|  |
| --- |
|  |
| МИНОБРНАУКИ РОССИИ |
| Федеральное государственное бюджетное образовательное учреждение  высшего образования  **«МИРЭА - Российский технологический университет»**  **РТУ МИРЭА** |

Институт Информационных Технологий

Кафедра Вычислительной Техники (ВТ)

**КУРСОВАЯ РАБОТА**

по дисциплине «Схемотехника устройств компьютерных систем»

**Тема курсовой работы:** «Волк ловит яйца»

|  |  |
| --- | --- |
| **Студент группы** ИВБО-04-20 Аксёнов Александр Алексеевич  (учебная группа, фамилия, имя, отчество студента) | (подпись студента) |
|  |  |
| **Руководитель курсовой работы**  ассистент кафедры ВТ Люлява Д.В.  (должность, звание, ученая степень) | (подпись руководителя) |
| Работа представлена к защите «\_\_\_\_\_» \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ 2022 г. |  |

Допущен к защите «\_\_\_\_\_» \_\_\_\_\_­­\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_2022 г.

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| |  |  |  | | --- | --- | --- | |  |  |  | | МИНОБРНАУКИ РОССИИ | | | | Федеральное государственное бюджетное образовательное учреждение  высшего образования  **«МИРЭА – Российский технологический университет»**  **РТУ МИРЭА** | | |   Институт информационных технологий | |
| Кафедра вычислительной техники | |
|  | **Утверждаю** |
|  | ассистент кафедры ВТ  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Люлява Д.В.  *Подпись ФИО* |
|  | «\_\_\_\_» \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ 2022г. |
| **ЗАДАНИЕ** | |
| **на выполнение курсовой работы по дисциплине** | |
| «**Схемотехника устройств компьютерных систем**» | |
|  | |

**Студент:** Аксёнов Александр Алексеевич **Группа:** ИВБО-04-20

# Тема: «Волк ловит яйца»

**Исходные данные:**

◦ ПЛИС XC7A100TCSG324-1L семейства Artix-7 в составе отладочной платы Xilinx Nexys A7

◦ Клавиатура, работающая по протоколу PS/2

◦ Монитор с интерфейсом, работающим по протоколу VGA.

◦ Маршрут проектирования СБИС

◦ Язык описания аппаратуры Verilog

**Задание:**

1. Ознакомиться с правилами игры, провести анализ предметной области.

2. Разработать набор управляющих модулей согласно правилам игры.

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.

8. Составить отчетную документацию по проделанной работе.

|  |
| --- |
| **Срок представления к защите курсовой работы: до** **«\_\_\_» \_\_\_\_\_\_\_\_\_\_\_ 2022 г.** |
|  |
| **Задание на выполнение курсовой работы выдал \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**  (Люлява Д.В.)  *Подпись руководителя Ф.И.О. руководителя* |
|  |

**«\_\_\_» \_\_\_\_\_\_\_\_\_\_\_ 2022 г.**

**Задание на курсовую работу получил** \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ (Аксёнов А.А.)

*Подпись обучающегося* *Ф.И.О. исполнителя*

**«\_\_\_» \_\_\_\_\_\_\_\_\_\_\_ 2022 г.**

### ОТЗЫВ

### на курсовую работу

**по дисциплине**

«Схемотехника устройств компьютерных систем»

|  |  |  |  |
| --- | --- | --- | --- |
| **Студент(ка)** | Аксёнов Александр Алексеевич | **Группа** | ИВБО-04-20 |

Характеристика курсовой работы

|  |  |  |  |
| --- | --- | --- | --- |
| Критерий | Да | Нет | Не полностью |
| 1. Соответствие содержания курсовой работы указанной теме |  |  |  |
| 1. Соответствие курсовой работы заданию |  |  |  |
| 1. Соответствие рекомендациям по оформлению текста, таблиц, рисунков и пр. |  |  |  |
| 1. Полнота выполнения всех пунктов задания |  |  |  |
| 1. Логичность и системность содержания курсовой работы |  |  |  |
| 1. Отсутствие фактических грубых ошибок |  |  |  |

Рекомендуемая оценка: удовлетворительно, хорошо, отлично

|  |  |  |
| --- | --- | --- |
|  | *Подпись руководителя* | Люлява Д.В.  *(ФИО руководителя)* |
| «\_\_\_»\_\_\_\_\_\_\_\_\_\_\_\_\_2022 г. | | |

АННОТАЦИЯ

Согласно теме курсовой работы, была разработана системная модель игры «Волк ловит яйца», на высокоуровневом языке программирования JAVA.

После прохождения предварительного тестирования, на основе системной модели, были созданы модули соответствующих периферийных устройств на языке описания аппаратуры Verilog и разработан конечный автомат, выполняющий основную логику алгоритма. Все модули были протестированы с использование временной диаграммы, а также был сформирован файл проектных ограничений для загрузки на отладочную плату. Тестирования на отладочной плате провести не удалось из-за технических ограничений и неполадок.

Данная работа включает в себя 13 рисунков, 11 листингов, 17 приложений. Количество страниц в работе — 40.

СОДЕРЖАНИЕ

[ВВЕДЕНИЕ 7](#_Toc122914463)

[1 ТЕОРЕТИЧЕСКАЯ ЧАСТЬ 8](#_Toc122914465)

[1.1 Описание основного алгоритма игры](#_Toc122914466) 7

[1.2 Описание работы протокола PS/2](#_Toc122914466) 8

[1.3 Описание работы протокола VGA](#_Toc122914466) 8

[2 ПРАКТИЧЕСКАЯ ЧАСТЬ](#_Toc122914465) 9

[2.1 Cоздание системной модели](#_Toc122914466) 10

[2.2 Cоздание RTL-модели](#_Toc122914466) 12

[2.2.1 Модуль PS/2](#_Toc122914466) 13

[2.2.2 Модуль конечного автомата](#_Toc122914466) 13

[2.2.3 Модуль VGA](#_Toc122914466) 13

[ТЕСТИРОВАНИЕ](#_Toc122914466) 14

[ЗАКЛЮЧЕНИЕ](#_Toc122914466) 17

СПИСОК ИНФОРМАЦИОННЫХ ИСТОЧНИКОВ18

[ПРИЛОЖЕНИЯ](#_Toc122914466) 19

[Приложение А1](#_Toc122914466) 20

[Приложение А2](#_Toc122914466) 24

[Приложение А3](#_Toc122914466) 26

[Приложение А4](#_Toc122914466) 27

[Приложение А5](#_Toc122914466) 29

[Приложение А6](#_Toc122914466) 31

[Приложение А7](#_Toc122914466) 32

[Приложение Б1](#_Toc122914466) 33

[Приложение Б2](#_Toc122914466) 34

[Приложение Б3](#_Toc122914466) 34

[Приложение Б4](#_Toc122914466) 35

[Приложение В1](#_Toc122914466) 35

[Приложение В2](#_Toc122914466) 38

[Приложение В3](#_Toc122914466) 38

[Приложение Д1](#_Toc122914466) 39

[Приложение Д2](#_Toc122914466) 39

**ВВЕДЕНИЕ**

Разработка видеоигр – это широкая сфера It-индустрии, в которой используются одни из самых передовых технологий. Одну из ведущих ролей при проектировании продуктов в данной отрасли, играет создание полноценной игровой платформы, реализующей в себе основные функции, по взаимодействию с игроком, и обработке входящий информации[1]. Отличным решением в данной ситуации может послужить перенос игровой платформы на программируемую логическую интегральную схему, в виду её гибкости в плане различного уровня дополнительных настроек и портативности получаемого устройства. Также есть возможность использовать широкий пласт компонентов так, или иначе расширяющих возможности по взаимодействию с конечным пользователем. В качестве примера устройства созданного на основе интегральной схемы можно вспомнить старую советскую игру под названием «Ну погоди!» с микропроцессором: КБ1013ВК1-2 [2]. Игра имела огромную популярность в 90-х, но немного устарела по нынешним меркам. Поэтому небольшие изменения в алгоритме и перенос на новую платформу, способны дать игре второе дыхание и оживить общественный интерес.

# 1 ТЕОРЕТИЧЕСКАЯ ЧАСТЬ

**1.1 Описание основного алгоритм игры**

Основной алгоритм игры состоит в постоянном отслеживании координат волка, которым непосредственно управляет игрок, а также отслеживания соприкосновения данных координат с координатами яйца, для возможности установки факта поимки яйца. Без учёта графической составляющей можно представить волка в виде платформы, которая имеет возможность двигаться, как по оси x, так и по оси y, а яйцо в виде точки, с постоянной изменяющийся координатой y. После поимки происходит генерация новой горизонтальной координаты для яйца, и возвращение его в верхнею область экрана. Также отдельно увеличивается показатель счётчика пойманных яйиц. Если яйцо не было поймано в соответствующий момент и достигло предельной нижней границы, то игра автоматически завещается, а управление волком становится недоступным. С учётом графической, составляющей стоит делать поправку на определение координат волка и яйца, и брать за основу координату левого верхнего угла соответствующих фигур.

В механику поимки яйца тоже нужно вносить корректировки, допуская полное совпадение координат только по оси y, cответственно по оси х, координата яйца должна будет находится в пределах интервала, который определяется левым и правом углом корзины в руках игрового персонажа.

**1.2 Описание работы протокола PS/2**

PS/2 — специализированный [компьютерный порт](https://ru.wikipedia.org/wiki/%D0%90%D0%BF%D0%BF%D0%B0%D1%80%D0%B0%D1%82%D0%BD%D1%8B%D0%B9_%D0%BF%D0%BE%D1%80%D1%82), применяемый для подключения [клавиатуры](https://ru.wikipedia.org/wiki/%D0%9A%D0%BB%D0%B0%D0%B2%D0%B8%D0%B0%D1%82%D1%83%D1%80%D0%B0_IBM_PC) и [мыши](https://ru.wikipedia.org/wiki/%D0%9A%D0%BE%D0%BC%D0%BF%D1%8C%D1%8E%D1%82%D0%B5%D1%80%D0%BD%D0%B0%D1%8F_%D0%BC%D1%8B%D1%88%D1%8C), использующий 6-контактный разъём [mini-DIN](https://ru.wikipedia.org/wiki/Mini-DIN" \o "Mini-DIN).  При передаче от устройства (PS/2-клавиатуры или PS/2-мыши) к компьютеру используется следующий протокол. Устройство не начинает передачу, если Clock не находился в «1» по крайней мере 50 микросекунд. Устройство передаёт последовательно:

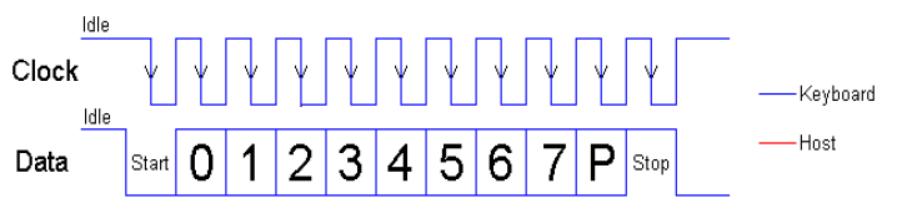
старт-бит — всегда ноль;

8 бит данных;

бит чётности;

стоп-бит — всегда единица.

Устройство устанавливает/меняет сигнал Data, когда Clock находится в логической единице. Контроллер на материнской плате читает данные, когда Clock находится в логическом нуле [3]. Детальное описание портов и кодов соответствующих клавиш представлено в методических рекомендациях ко второй лабораторной работе [4].

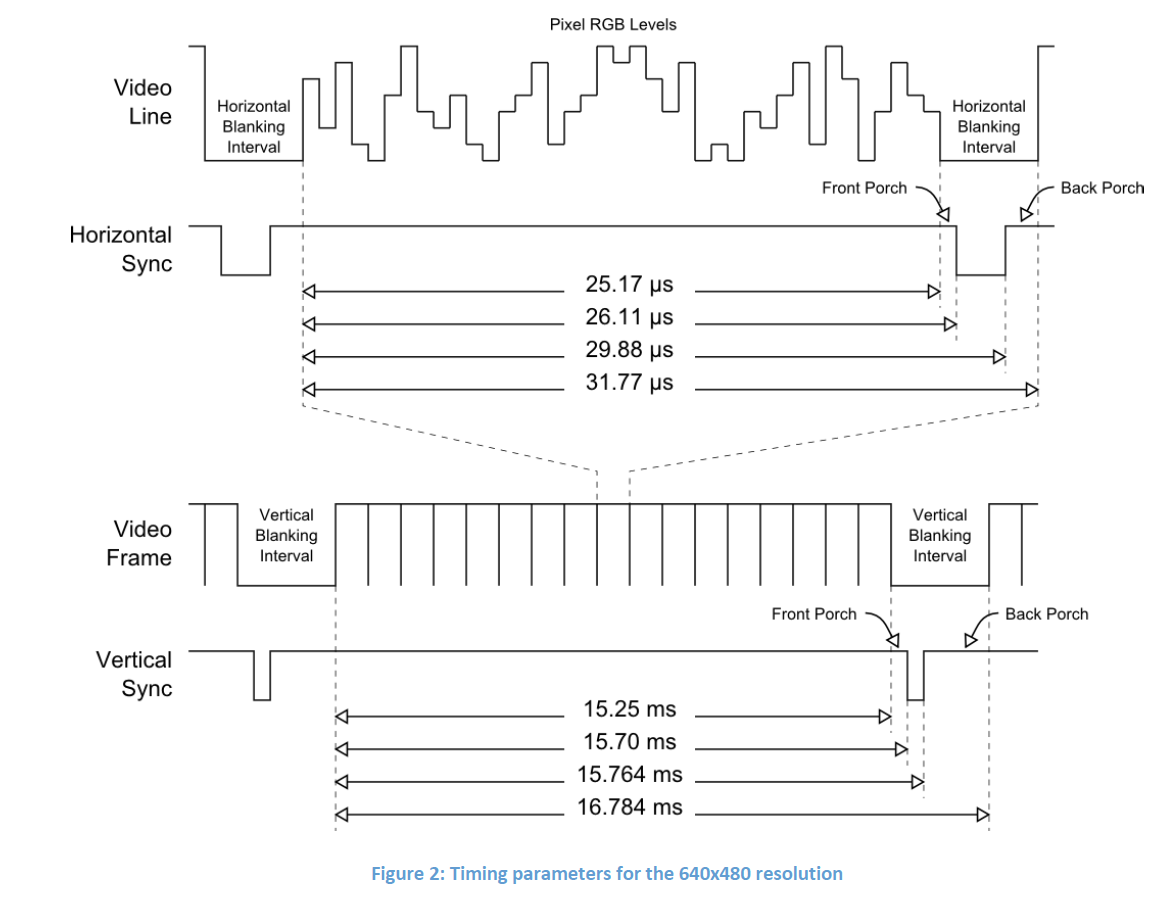


**Рисунок 1.1 – Передача данных по протоколу ps/2**

**1.3 Описание работы протокола VGA**

**VGA** ([англ.](https://ru.wikipedia.org/wiki/%D0%90%D0%BD%D0%B3%D0%BB%D0%B8%D0%B9%D1%81%D0%BA%D0%B8%D0%B9_%D1%8F%D0%B7%D1%8B%D0%BA) *Video Graphics Array*) — компонентный видеоинтерфейс, используемый в мониторах и видеоадаптерах. Выпущен IBM в 1987 году для компьютеров PS/2 Model 50 и более старших.

Ключевое отличие VGA от предыдущего видеоадаптера — аналоговый сигнала для передачи цветовой информации. Переход на аналоговый сигнал был обусловлен необходимостью сокращения числа проводов в кабеле. Также аналоговый сигнал давал возможность использовать VGA-мониторы с последующими видеоадаптерами, которые могут выводить большее количество цветов (например, HDMI) [5]. Отображение картинки происходит побитно, с горизонтальной и вертикальной синхронизацией кадра. Более детальное описание портов и соответствующих временных показателей для каждого разрешения представлено в четвертой лабораторной работе [6].

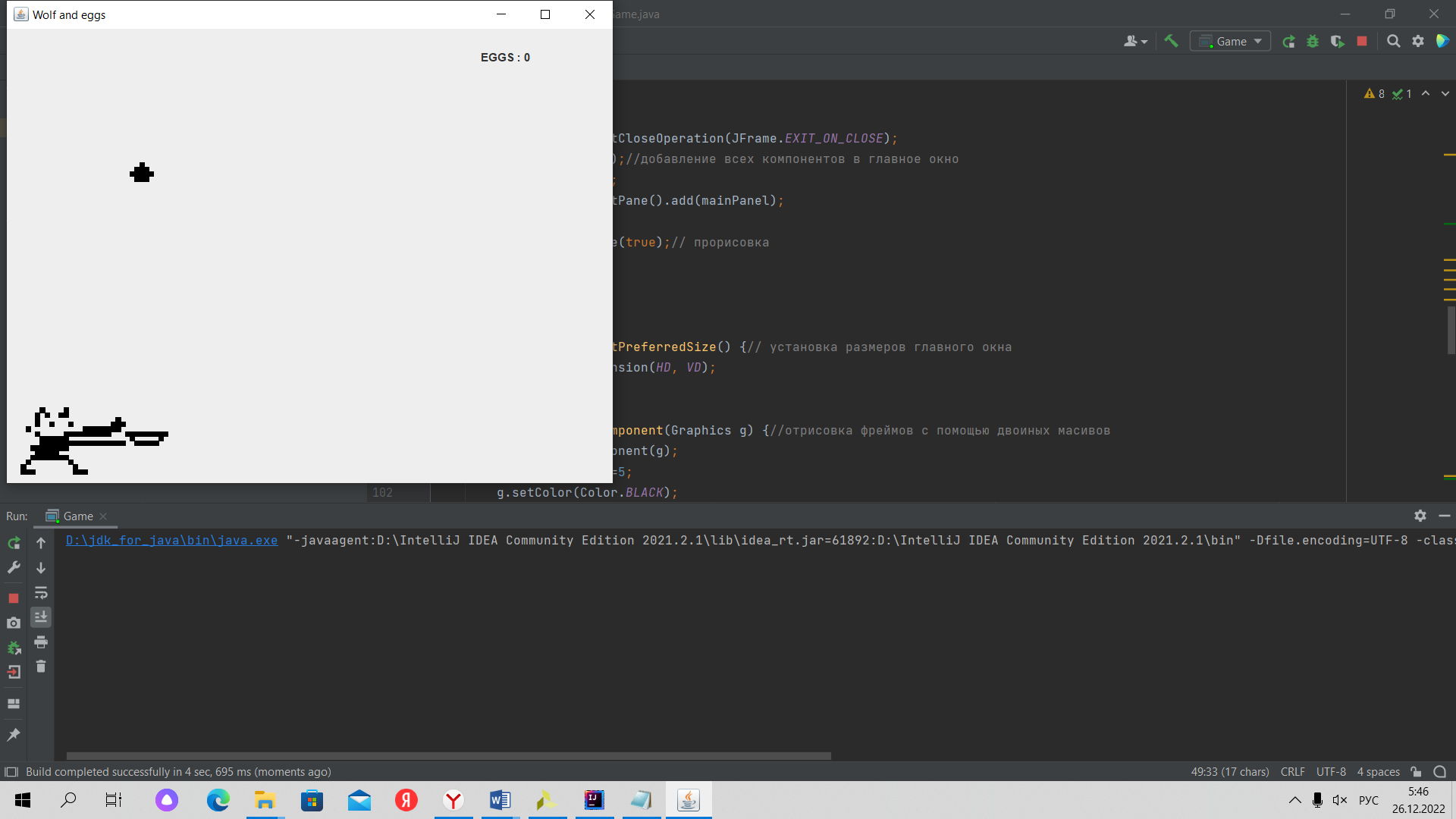


**Рисунок 1.2 – Временная диаграмма VGA**

# 2 ПракТИЧЕСКАЯ ЧАСТЬ

**2.1 Создание системной модели**

Для проверки основного алгоритма лучшем решением будет представление системной модели в виде игры, написанной на языке программирования высокого уровня [7]. В качестве инструмента был выбран язык JAVA, в следствии удобного взаимодействия с графической библиотекой и удобной реализацией объектно-ориентированного программирования. В основе программы лежит класс Game наследующий методы от JComponet (Класс графического компонента), где переопределяется метод paintComponent() (метод отображения компонента). В данном методе с помощью двухмерных массивов, задаются пиксельный изображения волка и яйца, а также происходит отрисовка соответствующих битов во вложенном цикле. Также здесь изображаются графические границы разрешения. В методе createAndShowGui() происходит создание окна для вывода изображения, соответствующих заголовков для отображения количества пойманных яиц, и отображения проигрыша, также к окну добавляются «слушатели» клавиатуры, для каждой кнопки задаётся взаимодействие, и устанавливаются границы. Метод getPreferedSize отдаёт размеры создаваемого окна, а в конструкторе класса происходит установка таймера отрисовки кадра, также происходид проверка на совпадение координат яйца и волка, и индикация провала игры.

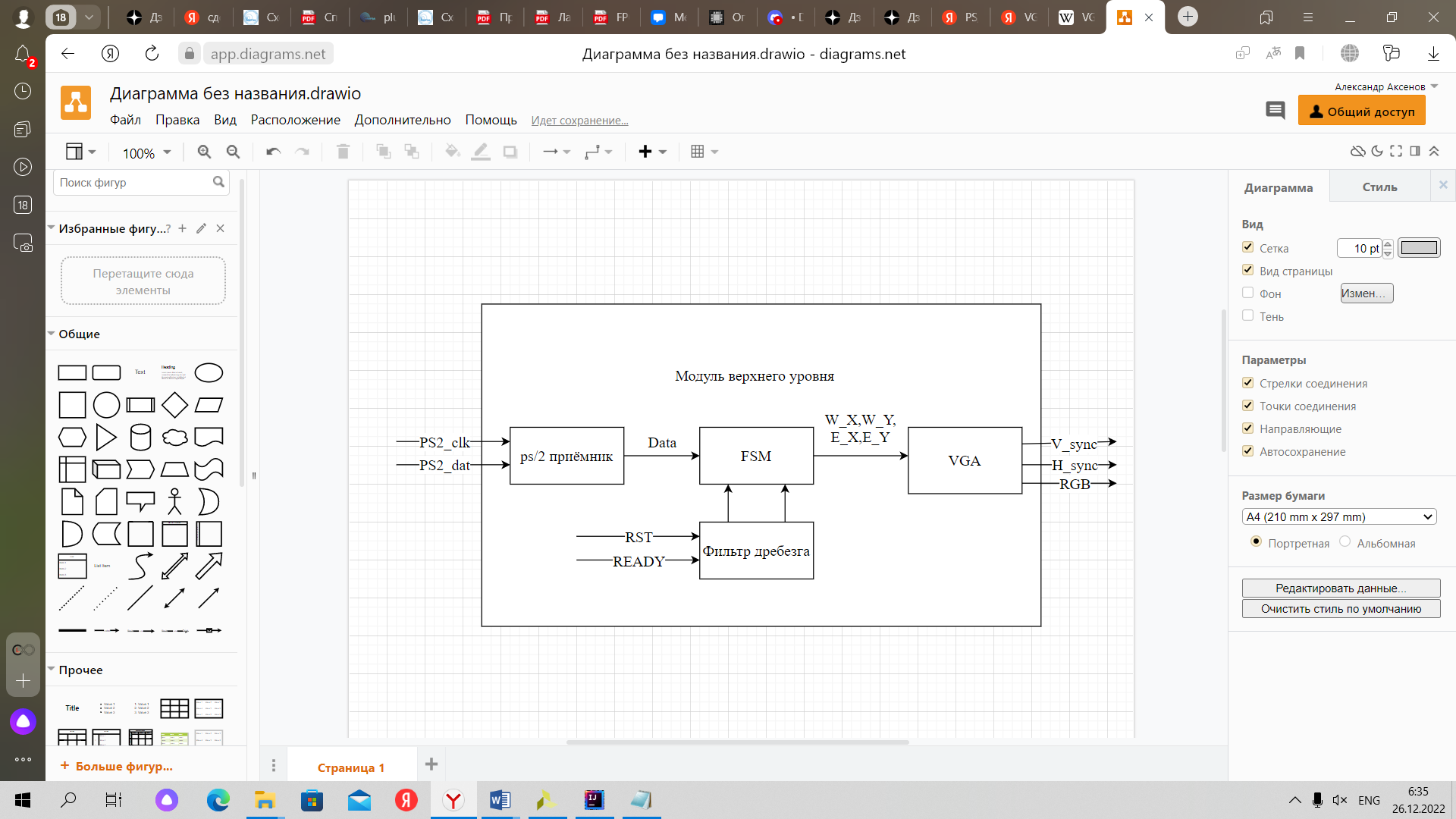


**Рисунок 2.1 – Демонстрация системной модели**

Итоговая системная модель показывает свою жизнеспособность, существуют некоторые технические особенности реализации, например, обновления счётчика яиц, только после нажатия какой-либо клавиши, или высвечивания индикации проигрыша, только после нажатия клавиши. Но основной поток команд остаётся верным, волк исправно передвигается в четырёх направления, с учётом ограничений рабочей области, ловит яйца только с помощью корзины. Листинг системной модели представлен в приложении А.1.

**2.2 Создание RTL-модели**

Итоговое устройство можно разделить на три главных модуля: приёмник ps/2, конечный автомат [8] с логикой игры , VGA обработчик. Всё это будет скомпоновано в модуль верхнего уровня. Дополнительно к этому стоит добавить фильтр дребезга контактов. Итоговое графическое представление устройств показано на рисунке 2.2.



**Рисунок 2.2 – Графическое представление устройства**

После предварительного проектирования устройства в общем плане перейдём к рассмотрению его отдельных составляющих.

**2.2.1 Модуль ps/2**

Основной код модуля ps/2 был взят из учебника по проектированию плис [9. ст 237] и доработан, согласно варианту курсовой работы. В основе всего модуля лежит конечный автомат, отсчитывающий нужное количество бит данных, использую индикацию по заднему фронту (fall\_edge), также во избежание мета стабильности был добавлен фильтр на p2c. Листинг модуля ps/2 представлен в приложении А.2.

**2.2.2 Модуль Конечного автомата**

Автомат содержит в себе четыре состояния: нулевое состояние -ожидание кнопки Ready, для начала игры, первое состояния – проверка нажатия клавиши и изменения соответствующей координаты волка, второе состояние - проверка на поимку яйца и проверка на проигрыш, четвертое состояние - состояние проигрыша блокировка управления. Также есть отдельный блок always, отвечающий за передвижение яйца по координате y. Листинг модуля конечного автомата представлен в приложении А.3.

**2.2.3 Модуль VGA**

Основной код модуля VGA был взят из учебника по проектированию плис [9. ст 335] и доработан, согласно варианту курсовой работы. Он состоит из двух модулей vga\_sync и vga\_top. Модуль vga\_sync отвечает за горизонтальную и вертикальную синхронизацию, в нём устанавливаются параметры разращения экрана (640х480) , входящая частота преобразуется в 25.175 мгц с помощью pll. Основная задача модуля вести верный подсчёт пикселей и выдавать вертикальную и горизонтальную синхронизацию между front\_porch и back\_porch. В свою очередь модуль vga\_top обеспечивает вывод изображения с помощью псевдоматрицы (схожее с вариантом задачи двухмерном массивом в системной модели) и обеспечивает вывод цветов rgb.

Листинги модулей vga\_sync и vga\_top представлен в приложении А.4 и A.5.

**2.2.4 Модуль верхнего уровня и файл проектных ограничений**

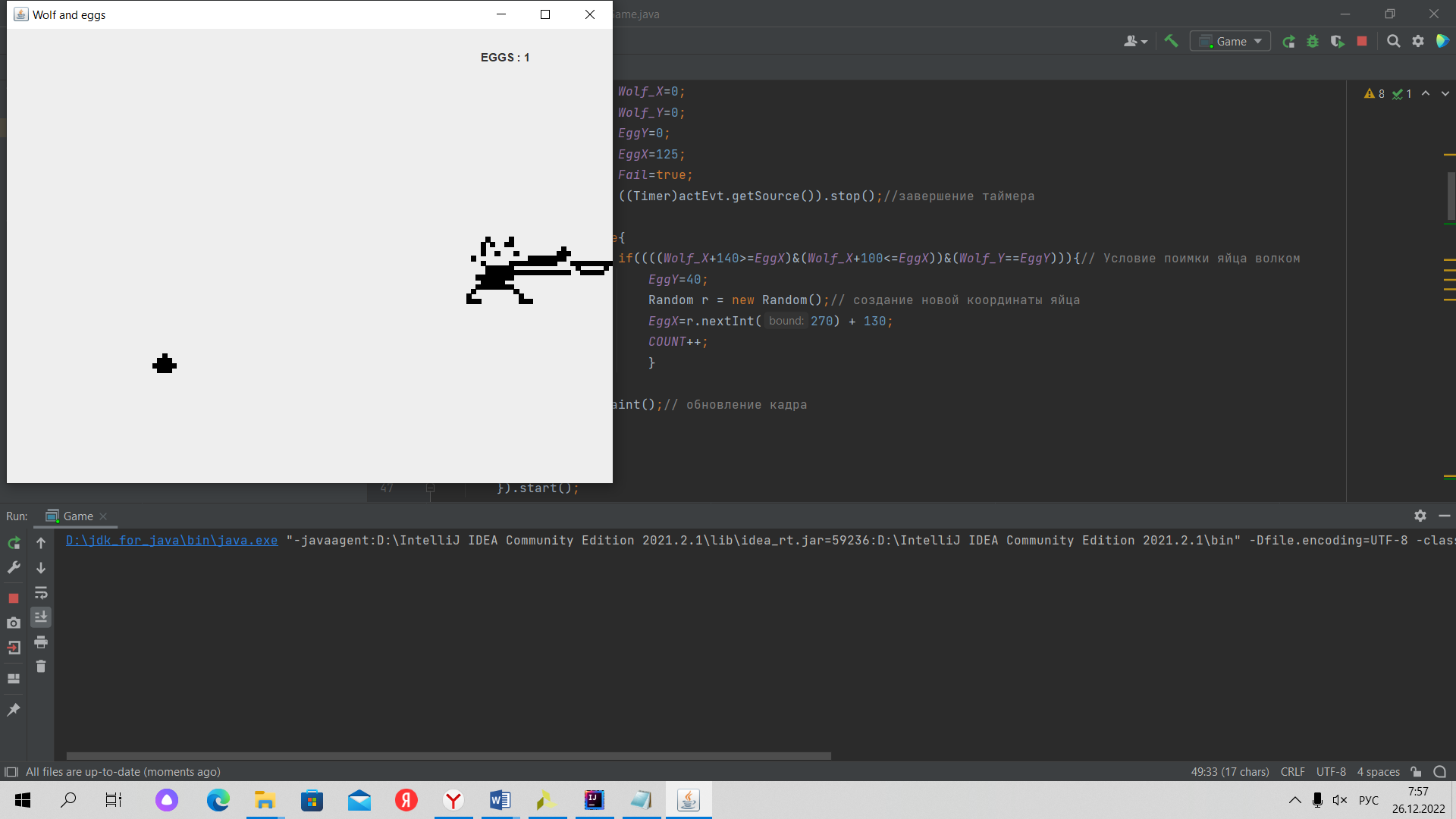
В модуле верхнего уровня объединены все модули, перечисленные выше, также добавлен модуль фильтра дребезга контактов, взятый из методического указания к первой лабораторной работе [10]. Также синхросигнал проведён через блок pll во избежание мета стабильности.

Листинги модулей фильтра и модуля верхнего уровня представлен в приложении А.6 и A.7. Листинг файла проектных ограничений представлен в приложении Д.1.

**ТЕСТИРОВАНИЕ**

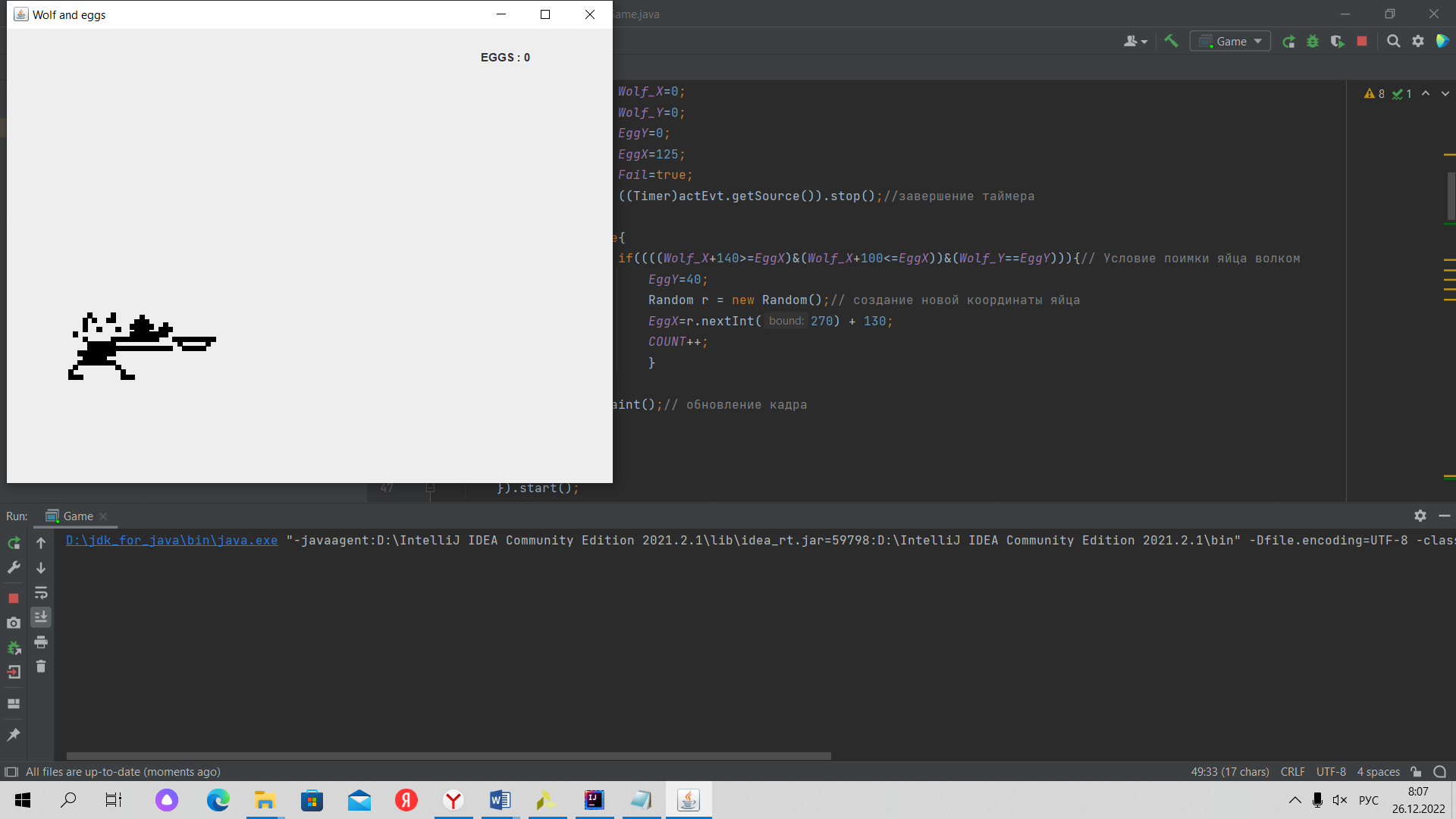
Проведём комплексной тестирование системной модели. Для начала рассмотрим возможность волка уйти за пределы границ окна. На рисунке 3.1

Представлена попытка выйти за границы экрана. Тест показал, что в этой части программы неисправностей нет.



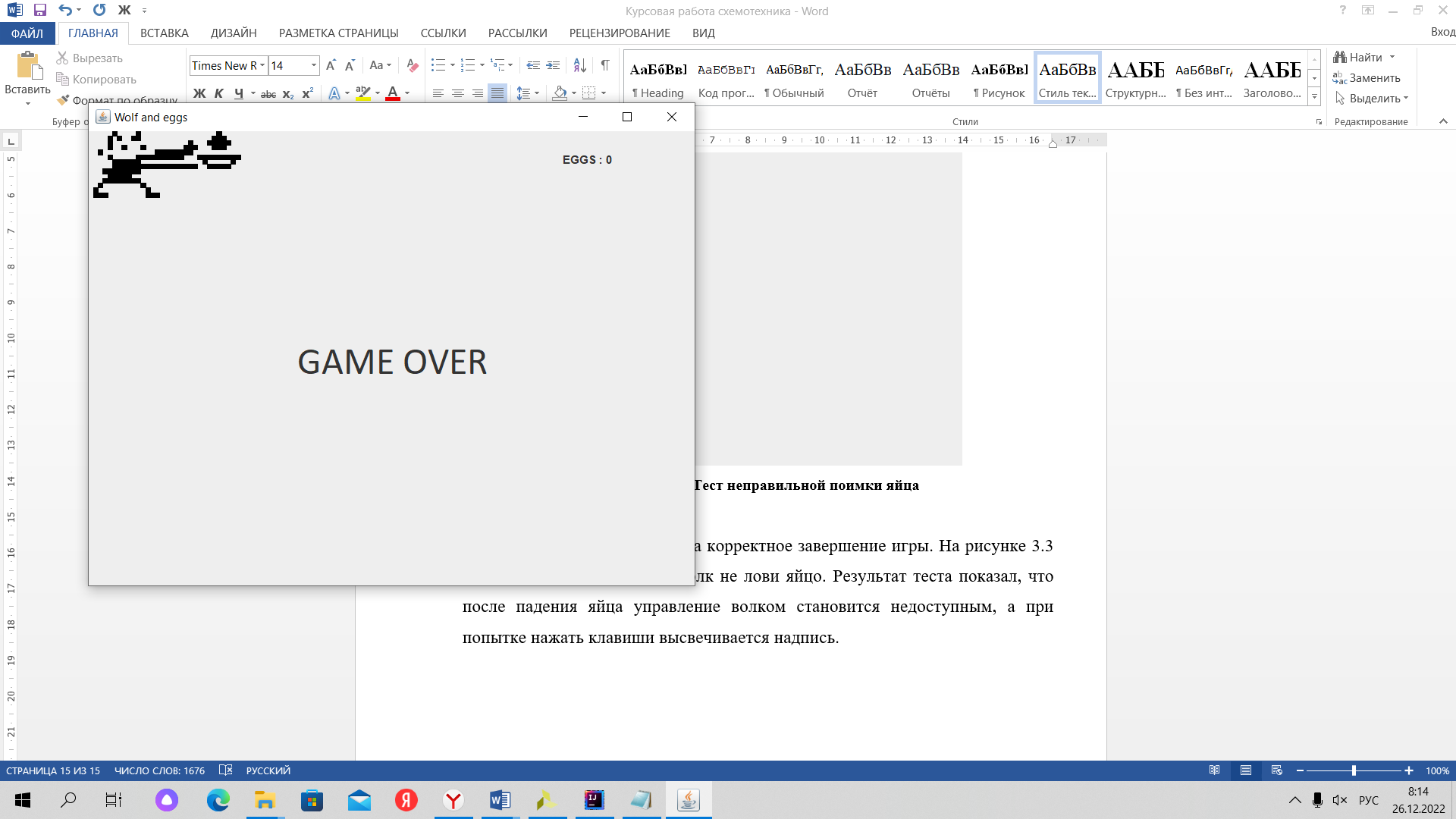
**Рисунок 3.1 – Тест выхода за границы экрана**

Следующий тест, который стоит проверить это возможность поймать яйцо не с помощью корзины. На рисунке 3.2 Представлена попытка поймать яйцо не корзиной. Результат теста показал, что яйцо просто проходит сквозь волка.



**Рисунок 3.2 – Тест неправильной поимки яйца**

Последний тест – тест на корректное завершение игры. На рисунке 3.3 Представлена попытка, где волк не лови яйцо. Результат теста показал, что после падения яйца управление волком становится недоступным, а при попытке нажать клавиши высвечивается надпись.



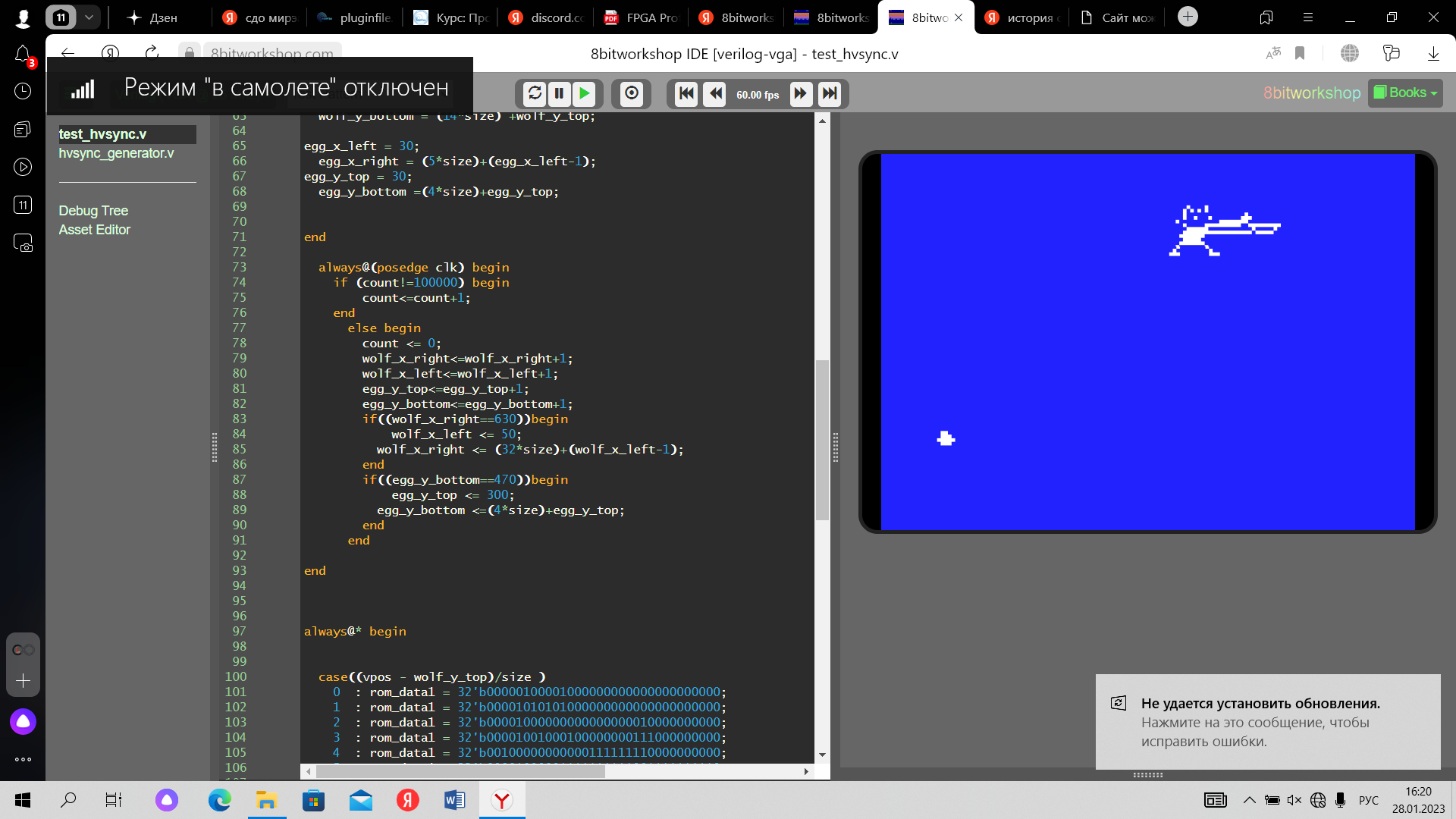
**Рисунок 3.3 – Тест на окончание игры**

Тестирование rtl-модели подразумевает похожие тесты для конечного автомата и отдельные тесты для каждого из двух модулей VGA b и PS/2.

Временные диаграммы тестов представлены в приложении Б. Листинг тестов представлены в приложении В.

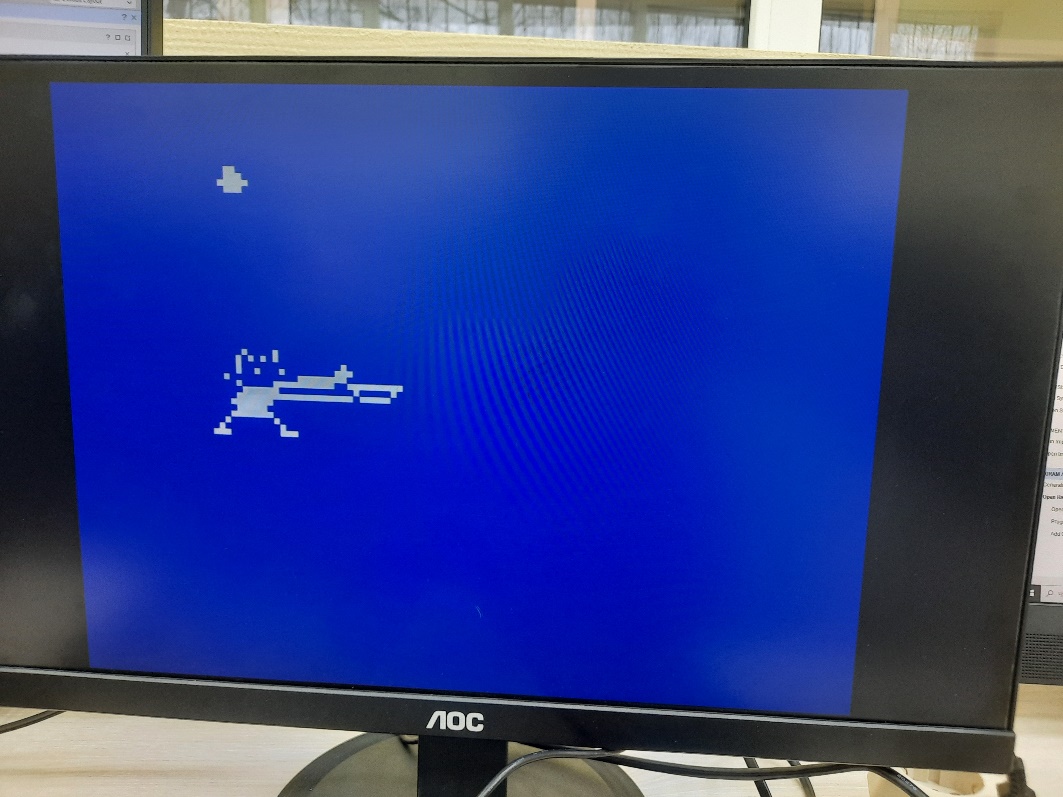
Также было произведено тестирование VGA на платформе 8bitworkshop

(Рисунок 3.4).



**Рисунок 3.4 – Тест VGA платформе 8bitworkshop**

Также было произведено финальное тестирование проекта на отладочной плате, в результате которого были выявлены неполадки с передвижением персонажа по экрану. Модули VGA и PS/2 показали свою корректную работу. Результат тестирования на отладочной плате показан на рисунке 3.5.



**Рисунок 3.5– Тестирование на отладочной плате**

**ЗАКЛЮЧЕНИЕ**

По итогам проделанной работы был разработан рабочий концепт игры «Волк ловит яйца», составлены соответствующие модули на языке описания аппаратуры Verilog, однако не удалось провести полное тестирование полученного продукта. Дальнейшие развития проекта должно включать в себя перенос проекта на отладочную плату и доработку с учётом физических особенностей интегральных схем. По взаимодействию с игрой было создано специальное руководство пользователя, представленное в приложении Д.2.

**СПИСОК ИНФОРМАЦИОННЫХ ИСТОЧНИКОВ**

1.Статья на Википедии про разработку видеоигр — URL: https://ru.wikipedia.org/wiki/Разработка\_компьютерных\_игр

(Дата обращения: 25.12.2022).

2.Статья на Википедии про игру «Ну погоди!» — URL: https://ru.wikipedia.org/wiki/Ну,\_погоди!\_(электронная\_игра)

(Дата обращения: 25.12.2022).

3.Статья на Википедии про порт PS/2— URL: https://ru.wikipedia.org/wiki/PS/2\_(порт)

(Дата обращения: 25.12.2022).

4. Методические указания по ЛР № 2 — URL:

https://online-edu.mirea.ru/pluginfile.php?file=%2F969868%2Fmod\_folder%2Fcontent%2F0%2FСправочная%20информация%20к%202-й%20работе.pdf&forcedownload=1 (Дата обращения: 25.12.2022).

5.Статья на Википедии про интерфейс VGA — URL: https://ru.wikipedia.org/wiki/VGA

(Дата обращения: 25.12.2022).

6.Методические указания по ЛР № 4 — URL: https://online-edu.mirea.ru/pluginfile.php?file=%2F969868%2Fmod\_folder%2Fcontent%2F0%2FСправочная%20информация%20к%204-й%20работе.pdf&forcedownload=1

(Дата обращения: 25.12.2022).

7. Тарасов И.Е. ПЛИС Xilinx. Языки описания аппаратуры VHDL и Verilog, САПР, приемы проектирования. — М.: Горячая линия — Телеком, 2021. — 538 с.: ил.

8. 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).

9. Учебник по проектированию ПЛИС — FPGA Prototyping by Verilog Examples: Xilinx Spartan-3 Version Pong P. Chu published September 2011 528 Pages.

10.Методические указания по ЛР № 1 — URL: https://online-edu.mirea.ru/pluginfile.php?file=%2F969868%2Fmod\_folder%2Fcontent%2F0%2FЛабораторная%20работа%20№%201.pdf&forcedownload=1

**ПРИЛОЖЕНИЯ**

Приложение А1 — Листинг системной модели

Приложение A2 — Листинг модуля ps/2

Приложение A3 — Листинг модуля конечного автомата

Приложение A4 — Листинг модуля vga\_sync

Приложение A5 — Листинг модуля vga\_top

Приложение A6 — Листинг модуля фильтра дребезга контактов

Приложение A7 — Листинг модуля верхнего уровня

Приложение Б1 — Временная диаграмма теста на выход заграницы

Приложение Б2 — Временная диаграмма теста на завершение игры

Приложение Б3 — Временная диаграмма теста PS/2

Приложение Б4 — Временная диаграмма теста VGA

Приложение В1 — Листинг тест-модуля PS/2

Приложение В2 — Листинг тест-модуля VGA

Приложение В3 — Листинг тест-модуля конечного автомата

Приложение Д1 — Листинг файла проектных ограничений

Приложение Д2 — Руководство пользователя

**Приложение А1**

Листинг А1 — Описание системной модели

|  |
| --- |
| module test\_main();  import java.awt.\*;//Граф. библиотеки import java.awt.event.\*; import javax.swing.\*; import java.util.Random;// Библиотека рандомной генерации   public class Game extends JComponent {   private static final long *serialVersionUID* = 1L;  private static final int *HD* = 640;// Горизонт. разрешение  private static final int *VD* = 480;// Вертикальное разрешение  private static final int *TIMER\_DELAY* = 30; // Задержка отрисовки кадров и передвижения  public static int *Wolf\_X* = 10;  public static int *Wolf\_Y* = 400;  public static int *EggX* = 130;  public static int *EggY* = 40;  public static int *egg\_speed* =1;  public static int *COUNT*=0 ;// Количество яйиц  public static boolean *Fail*=false ;// индекатор провала  public Game() {  new Timer(*TIMER\_DELAY*, new ActionListener() {// Создание таймера на отрисовку   @Override  public void actionPerformed(ActionEvent actEvt) {  if (*Wolf\_X* < *HD* && *Wolf\_Y* < *VD*) {// условие выполнение таймера  *EggY*=*EggY*+*egg\_speed*;  if((*EggY*==400)){// условия провала  *Wolf\_X*=0;  *Wolf\_Y*=0;  *EggY*=0;  *EggX*=125;  *Fail*=true;  ((Timer)actEvt.getSource()).stop();//завершение таймера  }  else{  if((((*Wolf\_X*+140>=*EggX*)&(*Wolf\_X*+100<=*EggX*))&(*Wolf\_Y*==*EggY*))){// Условие поимки яйца волком  *EggY*=40;  Random r = new Random();// создание новой координаты яйца  *EggX*=r.nextInt(270) + 130;  *COUNT*++;  }  }  repaint();// обновление кадра  }   }  }).start();  }  private static void createAndShowGui() {  Game mainPanel = new Game();// создание панели для рисования |

Продолжение Листинг А1

|  |
| --- |
| JFrame frame = new JFrame("Wolf and eggs");//Создание главного окна  JLabel count = new JLabel("EGGS : "+*COUNT*); //Создание Индикатора яйиц  JLabel fail = new JLabel("GAME OVER");// Создание индикатора провала  fail.setFont(new Font("Calibri", Font.*PLAIN*, 40));  fail.setBounds(170,150,300,200);  fail.setHorizontalAlignment(JLabel.*CENTER*);  fail.setVerticalAlignment(JLabel.*CENTER*);//дополнительные настройки индикаторов  fail.setVisible(false);  count.setBounds(500,20,150,20);   frame.addKeyListener(new KeyAdapter() {//отслеживание активности клавиатуры  @Override  public void keyPressed(KeyEvent e) {  if ((e.*getKeyText*(e.getKeyCode()).equals("D"))&(*Wolf\_X*<=470)) {  *Wolf\_X* = *Wolf\_X* + 10;   }  if ((e.*getKeyText*(e.getKeyCode()).equals("A"))&(*Wolf\_X*>=10)) {  *Wolf\_X* = *Wolf\_X* - 10;  }   if ((e.*getKeyText*(e.getKeyCode()).equals("W"))&(*Wolf\_Y*>=60)) {  *Wolf\_Y* = *Wolf\_Y* - 10;   }  if ((e.*getKeyText*(e.getKeyCode()).equals("S"))&(*Wolf\_Y*<=400)) {  *Wolf\_Y* = *Wolf\_Y* + 10;  }  count.setText("EGGS : "+*COUNT*);// обновление показателя индикатора яйиц  if (*Fail*==true){//установка провала  fail.setVisible(true);  }  }  });  frame.setDefaultCloseOperation(JFrame.*EXIT\_ON\_CLOSE*);  frame.add(count);//добавление всех компонентов в главное окно  frame.add(fail);  frame.getContentPane().add(mainPanel);  frame.pack();  frame.setVisible(true);// прорисовка } |

Продолжение Листинг А1

|  |
| --- |
| @Override public Dimension getPreferredSize() {// установка размеров главного окна  return new Dimension(*HD*, *VD*); }  public void paintComponent(Graphics g) {//отрисовка фреймов с помощью двоиных масивов  super.paintComponent(g);  int pixel\_size =5;  g.setColor(Color.*BLACK*);  g.drawRect(0, 480, 640, pixel\_size);  g.fillRect(0, 480, 640, pixel\_size);  g.drawRect(640, 0, pixel\_size, 480);  g.fillRect(640, 0, pixel\_size, 480);  int[][] woolf = new int[][]{  { 0,0,0,0, 0,1,0,0, 0,0,1,0, 0,0,0,0 ,0,0,0,0 ,0,0,0,0 ,0,0,0,0 ,0,0,0,0},  { 0,0,0,0, 1,0,1,0, 0,1,1,0, 0,0,0,0 ,0,0,0,0 ,0,0,0,0 ,0,0,0,0 ,0,0,0,0},  { 0,0,0,0, 1,0,0,0, 0,0,0,0, 0,0,0,0 ,0,0,0,0 ,0,1,0,0 ,0,0,0,0 ,0,0,0,0},  { 0,0,0,0, 1,0,0,1, 0,0,0,1, 0,0,0,0 ,0,0,0,0 ,1,1,1,0 ,0,0,0,0 ,0,0,0,0},  { 0,0,1,0, 0,0,0,0, 0,0,0,0, 0,0,1,1 ,1,1,1,1 ,1,1,0,0 ,0,0,0,0 ,0,0,0,0},  { 0,0,0,0, 1,0,0,0, 0,0,1,1, 1,1,1,1 ,1,1,1,1 ,0,0,0,1 ,1,1,1,1 ,1,1,1,1},  { 0,0,0,0, 0,1,1,1, 1,1,1,0, 0,0,0,0 ,0,0,0,0 ,0,0,0,0 ,1,0,0,0 ,0,0,1,0},  { 0,0,0,0, 0,1,1,1, 1,1,1,1, 1,1,1,1 ,1,1,1,1 ,1,1,1,0 ,0,1,1,1 ,1,1,0,0},  { 0,0,0,1, 1,1,1,1, 1,1,1,0, 0,0,0,0 ,0,0,0,0 ,0,0,0,0 ,0,0,0,0 ,0,0,0,0},  { 0,0,0,0, 1,1,1,1, 1,0,0,0, 0,0,0,0 ,0,0,0,0 ,0,0,0,0 ,0,0,0,0 ,0,0,0,0},  { 0,0,0,1, 1,1,1,1, 1,1,1,0, 0,0,0,0 ,0,0,0,0 ,0,0,0,0 ,0,0,0,0 ,0,0,0,0},  { 0,0,1,0, 0,0,0,0, 0,0,0,1, 0,0,0,0 ,0,0,0,0 ,0,0,0,0 ,0,0,0,0 ,0,0,0,0},  { 0,1,0,0, 0,0,0,0, 0,0,0,0, 1,0,0,0 ,0,0,0,0 ,0,0,0,0 ,0,0,0,0 ,0,0,0,0},  { 0,1,1,1, 0,0,0,0, 0,0,0,0, 1,1,1,0 ,0,0,0,0 ,0,0,0,0 ,0,0,0,0 ,0,0,0,0},  };  int [][] egg = new int[][]{  {0,0,1,0,0,},  {0,1,1,1,0,},  {1,1,1,1,1,},  {0,1,1,1,0,},  }; |

Продолжение Листинг А1

|  |
| --- |
| for (int i =0;i<14;i++){  for (int j =0;j<32;j++){  if (woolf[i][j]==1){  g.setColor(Color.*BLACK*);  g.drawRect(*Wolf\_X*+(j\*pixel\_size), *Wolf\_Y*+(i\*pixel\_size), pixel\_size, pixel\_size);  g.fillRect(*Wolf\_X*+(j\*pixel\_size), *Wolf\_Y*+(i\*pixel\_size), pixel\_size, pixel\_size);  }  }  }  for (int i =0;i<4;i++){  for (int j =0;j<5;j++){  if (egg[i][j]==1){  g.setColor(Color.*BLACK*);  g.drawRect(*EggX*+(j\*pixel\_size), *EggY*+(i\*pixel\_size), pixel\_size, pixel\_size);  g.fillRect(*EggX*+(j\*pixel\_size), *EggY*+(i\*pixel\_size), pixel\_size, pixel\_size);  }  }  }   }   public static void main(String[] args) {// запуск приложения  SwingUtilities.*invokeLater*(new Runnable() {  public void run() {  *createAndShowGui*();  }  });  }  } |

**Приложение А2**

*Листинг А2* — *описание модуля ps/2*

|  |
| --- |
| module ps\_2\_rx  (  input wire clk, reset,  input wire ps2d, ps2c, rx\_en,  output reg rx\_done\_tick,  output wire [7:0] dout  );  // symbolic state declaration  localparam [1:0]  idle = 2 'b00,  dps = 2'b01,  load = 2'b10;  // signul declaration  reg [1 :0] state\_reg , state\_next ;  reg [7:0] filter\_reg;  wire [7:0] filter\_next;  reg f\_ps2c\_reg ;  wire f\_ps2c\_next ;  reg [3:0] n\_reg , n\_next ;  reg [10:0] b\_reg, b\_next;  wire fall\_edge ;  initial  begin  filter\_reg <= 0;  f\_ps2c\_reg <= 0;  state\_reg <= idle;  n\_reg <= 0;  b\_reg <= 0;  end  // filter and falling-edge tick generation for ps2c ...................................................  always @ ( posedge clk , posedge reset )  if (reset)  begin  filter\_reg <= 0;  f\_ps2c\_reg <= 0;  end  else  begin  filter\_reg <= filter\_next ;  f\_ps2c\_reg <= f\_ps2c\_next ;  end  assign filter\_next = {ps2c, filter\_reg [7: 1]};  assign f\_ps2c\_next =ps2c;// (filter\_reg==8'b11111111) ? 1'b1 :  //(filter\_reg==8'b00000000) ? 1'b0 :  //f\_ps2c\_reg;  assign fall\_edge = f\_ps2c\_reg & ~f\_ps2c\_next ; |

*Продолжение Листинг А2*

|  |
| --- |
| // FSMD ...................................................  // FSMD state & data registers  always@(posedge clk , posedge reset )  if (reset)  begin  state\_reg <= idle;  n\_reg <= 0;  b\_reg <= 0;  end  else  begin  state\_reg <= state\_next ;  n\_reg <= n\_next;  b\_reg <= b\_next;  end  // FSMD next-state logic  always @\*  begin  state\_next = state\_reg;  rx\_done\_tick = 1'b0;  n\_next = n\_reg;  b\_next = b\_reg;  case (state\_reg)  idle :  if (fall\_edge & rx\_en)  begin  // shift in start bit  b\_next = {ps2d, b\_reg [10: 1] };  n\_next = 4'b1001;  state\_next = dps;  end  dps: // 8 data + 1 parity + 1 stop  if (fall\_edge)  begin  b\_next = {ps2d, b\_reg [10 : 1] } ;  if (n\_reg==0)  state\_next = load;  else  n\_next = n\_reg - 1;  end  load: // I extra clock to complete the last shift  begin  state\_next = idle;  rx\_done\_tick = 1'b1;  end  endcase  end  // output  assign dout = b\_reg[8:1]; // data bits  endmodule |

**Приложение А3**

*Листинг А3* — *описание модуля конечного автомата*

|  |
| --- |
| module game(  input rx\_done\_tick,  input clk,rst,ready,  input [7:0] key\_kode,  output [9:0] wolf\_x,wolf\_y,egg\_x,egg\_y  //output [6:0] egg\_count,  );  reg [9:0] Wolf\_x,Wolf\_y,Egg\_x,Egg\_y;  reg [6:0] Egg\_count;  reg [3:0] state;  reg Fail;  reg Ready;  initial begin  state = 0;  Fail = 0;  Wolf\_x = 10;  Wolf\_y = 200;  Egg\_x = 130;  Egg\_y =40;  Egg\_count=0;  end    always@(posedge clk)  begin  if(rst==1)begin  state = 0;  Fail = 0;  Wolf\_x = 10;  Wolf\_y = 400;  Egg\_x = 130;  Egg\_y =40;  Egg\_count=0;  end  case(state)  4'h0 : if( ready==1) state=4'h1;  4'h1 : if((rx\_done\_tick==1))  begin  case(key\_kode)  8'h41 : if (Wolf\_x >= 10) Wolf\_x =Wolf\_x-10;  8'h44 : if (Wolf\_x <= 470) Wolf\_x =Wolf\_x+10;  8'h57 : if (Wolf\_y >= 60) Wolf\_y =Wolf\_y-10;  8'h53 : if (Wolf\_y <= 400) Wolf\_y =Wolf\_y+10;  endcase  state =4'h2;  end    4'h2: if((((Wolf\_x+140>=Egg\_x)&(Wolf\_x+100<=Egg\_x))&(Wolf\_y==Egg\_y)))begin  Egg\_y=40;  if(Wolf\_x>=300)  Egg\_x = Wolf\_x -80;  else Egg\_x = Wolf\_x+80;  Egg\_count=Egg\_count+1;  state =4'h1;  end  else begin  if(Egg\_y >=400) begin  state = 4'h3;  end |

*Продолжение Листинг А3*

|  |
| --- |
| else state = 4'h1;  end  4'h3: begin  Fail = 1;  Wolf\_x = 10;  Wolf\_y = 400;  Egg\_x = 130;  Egg\_y =40;  state = 4'h3;  end  endcase  end    always# 3000000000 begin //3000000000  if (Fail==0)  Egg\_y= Egg\_y+1;  end    assign wolf\_x =Wolf\_x;  assign wolf\_y =Wolf\_y;  assign egg\_x =Egg\_x;  assign egg\_y =Egg\_y;  //assign egg\_count = Egg\_count;  endmodule |

**Приложение А4**

*Листинг А4* — *описание модуля vga\_sync*

|  |
| --- |
| module vga\_sync  (  input wire clk,reset,  output wire hsync, vsync, video\_on, p\_tick,  output wire [9:0] pixel\_x, pixel\_y  );    localparam HD = 640; //horizontal display area  localparam HF = 48; //h.front (left) border  localparam HB = 16; // h.back (right) boarder  localparam HR = 96; // h.retrace  localparam VD = 480;//vertical display area  localparam VF = 33; // v.fraont (top) boarder  localparam VB = 10; // v.back (bottom) boarder  localparam VR= 2; //v.retrace        //sync counters  reg [9:0] h\_count\_reg, h\_count\_next;  reg [9:0] v\_count\_reg, v\_count\_next;  // output buffer  reg v\_sync\_reg, h\_sync\_reg;  wire v\_sync\_next, h\_sync\_next;  // status signal  wire h\_end , v\_end , pixel\_tick;  initial begin  v\_count\_reg <= 0;  h\_count\_reg <=0;  v\_sync\_reg <= 1'b0;  h\_sync\_reg <= 1'b0;  end |

*Продолжение Листинг А4*

|  |
| --- |
| clk\_wiz\_0 inst(  .clk\_in1(clk),  .clk\_out1(pixel\_tick)  );  //body  //registers  always@(posedge clk)  begin  if(reset)  begin  v\_count\_reg <= 0;  h\_count\_reg <=0;  v\_sync\_reg <= 1'b0;  h\_sync\_reg <= 1'b0;  end  else  begin  v\_count\_reg <= v\_count\_next;  h\_count\_reg <= h\_count\_next;  v\_sync\_reg <= v\_sync\_next;  h\_sync\_reg <= h\_sync\_next;  end  end    //status signals  //end of horizontal counter (799)  assign h\_end =(h\_count\_reg==(HD+HF+HB+HR-1));  // end of vertical counter (524)  assign v\_end =(v\_count\_reg==(VD+VF+VB+VR-1));    //next-state logic of mod~800 horizontal sync counter  always @\*  if(pixel\_tick) // 25 Mhz pulse  if(h\_end)  h\_count\_next = 0;  else  h\_count\_next = h\_count\_reg + 1;  else  h\_count\_next= h\_count\_reg;    //next-state logic of mod-525 vertical sync counter    always @\*  if(pixel\_tick & h\_end)  if(v\_end)  v\_count\_next = 0;  else  v\_count\_next = v\_count\_reg+1;  else  v\_count\_next = v\_count\_reg;    // horizontal and vertical sync, buffered to avoid glitch  // h\_sync\_next asserted between 656 and 751  assign h\_sync\_next = (h\_count\_reg >= (HD+HB) && h\_count\_reg <=(HD+HB+HR-1));    // v\_sync\_next asserted between 490 and 491  assign v\_sync\_next =(v\_count\_reg >= (VD+VB) && v\_count\_reg <= (VD+VB+VR-1)); |

*Продолжение Листинг А4*

|  |
| --- |
| //video on/off  assign video\_on = (h\_count\_reg < HD) && (v\_count\_reg < VD);    //outputs  assign hsync = h\_sync\_reg;  assign vsync = v\_sync\_reg;  assign pixel\_x = h\_count\_reg;  assign pixel\_y = v\_count\_reg;  assign p\_tick = pixel\_tick;      endmodule |

**Приложение A5**

*Листинг А5* — *описание модуля top\_vga*

|  |
| --- |
| module vga\_top(  input wire clk, reset,  input wire [2:0] sw,  input wire [9:0] wolf\_x,wolf\_y,egg\_x,egg\_y,  output wire hsync, vsync,  output wire [2:0] rgb  );  //signal declaration  reg [2:0] rgb\_reg;  wire video\_on;  wire [9:0] pix\_x, pix\_y;    //instantiate vga sync circuit  vga\_sync vsync\_unit (  .clk(clk),  .reset(reset),  .hsync(hsync),  .vsync(vsync),  .video\_on(video\_on),  .p\_tick(),  .pixel\_x(pix\_x),  .pixel\_y(pix\_y)  );    reg [9:0] Wolf\_x,Wolf\_y,Egg\_x,Egg\_y;      reg [9:0] wolf\_x\_left;  reg [9:0] wolf\_x\_right;  reg [9:0] wolf\_y\_top;  reg [9:0] wolf\_y\_bottom;    reg [9:0] egg\_x\_left;  reg [9:0] egg\_x\_right;  reg [9:0] egg\_y\_top;  reg [9:0] egg\_y\_bottom; |

*Продолжение Листинг А5*

|  |
| --- |
| wire wolf\_sync\_on;  wire egg\_sync\_on;  wire [4:0] rom\_col1;  wire [2:0] rom\_col2;  wire [3:0] rom\_addr1 ;  wire [1:0] rom\_addr2 ;  reg [0:31] rom\_data1;  reg [0:4] rom\_data2;  wire [2:0] wolf\_rgb;  wire rom\_bit1;  wire rom\_bit2;  //sprites  always@\* begin  Wolf\_x =wolf\_x;  Wolf\_y =wolf\_y;  Egg\_x =egg\_x;  Egg\_y = egg\_y;  wolf\_x\_left = 50+Wolf\_x;  wolf\_x\_right = 82+Wolf\_x;  wolf\_y\_top = 40+Wolf\_y;  wolf\_y\_bottom = 54+Wolf\_y;  egg\_x\_left = 50+Egg\_x;  egg\_x\_right = 55+Egg\_x;  egg\_y\_top = 40+Egg\_y;  egg\_y\_bottom =44+Egg\_y;  case(rom\_addr1)  4'h0 : rom\_data1 = 32'b00000100001000000000000000000000;  4'h1 : rom\_data1 = 32'b00001010101000000000000000000000;  4'h2 : rom\_data1 = 32'b00001000000000000000010000000000;  4'h3 : rom\_data1 = 32'b00001001000100000000111000000000;  4'h4 : rom\_data1 = 32'b00100000000000111111110000000000;  4'h5 : rom\_data1 = 32'b00001000001111111111000111111111;  4'h6 : rom\_data1 = 32'b00000111111000000000000010000010;  4'h7 : rom\_data1 = 32'b00000001111111111111111001111100;  4'h8 : rom\_data1 = 32'b00011111110000000000000000000000;  4'h9 : rom\_data1 = 32'b00001111100000000000000000000000;  4'ha : rom\_data1 = 32'b00011111110000000000000000000000;  4'hb : rom\_data1 = 32'b00100000001000000000000000000000;  4'hc : rom\_data1 = 32'b01000000000100000000000000000000;  4'hd : rom\_data1 = 32'b11100000000111000000000000000000;  endcase  case(rom\_addr2)  2'b00 : rom\_data2 = 5'b00100;  2'b01 : rom\_data2 = 5'b01110;  2'b10 : rom\_data2 = 5'b11111;  2'b11 : rom\_data2 = 5'b01110; |

*Продолжение Листинг А5*

|  |
| --- |
| endcase  end  assign wolf\_sync\_on =(wolf\_x\_left<=pix\_x) &&(pix\_x <=wolf\_x\_right) &&(wolf\_y\_top <=pix\_y) && (pix\_y <=wolf\_y\_bottom) ;  assign egg\_sync\_on =(egg\_x\_left<=pix\_x) &&(pix\_x <=egg\_x\_right) &&(egg\_y\_top <=pix\_y) && (pix\_y <=egg\_y\_bottom) ;  assign rom\_addr1 = pix\_y - wolf\_y\_top ;  assign rom\_addr2 = pix\_y - egg\_y\_top ;  assign rom\_col1 = pix\_x - wolf\_x\_left;  assign rom\_col2 = pix\_x - egg\_x\_left;  assign rom\_bit1 = rom\_data1[rom\_col1];  assign rom\_bit2 = rom\_data2[rom\_col2];  assign wolf\_rgb = 3'b111;      always @\*  if((wolf\_sync\_on & rom\_bit1)|(egg\_sync\_on & rom\_bit2))  rgb\_reg <= wolf\_rgb;  else  rgb\_reg <=sw;  //rgb buffer  always @(posedge clk)  if(reset)  rgb\_reg<=0;    // output  assign rgb = (video\_on) ? rgb\_reg : 3'b0;    endmodule |

**Приложение A6**

*Листинг А6* — *описание модуля фильтра дребезга контактов*

|  |
| --- |
| module FILTER #(size = 3) (  input CLK, CLOCK\_ENABLE, IN\_SIGNAL,  output wire OUT\_SIGNAL\_wire, OUT\_SIGNAL\_ENABLE\_wire  );  reg [1:0] IN\_SIGNAL\_SYNC;  reg [size-1:0] counter;  reg OUT\_SIGNAL, OUT\_SIGNAL\_ENABLE;  initial  begin  IN\_SIGNAL\_SYNC = 0; counter = 0;  OUT\_SIGNAL = 0; OUT\_SIGNAL\_ENABLE = 0;  end  always @(posedge CLK)  begin  IN\_SIGNAL\_SYNC <= {IN\_SIGNAL\_SYNC[0], IN\_SIGNAL};  counter <= (IN\_SIGNAL\_SYNC[1] ~^ OUT\_SIGNAL) ?  {size{1'd0}} : (CLOCK\_ENABLE ? counter + 1 : counter);  if (&(counter) & CLOCK\_ENABLE)  OUT\_SIGNAL <= IN\_SIGNAL\_SYNC[1];  OUT\_SIGNAL\_ENABLE <= &(counter) & CLOCK\_ENABLE & IN\_SIGNAL\_SYNC[1];  end  assign OUT\_SIGNAL\_ENABLE\_wire =OUT\_SIGNAL\_ENABLE;  assign OUT\_SIGNAL\_wire = OUT\_SIGNAL;  endmodule |

**Приложение A7**

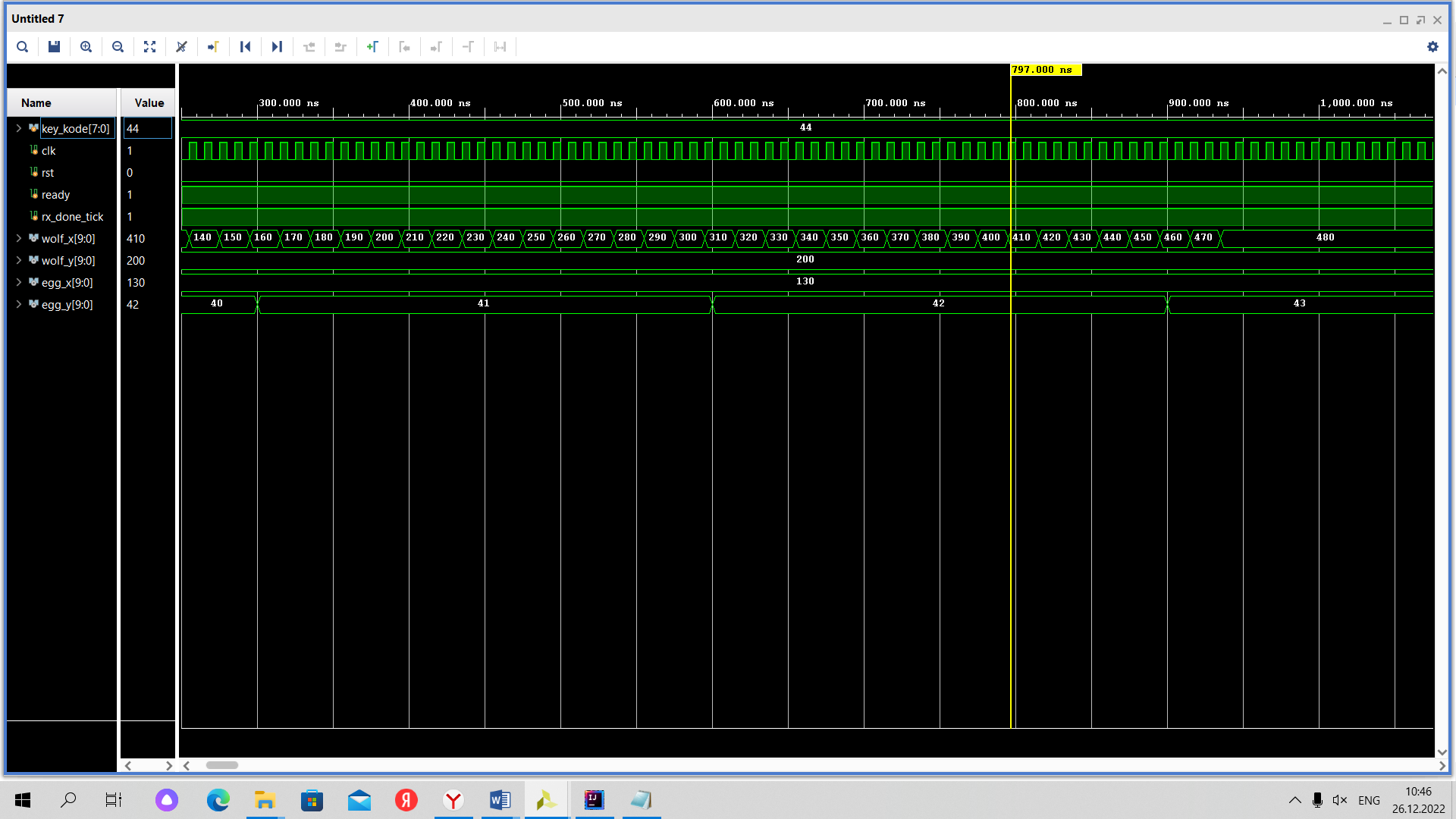
*Листинг А7*— *описание модуля верхнего уровня*

|  |
| --- |
| module Main(  input wire clk, reset, ready,  input wire ps2d, ps2c,  output wire h\_sync, v\_sync,  output wire VGA\_R\_3 ,  output wire VGA\_G\_3 ,  output wire VGA\_B\_3  );    clk\_wiz\_1 inst(  .clk\_in1(clk),  .clk\_out1(clk\_out)  );    reg PS2\_clk, PS2\_dat;  wire [7:0] key\_kode;  wire clk\_out;  reg rst;  wire RESET, READY;  reg CLOCK\_ENABLE = 0;  reg [9:0] Wolf\_x,Wolf\_y,Egg\_x,Egg\_y;  wire rx\_done\_tick;  always @(posedge clk)  CLOCK\_ENABLE <= ~CLOCK\_ENABLE;  FILTER m0 (  .CLK(clk\_out),  .CLOCK\_ENABLE(CLOCK\_ENABLE),  .IN\_SIGNAL(reset),  .OUT\_SIGNAL\_ENABLE\_wire(RESET)  );    FILTER m1 (  .CLK(clk\_out),  .CLOCK\_ENABLE(CLOCK\_ENABLE),  .IN\_SIGNAL(ready),  .OUT\_SIGNAL\_ENABLE\_wire(READY)  );    ps\_2\_rx uut1 (  .clk(clk\_out),  .rx\_en(1'b1),  .rx\_done\_tick(rx\_done\_tick),  .ps2c(PS2\_clk),  .ps2d(PS2\_dat),  .dout(key\_kode),  .reset(RESET)  );  game uut2 (  .clk(clk\_out),  .rst(RESET),  .ready(READY),  .rx\_done\_tick(rx\_done\_tick),  .key\_kode(key\_kode),  .wolf\_x(wolf\_x),  .wolf\_y(wolf\_y),  .egg\_x (egg\_x),  .egg\_y (egg\_y)  ); |

*Продолжение Листинг А7*

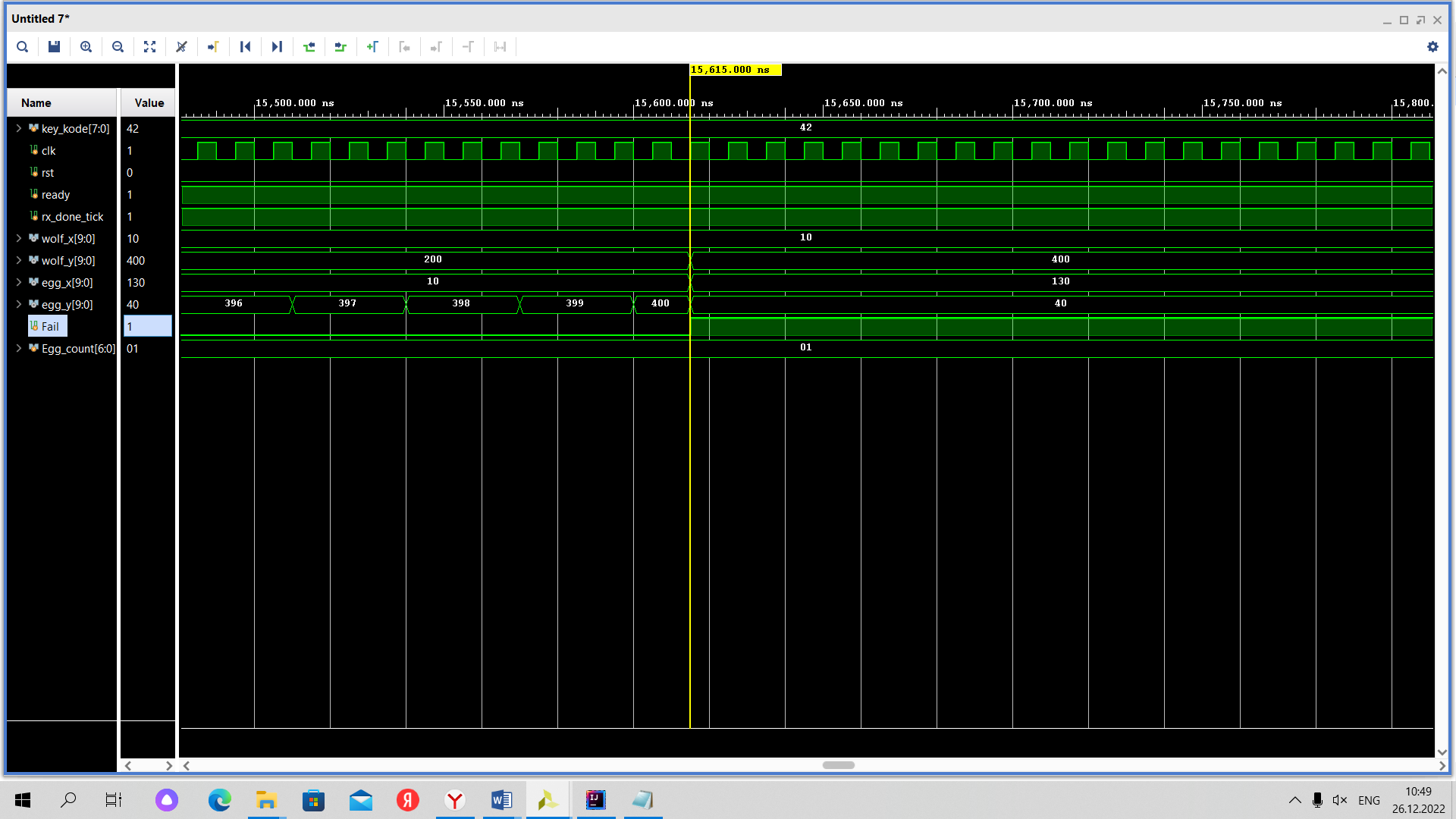
|  |
| --- |
| vga\_top uut3(  .clk(clk\_out),  .reset(RESET),  .sw(3'b111),  .wolf\_x(wolf\_x),  .wolf\_y(wolf\_y),  .egg\_x (egg\_x),  .egg\_y (egg\_y),  .hsync(h\_sync),  .vsync(v\_sync),  .rgb({VGA\_R\_3,VGA\_G\_3,VGA\_B\_3})  );    endmodule |

**Приложение Б1**



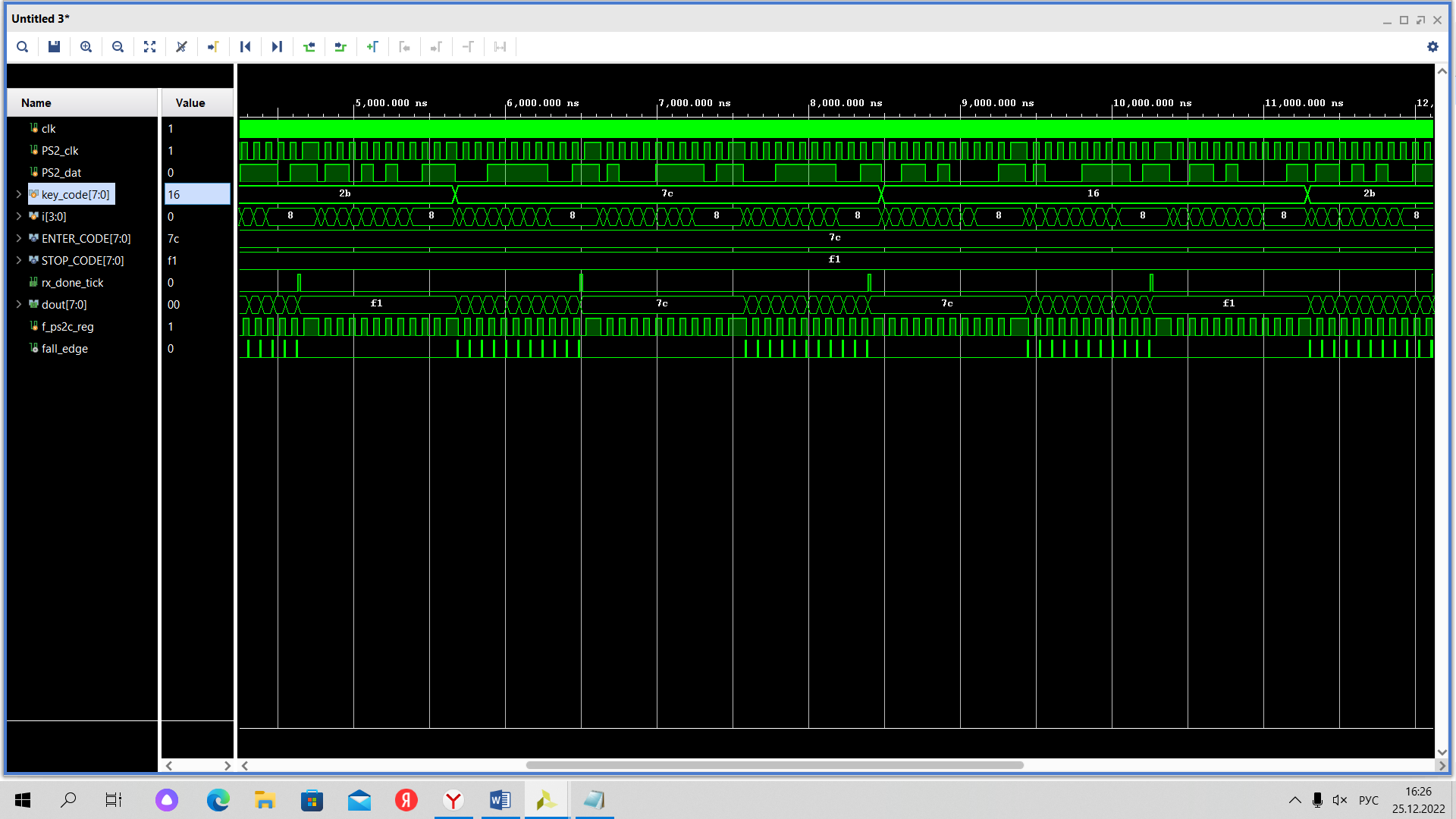
**Рисунок Б1 – Временная диаграмма теста на выход за гарницы**

**Приложение Б2**



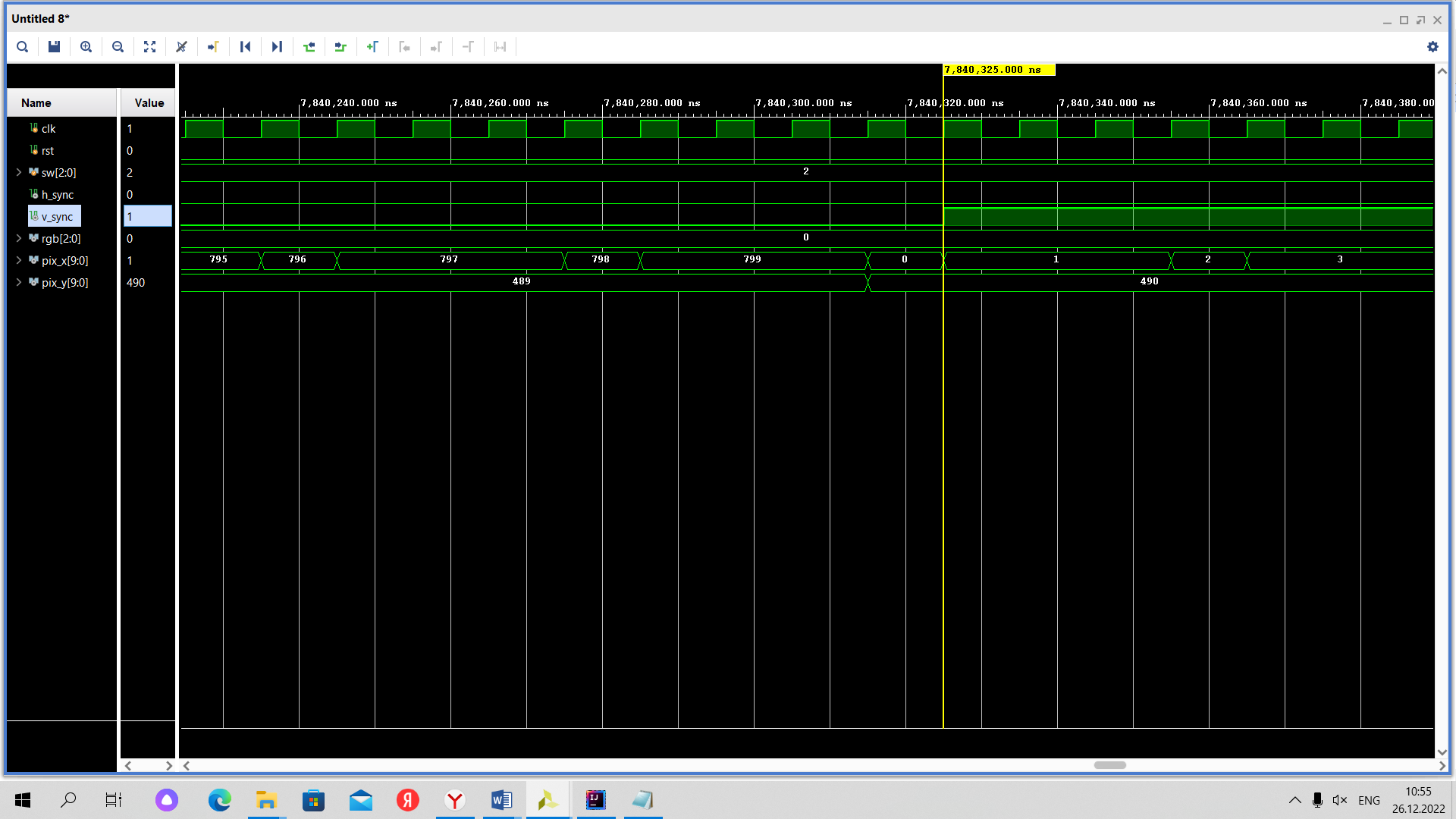
**Рисунок Б2 – Временная диаграмма теста на завершения игры**

**Приложение Б3**



**Рисунок Б3 – Временная диаграмма теста ps/2**

**Приложение Б4**



**Рисунок Б4 – Временная диаграмма теста vga**

**Приложение В1**

*Листинг В1 — тест-модуля PS/2*

|  |
| --- |
| module Test();  parameter ENTER\_CODE = 8'h7C;  parameter STOP\_CODE = 8'hF1;  parameter clk\_period = 10;  parameter PS2\_clk\_period = 40;  parameter code\_space\_period = 60;    reg clk, PS2\_clk, PS2\_dat;  reg [7:0] key\_code;  reg [3:0] i;  wire [7:0] AN;  wire [6:0] SEG;  wire [7:0] Dout;  wire clk\_out;  reg rst;  ps\_2\_rx uut (  .clk(clk),  .rx\_en(1'b1),  .rx\_done\_tick(clk\_out),  .ps2c(PS2\_clk),  .ps2d(PS2\_dat),  .dout(Dout),  .reset(rst)  ); |

*Продолжение Листинг В1*

|  |
| --- |
| always #(clk\_period) clk <= ~clk;  initial  begin  rst =0;  #5  rst =0;  PS2\_clk <= 1;  PS2\_dat <= 1;  key\_code <= 0;    clk <= 0;  @(posedge clk)  @(posedge clk)    #(2\*clk\_period) key\_code = HEX\_CD(1);  PS2\_press\_and\_release\_key(key\_code);  #(2\*clk\_period) key\_code = HEX\_CD(4'hF);  PS2\_press\_and\_release\_key(key\_code);  #(2\*clk\_period) key\_code = ENTER\_CODE;  PS2\_press\_and\_release\_key(key\_code);    #(2\*clk\_period) key\_code = HEX\_CD(1);  PS2\_press\_and\_release\_key(key\_code);  #(2\*clk\_period) key\_code = HEX\_CD(4'hF);  PS2\_press\_and\_release\_key(key\_code);  #(2\*clk\_period) key\_code = ENTER\_CODE;  PS2\_press\_and\_release\_key(key\_code);    #(2\*clk\_period)  $finish;  end  // Нажатие и отжатие клавиши  task automatic PS2\_press\_and\_release\_key(  input [7:0] code  );  begin  fork  PS2\_gen\_byte\_clk();  PS2\_code\_input(code);  join  #code\_space\_period;  fork  PS2\_gen\_byte\_clk();  PS2\_code\_input(STOP\_CODE);  join  #code\_space\_period;  fork  PS2\_gen\_byte\_clk();  PS2\_code\_input(code);  join  end  endtask |

*Продолжение Листинг В1*

|  |
| --- |
| // Генерация пакета данных  task automatic PS2\_code\_input(  input [7:0] code  );  begin  PS2\_dat <= 0;  for (i = 0; i < 8; i = i + 1)  begin  @(posedge PS2\_clk)  PS2\_dat <= code[i];  end    @(posedge PS2\_clk)  PS2\_dat <= ~^(code);    @(posedge PS2\_clk)  PS2\_dat <= 1;  end  endtask  // Генерация синхросигнала для передачи пакета  task automatic PS2\_gen\_byte\_clk;  begin  #(clk\_period);  repeat(22)  begin  PS2\_clk <= ~PS2\_clk;  #(PS2\_clk\_period);  end  PS2\_clk <= 1;  end  endtask  function [7:0] HEX\_CD;  input [3:0] number\_in;  begin  case(number\_in)  4'h0: HEX\_CD = 8'h45;  4'h1: HEX\_CD = 8'h16;  4'h2: HEX\_CD = 8'h1E;  4'h3: HEX\_CD = 8'h26;  4'h4: HEX\_CD = 8'h25;  4'h5: HEX\_CD = 8'h2E;  4'h6: HEX\_CD = 8'h36;  4'h7: HEX\_CD = 8'h3D;  4'h8: HEX\_CD = 8'h3E;  4'h9: HEX\_CD = 8'h46;  4'hA: HEX\_CD = 8'h1C;  4'hB: HEX\_CD = 8'h32;  4'hC: HEX\_CD = 8'h21;  4'hD: HEX\_CD = 8'h23;  4'hE: HEX\_CD = 8'h24;  4'hF: HEX\_CD = 8'h2B;  default: HEX\_CD = 0;  endcase  end  endfunction  endmodule |

**Приложение В2**

*Листинг В2 — описание тест-модуля VGA*

|  |
| --- |
| module vga\_test();  reg clk;  reg rst;  reg [2:0] sw;    wire h\_sync, v\_sync;  wire [2:0] rgb ;    initial  begin  rst =0;  clk =0;  sw =3'b010;  end    always # 5 //100 MHz  clk = ~clk;    vga\_top uut (  .clk(clk),  .reset(rst),  .sw(sw),  .hsync(h\_sync),  .vsync(v\_sync),  .rgb(rgb)  );      endmodule |

**Приложение В3**

*Листинг В3— описание тест-модуля конечного автомата*

|  |
| --- |
| module Test\_game();  reg [7:0] key\_kode;  reg clk,rst,ready,rx\_done\_tick;  wire [9:0] wolf\_x,wolf\_y,egg\_x,egg\_y;  wire [6:0] egg\_count;  wire fail;  game uut (  .clk(clk),  .rst(rst),  .ready(ready),  .rx\_done\_tick(rx\_done\_tick),  .key\_kode(key\_kode),  .wolf\_x(wolf\_x),  .wolf\_y(wolf\_y),  .egg\_x (egg\_x),  .egg\_y (egg\_y)    ); initial  begin  clk =0;  rst =0;  ready=1;  rx\_done\_tick =1;  key\_kode =8'h42;  end  always#5 clk=~clk;  endmodule |

**Приложение Д1**

*Листинг Д1— файл проектных ограничений*

|  |
| --- |
| create\_clock -add -name sys\_clk\_pin -period 10.00 -waveform {0 5} [get\_ports  {clk}]  set\_property IOSTANDARD LVCMOS33 [get\_ports { clk }]  set\_property PACKAGE\_PIN E3 [get\_ports { clk }]  set\_property PACKAGE\_PIN N17 [get\_ports { reset }]  set\_property IOSTANDARD LVCMOS33 [get\_ports { reset }]  set\_property PACKAGE\_PIN P17 [get\_ports { ready } ]  set\_property IOSTANDARD LVCMOS33 [get\_ports { ready } ]  set\_property PACKAGE\_PIN F4 [get\_ports { ps2c }]  set\_property IOSTANDARD LVCMOS33 [get\_ports { ps2c }]  set\_property PACKAGE\_PIN B2 [get\_ports { ps2d }]  set\_property IOSTANDARD LVCMOS33 [get\_ports { ps2d }]  set\_property PACKAGE\_PIN B11 [get\_ports { h\_sync }]  set\_property IOSTANDARD LVCMOS33 [get\_ports { h\_sync }]  set\_property PACKAGE\_PIN B12 [get\_ports { v\_sync }]  set\_property IOSTANDARD LVCMOS33 [get\_ports { v\_sync }]  set\_property PACKAGE\_PIN A4 [get\_ports { VGA\_R\_3 }]  set\_property IOSTANDARD LVCMOS33 [get\_ports { VGA\_R\_3 }]  set\_property PACKAGE\_PIN A6 [get\_ports { VGA\_G\_3 }]  set\_property IOSTANDARD LVCMOS33 [get\_ports { VGA\_G\_3 }]  set\_property PACKAGE\_PIN D8 [get\_ports { VGA\_B\_3 }]  set\_property IOSTANDARD LVCMOS33 [get\_ports { VGA\_B\_3 }] |

**Приложение Д2**

Руководство пользователя:

1.Для запуска игры нажмите кнопку Ready, на отладочной плате

2.Для управления волком используете клавиши клавиатуры «W»

, «A», «S», «D».

3.Если вы проиграли, нажмите кнопку Reset для сброса игры.

4.При проигрыше вы теряете управление.