Міністерство освіти і науки, молоді та спорту України

Національний аерокосмічний університет ім. М.Є. Жуковського

Кафедра 503

Лабораторна робота № 3

з дисципліни

«Компонентно - орієнтоване проектування»»

Тема: «ОПИСАНИЕ ПОЛЬЗОВАТЕЛЬСКОГО КОМПОНЕНТА НА ЯЗЫКЕ VHDL ДЛЯ СИСТЕМ КЛАССА “SYSTEM-ON-CHIP”»

Варіант № 2

|  |  |  |  |
| --- | --- | --- | --- |
|  |  | Виконала: | ст. гр. 535а |
|  |  |  | Веприцька О.Ю. |
|  |  | Перевірив: | к.т.н. доцент |
|  |  |  | Перепелицин А.Є. |

#### Харків 2019

**Цель работы -** описание пользовательского компонента (модулей в составе компонента) с использованием языка VHDL.

**Задание**

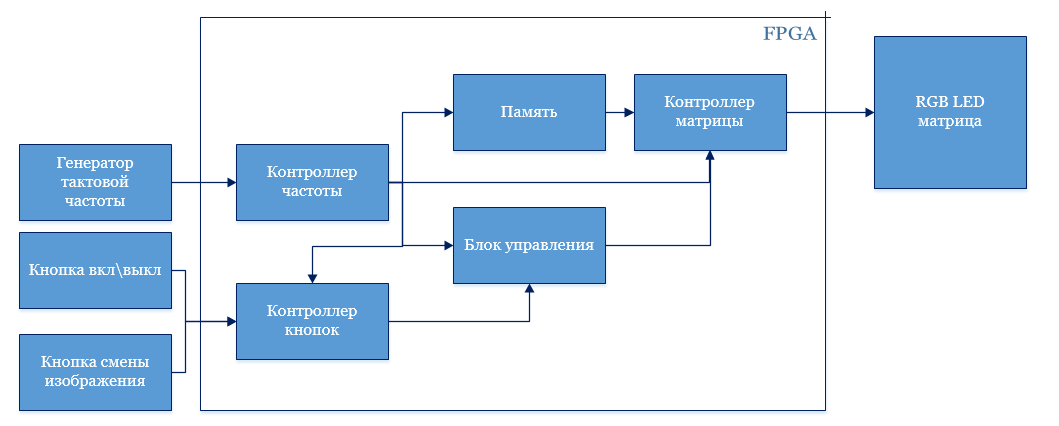
Согласно заданию, утвержденному по итогам лабораторной работы №1, а также полученной структуре пользовательского компонента (с детализированным интерфейсом каждого из модулей в составе компонента на уровне описания сущностей), необходимо получить конечную реализацию проекта на языке описания аппаратуры VHDL. Для полученного описания необходимо предоставить (разработать) модуль тестирования (testbench-модуль), который бы подтверждал корректность работы компонента (согласно спецификации). В качестве среды моделирования необходимо использоваться ModelSim.

1. Общее описание компонента, его назначение и предполагаемая (предварительная) характеристика.

Разработка системы динамической индикации RGB LED матрицы в FPGA.

Предусматривается наличие двух кнопок для выключения и включения; смены изображения. Вывод осуществляется при помощи RGB LED матрицы.

1. Общая структура компонента (схема /рисунок) со всеми необходимыми обозначениями (характер связей и т.д.).



1. Описание модулей системы, портов ввода /вывода и др.
2. Кнопка вкл/выкл – предназначена для выключения и включения матрицы.
3. Кнопка смены изображения – предназначена для изменения изображения.
4. Контроллер кнопок – выполняет функции приема сигналов от кнопок и передачи их управляющему блоку.
5. Блок управления – выполняет действия в зависимости от полученных данных от контроллера кнопок и передает их контроллеру матрицы .
6. Память – выполняет функцию временного хранения данных.
7. Генератор тактовой частоты – источник тактирования, предназначен для генерации электрических импульсов заданной частоты.
8. RGB LED matrix – матрица для вывода изображения.
9. Контроллер частоты – электрическое устройство для изменения частоты электрического тока (напряжения);
10. Контроллер матрицы – получает данные из памяти и сигналы от блока управления, преобразовывает их и передает в RGB LED матрице.

**Контроллер частоты**

**library** IEEE;

**use** IEEE.std\_logic\_1164.**all**;

**use** ieee.std\_logic\_arith.**all**;

**use** ieee.std\_logic\_unsigned.**all**;

**entity** frequency\_converter **is**

**generic**(

*INPUT\_CLK* : INTEGER := 50\_000\_000;

*OUT\_CLK* : INTEGER := 2

);

**port**(

clk\_in : **in** STD\_LOGIC;

clk\_out : **out** STD\_LOGIC

);

**end** frequency\_converter;

**architecture** logic **of** frequency\_converter **is**

**signal** temporal : STD\_LOGIC := '0';

**begin**

frequency\_divider : **process**(clk\_in)

**constant** *max\_counter\_v* : natural := (*INPUT\_CLK* / (*OUT\_CLK* \* 2)) - 1;

**variable** counter : natural **range** 0 **to** *max\_counter\_v* := 0;

**begin**

**if** rising\_edge(clk\_in) **then**

**if** (counter = *max\_counter\_v*) **then**

temporal <= **NOT** (temporal);

counter := 0;

**else**

counter := counter + 1;

**end** **if**;

**end** **if**;

**end** **process**;

clk\_out <= temporal;

**end** logic;

**ROM**

-- Quartus II VHDL Template

-- Single-Port ROM

**library** ieee;

**use** ieee.std\_logic\_1164.**all**;

**use** ieee.numeric\_std.**all**;

**entity** single\_port\_rom **is**

**generic**(

*DATA\_WIDTH* : natural := 3 \* 8;

*ADDR\_WIDTH* : natural := 5

);

**port**(

clk : **in** std\_logic;

addr : **in** natural **range** 0 **to** 2\*\**ADDR\_WIDTH* - 1;

q : **out** std\_logic\_vector((DATA\_WIDTH - 1) **downto** 0)

);

**end** **entity**;

**architecture** rtl **of** single\_port\_rom **is**

-- Build a 2-D array type for the RoM

**subtype** word\_t **is** std\_logic\_vector(0 **to** (DATA\_WIDTH - 1));

**type** memory\_t **is** **array** (0 **to** 2\*\**ADDR\_WIDTH* - 1) **of** word\_t;

-- Declare the ROM signal and specify a default value. Quartus II

-- will create a memory initialization file (.mif) based on the

-- default value.

**signal** rom : memory\_t := (

o"11111111",

o"17711771",

o"71177117",

o"71111117",

o"17111171",

o"11711711",

o"11177111",

o"11111111",

o"22222222",

o"22255222",

o"22522522",

o"25222252",

o"25555552",

o"25222252",

o"25222252",

o"22222222",

o"33333333",

o"33444443",

o"33433333",

o"33444443",

o"33433333",

o"33433333",

o"33444443",

o"33333333",

o"00000000",

o"06600660",

o"60066006",

o"60000006",

o"06000060",

o"00600600",

o"00066000",

o"00000000");

**begin**

**process**(clk)

**begin**

**if** (rising\_edge(clk)) **then**

q <= rom(addr);

**end** **if**;

**end** **process**;

**end** rtl;

**Счетчик**

-- Quartus II VHDL Template

-- Binary Counter

**library** ieee;

**use** ieee.std\_logic\_1164.**all**;

**use** ieee.numeric\_std.**all**;

**entity** binary\_counter **is**

**generic**(

*MIN\_COUNT* : natural := 0;

*MAX\_COUNT* : natural := 96

);

**port**(

clk : **in** std\_logic;

q : **out** integer **range** *MIN\_COUNT* **to** *MAX\_COUNT*;

addr : **out** std\_logic\_vector(6 **downto** 0)

);

**end** **entity**;

**architecture** rtl **of** binary\_counter **is**

**signal** qq : integer **range** *MIN\_COUNT* **to** *MAX\_COUNT*;

**begin**

**process**(clk)

**variable** cnt : integer **range** *MIN\_COUNT* **to** *MAX\_COUNT*;

**begin**

**if** (rising\_edge(clk)) **then**

**if** cnt < *MAX\_COUNT* **then**

cnt := cnt + 1;

**else**

cnt := 0;

**end** **if**;

**end** **if**;

qq <= cnt;

q <= cnt;

addr <= std\_logic\_vector(to\_unsigned(qq, addr'length) - to\_unsigned(1, addr'length));

**end** **process**;

**end** rtl;

**Декодер**

**library** ieee;

**use** ieee.std\_logic\_1164.**all**;

**use** ieee.numeric\_std.**all**;

**entity** decode **is**

**generic**(

*ADDR\_WIDTH* : natural := 5

);

**port**(

row\_in : **in** STD\_LOGIC\_VECTOR(ADDR\_WIDTH - 1 **DOWNTO** 0);

row\_out : **out** STD\_LOGIC\_VECTOR((2\*\*ADDR\_WIDTH) - 1 **DOWNTO** 0)

);

**end** decode;

**architecture** rtl **of** decode **is**

**begin**

**process**(row\_in)

**begin**

**for** i **in** row\_out'**range** **loop**

**if** row\_in = STD\_LOGIC\_VECTOR(to\_unsigned(i, *ADDR\_WIDTH*)) **then**

row\_out(i) <= '1';

**else**

row\_out(i) <= '0';

**end** **if**;

**end** **loop**;

**end** **process**;

**end** rtl;

**Контроллер кнопок**

**library** IEEE;

**use** IEEE.std\_logic\_1164.**all**;

**entity** button\_cntr **is**

**port**(clk : **in** std\_logic;

on\_off\_in : **in** std\_logic;

swich\_in : **in** std\_logic;

on\_off\_out : **out** std\_logic;

swich\_out : **out** std\_logic

);

**end** button\_cntr;

**architecture** rtl **of** button\_cntr **is**

**begin**

**end** rtl;

**Блок управления**

**library** IEEE;

**use** IEEE.std\_logic\_1164.**all**;

**entity** button\_cntr **is**

**port**(clk : **in** std\_logic;

on\_off\_in : **in** std\_logic;

swich\_in : **in** std\_logic;

on\_off\_out : **out** std\_logic;

swich\_out : **out** std\_logic

);

**end** button\_cntr;

**architecture** rtl **of** button\_cntr **is**

**begin**

**end** rtl;

**Контроллер матрицы**

**library** IEEE;

**use** IEEE.std\_logic\_1164.**all**;

**entity** matrix\_cntr **is**

**generic**(

*DATA\_WIDTH* : natural := 8;

*ADDR\_WIDTH* : natural := 7

);

**port**(

clk : **in** std\_logic;

we : **in** std\_logic;

waddr : **in** natural **range** 0 **to** 2\*\**ADDR\_WIDTH* - 1;

data : **in** std\_logic\_vector((DATA\_WIDTH - 1) **downto** 0);

row\_out : **OUT** STD\_LOGIC\_VECTOR(31 **DOWNTO** 0);

R : **OUT** STD\_LOGIC\_VECTOR(7 **DOWNTO** 0);

G : **OUT** STD\_LOGIC\_VECTOR(7 **DOWNTO** 0);

B : **OUT** STD\_LOGIC\_VECTOR(7 **DOWNTO** 0)

);

**end** matrix\_cntr;

**architecture** rtl **of** matrix\_cntr **is**

---------------------------------------------------------------

**component** simple\_dual\_port\_ram\_single\_clock **is**

**generic**(

*DATA\_WIDTH* : natural := 8;

*ADDR\_WIDTH* : natural := 7

);

**port**(

clk : **in** std\_logic;

raddr : **in** natural **range** 0 **to** 2\*\**ADDR\_WIDTH* - 1;

waddr : **in** natural **range** 0 **to** 2\*\**ADDR\_WIDTH* - 1;

data : **in** std\_logic\_vector((DATA\_WIDTH - 1) **downto** 0);

we : **in** std\_logic;

q : **out** std\_logic\_vector((DATA\_WIDTH - 1) **downto** 0)

);

**end** **component** simple\_dual\_port\_ram\_single\_clock;

---------------------------------------------------------------

**component** binary\_counter **is**

**generic**(

*MIN\_COUNT* : natural := 0;

*MAX\_COUNT* : natural := 96

);

**port**(

clk : **in** std\_logic;

q : **out** integer **range** *MIN\_COUNT* **to** *MAX\_COUNT*;

addr : **out** std\_logic\_vector(6 **downto** 0)

);

**end** **component**;

---------------------------------------------------------------

**component** decode **is**

**port**(

row\_in : **in** STD\_LOGIC\_VECTOR(4 **DOWNTO** 0);

row\_out : **out** STD\_LOGIC\_VECTOR(31 **DOWNTO** 0)

);

**end** **component** decode;

---------------------------------------------------------------

**component** DEmux **is**

**port**(

data : **IN** STD\_LOGIC\_VECTOR(7 **DOWNTO** 0);

sel : **IN** STD\_LOGIC\_VECTOR(1 **DOWNTO** 0);

R : **OUT** STD\_LOGIC\_VECTOR(7 **DOWNTO** 0);

G : **OUT** STD\_LOGIC\_VECTOR(7 **DOWNTO** 0);

B : **OUT** STD\_LOGIC\_VECTOR(7 **DOWNTO** 0)

);

**end** **component** DEmux;

---------------------------------------------------------------

**signal** dataD : std\_logic\_vector(4 **downto** 0);

**signal** sel : std\_logic\_vector(1 **downto** 0);

**signal** q : natural **range** 0 **to** 2\*\*7 - 1;

**signal** addr : std\_logic\_vector(6 **downto** 0);

**signal** dataR : STD\_LOGIC\_VECTOR(7 **DOWNTO** 0);

**begin**

p1 : binary\_counter **port** **map**(clk => clk, addr => addr, q => q);

p2 : simple\_dual\_port\_ram\_single\_clock **port** **map**(clk => clk, raddr => q, data => data, we => we, waddr => waddr, q => dataR);

sel <= addr(6 **downto** 5);

dataD <= addr(4 **downto** 0);

p3 : decode **port** **map**(row\_in => dataD, row\_out => row\_out);

p4 : DEmux **port** **map**(R => R, G => G, B => B, sel => sel, data => dataR);

**end** rtl;

**Блок управления матрицей**

**library** IEEE;

**use** IEEE.std\_logic\_1164.**all**;

**entity** matrix\_block **is**

**port**(

clk : **in** std\_logic;

b\_on\_off\_in : **in** std\_logic;

b\_swich\_in : **in** std\_logic;

row\_out : **out** STD\_LOGIC\_VECTOR(31 **DOWNTO** 0);

nR : **OUT** STD\_LOGIC\_VECTOR(7 **DOWNTO** 0);

nG : **OUT** STD\_LOGIC\_VECTOR(7 **DOWNTO** 0);

nB : **OUT** STD\_LOGIC\_VECTOR(7 **DOWNTO** 0)

);

**end** matrix\_block;

**architecture** rtl **of** matrix\_block **is**

------------------------------------------------------

**component** frequency\_converter **is**

**generic**(

*INPUT\_CLK* : INTEGER := 50\_000\_000;

*OUT\_CLK* : INTEGER := 150000

);

**port**(

clk\_in : **in** std\_logic;

clk\_out : **out** std\_logic

);

**end** **component** frequency\_converter;

------------------------------------------------------

**component** button\_cntr **is**

**port**(clk : **in** std\_logic;

on\_off\_in : **in** std\_logic;

swich\_in : **in** std\_logic;

on\_off\_out : **out** std\_logic;

swich\_out : **out** std\_logic

);

**end** **component** button\_cntr;

------------------------------------------------------

**component** single\_port\_rom **is**

**generic**(

*DATA\_WIDTH* : natural := 3 \* 8;

*ADDR\_WIDTH* : natural := 5

);

**port**(

clk : **in** std\_logic;

addr : **in** natural **range** 0 **to** 2\*\**ADDR\_WIDTH* - 1;

q : **out** std\_logic\_vector((DATA\_WIDTH - 1) **downto** 0)

);

**end** **component** single\_port\_rom;

------------------------------------------------------

**component** main\_block **is**

**port**(

clk : **in** std\_logic;

on\_off\_out : **in** std\_logic;

swich\_out : **in** std\_logic;

smth : **out** std\_logic

);

**end** **component** main\_block;

------------------------------------------------------

**component** matrix\_cntr **is**

**generic**(

*DATA\_WIDTH* : natural := 8;

*ADDR\_WIDTH* : natural := 7

);

**port**(

clk : **in** std\_logic;

waddr : **in** natural **range** 0 **to** 2\*\**ADDR\_WIDTH* - 1;

data : **in** std\_logic\_vector((DATA\_WIDTH - 1) **downto** 0);

we : **in** std\_logic;

row\_out : **OUT** STD\_LOGIC\_VECTOR(31 **DOWNTO** 0);

R : **OUT** STD\_LOGIC\_VECTOR(7 **DOWNTO** 0);

G : **OUT** STD\_LOGIC\_VECTOR(7 **DOWNTO** 0);

B : **OUT** STD\_LOGIC\_VECTOR(7 **DOWNTO** 0)

);

**end** **component** matrix\_cntr;

------------------------------------------------------

**component** printer **is**

**generic**(

*MIN\_COUNT* : natural := 0;

*MAX\_COUNT* : natural := 31;

*DATA\_WIDTH\_ROM* : natural := 3 \* 8;

*ADDR\_WIDTH\_ROM* : natural := 5;

*DATA\_WIDTH\_RAM* : natural := 8;

*ADDR\_WIDTH\_RAM* : natural := 7

);

**port**(

clk : **in** std\_logic;

smth : **in** std\_logic;

data\_f\_rom : **in** std\_logic\_vector((DATA\_WIDTH\_ROM - 1) **downto** 0);

we : **out** std\_logic;

data\_t\_ram : **out** std\_logic\_vector((DATA\_WIDTH\_RAM - 1) **downto** 0);

addr\_f\_rom : **out** natural **range** 0 **to** 2\*\**ADDR\_WIDTH\_ROM* - 1; -- 0 -> 31

waddr\_t\_ram : **out** natural **range** 0 **to** 2\*\**ADDR\_WIDTH\_RAM* - 1 -- 0 -> 127

);

**end** **component** printer;

------------------------------------------------------

**signal** clk\_out\_x : std\_logic;

**signal** swich\_out : std\_logic;

**signal** on\_off\_out : std\_logic;

**signal** smth : std\_logic;

**signal** addr\_f\_rom : natural **range** 0 **to** 2\*\*5 - 1;

**signal** waddr\_t\_ram : natural **range** 0 **to** 2\*\*7 - 1;

**signal** data\_f\_rom : std\_logic\_vector(((3 \* 8) - 1) **downto** 0);

**signal** data\_t\_ram : std\_logic\_vector(7 **downto** 0);

**signal** R, G, B : STD\_LOGIC\_VECTOR(7 **DOWNTO** 0);

**signal** we : std\_logic;

**begin**

nR <= **not** (R);

nG <= **not** (G);

nB <= **not** (B);

p1 : frequency\_converter **port** **map**(clk\_in => clk, clk\_out => clk\_out\_x);

p2 : button\_cntr **port** **map**(clk => clk\_out\_x, on\_off\_in => b\_on\_off\_in, swich\_in => b\_swich\_in, swich\_out => swich\_out, on\_off\_out => on\_off\_out);

p3 : main\_block **port** **map**(clk => clk\_out\_x, on\_off\_out => on\_off\_out, swich\_out => swich\_out, smth => smth);

p5 : printer **port** **map**(clk => clk\_out\_x, smth => smth, waddr\_t\_ram => waddr\_t\_ram, data\_f\_rom => data\_f\_rom, data\_t\_ram => data\_t\_ram, addr\_f\_rom => addr\_f\_rom, we => we);

p4 : single\_port\_rom **port** **map**(clk => clk\_out\_x, q => data\_f\_rom, addr => addr\_f\_rom);

p6 : matrix\_cntr **port** **map**(clk => clk\_out\_x, row\_out => row\_out, R => R, B => B, G => G, waddr => waddr\_t\_ram, data => data\_t\_ram, we => we);

**end** rtl;

**RAM**

-- Quartus II VHDL Template

-- Simple Dual-Port RAM with different read/write addresses but

-- single read/write clock

**library** ieee;

**use** ieee.std\_logic\_1164.**all**;

**entity** simple\_dual\_port\_ram\_single\_clock **is**

**generic**(

*DATA\_WIDTH* : natural := 8;

*ADDR\_WIDTH* : natural := 7

);

**port**(

clk : **in** std\_logic;

raddr : **in** natural **range** 2\*\**ADDR\_WIDTH* - 1 **downto** 0;

waddr : **in** natural **range** 0 **to** 2\*\**ADDR\_WIDTH* - 1;

data : **in** std\_logic\_vector((DATA\_WIDTH - 1) **downto** 0);

we : **in** std\_logic := '1';

q : **out** std\_logic\_vector((DATA\_WIDTH - 1) **downto** 0)

);

**end** simple\_dual\_port\_ram\_single\_clock;

**architecture** rtl **of** simple\_dual\_port\_ram\_single\_clock **is**

-- Build a 2-D array type for the RAM

**subtype** word\_t **is** std\_logic\_vector(0 **to** (DATA\_WIDTH - 1));

**type** memory\_t **is** **array** (0 **to** 2\*\**ADDR\_WIDTH* - 1) **of** word\_t;

-- Declare the RAM signal.

**signal** ram : memory\_t := ("00011000", -- R0

"00011000",

"00011000",

"00011000",

"00000000",

"11100000",

"00011000",

"00011000",

"11000000", --R1

"00000011",

"11000000",

"00000011",

"00000001",

"00000100",

"00010000",

"01000000",

"00110000", --R2

"00001100",

"00110000",

"00001100",

"00000011",

"00000000",

"00000000",

"00000000",

"00011000", --R3

"00000100",

"00001000",

"00100000",

"00001100",

"00000000",

"00000000",

"00000001",

--32

"11111111", --G0

"01111111",

"00111111",

"00011111",

"00001111",

"00000111",

"00000011",

"00000001",

"10100000", --G1

"01010000",

"00101000",

"00010100",

"00011010",

"00100101",

"01000010",

"10000001",

"10101000", --G2

"01010100",

"00101010",

"00010101",

"00001010",

"00000101",

"00000010",

"10000001",

"10101010", --G3

"01010101",

"00101010",

"00010101",

"00001010",

"00000101",

"00000010",

"10000001",

--64

"11111111", --B0

"01000000",

"00100000",

"00010000",

"00001000",

"00000100",

"00000010",

"00000001",

"10100000", --B1

"11010000",

"01100000",

"00110000",

"00011000",

"00001100",

"00000110",

"00000011",

"10100000", --B2

"11010000",

"11100000",

"01110000",

"00111000",

"00011100",

"00001110",

"00000111",

"10000000", --B3

"01000000",

"00100000",

"00010000",

"00001000",

"00000100",

"00000010",

"00000000",

--96

"11111111", -- A0

"01000000",

"11111111",

"00010000",

"11111111",

"00000100",

"11111111",

"00000001",

"11111111",

"11111111",

"00100000",

"11111111",

"11111111",

"00000100",

"00000010",

"00000000",

"11111111",

"11111111",

"11111111",

"00010000",

"00001000",

"00000100",

"00000010",

"00000001",

"11111111",

"11111111",

"11111111",

"11111111",

"11111111",

"00000100",

"00000010",

"00000000"

);

**begin**

**process**(clk)

**begin**

**if** (rising\_edge(clk)) **then**

**if** (we = '1') **then**

ram(waddr) <= data;

**end** **if**;

-- On a read during a write to the same address, the read will

-- return the OLD data at the address

q <= ram(raddr);

**end** **if**;

**end** **process**;

**end** rtl;

**Демультиплексор шин**

**library** ieee;

**use** ieee.std\_logic\_1164.**all**;

**use** ieee.numeric\_std.**all**;

**entity** DEmux **is**

**port**(

data : **IN** STD\_LOGIC\_VECTOR(7 **DOWNTO** 0);

sel : **IN** STD\_LOGIC\_VECTOR(1 **DOWNTO** 0);

R : **OUT** STD\_LOGIC\_VECTOR(7 **DOWNTO** 0);

G : **OUT** STD\_LOGIC\_VECTOR(7 **DOWNTO** 0);

B : **OUT** STD\_LOGIC\_VECTOR(7 **DOWNTO** 0)

);

**end** DEmux;

**architecture** rtl **of** DEmux **is**

**begin**

**process**(data, sel)

**begin**

**if** sel = "00" **then**

B <= "00000000";

G <= "00000000";

R <= data;

**end** **if**;

**if** sel = "01" **then**

B <= "00000000";

R <= "00000000";

G <= data;

**end** **if**;

**if** sel = "10" **then**

R <= "00000000";

G <= "00000000";

B <= data;

**end** **if**;

**if** sel = "11" **then**

R <= "00000000";

B <= "00000000";

G <= "00000000";

**end** **if**;

**end** **process**;

**end** rtl;

**Принтер**

**library** IEEE;

**use** IEEE.std\_logic\_1164.**all**;

**entity** printer **is**

**generic**(

*MIN\_COUNT* : natural := 0;

*MAX\_COUNT* : natural := 32;

*DATA\_WIDTH\_ROM* : natural := 3 \* 8;

*ADDR\_WIDTH\_ROM* : natural := 5;

*DATA\_WIDTH\_RAM* : natural := 8;

*ADDR\_WIDTH\_RAM* : natural := 7

);

**port**(

clk : **in** std\_logic;

smth : **in** std\_logic;

data\_f\_rom : **in** std\_logic\_vector((DATA\_WIDTH\_ROM - 1) **downto** 0); -- 4 -> 0

we : **out** std\_logic;

data\_t\_ram : **out** std\_logic\_vector((DATA\_WIDTH\_RAM - 1) **downto** 0); -- 7 -> 0

addr\_f\_rom : **out** natural **range** 0 **to** 2\*\**ADDR\_WIDTH\_ROM* - 1; -- 0 -> 31

waddr\_t\_ram : **out** natural **range** 0 **to** 2\*\**ADDR\_WIDTH\_RAM* - 1 -- 0 -> 127

);

**end** printer;

**architecture** rtl **of** printer **is**

-- Build an enumerated type for the state machine

**type** state\_type **is** (*s0*, *s1*, *s2*, *s3*, *s4*, *s5*, *s6*);

-- Register to hold the current state

**signal** state : state\_type;

**signal** red : std\_logic\_vector((DATA\_WIDTH\_RAM - 1) **downto** 0);

**signal** green : std\_logic\_vector((DATA\_WIDTH\_RAM - 1) **downto** 0);

**signal** blue : std\_logic\_vector((DATA\_WIDTH\_RAM - 1) **downto** 0);

**signal** d\_f\_r : std\_logic\_vector((DATA\_WIDTH\_ROM - 1) **downto** 0);

**begin**

**process**(clk)

**variable** cnt : natural **range** *MIN\_COUNT* **to** *MAX\_COUNT*;

**begin**

**if** (rising\_edge(clk)) **then**

**case** state **is**

**when** *s0* =>

-- set read addr ROM

addr\_f\_rom <= cnt;

state <= *s1*;

**when** *s1* =>

we <= '1';

state <= *s2*;

**when** *s2* =>

--write data colors to RAM & set write addr RAM

data\_t\_ram <= blue;

waddr\_t\_ram <= cnt;

state <= *s3*;

**when** *s3* =>

data\_t\_ram <= green;

waddr\_t\_ram <= cnt + 32;

state <= *s4*;

**when** *s4* =>

data\_t\_ram <= red;

waddr\_t\_ram <= cnt + 32 + 32;

state <= *s5*;

**when** *s5* =>

we <= '0';

state <= *s6*;

**when** *s6* =>

--inc counter

**if** cnt < *MAX\_COUNT* **then**

cnt := cnt + 1;

**else**

cnt := 0;

**end** **if**;

state <= *s0*;

**end** **case**;

**end** **if**;

**end** **process**;

**process**(data\_f\_rom)

**begin**

set\_color : **for** i **in** 0 **to** *DATA\_WIDTH\_RAM* - 1 **loop**

red(i) <= data\_f\_rom(i \* 3);

green(i) <= data\_f\_rom(i \* 3 + 1);

blue(i) <= data\_f\_rom(i \* 3 + 2);

**end** **loop** set\_color;

**end** **process**;

**end** rtl;

**Регистр**

**library** IEEE;

**use** IEEE.std\_logic\_1164.**all**;

**use** ieee.std\_logic\_arith.**all**;

**use** ieee.std\_logic\_unsigned.**all**;

**entity** register\_X **is**

**port**(

clk : **in** std\_logic;

b\_on\_off\_in : **in** std\_logic;

b\_swich\_in : **in** std\_logic;

--enable : in std\_logic;

--srst : in std\_logic;

data\_in : **in** std\_logic\_vector(31 **downto** 0);

data\_out\_en : **out** std\_logic;

data\_out : **out** std\_logic

);

**end** register\_X;

**architecture** arh **of** register\_X **is**

**begin**

**process**(clk, b\_swich\_in, data\_in, b\_on\_off\_in)

**variable** reg\_var : std\_logic;

**variable** reg\_var2 : std\_logic;

**begin**

**if** rising\_edge(clk) **then**

**if** (b\_swich\_in = '1') **then**

reg\_var := '0';

**elsif** (b\_on\_off\_in = '1') **then**

reg\_var := data\_in(0);

reg\_var2 := data\_in(1);

**end** **if**;

**end** **if**;

data\_out\_en <= reg\_var;

data\_out <= reg\_var2;

**end** **process**;

**end** arh;

**Top level**

**library** IEEE;

**use** IEEE.std\_logic\_1164.**all**;

**use** ieee.std\_logic\_arith.**all**;

**use** ieee.std\_logic\_unsigned.**all**;

**entity** top\_level **is**

**port**(

clk : **in** std\_logic;

b\_on\_off\_in : **in** std\_logic; --en

b\_swich\_in : **in** std\_logic; --reset

data\_in : **in** std\_logic\_vector(31 **downto** 0);

row\_out : **OUT** STD\_LOGIC\_VECTOR(31 **DOWNTO** 0);

nR : **OUT** STD\_LOGIC\_VECTOR(7 **DOWNTO** 0);

nG : **OUT** STD\_LOGIC\_VECTOR(7 **DOWNTO** 0);

nB : **OUT** STD\_LOGIC\_VECTOR(7 **DOWNTO** 0)

);

**end** top\_level;

**architecture** rtl **of** top\_level **is**

-----------------------------------------------------

**component** register\_X **is**

**port**(

clk : **in** std\_logic;

b\_on\_off\_in : **in** std\_logic;

b\_swich\_in : **in** std\_logic;

data\_in : **in** std\_logic\_vector(31 **downto** 0);

data\_out\_en : **out** std\_logic;

data\_out : **out** std\_logic

);

**end** **component** register\_X;

-----------------------------------------------------

**component** matrix\_block **is**

**port**(

clk : **in** std\_logic;

b\_on\_off\_in : **in** std\_logic;

b\_swich\_in : **in** std\_logic;

row\_out : **out** STD\_LOGIC\_VECTOR(31 **DOWNTO** 0);

nR : **OUT** STD\_LOGIC\_VECTOR(7 **DOWNTO** 0);

nG : **OUT** STD\_LOGIC\_VECTOR(7 **DOWNTO** 0);

nB : **OUT** STD\_LOGIC\_VECTOR(7 **DOWNTO** 0)

);

**end** **component** matrix\_block;

-----------------------------------------------------

**signal** data\_out\_en : std\_logic;

**signal** data\_out : std\_logic;

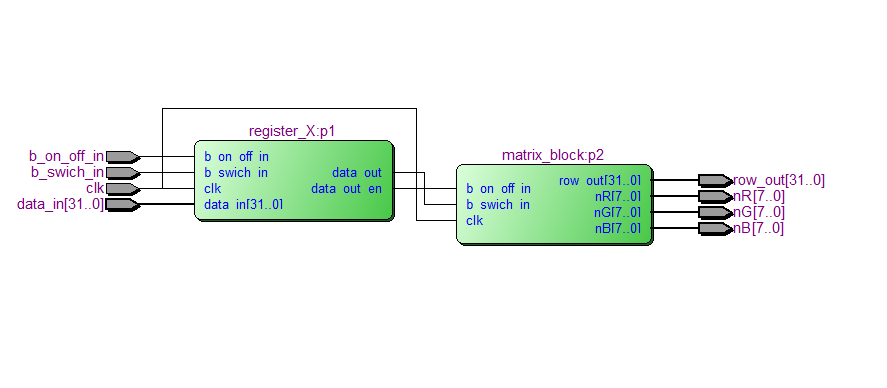
**begin**

p1 : register\_X **port** **map**(clk => clk, b\_on\_off\_in => b\_on\_off\_in, b\_swich\_in => b\_swich\_in, data\_in => data\_in, data\_out\_en => data\_out\_en,data\_out => data\_out);

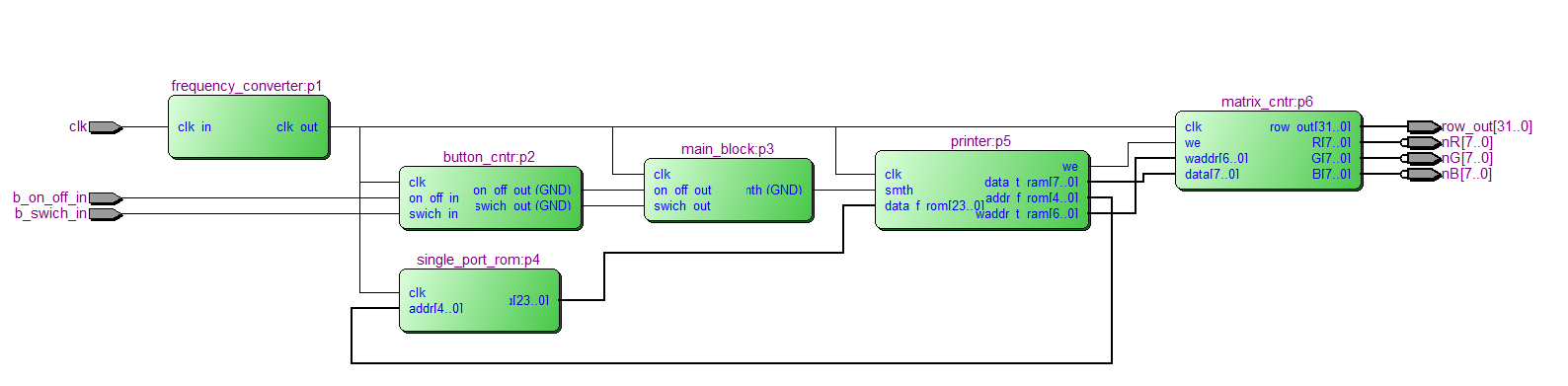
p2 : matrix\_block **port** **map**(clk => clk, b\_on\_off\_in => data\_out\_en, b\_swich\_in => data\_out, nR => nR, nG => nG, nB => nB, row\_out => row\_out);

**end** **architecture**;

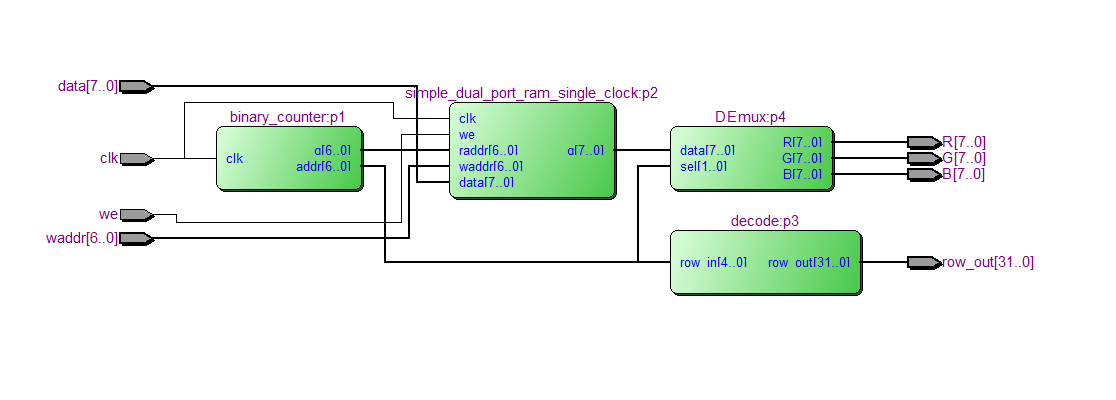
**RTL верхнего уровня**



**RTL matrix\_block**



**RTL matrix\_cntr**



**Выводы:** Согласно заданию, утвержденному по итогам лабораторной работы №1, а также полученной структуре пользовательского компонента (с детализированным интерфейсом каждого из модулей в составе компонента на уровне описания сущностей), получила конечную реализацию проекта на языке описания аппаратуры VHDL.