ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ
ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«САМАРСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ
УНИВЕРСИТЕТ ИМЕНИ АКАДЕМИКА С.П. КОРОЛЕВА»

Институт информатики и кибернетики

Кафедра радиотехники

Отчет по лабораторной работе

”РАЗРАБОТКА ЦИФРОВЫХ УСТРОЙСТВ НА БАЗЕ ПЛИС – часть II”

Студент: Согонов Е.А.

Преподаватель: Корнилин Д.В.

Группа: 6364-120304D

Самара 2022

СОДЕРЖАНИЕ

1. Структура устройства

В данной лабораторной работе создается устройство, реагирующее на нажатие клавиш и затем в зависимости от клавиши увеличивающее или уменьшающее значение счетчика. Значение счетчика должно отображаться на семисегментном индикаторе, используя принцип динамической индикации. Структурная схема устройства представлена на рисунке 1. На ней показаны основные соединения блоков, за исключением схемы сброса, необходимой при работе с последовательными схемами.

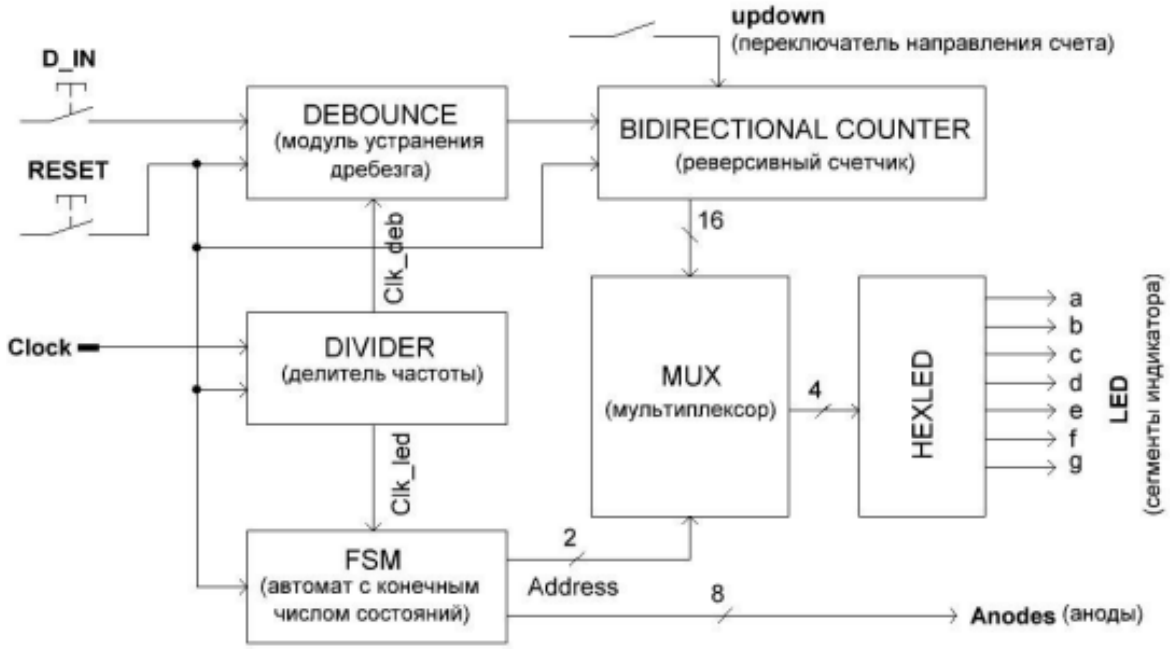

Рисунок 1 – Структурная схема

При проектировании данного устройства использовался принцип иерархического проектирования, где каждый из модулей описывается отдельным описанием устройства на языке VHDL.

Делитель частоты DIVIDER обеспечивает тактовые импульсы для модулей устранения дребезга и системы индикации. Он синхронизируется системным тактовым сигналом (100 МГц), подаваемым внешним осциллятором.

Модуль устранения дребезга DEBOUNCE подавляет паразитные импульсы, которые могут быть вызваны механическими контактами (кнопками). Это особая схема, синхронизируемая сигналом с выхода делителя частоты DIVIDER. Кнопки и ползунковые переключатели не являются частью ПЛИС, они реализованы на отладочной плате.

На реверсивный счетчик BIDIRECTIONAL COUNTER поступают импульсы от модуля устранения дребезга. Реверсивный счётчик увеличивает или уменьшает значение в зависимости от положения ползункового переключателя.

FSM модуль представляет собой автомат с конечным числом состояний (на англ. finite state machine – FSM) для переключения индикаторов и манипуляции адресами в бесконечном цикле. Мы используем автомат Мура с четырьмя состояниями (S0, S1, S2, S3) и переключением между ними. В каждом состоянии FSM устанавливает определенный адрес для мультиплексора и включает соответствующий индикатор.

MUX(мультиплексор) занимается динамической индикацией: на вход получая данные со счетчика и с модуля FSM. FSM включает один из индикаторов, передавая его номер мультиплексору, и мультиплексор выводит соответствующий разряд числа, полученного со счетчика.

Модуль hexled преобразует число, выданное мультиплексором, в последовательность бит, дающих на индикаторе соответствующую цифру.

2. Создание устройства

2.1. Создание схемы устройства

По методике, описанной в методических указаниях к лабораторной работе, в программном пакете Vivado v2016.4 была создана схема, объединяющая ранее описанные модули. После того, как все модули созданы, необходимо соединить их в единую схему. Полученная схема изображена на рисунке 2

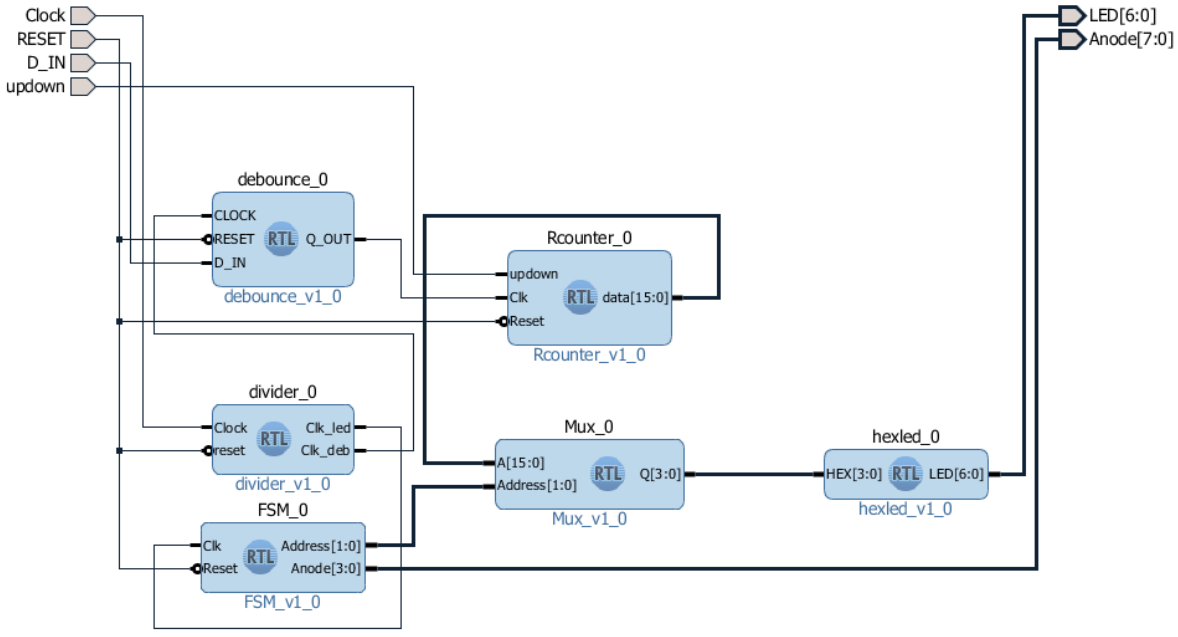

Рисунок 2 – Схема устройства

2.2. Симуляция

Далее был создан файл симуляции «simm» подобно тому, как делалось это в I части лабораторной работы. Для того, чтобы просимулировать поведение

устройства при всех возможных событиях, стоит немного изменить файл симуляции, приведенный в методических указаниях. Он примет вид:

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
entity simm is
end simm;
architecture Behavioral of simm is
component schematic_wrapper is
port (Anode : out STD_LOGIC_VECTOR ( 3 downto 0 );
   Clock : in STD_LOGIC;
   D_IN : in STD_LOGIC;
   LED : out STD_LOGIC_VECTOR ( 6 downto 0 );
   reset : in STD_LOGIC;
   updown : in STD_LOGIC);
end component schematic_wrapper;
signal Reset : STD_LOGIC := '0';
signal D_IN : STD_LOGIC := '0';
signal updown : STD_LOGIC := '0';
signal Clock : STD_LOGIC := '0';
signal Anode : STD_LOGIC_VECTOR (3 downto 0) := B"1111";
signal LED : STD_LOGIC_VECTOR (6 downto 0);
begin
insta: schematic_wrapper
PORT MAP(
   Anode(3 downto 0) => Anode(3 downto 0),
   Clock => Clock, D_IN => D_IN,
   LED(6 downto 0) => LED(6 downto 0),
   reset => reset, updown => updown);
clk: process
begin
   Clock <= '0'; wait for 1 ns;
   Clock <= '1'; wait for 1 ns;
end process;
reseting : process--создание тестового сигнала(симуляция нажатия на кнопку reset)
begin
   reset <= '0'; wait for 1 ns;
   reset <= '1'; wait for 5 ns;
   reset <= '0'; wait for 15 ms;
end process;
updowning : process--создание тестового сигнала(симуляция
-- переключения свитча-переключателя направления счета )
begin
   updown <= '0'; wait for 1 ms;
   updown <= '1'; wait for 2 ms;
end process;
tb : process--создание тестового сигнала(симуляция
 -- нажатий на кнопку D_in)
begin
   D_IN <= '0'; wait for 1 ms;
   D_IN <= '1'; wait for 50 ns;
   D_IN <= '0'; wait for 50 ns;
   D_IN <= '1'; wait for 50 ns;
   D_IN <= '0'; wait for 50 ns;
   D_IN <= '1'; wait for 50 ns;
   D_IN <= '0'; wait for 50 ns;
   D_IN <= '1'; wait for 50 ns;
end process;
end Behavioral;
```

Были получены временные диаграммы, показанные на рисунках 3, 4, 5, 6.

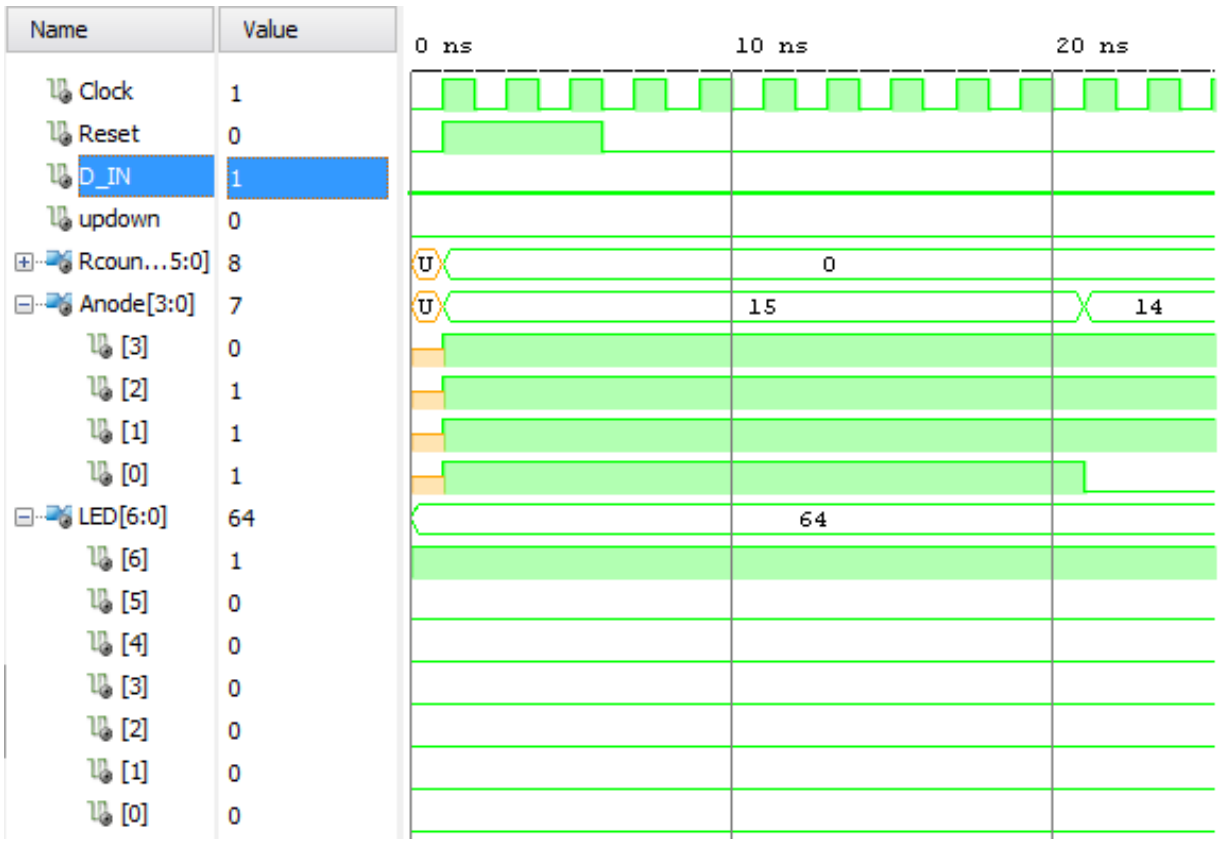

Рисунок 3 – Сброс в начальный момент времени

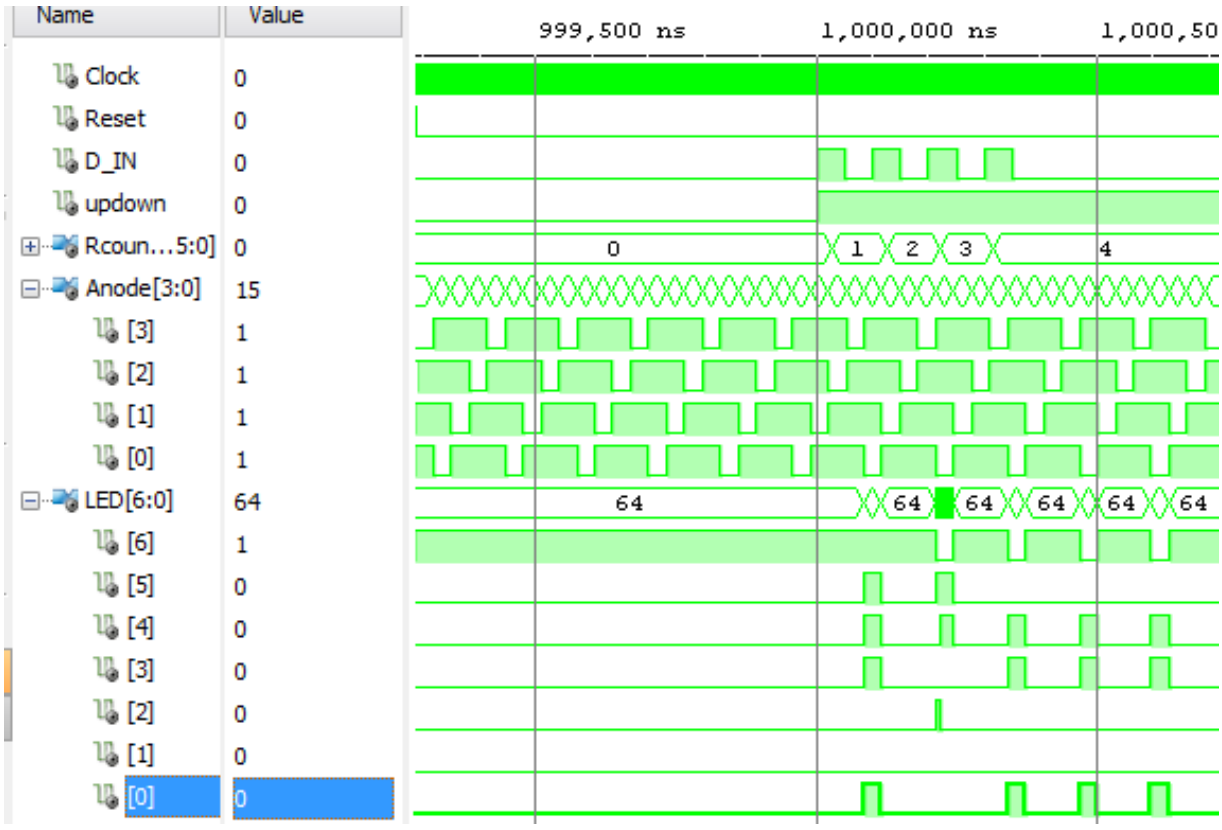

Рисунок 4 – Увеличение значения счетчика, при updown=’1’

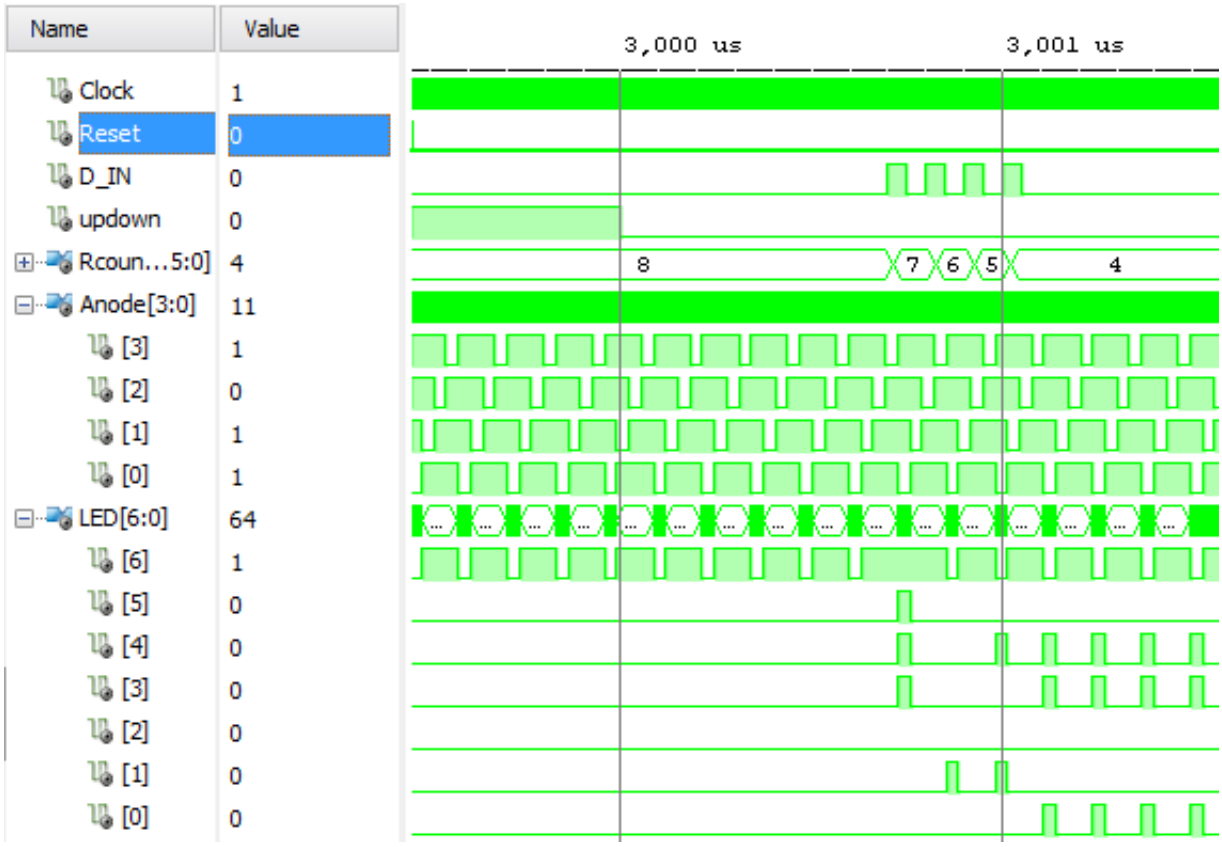

Рисунок 5 – Уменьшение значения счетчика, при updown=’0’

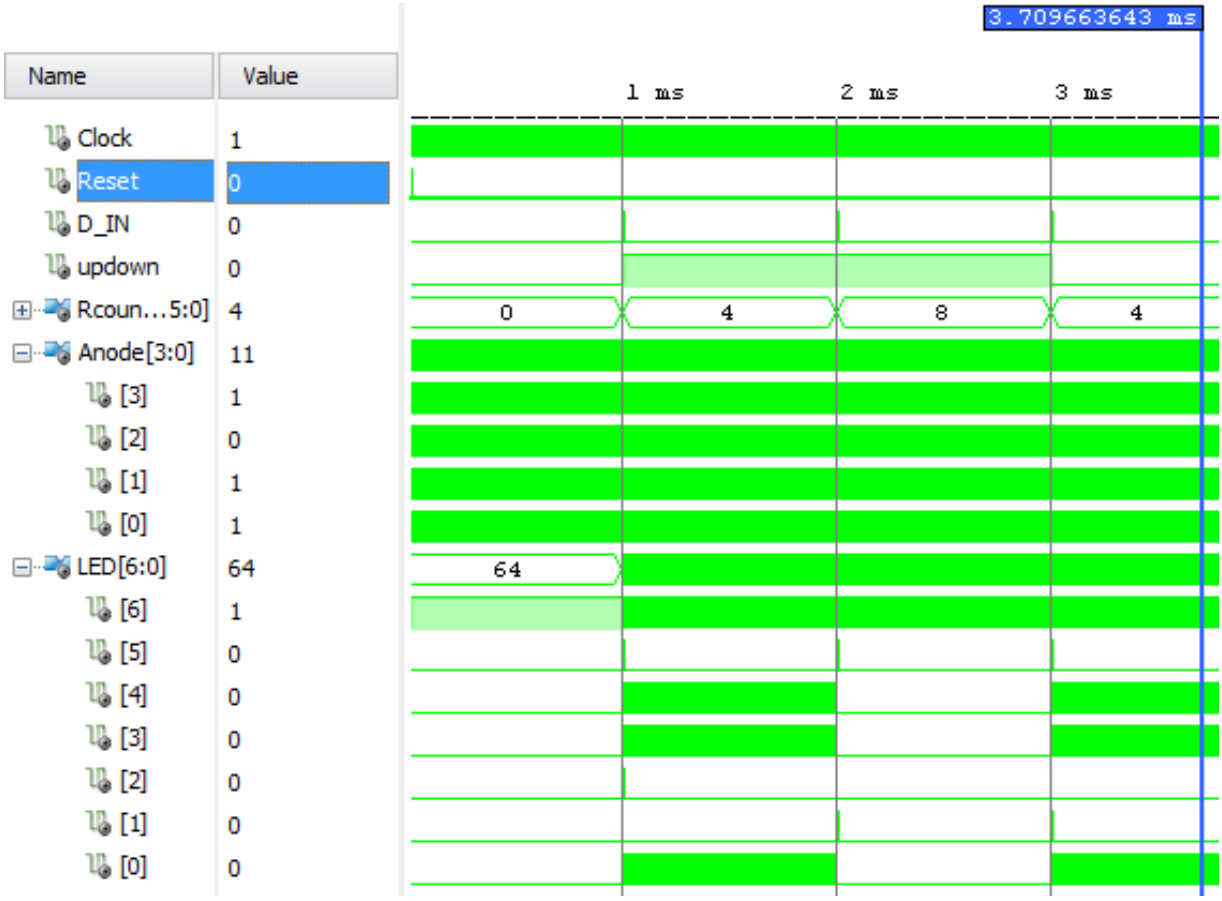

Рисунок 6 – Более наглядная иллюстрация

Таким образом, с помощью вышеописанного файла симуляции была проверена работоспособность устройства. Временные диаграммы показывают, что требуемые функции работают.

2.3. Синтез

В этом разделе необходимо задать временные ограничения, создать асинхронные группы, и прочие манипуляции, описанные в методических указаниях, добиваясь отсутствия ошибок, связанных с THS и TNS.

В результате синтеза, после исправления была получена следующая схема и сводка по времени(рисунок 8) Так же на этом этапе необходимо указать внешние порты ввода-вывода для размещения схемы внутри ПЛИС. Сделать это нужно вручную.

I/O Ports

Name	Direction	N...	Package Pin	Fixed	Bank	I/O Std	Vcco	Vref	Driv...	Slew Type	Pull T...	Off-Chip Termination	IN_TERM
All ports (15)													
Clock_43962 (1)	IN			☑	35	LVCMOS33*	3.300				NONE	NONE	
Scalar ports (1)													
Clock	IN		E3	☑	35	LVCMOS33*	3.300				NONE	NONE	
RESET_Reset...	IN			☑	14	LVCMOS33*	3.300				NONE	NONE	
Scalar ports (1)													
reset	IN		T16	☑	14	LVCMOS33*	3.300				NONE	NONE	
Anode (4)	OUT			☑	34	LVCMOS33*	3.300		12	SLOW	NONE	FP_VTT_50	
Anode[3]	OUT		N5	☑	34	LVCMOS33*	3.300		12	SLOW	NONE	FP_VTT_50	
Anode[2]	OUT		M3	☑	34	LVCMOS33*	3.300		12	SLOW	NONE	FP_VTT_50	
Anode[1]	OUT		M6	☑	34	LVCMOS33*	3.300		12	SLOW	NONE	FP_VTT_50	
Anode[0]	OUT		N6	☑	34	LVCMOS33*	3.300		12	SLOW	NONE	FP_VTT_50	
LED (7)	OUT			☑	34	LVCMOS33*	3.300		12	SLOW	NONE	FP_VTT_50	
LED[6]	OUT		L6	☑	34	LVCMOS33*	3.300		12	SLOW	NONE	FP_VTT_50	
LED[5]	OUT		M2	☑	34	LVCMOS33*	3.300		12	SLOW	NONE	FP_VTT_50	
LED[4]	OUT		K3	☑	34	LVCMOS33*	3.300		12	SLOW	NONE	FP_VTT_50	
LED[3]	OUT		L4	☑	34	LVCMOS33*	3.300		12	SLOW	NONE	FP_VTT_50	
LED[2]	OUT		L5	☑	34	LVCMOS33*	3.300		12	SLOW	NONE	FP_VTT_50	
LED[1]	OUT		N1	☑	34	LVCMOS33*	3.300		12	SLOW	NONE	FP_VTT_50	
LED[0]	OUT		L3	☑	34	LVCMOS33*	3.300		12	SLOW	NONE	FP_VTT_50	
Scalar ports (2)													
D_IN	IN		R10	☑	14	LVCMOS33*	3.300				NONE	NONE	
updown	IN		U9	☑	34	LVCMOS33*	3.300				NONE	NONE	

Рисунок 7 – Расположение портов ввода-вывода

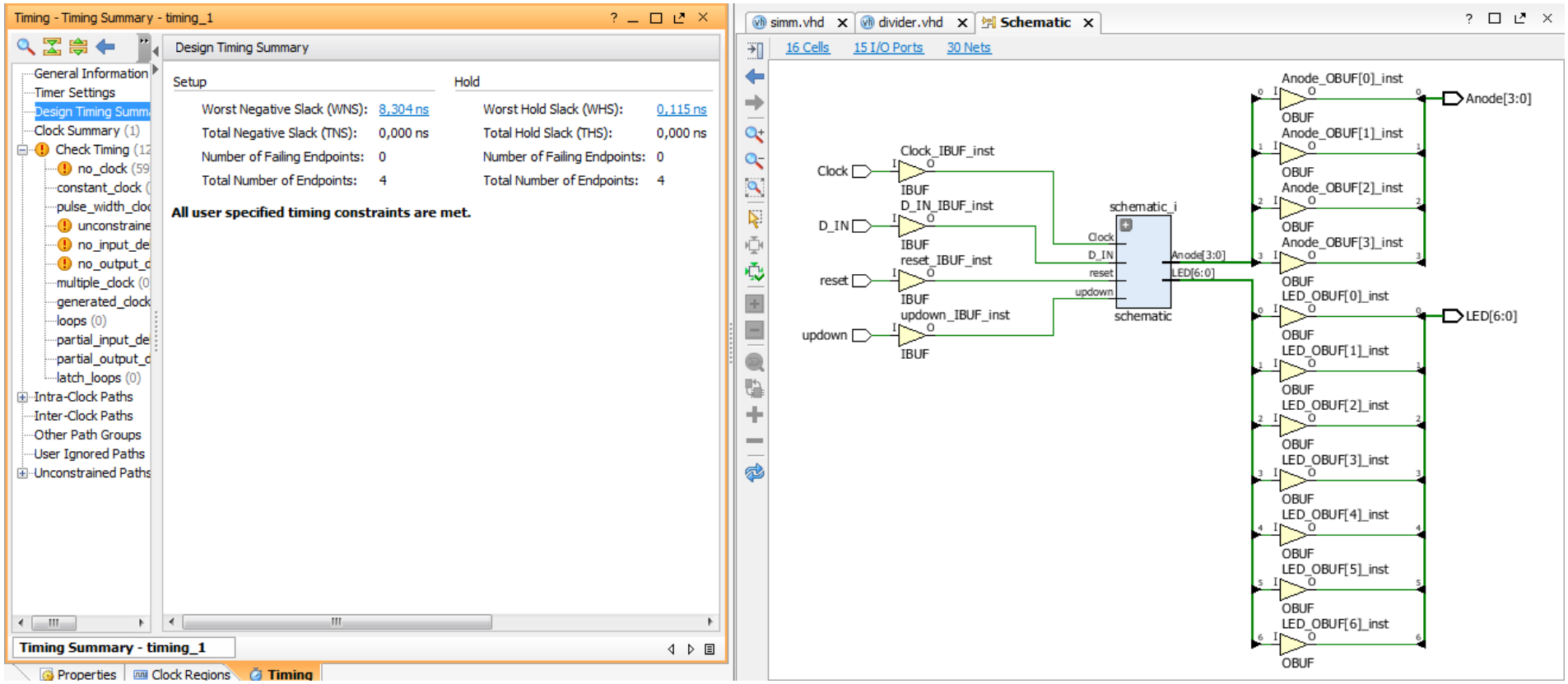

Рисунок 8 – Синтезированная схема и сводка по времени

3. Реализация

Следующий шаг разработки устройства – это реализация. После завершения процесса реализации нужно проверить выполнение временных ограничений (Report Timing Summary)(рисунок 9)

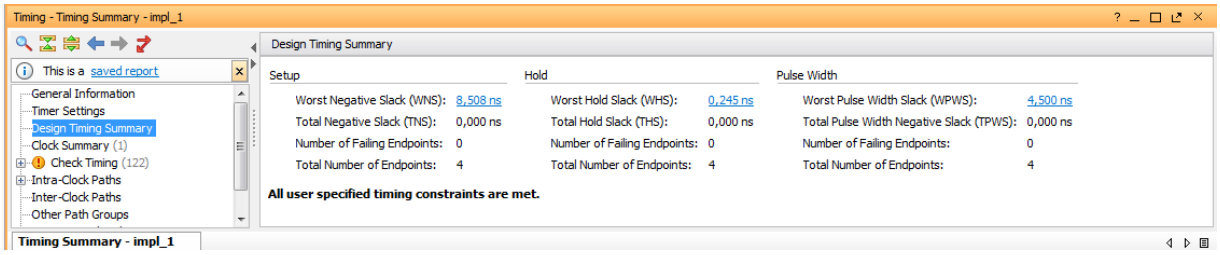

Рисунок 9 – временная сводка после реализации

4. Программирование ПЛИС

Для программирования ПЛИС нужно выбрать Generate Bitstream на левой панели (Flow Navigator), а после завершения процесса открыть Hardware Manager, подключить кабель USB к отладочной плате (разъем PROG) и затем к компьютеру. Включить плату (переключатель POWER на плате), а также проверить правильность установки перемычек JP1 и JP2.

В окне Hardware Manager необходимо выбрать ”Open Target”, а затем ”Auto Connect” Если соединение будет успешным, надпись ”unconnected” сменится на имя платы, появится кнопка ”Program Device”. При нажатиии на нее нужно выбрать отладочную плату, а затем файл с битовым потоком (с расширением .bit) – ”Bitstream file” . Нажать ”Program”.

После завершения программирования устройства была произведена проверка его работоспособности, в результате которой было выяснено, что устройство работает корректно. Функции, возложенные на кнопки reset, D_IN и updown правильно работают.

5. ОТВЕТЫ НА ВОПРОСЫ ДЛЯ САМОПРОВЕРКИ

1. Что такое иерархическое проектирование и в чем его преимущества?

Иерархические структуры могут упростить процесс разработки и разделить его между несколькими разработчиками. Несколько членов команды могут работать одновременно и независимо над разными частями устройства. Каждая часть может быть отлажена по отдельности и стать частью более сложной схемы.

2. Пользуясь рисунком 1, поясните работу разрабатываемого устройства.

Данный вопрос был прояснен в тексте работы выше.

3. Как работает семисегментный индикатор? Что такое статическая и динамическая индикация?

Индикатор состоит из семи отдельно управляемых (подсвечиваемых светодиодом) элементов - сегментов. Эти элементы позволяют отобразить любую цифру 0..9, а также некоторые другие символы, например: ’-’, ’A’, ’b’, ’C’, ’d’, ’E’, ’F’ и другие. Это даёт возможность использовать индикатор для вывода положительных и отрицательных десятичных и шестнадцатеричных чисел и даже текстовых сообщений. Обычно индикатор имеет также восьмой элемент - точку, используемую при отображении чисел с десятичной точкой.

При статической индикации выводы индикатора подключены к устройству независимо друг от друга и информация на них выводится постоянно. Этот способ управления проще динамического, но без использования дополнительных элементов, подключить многоразрядный семисегментный индикатор к устройству будет проблематично - может не хватить выводов.

Динамическая индикация подразумевает поочередное зажигание разрядов индикатора с частотой, не воспринимаемой человеческим глазом. Схема подключения индикатора в этом случае на порядок экономичнее благодаря тому, что одинаковые сегменты разрядов индикатора объединены.

В нашем случае использовался принцип динамической индикации.

4. Для чего необходим модуль устранения дребезга?

Дребезг контактов это явление, происходящее в электромеханических коммутационных устройствах и аппаратах, длящееся некоторое время после замыкания электрических контактов. После замыкания происходят многократные неконтролируемые замыкания и размыкания контактов за счёт упругости материалов и деталей контактной системы - некоторое время контакты отскакивают друг от друга при соударениях, размыкая и замыкая электрическую цепь.

Модуль устранения дребезга призван избавить дальнейшие цепи от этих многократных замыканий и размыканий, выдавая на выходе чистое нажатие.

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
entity debounce is
   Port ( CLOCK,  RESET,D_IN : in STD_LOGIC; Q_OUT : out STD_LOGIC);
end debounce;
architecture Behavioral of debounce is
signal Q1, Q2, Q3 : std_logic;
begin
process(clock)
begin
  if (clock'event and clock = '1') then --по переднему фронту тактового сигнала
    if (reset = '1') then -- сброс
      Q1 <= '0';
      Q2 <= '0';
      Q3 <= '0';
    else
      Q1 <= D_IN;
      Q2 <= Q1;
```

```
        Q3 <= Q2;
      end if;
    end if;
end process;
-- как это работает? пока нажатие/отжатие не завершилось, происходят многократные
--случайные замыкания и размыкания, таким образом D_IN изменчиво
--выход первого триггера становится входом для второго, аналогично для третьего.
--первый отстает от второго на один такт, так же как и третий от второго
--поэтому, в момент, когда кнопка дребезжит, Q1 не равно Q2
--когда состояние кнопки установилось, за время одного такта Q1
--станет равно Q2, но Q3 отстает на такт, и еще имеет противоположное состояние.
-- Именно это свидетельствует о том, что кнопка нажата. В этот момент происходит
--такт на выводе Q_OUT. Дальше Q3 может тоже перещелкнуться в противоположное
-- состояние, но это уже не будет соответствовать такту Q_OUT
--Ниже будут показаны временные диаграммы
Q_OUT <= Q1 and Q2 and (not Q3);
end Behavioral;
```

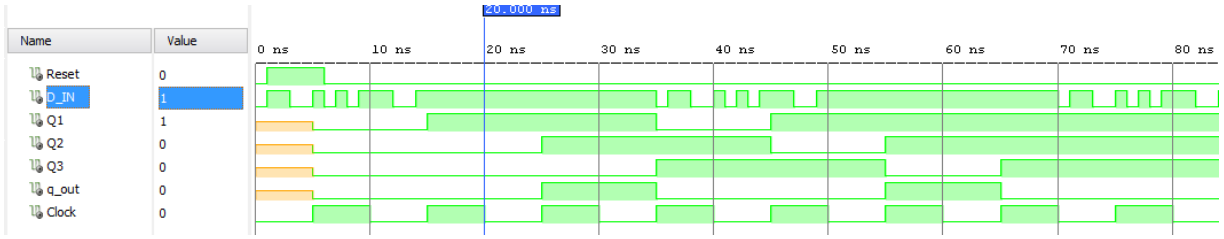

Рисунок 10 – временная диаграмма модуля устранения дребезга -1

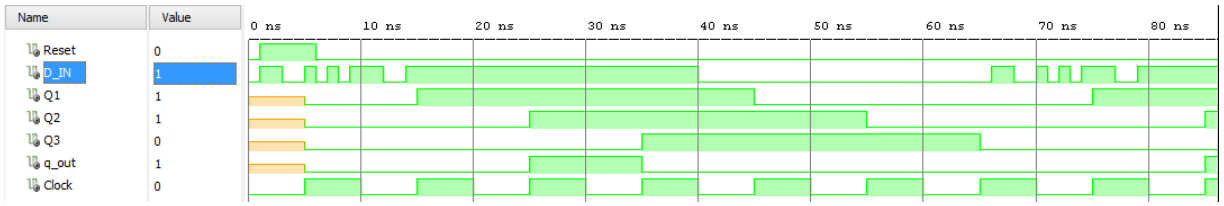

Рисунок 11 – временная диаграмма модуля устранения дребезга -2

5. Объясните работу делителя частоты. Как рассчитать частоту делителя?

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
--декларация entity для элемента "и"
entity ANDE is
port (
            X1,X2 : in STD_LOGIC;
            Y : out STD_LOGIC
            );
end ANDE;
-- Логика работы элемента "и"
architecture ANDA of ANDE is
begin
      Y<=X1 and X2;
end ANDA;

```

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
--декларация entity для триггера JK-типа
entity JFF is
port (
	J,C,R : in STD_LOGIC;
	Q : inout STD_LOGIC
);
end JFF;
-- Логика работы элемента "и"
architecture JK of JFF is
begin
process (C,R)
begin
	if R='1' then Q <= '0';
	elsif (C'event and C='1') then
		if J='1' then Q <= not Q;
		end if;
	end if;
end process;
end JK;

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
-- декларация entity для делителя частоты
entity divider is
generic (Nd : integer := 24);--объявление переменной Nd.
-- Эта переменная определяет, сколько будет создано ступеней делителя с помощью конструкции generate
Port ( Clock, reset: in STD_LOGIC;Clk_led,Clk_deb: out STD_LOGIC;);
end divider;

architecture Behavioral of divider is
 -- декларация компонентов, используемых в архитектуре делителя
component JFF
port ( J,C,R : in STD_LOGIC; Q : inout STD_LOGIC);
end component;
component ANDE is
port (X1,X2 : in STD_LOGIC; Y : out STD_LOGIC);
end component;

signal T,V: STD_LOGIC_VECTOR(0 to Nd);

begin
T(0)<='1';
Clk_deb<=V(18);--подключение выходов к необходимой ступени делителя
Clk_led<=V(19);
ST0: JFF --первый триггер JK-типа
port map(
	J=>T(0),
	C=>Clock,
	R=>reset,
	Q=>V(1)
);
JK1: for i in 1 to Nd-1 generate -- Генерация Nd-1 ступеней, состоящих из триггеров и элементов "и"
	begin
	ST1: ANDE
	port map(
		X1=>T(i-1),
		X2=>V(i),
		Y=>T(i)
```

```
76          );
77          ST2: JFF
78          port map(
79                J=>T(i),
80                C=>Clock,
81                R=>reset,
82                Q=>V(i+1)
83          );
84    end generate;
85    end Behavioral;
```

Наглядно работу данного описания можно объяснить, синтезировав его при разных значениях переменной Nd. Соответствующие рисунки будут ниже. Изменим строки 42 и 60-61 из вышеописанного кода:

```
------------------------
generic (Nd : integer := 2);--объявление переменной Nd.
------------------------
Clk_deb<=V(2);--подключение выходов к необходимой ступени делителя
Clk_led<=V(1);
---------------------
```

Nd - переменная, хранящая в себе количество итераций для конструкции generate. Две другие строки отвечают за расположение подключения выводов Clk_deb и Clk_led.

Если синтезировать описание с такими измененными значениями, получим простейший делитель, который делит на четыре (рисунок 12). Причем вывод Clk_deb делится на 4, а Clk_led делится на два, так как подключены к выходам разных триггеров.

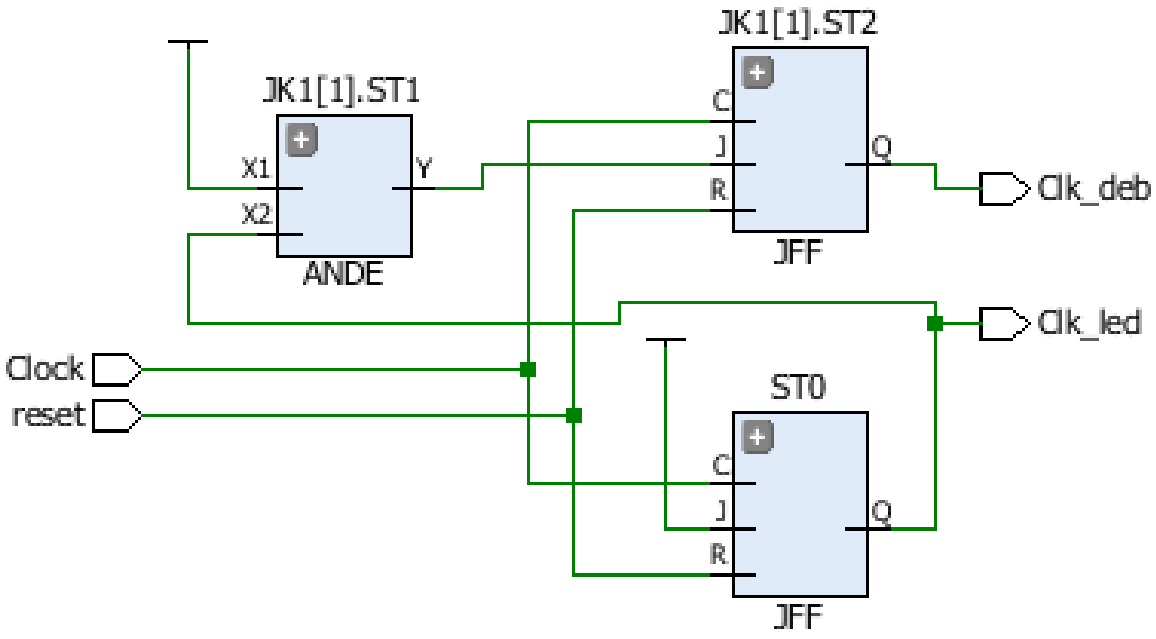

Рисунок 12 – делитель в минимальной комплектации

Снова изменив эти коэфициенты:

```
------------------------
generic (Nd : integer := 3);--объявление переменной Nd.
------------------------
Clk_deb<=V(3);--подключение выходов к необходимой ступени делителя
Clk_led<=V(1);
---------------------
```

Если синтезировать описание с такими измененными значениями, получим делитель, который делит на восемь (рисунок 12). Причем вывод Clk_deb делится на 8, а Clk_led все еще делится на два.

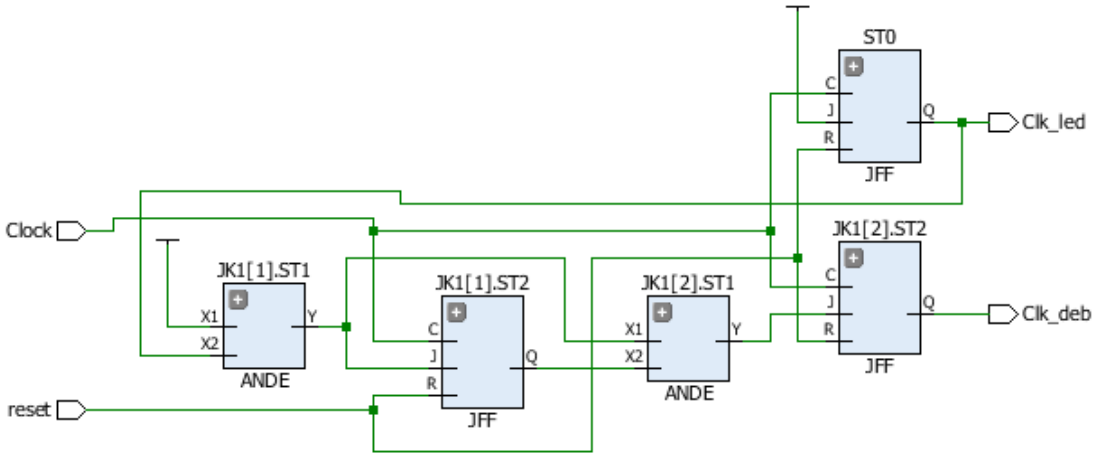

Рисунок 13 – Clk_deb делим на 8, Clk_led делим на 2

Аналогично получаем такую схему

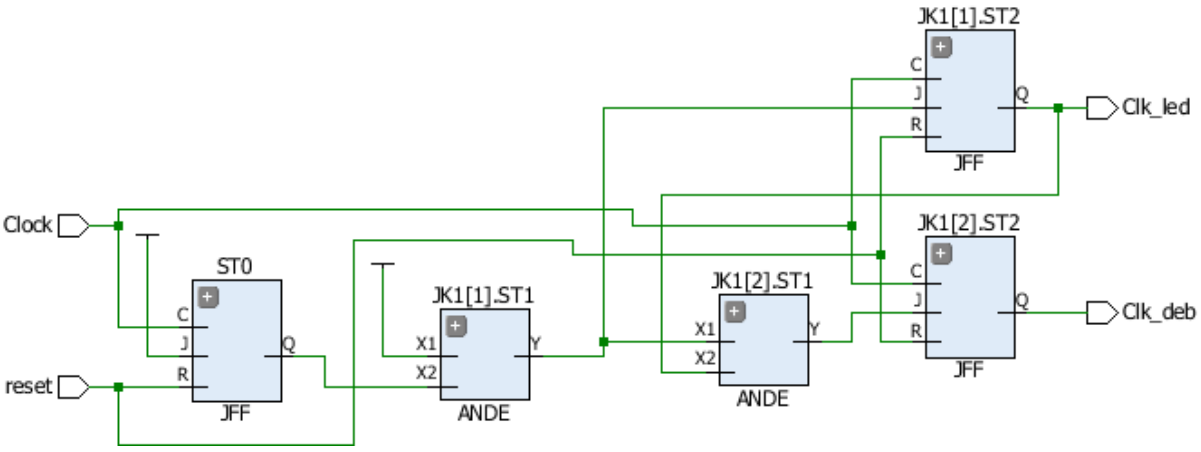

Рисунок 14 – Clk_deb делим на 8, Clk_led делим на 4

Если вернуть значения на изначальные, будет синтезирована такая схема(фрагмент показан на рисунке 15). Видно, что синтезировано 24 пары триггеров и элементов ”и”, выводы Clk_deb и Clk_led подключены соответственно к 18 и 19 ступени.

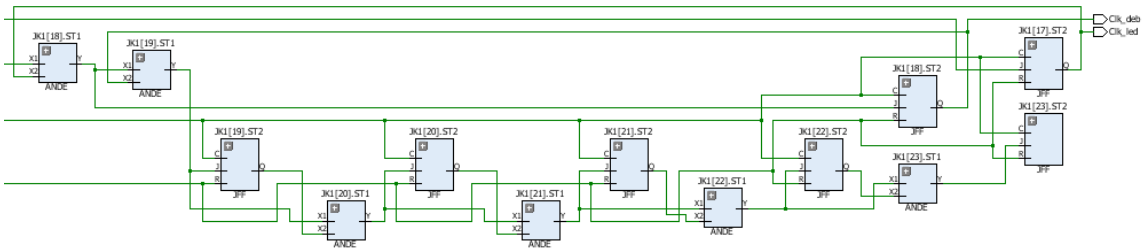

Рисунок 15 – Схема деления на 2^{18} и 2^{19}

6. Опишите строки кода в файле Rcounter.vhd.

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
entity Rcounter is
 Port (updown, Clk, Reset : in STD_LOGIC; --описываем входы и выходы модуля
       data : out STD_LOGIC_VECTOR (15 downto 0)
        );
end Rcounter;
architecture Behavioral of Rcounter is

begin
process (Clk,reset)
variable D : STD_LOGIC_VECTOR(15 downto 0);
--переменная, в которую записывается текущее значение счетчика
begin
      if reset='1' then D := (others => '0'); -- если нажата кнопка перезагрузки, сбросить счетчик в ноль
      elsif Clk='1' and Clk'event then --если не нажата кнопка перезагрузки, то
            if updown ='1' then D := D + 1; --если updown='1', то счетчик суммирует
                  else D := D - 1; --иначе - вычитает
            end if;
      end if;
      data <= D;--считать значение переменной и подать его на выход data
end process;
end Behavioral;
```

7. Объясните работу модуля FSM.

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
entity FSM is
Port (Clk, Reset : in STD_LOGIC;
            Address : out STD_LOGIC_VECTOR (1 downto 0);
            Anode : out STD_LOGIC_VECTOR (3 downto 0)
);
end FSM;

architecture Behavioral of FSM is
type statetype is (S0,S1,S2,S3); --тип statetype(перечислимый),
--значениями которого являются состояния цифрового автомата
signal state : statetype; --сигнал (state) этого перечислимого типа,
-- в котором будет храниться текущее состояние автомата.

begin
process(Clk,Reset)
begin
```

```
    if Reset='1' then --сброс при нажати reset
     state <= S0;
     Anode <= B"1111";
    elsif rising_edge(Clk) then --по переднему фронту тактового сигнала
        case state is -- оператор case, где селектор - состояние автомата из state
            when S0 => --когда состояние S0
                Address <= B"00"; --сигнал address принимает значение B"00"
                --это код для мультиплексора, означающий, что зажжен первый индикатор
                Anode <= B"1110";--непосредственное включение первого индикатора
                --подачей нуля на анод индикатора
                state <= S1;--переключение на следующее состояние
                --остальные состояния работают аналогично
            when S1 =>
                Address <= B"01";
                Anode <= B"1101";
                state <= S2;
            when S2 =>
                Address <= B"10";
                Anode <= B"1011";
                state <= S3;
            when S3 =>
                Address <= B"11";
                Anode <= B"0111";
                state <= S0;
        end case;
    end if;
end process;
end Behavioral;
```

8. Посмотрите код в файлах divider.vhd и fsm.vhd и скажите, с какой частотой мерцает каждый отдельный семисегментный индикатор.

Модуль fsm.vhd управляет анодами семисегментных индикаторов и мультиплексором, включающих нужные для индикации катоды. Мерцание индикаторов обуславливается именно переключением напряжения на их анодах, то есть, это зависит только от модуля fsm.vhd

fsm.vhd тактируется сигналом, получаемого из модуля divider.vhd, имеющего частоту $100\text{MHz}/2^{19} \approx 190{,}734\text{Hz}$. Именно с такой частотой переключаются индикаторы.

9. Прокомментируйте результаты моделирования (рисунок 10 из методических указаний).

В ходе работы были подробно прокомментированы результаты моделирования. Рисунок 10 из методических указаний свидетельствует лишь о том, что сброс происходит, аноды индикатора последовательно переключаются, сигнал D_in не влияет ни на что, так как updown=’0’

10. Обоснуйте выбор портов ввода/вывода. Почему вы назначали именно эти выводы?

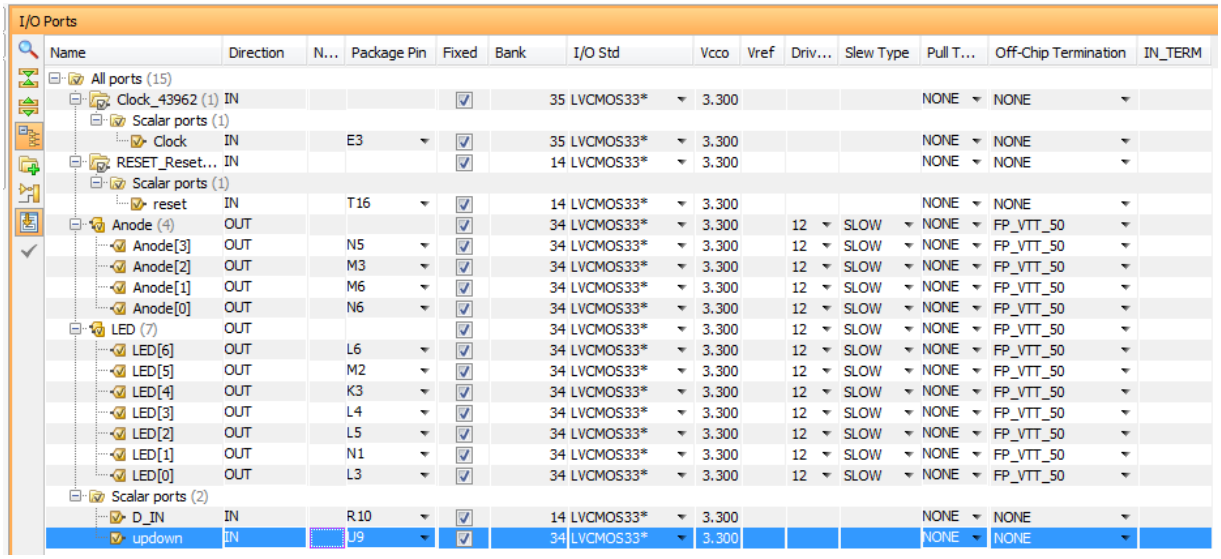

Рисунок 16 – Расположение портов ввода-вывода

Эти выводы назначены были в соответствии со схемой выводов, взятой из даташита на отладочную плату(рисунок 17)

Конструкцией отладочной платы жестко прописаны порты для тактового сигнала, для катодов и анодов семисегментного индикатора. Переключатель для updown можно было выбрать из 15 доступных на плате, кнопки reset и D_in можно было выбрать из 5 доступных.

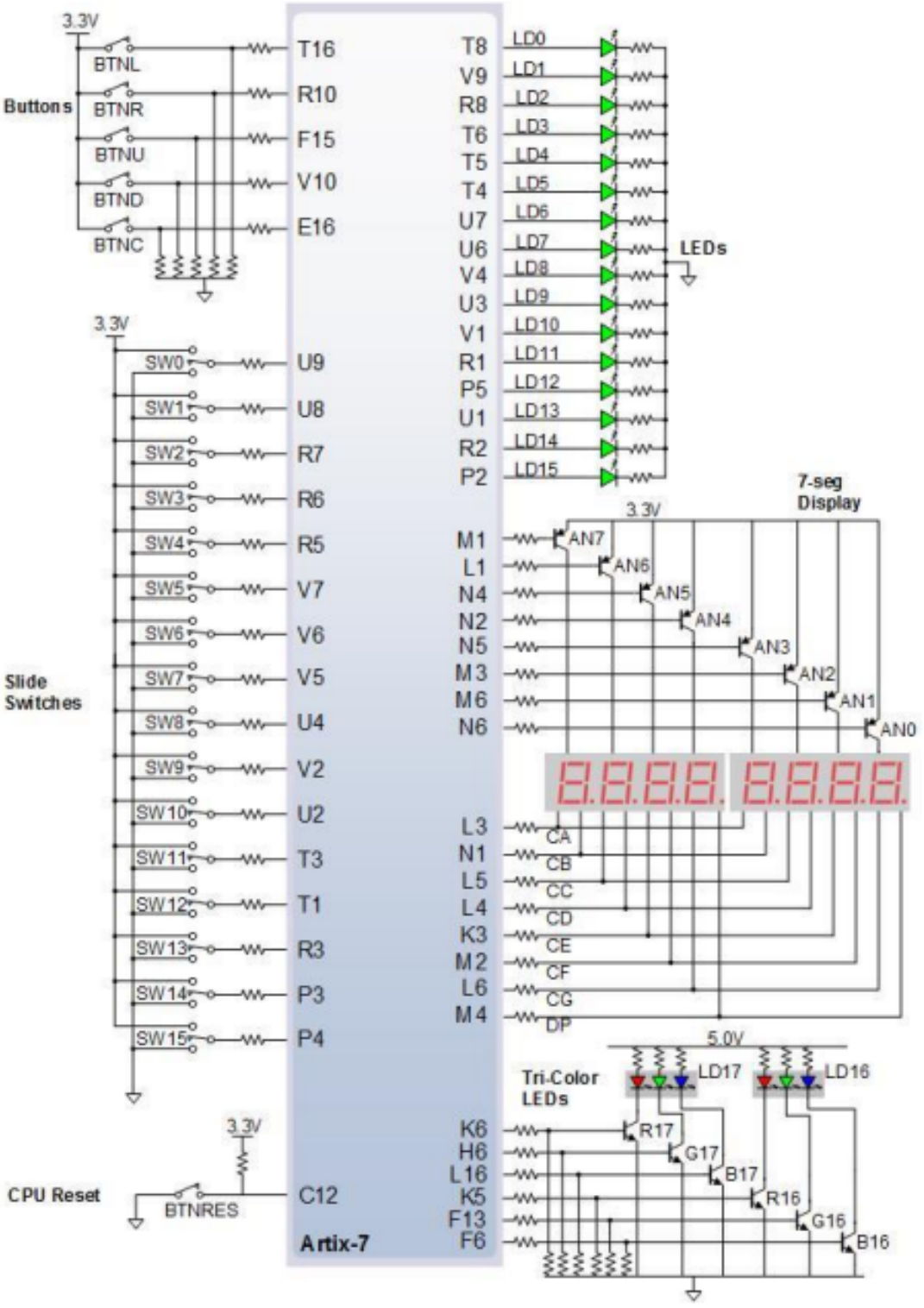

Figure 16. General Purpose I/O devices on the Nexys4

Рисунок 17 – Фрагмент даташита на отладочную плату

11. Опишите процесс подключения отладочной платы к компьютеру и ее подготовки к программированию ПЛИС

Этот процесс был описан в тексте работы выше.