

Языки описания схем

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

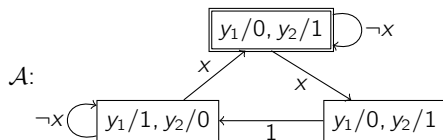
Блок 28

Verilog:
типовая реализация автомата

Лектор:
Подымов Владислав Васильевич
E-mail:
valdus@yandex.ru

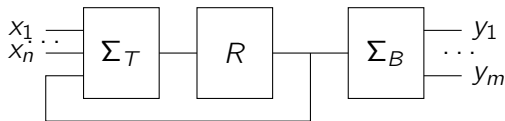
ВМК МГУ, 2023/2024, осенний семестр

Вступление



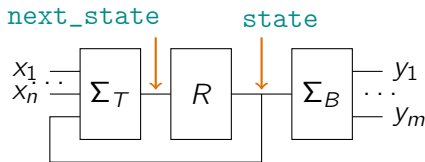
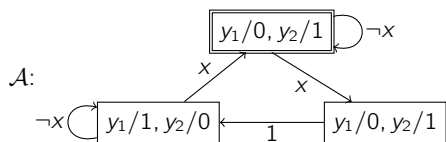
Типичная схемная реализация автомата содержит:

- ▶ параллельный регистр R , хранящий состояние автомата
- ▶ комбинационную схему Σ_B , реализующую функцию выхода
- ▶ комбинационную схему Σ_T , реализующую функцию переходов



Обсудим то, как выглядит такая типичная реализация на языке Verilog

Автомат $\rightarrow \mathcal{V}$: состояния



Значение на выходе R в каждый момент времени — текущее состояние

Значение на входе R в каждый момент времени — следующее состояние:
по переднему фронту тактового сигнала это состояние сохраняется в регистре и становится текущим

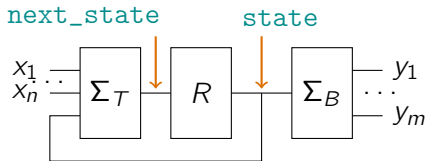
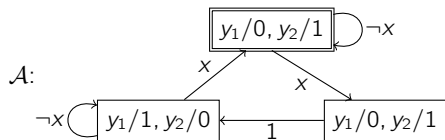
Объявим соответствующие точки `state` и `next_state`

Например, для \mathcal{A} наименьшая ширина этих точек — $\lceil \log_2 3 \rceil = 2$:

```
reg [1:0] state, next_state;
```

(next_state может быть и соединением — смотря как реализована схема Σ_T)

Автомат $\rightarrow \mathcal{V}$: состояния



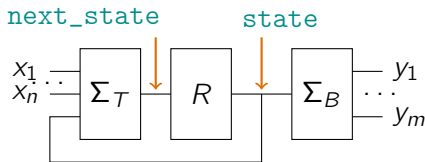
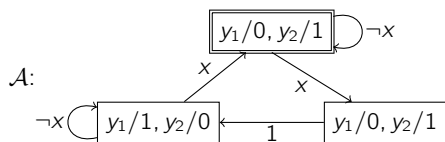
Чтобы избежать опечаток, связанных с нумерацией состояний, присвоим каждому состоянию автомата уникальное «наглядное» имя. Это имя не имеет смысла для пользователя и должно быть недоступно для изменения им, и для задания таких имён хорошо подходят **локальные параметры**:

```
localparam S_TOP = 0,
           S_RIGHT = 1,
           S_LEFT = 2;
```

Реализуем регистр R типовым способом:

```
always @(posedge clk, posedge rst)
    if(rst) state <= S_TOP;
    else state <= next_state;
```

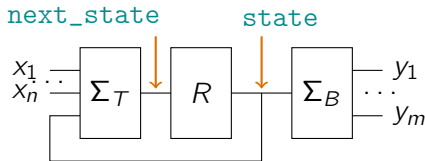
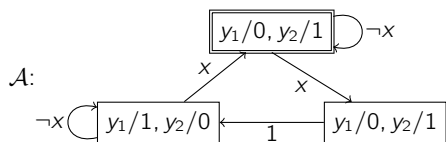
Автомат $\rightarrow \mathcal{V}$: переходы



Реализуем функцию переходов (Σ_T) в отдельной процедуре:

```
always @* begin
    next_state = 1'bx; // Все случаи, не перечисленные дальше, неважны
    case(state) // Для каждого состояния перечислим исходящие из него дуги
        // Дуги символического автомата над подходящими предикатами
        // легко переписываются как код
    S_TOP: if(x) next_state = S_RIGHT;
           else next_state = S_TOP;
    S_RIGHT: next_state = S_LEFT;
    S_LEFT: if(x) next_state = S_TOP;
            else next_state = S_LEFT;
    endcase
end
```

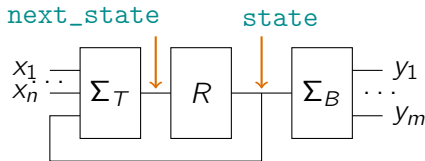
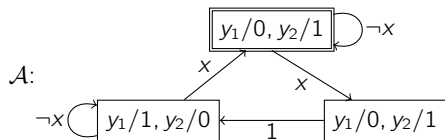
Автомат $\rightarrow \mathcal{V}$: выход



Реализуем функцию выхода (Σ_B) в отдельной процедуре:

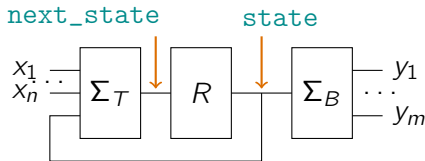
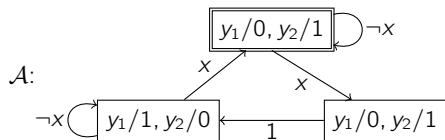
```
always @* begin
    y1 = 1'bx; y2 = 1'bx; // Все случаи, не перечисленные дальше, неважны
    case(state) // Реализуем функцию выхода таблично
        S_TOP, S_RIGHT: begin
            y1 = 0;
            y2 = 1;
        end
        S_LEFT: begin
            y1 = 1;
            y2 = 0;
        end
    endcase
end
```

Автомат $\rightarrow \mathcal{V}$: всё вместе



```
reg [1:0] state, next_state;
localparam S_TOP = 0, S_RIGHT = 1, S_LEFT = 2;
always @(posedge clk, posedge rst)
    if(rst) state <= S_TOP;
    else state <= next_state;
always @* begin
    next_state = 1'bx;
    case(state)
        S_TOP: if(x) next_state = S_RIGHT; else next_state = S_TOP;
        S_RIGHT: next_state = S_LEFT;
        S_LEFT: if(x) next_state = S_TOP; else next_state = S_LEFT;
    endcase
end
always @* begin
    y1 = 1'bx; y2 = 1'bx;
    case(state)
        S_TOP, S_RIGHT: begin y1 = 0; y2 = 1; end
        S_LEFT: begin y1 = 1; y2 = 0; end
    endcase
end
```

Автомат $\rightarrow \mathcal{V}$: другие варианты (переходы)

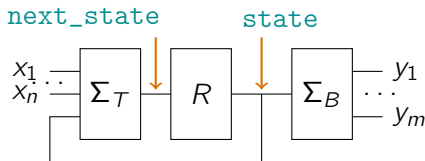
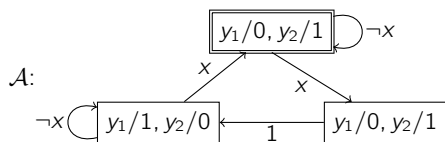


Не всегда бывает удобно начинать процедуру Σ_T с присваиваний «по умолчанию значение неважно»

Например, если в автомате много петель:

```
always @* begin
    next_state = state; // По умолчанию не изменяем состояние
    case(state) // Перечисляем все случаи изменения состояния
    S_TOP: if(x) next_state = S_RIGHT;
    S_RIGHT: next_state = S_LEFT;
    S_LEFT: if(x) next_state = S_TOP;
    endcase
end
```

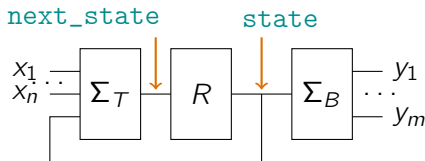
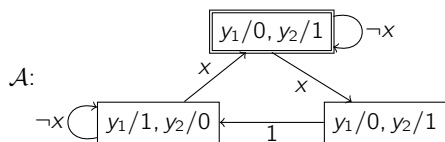

Автомат $\rightarrow \mathcal{V}$: другие варианты (переходы)



Другой пример — если в автомате много переходов в заданное состояние:

```
always @* begin
    next_state = S_TOP; // По умолчанию переходим в выделенное состояние
    case(state) // Перечисляем все случаи переходов в другие состояния
    S_TOP: if(x) next_state = S_RIGHT;
    S_RIGHT: next_state = S_LEFT;
    S_LEFT: if(!x) next_state = S_LEFT;
    endcase
end
```

Автомат $\rightarrow \mathcal{V}$: другие варианты (выходы)

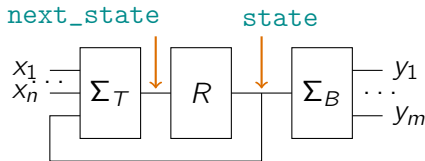
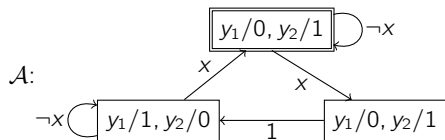


Не всегда бывает удобно начинать процедуру Σ_B с присваиваний «по умолчанию выходы произвольны»

Например, если в автомате есть «преобладающие» выходы:

```
always @* begin
    y1 = 0; y2 = 1; // Преобладающие выходные значения
    if(state == S_LEFT) begin // Редкие выходные значения
        y1 = 1;
        y2 = 0;
    end
end
```

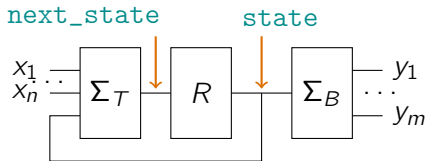
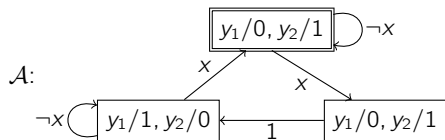
Автомат $\rightarrow \mathcal{V}$: другие варианты (выходы)



Другой пример: если разнообразие выходных значений невелико, то комбинационные выражения могут оказаться нагляднее процедур:

```
assign y1 = (state == S_LEFT);
assign y2 = !y1;
```

Автомат $\rightarrow \mathcal{V}$: другие варианты (всё вместе)



Если кажется, что схемы Σ_T , Σ_B в двух разных процедурах ненаглядны, то ничто не запрещает совместить эти процедуры (*не переусердствуя*):

```
always @* begin
    next_state = state; y1 = 0; y2 = 1;
    case(state)
    S_TOP: if(x) next_state = S_RIGHT;
    S_RIGHT: next_state = S_LEFT;
    S_LEFT: begin
        if(x) next_state = S_TOP;
        y1 = 1;
        y2 = 0;
    end
    endcase
end
```

(«Наглядность» — субъективное понятие, лишь бы только было проще избежать глупых ошибок)