

АННОТАЦИЯ

В данной курсовой работе разработан модуль клавиатурного тренажера, модули для организации приема и обработки данных по протоколу PS/2, модули для протокола VGA, модуль верхнего уровня для совместной работы всех модулей.

Разработка произведена с использованием отладочной платы Xilinx Nexys A7 и ПЛИС XC7A100TCSG324-1L семейства Artix-7 в ее составе.

Отображение информации клавиатурного тренажера осуществляется на мониторе по протоколу VGA. Для ввода символов используется клавиатура, работающая по протоколу PS/2. Для описания аппаратуры используется язык Verilog. Питание клавиатурного тренажера производится через USB порт.

Для поиска и исправления ошибок в конфигурации ПЛИС проведена верификация при помощи симуляции и на отладочной плате, в результате сделан вывод о корректной работе клавиатурного тренажера и эффективности способа его реализации, который можно рекомендовать для дальнейшего использования.

Работа включает в себя 28 рисунков, 1 таблицу, 10 листингов, 2 приложения. Количество страниц в работе — 52.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ	6
1 ТЕОРИТИЧЕСКАЯ ЧАСТЬ.....	8
1.1 Клавиатурный тренажер	8
1.2 PS/2	9
1.3 VGA	10
2 ПРАКТИЧЕСКАЯ ЧАСТЬ	12
2.1 Системная модель	12
2.1.1 Входные значения	12
2.1.2 Алгоритм	12
2.1.3 Блок-схема алгоритма.....	13
2.1.4 Программная реализация системной модели.....	13
2.2 RTL-уровень	14
2.2.1 Реализация модуля клавиатурного тренажера	14
2.2.2 Блок-схема модуля клавиатурного тренажера.....	16
2.2.3 Реализация модуля ресивера PS/2	18
2.2.4 Реализация модуля дешифратора PS/2	19
2.2.5 Реализация модуля синхронизатора VGA.....	20
2.2.6 Реализация модуля делителя частоты.....	21
2.2.7 Реализация модуля формирования изображения символов	21
2.2.8 Реализация управляющего модуля VGA.....	22
2.2.9 RTL схемы.....	23
2.3 Прототипирование на ПЛИС	25
2.3.1 Реализация модуля верхнего уровня.....	25
2.3.2 Файл проектных ограничений	25
3 ТЕСТИРОВАНИЕ	26
3.1 Тестирование работы модулей с помощью симуляции	26
3.1.1 Тестовое покрытие системной модели	26
3.1.2 Тестирование модуля клавиатурного тренажера.....	26

3.1.3 Тестирование модулей PS/2	27
3.1.4 Тестирование модуля VGA	27
3.1.5 Тестирование модуля верхнего уровня.....	28
3.2 Тестирование работы устройства на ПЛИС	28
ЗАКЛЮЧЕНИЕ	30
СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ	31
ПРИЛОЖЕНИЯ.....	32
Приложение А	33
Приложение Б	52

ВВЕДЕНИЕ

Клавиатура — стандартное устройство ввода информации, позволяющее вводить в компьютер буквенно-цифровую информацию, управлять состоянием текущего окна или вводить специальные управляющие сигналы [1].

Клавиатура неразрывно связана с компьютером, из-за чего так важно уметь взаимодействовать с ней в нынешнее время развития информационных технологий. При начальном изучении обычным является «зрячий» или «двупальцевый» метод печати на клавиатуре, когда символ ищется каждый раз заново. Обучение же «слепому» методу печати позволяет вводить текст быстро и безошибочно, отчего такая печать актуальна и востребована. Научиться этому умению можно с помощью клавиатурного тренажера — специальной программы, предназначенной для приобретения навыка десятипальцевого набора и запоминания расположения клавиш.

Все больше людей использует именно FPGA (он же ПЛИС) для разработки различных систем, в том числе и для клавиатурного тренажера.

Программируемая логическая интегральная схема (ПЛИС) — это электронная микросхема, которая построена на гибких логических блоках и используется для создания цифровых микросхем. Программирование ПЛИС осуществляется самим пользователем, конструктором аппаратуры [2].

В настоящее время известны два стандартных языка описания аппаратуры (HDL) — VHDL и Verilog [3].

Языком, выбранным для проекта, является Verilog. Он часто используется в проектировании, верификации аналоговых, цифровых и смешанных электронных систем на различных уровнях абстракции, а также имеет конструкции для описания элементов аппаратуры в краткой и читаемой форме. Используя Verilog, разработчик должен изучить только один язык для всех аспектов логического проектирования.

Цель курсовой работы состоит в изучении и разработке клавиатурного тренажера, который способен сравнивать введенный с клавиатуры символ с

символом на мониторе на правильность нажатой клавиши, клавиатуры с интерфейсом PS/2 для ввода значений, а также интерфейса VGA для демонстрации всего процесса на мониторе при помощи языка описания аппаратуры Verilog.

Для достижения поставленной цели необходимо реализовать ряд задач:

1. Ознакомиться с принципами работы клавиатурного тренажера, провести анализ предметной области.
2. Разработать набор управляющих модулей согласно принципам работы.
3. Разработать набор модулей, описывающих процессы приёма и обработки данных по протоколу PS/2.
4. Разработать набор модулей, описывающих драйвер для вывода изображения на монитор по протоколу VGA.
5. Разработать модуль верхнего уровня, реализующий управление для совместной работы всех модулей.
6. Провести верификацию при помощи симуляции.
7. Провести верификацию на отладочной плате.

Объектом исследования данной работы является алгоритм работы клавиатурного тренажера, а предметом исследования — настраиваемый модуль клавиатурного тренажера.

Методы исследования: аналитические, практические.

Теоретическая часть курсовой работы написана при использовании литературы по теории работы с клавиатурой и теории автоматов. Практическая часть написана при использовании литературы по проектированию цифровой аппаратуры. Информационная база исследования — общий перечень всех информационных источников, которые были использованы, — представлена в «Списке использованных источников».

Руководство пользователя прикреплено в Приложении Б.

1 ТЕОРИТИЧЕСКАЯ ЧАСТЬ

1.1 Клавиатурный тренажер

Клавиатурный тренажёр представляет собой набор упражнений для механического набора текста на клавиатуре и предназначен для обучения методу слепой печати двумя руками.

При слепом наборе необходимо не только знать, где находится определенная клавиша, но и нажимать символ на клавиатуре согласно строгому распределению символов между всеми десятью пальцами, чтобы беспрепятственно печатать даже с закрытыми глазами. В русской раскладке изначально пальцы (кроме больших) располагаются в позиции «Ф Ы В А» для правой руки и «О Л Д Ж» для левой, а в английской — «A S D F» и «J K L ;». Правильное расположение рук показано на Рисунке 1.1.

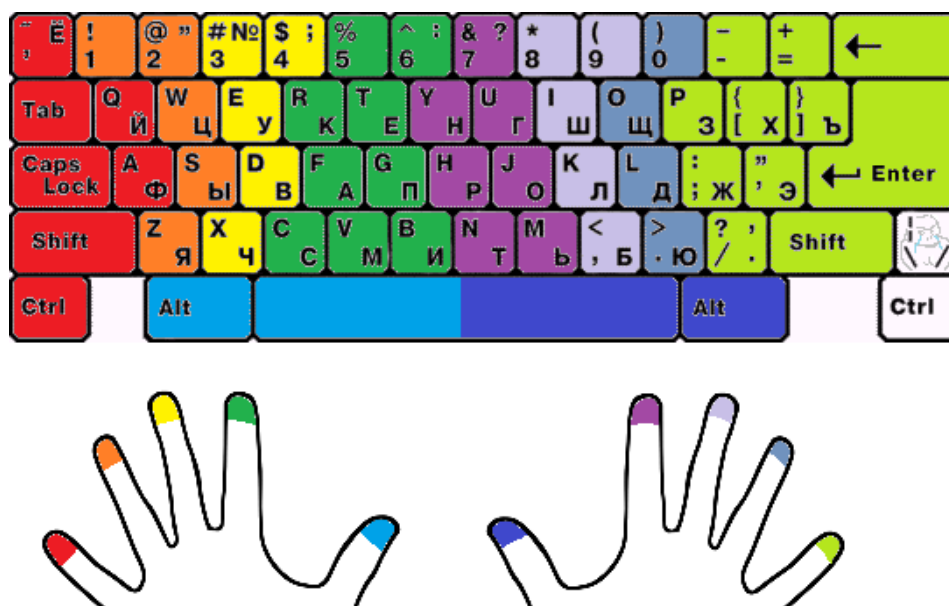


Рисунок 1.1 — Позиция рук при слепой печати

Клавиатурный тренажер проверяет, тем ли пальцем нажата клавиша, не в состоянии, ответственность за это лежит на самом пользователе. Однако данный тренажер не зря выступает помощником для обучения: он выдает для человека некоторый символ на экране (так называемый, «эталонный»), который затем сравнивается с символом нажатой клавиши. Если все верно, то происходит переход к новому элементу текста, иначе символ на мониторе меняться не будет.

1.2 PS/2

PS/2 — простейший синхронный последовательный симплексный интерфейс для ввода информации, для передачи данных от клавиатуры на вычислительное устройство.

Размер пакета протокола передачи данных PS/2 составляет 11 бит. Состав пакета отображен в Таблице 1.1.

Таблица 1.1 — Состав пакета

Номер бита	Назначение
0	Стартовый бит (всегда равен нулю)
1	Биты данных
2	
3	
4	
5	
6	
7	
8	
9	Бит четности (используется для проверки корректности полученных данных)
10	Стоп бит (всегда равен единице)

Контакты PS/2 представлены на Рисунке 1.2, где номером 1 обозначена шина передачи данных DATA, номером 3 — общая шина земли GND, номером 4 — шина питания VCC, номеру 5 соответствует CLK — синхросигнал, генерируемый устройством передачи данных, а второй и шестой контакты не используются.

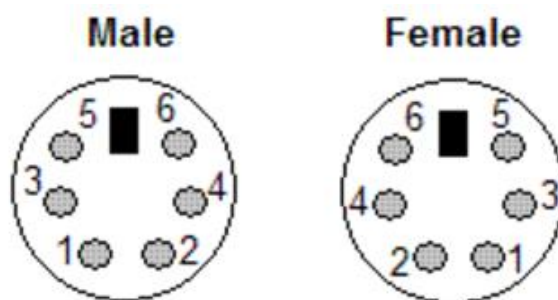


Рисунок 1.2 — Выводы для PS/2

При нажатии клавиши на клавиатуре передается только код нажатой клавиши, а при отпускании клавиши сначала идет код F0, затем — код отпущенной клавиши.

Временная диаграмма сигналов PS/2 приведена на Рисунке 1.3.

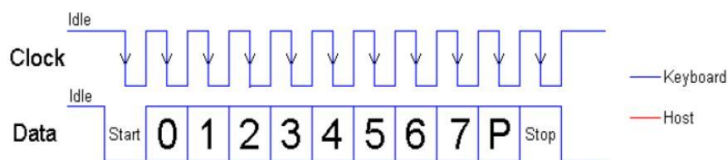


Рисунок 1.3 — Временная диаграмма работы клавиатуры

1.3 VGA

VGA (Video Graphics Array) — формат видеосигналов, предназначенный для вывода информации на мониторы.

Интерфейс VGA пользуется спросом на протяжении тридцати лет и использует построчную передачу видеосигнала. Он имеет следующие контакты, представленные на Рисунке 1.4:

- контакт RED, отвечающий за передачу глубины красного цвета (контакт под номером 1 на Рисунке 1.4);
- контакт GREEN, отвечающий за передачу глубины зеленого цвета (2);
- контакт BLUE, отвечающий за передачу глубины синего цвета (3);
- контакты шины земли (5-10);
- контакт H_SYNC для вывода сигнала горизонтальной синхронизации (13);
- контакт V_SYNC для вывода сигнала вертикальной синхронизации (14);
- остальные контакты для передачи по протоколу VGA не используются.

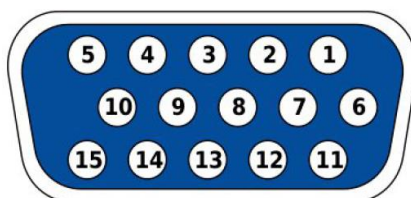


Рисунок 1.4 — Графическое изображение штекера VGA

Временная диаграмма формирования кадра с отображением сигналов для вертикальной и горизонтальной синхронизации VGA изображена на Рисунке 1.5, согласно которому для горизонтальной синхронизации предназначены

Horizontal Timing (период формирования одной строки кадра), Display (промежуток времени, на протяжении которого происходит передача данных), Synch Pulse (синхроимпульс окончания передачи пикселей строки), Front и Back Porch. Вертикальная синхронизация: Vertical Timing (период формирования целого кадра), а также аналогично Display, Front и Back Porch, и синхроимпульс Synch Pulse окончания передачи пикселей кадра. Blanking Time — период времени, на протяжении которого не передаются данные кадра.

Вертикальная синхронизация — синхронизация кадровой частоты в компьютерной графике с частотой вертикальной развёртки монитора [4].

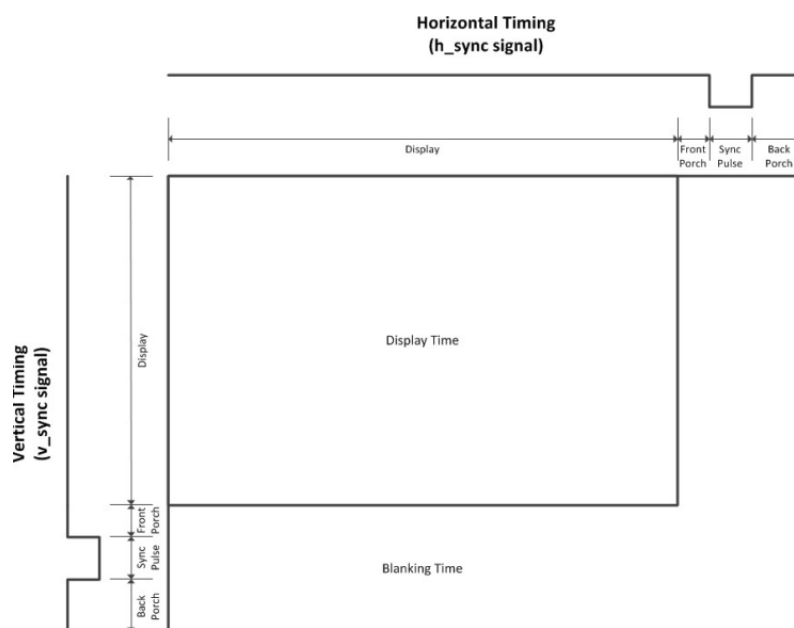


Рисунок 1.5 — Диаграмма формирования кадра

Нужные технические характеристики для VGA данной курсовой работы были взяты согласно красной пометке Рисунка 1.6.

Format	Pixel Clock (MHz)	Horizontal (in Pixels)				Vertical (in Lines)			
		Active Video	Front Porch	Sync Pulse	Back Porch	Active Video	Front Porch	Sync Pulse	Back Porch
640x480, 60Hz	25.175	640	16	96	48	480	11	2	31
640x480, 72Hz	31.500	640	24	40	128	480	9	3	28
640x480, 75Hz	31.500	640	16	96	48	480	11	2	32
640x480, 85Hz	36.000	640	32	48	112	480	1	3	25
800x600, 56Hz	38.100	800	32	128	128	600	1	4	14
800x600, 60Hz	40.000	800	40	128	88	600	1	4	23
800x600, 72Hz	50.000	800	56	120	64	600	37	6	23
800x600, 75Hz	49.500	800	16	80	160	600	1	2	21
800x600, 85Hz	56.250	800	32	64	152	600	1	3	27
1024x768, 60Hz	65.000	1024	24	136	160	768	3	6	29
1024x768, 70Hz	75.000	1024	24	136	144	768	3	6	29
1024x768, 75Hz	78.750	1024	16	96	176	768	1	3	28
1024x768, 85Hz	94.500	1024	48	96	208	768	1	3	36

Рисунок 1.6 — Технические характеристики для разных форматов VGA

2 ПРАКТИЧЕСКАЯ ЧАСТЬ

2.1 Системная модель

2.1.1 Входные значения

В качестве входных значений рассматриваются вводимые с клавиатуры символы. Для ввода данных используются клавиши на клавиатуре, подключенной к отладочной плате, работающей по протоколу PS/2. Запуск клавиатурного тренажера, как и его сброс, осуществляется путем нажатия специальных клавиш.

2.1.2 Алгоритм

В основе разрабатываемого проекта клавиатурного тренажера лежит сравнение двух символов.

Алгоритм работы тренажера состоит из шести логических частей:

1. Установка и вывод начального «эталонного» значения.
2. Ввод символа при помощи клавиатуры.
3. Проверка символа на принадлежность к символу сброса — выключения клавиатурного тренажера.
4. Сравнение введенного символа с эталонным, который в момент нажатия клавиши отображается на экране, в случае отсутствия выключения.
5. Вывод нового или старого эталонного значения с зависимости от сравнения.
6. Переход ко второму пункту.

Пятая часть работы алгоритма применяется для того, чтобы задать символ, который будет изображаться на мониторе до следующего нажатия клавиши пользователем. Если эталонный и нажатый символ равны, значит, пользователь выбрал верную клавишу и следующий экранный символ будет выведен согласно

определенной последовательности или случайно, иначе значение остается прежним — человек нажал неправильную клавишу и должен попытаться снова.

2.1.3 Блок-схема алгоритма

На основании приведенного выше алгоритма была создана блок-схема клавиатурного тренажера (Рисунок 2.1).

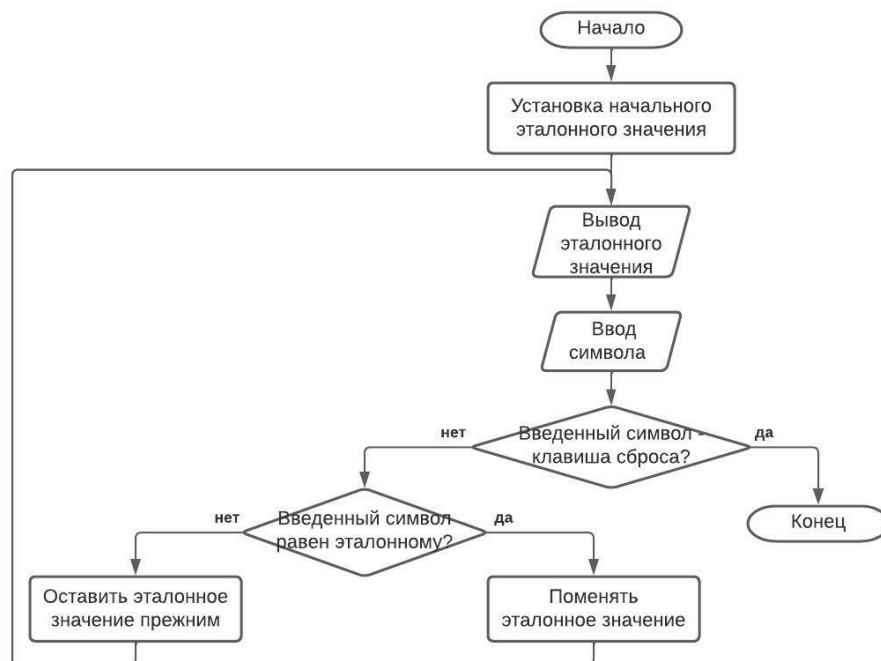


Рисунок 2.1 — Блок-схема алгоритма клавиатурного тренажера

2.1.4 Программная реализация системной модели

Для реализации системной модели был выбран язык C++, так как он удобен и быстр в использовании. Программная реализация для поставленной задачи согласно системной модели представлена в Листинге 2.1.

Листинг 2.1 — Программная реализация клавиатурного тренажера на языке C++

```
#include <iostream>
#include <stdlib.h>
#include <string.h>
using namespace std;

int main(){
    const int N = 10;
    const int MAX_LEN = 255;
    /*словарь символов*/
    const char* const DICTIONARY_OF_WORDS[N] = {
        "h",
```

```
        "w",
        "m",
        "f",
        "r",
        "l",
        "m",
        "a",
        "b"
    };
    int random;
    char S[N] = ""; //вводимый символ
    while (cin.getline(S,MAX_LEN) != "r") {
        random = rand() % N; //случайное число, индекс
        cout << "input word:\t " << DICTIONARY_OF_WORDS[random] << '\n';
        cin.getline(S,MAX_LEN);
        if (cin.getline(S,MAX_LEN) != "r") {
            if (strcmp(S, DICTIONARY_OF_WORDS[random]) == 0) {
                random = rand() % N; //Случайное число, индекс
                cout << "input word:\t " << DICTIONARY_OF_WORDS[random] << '\n';
            }
        }
    }
    exit(0);
}
```

2.2 RTL-уровень

2.2.1 Реализация модуля клавиатурного тренажера

На этапе разработки устройства на уровне RTL был рассмотрен модуль клавиатурного тренажера, отвечающий за его логику, и сформирован необходимый набор узлов для него.

Модуль — основной элемент описания на языке Verilog [5].

Подход к определению аппаратного состава был следующим: необходимо принимать значения, полученные с клавиатуры, поэтому в модуле присутствует входной регистр [7:0] code, предназначенный для кода нажатой клавиши, и входные сигналы установки set_signal для возможности отследить новое нажатие и сброса reset_signal для возврата к исходным значениям, а также синхросигнал clk. В качестве выходных регистров выступают reg [7:0] res_code, что хранит в себе код клавиши эталонного значения, с помощью которого после будет выведено изображение на монитор, и результат сравнения символов reg res_out, который принимает значение единицы только в том случае, когда они не равны. Вспомогательный регистр reg [7:0] need_symbol отвечает за текущий эталон, с

которым сравнивается код нажатой кнопки клавиатуры. Регистр `reg [2:0] state` — состояния автомата, всего их пять.

Нулевое состояние автомата производит присвоение значению регистра `need_symbol` кода «нереального» эталонного, равного коду символа включения клавиатурного тренажера «Enter», чтобы запустить последовательность эталонных кодов.

Первое состояние по сути — начало работы автомата, так как в блоке инициализации обнуляется значение регистра `res_out` и именно это состояние указано как начальное. Если в качестве входных данных пришел код клавиши «Enter» или неизвестной клавиши (`code` равняется 0), то переход в 0 состояние, иначе переход во 2 состояние.

Второе состояние — сравнение пришедшего с клавиатуры кода клавиши `code` с эталонным `need_symbol`. Здесь три варианта:

1. Код нажатой клавиши равен эталонному коду запуска клавиатурного тренажера. Тогда код первого эталонного значения (символ «S» с кодом «1В») записывается в регистр результата `res_code`. Переход в 5 состояние.
2. Код нажатой клавиши равен эталонному, с условием, что код эталонного — не код символа запуска клавиатурного тренажера. Переход в 3 состояние.
3. Код нажатой клавиши не равен эталонному. Происходит присвоение значению регистра `res_out` единицы. Переход в 4 состояние.

В третьем состоянии посредством оператора множественного выбора `case` обновляется значение эталонного кода `need_symbol`. Переход в 4 состояние.

Четвертое состояние заносит данные `need_symbol` в выходной регистр `res_code`, если `code` не является кодом клавиши «Enter», иначе совершается присвоение `need_symbol` значения первого эталонного кода «1В». Ожидание сигнала `set_signal` для перехода в 1 состояние.

Исходный код модуля автомата клавиатурного тренажера представлен в Приложении А, Листинге А.1.

2.2.2 Блок-схема модуля клавиатурного тренажера

Блок-схема автомата управляющего типа — это некоторая графическая вариация диаграммы автомата. Выходящие из условной вершины стрелки помечаются значениями функций [6].

Разработанная в рамках системной модели блок-схема была преобразована в блок-схему, учитывающую особенности описания на уровне регистровых передач и показанную на Рисунках 2.2-2.3.

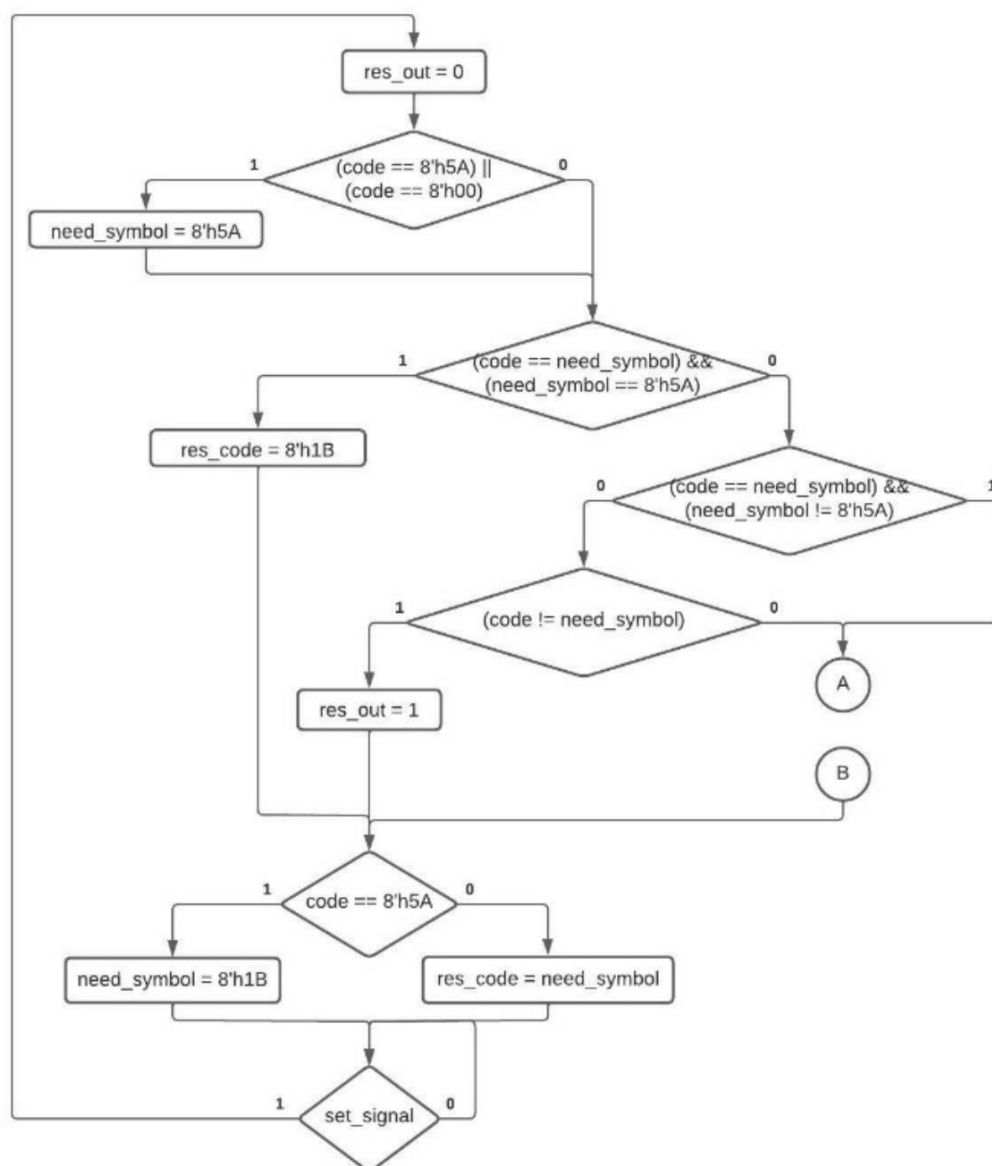


Рисунок 2.2 — Блок-схема модуля клавиатурного тренажера (часть 1)

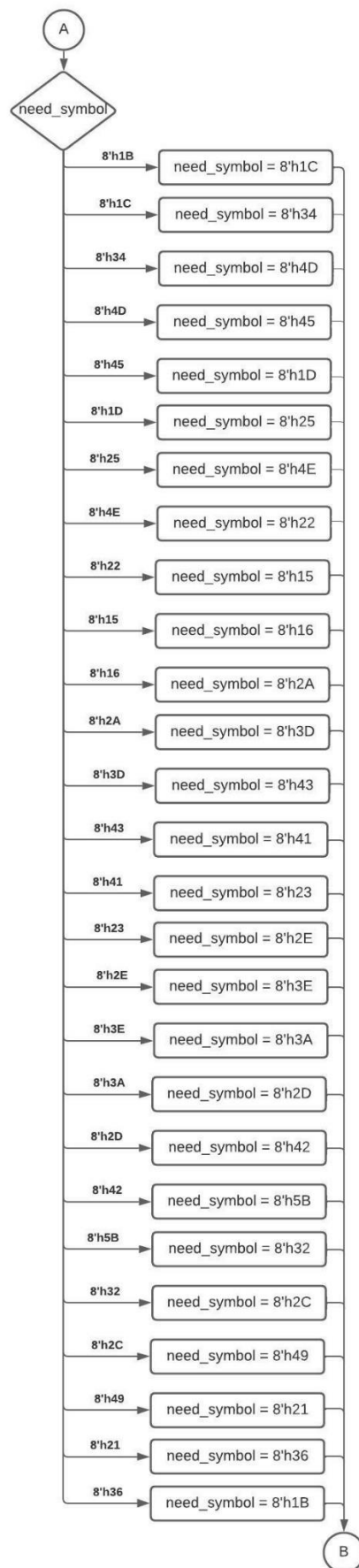


Рисунок 2.3 — Блок-схема модуля клавиатурного тренажера (часть 2)

2.2.3 Реализация модуля ресивера PS/2

Для корректной работы клавиатуры по протоколу PS/2 понадобится два модуля — дешифратор и ресивер.

После ключевого слова `module` следует его имя (идентификатор), назначаемое HDL-программистом, с последующим перечислением входных и выходных портов модуля, которые после синтеза схемы (и ее последующей прошивки в ПЛИС) уже можно будет называть входными и выходными портами (контактами) разработанной схемы [7].

Модуль `ps2_reciever` имеет три входа и два выхода:

- `input clk;`
- `input PC2_CLK;`
- `input r;`
- `output reg [7:0] recieved_data;`
- `output reg R_O.`

Модуль считывает входные сигналы и объединяет их в пакет данных PS/2 `recieved_data`, который впоследствии будет расшифрован. Модуль работает по заднему фронту синхросигнала с клавиатуры `PC2_CLK` и реализован с помощью блока `always` и оператора множественного выбора `case`: нулевое состояние `state` сбрасывает сигнал готовности считывания данных `ready_output R_O` в ноль, передается стартовый бит, далее в состояниях 1-8 следует запись самих битов данных: в регистр результата `received_data` заносятся соответствующие входные данные с клавиатуры `r`. Состояние 9 отведено для бита четности количества единичных битов, используемого для проверки корректности полученных данных. Последний бит является стоповым единичным битом. Если пакет передан некорректно, то необходимо вернуться в нулевое состояние. Если ошибок выявлено не было, на `R_O` подается единица.

Исходный код модуля `ps2_reciever` показан в Приложении А, Листинге А.2.

2.2.4 Реализация модуля дешифратора PS/2

Дешифраторы — это комбинационные схемы с несколькими входами и выходами, преобразующие код, подаваемый на входы в сигнал на одном из выходов [8].

Модуль `ps2_dc` имеет три входа и три выхода:

- `input clk;`
- `input [7:0] pc2_data;`
- `input pc2_ro;`
- `output reg [7:0] fsm_data;`
- `output reg set_signal;`
- `output reg reset_signal.`

Модуль декодирует коды клавиш клавиатуры в зависимости от входного значения `pc2_data` (из предыдущего модуля `ps2_reciever`) при помощи оператора множественного выбора `case`. Также в дешифраторе есть флаг `flag`, указывающий на то, что пришел код отжатия кнопки.

Модуль преобразует код PS/2 в выходной сигнал. В зависимости от нажатой клавиши данный модуль либо выдаст новое значение памяти, либо сигнал нажатия кнопки [9].

На выход модуля `ps2_dc` подаются сигналы `set_signal` и `reset_signal`, которые принимают значение единицы в том случае, если код нажатой клавиши совпадает с указанными кодами сброса `r_key`, `re_key` в параметрах модуля и если есть ненулевой разрешающий сигнал `pc2_ro`. Значение регистра `fsm_data` равно нулю в случае единичного `reset_signal` и при несовпадении полученного значения `pc2_data` в `case`. Регистр `fsm_data` содержит коды клавиш, необходимые в дальнейшем для автомата FSM, за исключением клавиш подтверждения `s_key` и клавиш сброса.

Код модуля `ps2_dc` изображен в Приложении А, Листинге А.3

В качестве параметров `parameter` и в `case` указаны коды клавиш на клавиатуре при использовании протокола PS/2 согласно Рисунку 2.4.

Key	Scancode	Key	Scancode	Key	Scancode
ESC	76	I	43	Alt (right)	E011
F1	05	O	44	Windows (right)	E027
F2	06	P	4D	Menus	E02F
F3	04	[54	Ctrl (right)	E014
F4	0C]	5B	Insert	E070
F5	03	\	5D	Home	E06C
F6	0B	Caps Lock	58	Page Up	E07D
F7	83	A	1C	Delete	E071
F8	0A	S	1B	End	E069
F9	01	D	23	Page Down	E07A
F10	09	F	2B	Up Arrow	E075
F11	78	G	34	Left Arrow	E06B
F12	07	H	33	Down Arrow	E072
`	0E	J	3B	Right Arrow	E074
1	16	K	42	Num Lock	77
2	1E	L	4B	/	E04A
3	26	:	4C	*	7C
4	25	'	52	-	7B
5	2E	Enter	5A	7	6C
6	36	Shift (Left)	12	8	75
7	3D	Z	1A	9	7D
8	3E	X	22	+	79
9	46	C	21	4	6B
0	45	V	2A	5	73
-	4E	B	32	6	74
=	55	N	31	1	69
Backspace	66	M	3A	2	72
Tab	0D	.	41	3	7A
Q	15	,	49	0	70
W	1D	/	4A	.	71
E	24	Shift (Right)	59	Enter	E05A
R	2D	Ctrl (left)	14	Scroll Lock	7E
T	2C	Windows (left)	E01F		
Y	35	Alt (left)	11		
U	3C	Spacebar	29		

Рисунок 2.4 — Список кодов клавиш на клавиатуре при использовании протокола

2.2.5 Реализация модуля синхронизатора VGA

Для отображения информации на мониторе необходим интерфейс VGA. Сигнал формата VGA представляет собой компонентный сигнал RGBHV (сигнал RGB + синхронизация «по горизонтали» + синхронизация «по вертикали») [10].

Первый модуль в курсовой работе, связанный с VGA, — модуль vga, который отвечает за горизонтальную и вертикальную синхронизацию (Приложение А, Листинг А.4).

Модуль vga имеет один вход и семь выходов:

- input clk50;
- output reg [9:0] h_counter;
- output reg [9:0] v_counter;
- output hsync;
- output vsync;
- output valid;
- output reg newframe;
- output reg newline.

В данном модуле в качестве параметров указываются значения из таблицы характеристик для интерфейса согласно Рисунку 1.6. Также здесь реализованы выходные регистры горизонтального `h_counter` и вертикального `v_counter` счетчиков, необходимые для перемещения по состояниям `h_state` (точнее: `H_ACTIVE_STATE`, `H_FRONT_STATE`, `H_PULSE_STATE`, `H_BACK_STATE`) и аналогичные состояния для `v_state`, а также сами выходные сигналы `hsync` горизонтальной и `vsync` вертикальной синхронизации. Сигнал `valid` отвечает за действующую область отображения, регистры `newframe` и `newline` — за новый кадр и строку. Все перечисленные переменные применяются для того, чтобы сформировать подходящие сигналы для кадра VGA, которые в определенные промежутки времени должны принимать требуемые значения, подробно описанные на Рисунке 1.5.

2.2.6 Реализация модуля делителя частоты

Частота пикселей `pixel clock` выбранного для проекта VGA интерфейса равняется 50 МГц, поэтому требуется делитель частоты — один из вариантов использования счётчика.

Модуль `div_clk` имеет один вход и один выход:

- `input clk;`
- `output new_clk.`

На вход `clk` подаётся сигнал тактового генератора, на выход — новая частота `new_clk`, которая задействована в модуле `vga` в качестве входа `clk50`. В данном модуле организован регистр, который увеличивается на единицу при каждом восходящем фронте `clk`.

Код модуля `div_clk` представлен в Приложении А, Листинге А.6.

2.2.7 Реализация модуля формирования изображения символов

Модуль `chars` необходим для формирования изображения (Приложение А, Листинг А.5).

Данный модуль имеет два входа и один выход:

- input [5:0] char;
- input [2:0] rownum;
- output reg [7:0] pixels.

При помощи оператора множественного выбора в зависимости от поступивших значений символа char и rownum определяется выходной регистр pixels, ответственный за визуальные представления, которые сформированы в модуле при помощи единиц и нулей, всех английских букв клавиатуры, цифр и таких знаков, как «-», «=», «[», «]», «;», «'», «,», «.», «/», «\».

2.2.8 Реализация управляющего модуля VGA

Модуль top_vga координирует работу трех описанных выше модулей (vga, chars, div_clk) для вывода изображения на монитор по протоколу VGA.

Данный модуль имеет три входа и пять выходов:

- input clk;
- input [7:0] res_code;
- input res_out;
- output hsync;
- output vsync;
- output [3:0] r;
- output [3:0] g;
- output [3:0] b.

На вход помимо синхросигнала clk подаются эталонные коды клавиш res_code, которые должны отображаться на дисплее, и регистр результата сравнения res_out из автомата FSM. В качестве выходных портов выступают сигналы горизонтальной hsync и вертикальной vsync синхронизации, образуемые в модуле vga, а также регистры цвета r, g, b — красный, зеленый, синий, изначально равные нулю. Параметры WIDTH и HEIGHT взяты из Рисунка 1.6. Подключается модуль div_clk. На вход модуля chars идет значение из

оператора множественного выбора case, определяемое в зависимости от res_code, для последующей визуализации символов, которые, если res_out равен нулю, будут окрашены в зеленый цвет: регистру g присвоится значение 1111, в то время как красный и синий по-прежнему останутся нулевыми. В случае, когда res_out равняется единице, символ будет гореть красным цветом с соответствующими параметрами rgb. Сами символы расположены согласно определенному столбцу column_rx и строке line_rx дисплея, учитывая значение регистра количества строк line_counter.

Код модуля top_vga представлен в Приложении А, Листинге А.7.

2.2.9 RTL схемы

На Рисунках 2.5-2.10 показаны RTL схемы модуля клавиатурного тренажера, модуля ресивера PS/2, дешифратора PS/2, VGA и модуля верхнего уровня, описанного ниже.

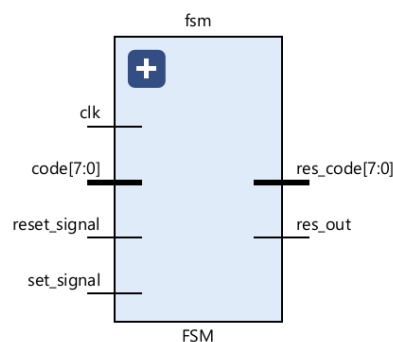


Рисунок 2.5 — RTL схема клавиатурного тренажера (часть 1)

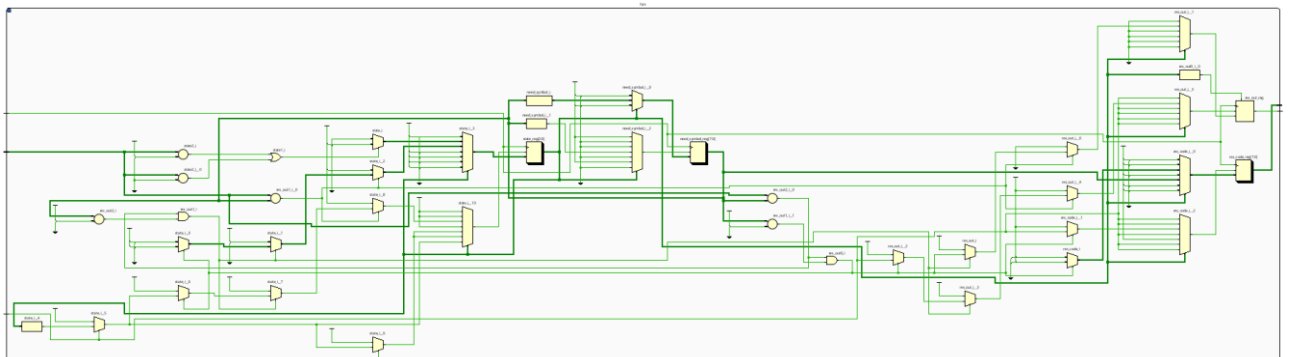


Рисунок 2.6 — RTL схема клавиатурного тренажера (часть 2)

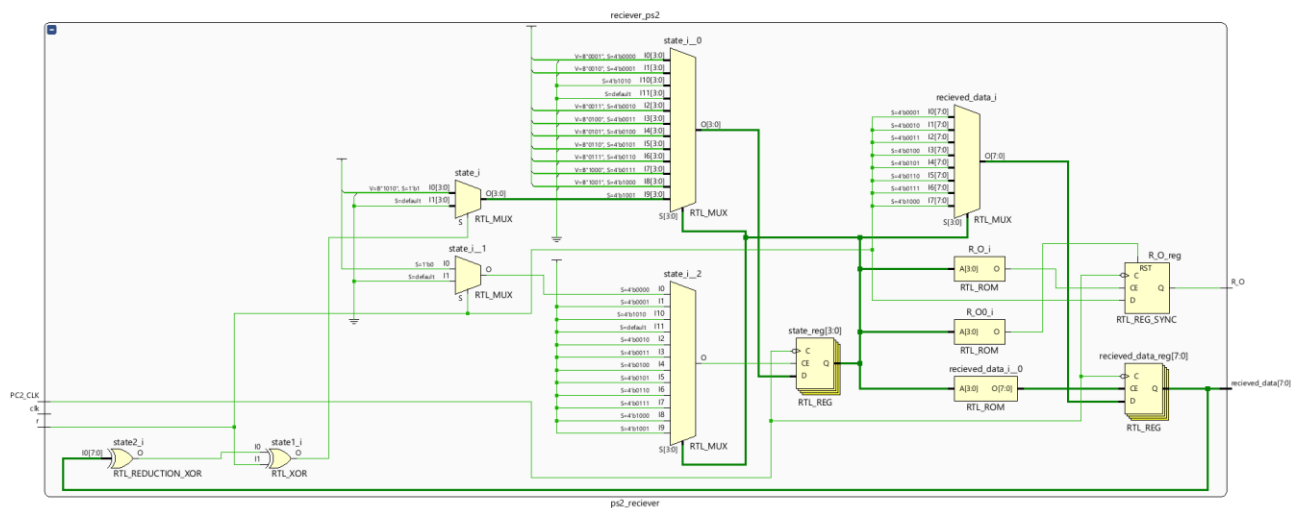


Рисунок 2.7 — RTL схема ресивера PS/2

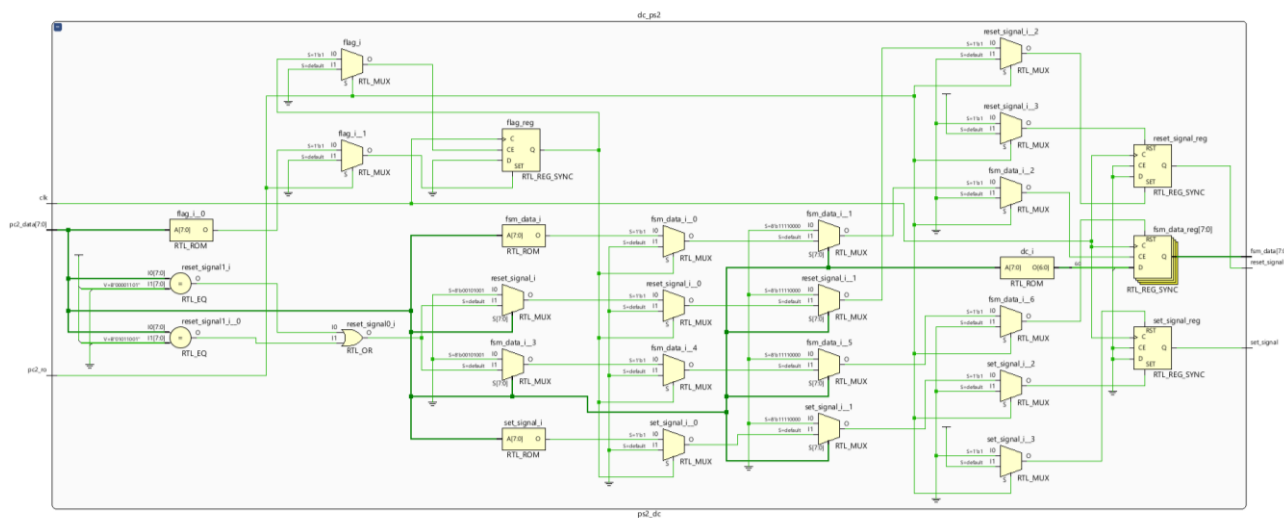


Рисунок 2.8 — RTL схема дешифратора PS/2

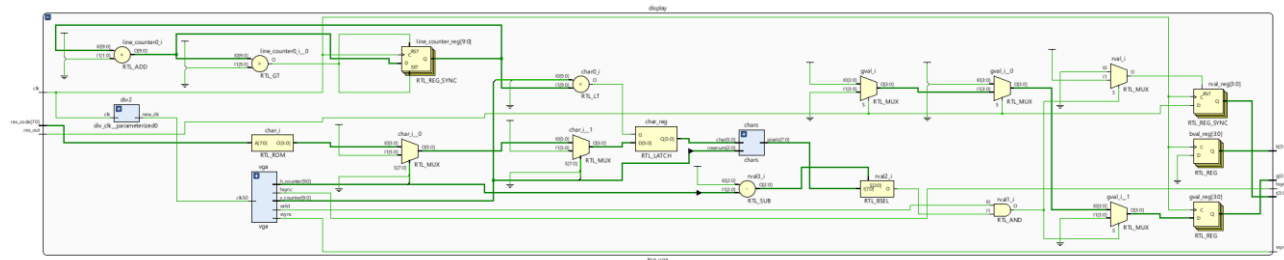


Рисунок 2.9 — RTL схема управляющего модуля VGA

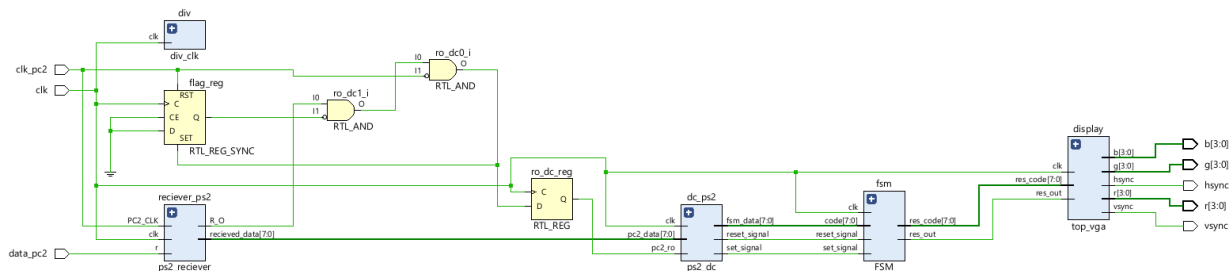


Рисунок 2.10 — RTL схема DEVICE

2.3 Прототипирование на ПЛИС

2.3.1 Реализация модуля верхнего уровня

После разработки всех основных модулей устройства необходимо запустить и протестировать его на ПЛИС, следовательно, нужно провести их подключение для совместной работы в топовом модуле верхнего уровня DEVICE (Приложение А, Листинг А.8).

2.3.2 Файл проектных ограничений

Для организации связи портов модуля верхнего уровня с контактами ПЛИС добавлен файл проектных ограничений (Листинг 2.2).

Листинг 2.2 — Файл проектных ограничений

```
set_property -dict {IOSTANDARD LVCMOS33 PACKAGE_PIN E3} [get_ports {clk}]

set_property -dict {IOSTANDARD LVCMOS33 PACKAGE_PIN F4} [get_ports {clk_pc2}]
set_property -dict {IOSTANDARD LVCMOS33 PACKAGE_PIN B2} [get_ports {data_pc2}]

set_property -dict {IOSTANDARD LVCMOS33 PACKAGE_PIN B11} [get_ports {hsync}]
set_property -dict {IOSTANDARD LVCMOS33 PACKAGE_PIN B12} [get_ports {vsync}]
set_property -dict {IOSTANDARD LVCMOS33 PACKAGE_PIN A3} [get_ports {r[0]}]
set_property -dict {IOSTANDARD LVCMOS33 PACKAGE_PIN B4} [get_ports {r[1]}]
set_property -dict {IOSTANDARD LVCMOS33 PACKAGE_PIN C5} [get_ports {r[2]}]
set_property -dict {IOSTANDARD LVCMOS33 PACKAGE_PIN A4} [get_ports {r[3]}]
set_property -dict {IOSTANDARD LVCMOS33 PACKAGE_PIN C6} [get_ports {g[0]}]
set_property -dict {IOSTANDARD LVCMOS33 PACKAGE_PIN A5} [get_ports {g[1]}]
set_property -dict {IOSTANDARD LVCMOS33 PACKAGE_PIN B6} [get_ports {g[2]}]
set_property -dict {IOSTANDARD LVCMOS33 PACKAGE_PIN A6} [get_ports {g[3]}]
set_property -dict {IOSTANDARD LVCMOS33 PACKAGE_PIN B7} [get_ports {b[0]}]
set_property -dict {IOSTANDARD LVCMOS33 PACKAGE_PIN C7} [get_ports {b[1]}]
set_property -dict {IOSTANDARD LVCMOS33 PACKAGE_PIN D7} [get_ports {b[2]}]
set_property -dict {IOSTANDARD LVCMOS33 PACKAGE_PIN D8} [get_ports {b[3]}]
```

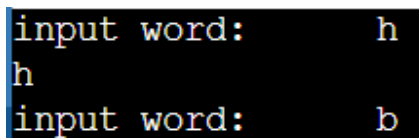
3 ТЕСТИРОВАНИЕ

3.1 Тестирование работы модулей с помощью симуляции

3.1.1 Тестовое покрытие системной модели

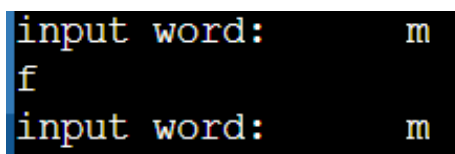
На данном этапе основная цель — проверка различных особенностей работы схемы. Полученные ниже результаты демонстрируют правильно введенный пользователем символ клавиатуры, равный эталонному, и неправильный, а также завершение работы клавиатурного тренажера посредством нажатия клавиши «r».

На Рисунках 3.1-3.3 представлены результаты программной реализации клавиатурного тренажера.



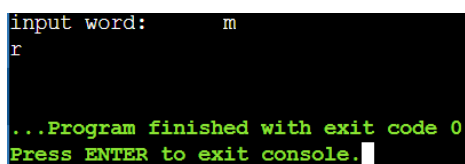
```
input word:      h
h
input word:      b
```

Рисунок 3.1 — Результат при правильно введенном символе



```
input word:      m
f
input word:      m
```

Рисунок 3.2 — Результат при неправильно введенном символе

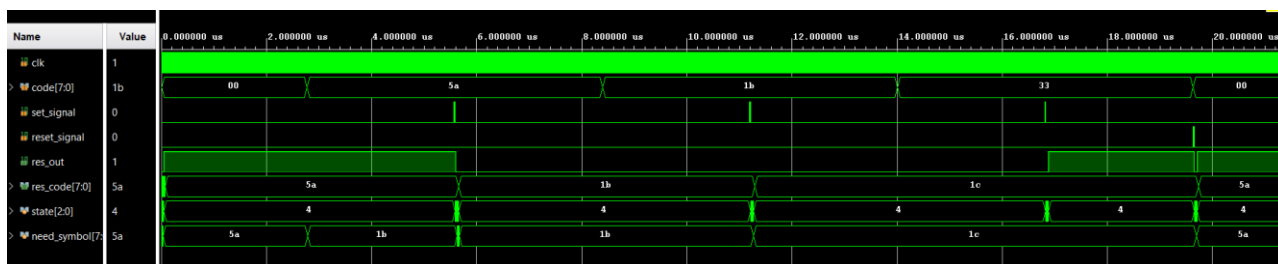


```
input word:      m
r
...Program finished with exit code 0
Press ENTER to exit console.
```

Рисунок 3.3 — Результат при нажатии клавиши сброса «r»

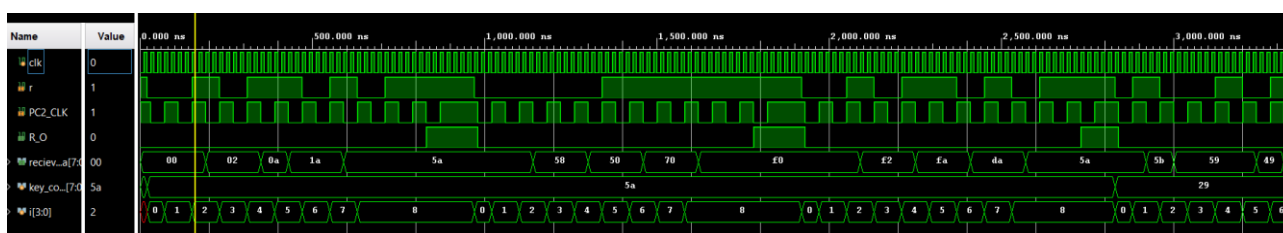
3.1.2 Тестирование модуля клавиатурного тренажера

На Рисунке 3.4 показан скриншот верификации работы модуля, отвечающего за основную логику клавиатурного тренажера (модуль FSM).



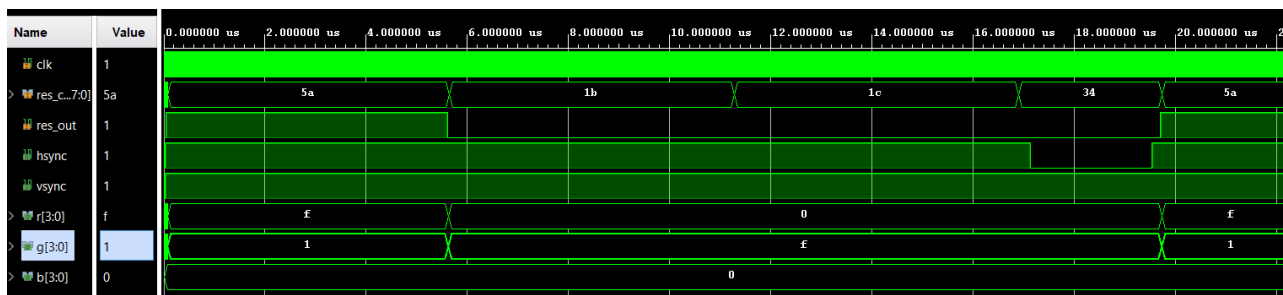
3.1.3 Тестирование модулей PS/2

На Рисунках 3.5-3.6 представлены скриншоты верификации работы клавиатуры по протоколу PS/2.



3.1.4 Тестирование модуля VGA

На Рисунках 3.7 изображен скриншот верификации работы управляющего модуля VGA.



3.1.5 Тестирование модуля верхнего уровня

На Рисунке 3.8 представлен скриншот верификации работы модуля верхнего уровня (модуль DEVICE).

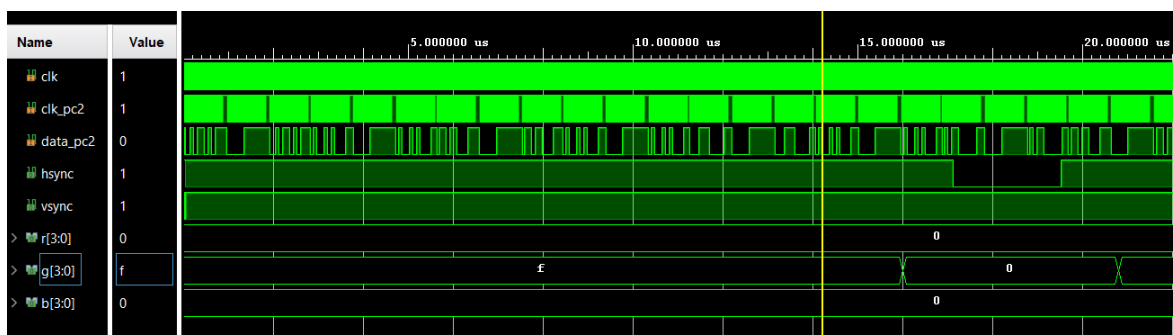


Рисунок 3.8 — Верификация тестового модуля для верхнего уровня

3.2 Тестирование работы устройства на ПЛИС

Финальным этапом тестирования является запуск и проверка работоспособности устройства на ПЛИС. Результат тестирования показан на Рисунках 3.9-3.12.



Рисунок 3.9 — Начальное состояние, клавиатурный тренажер не включен

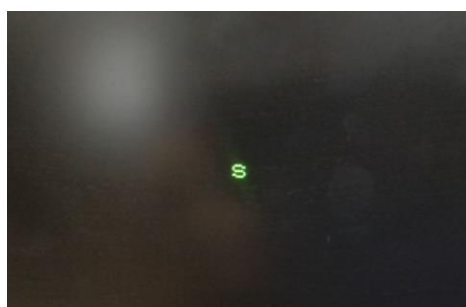


Рисунок 3.10 — Клавиатурный тренажер включен, отображение первого эталонного значения



Рисунок 3.11 — Отображение старого эталонного значения, подсвеченного красным цветом, при неправильном вводе символа

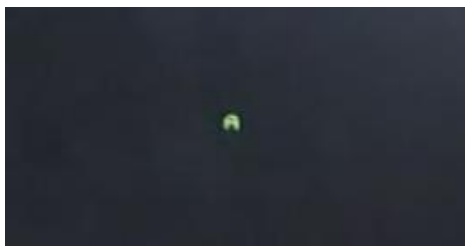


Рисунок 3.12 — Отображение нового эталонного значения при правильном вводе предыдущего

ЗАКЛЮЧЕНИЕ

При создании данного цифрового автомата были получены практические навыки работы с ПЛИС XC7A100TCSG324-1L семейства Artix-7 в составе отладочной платы Xilinx Nexys A7, клавиатурой, работающая по протоколу PS/2, монитора с интерфейсом, работающим по протоколу VGA, языком описания аппаратуры Verilog.

Были выполнены и интегрированы в цифровой автомат следующие задачи: ознакомиться с правилами работы клавиатурного тренажера, провести анализ предметной области; разработать набор управляющих модулей согласно принципам работы клавиатурного тренажера; разработать набор модулей, описывающих процессы приёма и обработки данных по протоколу PS/2; разработать набор модулей, описывающих драйвер для вывода изображения на монитор по протоколу VGA; разработать модуль верхнего уровня, реализующий управление для совместной работы всех модулей.

Работа данного устройства была протестирована с помощью программных средств САПР. Также произведено тестирование прототипа с помощью ПЛИС. Оба тестирования прошли успешно.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Грошев А. С., Закляков П. В. Информатика. Учебник. М.: ДМК Пресс, 2019. 674 с.
2. Строгонов А.В. Системное проектирование программируемых логических интегральных схем: учеб. пособие / А.В. Строгонов. Воронеж: ФГБОУ ВПО «Воронежский государственный технический университет», 2016. 322 с.
3. Языки VHDL и VERILOG в проектировании цифровой аппаратуры. — М. СОЛОН-Пресс, 2017. — 320 с.: ил. — (Серия «Системы проектирования»).
4. Бибило П.Н. Основы VHDL языка. — М.: СОЛОН-Р, 2016.
5. Методические указания по ЛР № 3 — URL: <https://online-edu.mirea.ru/mod/resource/view.php?id=369883> (Дата обращения: 18.12.2022).
6. Антик М.И. Теория автоматов в проектировании цифровых схем [Электронный ресурс]: Учебное пособие / Антик М.И., Казанцева Л.В. — М.: МИРЭА — Российский технологический университет, 2020. — 1 электрон. опт. диск (CD-ROM).
7. Акчурин А.Д., Юсупов К.М. Программирование на языке Verilog. Учебное пособие. — Казань, 2016. — 90 с.
8. Соловьев В.В. Проектирование цифровых систем на основе программируемых логических интегральных схем. — М.: Горячая линия – Телеком, 2016.
9. Стешенко В. Б. ПЛИС фирмы Altera: проектирование устройств обработки сигналов. — М.: ДОДЭКА, 2016.
10. Логическое проектирование и верификация систем на System Verilog / пер. с ант. А. А. Слинкина, А. С. Камкина, М. М. Чупилко; науч. ред. А. С. Камкин, М. М. Чупилко. — М.: ДМК Пресс, 2019. — 384 с.: ил.

ПРИЛОЖЕНИЯ

Приложение А — Код модулей

Приложение Б — Руководство пользователя

Приложение А

Код модулей

Листинг А.1 — Код модуля клавиатурного тренажера FSM

```
`timescale 1ns / 1ps

module FSM(input clk, input [7:0] code, input set_signal, input reset_signal,
output reg res_out, output reg [7:0] res_code);

// code - код нажатой клавиши
// set_signal\reset_signal - сигналы установки\сброса
// res_out - true/false в зависимости от сравнения
// res_code - код эталонной клавиши, который передается в vga для отображения
reg [2:0] state; // состояния автомата
reg [7:0] need_symbol; // эталонное значение, с которым сравнивается нажатая
клавиша

initial
begin
    res_out <= 0;
    state <= 3'd1;
end

always@(posedge clk)
begin
    if (reset_signal) begin
        res_out <= 0;
        res_code <= 8'h5A;
        state <= 3'd1;
    end
    case (state)
        3'd0: // задаем первое "нереальное" эталонное, которое равно
        символу включения клавиатурного тренажера
        begin
            need_symbol[7:0] <= 8'h5A; // тот символ, который запускает
            все - enter
            state <= 3'd2;
        end
        3'd1: // начало
        begin
            res_out <= 0;
            if ((code[7:0] == 8'h5A) || (code[7:0] == 8'h00)) // если
            запускаем (клавиша enter) или неизвестная клавиша
            state <= 3'd0;
            else
            state <= 3'd2;
        end
        3'd2: //сравнение
        begin
            if ((code[7:0] == need_symbol[7:0]) && (need_symbol[7:0] ==
            8'h5A)) begin
                res_code[7:0] <= 8'h1B; // первое нормальное(!) эталонное
                (S), которое и нужно будет нажимать на клавиатуре
                state <= 3'd4;
            end
            if ((code[7:0] == need_symbol[7:0]) && (need_symbol[7:0] !=
            8'h5A)) begin
                state <= 3'd3; // задать новое эталонное
            end
            if (code != need_symbol) begin
```

Продолжение Листинга А.1

```
        res_out <= 1;
        state <= 3'd4; // эталонное остается прежним
    end
end
3'd3: begin //задаем эталонное
    case (need_symbol)
        8'h1B: need_symbol <= 8'h1C; // S->A
        8'h1C: need_symbol <= 8'h34; // A->G
        8'h34: need_symbol <= 8'h4D; // P
        8'h4D: need_symbol <= 8'h45; // 0
        8'h45: need_symbol <= 8'h1D; // W
        8'h1D: need_symbol <= 8'h25; // 4
        8'h25: need_symbol <= 8'h4E; // -
        8'h4E: need_symbol <= 8'h22; // X
        8'h22: need_symbol <= 8'h15; // Q
        8'h15: need_symbol <= 8'h16; // 1
        8'h16: need_symbol <= 8'h2A; // V
        8'h2A: need_symbol <= 8'h3D; // 7
        8'h3D: need_symbol <= 8'h43; // I
        8'h43: need_symbol <= 8'h41; // ,
        8'h41: need_symbol <= 8'h23; // D
        8'h23: need_symbol <= 8'h2E; // 5
        8'h2E: need_symbol <= 8'h3E; // 8
        8'h3E: need_symbol <= 8'h3A; // M
        8'h3A: need_symbol <= 8'h2D; // R
        8'h2D: need_symbol <= 8'h42; // K
        8'h42: need_symbol <= 8'h5B; // ]
        8'h5B: need_symbol <= 8'h32; // B
        8'h32: need_symbol <= 8'h2C; // T
        8'h2C: need_symbol <= 8'h49; // .
        8'h49: need_symbol <= 8'h21; // C
        8'h21: need_symbol <= 8'h54; // [
        8'h54: need_symbol <= 8'h3C; // U
        8'h3C: need_symbol <= 8'h1A; // Z
        8'h1A: need_symbol <= 8'h4C; // ;
        8'h4C: need_symbol <= 8'h46; // 9
        8'h46: need_symbol <= 8'h1E; // 2
        8'h1E: need_symbol <= 8'h52; // '
        8'h52: need_symbol <= 8'h2B; // F
        8'h2B: need_symbol <= 8'h3B; // J
        8'h3B: need_symbol <= 8'h55; // =
        8'h55: need_symbol <= 8'h24; // E
        8'h24: need_symbol <= 8'h4B; // L
        8'h4B: need_symbol <= 8'h31; // N
        8'h31: need_symbol <= 8'h5D; // \
        8'h5D: need_symbol <= 8'h4A; // /
        8'h4A: need_symbol <= 8'h33; // H
        8'h33: need_symbol <= 8'h44; // O
        8'h44: need_symbol <= 8'h26; // 3
        8'h26: need_symbol <= 8'h35; // Y
        8'h35: need_symbol <= 8'h36; // 6
        8'h36: need_symbol <= 8'h1B; // S (заново цикл)
    endcase
    state <= 3'd4;
end
3'd4: begin // результат для vga
    if (code[7:0] == 8'h5A) begin
        need_symbol <= 8'h1B;
    end
    else begin
        res_code[7:0] <= need_symbol[7:0];
    end
end
if (set_signal) begin
```


Продолжение Листинга А.1

```
        state <= 3'd1; // если пришло новое значение символа с клавиатуры
    end
    end
    default: state = 3'd1;
endcase
end
endmodule
```

Листинг А.2 — Код модуля ps2_reciever

```
`timescale 1ns / 1ps

module ps2_reciever (input r, input clk, input PC2_CLK, output reg R_O, output
reg [7:0] recieved_data);

reg [3:0] state;

initial
begin
    state = 4'b0;
    R_O = 0;
    recieved_data = 8'b0;
end

always @(negedge PC2_CLK)
begin
    case (state)
        4'd0:
            begin
                R_O <= 0;
                if (~r) state <= state + 1;
            end
        4'd1:
            begin
                // записываем биты данных
                recieved_data[0] <= r;
                state <= state + 1;
            end
        4'd2:
            begin
                recieved_data[1] <= r;
                state <= state + 1;
            end
        4'd3:
            begin
                recieved_data[2] <= r;
                state <= state + 1;
            end
        4'd4:
            begin
                recieved_data[3] <= r;
                state <= state + 1;
            end
        4'd5:
            begin
                recieved_data[4] <= r;
                state <= state + 1;
            end
        4'd6:
            begin
                recieved_data[5] <= r;
                state <= state + 1;
            end
        4'd7:
            begin
                recieved_data[6] <= r;
                state <= state + 1;
            end
    end
end
```

Продолжение Листинга A.2

```
        begin
            recieved_data[6] <= r;
            state <= state + 1;
        end
        4'd8:
        begin
            recieved_data[7] <= r;
            state <= state + 1;
        end
        4'd9:
        begin // проверяем бит четности
            if (^recieved_data^r)
                state <= state + 1;
            else state <= 0;
        end
        4'd10:
        begin
            R_O <= r ? 1 : 0;
            state <= 0;
        end
        default: state <= 0;
    endcase
end
endmodule
```

Листинг A.3 — Код модуля ps2_dc

```
`timescale 1ns / 1ps

module ps2_dc (input clk, input [7:0] pc2_data, input pc2_ro, output reg [7:0]
fsm_data, output reg set_signal, output reg reset_signal);

    reg flag; // флаг, указывающий на то, что пришел код отжатия

    parameter [7:0]
        releas = 8'hf0, // код отжатия
        s_key = 8'h29, // код клавиши пробел (set)
        r_key = 8'h0D, // код клавиши таб (reset)
        re_key = 8'h59; // код клавиши shift-правый (reset)

    reg [7:0] dc; // код клавиши

    initial
    begin
        fsm_data = 8'b0; // данные (адрес), которые потом будут отправляться в
автомат клавиатурного тренажера
        flag = 0; // флаг, указывающий на то, что пришел код отжатия
        set_signal = 0;
        reset_signal = 0;
        dc = 8'b0;
    end

    always @(*)
    begin
        case (pc2_data)
            // данные dc в зависимости от нажатой клавиши
            8'h15: dc <= 8'h15; // q
            8'h1D: dc <= 8'h1D; // w
            8'h24: dc <= 8'h24; // e
            8'h2D: dc <= 8'h2D; // r
            8'h2C: dc <= 8'h2C; // t
            8'h35: dc <= 8'h35; // y
            8'h3C: dc <= 8'h3C; // u
```

Продолжение Листинга А.3

```
8'h43: dc <= 8'h43; // i
8'h44: dc <= 8'h44; // o
8'h4D: dc <= 8'h4D; // p
8'h1C: dc <= 8'h1C; // a
8'h1B: dc <= 8'h1B; // s
8'h23: dc <= 8'h23; // d
8'h2B: dc <= 8'h2B; // f
8'h34: dc <= 8'h34; // g
8'h33: dc <= 8'h33; // h
8'h3B: dc <= 8'h3B; // j
8'h42: dc <= 8'h42; // k
8'h4B: dc <= 8'h4B; // l
8'h1A: dc <= 8'h1A; // z
8'h22: dc <= 8'h22; // x
8'h21: dc <= 8'h21; // c
8'h2A: dc <= 8'h2A; // v
8'h32: dc <= 8'h32; // b
8'h31: dc <= 8'h31; // n
8'h3A: dc <= 8'h3A; // m
8'h5A: dc <= 8'h5A; // enter
// плюс
8'h45: dc <= 8'h45; // 0
8'h16: dc <= 8'h16; // 1
8'h1E: dc <= 8'h1E; // 2
8'h26: dc <= 8'h26; // 3
8'h25: dc <= 8'h25; // 4
8'h2E: dc <= 8'h2E; // 5
8'h36: dc <= 8'h36; // 6
8'h3D: dc <= 8'h3D; // 7
8'h3E: dc <= 8'h3E; // 8
8'h46: dc <= 8'h46; // 9
8'h41: dc <= 8'h41; // ,
8'h49: dc <= 8'h49; // .
8'h4A: dc <= 8'h4A; // /
8'h4C: dc <= 8'h4C; // ;
8'h52: dc <= 8'h52; // '
8'h4E: dc <= 8'h4E; // -
8'h55: dc <= 8'h55; // =
8'h54: dc <= 8'h54; // [
8'h5B: dc <= 8'h5B; // ]
8'h5D: dc <= 8'h5D; // \
// итого 47 символов + tab + shift + пробел
default: dc <= 0;
endcase
end

always @(posedge clk)
begin
    if (pc2_ro)
    begin
        if (pc2_data == releas) flag <= 1;
        // если флаг уже установлен, то сейчас нам предоставлен код
отжатой клавиши
        else if (flag)
        begin
            flag <= 0;
            if (pc2_data == s_key)
            set_signal <= 1;
            else if ((pc2_data == r_key) || (pc2_data == re_key))
            begin
                reset_signal <= 1;
                fsm_data <= 0;
            end
        end
    end
end
```

Продолжение Листинга А.3

```
        else fsm_data <= dc;
    end
end
else
begin
    set_signal <= 0;
    reset_signal <= 0;
end
end
endmodule
```

Листинг А.4 — Код модуля vga

```
`timescale 1ns / 1ps

module vga(input clk50,
    output reg [9:0] h_counter,
    output reg [9:0] v_counter,
    output hsync,
    output vsync,
    output valid,
    output reg newframe, // 1 clock pulse когда начинается newframe
    output reg newline // 1 clock pulse когда начинается newline
);

// Параметры горизонтальной синхронизации (измеряются в тактах)
parameter [9:0] H_ACTIVE   = 10'd799;
parameter [9:0] H_FRONT   = 10'd55;
parameter [9:0] H_PULSE   = 10'd119;
parameter [9:0] H_BACK    = 10'd63;

// Параметры вертикальной синхронизации (в линиях)
parameter [9:0] V_ACTIVE   = 10'd599;
parameter [9:0] V_FRONT   = 10'd36;
parameter [9:0] V_PULSE   = 10'd5;
parameter [9:0] V_BACK    = 10'd22;

parameter [7:0] H_ACTIVE_STATE = 8'd0;
parameter [7:0] H_FRONT_STATE  = 8'd1;
parameter [7:0] H_PULSE_STATE  = 8'd2;
parameter [7:0] H_BACK_STATE   = 8'd3;
parameter [7:0] V_ACTIVE_STATE = 8'd0;
parameter [7:0] V_FRONT_STATE  = 8'd1;
parameter [7:0] V_PULSE_STATE  = 8'd2;
parameter [7:0] V_BACK_STATE   = 8'd3;

reg [7:0] h_state;
reg [7:0] v_state;
reg hsync_reg;
reg vsync_reg;

initial begin
    h_state = 1'b0;
    v_state = 1'b0;
    hsync_reg = 1'b0;
    vsync_reg = 1'b0;
    h_counter = 1'b0;
    v_counter = 1'b0;
    newframe = 1'b0;
    newline = 1'b0;
end
```

Продолжение Листинга А.4

```
always @(posedge clk50) begin
    newframe <= 1'b0;
    newline <= 1'b0;

    //////горизонтально////////
    case(h_state)
        H_ACTIVE_STATE: begin
            // итерация горизонтального счетчика h_counter, ноль при
H_ACTIVE
            h_counter <= (h_counter==H_ACTIVE)?10'd0:(h_counter + 10'd1);
            // установка hsync в 1
            hsync_reg <= 1'b1;
            // сброс newline
            newline <= 1'b0;
            // переход состояния
            h_state <= (h_counter ==
H_ACTIVE)?H_FRONT_STATE:H_ACTIVE_STATE;
        end

        H_FRONT_STATE: begin
            // итерация горизонтального счетчика h_counter, ноль при
H_FRONT
            h_counter <= (h_counter==H_FRONT)?10'd0:(h_counter + 10'd1);
            // установка hsync в 1
            hsync_reg <= 1'b1;
            // переход в другое состояние
            h_state <= (h_counter == H_FRONT)?H_PULSE_STATE:H_FRONT_STATE;
        end

        H_PULSE_STATE: begin
            // итерация горизонтального счетчика h_counter, ноль при
H_PULSE
            h_counter <= (h_counter==H_PULSE)?10'd0:(h_counter + 10'd1);
            // сброс hsync
            hsync_reg <= 1'b0;
            // переход в другое состояние
            h_state <= (h_counter == H_PULSE)?H_BACK_STATE:H_PULSE_STATE;
        end

        H_BACK_STATE: begin
            // итерация горизонтального счетчика h_counter, ноль при
H_BACK
            h_counter <= (h_counter==H_BACK)?10'd0:(h_counter + 10'd1) ;
            // установка hsync в 1
            hsync_reg <= 1'b1 ;
            // переход в другое состояние
            h_state <= (h_counter == H_BACK)?H_ACTIVE_STATE:H_BACK_STATE ;
            // линия сигнала завершается при переходе состояния (смещение
на 1 для синхронного перехода состояния)
            newline <= (h_counter == (H_BACK-1))?1'b1:1'b0 ;
        end
        default;;
    endcase

    //////вертикально////////
    case(v_state)
        V_ACTIVE_STATE: begin
            // увеличение вертикального счетчика v_counter в конце строки,
ноль при переходе состояния
v_counter<=(newline==1'b1)?((v_counter==V_ACTIVE)?10'd0:(v_counter+10'd1)):v_c
ounter ;

            // установка vsync в активный режим 1
```

Продолжение Листинга А.4

```
        vsync_reg <= 1'b1 ;
        // переход состояния, причем только в конце строк

v_state<=(newline==1'b1)?((v_counter==V_ACTIVE)?V_FRONT_STATE:V_ACTIVE_STATE):
V_ACTIVE_STATE ;
        // сброс newframe
        newframe <= 1'b0 ;
    end

    V_FRONT_STATE: begin
        // увеличение вертикального счетчика v_counter в конце
        строки, ноль при переходе состояния

v_counter<=(newline==1'b1)?((v_counter==V_FRONT)?10'd0:(v_counter +
10'd1)):v_counter ;
        // установка vsync в front porch
        vsync_reg <= 1'b1 ;
        // переход состояния

v_state<=(newline==1'b1)?((v_counter==V_FRONT)?V_PULSE_STATE:V_FRONT_STATE):V_
FRONT_STATE;
    end

    V_PULSE_STATE: begin
        // увеличение вертикального счетчика v_counter в конце строки,
        ноль при переходе состояния

v_counter<=(newline==1'b1)?((v_counter==V_PULSE)?10'd0:(v_counter +
10'd1)):v_counter ;
        // сброс vsync
        vsync_reg <= 1'b0 ;
        // переход состояния

v_state<=(newline==1'b1)?((v_counter==V_PULSE)?V_BACK_STATE:V_PULSE_STATE):V_P
ULSE_STATE;
    end

    V_BACK_STATE: begin
        // увеличение вертикального счетчика v_counter в конце строки,
        ноль при переходе состояния

v_counter<=(newline==1'b1)?((v_counter==V_BACK)?10'd0:(v_counter +
10'd1)):v_counter ;
        // установка vsync в back porch
        vsync_reg <= 1'b1 ;
        // переход состояния

v_state<=(newline==1'b1)?((v_counter==V_BACK)?V_ACTIVE_STATE:V_BACK_STATE):V_B
ACK_STATE ;
        // newframe готов
        newframe <= 1'b1 ;
    end
    default;;
endcase
end

assign hsync = hsync_reg ;
assign vsync = vsync_reg ;
assign valid = (h_state == H_ACTIVE_STATE) && (v_state == V_ACTIVE_STATE);

endmodule
```

Листинг А.5 — Код модуля chars

```
`timescale 1ns / 1ps

module chars(
    input [5:0] char,
    input [2:0] rownum,
    output reg [7:0] pixels
);

    initial pixels = 0;

    always@(*) begin
        case({char, rownum})
            9'b000000_000: pixels = 8'b011111100; // q
            9'b000000_001: pixels = 8'b100000010;
            9'b000000_010: pixels = 8'b100000010;
            9'b000000_011: pixels = 8'b100000010;
            9'b000000_100: pixels = 8'b1000101010;
            9'b000000_101: pixels = 8'b011111100;
            9'b000000_110: pixels = 8'b000000010;
            9'b000000_111: pixels = 8'b000000000;

            9'b000001_000: pixels = 8'b100000010; // w
            9'b000001_001: pixels = 8'b100000010;
            9'b000001_010: pixels = 8'b100000010;
            9'b000001_011: pixels = 8'b100000010;
            9'b000001_100: pixels = 8'b100000010;
            9'b000001_101: pixels = 8'b0101010100;
            9'b000001_110: pixels = 8'b011011100;
            9'b000001_111: pixels = 8'b000000000;

            9'b000010_000: pixels = 8'b111111110; // e
            9'b000010_001: pixels = 8'b100000000;
            9'b000010_010: pixels = 8'b100000000;
            9'b000010_011: pixels = 8'b111111110;
            9'b000010_100: pixels = 8'b100000000;
            9'b000010_101: pixels = 8'b100000000;
            9'b000010_110: pixels = 8'b111111110;
            9'b000010_111: pixels = 8'b000000000;

            9'b000011_000: pixels = 8'b111111100; // r
            9'b000011_001: pixels = 8'b100000010;
            9'b000011_010: pixels = 8'b100000010;
            9'b000011_011: pixels = 8'b111111100;
            9'b000011_100: pixels = 8'b100010100;
            9'b000011_101: pixels = 8'b100000100;
            9'b000011_110: pixels = 8'b100000010;
            9'b000011_111: pixels = 8'b000000000;

            9'b000100_000: pixels = 8'b111111110; // t
            9'b000100_001: pixels = 8'b1011101010;
            9'b000100_010: pixels = 8'b00111000;
            9'b000100_011: pixels = 8'b00111000;
            9'b000100_100: pixels = 8'b00111000;
            9'b000100_101: pixels = 8'b00111000;
            9'b000100_110: pixels = 8'b00111000;
            9'b000100_111: pixels = 8'b000000000;

            9'b000101_000: pixels = 8'b110001110; // y
            9'b000101_001: pixels = 8'b110001110;
            9'b000101_010: pixels = 8'b110001110;
            9'b000101_011: pixels = 8'b011011100;
            9'b000101_100: pixels = 8'b00111000;
            9'b000101_101: pixels = 8'b00111000;
```

Продолжение Листинга А.5

```
9'b000101_110: pixels = 8'b00111000;
9'b000101_111: pixels = 8'b00000000;

9'b000110_000: pixels = 8'b10000010; // u
9'b000110_001: pixels = 8'b10000010;
9'b000110_010: pixels = 8'b10000010;
9'b000110_011: pixels = 8'b10000010;
9'b000110_100: pixels = 8'b10000010;
9'b000110_101: pixels = 8'b10000010;
9'b000110_110: pixels = 8'b01111100;
9'b000110_111: pixels = 8'b00000000;

9'b000111_000: pixels = 8'b11111110; // i
9'b000111_001: pixels = 8'b00111000;
9'b000111_010: pixels = 8'b00111000;
9'b000111_011: pixels = 8'b00111000;
9'b000111_100: pixels = 8'b00111000;
9'b000111_101: pixels = 8'b00111000;
9'b000111_110: pixels = 8'b11111110;
9'b000111_111: pixels = 8'b00000000;

9'b001000_000: pixels = 8'b01111100; // o
9'b001000_001: pixels = 8'b11000110;
9'b001000_010: pixels = 8'b11000110;
9'b001000_011: pixels = 8'b11000110;
9'b001000_100: pixels = 8'b11000110;
9'b001000_101: pixels = 8'b11000110;
9'b001000_110: pixels = 8'b01111100;
9'b001000_111: pixels = 8'b00000000;

9'b001001_000: pixels = 8'b11111100; // p
9'b001001_001: pixels = 8'b10000010;
9'b001001_010: pixels = 8'b10000010;
9'b001001_011: pixels = 8'b11111100;
9'b001001_100: pixels = 8'b10000000;
9'b001001_101: pixels = 8'b10000000;
9'b001001_110: pixels = 8'b10000000;
9'b001001_111: pixels = 8'b00000000;

9'b001010_000: pixels = 8'b00111100; // a
9'b001010_001: pixels = 8'b01000100;
9'b001010_010: pixels = 8'b10000010;
9'b001010_011: pixels = 8'b11111110;
9'b001010_100: pixels = 8'b10000010;
9'b001010_101: pixels = 8'b10000010;
9'b001010_110: pixels = 8'b10000010;
9'b001010_111: pixels = 8'b00000000;

9'b001011_000: pixels = 8'b01111000; // s
9'b001011_001: pixels = 8'b10000110;
9'b001011_010: pixels = 8'b10000000;
9'b001011_011: pixels = 8'b01111100;
9'b001011_100: pixels = 8'b00000010;
9'b001011_101: pixels = 8'b11000010;
9'b001011_110: pixels = 8'b00111100;
9'b001011_111: pixels = 8'b00000000;

9'b001100_000: pixels = 8'b11110000; // d
9'b001100_001: pixels = 8'b10001100;
9'b001100_010: pixels = 8'b10000010;
9'b001100_011: pixels = 8'b10000010;
9'b001100_100: pixels = 8'b10000010;
9'b001100_101: pixels = 8'b10001100;
```


Продолжение Листинга А.5

```
9'b001100_110: pixels = 8'b11111000;
9'b001100_111: pixels = 8'b00000000;

9'b001101_000: pixels = 8'b11111110; // f
9'b001101_001: pixels = 8'b10000000;
9'b001101_010: pixels = 8'b10000000;
9'b001101_011: pixels = 8'b11111110;
9'b001101_100: pixels = 8'b10000000;
9'b001101_101: pixels = 8'b10000000;
9'b001101_110: pixels = 8'b10000000;
9'b001101_111: pixels = 8'b00000000;

9'b001110_000: pixels = 8'b00111000; // g
9'b001110_001: pixels = 8'b01000100;
9'b001110_010: pixels = 8'b10000000;
9'b001110_011: pixels = 8'b10000000;
9'b001110_100: pixels = 8'b10001100;
9'b001110_101: pixels = 8'b01000010;
9'b001110_110: pixels = 8'b00111100;
9'b001110_111: pixels = 8'b00000000;

9'b001111_000: pixels = 8'b11000110; // h
9'b001111_001: pixels = 8'b11000110;
9'b001111_010: pixels = 8'b11000110;
9'b001111_011: pixels = 8'b11111110;
9'b001111_100: pixels = 8'b11000110;
9'b001111_101: pixels = 8'b11000110;
9'b001111_110: pixels = 8'b11000110;
9'b001111_111: pixels = 8'b00000000;

9'b010000_000: pixels = 8'b00001110; // j
9'b010000_001: pixels = 8'b00000100;
9'b010000_010: pixels = 8'b00000100;
9'b010000_011: pixels = 8'b00000100;
9'b010000_100: pixels = 8'b00000100;
9'b010000_101: pixels = 8'b10000100;
9'b010000_110: pixels = 8'b11111000;
9'b010000_111: pixels = 8'b00000000;

9'b010001_000: pixels = 8'b10001000; // k
9'b010001_001: pixels = 8'b10010000;
9'b010001_010: pixels = 8'b10100000;
9'b010001_011: pixels = 8'b11000000;
9'b010001_100: pixels = 8'b10100000;
9'b010001_101: pixels = 8'b10010000;
9'b010001_110: pixels = 8'b10001000;
9'b010001_111: pixels = 8'b00000000;

9'b010010_000: pixels = 8'b01000000; // l
9'b010010_001: pixels = 8'b01000000;
9'b010010_010: pixels = 8'b01000000;
9'b010010_011: pixels = 8'b01000000;
9'b010010_100: pixels = 8'b01000000;
9'b010010_101: pixels = 8'b01000010;
9'b010010_110: pixels = 8'b01111110;
9'b010010_111: pixels = 8'b00000000;

9'b010011_000: pixels = 8'b11111110; // z
9'b010011_001: pixels = 8'b00000110;
9'b010011_010: pixels = 8'b00001000;
9'b010011_011: pixels = 8'b00010000;
9'b010011_100: pixels = 8'b00100000;
9'b010011_101: pixels = 8'b11000000;
```

Продолжение Листинга А.5

```
9'b010011_110: pixels = 8'b11111110;
9'b010011_111: pixels = 8'b00000000;

9'b010100_000: pixels = 8'b10000010; // x
9'b010100_001: pixels = 8'b01000100;
9'b010100_010: pixels = 8'b00101000;
9'b010100_011: pixels = 8'b00010000;
9'b010100_100: pixels = 8'b00101000;
9'b010100_101: pixels = 8'b01000100;
9'b010100_110: pixels = 8'b10000010;
9'b010100_111: pixels = 8'b00000000;

9'b010101_000: pixels = 8'b00011110; // c
9'b010101_001: pixels = 8'b01100000;
9'b010101_010: pixels = 8'b11000000;
9'b010101_011: pixels = 8'b11000000;
9'b010101_100: pixels = 8'b11000000;
9'b010101_101: pixels = 8'b01100000;
9'b010101_110: pixels = 8'b00011110;
9'b010101_111: pixels = 8'b00000000;

9'b010110_000: pixels = 8'b10000010; // v
9'b010110_001: pixels = 8'b10000010;
9'b010110_010: pixels = 8'b01000100;
9'b010110_011: pixels = 8'b01000100;
9'b010110_100: pixels = 8'b00101000;
9'b010110_101: pixels = 8'b00101000;
9'b010110_110: pixels = 8'b00010000;
9'b010110_111: pixels = 8'b00000000;

9'b010111_000: pixels = 8'b11111110; // b
9'b010111_001: pixels = 8'b01100110;
9'b010111_010: pixels = 8'b01100110;
9'b010111_011: pixels = 8'b01111000;
9'b010111_100: pixels = 8'b01100110;
9'b010111_101: pixels = 8'b01100110;
9'b010111_110: pixels = 8'b11111110;
9'b010111_111: pixels = 8'b00000000;

9'b011000_000: pixels = 8'b10000010; // n
9'b011000_001: pixels = 8'b11000010;
9'b011000_010: pixels = 8'b10100010;
9'b011000_011: pixels = 8'b10010010;
9'b011000_100: pixels = 8'b10001010;
9'b011000_101: pixels = 8'b10000110;
9'b011000_110: pixels = 8'b10000010;
9'b011000_111: pixels = 8'b00000000;

9'b011001_000: pixels = 8'b10000010; // m
9'b011001_001: pixels = 8'b11000110;
9'b011001_010: pixels = 8'b10101010;
9'b011001_011: pixels = 8'b10010010;
9'b011001_100: pixels = 8'b10000010;
9'b011001_101: pixels = 8'b10000010;
9'b011001_110: pixels = 8'b10000010;
9'b011001_111: pixels = 8'b00000000;

9'b011010_000: pixels = 8'b11111110; // 0
9'b011010_001: pixels = 8'b10000110;
9'b011010_010: pixels = 8'b10001010;
9'b011010_011: pixels = 8'b10010010;
9'b011010_100: pixels = 8'b10100010;
9'b011010_101: pixels = 8'b11000010;
```

Продолжение Листинга А.5

```
9'b011010_110: pixels = 8'b11111110;
  9'b011010_111: pixels = 8'b00000000;

  9'b011011_000: pixels = 8'b00000001; // 1
  9'b011011_001: pixels = 8'b00000110;
  9'b011011_010: pixels = 8'b00001010;
  9'b011011_011: pixels = 8'b00010010;
  9'b011011_100: pixels = 8'b00000001;
  9'b011011_101: pixels = 8'b00000001;
  9'b011011_110: pixels = 8'b00000001;
  9'b011011_111: pixels = 8'b00000000;

  9'b011100_000: pixels = 8'b11111110; // 2
  9'b011100_001: pixels = 8'b00000001;
  9'b011100_010: pixels = 8'b00000001;
  9'b011100_011: pixels = 8'b11111110;
  9'b011100_100: pixels = 8'b10000000;
  9'b011100_101: pixels = 8'b10000000;
  9'b011100_110: pixels = 8'b11111110;
  9'b011100_111: pixels = 8'b00000000;

  9'b011101_000: pixels = 8'b11111110; // 3
  9'b011101_001: pixels = 8'b00000001;
  9'b011101_010: pixels = 8'b00000001;
  9'b011101_011: pixels = 8'b11111110;
  9'b011101_100: pixels = 8'b00000001;
  9'b011101_101: pixels = 8'b00000001;
  9'b011101_110: pixels = 8'b11111110;
  9'b011101_111: pixels = 8'b00000000;

  9'b011110_000: pixels = 8'b10000001; // 4
  9'b011110_001: pixels = 8'b10000001;
  9'b011110_010: pixels = 8'b10000001;
  9'b011110_011: pixels = 8'b11111110;
  9'b011110_100: pixels = 8'b00000001;
  9'b011110_101: pixels = 8'b00000001;
  9'b011110_110: pixels = 8'b00000001;
  9'b011110_111: pixels = 8'b00000000;

  9'b011111_000: pixels = 8'b11111110; // 5
  9'b011111_001: pixels = 8'b10000000;
  9'b011111_010: pixels = 8'b10000000;
  9'b011111_011: pixels = 8'b11111110;
  9'b011111_100: pixels = 8'b00000001;
  9'b011111_101: pixels = 8'b00000001;
  9'b011111_110: pixels = 8'b11111110;
  9'b011111_111: pixels = 8'b00000000;

  9'b100000_000: pixels = 8'b11111110; // 6
  9'b100000_001: pixels = 8'b10000000;
  9'b100000_010: pixels = 8'b10000000;
  9'b100000_011: pixels = 8'b11111110;
  9'b100000_100: pixels = 8'b10000001;
  9'b100000_101: pixels = 8'b10000001;
  9'b100000_110: pixels = 8'b11111110;
  9'b100000_111: pixels = 8'b00000000;

  9'b100001_000: pixels = 8'b11111110; // 7
  9'b100001_001: pixels = 8'b00000100;
  9'b100001_010: pixels = 8'b00001000;
  9'b100001_011: pixels = 8'b00010000;
  9'b100001_100: pixels = 8'b00100000;
  9'b100001_101: pixels = 8'b01000000;
```

Продолжение Листинга А.5

```
9'b100001_110: pixels = 8'b100000000;
  9'b100001_111: pixels = 8'b000000000;

  9'b100010_000: pixels = 8'b111111110; // 8
  9'b100010_001: pixels = 8'b100000010;
  9'b100010_010: pixels = 8'b100000010;
  9'b100010_011: pixels = 8'b111111110;
  9'b100010_100: pixels = 8'b100000010;
  9'b100010_101: pixels = 8'b100000010;
  9'b100010_110: pixels = 8'b111111110;
  9'b100010_111: pixels = 8'b000000000;

  9'b100011_000: pixels = 8'b111111110; // 9
  9'b100011_001: pixels = 8'b100000010;
  9'b100011_010: pixels = 8'b100000010;
  9'b100011_011: pixels = 8'b111111110;
  9'b100011_100: pixels = 8'b000000010;
  9'b100011_101: pixels = 8'b000000010;
  9'b100011_110: pixels = 8'b111111110;
  9'b100011_111: pixels = 8'b000000000;

  9'b100100_000: pixels = 8'b000000000; // ,
  9'b100100_001: pixels = 8'b000000000;
  9'b100100_010: pixels = 8'b000000000;
  9'b100100_011: pixels = 8'b000000000;
  9'b100100_100: pixels = 8'b011000000;
  9'b100100_101: pixels = 8'b000100000;
  9'b100100_110: pixels = 8'b001000000;
  9'b100100_111: pixels = 8'b000000000;

  9'b100101_000: pixels = 8'b000000000; // .
  9'b100101_001: pixels = 8'b000000000;
  9'b100101_010: pixels = 8'b000000000;
  9'b100101_011: pixels = 8'b000000000;
  9'b100101_100: pixels = 8'b000000000;
  9'b100101_101: pixels = 8'b001110000;
  9'b100101_110: pixels = 8'b001110000;
  9'b100101_111: pixels = 8'b000000000;

  9'b100110_000: pixels = 8'b000000010; // /
  9'b100110_001: pixels = 8'b000001000;
  9'b100110_010: pixels = 8'b000010000;
  9'b100110_011: pixels = 8'b000100000;
  9'b100110_100: pixels = 8'b001000000;
  9'b100110_101: pixels = 8'b010000000;
  9'b100110_110: pixels = 8'b100000000;
  9'b100110_111: pixels = 8'b000000000;

  9'b100111_000: pixels = 8'b000000000; // ;
  9'b100111_001: pixels = 8'b001100000;
  9'b100111_010: pixels = 8'b001100000;
  9'b100111_011: pixels = 8'b000000000;
  9'b100111_100: pixels = 8'b001100000;
  9'b100111_101: pixels = 8'b000010000;
  9'b100111_110: pixels = 8'b000100000;
  9'b100111_111: pixels = 8'b000000000;

  9'b101000_000: pixels = 8'b000001000; // '
  9'b101000_001: pixels = 8'b000001000;
  9'b101000_010: pixels = 8'b000001000;
  9'b101000_011: pixels = 8'b000000000;
  9'b101000_100: pixels = 8'b000000000;
  9'b101000_101: pixels = 8'b000000000;
```

Продолжение Листинга А.5

```
9'b101000_110: pixels = 8'b000000000;
    9'b101000_111: pixels = 8'b000000000;

    9'b101001_000: pixels = 8'b000000000; // -
    9'b101001_001: pixels = 8'b000000000;
    9'b101001_010: pixels = 8'b000000000;
    9'b101001_011: pixels = 8'b111111110;
    9'b101001_100: pixels = 8'b000000000;
    9'b101001_101: pixels = 8'b000000000;
    9'b101001_110: pixels = 8'b000000000;
    9'b101001_111: pixels = 8'b000000000;

    9'b101010_000: pixels = 8'b000000000; // =
    9'b101010_001: pixels = 8'b000000000;
    9'b101010_010: pixels = 8'b111111110;
    9'b101010_011: pixels = 8'b000000000;
    9'b101010_100: pixels = 8'b111111110;
    9'b101010_101: pixels = 8'b000000000;
    9'b101010_110: pixels = 8'b000000000;
    9'b101010_111: pixels = 8'b000000000;

    9'b101011_000: pixels = 8'b111100000; // [
    9'b101011_001: pixels = 8'b100000000;
    9'b101011_010: pixels = 8'b100000000;
    9'b101011_011: pixels = 8'b100000000;
    9'b101011_100: pixels = 8'b100000000;
    9'b101011_101: pixels = 8'b100000000;
    9'b101011_110: pixels = 8'b111100000;
    9'b101011_111: pixels = 8'b000000000;

    9'b101100_000: pixels = 8'b000011110; // ]
    9'b101100_001: pixels = 8'b000000010;
    9'b101100_010: pixels = 8'b000000010;
    9'b101100_011: pixels = 8'b000000010;
    9'b101100_100: pixels = 8'b000000010;
    9'b101100_101: pixels = 8'b000000010;
    9'b101100_110: pixels = 8'b000011110;
    9'b101100_111: pixels = 8'b000000000;

    9'b101101_000: pixels = 8'b100000000; // \
    9'b101101_001: pixels = 8'b010000000;
    9'b101101_010: pixels = 8'b001000000;
    9'b101101_011: pixels = 8'b000010000;
    9'b101101_100: pixels = 8'b000001000;
    9'b101101_101: pixels = 8'b000000100;
    9'b101101_110: pixels = 8'b000000010;
    9'b101101_111: pixels = 8'b000000000;

    9'b111111_000: pixels = 8'b000000000; // null
    9'b111111_001: pixels = 8'b000000000;
    9'b111111_010: pixels = 8'b000000000;
    9'b111111_011: pixels = 8'b000000000;
    9'b111111_100: pixels = 8'b000000000;
    9'b111111_101: pixels = 8'b000000000;
    9'b111111_110: pixels = 8'b000000000;
    9'b111111_111: pixels = 8'b000000000;
    default: pixels = 8'b000000000;

endcase
end
endmodule
```

Листинг А.6 — Код модуля div_clk

```
`timescale 1ns / 1ps

module div_clk #(parameter MAX = 15) (
    input clk,
    output new_clk);

    reg [MAX:0] cnt = 0;
    assign new_clk = cnt[MAX];
    always@(posedge clk) begin
        cnt <= cnt + 1'b1;
    end
endmodule
```

Листинг А.7 — Код модуля top_vga

```
`timescale 1ns / 1ps

module top_vga(input clk, input [7:0] res_code, input res_out, output hsync,
output vsync, output [3:0] r, output [3:0] g, output [3:0] b);

parameter [9:0] WIDTH = 10'd80; // 800 = 80 column
parameter [9:0] HEIGHT = 10'd60; // 600 = 60 lines

wire clk50; // 50МГц (clk делится на 2)
div_clk #(.MAX(1'd0)) div2(.clk(clk), .new_clk(clk50));

wire [9:0] x;
wire [9:0] y;
wire newframe;
wire newline;
wire valid;
reg [3:0] rval;
reg [3:0] gval;
reg [3:0] bval;

vga vga(
    .clk50(clk50),
    .h_counter(x),
    .v_counter(y),
    .hsync(hsync),
    .vsync(vsync),
    .valid(valid),
    .newframe(newframe),
    .newline(newline));

wire [7:0] pixels;
wire [31:0] line;
reg [5:0] char;

wire [7:0] line_px;
wire [7:0] column_px;

assign line_px = {1'b0,y[9:3]};
assign column_px = {1'b0,x[9:3]};

reg [9:0] line_counter;

initial begin
    char = 6'b0;
    line_counter = 10'b1;
    rval = 4'b0;
    bval = 4'b0;
    gval = 4'b0;
end
```

Продолжение Листинга А.7

```
always @(*) begin
  if({2'b00,line_px} < line_counter) begin
    case (line_px)
      8'd37:
        case (column_px)
          8'd50: begin
            case (res_code)
              8'h15: char = 6'b0000000; // q
              8'h1D: char = 6'b0000001; // w
              8'h24: char = 6'b0000010; // e
              8'h2D: char = 6'b0000011; // r
              8'h2C: char = 6'b0000100; // t
              8'h35: char = 6'b0000101; // y
              8'h3C: char = 6'b0000110; // u
              8'h43: char = 6'b0000111; // i
              8'h44: char = 6'b0001000; // o
              8'h4D: char = 6'b0001001; // p
              8'h1C: char = 6'b0001010; // a
              8'h1B: char = 6'b0001011; // s
              8'h23: char = 6'b0001100; // d
              8'h2B: char = 6'b0001101; // f
              8'h34: char = 6'b0001110; // g
              8'h33: char = 6'b0001111; // h
              8'h3B: char = 6'b0100000; // j
              8'h42: char = 6'b0100001; // k
              8'h4B: char = 6'b0100010; // l
              8'h1A: char = 6'b0100011; // z
              8'h22: char = 6'b0101000; // x
              8'h21: char = 6'b0101001; // c
              8'h2A: char = 6'b0101010; // v
              8'h32: char = 6'b0101011; // b
              8'h31: char = 6'b0101000; // n
              8'h3A: char = 6'b0101001; // m
              8'h45: char = 6'b0101010; // 0
              8'h16: char = 6'b0101011; // 1
              8'h1E: char = 6'b0101100; // 2
              8'h26: char = 6'b0101101; // 3
              8'h25: char = 6'b0101110; // 4
              8'h2E: char = 6'b0101111; // 5
              8'h36: char = 6'b1000000; // 6
              8'h3D: char = 6'b1000001; // 7
              8'h3E: char = 6'b1000010; // 8
              8'h46: char = 6'b1000011; // 9
              8'h41: char = 6'b1000100; // ,
              8'h49: char = 6'b1000101; // .
              8'h4A: char = 6'b1000110; // /
              8'h4C: char = 6'b1000111; // ;
              8'h52: char = 6'b1010000; // '
              8'h4E: char = 6'b1010001; // -
              8'h55: char = 6'b1010010; // =
              8'h54: char = 6'b1010011; // [
              8'h5B: char = 6'b1010100; // ]
              8'h5D: char = 6'b1010101; // \
              default: char = 6'b1111111;
            endcase
          end
          default: char = 6'b1111111;
        endcase
      endcase
    default: char = 6'b1111111;
  endcase
end
end
```

Продолжение Листинга А.7

```
chars chars(  
    .char(char),  
    .rownum(y[2:0]),  
    .pixels(pixels)  
);  
  
always@(posedge clk) begin  
    rval = 4'b0;  
    bval = 4'b0;  
    gval = 4'b0;  
    if (line_counter +10'd2 > HEIGHT) line_counter <= 10'hff;  
        else line_counter <= line_counter + 10'd2;  
    if (valid) begin  
        if (res_out == 0) begin  
            rval <= 4'b0000;  
            gval <= 4'b1111;  
            bval <= 4'b0000;  
        end  
        if (res_out == 1) begin  
            rval <= 4'b1111;  
            gval <= 4'b0001;  
            bval <= 4'b0000;  
        end  
    end  
end  
  
assign r = rval;  
assign g = gval;  
assign b = bval;  
endmodule
```

Листинг А.8 — Код модуля верхнего уровня DEVICE

```
`timescale 1ns / 1ps  
  
module DEVICE(  
    input clk,  
    input clk_pc2,  
    input data_pc2,  
    output hsync,  
    output vsync,  
    output [3:0] r,  
    output [3:0] g,  
    output [3:0] b  
);  
  
wire new_clk;  
div_clk div(.clk(clk), .new_clk(new_clk));  
  
wire [7:0] recieved_data; // то что выходит из ресивера для дешифратора  
wire [7:0] inForFSM; // то что выходит из дешифратора для автомата  
wire [7:0] res_code; // то что выходит из автомата для вга  
wire set_signal;  
wire reset_signal;  
reg flag = 0;  
wire pc2_ro;  
reg ro_dc = 0;  
wire res_out;  
  
ps2_reciever reciever_ps2(  
    .r(data_pc2),  
    .clk(clk),  
    .PC2_CLK(clk_pc2),  
    .R_O(pc2_ro),
```


Продолжение Листинга А.8

```
.recieved_data(recieved_data));

ps2_dc dc_ps2 (
    .clk(clk),
    .pc2_data(recieved_data),
    .pc2_ro(ro_dc),
    .fsm_data(inForFSM),
    .set_signal(set_signal),
    .reset_signal(reset_signal));

FSM fsm(
    .clk(clk),
    .set_signal(set_signal),
    .reset_signal(reset_signal),
    .code(inForFSM),
    .res_code(res_code),
    .res_out(res_out));

top_vga display(
    .clk(clk),
    .res_out(res_out),
    .res_code(res_code),
    .hsync(hsync),
    .vsync(vsync),
    .r(r),
    .g(g),
    .b(b));

always @(posedge clk)
begin
    if (pc2_ro&~flag&~clk_pc2) begin
        ro_dc <= 1;
        flag <= 1;
    end
    else ro_dc <= 0;
    if (clk_pc2) flag <= 0;
end

endmodule
```

Приложение Б

Руководство пользователя

Клавиатурный тренажер должен быть включен пользователем посредством последовательного нажатия клавиш «Enter» и «Spacebar». После на экране загорится, так называемое, «эталонное» значение, которое человеку требуется ввести на клавиатуре.

Для обучения слепой печати и прививания навыка быстрого и безошибочного набора человеку необходимо расположить руки согласно конкретному начальному положению следующим образом: пальцы (кроме больших) левой руки — на символы «A S D F», правой руки — на «J K L ;». Большие пальцы отвечают за пробел. Нажимать клавишу стоит без подглядывания на нее соответствующим пальцем одной из рук, исходя из правил слепого метода.

Итак, пользователь должен нажать символ, идентичный символу на экране, после чего нажать пробел для подтверждения выбора клавиши. Клавиатурный тренажер засчитает только ту клавишу, которая была непосредственно до пробела. Если это верная клавиша, то клавиатурный тренажер поменяет символ на экране, иначе символ останется прежним до момента правильного нажатия. Подобным образом с помощью тренажера человек пройдет по всем английским буквам клавиатуры, цифрам и таким символам, как «-», «=», «[», «]», «;», «'», «,», «.», «/», «\», столько кругов, сколько захочет.

Для выключения клавиатурного тренажера или его сброса с целью начать заново можно нажать специальные клавиши «Tab» и правый «Shift».