Лабораторная работа №2	Группа 06	2023
Построение логических схем	Прозорова Варвара Владиславовна	

Инструментарий и требования к работе:

Logisim-evolution и Icarus Verilog 10

Ссылка на репозиторий:

https://github.com/skkv-mkn/mkn-comp-arch-2023-circuit-stepashka-21

Схема в logisim.

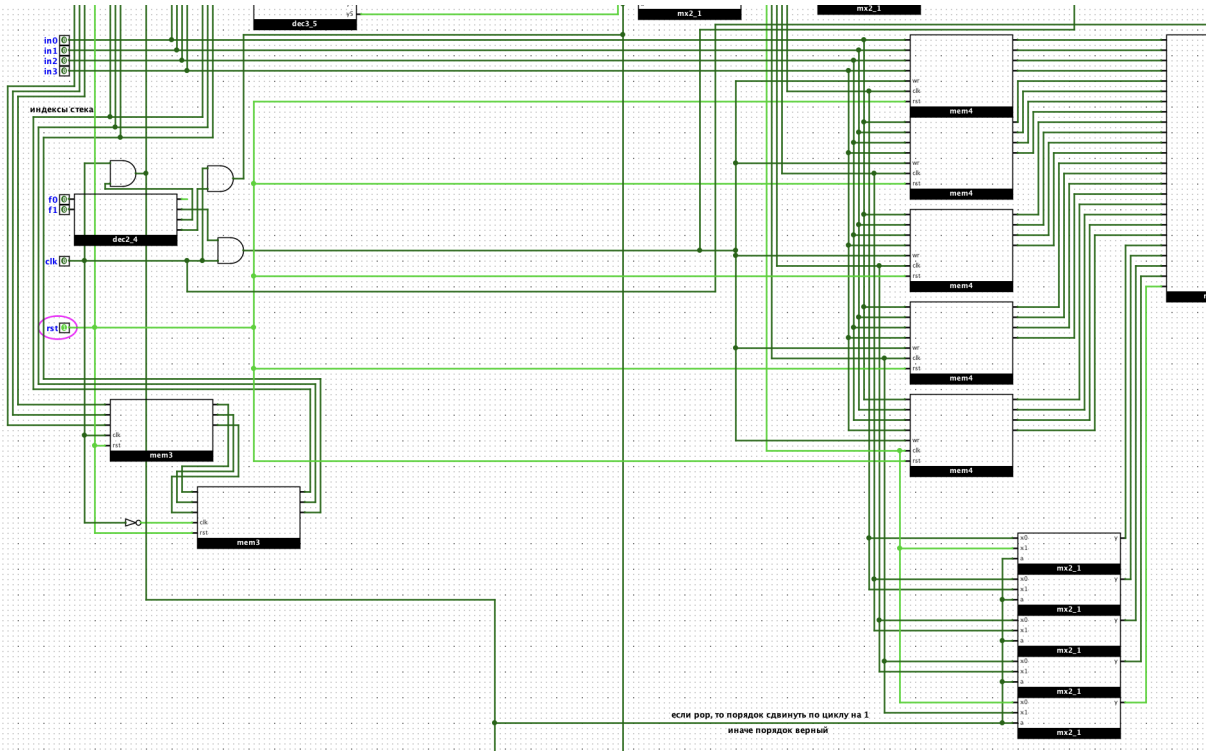
*___ – функция реализована в verilog, в момент производства схемы в logisim так сделать не догадалась.
dec3_5 – стандартный дешифратор (получается 5 битов, так как на вход подаются нормальные числа, приведенные по модулю 5), но выводит элементы снизу вверх, так как стек будет заполняться по уровням снизу вверх (если будет заполнена пятая ячейка, то потом он перескочит в первую, все норм).
dec2_4 – стандартный дешифратор.
c5_3, c4_2 – стандартные шифраторы.
mx2_1 – стандартный мультиплексор для двух битов.
***mx3_1** – мультиплексор для трех битов, получается из 3 обычных мультиплексоров попарным “сравнением”.
***mx4_1** – мультиплексор для четырех битов, получается из 4 обычных мультиплексоров попарным “сравнением”.
***mx5_1** – мультиплексор для пяти битов, получается из 5 обычных мультиплексоров попарным “сравнением”.
mx20_4 – мультиплексор для 20 битов с выводом 4 из них, то есть 5 раз сравнивает четверки битов.
jk_trig – обычный jk-триггер с reset (в verilog его нет, там сразу d_trig)
d_trig – тот же самый jk-триггер только j и k становятся d и -d соответственно.
mem1 – ячейка памяти для одного бита x из d_trig с функцией reset (вход rst).

mem2, mem3 – ячейки памяти для двух и трех битов из mem1.
mem4 – ячейка памяти для четырех битов с функцией wr – нужно ли что-то менять, пригодится при push.
add1mod5 – получает на вход 2 бита и отдельный бит для возможного переноса, складывает стандартно.
add3mod5 – получает на вход 3 бита одного числа и 3 бита другого числа, складывает стандартно в столбик, в случае первого сложения переноса нет, поэтому константа 0. Если получается 4 бита, то с помощью mod5_4bit переводит их в три бита.
mod5min3 – получает на вход число x, приведенное по модулю 5, кодирует его 5 элементов, затем переворачивает эти 5 элементов, получается число 5 - x, а потом нужно прибавить 1, так как шифратор тоже прибавляет 1, но при переворачивании получается -1.
mod5_4bit – получает на вход 4 бита (q0, q1, q2, q3), попарно сравнивает первый с третьим и второй с четвертым, так как двоичная запись кодируется как q0*1 + q1*2 + q2*4 + q3*(8 mod 5 = 3). То есть если первый и третий или второй и четвертый вместе или 1, или 0, то можно сразу писать два нуля, ведь по модулю 5 они и правда 0. Дальше прибавляет к концовке числа 11 (то есть как бы 3) для случая, если первый (четвертый) бит так и остался 1 (то есть можно стереть этот старший бит, но только тогда, когда к числу прибавится 3), смотрит на возможные переносы, в случае, когда получается 110, выводит 001.
stack – здесь происходит все работа стека; отдельно реализовывается работа с индексами: изначально 0, потом в зависимости от команды либо ничего не происходит, если nop; либо делаются замены в стеке и потом к индексу +1 mod 5, если push; либо выводит нужное значение по индексу = текущий индекс -1 mod 5, и сам индекс -1 mod 5, если pop; либо берет значение индекса со входа, вычитает это значение из 5 с помощью mod5min3 и прибавляет его к текущему индексу стека (x - y = x + (5 - y) mod5), сам индекс при этом сохраняется для последующих операций, а на выход идет значение из ячейки с получившимся индексом, если get. Каждая из ячеек хранится в mem4, в каждом из mem4 по 4 бита значения. Как работают все mem4? Если команда clk + push, то срабатывает wr, который является символом того, что нужно перезаписать стек, иначе ничего не происходит. В случае get и pop необходим вывод. Это реализуется в самом низу схемы. В случае pop сначала нужно сдвинуть индексы на 1 по циклу, если get, то все и так уже на нужном месте.

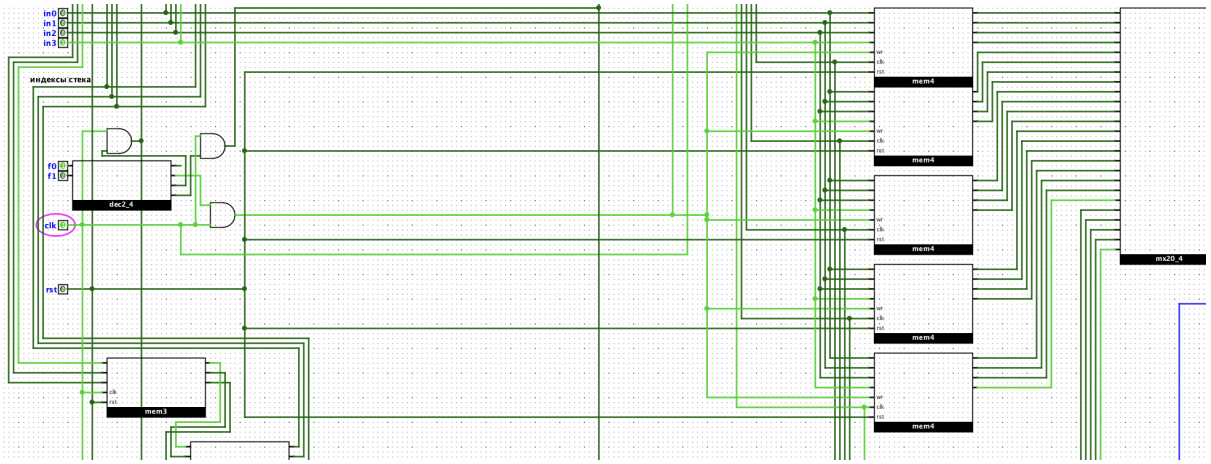
Проверка работы:

Приходится показывать работу именно на этой схеме, так как у меня реализован вывод значений только при командах pop и get.

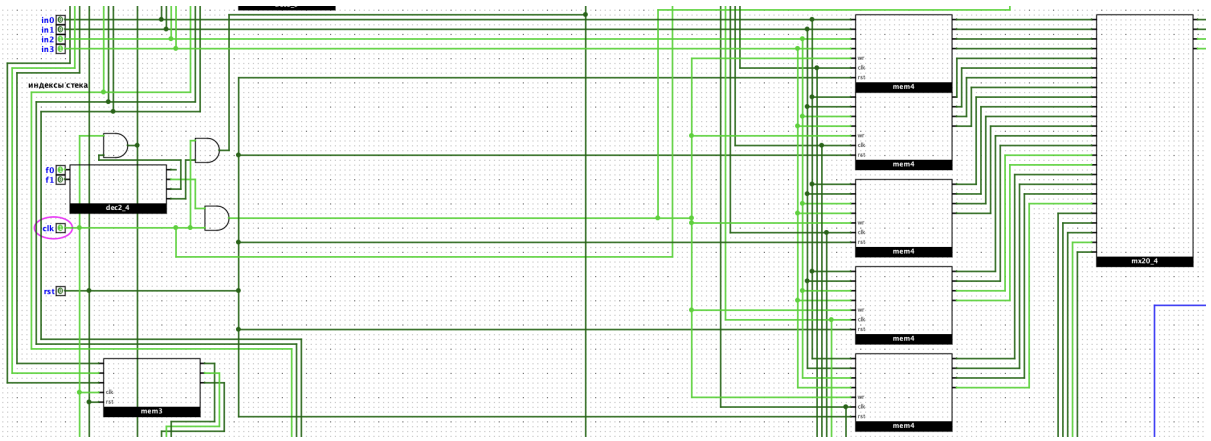
reset:



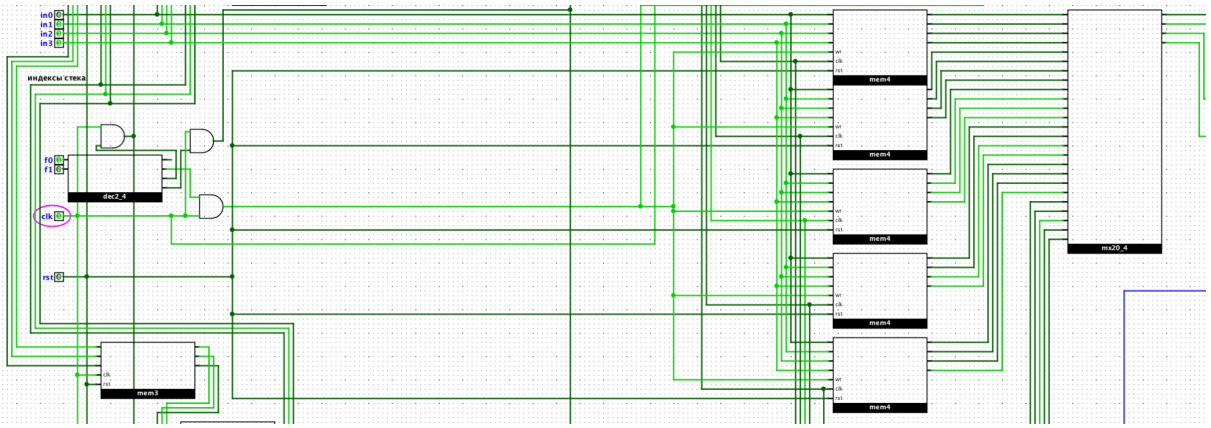
push 0001 (справа синий провод означает, что вывода нет; у нижнего mem’а видно, что нижний провод один стал светло зеленый, там записалась 1):



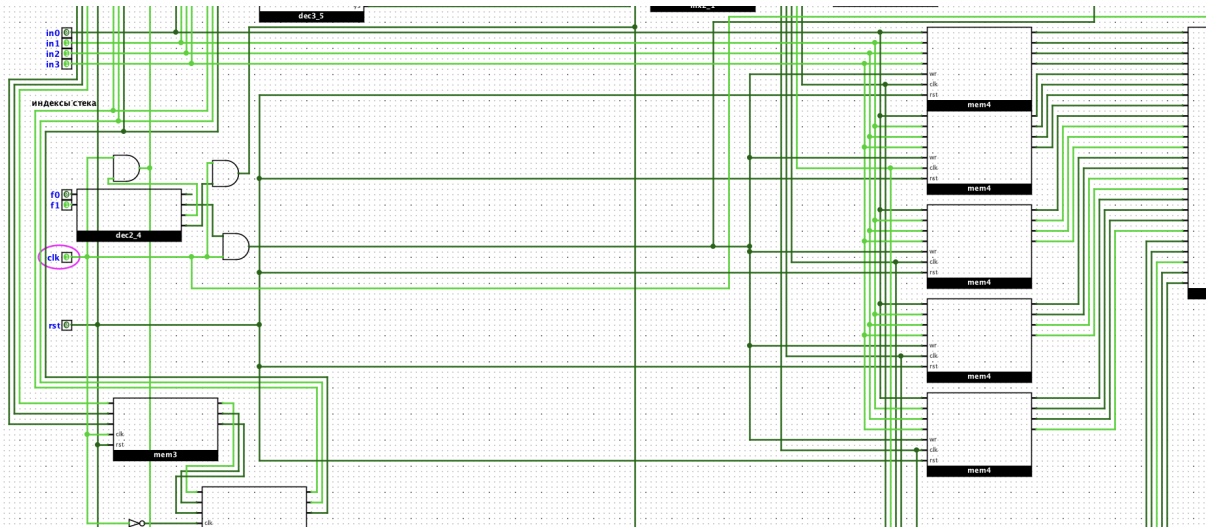
push 0011:



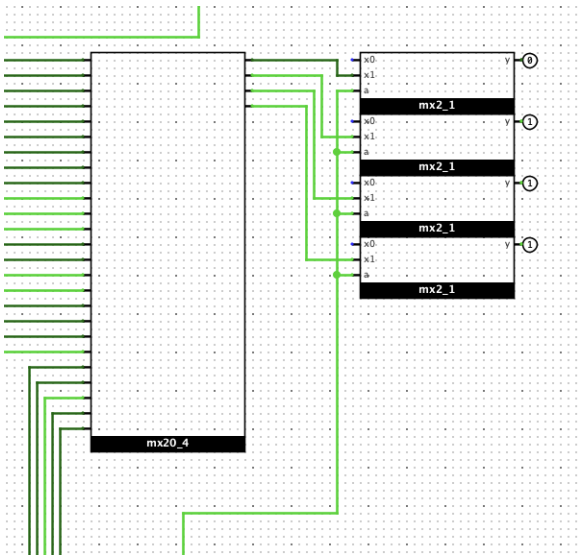
push 0111:



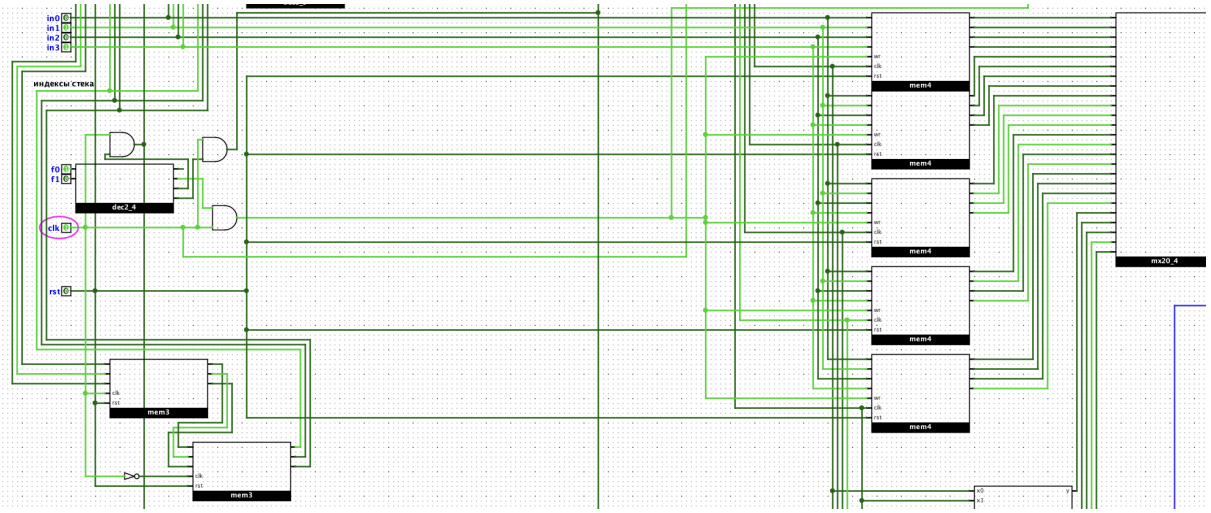
pop:



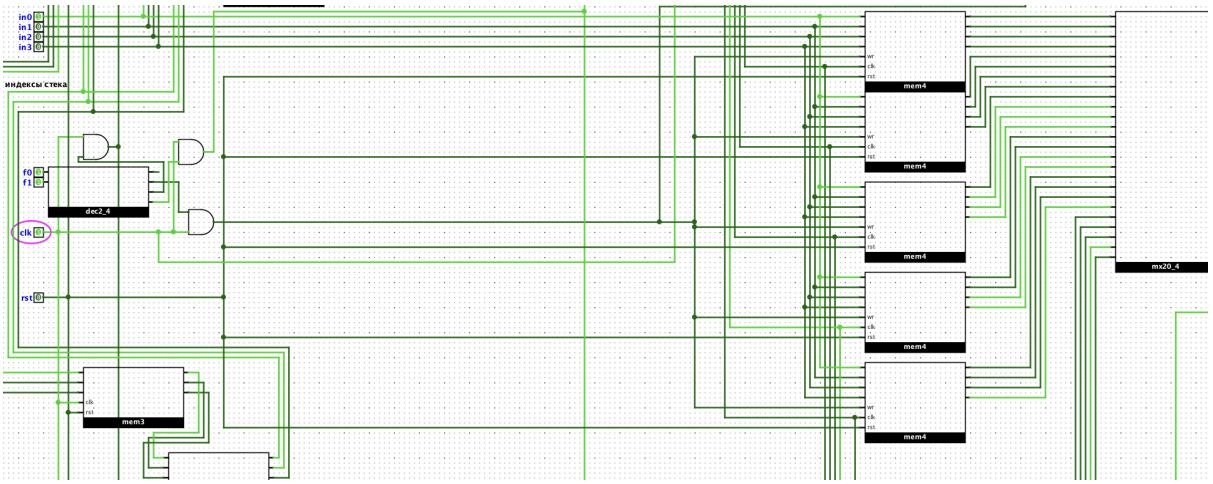
и выводится последний:

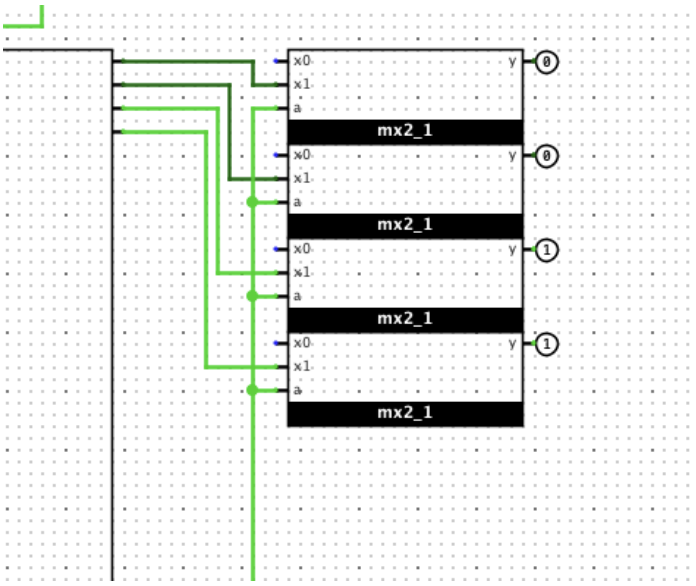


push 0101 (вывода нет, заменилась вторая снизу ячейка):

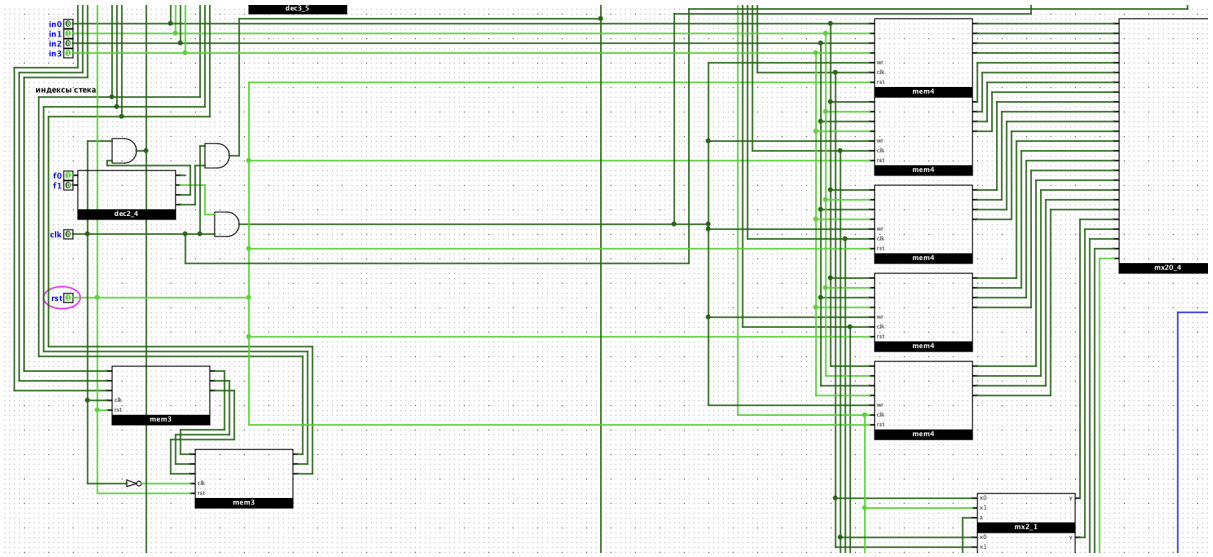


get 1000 (то есть минус один от текущей, вывелся предыдущий):
((вместо подачи индексы в моей схеме для get значение ячейки подается 4битным числом, вместо значения для push.))

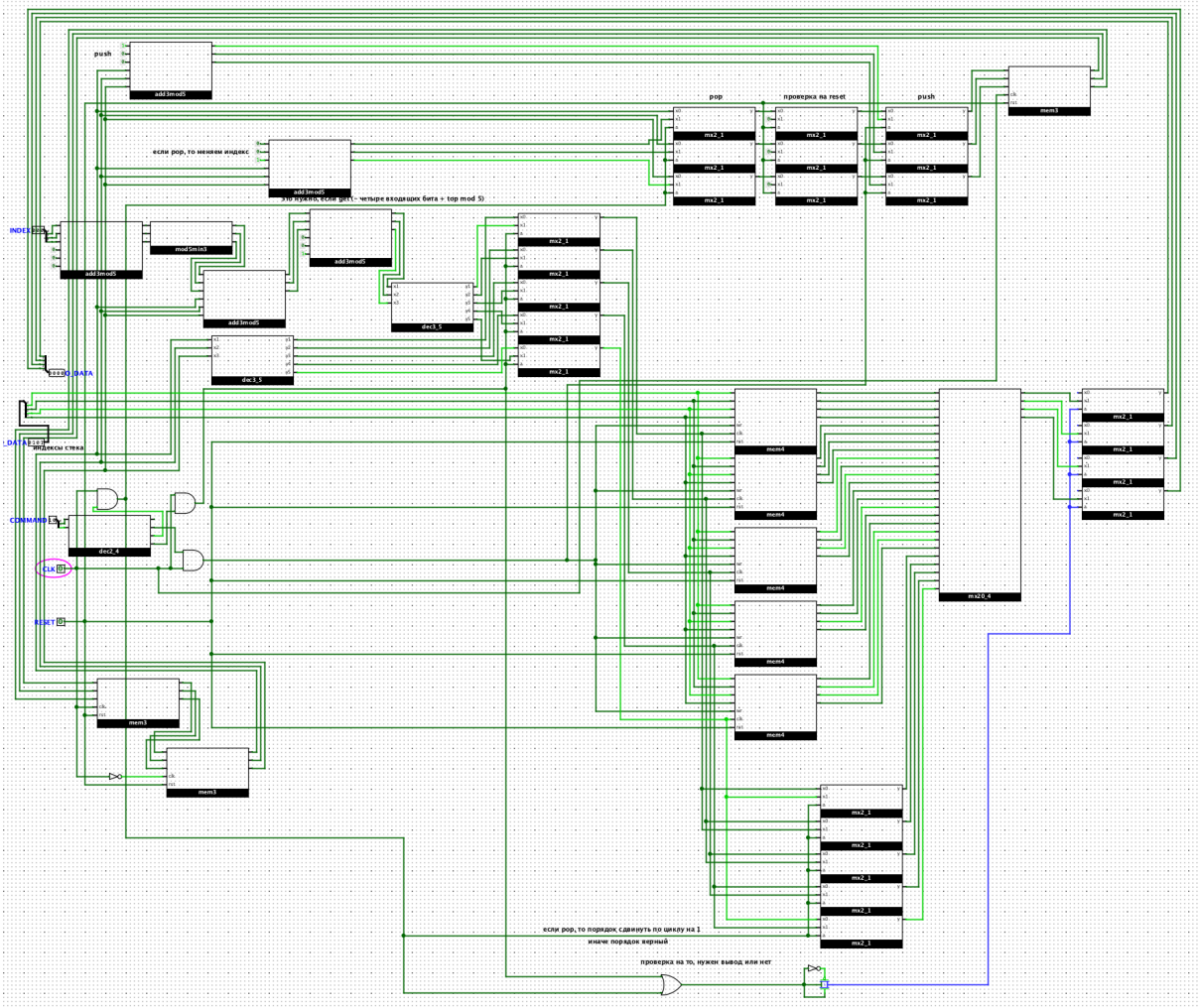




reset (все обнулилось, указатель на нижней ячейке):



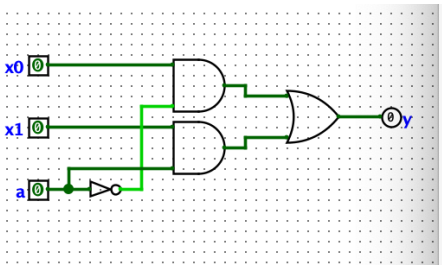
(сейчас схема выглядит немного по-другому, я заменила все входы, состоящие из четырех/трех/двух битов, на четырех/трех/двух-битовый один вход; также подвела выходы ко входам, чтобы было удобнее тестировать (и попытаться сделать входо-выход, но безуспешно..))



Verilog 1:

все функции названы и работают так же, как в схеме logisim.

(например, вот схема мультиплексора mx2_1 и тест для нее)




```
48  module mx2_1(output y, input x0, input x1, input a);
49      wire s1, s2;
50
51      not(b, a);
52      and(s1, x0, b);
53      and(s2, x1, a);
54      or(y, s1, s2);
55
56  endmodule
```

```
module mx2_1_tb_;
    reg x0, x1, a;
    wire y;
    mx2_1 w(y, x0, x1, a);
    initial begin
        x0 = 1; x1 = 0; a = 0;
        #1;
        $display(y);
    end
endmodule
```

```
1
```

Тесты такие же, как и в Verilog 2 (см. ниже).

Verilog 2:

Стек – массив длины 5.

Каждая операция меняет индекс (index) в соответствии с командой:

nop: skip

push: ячейке присваивается значение из I_DATA; index: если был 4, то станет 0, иначе +1 mod 5

pop: если index был 0, то станет 4, иначе -1 mod 5; выводится значение по получившемуся индексу

get: index каким был, таким и остается; если index == 0, то выводится ячейка с индексом 4 - INDEX mod 5, иначе выводится ячейка с индексом = (index - 1 < INDEX % 5) ? (index - 1 + 5 - INDEX % 5) : (index - 1 - INDEX % 5). Все условия нужны для того, чтобы избежать переполнения. Так, например, 4 + 5 mod 5 = 1, а на деле 4, или 4+4 получается 0, хотя по факту 3.

Сам код работает только в случае, если выдано либо clk, либо reset (для этого есть: always@(...)).

Тесты:

Вывод в формате:

CLK, RESET, COMMAND, INDEX, I_DATA, O_DATA

reset

0 1 00 000 0000 0000

push: 0001 и 0011

1 0 01 000 0001 0000

1 0 01 000 0011 0000

get по индексам от 0 до 7

1 0 11 000 0011 0011

1 0 11 001 0011 0001

1 0 11 010 0011 0000

1 0 11 011 0011 0000

1 0 11 100 0011 0000
1 0 11 101 0011 0011
1 0 11 110 0011 0001

push: 0111, 1111, 1110, 1100, 1000 (первые две ячейки перезапишутся)

1 0 01 110 0111 0000
1 0 01 110 1111 0000
1 0 01 110 1110 0000
1 0 01 110 1100 0000
1 0 01 110 1000 0000

pop по индексам от 0 до 10 (проверка перезаписи; это два цикла, вернемся в исходную)

1 0 10 110 1000 1000
1 0 10 110 1000 1100
1 0 10 110 1000 1110
1 0 10 110 1000 1111
1 0 10 110 1000 0111
1 0 10 110 1000 1000
1 0 10 110 1000 1100
1 0 10 110 1000 1110
1 0 10 110 1000 1111
1 0 10 110 1000 0111

push: 1001 и 1011

1 0 01 110 1001 0000
1 0 01 110 1011 0000

pop по индексам от 0 до 5

1 0 10 110 1011 1011
1 0 10 110 1011 1001
1 0 10 110 1011 1000
1 0 10 110 1011 1100
1 0 10 110 1011 1110

get по индексам от 0 до 7

1 0 11 000 1011 1011
1 0 11 001 1011 1001
1 0 11 010 1011 1000
1 0 11 011 1011 1100
1 0 11 100 1011 1110
1 0 11 101 1011 1011
1 0 11 110 1011 1001

reset

0 1 11 110 1011 0000

pop по индексам от 0 до 5

1 0 10 110 1011 0000

1 0 10 110 1011 0000
1 0 10 110 1011 0000
1 0 10 110 1011 0000
1 0 10 110 1011 0000

push: 0110

1 0 01 110 0110 0000

get по индексам от 0 до 7

1 0 11 000 0110 0110
1 0 11 001 0110 0000
1 0 11 010 0110 0000
1 0 11 011 0110 0000
1 0 11 100 0110 0000
1 0 11 101 0110 0110
1 0 11 110 0110 0000

Что изменилось?

В отчете: подробнее описала работу функций, добавила тесты для verilog

В logisim: исправила get (раньше он менял индекс вершины стека, сейчас – нет); улучшила проверку индекса по модулю 5 (при get); исправила внешний вид (видимо, в худшую сторону), изменила названия входов/выходов

В verilog: написала тесты (они одинаковые и для 2.1, и для 2.2). Написала структурную модель (с нуля) и исправила поведенческую модель.