

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 2017

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

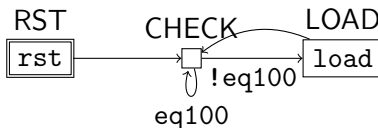
Вступление

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

“Классическая” реализация управляющего автомата

- ▶ содержит эти в точности эти три компонента
- ▶ позволяет пользователю разрабатывать схему автомата быстрее и безошибочнее
- ▶ позволяет средству синтеза восстанавливать диаграмму Мура по схеме

Сквозной пример



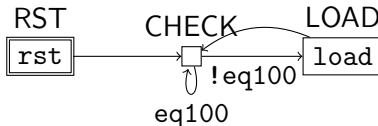
Это диаграмма Мура управляющего автомата для “сложного” счётчика S_2 из *блока 17*

Этот автомат содержит

- ▶ тактовый вход `clk`
- ▶ вход асинхронного сброса `reset`
- ▶ однобитовый управляющий вход `eq100`
- ▶ однобитовые управляющие выходы `rst` и `load`

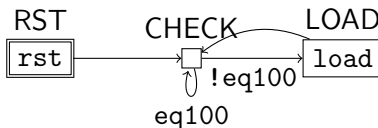
Далее будут приведены несколько вариантов “классических” реализаций предложенной диаграммы Мура с пояснениями, как это выглядит в общем случае

(V) Управляющий автомат, вариант 1



```
module fsm(input clk, reset, eq100,
           output reg rst, load);
    // Шины для текущего и следующего состояний
    // Ширина - достаточная для кодирования состояний
    reg [1:0] state, next_state;
    // 0 - начальное состояние,
    // по сбросу переходим в него,
    // а иначе - в следующее состояние
    always @(posedge clk, posedge reset)
        if(reset) state <= 0;
        else state <= next_state;
```

(V) Управляющий автомат, вариант 1



// Комбинационная схема - функция переходов

```
always @(*)
```

```
    case(state)
```

```
    0: next_state = 1;
```

```
    1: if(eq100) next_state = 1; else next_state = 2;
```

```
    2: next_state = 1;
```

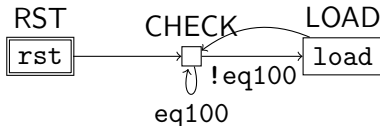
```
    default: next_state = 0;
```

```
    endcase;
```

```
    // Ветка default - фиктивная, и нужна только для того,
```

```
    // чтобы сказать "это не набор защёлок"
```

(V) Управляющий автомат, вариант 1



// Комбинационная схема - функция выхода

```
always @(*) begin
```

```
    rst = 0; load = 0;
```

```
    case(state)
```

```
    0: rst = 1;
```

```
    2: load = 1;
```

```
    endcase;
```

```
end
```

// Второй удобный способ сказать

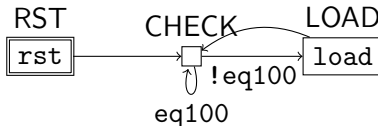
// “это не набор защёлок” - присвоить всем выходам

// значения по умолчанию, и дальше явно присваивать

// только значения не по умолчанию

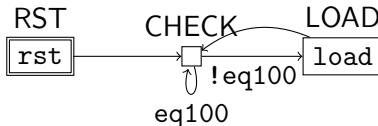
```
endmodule
```

(V) Управляющий автомат, вариант 2



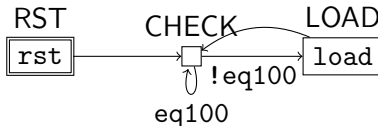
```
module fsm(input clk, reset, eq100,  
           output reg rst, load);  
    // Читаемость можно повысить,  
    // если использовать локальные параметры  
    localparam S_RST = 0;  
    localparam S_CHECK = 1;  
    localparam S_LOAD = 2;  
    reg [1:0] state, next_state;  
    always @(posedge clk, posedge reset)  
        if(reset) state <= S_RST;  
        else state <= next_state;
```


(V) Управляющий автомат, вариант 2



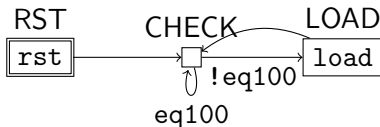
```
// Комбинационная схема - функция переходов
always @(*) begin
    next_state = S_CHECK;
    if(state == S_CHECK && !eq100)
        next_state = S_LOAD;
    // писать case-оператор - это удобочитаемо,
    // но иногда бывает громоздко
end
```

(V) Управляющий автомат, вариант 2



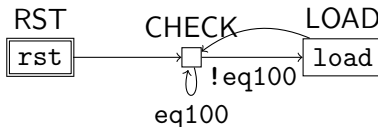
```
// Комбинационная схема - функция выхода
always @(*) begin
    rst = 0; load = 0;
    case(state)
        S_RST: rst = 1;
        S_LOAD: load = 1;
    endcase;
end
endmodule
```

(V) Управляющий автомат, вариант 3



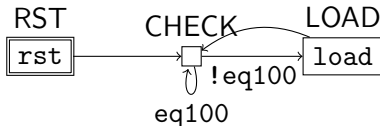
```
module fsm(input clk, reset, eq100,  
           output reg rst, load);  
    localparam S_RST = 0;  
    localparam S_CHECK = 1;  
    localparam S_LOAD = 2;  
    reg [1:0] state, next_state;  
    always @(posedge clk, posedge reset)  
        if(reset) state <= S_RST;  
        else state <= next_state;
```

(V) Управляющий автомат, вариант 3



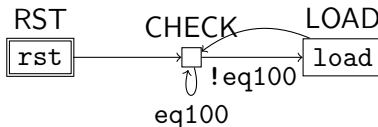
```
// Комбинационная схема - функции переходов
// и выхода вместе
always @(*) begin
    next_state = S_CHECK;
    rst = 0; load = 0;
    case(state)
        S_RST: rst = 1;
        S_CHECK: if(!eq100) next_state = S_/LOAD;
        S_LOAD: load = 1;
    endcase;
end
endmodule
```

(V) Управляющий автомат, вариант 4



```
module fsm(input clk, reset, eq100,
           output reg rst, load);
    localparam S_RST = 0;
    localparam S_CHECK = 1;
    localparam S_LOAD = 2;
    // next_state можно и не заводить
    reg [1:0] state;
    // тогда смена состояний "сливается" в один блок
    always @(posedge clk, posedge reset)
        if(reset) state <= S_RST;
        else
            if(state == S_CHECK && !eq100) state <= S_LOAD;
            else state <= S_CHECK;
```

(V) Управляющий автомат, вариант 4



```
// Комбинационная схема - функция выхода
assign rst = state == S_RST;
assign load = state == S_LOAD;
// Никто не говорил, что always-блок обязателен
endmodule
```