Программируемые логические интегральные схемы

Лектор:

Шуплецов Михаил Сергеевич

e-mail:

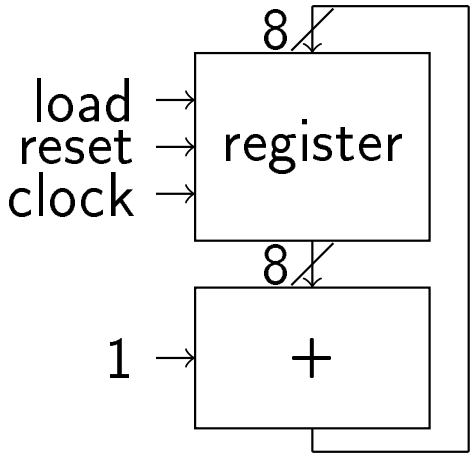
shupletsov@cs.msu.ru

Осень 2016

Лекция 3

Управляющие автоматы

Введение: зачем нужен управляющий автомат

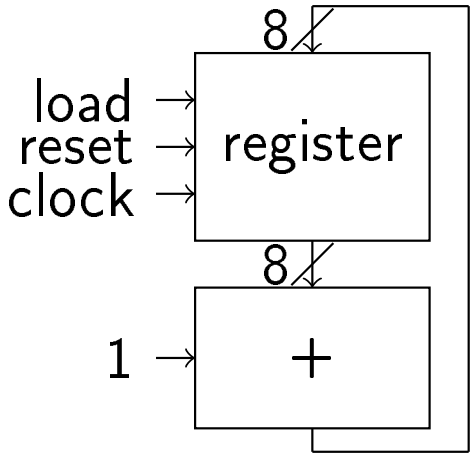


Эта схема определяет то, как преобразуются данные, то есть операционный автомат

В зависимости от того, что подаётся на входные сигналы `load`, `reset`, `clock`, данные, записанные в регистр, могут

- оставаться такими же, как и были
- увеличиваться на единицу
- сбрасываться в ноль

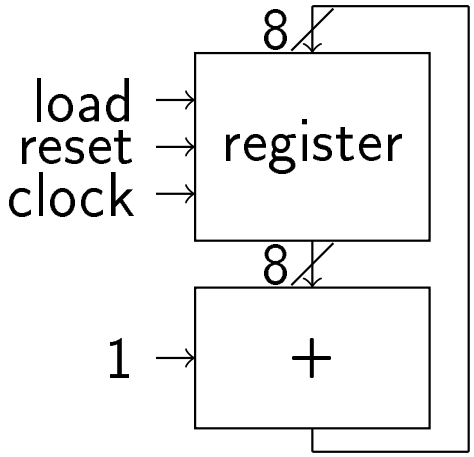
Введение: зачем нужен управляющий автомат



А как заставить регистр делать то, что мы хотим?

1. Подвести ко входам регистра имеющиеся элементы управления (в DE0-Nano — KEY[i], SW[i], CLOCK_50) и управлять регистром, нажимая на кнопки и щёлкая выключателями
2. **Заставить схему делать эту работу за нас**
   - что особенно полезно, если управляющих входов больше, чем элементов управления, а часто без этого в принципе не обойтись

Введение: зачем нужен управляющий автомат



Схема, которая выставляет за нас управляющие сигналы нужным образом в нужные моменты времени — это управляющий автомат

Простенькая задачка

А как разработать подходящий управляющий автомат?

Начнём с такой задачки: **регистр должен посчитать числа от ноля до двух и остановиться**

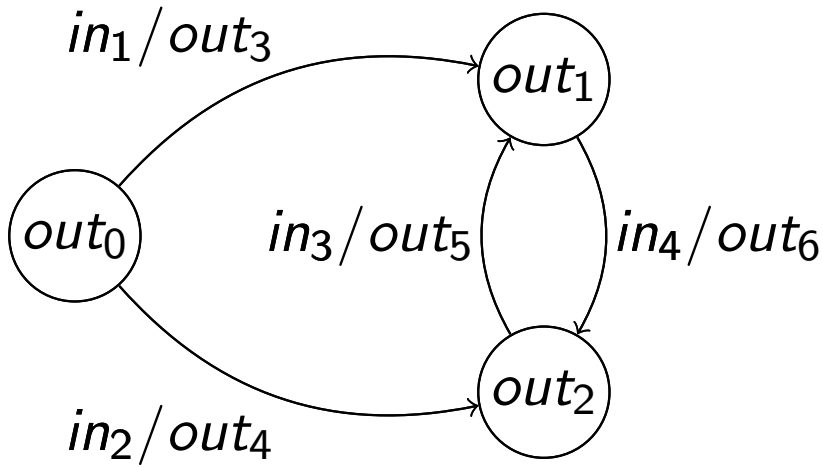
Можно пошагово расписать последовательность действий, которые должен сделать регистр, чтобы эту задачку решить:

1. записать в себя ноль
2. прибавить единицу
3. прибавить единицу
4. остановиться

Автоматы

Что такое автомат?

- У него есть конечное множество состояний
- Общаясь с внешней средой, он переходит из одного состояния в другое *в дискретном времени* (т.е. пошагово)
- В зависимости от текущего состояния, он выдаёт нечто на выход, то есть во внешнюю среду (**автомат Мура**)
- Совершая переход, он также способен выдавать нечто на выход (**автомат Мили**)



Автоматы

1. записать в себя ноль
2. прибавить единицу
3. прибавить единицу
4. остановиться

Попробуем записать этот алгоритм в автоматном виде

Откуда взять дискретное время?

Есть входной провод CLOCK_50, и можно дискретно отсчитывать моменты времени по передним фронтам приходящих от него сигналов

Автоматы

1. записать в себя ноль
2. прибавить единицу
3. прибавить единицу
4. остановиться

Попробуем записать этот алгоритм в автоматном виде

Откуда взять состояния?

Четыре пункта алгоритма — это, по большому счёту, четыре состояния:

- каждый пункт точно описывает, что автомат должен послать во внешнюю среду (*то есть в операционный автомат*)
- каждый пункт может быть сделан за один такт времени

Автоматы

1. записать в себя ноль
2. прибавить единицу
3. прибавить единицу
4. остановиться

Попробуем записать этот алгоритм в автоматном виде

Как соединить между собой эти состояния?

По цепочке от предыдущего к следующему, не обращая внимания на то, что происходит во внешней среде

Автоматы

1. записать в себя ноль
2. прибавить единицу
3. прибавить единицу
4. остановиться

Попробуем записать этот алгоритм в автоматном виде

Что когда выдавать на выход?

При переходе ничего не нужно делать (*в более сложных случаях может понадобиться, но не тут*)

В каждом состоянии достаточно выставить нужные сигналы на входах `load` и `reset` регистра:

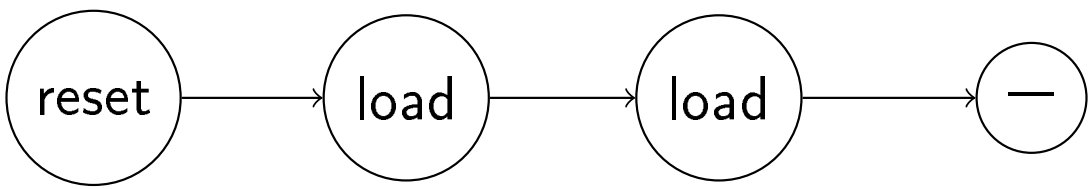
- выставляем `reset = 0` — регистр сбрасывается (*немедленно*)
- выставляем `load = 0` — значение в регистре увеличивается (*по переднему фронту CLOCK_50*)

Автоматы

1. записать в себя ноль
2. прибавить единицу
3. прибавить единицу
4. остановиться

Попробуем записать этот алгоритм в автоматном виде

Диаграмма автомата (диаграмма Мура?) автомата, описывающего алгоритм:

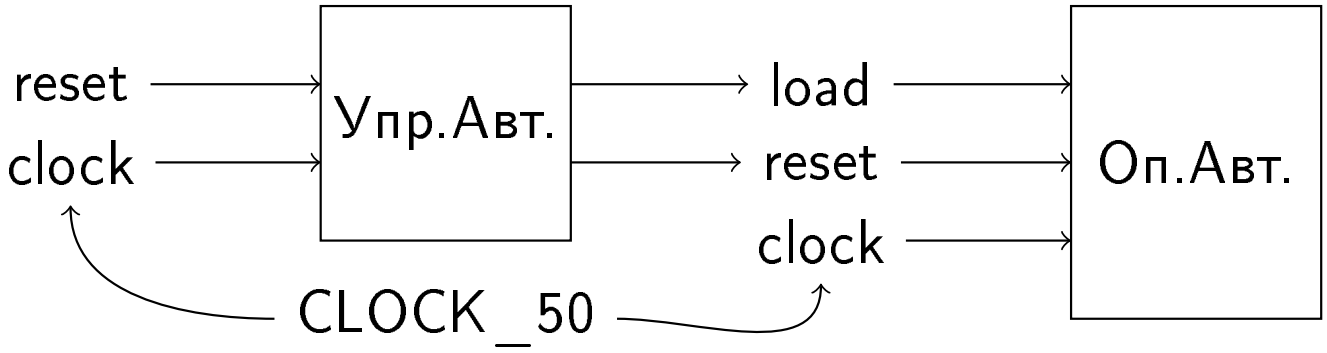


Взаимодействие операционного и управляющего автоматов

И как это всё будет выглядеть “в железе”?

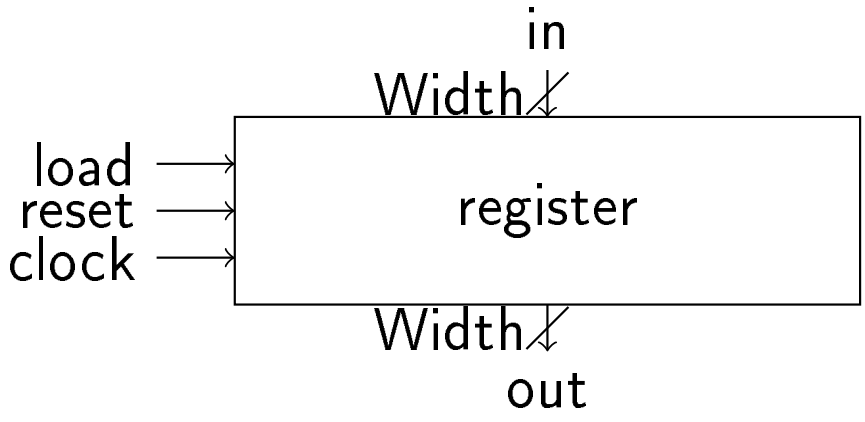
Чтобы всё заработало, достаточно заиметь и верно соединить:

- операционный автомат: коробочку, в которой всё работает однозначно, кроме сигналов `load`, `reset`, `clock`
- управляющий автомат:
  - для нормальной работы требуется *дискретное время* (clock) и инициализация (`reset`)
  - основное назначение — в нужное время в нужном порядке выставлять сигналы `load`, `reset`
- тактовый генератор CLOCK_50



Решение задачки

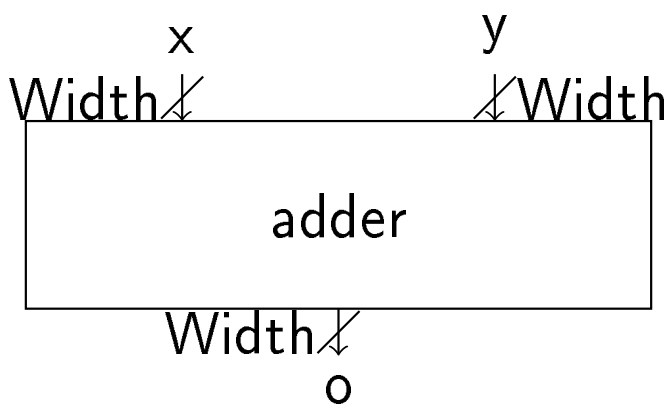
Регистр:




```
module register
  #(parameter Width = 8)
  ( input [Width-1:0] in,
    output reg [Width-1:0] out,
   input load, reset, clock
  );
  always @(posedge clock, negedge reset)
    if(~reset) out <= 0;
   else if(~load) out <= in;
endmodule
```

Решение задачки

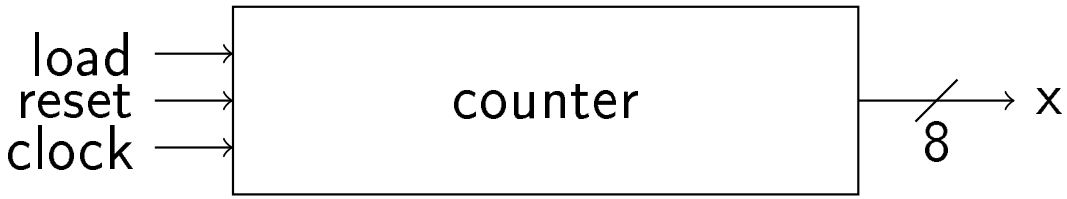
Сумматор:




```
module adder
  #(parameter Width = 8)
  ( input [Width-1:0] x,
    input [Width-1:0] y,
    output [Width-1:0] o
  );
  assign o = x + y;
endmodule
```

Решение задачки

Операционный автомат (с выводом значения регистра в LED):




```
module counter(input load, reset, clock,
               output [7:0] x);
  parameter Width = 8;
  wire [Width-1:0] in, out;

  register r(.in(in), .out(out), .load(load),
             .reset(reset), .clock(clock));
  adder a(.x(out), .y(8'b00000001), .o(in));

  assign x = out;
endmodule
```

Решение задачки

Управляющий автомат:


```
module fsm(input clock, reset,
           output reg load, o_reset);
  reg [1:0] c_state, n_state;

  always @(c_state)
    case(c_state)
     2'b00:
    begin
      load = 1;
      o_reset = 0;
      n_state = 2'b01;
    end
```

Решение задачки

Управляющий автомат:


```
 2'b01:
begin
   load = 0;
   o_reset = 1;
   n_state = 2'b10;
end
 2'b10:
begin
   load = 0;
   o_reset = 1;
   n_state = 2'b11;
end
```

Решение задачки

Управляющий автомат:


```
      2'b11:
     begin
       load = 1;
       o_reset = 1;
       n_state = 2'b11;
     end
   endcase

  always @(posedge clock, negedge reset)
    if(~reset) c_state <= 0;
   else c_state <= n_state;
endmodule
```

Решение задачки

Главный модуль (reset выведен на KEY[1], и мы, нажимая на KEY[0], генерируем тактовые импульсы):

```
module top( SW, KEY, LED, CLOCK_50);
  input wire [3:0] SW;
  input wire [1:0] KEY;
  output [7:0] LED;
  input wire CLOCK_50;

  wire load, reset;
  counter op_aut(.load(load), .reset(reset),
                 .clock(KEY[0]), .x(LED));
  fsm c_aut(.clock(KEY[0]), .reset(KEY[1]),
            .load(load), .o_reset(reset));
endmodule
```

А теперь задачка посложнее

Хочу, чтобы счётчик работал так:

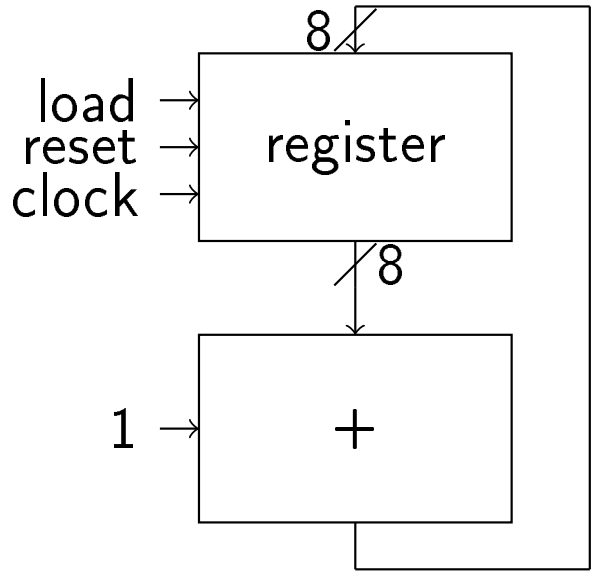
- выключателями SW составляю двоичную четырёхбитную запись числа
- кнопкой KEY[1] запускаю алгоритм
- счётчик отсчитывает с ноля до составленного числа, прибавляет единицу и останавливается

В чём здесь сложности?

1. Диаграмма Мура нелинейна (*есть циклы*)
2. Управляющий автомат, чтобы знать, что делать, должен анализировать информацию из внешнего мира
3. Передавать **данные** в управляющий автомат — плохо (*управляющий автомат должен **управлять**, а не вычислять*)
4. Значит, нужно добавить в операционный автомат схему, работающую с данными (*проверяющую, досчитал ли регистр до конца*) и передающую результат работы в управляющий автомат

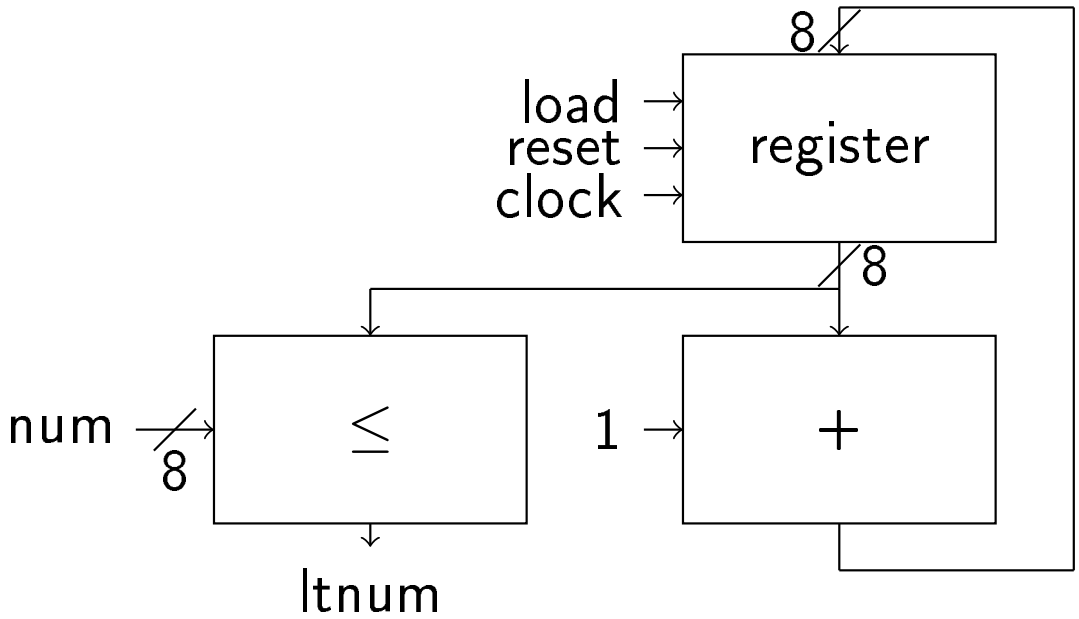
Как изменится операционный автомат

Было:



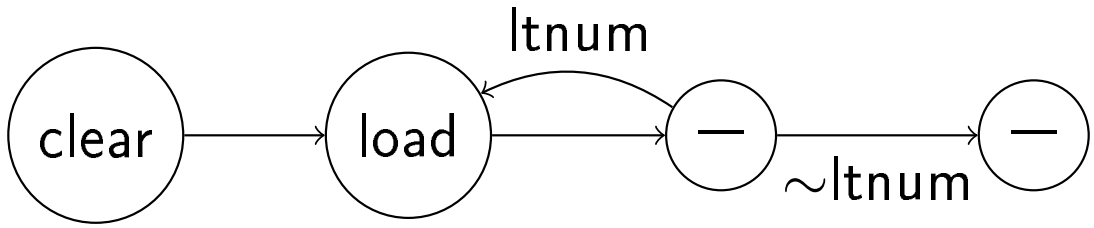
Как изменится операционный автомат

Стало:



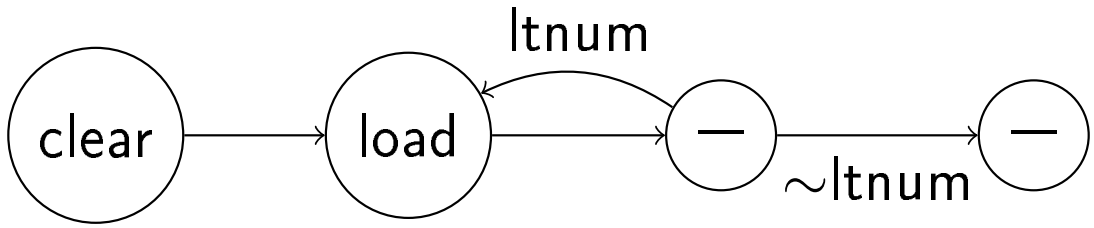
- Добавился блок сравнения
- Добавилась входная шина `num`
- Добавился выходной сигнал `ltnum` — он будет пересылаться управляющему автомату

Как будет выглядеть управляющий автомат



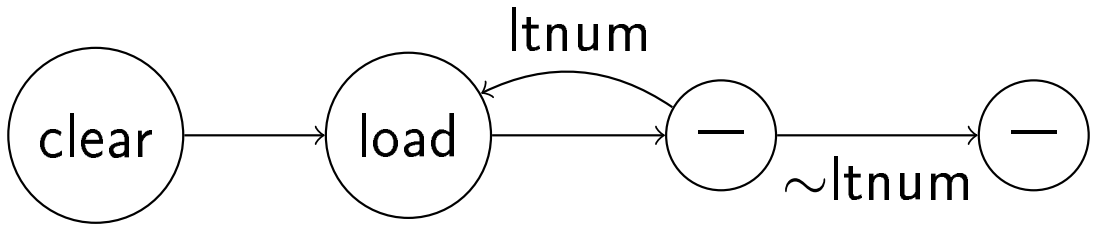

```
module fsm(input clock, reset, ltnum,
           output reg load, o_reset);
...
```

Как будет выглядеть управляющий автомат



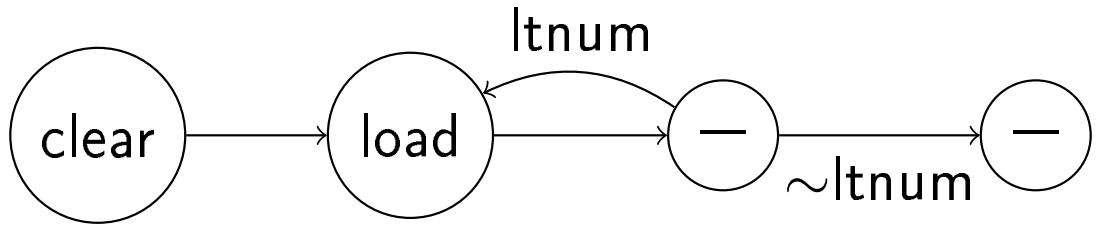

```
...
    2'b00:
    begin
      o_reset = 0;
      load = 1;
      n_state = 2'b01;
    end
...
```

Как будет выглядеть управляющий автомат



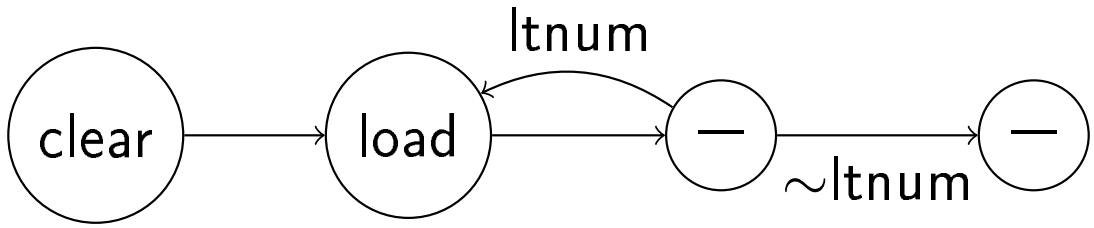

```
...
    2'b01:
    begin
      o_reset = 1;
      load = 0;
      n_state = 2'b10;
    end
...
```

Как будет выглядеть управляющий автомат



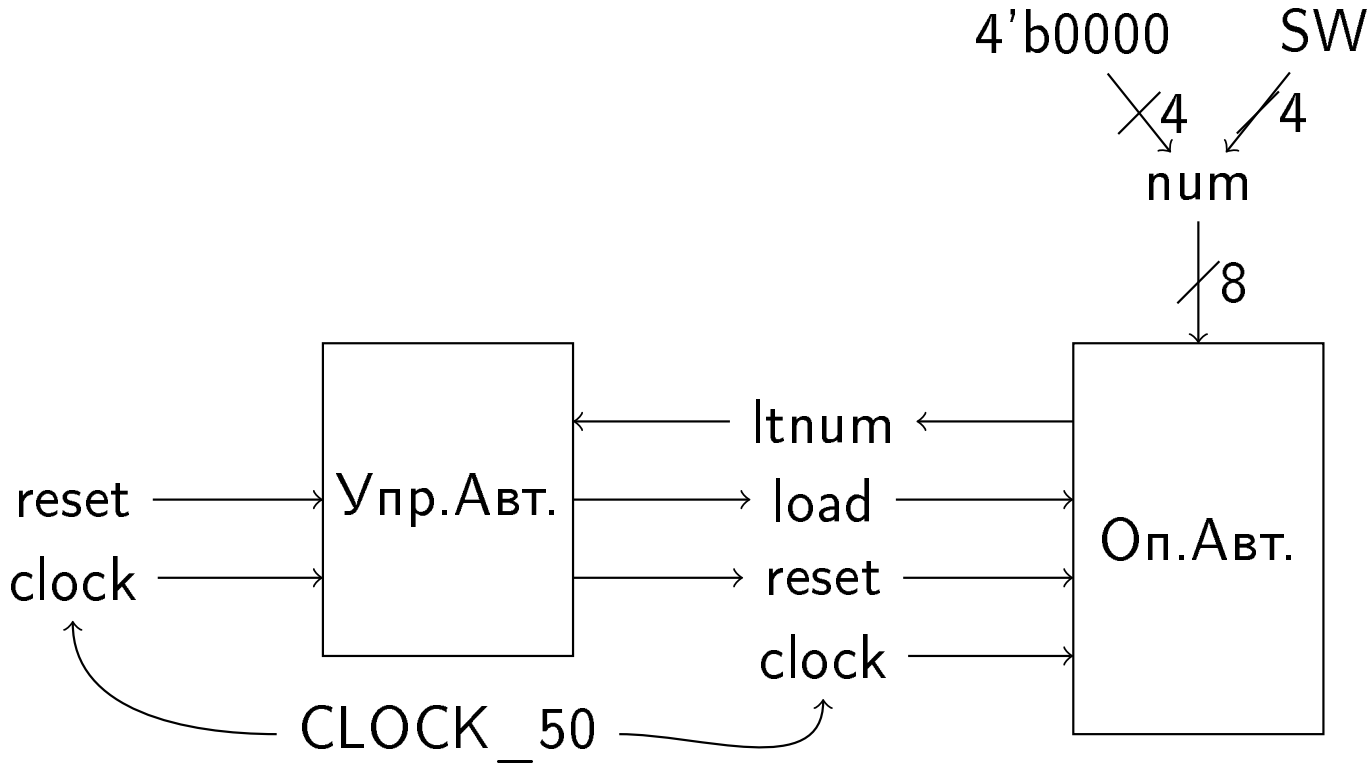

```
...
    2'b10:
    begin
      o_reset = 1;
      load = 1;
      if(ltnum) n_state = 2'b01;
      else n_state = 2'b11;
    end
...
```

Как будет выглядеть управляющий автомат




```
...
    2'b11:
    begin
      o_reset = 1;
      load = 1;
      n_state = 2'b11;
    end
...
```

Как будет выглядеть взаимодействие управляющего и операционного автоматов



А остальную часть решения додумайте сами

Конец лекции 3