**МИНОБРНАУКИ РОССИИ**

**САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ**

**ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ**

**«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)**

**Кафедра Вычислительной Техники**

**ОТЧЕТ**

**по лабораторной работе №2**

**по дисциплине «Верификация и тестирование встраиваемых систем»**

**ТЕМА: Представление комбинационных схем и простых триггерных устройств**

**Вариант 1**

|  |  |  |
| --- | --- | --- |
| Студент гр. 6309 |  | Васин А. М. |
| Преподаватель |  | Мурсаев А. Х. |
|  |

Санкт-Петербург

2021

# Цель работы

Выполнить модификацию и осуществить моделирование комбинационной схемы на основе различных способов задания логической функции, а также исследовать использование комбинационной схемы в сочетании с запоминающим элементом.

Вариант №1

|  |  |  |  |
| --- | --- | --- | --- |
| x2 | x1 | x0 | F |
| 0 | 0 | 0 | 1 |
| 0 | 0 | 1 | 1 |
| 0 | 1 | 0 | 0 |
| 0 | 1 | 1 | 0 |
| 1 | 0 | 0 | 0 |
| 1 | 0 | 1 | 0 |
| 1 | 1 | 0 | 1 |
| 1 | 1 | 1 | 0 |

# Ход работы

1. Создали новый проект и реализовали выполнение функции, согласно варианту.
2. Провели симуляцию и запуск проекта, результаты на рисунках ниже.

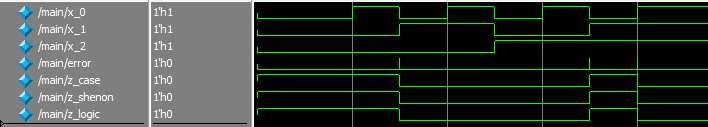


Рисунок 1 Результат совпал с ожиданием

Код программы:

module lab2;

parameter truth\_table=8'h43; //8'b01000011

wire error;

reg x\_0, x\_1, x\_2, z1;

reg [3:0] takt;

initial begin

clock='b0;

x\_0='b0;

x\_1='b0;

x\_2 ='b0;

for (takt=0;takt!=8;takt=takt+1)

begin

#20

#5 x\_0= takt[0];

x\_1= takt[1];

x\_2= takt[2];

end

end

assign

error=(z1==truth\_table[{x\_2, x\_1, x\_0}])? 0:1;

always @(x\_0,x\_1,x\_2)

begin

z\_logic = (~x\_0 & ~x\_1 & ~x\_2) | (x\_0 & ~x\_1 & ~x\_2) | (x\_1 & x\_2 & ~x\_0);

end

endmodule

1. Доработали программу вычислением функции на основе разложения Шеннона по двум переменным.

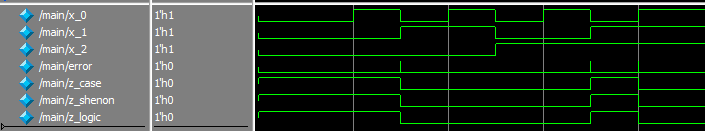


Рисунок 3 Разложение Шеннона - z\_shenon

Код блока always, в котором происходит вычисление функции с помощью разложения Шеннона:

always @(x\_0,x\_1,x\_2)

begin

if(x\_1)

z\_shenon = (x\_2) ? ~x\_0 : 0;

else

z\_shenon = (x\_2) ? 0 : 1;

end

1. Вычисление с помощью case

Код блока always, в котором происходит вычисление функции с помощью разложения case:

always @(x\_0,x\_1,x\_2)

begin

case ({x\_2, x\_1, x\_0})

3'd0: z\_case = 1;

3'd1: z\_case = 1;

3'd2: z\_case = 0;

3'd3: z\_case = 0;

3'd4: z\_case = 0;

3'd5: z\_case = 0;

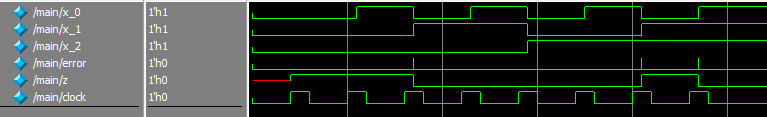
3'd6: z\_case = 1;

3'd7: z\_case = 0;

endcase

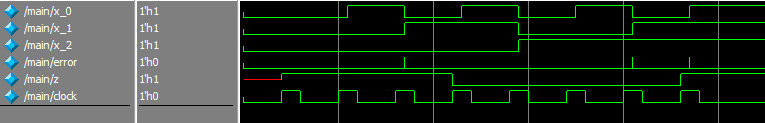
end

1. Вставили триггер со статическим управлением



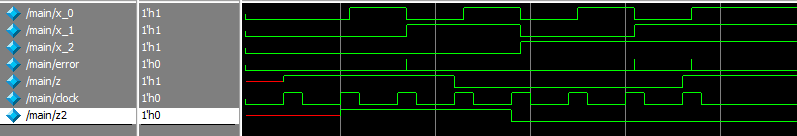
Как мы видим, из результатов, триггер работает правильно – значение функции меняется только при положительном clock.

1. Реализуем триггер с динамическим управлением:



Произошла задержка на один такт clock.

1. Добавим ещё один такт задержки с помощью нового триггера:



Код программы с триггерами:

module lab2;

parameter truth\_table=8'h43; //8'b01000011

wire error;

reg x\_0, x\_1, x\_2, z1, z, z2, clock;

reg [3:0] takt;

initial begin

clock='b0;

x\_0='b0;

x\_1='b0;

x\_2 ='b0;

for (takt=0;takt!=8;takt=takt+1)

begin

#20 clock='b1;

#5 x\_0= takt[0];

x\_1= takt[1];

x\_2= takt[2];

#5 clock = 'b0;

end

end

//D-триггер со статическим управлением

/\*always @(z1, clock)

begin

if (clock)

z = z1;

end\*/

//D-триггер с динамическим управлением по положительному фронту - задержка на первый такт

always @(posedge clock)

begin

z <= z1;

end

//D-триггер с динамическим управлением по положительному фронту - задержка на второй такт

always @(posedge clock)

begin

z2 <= z;

end

assign

error=(z1==truth\_table[{x\_2, x\_1, x\_0}])? 0:1;

always @(x\_0,x\_1,x\_2)

begin

z1 = (~x\_0 & ~x\_1 & ~x\_2) | (x\_0 & ~x\_1 & ~x\_2) | (x\_1 & x\_2 & ~x\_0);

end

endmodule

Если выполнить блокирующие присваивания, то задержки не произойдёт, потому что присваивание произойдёт незамедлительно, вот результат:

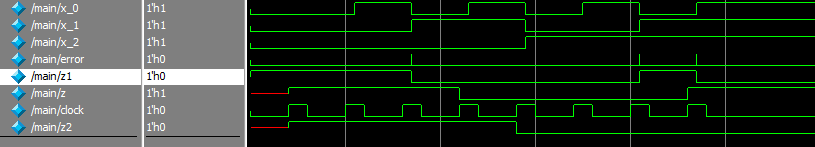


Рисунок 5 Блокирующие присваивания триггеров

# Контрольные вопросы

1. **Представьте синтаксическое описание определение и порядок исполнения операторов if, case.**

if(«условие-выражение»)

begin

«действия»

end

else

begin

«действия»

end

case(«условие-выражение»)

опция-1:

begin

«действия»

End

опция-2:

begin

«действия»

end

опция-n:

begin

«действия»

end

default:

begin

«действия»

end

endcase

x\_1 = (x\_0) ? 1 : 0;

1. **Как выполняется переход от табличного задания логической функции к алгебраическому?**

Берём и строим либо ДНФ, либо КНФ. Если единичек меньше, то строится по единичкам.

**Пример**

|  |  |  |  |
| --- | --- | --- | --- |
| x2 | x1 | x0 | F |
| 0 | 0 | 0 | 1 |
| 0 | 0 | 1 | 1 |
| 0 | 1 | 0 | 0 |
| 0 | 1 | 1 | 0 |
| 1 | 0 | 0 | 0 |
| 1 | 0 | 1 | 0 |
| 1 | 1 | 0 | 1 |
| 1 | 1 | 1 | 0 |

Три единички:

1. (~x2) & (~x1) & (~x0)
2. (~x2) & (~x1) & (x0)
3. (x2) & (~x1) & (~x0)

Объединяем с помощью «или» и получаем алгебраическое задание функции:

((~x2) & (~x1) & (~x0)) | ((~x2) & (~x1) & (x0)) | ((x2) & (~x1) & (~x0))

Также эту форму можно упрощать и делать более короткой.

1. **Продемонстрируйте на примере применение разложения Шеннона.**

|  |  |  |  |
| --- | --- | --- | --- |
| x2 | x1 | x0 | F |
| 0 | 0 | 0 | 1 |
| 0 | 0 | 1 | 1 |
| 0 | 1 | 0 | 0 |
| 0 | 1 | 1 | 0 |
| 1 | 0 | 0 | 0 |
| 1 | 0 | 1 | 0 |
| 1 | 1 | 0 | 1 |
| 1 | 1 | 1 | 0 |

Берем x2: если x2 == 1, то ответ равен x1;

x2 == 0, то при x1 == 0 результат 0,

а при x1== 1 результат ~x0

В коде:

if(x\_2)

z\_shenon = x\_1;

else

z\_shenon = (x\_1) ? ~x\_0 : 0;

1. **Способы управления триггерными устройствами и их описание в языке VerilogHDL?**

В связи с тем, что в отличие от комбинационных схем различные входы в триггере инициируют различные действия (в том числе изменение некоторых входов вообще не инициируют переключений) существенными являются состав списка чувствительности и порядок анализа условий, определяющих очередное состояние триггера (например в операторах if и case).

Список чувствительности задает способ управления триггером. Известны следующие способы

- асинхронное управление;

- статическое управление, или управление уровнем;

- динамическое управление, или управление фронтом;\

- смешанные варианты.

Асинхронное управление характеризуется тем, что изменение сигналов на информационных входах непосредственно влияет на состояние. Единственный нетривиальный триггер с асинхронным управлением-(исключая смешанное управление) это RS-триггер. Список чувствительности включает оба входа r и s. Тогда можно записать для выхода q

**always** @ (r,s)

**if** (r && s) q=’bx;// при одновременной подаче активных (единичных)

// уровней на оба входа состояние не определено

**else if** (r) q=’b0;

**else**  q= s ? ‘b1:q;// при одновременной подаче нулей на оба входа

// состояние сохраняется

Триггер со статическим управлением чувствителен и к синхросигналу, и к информационным входам, но может менять состояние только при активном уровне на входе синхронизации. При активном уровне триггер отслеживает изменения сигналов на информационных входах. Значит триггер типа d может быть представлен таким фрагментом

a**lways** @ (clock,data)

**if** (clock) q=data;

Оператор исполняется после любого изменения, как сигнала clock (тактирующий сигнал), так и сигнала data (информационный сигнал), однако при clock='0' изменение состояния Q не происходит.

Заметим, что при задании асинхронного и статического управления следует использовать блокирующие присваивания.

При динамическом управления изменение состояния триггера проиcходит только по фронту синхросигнала. Информационные сигналы не входят в список чувствительности. Если несколько триггеров запускаются общим тактовым сигналом, и следующее состояние каждого триггера в такой совокупности зависит от предыдущего состояниями триггеров, то предыдущее состояние должно сохраняться до выполнения всех операторов инициируемых этим сигналом. Значит, следует использовать неблокирующие присваивания сигналам, представляющим состояния. Для описания триггеров со смешанным управлением необходимо учитывать относительные приоритеты сигналов управлении ( например, асинхронные входы более приоритетны в сравнение входами тактирования).

Исходя из этого описание J-K с динамическим входом и асинхронным сбросом может быть таким

**always** @ (posedge clock **or** posedge reset)

**if** (reset) q=’b0;

**else case**({j,k})

2’b00; q<=q

2’b01: q<=’b0;

2’b10: q<=’b1;

2’b11: q<=~q;

**Default**: q<=bx;

1. **Когда могут возникнуть в схеме «неожиданные» триггеры?**

Неожиданные триггеры могут возникнуть в схеме если вместо блокирующего присваивания использовать неблокирующее в ситуации, когда присвоения должны происходить последовательно, так же они могут возникнуть при неполном указании списка чувствительности в реализации комбинаторной схемы.

1. **Синтаксис декларации примитива и его вхождения в проект.**

Примитив – это модуль с одним выходом, поведение которого задаётся в табличной форме, причем входы и выходы – однобитовые сигналы. По уровню примитив эквивалентен модулю и может включаться в другие проекты по тем же правилам, что и модуль.

Примитив включается с помощью оператора вхождения .

**Пример**

primitive table\_func (z\_out, in2, in1, in0);

output z\_out;

input in2, in1, in0;

table

// in2, in1, in0 : z\_out

0 0 ? : 1;

0 1 ? : 0;

1 0 ? : 0;

1 1 0 : 1;

1 1 1 : 0;

endtable

endprimitive

Пример использования примитива в модуле:

table\_func v1 (.z\_out(z\_table), .in2(x\_2), .in1(x\_1), .in0(x\_0));