# Языки описания схем

(mk.cs.msu.ru → Лекционные курсы → Языки описания схем)

Блок 19
Verilog:
реализация управляющего автомата

лектор:
Подымов Владислав Васильевич

e-mail:
**valdus@yandex.ru**

Осень 2018

# Вступление

Управляющий автомат — это *в конечном итоге* последовательная схема, как и операционный автомат

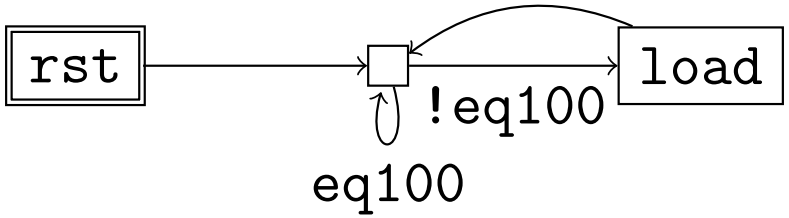
Реализация управляющего автомата — это **любая** реализация соответствующей последовательной схемы

При этом управляющий автомат **как математический объект** (*блоки 14, 17*) легко транслируется в последовательную схему, содержащую:

1. **один** параллельный регистр со сбросом, ширина $w$ которого определяется количеством состояний $Q$: $w = \lceil \log_2 |Q| \rceil$
2. комбинационную схему, порождаемую функцией переходов
3. комбинационную схему, порождаемую функцией выхода

Далее приведены несколько "классических" способов описания этих компонентов на языке `Verilog`, позволяющих меньше ошибаться и дающих возможность средству синтеза извлечь диаграмму Мура из схемы

# Сквозной пример




Это диаграмма Мура управляющего автомата для “сложного” счётчика $S_2$ из *блока 17*

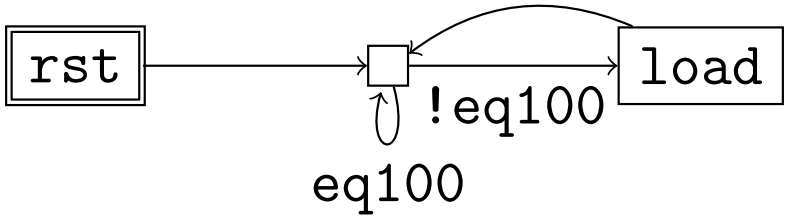
Входы:

- `clk`: тактовый
- `reset`: асинхронный сброс
- `eq100`: “число в операционном автомате равно 100”

Выходы:

- `rst`: “обнулить число в операционном автомате”
- `load`: “прибавить 1 к числу в операционном автомате”

# (V) Сквозной пример



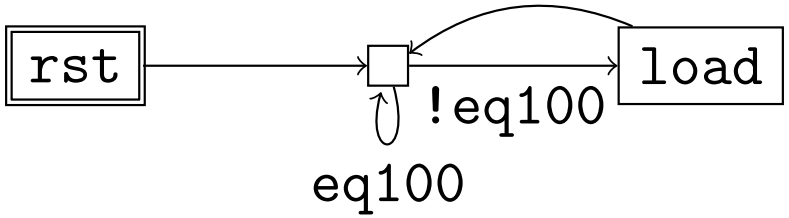

Начало всех “классических” реализаций автомата одинаковое:

**название модуля и описание портов**

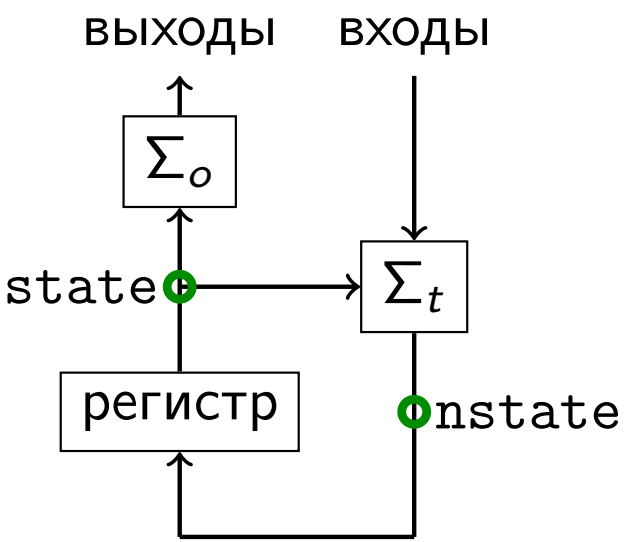
Например:

```
module fsm(input  clk, reset, eq100,
           output rst, load);
  // тело модуля
endmodule
```

# (V) Управляющий автомат, вариант 1

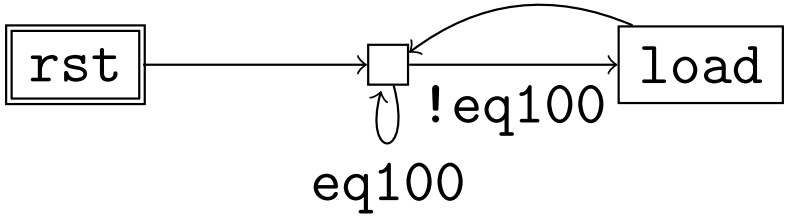



Тело модуля:




$\Sigma_o$ — комбинационная схема для функции выхода
$\Sigma_t$ — комбинационная схема для функции переходов
`state` — текущее состояние (текущее значение регистра)
`nstate` — новое состояние (новое значение регистра)

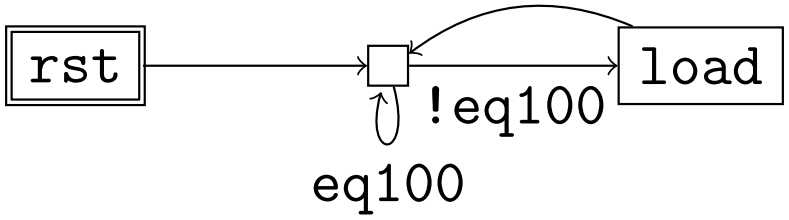
# (V) Управляющий автомат, вариант 1




Регистр автомата:

```
reg [1:0] state, nstate;
always @(posedge clk, posedge reset)
  if(reset) state <= 0;
  else      state <= nstate;
```

# (V) Управляющий автомат, вариант 1

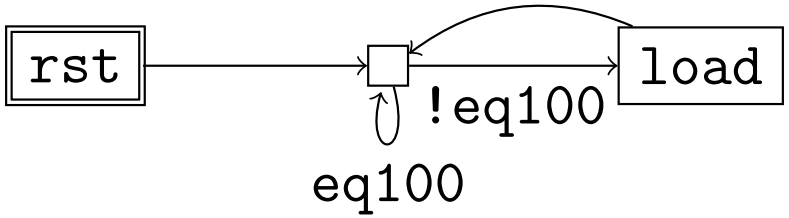



Описание $\Sigma_t$ — произвольное, например:

```
always @(*)
  case(state)
  0: nstate = 1;
  1: if(eq100) nstate = 1;
     else      nstate = 2;
  2: nstate = 1;
  default: nstate = 0;
  endcase
```

```
always @(*)
begin
  nstate = 1;
  if(state == 1
     && !eq100)
    nstate = 2;
end
```

# (V) Управляющий автомат, вариант 1

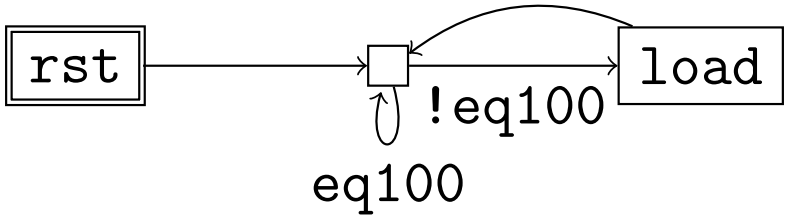



Описание $\Sigma_o$ — произвольное, например:

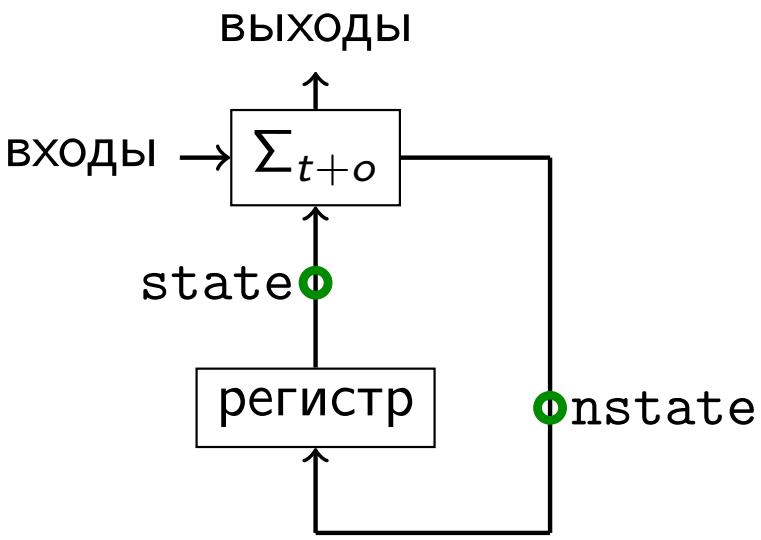
```
reg rst, load;
always @(*)
begin
  rst = 0;
  load = 0;
  case(state)
  0: rst = 1;
  2: load = 1;
  endcase;
end
```

```
assign rst =
  (state == 0);
assign load =
  (state == 2);
```

# (V) Управляющий автомат, вариант 2




Тело модуля:

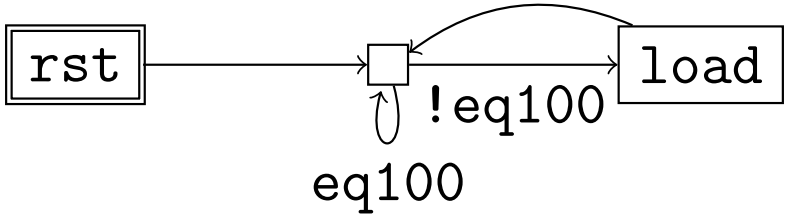



$\Sigma_{t+o}$ — комбинационная схема, реализующая
и функцию переходов, и функцию выхода

`state` — текущее состояние (текущее значение регистра)

`nstate` — новое состояние (новое значение регистра)

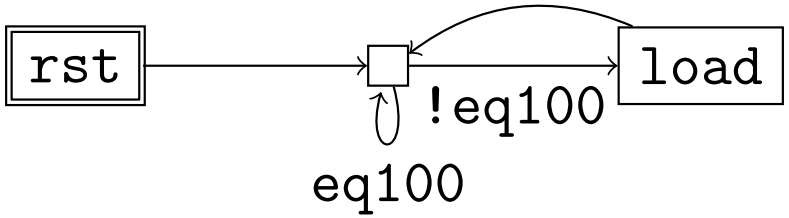
# (V) Управляющий автомат, вариант 2




Описание $\Sigma_{t+o}$ может выглядеть, например, так:

```
reg rst, load;
always @(*)
begin
  rst = 0;
  load = 0;
  nstate = 1;
  case(state)
  0: rst = 1;
  1: if(!eq100) nstate = 2;
  2: load = 1;
  endcase;
end
```

# (V) Управляющий автомат, варианты 1*, 2*

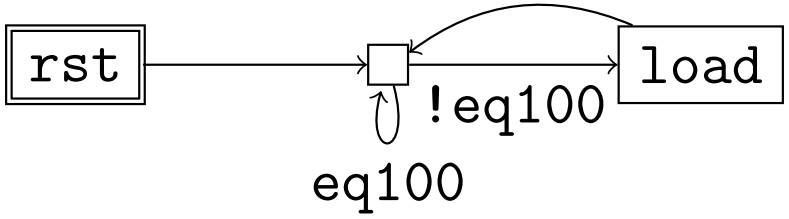



Как бы ни проектировался код автомата,
можно повысить его наглядность и избежать опечаток
использованием “говорящих” названий состояний

Например: (*в теле модуля*)

```
localparam integer
  stRst = 0,
  stCheck = 1,
  stLoad = 2;
```

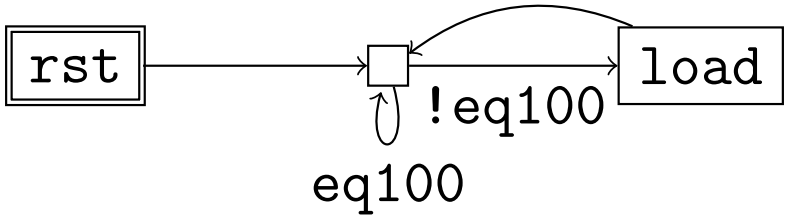
# (V) Управляющий автомат, варианты 1*, 2*




Регистр автомата с учётом имён состояний:

```
reg [1:0] state, nstate;
always @(posedge clk, posedge reset)
  if(reset) state <= stRst;
  else      state <= nstate;
```

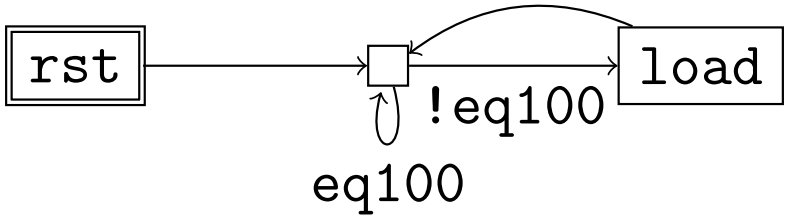
# (V) Управляющий автомат, варианты 1*, 2*




Возможное описание $\Sigma_t$ с учётом имён состояний:

```
always @(*)
begin
  nstate = stCheck;
  if(nstate == stCheck && !eq100)
    nstate = stLoad;
end
```

# (V) Управляющий автомат, варианты 1*, 2*




Возможное описание $\Sigma_o$ с учётом имён состояний:

```
reg rst, load;
always @(*)
begin
  rst = 0;
  load = 0;
  case(state)
  stRst:  rst = 1;
  stLoad: load = 1;
  endcase;
end
```

```
assign rst =
  (state == stRst);
assign load =
  (state == stLoad);
```