

Языки описания схем

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

Блок 23

Verilog:

Поддерживаемое использование
постоянной процедуры

Лектор:

Подымов Владислав Васильевич

E-mail:

valdus@yandex.ru

ВМК МГУ, 2023/2024, осенний семестр

Вступление

Если *содержательно* представить себе поведение известных **триггеров** и **регистров**, то оно будет выглядеть примерно так:

всегда, когда <что-то> **происходит,**
значения на выходах <как-то> **изменяются**

Синхронный триггер/регистр:

- ▶ <что-то>: передний фронт тактового сигнала
- ▶ <как-то>: как записано в таблице значений

Асинхронный триггер/регистр:

- ▶ <что-то>: изменение значений на входах
- ▶ <как-то>: сохраняются или изменяются согласно таблице значений

Такое описание поведения асинхронных регистров подходит и для **комбинационных схем**

∪: поддержка значения \mathcal{X} в процедурах

Явное использование

значения \mathcal{X} в коде **поддерживается** только в следующих случаях:

1. В правой части присваиваний,

если к этому значению не применяются никакие операции

Можно

```
x <= 2'bx0;  
assign x = 2'bx0;
```

Нельзя

```
x <= 2'bx0 + 1;  
assign x = 2'bx0 + 1;
```

2. В выражениях случаев caseх

Можно

```
caseх(x)  
  2'b00: y = 0;  
  2'bx0: y = 1;  
  default: y = 1'bx;  
endcase
```

Нельзя

```
case(x)  
  2'b00: y = 0;  
  2'bx0: y = 1;  
  default: y = 1'bx;  
endcase
```

Все другие виды **явного** использования значения \mathcal{X} являются **неподдерживаемыми**

∪: реагирующая процедура

всегда, когда <что-то> происходит,
значения на выходах <как-то> изменяются

В ∪ в **программной семантике**:

- ▶ «всегда» = «процедура always»
- ▶ «<что-то> происходит» = «выполняется событие (<что-то>)»
- ▶ «значение изменяется» =
«блокирующее/неблокирующее присваивание»¹

Процесс, имеющий такое поведение, записывается следующим образом:²

always @(<список событий>) <команда>

Для ясности будем называть такой процесс **реагирующей процедурой**

¹ Есть и другие виды присваиваний, но можно обойтись и без них

² Более точно, <список событий> относится не к процессу, а к <команде> — но в ∪ **поддерживается** только такая форма записи, на ней и остановимся

∪: реагирующая процедура

always @(<список событий>) <команда>

Поведение: <команда> выполняется после каждого выполнения любого <события> из <списка>

Основные виды <событий>:

- ▶ <имя точки>: изменение значения в точке
- ▶ posedge <имя одноразрядной точки>: положительный фронт в одноразрядной точке
- ▶ negedge <имя одноразрядной точки>: отрицательный фронт в одноразрядной точке

Внимание! Положительными фронтами в ∪ также считаются изменения значений $0 \rightarrow \mathcal{X}$ и $\mathcal{X} \rightarrow 1$, и отрицательными — $1 \rightarrow \mathcal{X}$ и $\mathcal{X} \rightarrow 0$

<События> в <списке> разделяются запятой или словом `or`

∪: always и синхронные регистры

Пример: поддерживаемое описание D-триггера

```
reg q;  
always @(posedge clk) q <= d;
```

Общие требования к поддерживаемому описанию синхронного регистра (как реагирующей процедуры)¹

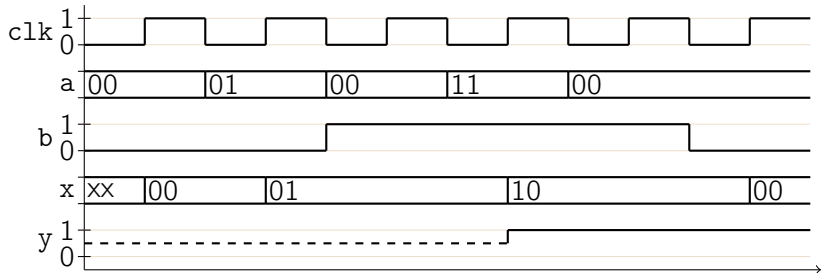
1. Список событий: **ровно одно** событие, и это событие фронта
2. Все присваивания процедуры **неблокирующие**
3. Выходы регистра: все переменные в левых частях присваиваний
4. Входы регистра:
 - ▶ тактовый: указанный в списке событий
 - ▶ остальные: точки, значения которых используются в процедуре

¹ Также необходимо соблюдать все требования поддержки отдельных процедурных команд

⋃: always и синхронные регистры

Чуть более нетривиальный пример:

```
reg [1:0] x;  
reg y;  
always @(posedge clk) begin  
    if(& a) begin  
        x <= 2;  
        y <= 1;  
    end  
    else if(!b) x <= a;  
end
```



∪: always и синхронные регистры

Чуть более нетривиальный пример:

```
reg [1:0] x;  
reg y;  
always @(posedge clk) begin  
    if(&a) begin  
        x <= 2;  
        y <= 1;  
    end  
    else if(!b) x <= a;  
end
```

Таблица значений регистра:

		x y			
		00	01	10	11
b	a	00	01	10	11
	0	00 y	01 y	10 y	10 1
1	x y	x y	x y	10 1	

∪: always и синхронные регистры

Пример: поддерживаемое описание D-триггера с асинхронным сбросом

```
reg q;  
always @(posedge clk, posedge rst)  
    if(rst) q <= 0;  
    else q <= d;
```

Общие требования к поддерживаемому описанию синхронного регистра с дополнительными асинхронными входами

1. Список событий: непустой набор событий фронтов

2. Команда процедуры:

```
if(<усл1>) <команда> else if(<усл2>) <команда>  
    else ... if(<услк>) <команда> else <команда>,
```

где <усл1>...<услк> — выражения,

соответствующие (\mapsto) всем событиям списка, кроме одного:

- ▶ `posedge x` \mapsto `x`
- ▶ `negedge x` \mapsto `!x` или `~x`

3. Все присваивания процедуры **неблокирующие**

∪: always и синхронные регистры

Пример: поддерживаемое описание D-триггера с асинхронным сбросом

```
reg q;  
always @(posedge clk, posedge rst)  
    if(rst) q <= 0;  
    else q <= d;
```

Общие требования к поддерживаемому описанию синхронного регистра с дополнительными асинхронными входами

4. Выходы регистра: переменные в левых частях присваиваний
5. Входы регистра:
 - ▶ тактовый: указанный в списке событий и не имеющий соответствующего выражения в цепочке ветвлений
 - ▶ дополнительные асинхронные: остальные указанные в списке событий
 - ▶ остальные: точки, значения которых используются в процедуре

∪: always и синхронные регистры

Расхождение между программной семантикой процедуры и соответствующим синхронным регистром с дополнительными асинхронными входами:

▶ Программная семантика:

- ▶ порядок условий в ветвлениях = **приоритеты** асинхронных входов
- ▶ асинхронные входы «срабатывают» только в моменты фронтов согласно списку условий

▶ Аппаратная семантика:

- ▶ если несколько условий, соответствующих асинхронным входам, выполняются одновременно, то поведение не специфицировано
- ▶ асинхронные входы «работают» всегда от «включающего» фронта до «выключающего»

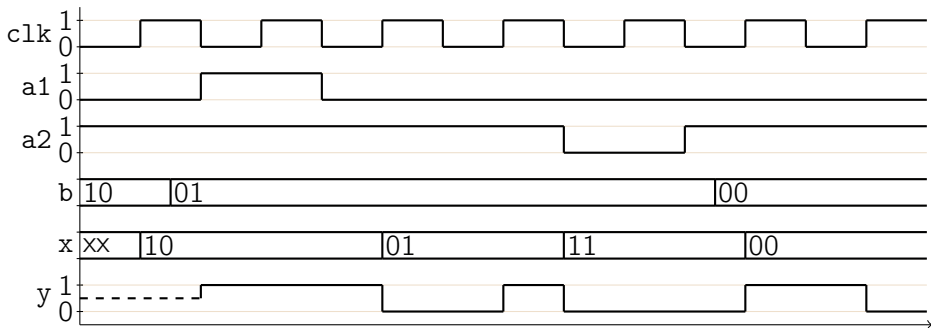
Компромисс:

- ▶ В программной симуляции не «включать» несколько асинхронных входов одновременно
- ▶ В командах, описывающих действие асинхронных входов, присваивать **только** константные выражения

∪: always и синхронные регистры

Чуть более нетривиальный пример:

```
reg [1:0] x;  
reg y;  
always @(posedge clk, posedge a1, negedge a2)  
    if(!a2) begin x <= 3; y <= 0; end  
    else if(a1) y <= 1;  
    else begin x <= b; y <= x[0]; end
```



∪: always и асинхронные регистры

Пример: поддерживаемые описания RS-триггера

```
reg q;  
always @(s, r) if(s) q <= 1; else if(r) q <= 0;  
always @(s, r) begin if(s) q <= 1; if(r) q <= 0; end
```

Общие требования к поддерживаемому описанию

асинхронного регистра

1. Список событий: непустой набор событий смены значения
2. В списке событий перечислены все точки, значения которых используются в процедуре
3. Все присваивания процедуры **неблокирующие**
4. Выходы регистра: переменные в левых частях присваиваний
5. Входы регистра: точки, значения которых используются в процедуре
6. Для каждого выходного разряда должен существовать набор значений на входах, для которого никакое значение не присваивается в этот разряд

∪: always и комбинационные схемы

Пример: поддерживаемое описание мультиплексора

```
reg out;  
always @(in0, in1, s) begin  
    out = 1'bx;  
    case(s)  
        1'd0: out = in0;  
        1'd1: out = in1;  
    endcase  
end
```

Общие требования к поддерживаемому описанию комбинационной схемы

1. Список событий: непустой набор событий смены значения
2. Для каждого разряда каждой переменной, встречающейся в левых частях присваиваний
 - ▶ либо все присваивания в этот разряд **блокирующие**,
 - ▶ либо все присваивания в этот разряд **неблокирующие**

У: always и комбинационные схемы

Пример: поддерживаемое описание мультиплексора

```
reg out;
always @(in0, in1, s) begin
    out = 1'bx;
    case(s)
        1'd0: out = in0;
        1'd1: out = in1;
    endcase
end
```

Общие требования к поддерживаемому описанию комбинационной схемы

3. Выходы схемы: переменные в левых частях присваиваний
4. Входы схемы: точки, значения которых на момент начала выполнения процедуры используются в присваиваниях
5. В списке событий должны быть перечислены **все** входы схемы
6. Для каждого набора значений на входах должно выполняться хотя бы одно присваивание значения в каждый выход

У: always и комбинационные схемы

Чуть более нетривиальный пример:

```
wire [1:0] z;  
wire u;  
reg x, y;  
always @(z, u) begin  
    x = 1'bx; y = 0;  
    case(z)  
        2'd0: x = u;  
        2'd3: begin y = !u; x = y; end  
    endcase  
end
```

Аппаратная семантика — комбинационная схема

со входами u (ширины 1), z (ширины 2) и выходами x , y (ширины 1), реализующая систему функций со следующей таблицей значений:

		x y			
		z	00	01	10
u	0	00	-0	-0	11
	1	10	-0	-0	00

У: краткое задание списка событий

Список событий **поддерживаемой** реагирующей процедуры, задающей асинхронный регистр или комбинационную схему, можно устроить так: перечислить **в точности все** точки, встречающиеся в правых частях присваиваний и условиях ветвлений и команд выбора

Перечислять эти точки вручную — это долго и приводит к ошибкам:

- ▶ описали процедуру, проглядели точку в правой части и не включили её в список \Rightarrow не поддерживается
- ▶ переделали процедуру, добавив зависимость от новой точки, и не обновили список \Rightarrow не поддерживается

Рекомендуемый способ записи реагирующей процедуры, позволяющий избежать таких ошибок:

```
always @(*) <команда>      или      always @* <команда>
```

@(*) и @* = «список событий смены значения для всех точек, встречающихся в правых частях присваиваний и в условиях <команды>»

∪: краткое задание списка событий

Пример:

```
always @* begin
  x = 1'bx; y = 0;
  case(z)
    2'd0: x = u;
    2'd3: begin
      y = !u;
      x = y;
    end
  endcase
end
```

=

```
always @(z, u, y) begin
  x = 1'bx; y = 0;
  case(z)
    2'd0: x = u;
    2'd3: begin
      y = !u;
      x = y;
    end
  endcase
end
```

Небольшая тонкость

Переменная y является выходом и при этом включена в список событий

В этом нет ничего страшного: значение y на момент начала выполнения процедуры не используется ни в одном присваивании, а значит, не посылается на вход соответствующей схемы