

**Лабораторная работа № 2**

Вариант 6

по дисциплине

«Функциональная схемотехника»

Выполнил: Студент группы P33301

Савон Галина

Санкт-Петербург

2022 г.

**Цель работы:**

Получить навыки описания арифметических блоков на RTL-уровне с использованием языка описания аппаратуры Verilog HDL.

**Задание:**

Text

Description automatically generated with medium confidence

**Описание блока:**

\*Работает на **одном** сумматоре и **одном** умножителе\*

Разработанный блок состоит из нескольких модулей:

1. adder (сумматор)
2. multipl (мультиплексор)
3. main (модуль, который реализует конечный автомат, поочередно подсчитывающий составляющие результата)
4. seg7 (основной внешний модуль, который реализует получение входных данных с платы и вывод результата на нее)

**Код блока:**

Сумматор: получает на вход два числа и с помощью операции сложения подсчитывает результат

**module** adder**(**

**input** clk**,**

**input** **wire** **[**23**:**0**]** a**,**

**input** **wire** **[**23**:**0**]** b**,**

**output** **reg** **[**23**:**0**]** result

**);**

**always** **@(posedge** clk**)** **begin**

result **<=** a **+** b**;**

**end**

**endmodule**

Мультиплексор: получает на вход тактовый сигнал clk\_i, сигнал сброса rst\_i, два входных числа a\_bi, b\_bi, сигнал начала работы start\_i.

Имеет два состояния (IDLE, WORK). При установленном сигнале сброса – возвращает 0, и состояние переходит в IDLE.

Если rst = 0, start = 1, то переходит в состояние WORK, счетчик разрядов (ctr) устанавливается в 0.

Пока ctr меньше 16 - к промежуточному результату (part\_res) прибавляется смещенное влево на значение счетчика логическое “и” первого числа и одной из цифр второго числа, ка которую указывает счетчик. Счетчик инкрементируется. Когда счетчик достигает 8, результат записывается в y\_bo, модуль переходит в состояние IDLE, флаг работы устанавливается в 0.

**module** multipl**(**

**input** clk\_i**,**

**input** rst\_i**,**

**input** **[**7**:**0**]** a\_bi**,**

**input** **[**15**:**0**]** b\_bi**,**

**input** start\_i**,**

**output** busy\_o**,**

**output** **reg** **[**23**:**0**]** y\_bo

**);**

**localparam** IDLE **=** 1'b0**;**

**localparam** WORK **=** 1'b1**;**

**reg** **[**4**:**0**]** ctr**;** //perenos

**wire** **[**3**:**0**]** end\_step**;** //number length

**wire** **[**7**:**0**]** part\_sum**;**

**wire** **[**23**:**0**]** shifted\_part\_sum**;**

**reg** **[**7**:**0**]** a**;**

**reg** **[**15**:**0**]** b**;**

**reg** **[**23**:**0**]** part\_res**;**

**reg** state **=** IDLE**;**

**assign** part\_sum **=** a **&** **{**16**{**b**[**ctr**]}};**

**assign** shifted\_part\_sum **=** part\_sum **<<** ctr**;**

**assign** end\_step **=** **(**ctr **==** 5'b10000**);**

**assign** busy\_o **=** state**;**

**always** **@(posedge** clk\_i**)**

**if** **(**rst\_i**)** **begin**

ctr **<=** 0**;**

part\_res **<=** 0**;**

y\_bo **<=** 0**;**

state **<=** IDLE**;**

**end** **else** **begin**

**case** **(**state**)**

IDLE**:**

**if** **(**start\_i**)** **begin**

state **<=** WORK**;**

a **<=** a\_bi**;**

b **<=** b\_bi**;**

ctr **<=** 0**;**

part\_res **<=** 0**;**

**end**

WORK**:**

**begin**

**if** **(**end\_step**)** **begin**

state **<=** IDLE**;**

y\_bo **<=** part\_res**;**

**end**

part\_res **<=** part\_res **+** shifted\_part\_sum**;**

ctr **<=** ctr **+** 1 **;**

**end**

**endcase**

**end**

**endmodule**

Main : на вход принимает тактовый сигнал, входные значение свичей, значения сигнала старта и сигнала сброса. Возвращает подсчитанный результат и нынешнее состояние.

Имеет 4 состояния:

IDLE – ожидание

WORK1 – подсчет a\*b

WORK2 – подсчет a^2

WORK3 – подсчет a^3

**module** main**(**

**input** clk**,**

**input** **wire** **[**15**:**0**]** SW**,**

**input** **wire** start\_in**,**

**input** **wire** rst\_in**,**

**output** **wire** **[**23**:**0**]** res**,**

**output** **[**1**:**0**]** state\_o

**);**

**localparam** IDLE **=** 2'b00**;**

**localparam** WORK1 **=** 2'b01**;**

**localparam** WORK2 **=** 2'b10**;**

**localparam** WORK3 **=** 2'b11**;**

**reg** start\_r**;**

**reg** rst\_r**;**

**wire** rst\_m**;**

**reg** rst\_m\_r**;**

**assign** rst\_m **=** rst\_m\_r**;**

**wire** start\_m**;**

**reg** start\_m\_r**;**

**assign** start\_m **=** start\_m\_r**;**

**reg** **[**1**:**0**]** state **=** IDLE**;**

**assign** state\_o **=** state**;**

**wire** busy**;**

**wire[**7**:**0**]** a**; //отправляемое в мультиплексор значение**

**reg** **[**7**:**0**]** a\_r**;**

**assign** a **=** a\_r**;**

**wire[**15**:**0**]** b**; //отправляемое в мультиплексор значение**

**reg[**15**:**0**]** b\_r**;**

**assign** b **=** b\_r**;**

**wire** **[**23**:**0**]** result\_func**;**

**reg** **[**23**:**0**]** result\_mult**; //результат работы мультиплексора**

**reg** **[**23**:**0**]** result\_cube**; //результат работы work3**

**reg** **[**23**:**0**]** send\_a\_adder**; //отправляемое в adder значение**

**reg** **[**23**:**0**]** send\_b\_adder**; //отправляемое в adder значение**

**reg** **[**19**:**0**]** clkdiv **=** 0**;**

**always** **@(posedge** clk**)**

**if** **(**rst\_r **&&** rst\_in**)** **begin**

send\_a\_adder **<=** 0**;**

send\_b\_adder **<=** 0**;**

result\_mult **<=** 0**;**

result\_cube **<=** 0**;**

state **<=** IDLE**;**

rst\_m\_r **<=** 1**;**

**end** **else** **begin**

**case** **(**state**)**

IDLE**:**

**if** **(**start\_in **&&** start\_r**)**

**begin**

state **<=** WORK1**;**

b\_r **<=** SW**[**15**:**8**];**

a\_r **<=** SW**[**7**:**0**];**

rst\_m\_r **<=** 0**;**

start\_m\_r **<=** start\_in**;**

**end** **else** **begin**

send\_a\_adder **<=** result\_mult**;**

send\_b\_adder **<=** result\_cube**;**

**end**

WORK1**:**

**begin**

**if(!**busy **&&** **!**start\_m**)** **begin**

result\_mult **<=** result\_func**;**

state **<=** WORK2**;**

b\_r **<=** SW**[**7**:**0**];**

rst\_m\_r **<=** 0**;**

start\_m\_r **<=** 1**;**

start\_r **<=** 0**;**

**end** **else** **if(**busy**)** **begin**

start\_m\_r **<=** 0**;**

**end**

**end**

WORK2**:**

**begin**

**if(!**busy **&&** **!**start\_m**)** **begin**

state **<=** WORK3**;**

b\_r **<=** result\_func**;**

rst\_m\_r **<=** 0**;**

start\_m\_r **<=** 1**;**

start\_r **<=** 0**;**

**end** **else** **if(**busy**)** **begin**

start\_m\_r **<=** 0**;**

**end**

**end**

WORK3**:**

**begin**

**if(!**busy **&&** **!**start\_m**)** **begin**

result\_cube **<=** result\_func**;**

start\_m\_r **<=** 0**;**

state **<=** IDLE**;**

start\_r **<=** 0**;**

**end** **else** **if(**busy**)** **begin**

start\_m\_r **<=** 0**;**

**end**

**end**

**endcase**

**end**

**always** **@(posedge** clk**)** **begin**

clkdiv **<=** clkdiv**+**1**;**

**end**

**always** **@(**start\_in**)begin**

start\_r **<=** 1**;**

**end**

**always** **@(**rst\_in**)begin**

rst\_r **<=** 1**;**

**end**

adder add**(**clk**,** send\_a\_adder**,** send\_b\_adder**,** res**);**

multipl ml**(**clk**,** rst\_m**,** a**,** b**,** start\_m**,** busy**,** result\_func**);**

**endmodule**

seg7: на вход получает значения свичей, тактовый сигнал, и состояния кнопок BTNR (rst) и BTND (start). На выход LED(для удобства и понимания, какие свичи включены), LED16\_B, LED17\_B (соответствуют кнопкам), CA – CG, AN(вывод результата)

Если основной вычисляющий блок находится в ожидающем состоянии, то передающиеся в него значения rst и start обновятся.

**module** seg7**(**

**input** **wire[**15**:**0**]** SW**,**

**input** CLK100MHZ**,**

**input** BTND**,**

**input** BTNR**,**

**output** **wire[**15**:**0**]** LED**,**

**output** LED16\_B**,**

**output** LED17\_B**,**

**output** **reg** CA**,** //display's segments

**output** **reg** CB**,**

**output** **reg** CC**,**

**output** **reg** CD**,**

**output** **reg** CE**,**

**output** **reg** CF**,**

**output** **reg** CG**,**

**output** **reg** **[**7**:**0**]** AN//numbers of displays

**);**

**wire** **[**23**:**0**]** res**;**

**wire** **[**2**:**0**]** s**;** //part of clkdiv to make delay

**reg** **[**3**:**0**]** digit**;** //display num

**reg** **[**19**:**0**]** clkdiv **=** 0**;** //pointer to make delay

**wire** start**;**

**wire** rst**;**

**reg** rst\_send**;**

**reg** start\_send**;**

**wire** state**;**

main m**(**CLK100MHZ**,** SW**,** start\_send**,** rst\_send**,** res**,** state**);**

**assign** s **=** clkdiv**[**19**:**17**];**

**assign** LED**[**15**:**0**]** **=** SW**[**15**:**0**];**

**assign** LED17\_B **=** BTND**;**

**assign** rst **=** BTNR**;**

**assign** start **=** BTND**;**

**assign** LED16\_B **=** BTNR**;**

**always** **@(posedge** CLK100MHZ**)**

**case(**s**)**

0**:**digit **=** res**[**3**:**0**];**

1**:**digit **=** res**[**7**:**4**];**

2**:**digit **=** res**[**11**:**8**];**

3**:**digit **=** res**[**15**:**12**];**

4**:**digit **=** res**[**19**:**16**];**

5**:**digit **=** res**[**23**:**20**];**

6**:**digit **=** 4'b0000**;**

7**:**digit **=** 4'b0000**;**

**default:**digit **=** res**[**3**:**0**];**

**endcase**

**always** **@(\*)**

**case(**digit**)**

0**:{**CA**,** CB**,** CC**,** CD**,** CE**,** CF**,** CG**}** **=** 7'b0000001**;**

1**:{**CA**,** CB**,** CC**,** CD**,** CE**,** CF**,** CG**}** **=** 7'b1001111**;**

2**:{**CA**,** CB**,** CC**,** CD**,** CE**,** CF**,** CG**}** **=** 7'b0010010**;**

3**:{**CA**,** CB**,** CC**,** CD**,** CE**,** CF**,** CG**}** **=** 7'b0000110**;**

4**:{**CA**,** CB**,** CC**,** CD**,** CE**,** CF**,** CG**}** **=** 7'b1001100**;**

5**:{**CA**,** CB**,** CC**,** CD**,** CE**,** CF**,** CG**}** **=** 7'b0100100**;**

6**:{**CA**,** CB**,** CC**,** CD**,** CE**,** CF**,** CG**}** **=** 7'b0100000**;**

7**:{**CA**,** CB**,** CC**,** CD**,** CE**,** CF**,** CG**}** **=** 7'b0001111**;**

8**:{**CA**,** CB**,** CC**,** CD**,** CE**,** CF**,** CG**}** **=** 7'b0000000**;**

9**:{**CA**,** CB**,** CC**,** CD**,** CE**,** CF**,** CG**}** **=** 7'b0000100**;**

'hA**:{**CA**,** CB**,** CC**,** CD**,** CE**,** CF**,** CG**}** **=** 7'b0001000**;**

'hB**:{**CA**,** CB**,** CC**,** CD**,** CE**,** CF**,** CG**}** **=** 7'b1100000**;**

'hC**:{**CA**,** CB**,** CC**,** CD**,** CE**,** CF**,** CG**}** **=** 7'b0110001**;**

'hD**:{**CA**,** CB**,** CC**,** CD**,** CE**,** CF**,** CG**}** **=** 7'b1000010**;**

'hE**:{**CA**,** CB**,** CC**,** CD**,** CE**,** CF**,** CG**}** **=** 7'b0110000**;**

'hF**:{**CA**,** CB**,** CC**,** CD**,** CE**,** CF**,** CG**}** **=** 7'b0111000**;**

**default:** **{**CA**,** CB**,** CC**,** CD**,** CE**,** CF**,** CG**}** **=** 7'b0000000**;**

**endcase**

**always** **@(\*)begin**

AN**=**8'b11111111**;**

**#**1**;**

AN**[**s**]** **=** 0**;**

**end**

**always** **@(posedge** CLK100MHZ**)** **begin**

clkdiv **<=** clkdiv**+**1**;**

**if(**state **==** 2'b00**)begin**

start\_send **<=** start**;**

rst\_send **<=** rst**;**

**end**

**end**

**endmodule**

**Схема блока:**

**Область допустимых значений для разработанного блока:**

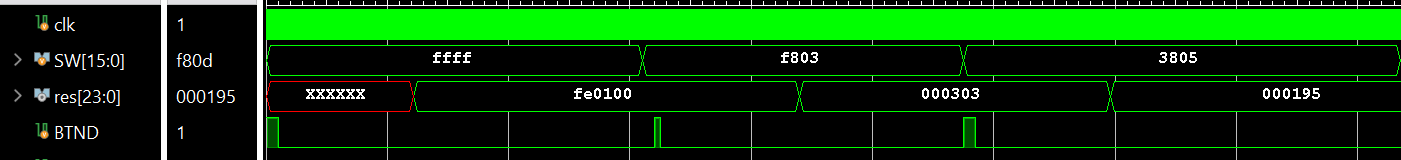
Входные значения – [0; ff]

Результат ab, a^2 – [0:fe01]

Результат a^3 – [0:fd02ff]

Результат ab+a^3 – [0:ff0100]

**Результат тестирования разработанного блока:**

****

**Время вычисления результата при тактовой частоте сигнала 100 МГц:**

**Graphical user interface

Description automatically generated**

**122 ns**

**Вывод:** были получены навыки описания арифметических блоков на RTL-уровне с использованием языка описания аппаратуры Verilog HDL. Модуль умножения можно ускорить при вычислении куба, если поменять числа, которые подаются на него, чтобы уменьшить количество итераций.