

Работа № 4. Автоматы конечных состояний

Проектирование последовательностной логики с использованием концепции автоматов конечных состояний (англ. FSM, Finite State Mashine) позволяет представить разрабатываемое устройство как некоторую машину, работающую в различных состояниях. Примером такой машины может быть интерфейс интегральной схемы для последовательного приёма-передачи данных, который работает в таких состояниях как приём, передача, ожидание, проверка, ответ и пр.

Теория конечных автоматов предполагает наличие некоторого количества различных типов автоматов. С точки зрения проектирования интегральных схем наиболее используемыми являются автомат Мура и автомат Мили.

Описание конечного автомата на Verilog

При описании конечного автомата рекомендуется следовать следующим правилам:

1. Каждый конечный автомат следует описывать в отдельном модуле с целью возможности лёгкости чтения кода, а также упрощения работы компилятора.
2. При кодировании состояний автомата не используйте препроцессорную директиву ``define`, используйте либо `parameter`, либо `localparam`.
3. Все последовательностные `always`-блоки используют неблокирующее присваивание, все комбинационные – блокирующие. Это позволит избежать гонок сигналов в автомате.
4. Описание конечного автомата должно легко поддаваться отладке.

Для описания регистра состояний используется последовательностный `always`-блок с неблокирующими присваиваниями.

Листинг 4.1 – Описание регистра состояний

```
1 always @(posedge clk, negedge _rst)
2   if ( !_rst )
3     state <= S0;
4   else
5     state <= nextstate;
```

Для определения следующего состояния используется комбинационный `always`-блок с блокирующим присваиванием. Общее представление такого блока показано в листинге ниже.

Листинг 4.2 – Описание комбинационной схемы, вычисляющей следующее состояние

```
1 always @(*)
2   case ( state )
3     S0 :
4       if ( input_signal )
5         nextstate = S1;
6       else
7         nextstate = S0;
8       // ...
9     default : nextstate = S0;
10  endcase
```

Здесь в каждом состоянии проверяется значение входного сигнала `input_signal`, после чего формируется значение следующего состояния, которое автомат примет при приходе переднего фронта тактового импульса согласно листинга 4.1.

Выходные сигналы могут так же формироваться с помощью комбинационных `always`-блоков, либо при помощи оператора непрерывного присваивания `assign`.

Листинг, реализующий автомат Мура для поиска последовательности «11» в битовом потоке данных приведён ниже

Листинг 4.3 – Описание автомата Мура

```
1 module search_Moore(D, clk, _rst, Q);
2
3   input D, clk, _rst;
4   output Q;
5
6   localparam SAD = 0, HOPE = 1, HOORAY = 2;
7
8   reg [1:0] state, nextstate;
9
10  //state register
11  always @(posedge clk, negedge _rst)
12    if ( !_rst )
13      state <= SAD;
14    else
15      state <= nextstate;
16
17  //combinatorial state function
18  always @(*)
19    case ( state )
20      SAD : nextstate = (D) ? HOPE : SAD;
21      HOPE, HOORAY : nextstate = (D) ? HOORAY : SAD;
22      default : nextstate = SAD;
23    endcase
24
```

```

25 //output logic
26 assign Q = (state == HOORAY) ? 1'b1 : 1'b0;
27
28 endmodule

```

Автомат Мили требует на одно состояние меньше по сравнению с автоматом Мура, так как выходное значение определяется также входными сигналами. Автомат Мили описан в листинге ниже

Листинг 4.4 – Описание автомата Мили

```

1 module search_Mealey(D, clk, _rst, Q);
2
3 input D, clk, _rst;
4 output Q;
5
6 localparam SAD = 0, HOPE = 1;
7
8 reg state, nextstate;
9
10 //state register
11 always @(posedge clk, negedge _rst)
12   if ( !_rst )
13     state <= SAD;
14   else
15     state <= nextstate;
16
17 //combinatorial state function
18 always @(*)
19   case ( state )
20     SAD, HOPE : nextstate = (D) ? HOPE : SAD;
21     default : nextstate = SAD;
22   endcase
23
24 //combinatorial output function
25 assign Q = (D == 1'b1 && state == HOPE) ? 1'b1 : 1'b0;
26
27 endmodule

```

Графическое задание конечного автомата

Одним из вариантов задания конечного автомата в среде Quartus Prime является графическое представление графа автомата. Для создания такого представления следует проделать следующие шаги (на примере автомата, реализующего поиск последовательности «11» в потоке битовых данных).

1. Создайте в проекте новый файл типа State Mashine File. Будет создан smf-файл. В появившемся окне в области слева можно увидеть входные и выходные порты автомата. Нажатием правой кнопки мыши можно вызвать контекстное меню, позволяющее добавить новые порты.

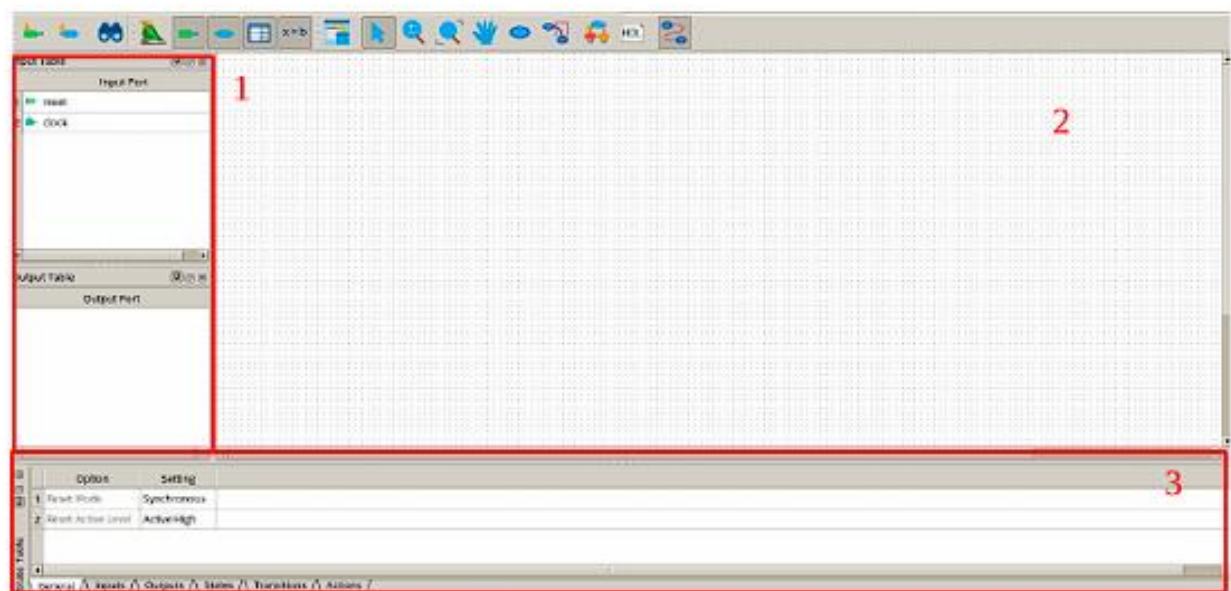



Рисунок 4.1 – Окно графического представления автомата:

1 – входные и выходные сигналы, 2 – область рисования графа автомата, 3 – область параметров

2. Для автомата, реализуемого нами, добавьте необходимые входные и выходные порты.
3. Добавьте состояния автомата. Для этого нажмите State Tool на панели меню , курсором установите состояние на свободном поле области рисования. За один раз можно добавить несколько состояний.
4. Двойным щелчком по состоянию откройте окно свойств, во вкладке General укажите название состояния, а также отметьте состояние по умолчанию (то, в которое переходит автомат по сигналу reset, оно отображается со стрелочкой).

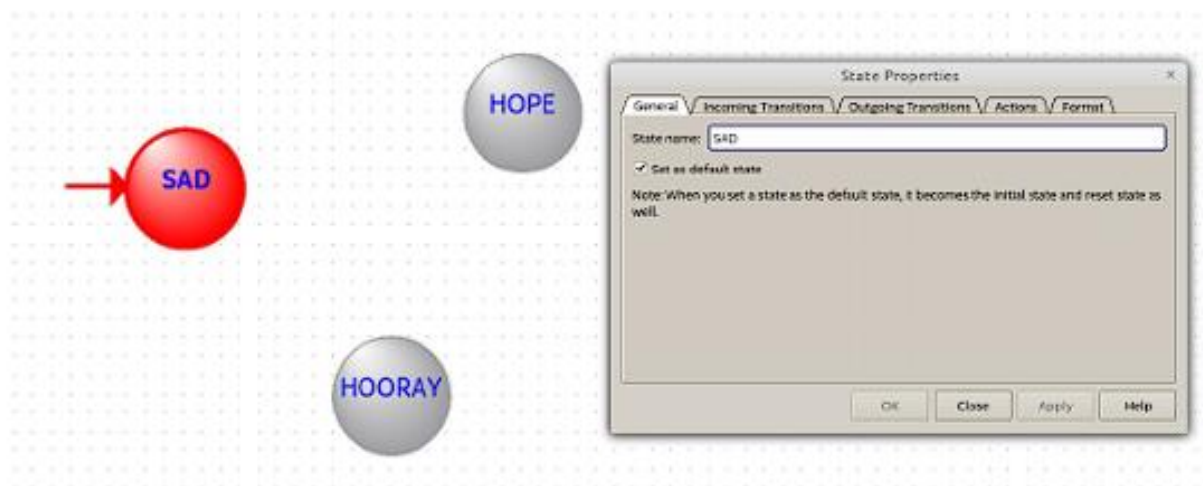



Рисунок 4.2 – Свойства состояния

5. Затем добавьте переходы между состояниями инструментом Transition Tool . Для задания петли достаточно один раз щёлкнуть по состоянию. Для задания перехода между состояниями необходимо с зажатой левой кнопкой мыши провести курсором от начального состояния к следующему.

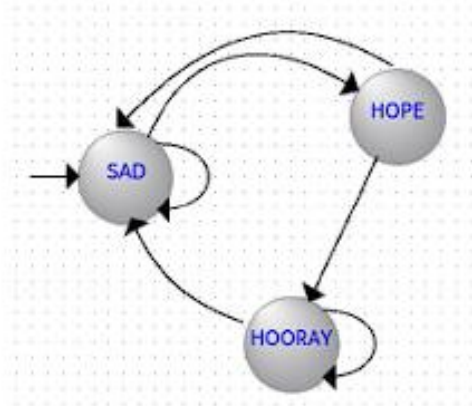


Рисунок 4.3 – Переходы между состояниями

6. Следующим шагом следует указать условия переходов из одного состояния в другое. Для этого можно использовать два способа.
- Первый заключается в указании условий входящих в состояние и исходящих из него переходов. Откройте окно свойств состояния и на вкладках Incoming Transitions и Outcoming Transitions введите в нотации Verilog условия переходов (для отсутствия условия, введите в нотации VHDL условие OTHERS)

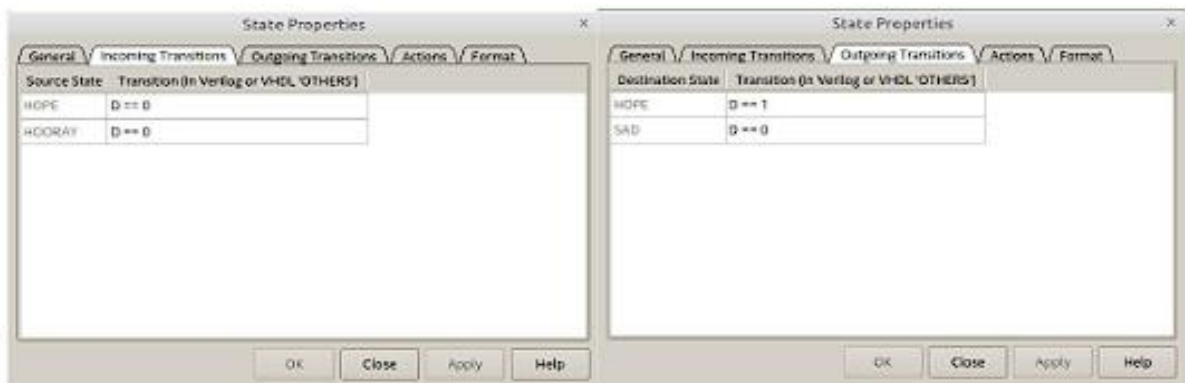


Рисунок 4.4 – Входящие и исходящие переходы состояния SAD

- Второй заключается в непосредственном указании условий каждого перехода. Откройте окно свойств перехода и также в нотации Verilog введите условие (или в нотации VHDL укажите OTHERS).

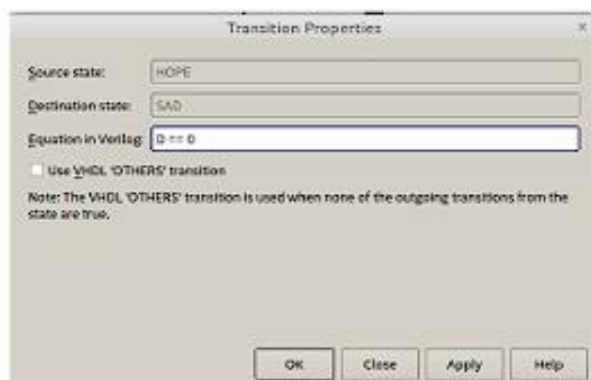


Рисунок 4.5 – Переход из состояния HOPE в состояние SAD

7. После указания всех переходов они отобразятся на графе автомата.

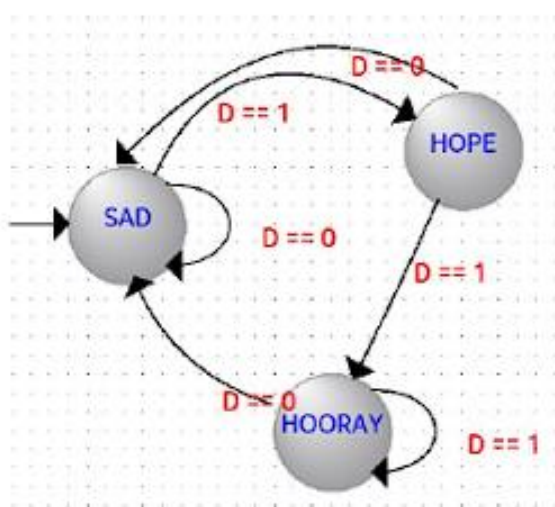


Рисунок 4.6 – Граф автомата с переходами

8. На вкладке Actions каждого состояния укажите значение выходных сигналов в этом состоянии.

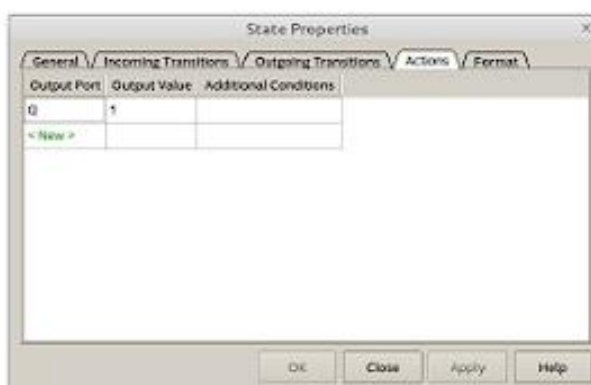



Рисунок 4.7 – Вкладка Actions для состояния HOORAY

9. В поле нижних вкладок на вкладке *General* можно указать, является ли сброс синхронным или асинхронным, а также задать активный уровень

сигнала сброса. На вкладке *Inputs* отображаются все входные сигналы с указанием сигнала сброса и тактового сигнала. На вкладке *Outputs* отображаются все выходные сигналы. Их можно задать как регистровыми, так и не регистровыми (синхронизируется ли значение выходного сигнала через регистр), можно указать, что состояние меняется в тот же самый цикл тактового сигнала или на следующий. На вкладке *States* перечислены все состояния, в *Transitions* – все функции переходов, а на *Actions* – все функции выходов.

10. После формирования графического представления конечного автомата, для дальнейшего его тестирования необходимо сформировать его HDL-описание. Для этого нажмите кнопку Generate HDL File  на панели меню и в открывшемся окне выберите Verilog. Сформируется HDL-описание автомата.

Моделирование конечного автомата в ModelSim

Для вывода текущего состояния на осциллограмму в ModelSim следует использовать следующий ход. В описании модуля добавить регистровую переменную и при помощи процедурного оператора выбора **case** задать ей строковые значения для каждого состояния. Например,

Листинг 4.1 – Определение текстовой переменной в описании модуля

```
1 reg [63:0] textstate;  
2 always @ ( state )  
3 case ( state )  
4     STATE1 : textstate = «STATE1»;  
5     STATE2 : textstate = «STATE2»;  
6     STATE3 : textstate = «STATE3»;  
7     ...  
8     STATEn : textstate = «STATEn»;  
9 endcase
```

Следует отметить, что длина переменной определяется максимальной длительностью имени состояния, каждая буква – это один байт или 8 бит, соответственно для слова длиной 8 символов необходимо определить переменную длиной 64 бита. Конструкция в листинге 4.1 несинтезируема.

Для вывода значения состояния в тестбенче следует обратиться к внутреннему сигналу тестируемого модуля. Это осуществляется при помощи иерархического обращения в нотации

```
<экземпляр_модуля_верхнего_уровня>.<экземпляр_модуля_уровня_2>...<экземпляр_модуля_уровня_n>.<сигнал_в_экземпляре_модуля_n>
```



```
1 myModule DUT(.clk(clk), .rst(rst), .D(D), .Q(Q));
2 wire [63:0] state = DUT.textstate; //обращаемся к внутренней переменной
textstate экземпляра DUT модуля myModule
```

При моделировании в ModelSim на сигнале состояния state необходимо выбрать представление Radix > ASCII.



Упражнения

1. Модифицируйте автоматы Мура и Мили для обнаружения уникальной последовательности 11 в битовом потоке данных.
2. Спроектируйте универсальный сдвиговый регистр, реализующий параллельный ввод 16-разрядного числа и параллельный вывод последовательный вывод, логический сдвиг влево, логический сдвиг вправо, кольцевой сдвиг влево, кольцевой сдвиг вправо в виде автомата.
3. Спроектируйте автомат, реализующий работу светофора в следующем режиме: 40 секунд горит красный свет, затем в течение 21 секунды горит зелёный, после чего загорается жёлтый свет на 3 секунды. Продемонстрируйте его работу на плате с ПЛИС. В качестве сброса автомата используйте кнопку, а индикаторы – светодиоды. *Указание:* для формирования тактового сигнала в 1 Гц создайте отдельный модуль делителя частоты, работающем на частоте 50 МГц (внутренняя частота ПЛИС).
4. Модифицируйте описанный выше автомат светофора одним из следующих способов (узнайте у преподавателя):
 - a. в последние 5 секунд зелёный сигнал мигает с частотой 1 Гц;
 - b. по нажатию кнопки во время красного света через 3 секунды загорается зелёный, если красному свету осталось гореть меньше 3 секунд, то нажатие кнопки игнорируется.
 - c. при переключении с красного на зелёный свет также загорается на 3 секунды жёлтый.