**МИНОБРНАУКИ РОССИИ**

**САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ**

**ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ**

**«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)**

**Кафедра Вычислительной Техники**

**ОТЧЕТ**

**по лабораторной работе №5**

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

**ТЕМА: Проектирование операционных устройств и квазистатическая верификация**

**Вариант 1**

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

Санкт-Петербург

2021

# Цель работы

Изучить процесс построения устройства и последовательного анализа разрядов входного слова по типовой схеме алгоритма.

Задание: подсчёт числа едини во входном коде.

# Ход работы

1. По заданию создали проект, добавили необходимые файлы и доработали их.

module control\_unit(clock, reset, start, y, c\_sh\_rg, c\_cnt, ready);

input clock, reset, start;

input y;

output c\_sh\_rg, c\_cnt, ready;

parameter n=8;

enum {init,cycle} state;

wire clock, reset, start,

y;// analyzing bit of word

reg ready;

reg [1:0]c\_sh\_rg; //control signal for module shift\_reg

reg [1:0] c\_cnt; //control signal for module counter

int k;

always @ (clock, reset)

begin

if (reset)

begin

state = init;

ready = 0;

end

else if (!clock)

case (state)

init:

begin

if (start)

begin

state = cycle;

ready = 0; k = 0;

c\_sh\_rg = 3; // was "1"

c\_cnt = 3; // loading

end

end

cycle:

begin

if (k==n) // control cycles count

begin

ready = 1;

k = 0;

c\_sh\_rg = 0;

c\_cnt = 0;// blocking load bits

state = init;

end

else

begin

//ready = 0;

c\_cnt[1] = y; c\_cnt[0] = 0; // sum/save

c\_sh\_rg = 2; // turn on shift reg

k = k+1;

end

end

endcase;

end

endmodule

1. Добавили тестовый модуль с генерацией входных данных случайным образом

module top;

sys\_bus bus (.\*);

mp\_mult mod\_mult (.dev(bus));

test\_bench prog\_tb (.dev(bus));

endmodule

program test\_bench (sys\_bus.monit dev);

class TEST\_DATA;

rand bit [dev.data\_size : 0] data;

constraint data\_valid { data > 0; }

endclass;

TEST\_DATA t\_data;

logic [4:0] takt;

int i = 1;

int results [2][100];

integer f;

initial

begin

f = $fopen("C:/Users/alvas/Documents/Univer/Mursaev/Lab\_5/outputs/output.txt","w");

if (f == 0)

begin

$info ("couldn't open file");

$finish;

end

else

$info ("file opened");

dev.clock = 0;

dev.reset = 0;

dev.in\_data = 8'hfa;

#15;

dev.reset = 1;

#15

dev.reset = 0;

t\_data = new();

assert (t\_data.randomize());

repeat(110)

begin

while(t\_data.randomize() != 1);

dev.in\_data = t\_data.data;

for (takt = 1; takt < 11; takt = takt + 1)

begin

if(takt == 1)

begin

dev.start = 1;

end

else if(takt == 2)

begin

dev.start = 0;

end

#5

dev.clock = 1;

#5

dev.clock = 0;

if(dev.ready)

begin

$fwrite(f,"%0d) %x - %b - %0d\n", i, dev.in\_data, dev.in\_data, dev.result);

end

end

i++;

end

$fclose(f);

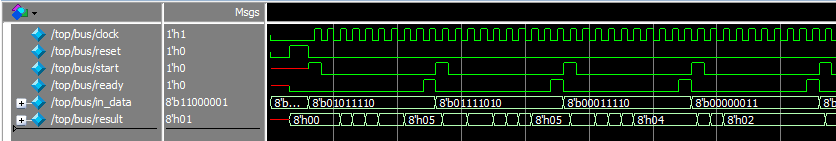
$info ("file closed");

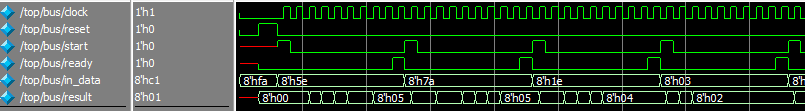
end

endprogram

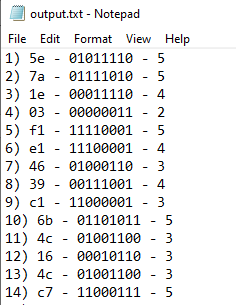
1. Правильность работы устройства невозможно определить простыми математическими операциями, поэтому было принято решение «выгрузить» все результат в файл outputs.txt и с помощью программы на языке программирования Python проверить, так как в данном языке программирования есть библиотека с встроенной функцией проверки количества вхождений определенного символа в число.
2. Результат работы программы от начала и до конца приведён на рисунках ниже.

Временная диаграмма моделирования:





Вывод результатов программы (уже тут можно убедиться в правильности работы программы):



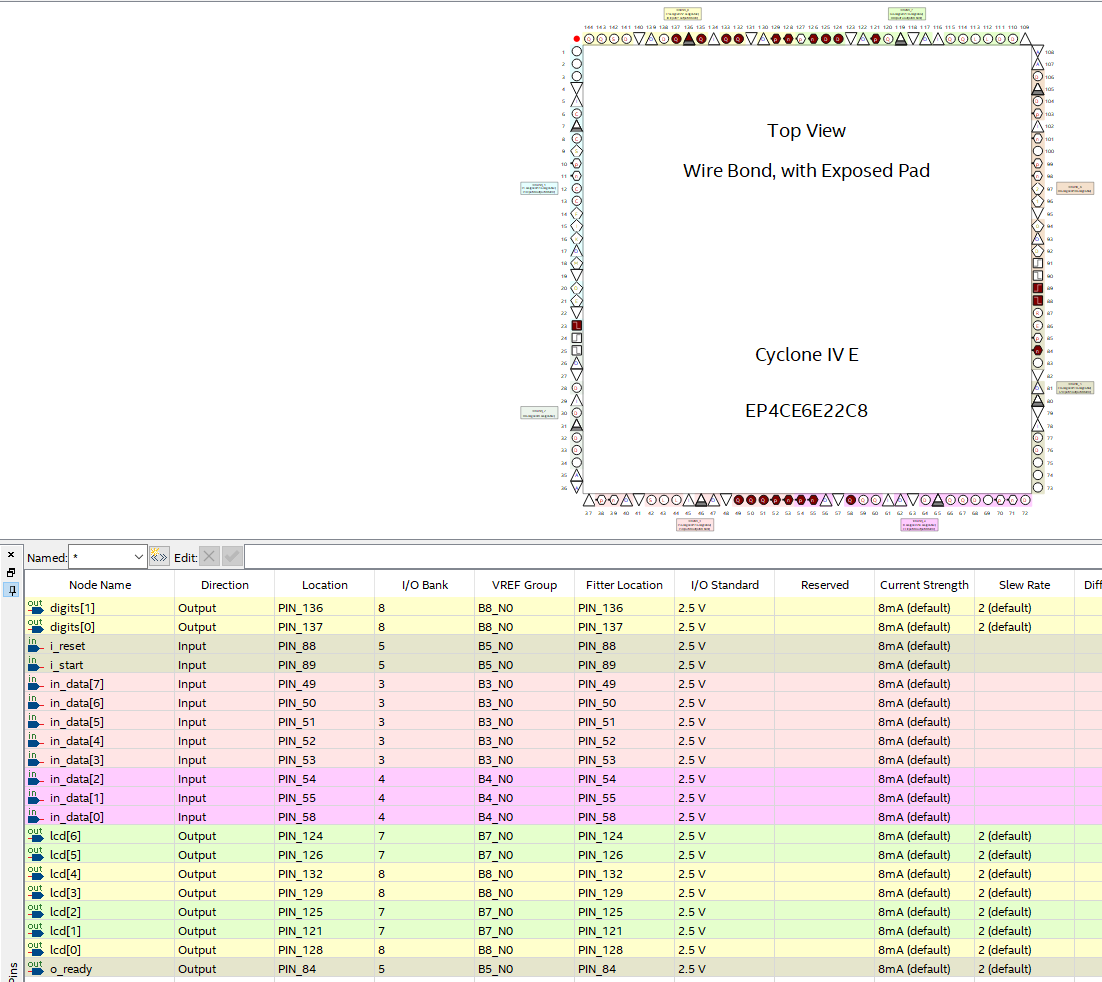
Вывод тестовой программы на ЯП Python:



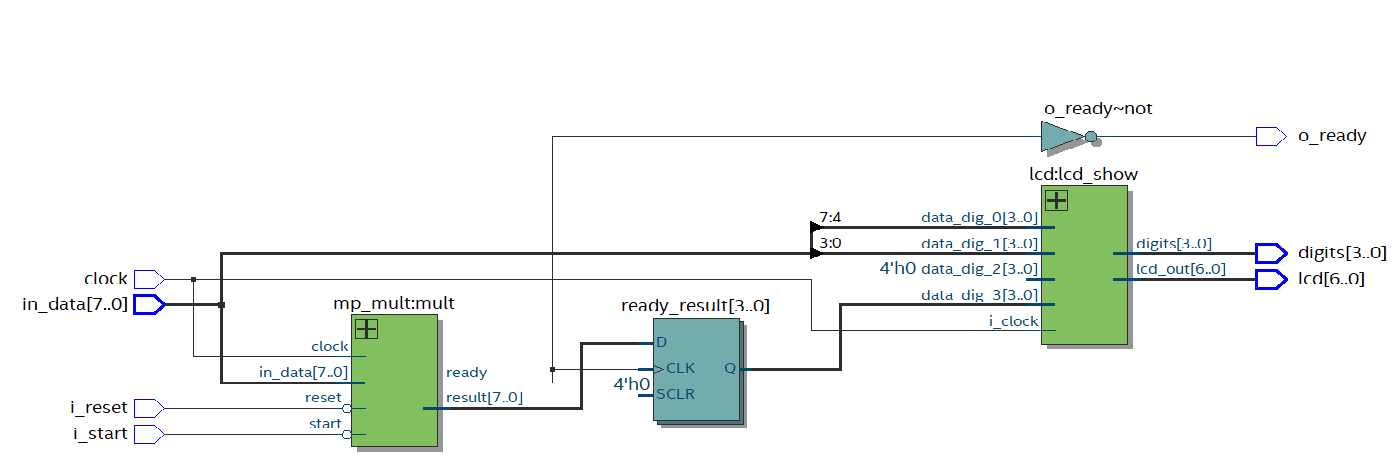
Как мы видим: результаты работы устройства и работы программы совпали, что говорит нам о том, что устройство спроектировано верно.

1. Макетирование

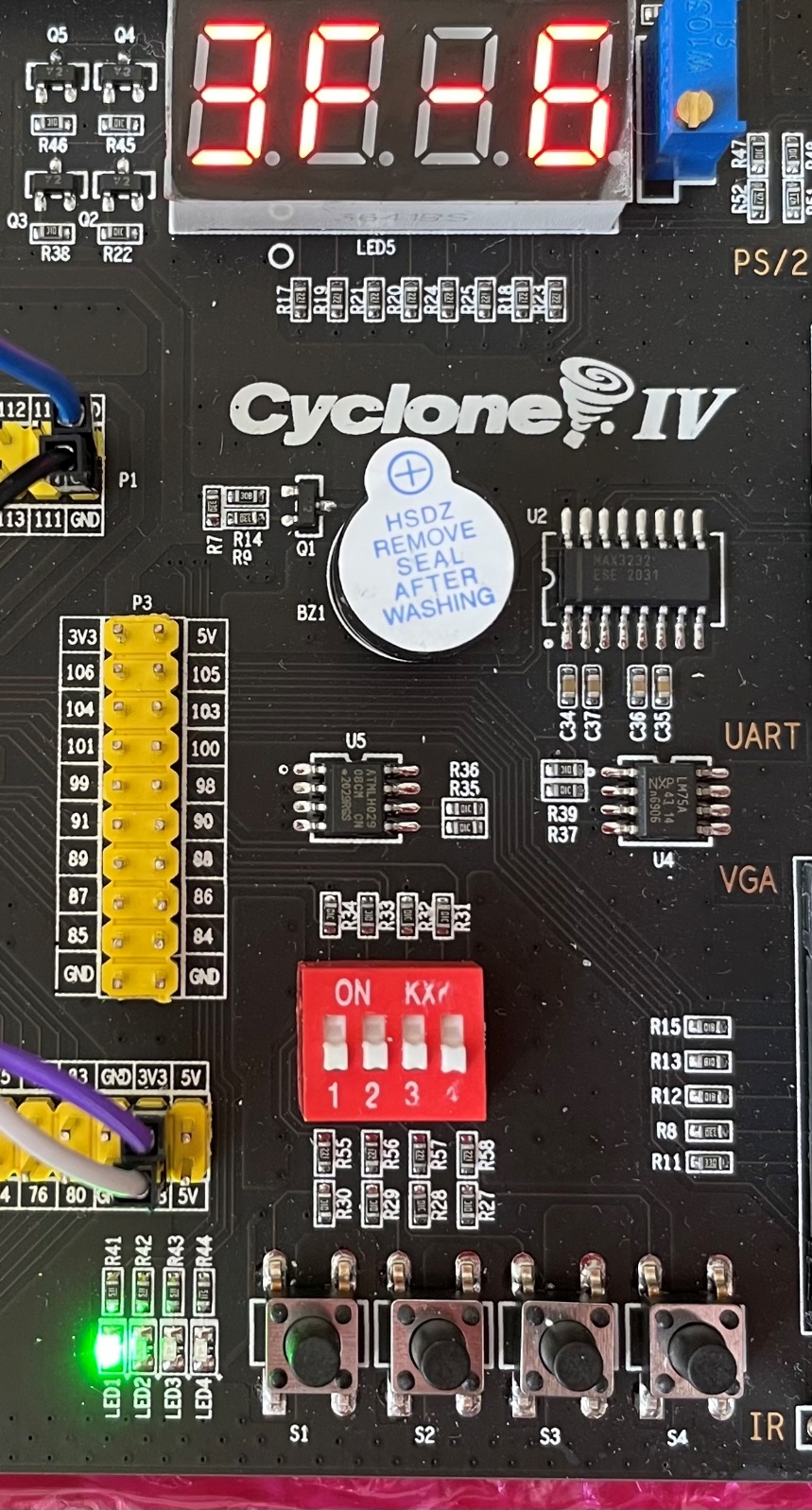
Для успешной выгрузки на плату мы создали проект в Quartus II и доработали программы. Добавили делитель частоты, модуль преобразования результата в код семисегментного индикатора, а также назначили выводы ПЛИС. Результаты макетирования представлены на рисунках ниже.



Распиновка выходных-выходных сигналов



RTL представление устройства



Пример работы на отладочной плате

Вывод: в процессе лабораторной работы были изучены методы построения устройства последовательного анализа разрядов входного слова по типовой схеме алгоритма.

# Приложение А

## Testbench

**module top;**

sys\_bus bus (.\*);

mp\_mult mod\_mult (.dev(bus));

test\_bench prog\_tb (.dev(bus));

**endmodule**

**program test\_bench (sys\_bus.monit dev);**

**class** TEST\_DATA;

rand bit [dev.data\_size : 0] data;

constraint data\_valid { data > 0; }

**endclass;**

TEST\_DATA t\_data;

logic [4:0] takt;

int i = 1;

int results [2][100];

integer f;

initial

begin

f = $fopen("C:/Users/alvas/Documents/Univer/Mursaev/Lab\_5/outputs/output.txt","w");

if (f == 0)

begin

$info ("couldn't open file");

$finish;

end

else

$info ("file opened");

dev.clock = 0;

dev.reset = 0;

dev.in\_data = 8'hfa;

#15;

dev.reset = 1;

#15

dev.reset = 0;

t\_data = new();

assert (t\_data.randomize());

repeat(110)

begin

while(t\_data.randomize() != 1);

dev.in\_data = t\_data.data;

for (takt = 1; takt < 11; takt = takt + 1)

begin

if(takt == 1)

begin

dev.start = 1;

end

else if(takt == 2)

begin

dev.start = 0;

end

#5

dev.clock = 1;

#5

dev.clock = 0;

if(dev.ready)

begin

$fwrite(f,"%0d) %x - %b - %0d\n", i, dev.in\_data, dev.in\_data, dev.result);

end

end

i++;

end

$fclose(f);

$info ("file closed");

end

**endprogram**

## Control

**module control\_unit(clock, reset, start, y, c\_sh\_rg, c\_cnt, ready);**

input clock, reset, start;

input y;

output c\_sh\_rg, c\_cnt, ready;

parameter n=8;

enum {init,cycle} state;

wire clock, reset, start,

y;// analyzing bit of word

reg ready;

reg [1:0]c\_sh\_rg; //control signal for module shift\_reg

reg [1:0] c\_cnt; //control signal for module counter

int k;

always @ (clock, reset)

begin

if (reset)

begin

state = init;

ready = 0;

end

else if (!clock)

case (state)

init:

begin

if (start)

begin

state = cycle;

ready = 0; k = 0;

c\_sh\_rg = 3; // was "1"

c\_cnt = 3; // loading

end

end

cycle:

begin

if (k==n) // control cycles count

begin

ready = 1;

k = 0;

c\_sh\_rg = 0;

c\_cnt = 0;// blocking load bits

state = init;

end

else

begin

//ready = 0;

c\_cnt[1] = y; c\_cnt[0] = 0; // sum/save

c\_sh\_rg = 2; // turn on shift reg

k = k+1;

end

end

endcase;

end

**endmodule**

## Components

**module counter (clk,reset,s,carry\_in, carry\_out,dat\_in,result);**

/\* s=>

00 - no oper

-- 10 -increm for 'bidir'

-- 01 - decrement for 'bidir'

-- 10 or 01 - inc for 'up' and decrement 'dowm'

-- 11 -load \*/

parameter num\_bits=8;

parameter max = num\_bits - 1;

parameter dir= "up" ; // dir := 'up','down','bidir'

input clk,reset, s,carry\_in, dat\_in;

output carry\_out,result;

reg [num\_bits-1:0] par\_out;

wire [num\_bits-1:0] result;

reg carry\_out;

reg [1:0] s;

reg carry\_in;

assign result=par\_out;

always @(result, dir)

if ((dir=="up") && (s!=2'b00) )

carry\_out=(par\_out==max) ? 1:0;

else if ((dir=="down") && (s!=2'b00))

carry\_out=(par\_out==0) ? 1:0;

else if (dir=="bidir")

if (s==2'b10) //up

carry\_out= (par\_out== max) ? 1:0;

else if (s==2'b01) //down

carry\_out= (par\_out== 0) ? 1:0;

always @(posedge clk or posedge reset)

if (reset) par\_out=0;

else if (dir=="up" )

if (s==2'b11) par\_out<=dat\_in;

else if (s==2'b10 || s==2'b01)

begin

par\_out<=par\_out+carry\_in;

end

else if (dir=="down" )

if (s==2'b11) par\_out<=dat\_in;

else if (s==2'b10 || s==2'b01)

begin

par\_out<=par\_out-carry\_in;

end

else if (dir=="bidir" )

if (s==2'b11)par\_out<=dat\_in ;

else if (s==2'b01)

par\_out<=par\_out+carry\_in;

else //(oper="10")

par\_out<=par\_out-carry\_in;

**endmodule**

**module shift\_register (clk,s,dl,dr,par\_inp,result);**

/\* 00 - no oper

-- 10 -shift letf for 'bidir'

-- 01 -shift right for 'bidir'

-- 10 or 01 - shift left for 'left' anf shift right for 'right'

-- 11 -load \*/

parameter num\_bits=8;

parameter shift\_len=1;

parameter dir= "left" ; // dir := 'left','right','bidir'

input clk,s,dr,dl, par\_inp;

output result;

wire [num\_bits-1:0] result;// avoid inoit

wire clk,dr,dl;

wire [1:0] s;

wire[num\_bits-1:0] par\_inp;

reg [num\_bits-1:0] par\_out,par\_outpr;

assign // fix ntw state

result=par\_out;

always @(posedge clk) // prepare new state

if (dir=="left" )

begin

if (s==2'b11) par\_out<=par\_inp;

else if (s==2'b10 || s==2'b01) begin

par\_outpr=par\_out<<shift\_len;

par\_outpr[0]=(shift\_len==1)? dl:0 ;

par\_out<=par\_outpr;

end

end

else if (dir=="right" )

begin

if (s==2'b11) par\_out<=par\_inp;

else if (s==2'b10 || s==2'b01) begin

par\_outpr=par\_out>>shift\_len;

par\_outpr[num\_bits-1]= (shift\_len==1)? dr:0 ;

par\_out<=par\_outpr;

end

end

else if (dir=="bidir" )

begin

if (s==2'b11)par\_out=par\_inp ;

else if (s==2'b01)

begin

par\_outpr=par\_out<<shift\_len;

par\_outpr[0]=(shift\_len==1)? dl:0 ;

par\_out<=par\_outpr;

end

else //(oper="10")

begin

par\_outpr=par\_out>>shift\_len;

par\_outpr[num\_bits-1]= (shift\_len==1)? dr:0 ;

par\_out<=par\_outpr;

end

end // bidir;

**endmodule**

## Sys\_bus

**interface sys\_bus();**

parameter data\_size = 8;

logic clock,reset,start,ready;

logic [data\_size - 1 : 0] in\_data;

logic [data\_size - 1 : 0] result;

modport device (input in\_data, clock, reset, start, output result, ready);

modport monit (output in\_data, clock, reset, start, input result, ready);

**endinterface**

## Mp mult

**module mp\_mult( sys\_bus.device dev);**

reg [dev.data\_size - 1 : 0] yp;

reg [1 : 0] contr\_sh\_reg;

reg [1 : 0] contr\_cnt;

reg cnt\_carry\_out;

counter #(dev.data\_size, dev.data\_size, "up")

cnt (.clk(dev.clock), .reset(dev.reset), .s(contr\_cnt), .carry\_in(1'h1), .carry\_out(cnt\_carry\_out), .dat\_in(1'b0), .result(dev.result));

shift\_register #(dev.data\_size,1,"right") shift\_mod(.clk(dev.clock),.s(contr\_sh\_reg),.dr(1'b0),.dl(1'b0),.par\_inp(dev.in\_data),.result(yp)) ;

control\_unit #(dev.data\_size) control\_mod (.clock(dev.clock), .reset(dev.reset), .start(dev.start), .y(yp[0]), .c\_sh\_rg(contr\_sh\_reg), .c\_cnt(contr\_cnt), .ready(dev.ready));

**endmodule**

## Quartus

**module struct\_l6(clock, i\_reset, i\_start, o\_ready, in\_data, lcd, digits);**

parameter data\_size = 8;

//input logic i\_clock; // input clock

output logic [6:0] lcd; // lcd semisegment pins

output logic [3:0] digits; //control semisegment digits

input logic clock;

input logic i\_start;

input logic i\_reset;

input logic [data\_size - 1 : 0] in\_data;

output logic o\_ready;

logic [data\_size - 1 : 0] result;

logic start;

assign start = ~i\_start;

logic reset;

assign reset = ~i\_reset;

logic ready;

assign o\_ready = ~ready;

mp\_mult #(8) mult (in\_data, clock, reset, start, result, ready);

logic [3:0] ready\_result;

always@ (posedge ready)

begin

ready\_result = result[3:0];

end

lcd lcd\_show (.i\_clock(clock), .data\_dig\_0(in\_data[7:4]), .data\_dig\_1(in\_data[3:0]), .data\_dig\_2(4'h0), .data\_dig\_3(ready\_result), .digits(digits), .lcd\_out(lcd));

**endmodule**

**module mp\_mult( in\_data, clock, reset, start, result, ready);**

parameter data\_size = 8;

input logic clock,reset,start;

output logic ready;

input logic [data\_size - 1 : 0] in\_data;

output logic [data\_size - 1 : 0] result;

reg [data\_size - 1 : 0] yp;

reg [1 : 0] contr\_sh\_reg;

reg [1 : 0] contr\_cnt;

reg cnt\_carry\_out;

counter #(data\_size, data\_size, "up")

cnt (.clk(clock), .reset(reset), .s(contr\_cnt), .carry\_in(1'h1), .carry\_out(cnt\_carry\_out), .dat\_in(1'b0), .result(result));

shift\_register #(data\_size,1,"right") shift\_mod(.clk(clock),.s(contr\_sh\_reg),.dr(1'b0),.dl(1'b0),.par\_inp(in\_data),.result(yp)) ;

control\_unit #(data\_size) control\_mod (.clock(clock), .reset(reset), .start(start), .y(yp[0]), .c\_sh\_rg(contr\_sh\_reg), .c\_cnt(contr\_cnt), .ready(ready));

**endmodule**

**module lcd (i\_clock, data\_dig\_0, data\_dig\_1, data\_dig\_2, data\_dig\_3, digits, lcd\_out);**

input logic i\_clock;

input logic [3:0] data\_dig\_0;

input logic [3:0] data\_dig\_1;

input logic [3:0] data\_dig\_2;

input logic [3:0] data\_dig\_3;

output logic [3:0] digits;

output logic [6:0] lcd\_out;

reg [6:0] led\_inv\_dig\_0;

reg [6:0] led\_inv\_dig\_1;

reg [6:0] led\_inv\_dig\_2;

reg [6:0] led\_inv\_dig\_3;

assign led\_inv\_dig\_2 = 7'b1000000; //symbol "-"

//Понижение частоты для переключения цифр, так как сегменты объединены в одну цепь

reg clk\_led;

generator #(100000) gen\_led (i\_clock, clk\_led);

semisegment seg\_dig\_1 (.number(data\_dig\_1), .result(led\_inv\_dig\_1));

semisegment seg\_dig\_0 (.number(data\_dig\_0), .result(led\_inv\_dig\_0));

semisegment seg\_dig\_3 (.number(data\_dig\_3), .result(led\_inv\_dig\_3));

always@(posedge clk\_led)

begin

if(digits[0] == 1'b0)

begin

digits[0] = 1'b1;

digits[1] = 1'b0;

digits[2] = 1'b1;

digits[3] = 1'b1;

lcd\_out = ~led\_inv\_dig\_1;

end

else if(digits[1] == 1'b0)

begin

digits[0] = 1'b1;

digits[1] = 1'b1;

digits[2] = 1'b0;

digits[3] = 1'b1;

lcd\_out = ~led\_inv\_dig\_2;

end

else if(digits[2] == 1'b0)

begin

digits[0] = 1'b1;

digits[1] = 1'b1;

digits[2] = 1'b1;

digits[3] = 1'b0;

lcd\_out = ~led\_inv\_dig\_3;

end

else

begin

digits[0] = 1'b0;

digits[1] = 1'b1;

digits[2] = 1'b1;

digits[3] = 1'b1;

lcd\_out = ~led\_inv\_dig\_0;

end

end

**endmodule**