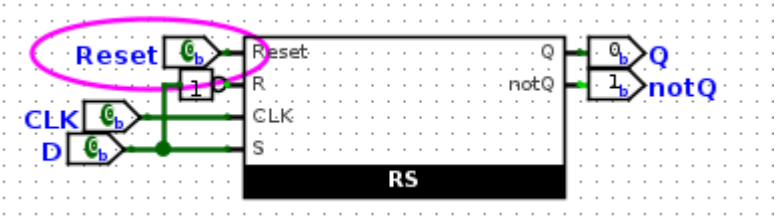
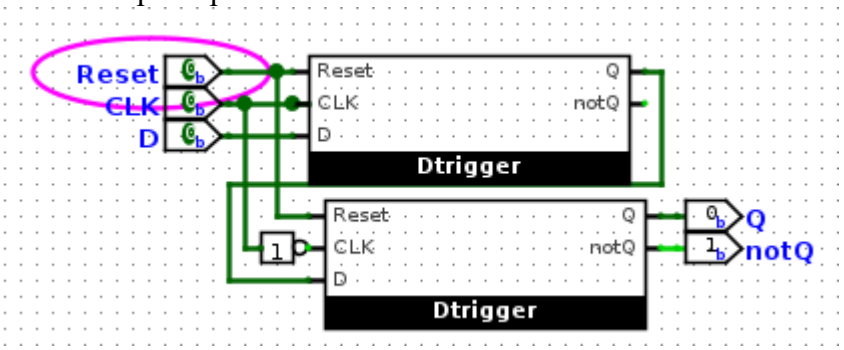


Схема D-Триггера



D-Триггер передает значение D при CLK равном одному и сохраняет последнее значение при CLK равном нулю. Так же на его основе собирается Динамически Д-Триггер, который сохраняет значение, а при переключении CLK с 0 на 1 запоминает значение входа D.
Схема Динамического D-Триггера:

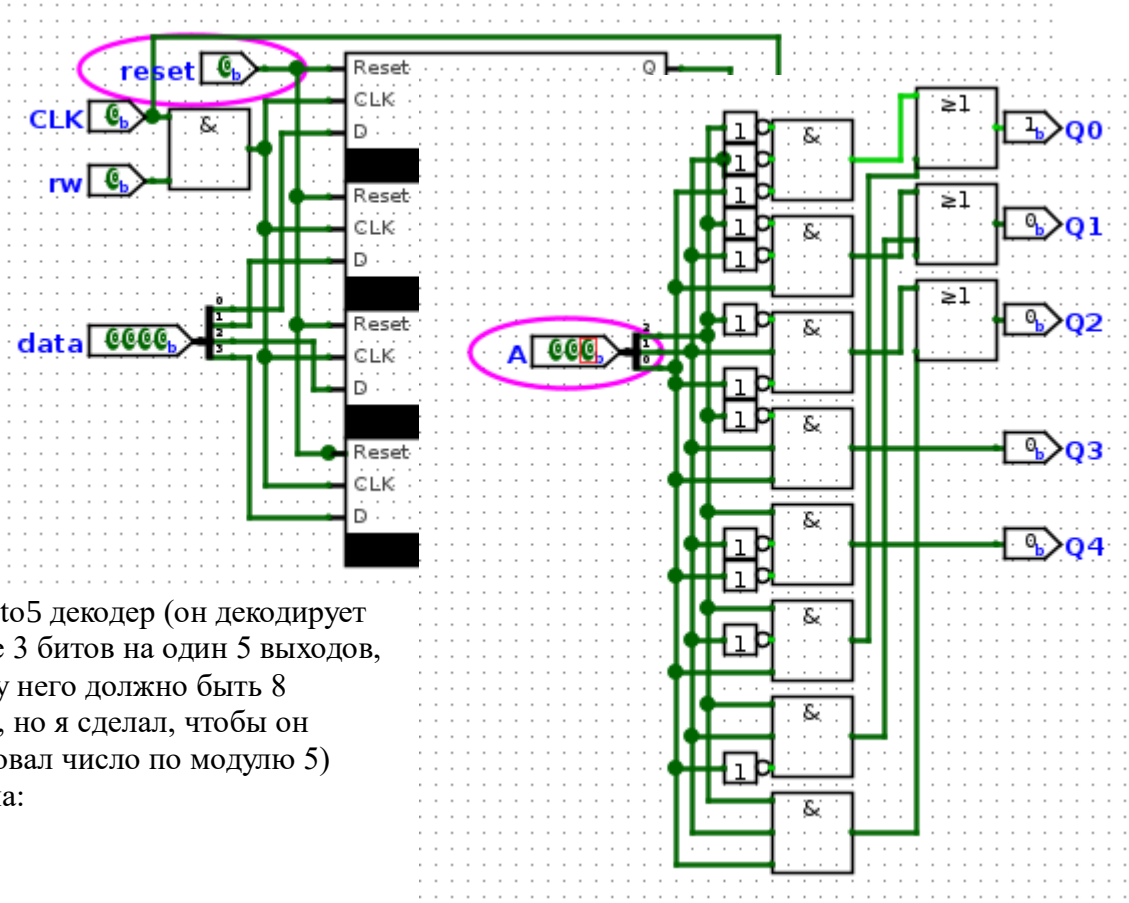


На основе динамического D-Триггера собирается Ttrigger, который по сути является счётчиком. Каждый раз, когда ClK меняется с 1 на 0, он изменяет свое значение на противоположенное.

Схема Ttrigger’a

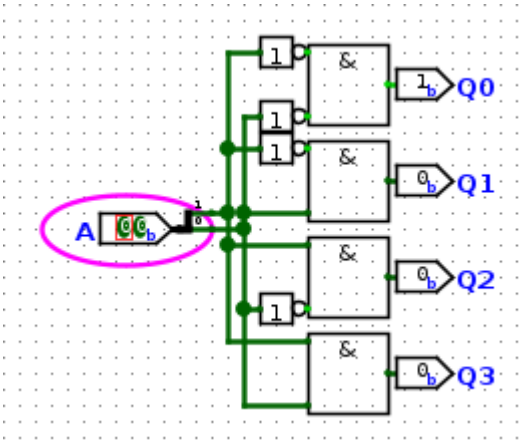
Далее я собрал блок памяти, который умеет записывать и хранить значение 4 битов. Он основан на обычных D-Триггерах, если CLK=1 и флаг rw=1(а значит, мы хотим записать значение) в CLK D-Триггеров подается 1, а в значение биты, которые мы хотим запомнить, и они их сохраняют. Если же мы не хотим изменять значение, а хотим лишь считать его, то в CLK триггеров подается 0 и они выводят то, что хранят.

Схема Memory:

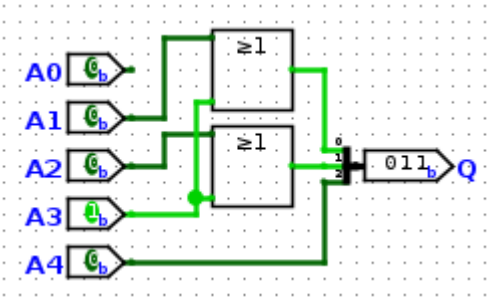


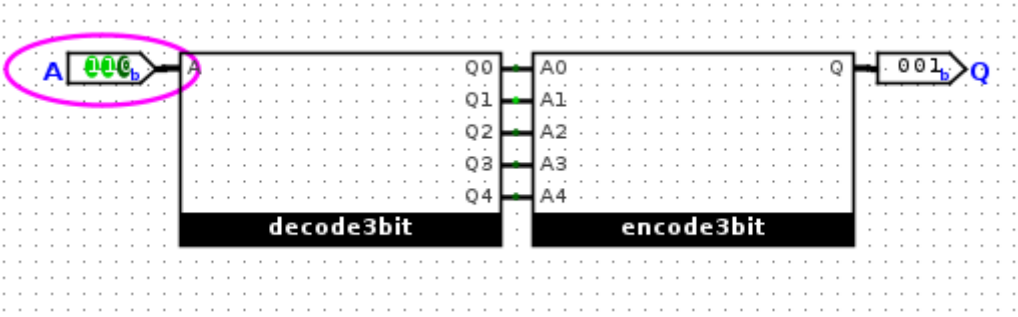
собрал 3to5 декодер (он декодирует значение 3 битов на один 5 выходов, по сути у него должно быть 8 выходов, но я сделал, чтобы он декодировал число по модулю 5)
Его схема:

Так же в процессе выполнения работы мне понадобился 2to4 декодер и его я тоже собрал.

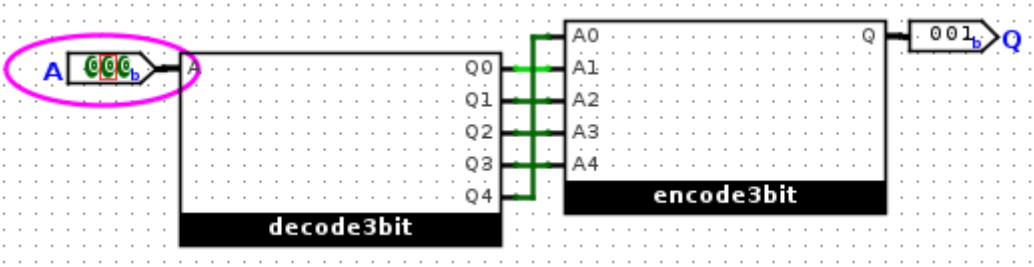


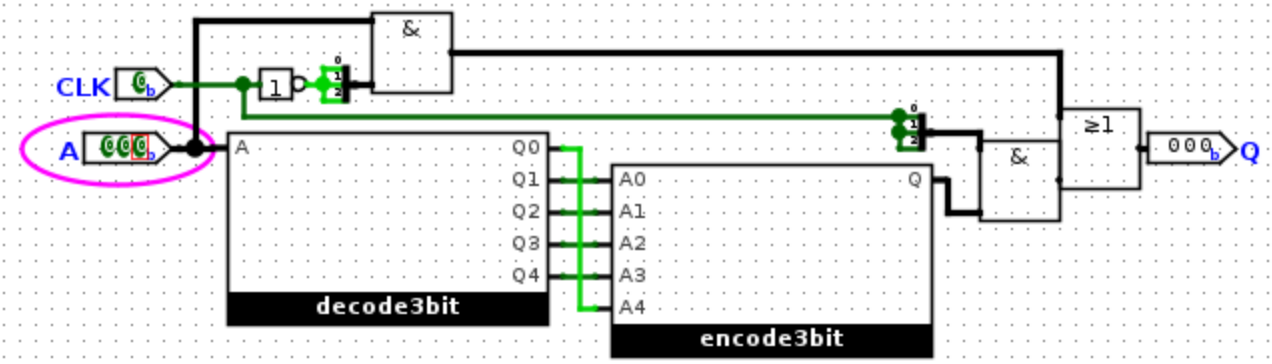
Ещё так как мой стек поддерживает только 5 элементов, мне нужно было взять значение индекса по модулю 5, поэтому я собрал схему, которая принимает единицу на один из 5 входов и выдает его номер (по сути делает обратное декодеру). И с помощью ее собрал схему, которая берет число по модулю 5 и возвращает его. Сами схемы:



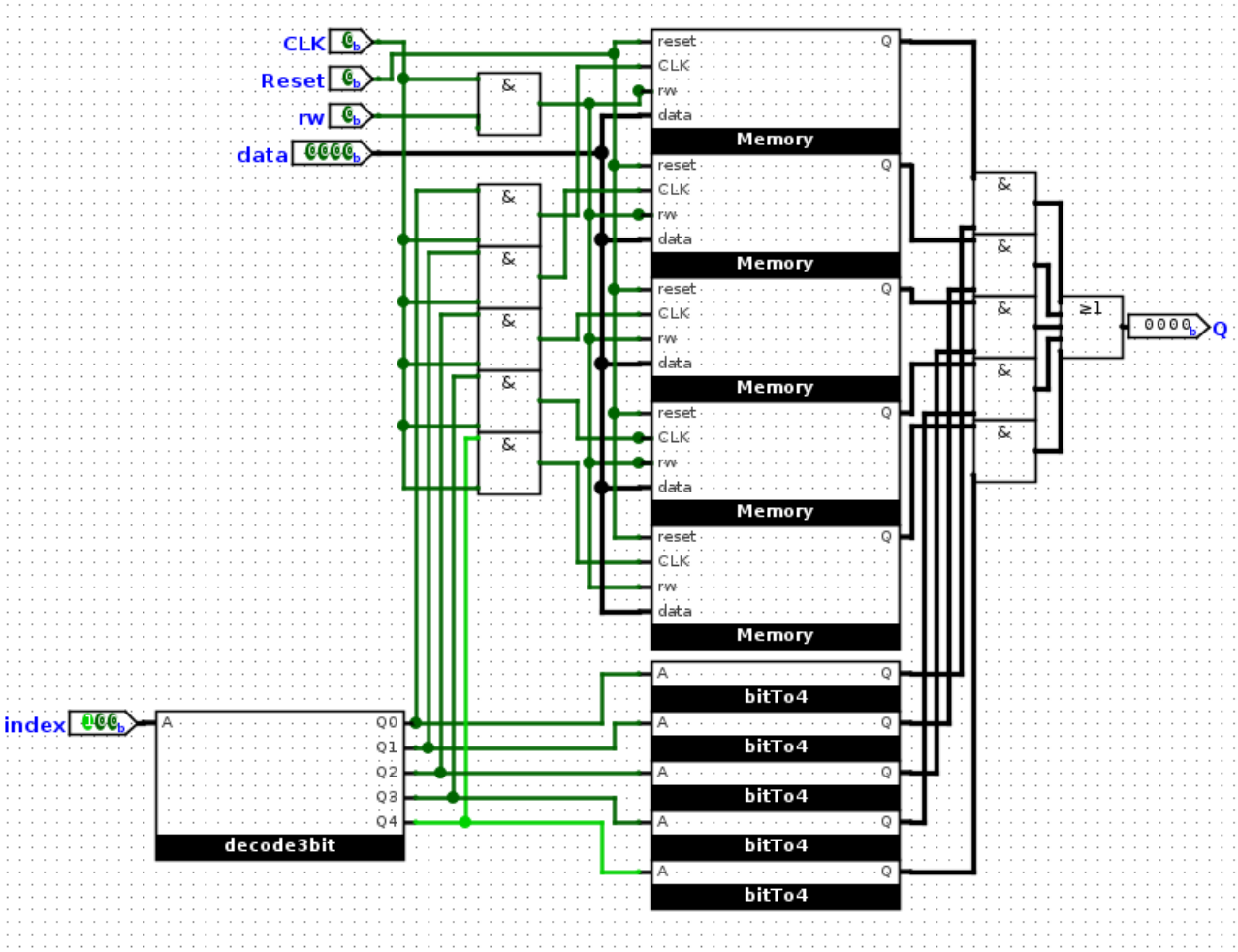


Ещё мне пригодились схемки, которые умеют делать +-1 к числу на входе и я собрал их.

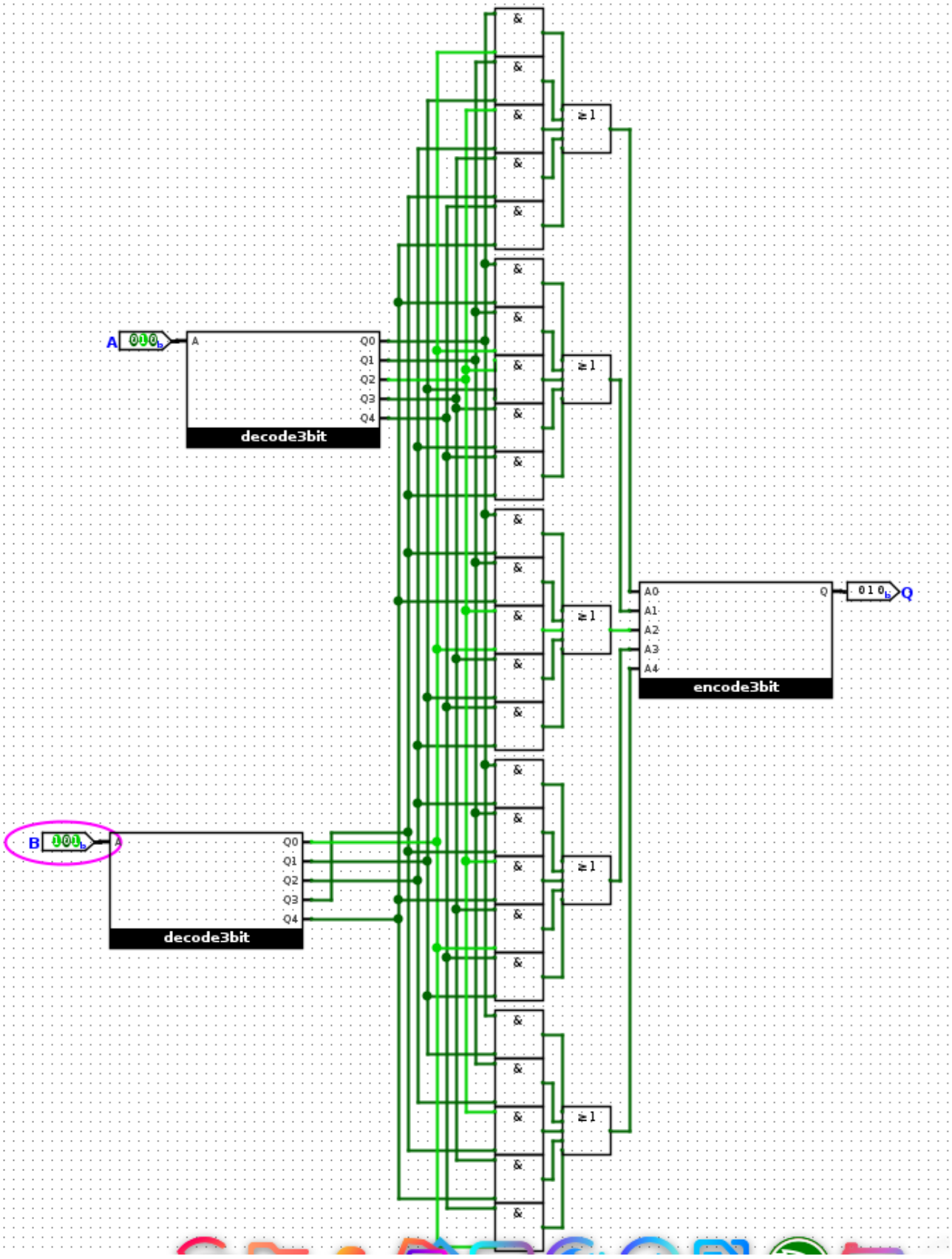




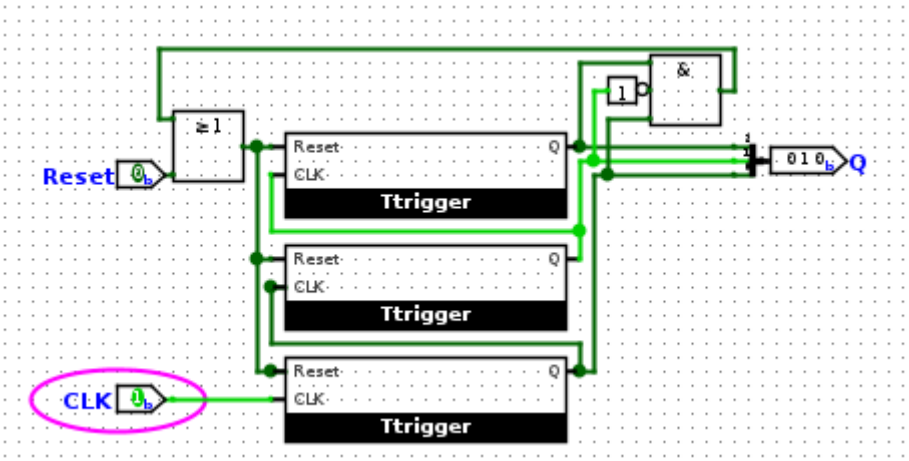
Затем для удобства я собрал модуль памяти для стека, состоящий из 5 обычных модулей памяти. Его схема:



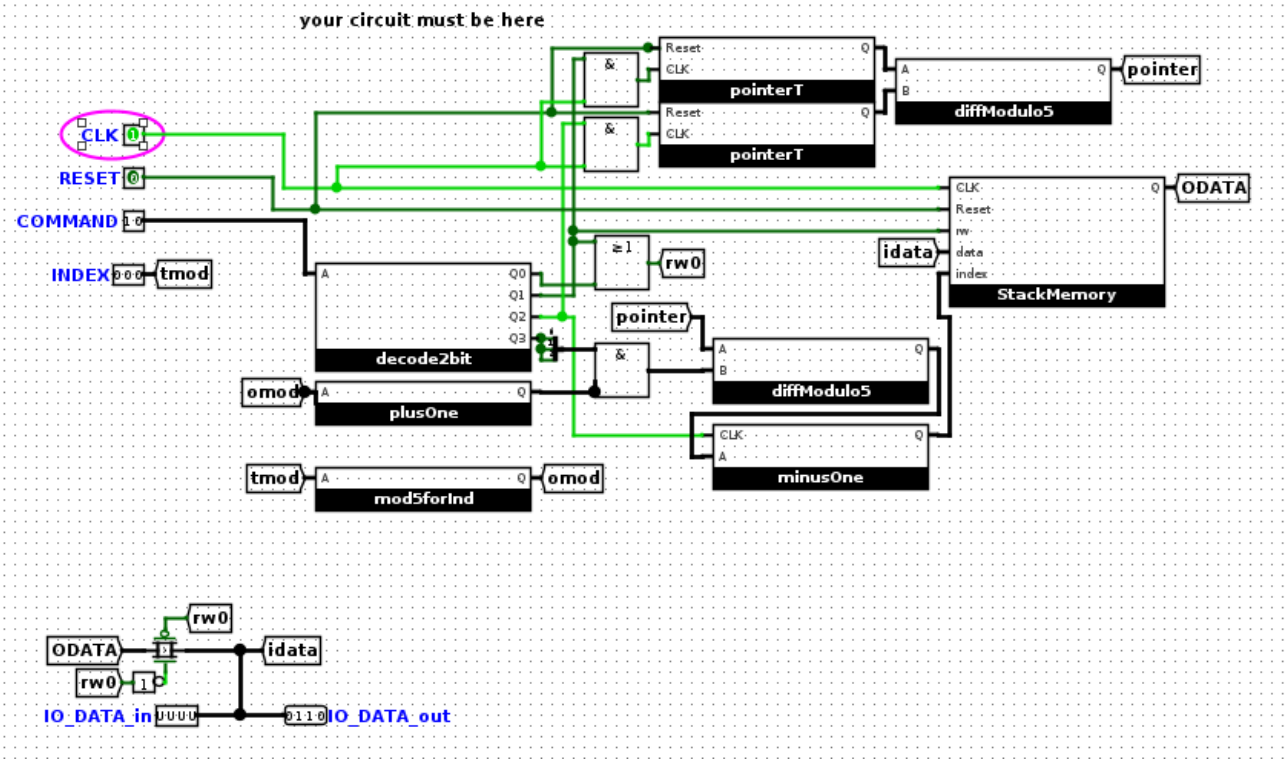
Ещё я собрал схему для разности 2 3-битных чисел по модулю 5, для этого я просто разобрал все 25 случаев.


A
010b
A
Q0
Q1
Q2
Q3
Q4
decode3bit
B
001b
A
Q0
Q1
Q2
Q3
Q4
decode3bit
&
≥1
A0
A1
A2
A3
A4
Q
010b
Q
encode3bit

Чтобы сохранять указатель и уметь увеличивать его на 1, я собрал для него отдельную схемку на 3 T-триггерах. Изначально указатель равен 1 и при изменении CLK с 0 на 1, он увеличивается на 1. Его значение считается по модулю 5.
Схема:



Ну и, наконец-то, сам стэк:



В задаче мы хотим увеличивать указатель на 1 для операции push и уменьшать на 1 для операции pop. Я сделал это при помощи 2 указателей:
в первый я прибавляю 1, когда операция push, во второй — когда операция pop. Тогда текущий элемент это разность push — pop по модулю 5.

Я беру команду и определяю ее тип, если команда 1 — я хочу передать в память флаг rw 1 и сказать, что мы хотим запомнить текущее значение. Я передаю памяти CLK, Reset, этот rw, , введенные данные и индекс элемента, который хочу записать — вычисленное значение указателя.

Формула, по которой вычисляется значение индекса:

index = (PointerPlus — PointerMinus — ((commad == 3) ? INDEXmod5 + 1 : 0);

Когда я генерирую запрос, он передается в память и ответ выходит в ODATA.
Отдельно стоит пояснить, как работает входо-выход. Между выходом и ODATA стоит передаточный вентиль, который открывается, если команда 2 или 3 и мы хотим вывести значение из памяти. В этом случае он передает ODATA на выход, который изначально заполнен UUUU. Если операция не требует считывать значение из памяти, то вентиль закрывается и выводится то же, что и ввелось.

Верилог(Behaviour)

Здесь проще приложить весь код, потому что он короткий.

SV/Verilog Testbench

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

    reg[3:0] stack[4:0];
    reg[2:0] last;

    integer rev = 3'b0;
    reg rw = 0;
    reg[3:0] out;
    assign IO_DATA = (rw) ? out : 4'bZZZZ;

    always @(RESET) begin
        for (integer i = 0; i < 5; i++) begin
            stack[i] = 5'b0;
        end
        last = 3'b0;
        rw = 0;
    end

    always @(posedge CLK) begin
        case (COMMAND)
            2'b00: begin rw = 0; end
            2'b01: begin
                stack[last] = IO_DATA;
                rw = 0;
                last = (last == 4) ? 0 : last + 1;
            end
            2'b10: begin
                last = (last == 0) ? 4 : last - 1;
                rw = 1;
                out = stack[last];
            end
            2'b11: begin
                rev = (last + 5 - ((INDEX + 1) % 5)) % 5;
                rw = 1;
                out = stack[rev];
            end
        endcase
    end
endmodule
```

Так же, как и в логисиме делаем флаг rw, который включен, если операция 2 или 3 типа.
При помощи тернарного условия привязываем к проводу IO_DATA регистр out, если rw=1 и мы хотим вывести значение из памяти, иначе привязываем ZZZZ;

Пока RESET включен, то обнулям начения стека, указатель last на элемент после последнего заполненного и флаг rw.

Когда CLK переключается с 0 на 1, обрабатываем команду следующим образом:
Если команда 0, то просто ставим rw на 0
Если команда 1, то записываем последний элемент в стек и увеличиваем указатель на 1 (по модулю 5)
Если команда 2, то уменьшаем указатель на 1 (по модулю 5), ставим rw=1, потому что хотим вывести значение и передаем на выход значение последнего элемента.

Иначе считываем индекс, считаем, какой элемент нас просят вывести и выводим его. Rw переключаем на 1.

Верилог(Structural)

Structural реализация верилога по сути является схемой из логисима, переписанной на веилог и ничем не отличается. В качестве примеров просто приведу пару фрагментов кода, как я реализовывал какие-то отдельные модули.

Триггеры:

```
module RS(output wire Q, output wire notQ, input wire reset, input wire R, input wire CLK, input wire S);
    and and1(and1_out, R, CLK), and2(and2_out, S, CLK);
    nor nor1(Q, notQ, and1_out), nor2(res, Q, and2_out);
    or or1(notQ, reset, res);
endmodule

module Dtrigger (output wire Q, output wire notQ,input wire Reset, input wire CLK, input wire D);
    wire notD;
    not not1(notD, D);
    RS RS1(Q, notQ, Reset, notD, CLK, D);
endmodule

module DynamicD(output wire Q, output wire notQ, input wire Reset, input wire CLK, input wire D);

    wire notCLK, D1, notD1;
    not not1(notCLK, CLK);

    Dtrigger Dtrigger1(D1, notD1, Reset, CLK, D);
    Dtrigger Dtrigger2(Q, notQ, Reset, notCLK, D1);
endmodule

module Ttrigger(output wire Q, input wire Reset, input wire CLK);
    wire nQ;
    DynamicD Dynamic1(Q, nQ, Reset, CLK, nQ);
endmodule
```

Память и память стека:

```
module Memory(output wire[3:0] Q, input wire reset, input wire CLK, input wire rw, input wire[3:0] data);
    wire CLKandrw, q1, q2, q3, q4, notq1, notq2, notq3, notq4;

    and and1(CLKandrw, CLK, rw);
    Dtrigger Dtrigger1(q1, notq1, reset, CLKandrw, data[0]), Dtrigger2(q2, notq2, reset, CLKandrw, data[1]),
Dtrigger3(q3, notq3, reset, CLKandrw, data[2]), Dtrigger4(q4, notq4, reset, CLKandrw, data[3]);

    wire[3:0] resMem, clk4;
    assign resMem[0] = q1, resMem[1] = q2, resMem[2] = q3, resMem[3] = q4;
    assign clk4[0] = CLK, clk4[1] = CLK, clk4[2] = CLK, clk4[3] = CLK;

    and4bit and2(Q, clk4, resMem);
endmodule

module StackMemory(output wire[3:0] Q, input wire CLK, input wire reset, input wire rw, input wire[3:0]
data, input wire[2:0] index);
    wire clkrw;
    and and999(clkrw, CLK, rw);

    wire i0, i1, i2, i3, i4;
    decode3bit dec(i0, i1, i2, i3, i4, index);

    wire clki0, clki1, clki2, clki3, clki4;
    and and0(clki0, CLK, i0), and1(clki1, CLK, i1), and2(clki2, CLK, i2), and3(clki3, CLK, i3), and4(clki4,
CLK, i4);

    wire[3:0] q0, q1, q2, q3, q4;
    Memory mem0(q0, reset, clki0, clkrw, data), mem1(q1, reset, clki1, clkrw, data), mem2(q2, reset, clki2,
clkrw, data), mem3(q3, reset, clki3, clkrw, data), mem4(q4, reset, clki4, clkrw, data);

    wire[3:0] i4b0, i4b1, i4b2, i4b3, i4b4;
    bitTo4 to0(i4b0, i0), to1(i4b1, i1), to2(i4b2, i2), to3(i4b3, i3), to4(i4b4, i4);

    wire[3:0] res0, res1, res2, res3, res4;
    and4bit and4b0(res0, q0, i4b0), and4b1(res1, q1, i4b1), and4b2(res2, q2, i4b2), and4b3(res3, q3, i4b3),
and4b4(res4, q4, i4b4);

    wire[3:0] Q1, Q2, Q3;
    or4bit or0(Q1, res0, res1), or1(Q2, Q1, res2), or2(Q3, Q2, res3), or3(Q, Q3, res4);
endmodule
```

Сам стек:

```
module stack_structural_normal(
    inout wire[3:0] IO_DATA,
    input wire RESET,
    input wire CLK,
    input wire[1:0] COMMAND,
    input wire[2:0] INDEX
    );

    wire cmd0, cmd1, cmd2, cmd3;
    decode2bit dec2bit(cmd0, cmd1, cmd2, cmd3, COMMAND);

    wire[2:0] cmd3bit3;
    assign cmd3bit3[0] = cmd3, cmd3bit3[1] = cmd3, cmd3bit3[2] = cmd3;

    wire clkPlus, clkMinus;
    and andplus(clkPlus, CLK, cmd1), andminus(clkMinus, CLK, cmd2);

    wire[2:0] pPlus, pMinus;

    pointerT plus(pPlus, RESET, clkPlus), minus(pMinus, RESET, clkMinus);

    wire[2:0] pointer;
    diffModulo5 plusMinus(pointer, pPlus, pMinus);

    wire rw0, notrw0;
    or orrw0(rw0, cmd0, cmd1);
    not notrw(notrw0, rw0);

    wire[2:0] modindex;
    mod5forInd mod(modindex, INDEX);

    wire[2:0] indPlOne;
    plusOne add(indPlOne, modindex);

    wire[2:0] indMask;
    and3bit maskInd(indMask, indPlOne, cmd3bit3);

    wire[2:0] pointerMinusInd;
    diffModulo5 pointerInd(pointerMinusInd, pointer, indMask);

    wire[2:0] finalIndex;
    minusOne pointerMinus(finalIndex, cmd2, pointerMinusInd);

    wire[3:0] memOut;
    StackMemory stMemory(memOut, CLK, RESET, cmd1, IO_DATA, finalIndex);

    cmos cmos0(IO_DATA[0], memOut[0], notrw0, rw0), cmos1(IO_DATA[1], memOut[1], notrw0, rw0),
cmos2(IO_DATA[2], memOut[2], notrw0, rw0), cmos3(IO_DATA[3], memOut[3], notrw0, rw0);
endmodule
```

Особенностью реализации схемы на верилоге является то, что для операций с базовыми модулями (например and, or, xor, cmos(транзистор) и.т.д.) не определены операции от проводов, передающих несколько битов(например, 3) поэтому, для удобства, приходилось самому писать такие модули. Например:

```
module and3bit(output wire[2:0] Q, input wire[2:0] A, input wire[2:0] B);
    and and1(Q[0], A[0], B[0]), and2(Q[1], A[1], B[1]), and3(Q[2], A[2], B[2]);
endmodule

module or3bit(output wire[2:0] Q, input wire[2:0] A, input wire[2:0] B);
    or or1(Q[0], A[0], B[0]), or2(Q[1], A[1], B[1]), or3(Q[2], A[2], B[2]);
endmodule
```