UNIVERSITATEA DE STAT DIN MOLDOVA

FACULTATEA DE FIZICĂ ȘI INGINERIE

DEPARTAMENTUL FIZICA TEORETICĂ “IU.PERLIN”

ABDULLAEV RAMIL

ELABORAREA HARDWARE ȘI SOFTWARE A UNEI LAMPE DECORATIVE

061 TEHNOLOGII ALE INFORMAȚIEI ȘI COMUNICAȚIILOR

TEHNOLOGII INFORMAȚIONALE ÎN MODELARE (MP)

Teză de master

Șef departament **\_\_\_\_\_\_\_\_\_\_\_\_\_\_ NICA Denis, dr.habil., prof.univ.**

Conducător ştiinţific **\_\_\_\_\_\_\_\_\_\_\_\_\_\_ NICA Denis, dr.hab., prof.univ.**

Autorul \_\_\_\_\_\_\_\_\_\_\_\_\_\_ **ABDULLAEV Ramil**

CHIȘINĂU-2023

МОЛДАВСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

ФАКУЛЬТЕТ ФИЗИКИ И ИНЖИНЕРИИ

ДЕПАРТАМЕНТ ТЕОРЕТИЧЕСКОЙ ФИЗИКИ ИМ. Ю. ПЕРЛИНА

АБДУЛЛАЕВ РАМИЛ

РАЗРАБОТКА ГЕНЕРАТОРА ТЕСТОВЫХ ВИДЕОСИГНАЛОВ:УПРАВЛЕНИЕ ПЛАТОЙ С ПОМОЩЬЮ МИКРОКОНТРОЛЛЕРА AtXmega128A1

061 ТЕХНОЛОГИИ ИНФОРМАЦИИ И КОММУНИКАЦИЙ

ИНФОРМАЦИОННЫЕ ТЕХНОЛОГИИ В МОДЕЛИРОВАНИИ (MP)

Магистерская диссертация

Зав. департаментом: **\_\_\_\_\_\_\_\_\_\_\_\_\_\_Денис НИКА, др. хаб., доцент**

Научный руководитель: **\_\_\_\_\_\_\_\_\_\_\_\_\_\_Денис НИКА, др. хаб., доцент**

Автор \_\_\_\_\_\_\_\_\_\_\_\_\_\_**Рамил АБДУЛЛАЕВ**

**КИШИНЕВ-2023**

СОДЕРЖАНИЕ

[СПИСОК АББРЕВИАТУР 5](#_Toc134782002)

[АННОТАЦИЯ 6](#_Toc134782003)

[ADNOTARE 7](#_Toc134782004)

[ВВЕДЕНИЕ 8](#_Toc134782005)

[Глава I. ОБЗОР ТЕХНИЧЕСКИХ ВОЗМОЖНОСТЕЙ ПОСТРОЕНИЯ СОВРЕМЕННЫХ СИСТЕМ НА БАЗЕ МИКРОКОНТРОЛЛЕРОВ И МИКРОСХЕММ ПРОГРАММИРУЕМОЙ ЛОГИКИ 10](#_Toc134782006)

[1.1 Краткий обзор различных архитектур микропроцессоров 11](#_Toc134782007)

[1.2 Особенности работы микроконтроллеров 12](#_Toc134782008)

[Отличия микроконтроллера от микропроцессора 13](#_Toc134782009)

[1.3 Микроконтроллеры семейства AtXmega 14](#_Toc134782010)

[1.4 Среда разработки IAR 5.3 16](#_Toc134782011)

[1.5 Способы передачи данных между микросхемами. 17](#_Toc134782012)

[1.6 Методы загрузки микросхем FPGA 21](#_Toc134782013)

[1.7 Заключение 22](#_Toc134782014)

[Глава II. ПОСТАНОВКА ЗАДАЧИ И АНАЛИЗ МЕТОДОВ РЕШЕНИЯ ПОСТАВЛЕННЫХ ЗАДАЧ 24](#_Toc134782015)

[2.1 Постановка задачи 24](#_Toc134782016)

[2.2 Описание выбранного способа загрузки FPGA 24](#_Toc134782017)

[2.3 Краткое описание возможностей микросхемы HDMI выхода AD9889B 25](#_Toc134782018)

[2.4 Конфигурация микросхемы AD9889B 26](#_Toc134782019)

[2.5 Использование роторного переключателя 27](#_Toc134782020)

[2.6 Заключение 29](#_Toc134782021)

[Глава III. РАЗРАБОТКА ПРОГРАММЫ УПРАВЛЕНИЯ ГЕНЕРАТОРА ТЕСТОВЫХ ВИДЕОСИГНАЛОВ 30](#_Toc134782022)

[3.1 Принципиальная схема 30](#_Toc134782023)

[3.2 Разработка программы управления 30](#_Toc134782024)

[3.3 Глобальные переменные 30](#_Toc134782025)

[3.3.1 Начальная инициализация 30](#_Toc134782026)

[3.3.2 Работа основной части программы 32](#_Toc134782027)

[3.3.3 Отработка прерывания по таймеру 34](#_Toc134782028)

[3.3.4 Обновление регистров выбранного видеосигнала 35](#_Toc134782029)

[3.4 Заключение 38](#_Toc134782030)

[ВЫВОДЫ И РЕКОМЕНДАЦИИ 39](#_Toc134782031)

[БИБЛИОГРАФИЯ 40](#_Toc134782032)

[[14] Цветовая температура, http://www.ylati.ru/czvetovaya\_temperatura.html 40](#_Toc134782033)

[ДЕКЛАРАЦИЯ ОБ ОТВЕТСТВЕННОСТИ 43](#_Toc134782034)

СПИСОК АББРЕВИАТУР

ARM – Advanced RISC (Reduced Instruction Set Computer) Machine

AVR – Alf and Vegard's RISC processor

EEPROM – Electrically Erasable Programmable Read-Only Memory

FPGA **-**  Field-Programmable Gate Array

SPI **-**  Serial Peripheral Interface

I2C – Philips two wire interface

ПО – Программное обеспечение

АННОТАЦИЯ

Абдуллаев Рамил, Разработка Генератора тестовых видеосигналов: программа микроконтроллера, Кишинев, 2023. Введение, три главы, выводы и рекомендации, библиография из 15 наименований, 63 страниц, 2 приложения, 17 рисунков, 1 таблица.

**Ключевые слова**

AVR, микроконтроллер, FPGA, схемотехника, роторный переключатель.

**Цели и задачи работы**

Целью работы является разработка устройства генерации тестовых видео изображений, основанного микросхеме программируемой логики, микроконтроллере фирмы Atmel, а также микросхеме физического уровня HDMI. Для осуществления поставленной цели в работе необходимо решить следующие задачи: разработка электрической принципиальной схемы, сборка конструктива и реализация программного обеспечения.

**Теоретическая и прикладная значимость**

В процессе выполнения работы были изучены способы загрузки микросхем программируемой логики. Кроме этого были получен опыт в реализации систем управления посредством роторного переключателя.

ADNOTARE

Abdullaev Ramil, Elaborarea hardware și software a unui dispozitiv dispozitiv de iluminat decorativ, Chișinău, 2023. Introducere, trei capitole, concluzii și recomandări, bibliografie din 15 titluri, 63 pagini, 2 anexe, 17 figuri, 1 tabelа.

**Cuvintele-cheie**

arduino, microcontroler, RGB, lumina diodice, proiectare a circuitelor.

**Scopul si obiectivele cercetarii**

Scopul acestei lucrări este elaborarea dispozitivului iluminat decorativ bazat pe bandă led de adresă. Pentru a îndeplini scopul stabilit în lucrare este necesar de rezolvat următoarele obiective: dezvoltarea unui concept electric, asamblarea unei construcții și implementarea de software.

**Valoarea teoretică şi aplicativă**

În procesul de realizare a lucrării, au fost studiate elementele de bază ale funcționării LED-urilor - surse de radiații optice semiconductoare. În plus, au fost obținute abilitățile de bază ale programării microcontrolerelor în mediul Arduino.

ВВЕДЕНИЕ

*Актуальность и важность темы*

В последнее время широкое распространение получило использование микросхемм программируемой логики. ПЛИС используются насегодняшний день в таких областях как:

* Цифровая обработка сигналов – это метод обработки информации, основанный на определенной последовательности с установленным периодом дискретизации.
* Телекоммуникации – ПЛИС идеально подходит для приложений, где важны скорость и энергоэффективность, но его также можно использовать в более сложных системах.
* Искусственный интеллект (ИИ) быстро развивается, регулярно появляются новые модели, методы и варианты использования нейронных сетей.
* Системы обраработки несжатого видео в реальном времени(видео микшеры, преобразователи видео сигналов)

*Цели и задачи работы*

Целью работы является изучение технических возможностей современных микросхемм ПЛИС и конструкций построенных на их основе. Для достижения поставленной цели в работе предлагается осуществить следующие задачи:

* разработка программного обеспечения для управления для первоначальной загрузки микросхемы ПЛИС;
* разработка программного обеспечения для настройки микросхемы физического уровня HDMI(AD9889B);
* реализация программного обеспечения для управления переключателем тестовых видео изображений.

*Методологическая и теоретико-научная основы работы (методы исследования)*

Цели данной работы были достигнуты при использовании следующих технологий: микроконтроллеры семейства AVR AtxMEGA, ПЛИС семейства Cyclone IV, протокол передачи данных SPI, протокол передачи данных I2C.

*Теоретическая и практическая важность*

Результаты данной работы могут послужить обогащению знаний в проектировании устройств генерации тестовых видео изображений, программирования микроконтроллеров, а также систем обработки несжатого видео.

*Новизна исследованной темы*

Новизна полученных в работе результатов, в основном связана с проведенным анализом существующих на рынке технологий и компонент для создания генераторов тестовых видео изображений.

*Ключевые слова:* AVR, микроконтроллер, FPGA, схемотехника.

Глава I. ОБЗОР ТЕХНИЧЕСКИХ ВОЗМОЖНОСТЕЙ ПОСТРОЕНИЯ СОВРЕМЕННЫХ СИСТЕМ НА БАЗЕ МИКРОКОНТРОЛЛЕРОВ И МИКРОСХЕММ ПРОГРАММИРУЕМОЙ ЛОГИКИ

Понятие устройств генерации тестовых изображений

Устройства генерации тестовых изображений, построенные на базе микросхем ПЛИС, в настоящее время получили широкое распространение в различных сферах. Технологии производства ПЛИС позволяют строить на их базе различные устройства высокоскоростной обработки несжатого видео.

Одной из направлений применения ПЛИС является создание устройств генерации тестовых видео изображений. Такие устройства лишены недостатков связанных с декомпрессией видео так как значения пикселей генерируются в режиме реального времени.

Для полноценной работы ПЛИС необходимо осуществить процедуру начальной загрузки и дальнейшее управление. Для этих целей в данной работе используется микроконтроллер фирмы Atmel семейства AtXmega. В этой главе будут рассмотренны возможности микроконтроллер, а также способы реализации поставленных задач.

* 1. Краткий обзор различных архитектур микропроцессоров

Архитектура, то есть логическая организациямикропроцессора, однозначно определяет свойства, особенности и возможности построения вычислительной системы на базе данного микропроцессора.[1]

Большинство современных микропроцессоров при всем разнообразии их типов, моделей и производителей имеют одну из трех типов архитектуры: CISC, RISC и MISC (это относится к микропроцессорам универсального, а не специального применения).

**Архитектура CISC** (Complex Instruction Set Computer) - командо-комплексная система управления компьютером. Отличается повышенной гибкостью и расширенными возможностями персонального компьютера (РС), выполненного на микропроцессоре, и характеризуется:

1. большим числом различных по длине и формату команд;
2. использованием различных систем адресации;
3. сложной кодировкой команд.

**Архитектура RISC** (Reduced Instrucktion Set Computer) - командо-однородная система управления компьютером, имеет свои особенности:

1. использует систему команд упрощенного типа: все команды имеют одинаковый формат с простой кодировкой, обращение к памяти осуществляется командами загрузки (данных из RAM в регистр микропроцессора) и записи (данных из регистра микропроцессора в память), остальные используемые команды - формата регистр-регистр;
2. при высоком быстродействии допускается более низкая тактовая частота;
3. команда меньше нагружает RAM;
4. отладка программ на RISC более сложна, чем на CISC;
5. с архитектурой CISC программно несовместима.

**Архитектура MISC** (Multipurpose Instruction Set Computer) - многоцелевая командная система управления компьютером, сочетает в себе преимущества CISC и RISC. Элементная база состоит из отдельных частей (могут быть объединены в одном корпусе): основная часть (HOST - ведущая), архитектуры RISC CPU, а расширяемая часть - с подключением ПЗУ (ROM) микропрограммного управления. При этом вычислительная система приобретает свойства CISC: - основные команды работают на HOST, а команды расширения образуют адрес микропрограммы для своего выполнения. HOST выполняет команды за один такт, а расширение эквивалентно CPU со сложным набором команд (CISC). Наличие ПЗУ устраняет недостаток RISC, связанный с тем, что при компиляции с языка высокого уровня код операции (микропрограмма) уже дешифрирована и открыта для программиста.

В данной работе применена плата семейства Arduino Leonardo, построенная на базе микроконтроллера Atmega32U4. Данный микроконтроллер построен на RISC архитектуре. Более подробно особенности выбранной аппаратной платформы будут рассмотрены ниже.

* 1. Особенности работы микроконтроллеров

Современные микроконтроллеры (МК) относятся к полупроводниковым компонентам, основное назначение которых – управление электронными устройствами. Они встраиваются непосредственно в исполнительный модуль, задавая режимы работы обслуживаемых изделий. На их основе реализована оригинальная идея, позволяющая совместить в одном кристалле сразу несколько функций.

Подобно классическому компьютеру они содержат в своем составе не только центральный процессор. Помимо него в МК входят рабочая память, ПЗУ, а также элементы периферии, объединенные в похожем на микросхему корпусе.

**Что такое микроконтроллеры**

Современный микроконтроллер, можно сравнить с основным элементом любого компьютерного устройства – микропроцессором. Последний представляет собой центральную часть ПК, выполненную на основе интегральных схем. Название «микропроцессор» (МП) указывает на то, что именно в этом модуле происходят все вычислительные операции, определяющие производительность системы в целом.

Чтобы получить на его основе полноценную ЭВМ – необходимо дополнить МП целым рядом внешних устройств. Добиться этого удается добавлением к этому модулю оперативной памяти и периферийных портов ввода-вывода информации. В результате получается законченное автономное устройство или компьютер.

Подобно ПК микроконтроллеры содержат в своем составе следующие компоненты:

* Микропроцессор.
* Оперативную память.
* Комплект периферии, состоящей из портов обмена информацией с обслуживаемым устройством.

В отличие от ПК он не является полностью самостоятельным (автономным) изделием. Микроконтроллеры, как правило, встраиваются в гаджеты, игрушки и в другие исполнительные устройства, функционированием которых они управляют.

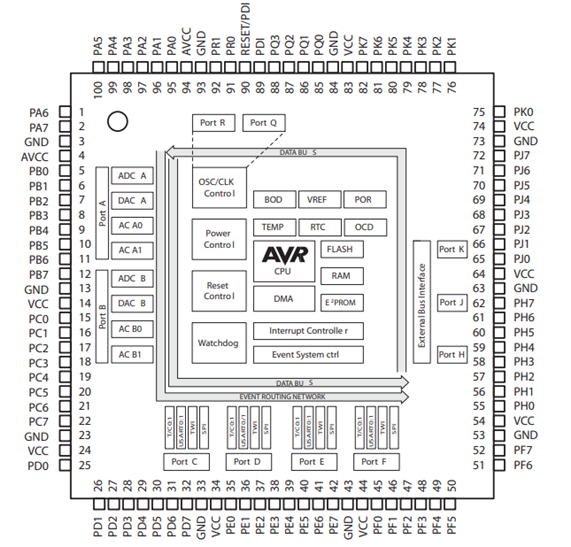
Отличия микроконтроллера от микропроцессора

  Разница между микропроцессором и микроконтроллером заключается в наличии оперативной памяти, ПЗУ и других периферийных устройств в микроконтроллере. Микропроцессор содержит только процессор и не имеет других компонентов.

Микропроцессор и микроконтроллер, оба являются основными процессорами, предназначенными для работы компьютеров. Функции обоих процессоров одинаковы. Основное различие между ними состоит в том, что микропроцессоры выполняют различные функции, тогда как микроконтроллеры - это небольшие компьютеры, предназначенные для конкретных задач.

* 1. Микроконтроллеры семейства AtXmega

Микроконтроллеры **AVR XMEGA** выполнены на основе: Технологии picoPower второго поколения Инновационной системы обработки событий "Event System", которая обеспечивает независимую от ЦПУ быстродействующую передачу данных между внутренними периферийными устройствами 4-канального контроллера ПДП, улучшающего характеристики микроконтроллера Быстродействующих 12-битных АЦП и ЦАП Ускорителя криптографических алгоритмов AES и DES  
      Микроконтроллеры AVR XMEGA производят такое же ошеломляющее впечатление, которое производили их предшественники микроконтроллеры AVR после своего появления. Новые микроконтроллеры способны работать при напряжении питания всего лишь 1.6 В и достигать производительности 32 MIPS на тактовой частоте 32 МГц. Микроконтроллеры содержат флэш-память размером 16…384 кбайт и поставляются в 44…100-выводных корпусах. Микроконтроллеры XMEGA являются микроконтроллерами общего назначения и могут использоваться в широком числе применений, в т.ч. аудиосистемы, ZigBee® системы, медицинская техника, контроллеры автоматизации, коммуникационное оборудование, измерительные приборы, оптические трансиверы, системы управления электроприводами, бытовое электрооборудование, оптические трансиверы и любая другая продукция с батарейным питанием.



**Рис. 1.2.** Структурная схема и система ввода вывода микроконтроллера AtXmega128.

Список основных возможностей микроконтроллеров семейства AVR xMega:

* Энергонезависимая память программ и данных
* 64–128 КБ внутрисистемной самопрограммируемой флэш-памяти
* 4K - 8Кб загрузочный раздел(область Boot Loader)
* 2 КБ EEPROM
* 4 КБ - 8 КБ внутренней SRAM
* Интерфейс внешней шины для SRAM до 16 Мбайт
* Интерфейс внешней шины для SDRAM до 128 Мбит
* Периферийные функции
  + Четырехканальный контроллер прямого доступа к памяти
  + Восьмиканальная система событий
  + Восемь 16-битных таймеров/счетчиков
    - Четыре таймера/счетчика с 4 выходными каналами сравнения или входными каналами захвата
    - Четыре таймера/счетчика с 2 выходными каналами сравнения или входными каналами захвата
    - Расширение высокого разрешения для всех таймеров/счетчиков
    - Усовершенствованное расширение сигнала (AWeX) на двух таймерах/счетчиках
  + Восемь USART с поддержкой IrDA для одного USART
  + Четыре двухпроводных интерфейса с двойным согласованием адресов (I2Совместимость с C и SMBus)
  + Четыре последовательных периферийных интерфейса (SPI)
  + Криптовалютный движок AES и DES
  + 16-битный счетчик реального времени (RTC) с отдельным генератором
  + Два шестнадцатиканальных 12-битных аналого-цифровых преобразователя со скоростью 2 мс
  + Два двухканальных 12-разрядных цифро-аналоговых преобразователя со скоростью 1 мс/с
  + Четыре аналоговых компаратора (AC) с функцией сравнения окна и источниками тока
  + Внешние прерывания на всех контактах ввода/вывода общего назначения
  + Программируемый сторожевой таймер с отдельным встроенным сверхмаломощным генератором
  + Поддержка библиотеки QTouch®
* Емкостные сенсорные кнопки, ползунки и колеса
* Специальные функции микроконтроллера
  + Сброс при включении питания и программируемое обнаружение отключения питания
  + Внутренние и внешние часы с PLL и предварительным делителем
  + Программируемый многоуровневый контроллер прерываний
  + Пять режимов сна
  + Интерфейсы программирования и отладки
    - Интерфейс JTAG (совместимый с IEEE 1149.1), включая сканирование границ
    - PDI (программный и отладочный интерфейс)
* Рабочее напряжение: 1,6 – 3,6 В
* Рабочая частота:
  + 0–12 МГц от 1,6 В
  + 0–32 МГц от 2,7 В
  1. Среда разработки IAR 5.3

Среда разработки IAR 5.3 - устаревшая версия среды разработки от компании IAR Systems, выпущенная более десяти лет назад. Тем не менее, она всё ещё может использоваться для разработки микроконтроллерных приложений на платформах, которые поддерживают эту версию.

Некоторые из возможностей, которые могут быть доступны в среде разработки IAR 5.3, включают в себя:

Поддержка различных микроконтроллеров: IAR 5.3 может поддерживать различные микроконтроллеры от разных производителей, таких как ARM, Renesas, Atmel, TI, и другие.

Интерфейс разработки: IAR 5.3 имеет интуитивный интерфейс разработки, который позволяет программистам создавать, отлаживать и тестировать свои проекты в одном месте.

Редактор кода: IAR 5.3 содержит редактор кода, который может автоматически завершать код и подсвечивать синтаксис для удобства работы с кодом.

Отладчик: IAR 5.3 содержит мощный отладчик, который позволяет программистам отлаживать свои проекты, искать ошибки и просматривать значения переменных и регистров в реальном времени.

Инструменты анализа: IAR 5.3 содержит инструменты анализа, которые позволяют оптимизировать код, анализировать использование памяти и тестировать производительность приложений.

Поддержка RTOS: IAR 5.3 может поддерживать различные операционные системы реального времени, такие как FreeRTOS, Micrium и другие.

Поддержка стандартов: IAR 5.3 поддерживает различные стандарты, такие как C, C++ и Assembler.

* 1. Способы передачи данных между микросхемами.

**SPI** (англ. Serial Peripheral Interface, SPI bus — последовательный периферийный интерфейс, шина SPI) — последовательный синхронный стандарт передачи данных в режиме полного дуплекса, предназначенный для обеспечения простого и недорогого высокоскоростного сопряжения микроконтроллеров и периферии. SPI также иногда называют четырёхпроводным (англ. four-wire) интерфейсом.



Рис. 1 Структурная схема сигналов шины SPI.

1. **MOSI** — выход ведущего, вход ведомого (англ. Master Out Slave In). Служит для передачи данных от ведущего устройства ведомому.
2. **MISO** — вход ведущего, выход ведомого (англ. Master In Slave Out). Служит для передачи данных от ведомого устройства ведущему.
3. **SCLK** — последовательный тактовый сигнал (англ. Serial Clock). Служит для передачи тактового сигнала для ведомых устройств.
4. **CS** или **SS** — выбор микросхемы, выбор ведомого (англ. Chip Select, Slave Select).

**Принцип работы**  
В простейшем случае к ведущему устройству подключено единственное ведомое устройство и необходим двусторонний обмен данными. В таком случае используется трехпроводная схема подключения. Интерфейс SPI позволяет подключать к одному ведущему устройству несколько ведомых устройств, причем подключение может быть осуществлено несколькими способами.  
Радиальная структура связи с несколькими ведомыми устройствами через SPI.

**Первый способ** позволяет реализовать радиальную структуру связи (топология типа «звезда»), его принято считать основным способом подключения нескольких ведомых устройств. В данном случае для обмена более чем с одним ведомым устройством ведущее устройство должно формировать соответствующее количество сигналов выбора ведомого устройства (SS). При обмене данными с ведомым устройством, соответствующий ему сигнал SS переводится в активное (низкое) состояние, при этом все остальные сигналы SS находятся в неактивном (высоком) состоянии. Выводы данных MISO ведомых устройств соединены параллельно, при этом они находятся в неактивном состоянии, а перед началом обмена один из выходов (выбранного ведомого устройства) переходит в активный режим.  
Кольцевая структура связи с несколькими ведомыми устройствами через SPI

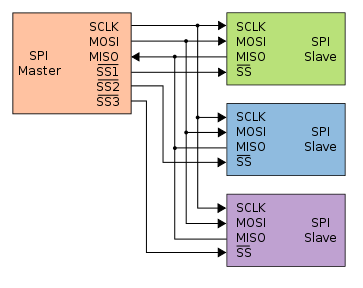


Рис. 1 Топология типа «звезда» при подключении сигналов шины SPI.

**Второй способ** позволяет реализовать структуру связи типа «кольцо». В данном случае для активации одновременно нескольких ведомых устройств используется один сигнал SS, а выводы данных всех устройств соединены последовательно и образуют замкнутую цепь. При передаче пакета от ведущего устройства этот пакет получает первое ведомое устройство, которое, в свою очередь, транслирует свой пакет следующему ведомому устройству и так далее. Для того, чтобы пакет от ведущего устройства достиг определенного ведомого устройства, ведущее устройство должно отправить еще несколько пакетов.

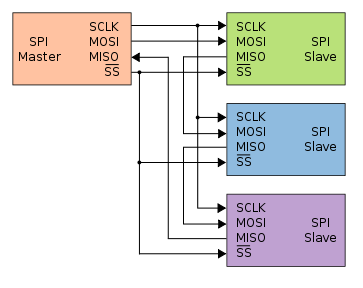


Рис. 1 Топология типа «кольцо» при подключении сигналов шины SPI.

**Особенности:**

Большим преимуществом шины SPI является наличие сигнала тактовой частоты. Многие SPI чипы и микроконтроллеры поддерживают работу на частоте более 10 МГц, обеспечивая намного более высокую скорость передачи данных, чем интерфейс I2C.

**Использование:**

Часто этот интерфейс используется для работы с SPI-Flash в которой хранится программное обеспечение. Кроме этого, SPI используется для сопряжения различных микросхем устройства для таких целей как конфигурация или считывание статусной информации.

**Протокол I2C.**

**I2C (Inter-Integrated Circuit)** — последовательная шина данных для связи интегральных схем, использующая две двунаправленные линии связи (SDA и SCL). Используется для соединения низкоскоростных периферийных компонентов с центральным микроконтроллером во встраиваемых системах.  
Иногда эту шину называют "квадратичной" или "квадратной" или "Ай-Ту-Си"

**Принцип работы:**



**Рис. 1 Топология подключения устройств по шине I2C.**

Микроконтроллер на рисунке — это ведущий элемент **(Master1)** им может быть процессор. На рисунке представлено 3 ведомых перефириных элемента **Slave** В качестве **Slave** могут быть память, ЦАП, АЦП и пр. К шине может быть подключено до 127 устройств.  
Процессор с памятью соединен в данном случае по двум шинам:  
**SDA** (Serial DATA)- шина последовательной передачи данных. Данные по этой шине могут передаваться в двух направлениях.  
**SCL** (Serial Clock) - шина, по которой идет тактирование шины данных. Шина синхронизации данных. Она также определяет в какой момент куда пойдут данные. В схеме Master-Master первым битом определяется, кто займет главную роль.  
**Скорость передачи данных.** Так как передаются по 1 биту за 1 такт, то скорость передачи данных составляет 1/8 от тактовой частоты.



Рис. 1 Осциллограмма состояний переходов сигналов при передаче данных по шине I2C.

**Состояние СТАРТ и СТОП**

В протоколе I2C существуют два важных состояния - СТАРТ (START) и СТОП (STOP). Они играют ключевую роль в инициации и завершении передачи данных по шине I2C. Вот краткое описание каждого из них:

1. **СТАРТ (START):**

* Состояние СТАРТ инициирует начало передачи данных по шине I2C.
* Мастер-устройство (генератор тактовых импульсов) генерирует СТАРТ-сигнал.
* СТАРТ-сигнал состоит из перехода линии SDA с высокого уровня (1) на низкий уровень (0) при нахождении линии SCL на высоком уровне (1).
* После СТАРТ-сигнала мастер выбирает адрес устройства, с которым он хочет общаться, и передает его по шине SDA.

1. **СТОП (STOP):**

* Состояние СТОП используется для завершения передачи данных по шине I2C.
* Мастер-устройство генерирует СТОП-сигнал.
* СТОП-сигнал состоит из перехода линии SDA с низкого уровня (0) на высокий уровень (1) при нахождении линии SCL на высоком уровне (1).
* СТОП-сигнал указывает, что передача данных завершена, и освобождает шину I2C для других устройств.

СТАРТ- и СТОП-сигналы определяют начало и конец каждой транзакции данных по шине I2C. Они служат важным механизмом для синхронизации и контроля передачи данных между мастером и слейвами на шине I2C.

* 1. Методы загрузки микросхем FPGA

**FPGA** (Field-Programmable Gate Array) — это интегральная схема, предназначенная для программирования после изготовления. FPGA представляет собой программируемую логическую матрицу, состоящую из логических элементов, макроклеток и программируемых межсоединений.

После подачи напряжения питания внутренние соединения микросхемы FPGA не настроены. Обычно настройка внутренних меж соединений хранится в отдельной микросхеме памяти (FLASH, EEPROM). Задача микроконтроллера в этом случае осуществить передачу данных из энергонезависимой памяти в микросхему программируемой логики.

Существует несколько методов загрузки микросхем FPGA, вот некоторые из них:

**Active Serial** - в этом методе конфигурационные данные передаются последовательно через один контакт микросхемы. Данный метод подходит для загрузки FPGA из внешней памяти, такой как EEPROM, Flash или FRAM. Микросхема FPGA при этом сама осуществляет синхронизацию и передачу данных.

**Passive Serial** - данный метод загрузки использует несколько контактов микросхемы FPGA для передачи конфигурационных данных. В этом методе используется контроллер, который передает данные на определенные контакты микросхемы FPGA.

**JTAG** - это метод, который использует интерфейс JTAG (Joint Test Action Group), который обычно используется для тестирования и отладки микросхем. Однако он также может быть использован для загрузки конфигурационных данных на FPGA.

**HPS** (Hard Processor System) - встроенный процессор загружает конфигурационные данные на FPGA. Данный метод может использоваться в FPGA с встроенным процессором.

**Remote Update** - в этом методе FPGA загружается через интерфейс Ethernet или другие беспроводные сети. Этот метод может использоваться для обновления FPGA на удаленном оборудовании.

Каждый из этих методов имеет свои преимущества и недостатки и выбор метода зависит от требований к конкретному приложению.

* 1. Заключение

В данной главе были рассмотрены аппаратные и программные средства для реализации генератора тестовых видеосигналов.

Были рассмотрены существующие архитектуры процессоров и в том числе архитектура AVR, используемая во встроенных системах. Были описаны способы обмена процессором информацией с периферийными устройствами посредством шин: I2C(two wire interface) и SPI(serial peripheral interface). Был рассмотрен микропроцессор AtXmega, его аппаратные компоненты и процесс написания программ под него.

Из периферийных устройств была представлена микросхема программируемой логики. Распараллеливание процессов а также высокая производительность позволяют решить задачу формирования видео изображения. Задача микроконтроллера в этом случае сводится к загрузке FPGA и управлению переключением выбора типа видео изображения.

Понимание принципа работы аппаратной и программной части является необходимым требованием и будет использоваться в дальнейшем при построении и анализе оптимальности построенного устройства.

Глава II. ПОСТАНОВКА ЗАДАЧИ И АНАЛИЗ МЕТОДОВ РЕШЕНИЯ ПОСТАВЛЕННЫХ ЗАДАЧ

1. Постановка задачи

Целью данной главы является изучение выбранного способа загрузки FPGA. Также будет рассмотрена процедура передачи данных при загрузке и способы решения задачи.

1. Описание выбранного способа загрузки FPGA

Для загрузки микросхемы FPGA будет использован метод Passive Serial.

Вот краткое описание этого способа:

1. Подготовка конфигурационных данных: сначала необходимо подготовить конфигурационные данные для FPGA. Эти данные определяют логику и соединения блоков внутри FPGA. Проект внутренней логики разрабатывается в специализированной IDE и не является частью этой мастерской работы. Результатом компиляции проекта является бинарный файл, который при помощи программатора переносится во Флеш память устройства. Задача микроконтроллера при этом обеспечить надежную передачу данных для загрузки FPGA из Флэш память в соответствии с выбранной процедурой загрузки.

2. Подключение микросхемы FPGA: Микросхема FPGA должна быть правильно подключена к программатору или контроллеру, который будет передавать конфигурационные данные. При правильном подключении микроконтроллер инициализирует начало процедуры конфигурации посредством сигнала nCONF (запрос на начало конфигурации). В ответ на запрос FPGA отвечает сигналом статуса nSTATUS. В случае если все условия удовлетворяют требуемым – ответ положительный (низкий уровень сигнала).

3. Последовательная передача данных: В методе passive serial данные передаются по нескольким контактам микросхемы FPGA последовательно. Контроллер или программатор отправляет биты данных последовательно через контакты FPGA. При этом микроконтроллер осуществляет побитную передачу загрузочных данных в FPGA.

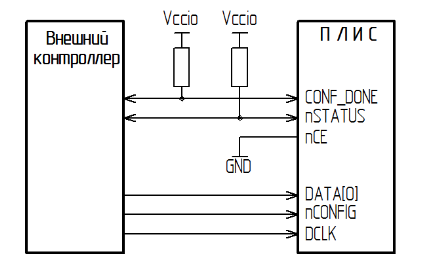
4. Управляющий сигнал: кроме передачи данных, также используется управляющий сигнал, обычно называемый "контрольным сигналом загрузки" (load control signal). Этот сигнал указывает микросхеме FPGA начало передачи новых данных.

5. Загрузка данных: Контроллер или программатор последовательно отправляет биты данных через контакты микросхемы FPGA в соответствии с протоколом передачи данных. Данные сохраняются во внутренней конфигурационной памяти FPGA.

6. Завершение загрузки: после передачи всех конфигурационных данных контроллер отправляет специальный сигнал, который указывает микросхеме FPGA завершить процесс загрузки и начать работу с новой конфигурацией.

7. Проверка и запуск: после успешной загрузки FPGA проверяет корректность конфигурации и переходит к выполнению задач, определенных внутри программы.

Метод passive serial прост в реализации и требует минимального количества контактов для передачи данных, но обычно имеет меньшую скорость загрузки по сравнению с некоторыми другими методами. Однако он широко используется во многих приложениях, особенно когда требуется простой и надежный способ загрузки FPGA.



**Рис.1.** Подключения ПЛИС к внешнему контроллеру для загрузки в режиме PS.

1. Краткое описание возможностей микросхемы HDMI выхода AD9889B

Микросхема AD9889B является высококачественным HDMI-трансмиттером, который обеспечивает широкий набор возможностей для передачи аудио и видео сигналов по интерфейсу HDMI. Вот краткое описание основных возможностей микросхемы AD9889B:

1. Поддержка HDMI-стандартов: Микросхема AD9889B полностью соответствует стандартам HDMI 1.3a и DVI 1.0. Она обеспечивает высокое качество передачи видео и аудио сигналов с разрешением до 1080p.

2. Входные интерфейсы: AD9889B имеет различные входные интерфейсы для подключения к источнику видеосигнала. Она поддерживает аналоговый компонентный видеоинтерфейс (YPbPr) и аналоговый RGB-интерфейс.

3. Кодирование видеосигнала: Микросхема AD9889B предоставляет возможность кодирования аналоговых видеосигналов в цифровой формат HDMI для передачи по интерфейсу HDMI.

4. Поддержка аудио: AD9889B поддерживает передачу аудиосигналов с различными форматами, включая двухканальный PCM (Pulse Code Modulation), многоканальный линейный PCM и сжатые аудиоформаты, такие как Dolby Digital и DTS.

5. Гибкая настройка: Микросхема AD9889B обеспечивает различные настройки и конфигурации, позволяющие пользователю оптимизировать передачу видео и аудио сигналов в соответствии с требованиями конкретного приложения.

6. Защита контента: AD9889B поддерживает технологию HDCP (High-bandwidth Digital Content Protection), которая обеспечивает защиту цифрового контента от несанкционированного копирования или перехвата.

7. Низкое энергопотребление: Микросхема AD9889B имеет низкое энергопотребление, что делает ее подходящей для применения в системах с ограниченным энергоснабжением.

Микросхема AD9889B предоставляет широкий набор функций и возможностей для передачи видео и аудио сигналов по интерфейсу HDMI. Она является надежным и высококачественным решением для передачи HDMI-сигналов в различных приложениях.

1. Конфигурация микросхемы AD9889B

Микросхема AD9889B конфигурируется путем программирования регистров внутри самой микросхемы. Конфигурация определяет параметры работы микросхемы, такие как разрешение видео, режимы цветности, настройки аудио и другие параметры.

Для конфигурации микросхемы AD9889B используется последовательный интерфейс контроля (I2C). Через этот интерфейс можно записывать значения в регистры микросхемы для установки нужных параметров.

Процесс конфигурации обычно включает следующие шаги:

1. Подключение: Микросхема AD9889B должна быть правильно подключена к контроллеру или микроконтроллеру, который будет осуществлять программирование регистров.

2. Инициализация: перед началом конфигурации микросхемы AD9889B необходимо выполнить инициализацию, включающую установку начальных значений регистров.

3. Запись значений в регистры: Контроллер или микроконтроллер записывает значения параметров в соответствующие регистры микросхемы AD9889B. Эти значения определяют требуемые настройки для разрешения видео, цветности, аудио и других функций.

4. Проверка: после записи значений в регистры микросхемы AD9889B можно выполнить проверку, чтобы убедиться, что конфигурация прошла успешно. Можно считать значения регистров и сравнить их с ожидаемыми значениями.

5. Рабочий режим: после завершения конфигурации микросхема AD9889B работает в соответствии с заданными параметрами, передавая видео- и аудиосигналы по интерфейсу HDMI.

Важно отметить, что конкретные регистры и их значения для конфигурации микросхемы AD9889B зависят от требований и настроек конкретного приложения. Руководство пользователя или документация микросхемы предоставят подробную информацию о доступных регистрах и их настройках.

1. Использование роторного переключателя

Роторный переключатель (Rotary Switch) — это механическое устройство, используемое для выбора одного из нескольких предопределенных вариантов. Он обычно состоит из вала и поворотной крышки с контактами, которые соответствуют различным положениям переключателя.

Использование роторного переключателя сводится к вращению его поворотной крышки для выбора нужного положения. При вращении контакты внутри переключателя переключаются между различными положениями и устанавливаются в соответствующем положении.

Роторные переключатели широко используются в различных приложениях, где требуется выбор между несколькими опциями или настройками. Например, они могут быть использованы в аудиоустройствах для выбора источника звука или регулировки громкости, в электронных приборах для выбора режимов работы или настройки параметров, а также в промышленных системах для выбора режимов работы или управления.

Для использования роторного переключателя необходимо принять во внимание его конструкцию и особенности. Некоторые роторные переключатели могут иметь дополнительные функции, такие как щелчок при переключении или индикацию текущего положения. Также важно правильно подключить контакты переключателя к соответствующим компонентам или цепям в системе.

Роторные переключатели предоставляют удобный и надежный способ выбора опций или настроек в различных устройствах и системах.

A picture containing circuit component, passive circuit component, electronic component

Description automatically generated

**Рис.1. Различные варианты исполнения роторных переключателей**.

В данной работе используется восьмипозиционный роторный переключатель. Не вдаваясь в детали его внутреннего устройства следует отметить что такой переключатель позволяет пользователю выбрать до восьми разных видео изображений.

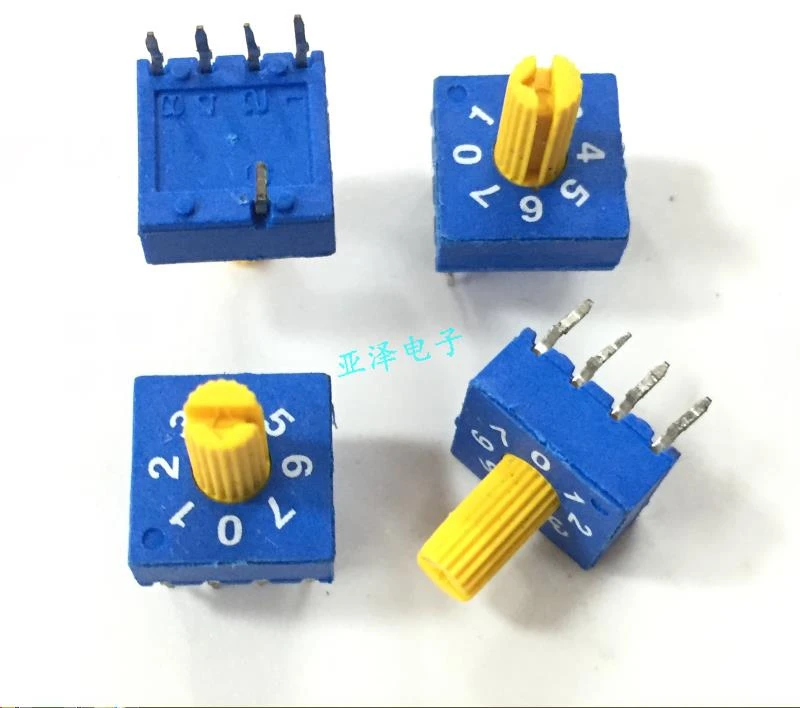


Рис.1.Роторный переключатель на 8 положений.

1. Заключение

Вторая глава была посвящена изучению способов решения поставленной задачи. В качестве основных компонентов будущего устройства были выбраны компоненты, соответствующие минимальным требованиям системы. В качестве метода загрузки FPGA был выбран способ Passive Serial. Такой способ загрузки микросхемы программируемой логики позволяет осуществить начальную перегрузку данных за приемлемое время. Также такой способ минимизирует количество используемых портов микроконтроллера. Кроме этого, микроконтроллер имеет полный доступ как для чтения, так и для записи к микросхеме Флэш памяти, в которой хранится загрузочный файл. Такое решение позволит обновлять загрузочный файл через какой-либо из доступных микроконтроллеру интерфейсов.

В качестве микросхемы вывода видео потока HDMI была выбрана микросхема фирмы Analog Devices серии AD9889B. Такое решение требует от микроконтроллера только первоначальной загрузки массива инициализации регистров. Массив инициализации регистров был сформирован исходя из файла описания микросхемы от производителя.

В качестве способа выбора выходного сигнала был выбран восьмипозиционный роторный переключатель. Такое решение позволяет пользователю изменять параметры устройства без применения других внешних устройств, например компьютера. Роторный переключатель подключен к портам ввода-вывода микроконтроллера. Состояние портов отслеживается в прерывании от таймера. Такое решение позволяет программным способом решить проблему дребезга контактов при переключении, а переключение картинки происходит для пользователя практически мгновенно.

Глава III. РАЗРАБОТКА ПРОГРАММЫ УПРАВЛЕНИЯ ГЕНЕРАТОРА ТЕСТОВЫХ ВИДЕОСИГНАЛОВ

В результате практической части данной работы был разработана программа управления генератором тестовых видеоизображений. Были реализованы программные модули загрузки микросхемы ПЛИС, инициализации микросхемы физического уровня HDMI, а также модуль прерывания по таймеру для обработки сигналов от роторного переключателя.

1. Принципиальная схема

Принципиальная схема разрабатываемого устройства состоит из трех основных компонентов - микроконтроллер, ПЛИС, микросхема физического уровня HDMI.

A picture containing text, diagram, line, number

Description automatically generated

Рис. 1 Принципиальная схема генератора.

1. Разработка программы управления

.

1. Инициализация портов ввода-вывода
   1. Начальная инициализация

В этой части необходимо определить необходимое для полной работы программы количество глобальных переменных. Конечным результатом работы программы будет отображение какой-либо информации светодиодами. Используемая в проекте библиотека FastLED предлагает свой класс для хранения и обработки значений цветов для светодиодов – тип CRGB. В программном коде создается массив элементов этого класса. Количество элементов массива равно количеству светодиодов в светильнике. Для начала найдем в файлах библиотеки описание этого класса:

struct CRGB {

union {

struct {

union {

uint8\_t r;

uint8\_t red;

};

union {

uint8\_t g;

uint8\_t green;

};

union {

uint8\_t b;

uint8\_t blue;

};

};

uint8\_t raw[3];

};

};

**Листинг 3.1.** Описание класса CRGB.

Как видно из описания, каждый объект такого класса содержит три байта информации. Каждый байт описывает одну из цветовых компонент светодиода – красную, зелёную и синюю. Кроме того, к элементам можно обращаться напрямую через массив элементов raw[3]. В классе также описаны некоторые процедуры работы с элементами класса.

В следующем листинге представлен пример использования директивы #define для определения размерности матрицы светодиодов.

#define WIDTH 8

#define HEIGHT 11

#define NUM\_LEDS WIDTH \* HEIGHT

**Листинг 3.2.** Определение размерности матрицы светодиодов.

При желании можно применить программный код для другого количества светодиодов. Эти определения будут использоваться во всей программе при работе со светодиодами. Первое применение одного из определений – создание массива объектов из 88 светодиодов.

CRGB leds[NUM\_LEDS];

**Листинг 3.3.** Создание массива обьектов.

Для работы с кнопкой создаем объект класса GButton. Атрибуты метода создания объекта это:

1. Номер пина к которому подключена кнопка (BTN\_PIN);
2. Активация внутренней подтяжки к питанию (HIGH\_PULL);
3. Исходное состояние кнопки – разомкнутое состояние (NORM\_OPEN)

GButton touch(BTN\_PIN, HIGH\_PULL, NORM\_OPEN);

**Листинг 3.4.** Создание обьекта для работы с кнопкой.

Для хранения настроек режимов работы в программе объявляется массив структур режимов. Каждый режим характеризуется тремя характеристиками: общая яркость, скорость выполнения эффекта, масштаб эффекта. Не все режимы используют каждый и этих параметров, но для единообразия это оказалось удобно. Для определения количества режимов используется директива #define. При изменении номера режима в сторону увеличения, программа остановится на определенном ранее количестве. Таким образом, всегда можно описать свой режим и добавить его в обработчик режимов или уменьшить количество режимов.

**struct** {

byte brightness = 50;

byte speed = 100;

byte scale = 20;

} modes[MODE\_AMOUNT];

**Листинг 3.5.** Объявление массива структур режимов.

* 1. Работа основной части программы

Правильная начальная инициализация переменных и объектов это залог хорошей работы всей программы. Программная среда Arduino предлагает использовать специальную функцию *setup()* для начальной инициализации объектов. Эта функция вызывается один раз при старте программы.

Первая часть функции настраивает объект класса FastLED при помощи процедуры *FastLED*.*addLeds*. Необходимо указать тип используемых светодиодов - WS2812B в нашем случае. Следующий параметр – номер пина, к которому подключена лента - LED\_PIN. Этот номер определён с помощью директивы #define выше в программе. Такое решение позволяет оперативно менять номер при настройке параметров платы. Также необходимо указать порядок цветов при передаче данных в ленту. Данные в ленту передаются последовательно – бит за битом. Для каждого светодиода необходимо передать три байта или двадцать четыре бита. Для разных лент эта последовательность может отличаться. В нашем случае это GRB – то есть сначала передается байт зеленого, за ним красный, а последним синий. Ниже представлен листинг части функции *setup(),* отвечающей за настройку параметров библиотеки светодиодной ленты.

//Настройка ленты

FastLED.addLeds<WS2812B, LED\_PIN, COLOR\_ORDER>(leds, NUM\_LEDS);

FastLED.setBrightness(BRIGHTNESS);

//Установка максимального тока блока питания

**if** (CURRENT\_LIMIT > 0)

FastLED.setMaxPowerInVoltsAndMilliamps(5, CURRENT\_LIMIT);

FastLED.clear();//Установка всех значений в ноль

FastLED.show();//Запись значений в светодиодную ленту для отображения

//

**Листинг 3.6.** Настройка библиотеки FastLED.

В проекте также используется библиотека работы с кнопкой. Для правильной работы кнопки устанавливаем временные интервалы между кратковременными нажатиями, а также тайм-аут окончания нажатия.

//настройка параметров кнопки

touch.setStepTimeout(100);//интервал между кликами

touch.setClickTimeout(500; )//тайм-аут

**Листинг 3.7.** Настройка параметров кнопки.

После первого включения и до того как настройки будут сохранены все режимы конфигурируются одинаково – настройками по умолчанию. Конфигурация по умолчанию происходит при определении массива структур. Значения настроек по умолчанию были подобраны по личному усмотрению. При каждом сохранении настроек в первый байт энергонезависимой памяти (с адресом ноль) записывается число сто два. Это своеобразный признак того, что настройки в памяти это не случайный набор чисел.

**if** (EEPROM.read(0) == 102) {// если было сохранение настроек, то восстанавливаем их из EEPROM

currentMode = EEPROM.read(1);

**for** (byte x = 0; x < MODE\_AMOUNT; x++) {

modes[x].brightness = EEPROM.read(x \* 3 + 11); // (2-10 байт - резерв)

modes[x].speed = EEPROM.read(x \* 3 + 12);

modes[x].scale = EEPROM.read(x \* 3 + 13);

}

}

**Листинг 3.8.** Чтение настроек из энергонезависимой памяти.

Кроме всего вышеперечисленного в функции начальной инициализации происходит настройка последовательно порта, при помощи которого происходила отладка программы. Так как в программе есть эффекты, которые требуют генерации случайных чисел, тут же вызывается настройка генератора псевдослучайных чисел. В листинге ниже представлен полный код функции *setup().*

**void** setup() {

//Настройка ленты

FastLED.addLeds<WS2812B, LED\_PIN, COLOR\_ORDER>(leds, NUM\_LEDS);

FastLED.setBrightness(BRIGHTNESS);

//Установка максимального тока блока питания

**if** (CURRENT\_LIMIT > 0)

FastLED.setMaxPowerInVoltsAndMilliamps(5, CURRENT\_LIMIT);

FastLED.clear();//Установка всех значений в ноль

FastLED.show();//Запись значений в светодиодную ленту для отображения

//

//настройка параметров кнопки

touch.setStepTimeout(100);//интервал между кликами

touch.setClickTimeout(500; )//тайм-аут

//

//Функция RandomSeed() инициализирует генератор псевдослучайных чисел.

randomSeed(analogRead(0));

//

//�нициализация последовательного порта

Serial.begin(9600);

  #ifdef DEBUG\_PRINT

delay(1000);

Serial.println();

Serial.println("LED lamp start");

  #endif

//

//Чтение Настроек из EEPROM

// если было сохранение настроек, то восстанавливаем их из EEPROM

**if** (EEPROM.read(0) == 102) {

currentMode = EEPROM.read(1);

**for** (byte x = 0; x < MODE\_AMOUNT; x++) {

modes[x].brightness = EEPROM.read(x \* 3 + 11); // (2-10 байт - резерв)

modes[x].speed = EEPROM.read(x \* 3 + 12);

modes[x].scale = EEPROM.read(x \* 3 + 13);

}

}

}

**Листинг 3.9.** Полный код функции *setup().*

* 1. Отработка прерывания по таймеру

В программной среде Arduino основной цикл программы определяется функцией *loop().* Для лучшего понимания можно себе представить, что содержимое этой функции вставляется в один из вариантов бесконечного цикла. Другой вариант – вызов функции *loop()* внутри того же бесконечного цикла. В моей программе содержимое этой функции это последовательный вызов двух других функций. Одна из них это **Менеджер эффектов**, а вторая **Отработка нажатий на кнопку**. Обе эти функции будут описаны ниже.

**void** loop() {

effectsTick();//Менеджер эффектов

buttonTick();//Отработка нажатий на кнопку

}

**Листинг 3.10.** Полный код функции *loop().*

* 1. Обновление регистров выбранного видеосигнала

Основная часть работы светильника это индикация текущего режима. Программный Менеджер эффектов индицирует текущий режим в зависимости от состояния переменной currentMode. Эта переменная является глобальной для проекта. Её начальное состояние определяется при создании. Если настройки были сохранены, то номер текущего режима загрузится из энергонезависимой памяти в функции setup(). Номер текущего режима можно изменить быстрым двойным нажатием на кнопку управления. Выбор режима для отображения в программе производится при помощи оператора принятия решений switch. Номер режима, хранящийся в переменной currentMode должен быть в интервале от 0 до значения, определенного директивой define.

#define MODE\_AMOUNT 9//количество режимов

**Листинг 3.11.** Определение количества режимов*.*

В случае если оператор *switch()* по каким-то причинам не нашел совпадения, то программа перейдет к оператору default и будет индицироваться режим по умолчанию – белая лампа (*whiteLamp();*).

Важно отметить как именно скорость текущего режима (в программном коде это *modes[currentMode].speed*) влияет на работу. В этом мне очень помогла функция, которая является стандартной для Программной среды Arduino - *millis().* Эта функция возвращает количество миллисекунд с момента начала выполнения текущей программы на плате Arduino. Это количество сбрасывается на ноль, в следствие переполнения 32-х битного счетчика, приблизительно через 50 дней. Этого времени с запасом хватает для работы программы. Моя программа засекает текущий момент при каждом вызове функции *effectsTick()* и дожидается когда пройдет интервал времени, определённый в параметре скорости текущего режима. Если необходимое время еще не прошло, то происходит выход из подпрограммы. В случае, когда необходимый интервал времени прошел, происходит вызов функции отображения текущего режима и сохранение текущего значения времени для ожидания следующего момента времени.

**void** effectsTick() {

uint32\_t CurrentTime = millis() - effTimer;

**if** (ONflag && (CurrentTime >= modes[currentMode].speed)) {

effTimer = millis();//сохранение текущего времени

//Вызов одной из функций индикации

**switch** (currentMode) {

**case** SPARKLES\_MODE:

sparklesRoutine();

**break**;

**case** FILL\_MODE:

fillRoutine();

**break**;

**case** RNBOW\_HORION\_MODE:

rainbowHorizontal();

**break**;

**case** RNBOW\_VERTIC\_MODE:

rainbowVertical();

**break**;

**case** COLOR\_S\_MODE:

colorsRoutine();

**break**;

**case** COLOR\_MODE:

colorRoutine();

**break**;

**case** SNOWFALL\_MODE:

snowRoutine();

**break**;

**case** MATRIX\_MODE:

matrixRoutine();

**break**;

**case** WHITE\_LAMP\_MODE:

whiteLamp();

**break**;

**default**:

whiteLamp();

**break**;

}

FastLED.show();//Запись массива индикации в Ленту

}

}

**Листинг 3.12.** Подпрограмма индикации выбранного режима*.*

Остановимся более подробно на первом режиме индикации – режим разноцветных вспышек. В проекте подпрограмма этого режима называется *sparklesRoutine.* Этот режим полностью построен на системной для программной среды Arduino функции генератора псевдослучайных чисел. Активация генератора псевдослучайных чисел была описана выше. Вызов функции активации производится внутри функции *setup*. Ниже представлен листинг подпрограммы.

**void** sparklesRoutine() {

**for** (byte i = 0; i < modes[SPARKLES\_MODE].scale; i++) {

byte x = random(0, WIDTH);

byte y = random(0, HEIGHT);

**if** (getPixColorXY(x, y) == 0)

leds[getPixelNumber(x, y)] = CHSV(random(0, 255), 255, 255);

}

fader(70);

}

**Листинг 3.13.** Подпрограмма индикации режима разноцветных вспышек*.*

Для того чтоб не загромождать листинг я удалила все комментарии из исходного кода. Описание работы подпрограммы представлено ниже.

Первая часть подпрограммы это цикл for по переменной i. Количество проходов цикла зависит от значения параметра scale описываемого режима. В программе мы обращаемся к параметру scale элемента массива режимов с индексом *SPARKLES\_MODE*. Параметр scale определяет, сколько светодиодов за один вызов функции будет включено. Внешне это выглядит как прореживание активных светодиодов. Тройное нажатие на кнопку с последующим удержанием меняет этот параметр.

В каждом проходе цикла определяются два случайных числа. Эти числа определяют координаты светодиода в двумерном массиве. В программе задается диапазон чисел для генератора случайных чисел. Для первого числа диапазон от нуля до параметра WIDTH – координата x. Для второго числа задается диапазон от нуля до параметра HEIGHT – координата y. Полученные координаты определяют светодиод ленты (матрицы) с которым мы работаем. По задумке этот режим должен выглядеть как вспыхивание светодиодов производным цветом и плавное затухание. Поэтому если светодиод, определенный в данном проходе координатами ещё не погас, то программа не меняет его значение. Если же значения всех трёх цветов выбранного светодиода равны нулю, то в него записываются новый цвет с максимальной яркостью и насыщенностью. Цвет представлен в цветовом пространстве HSV. Компонента Hue – случайное число, а компоненты Saturation и Value записывается число 255. Это и есть максимальная яркость и насыщенность. После этого вызывается функция fader. Эта функция осуществляет плавное гашение всех светодиодов ленты.

Описанный режим использует все три переменных параметра. Параметр brightness влияет на общую яркость светодиодов ленты. Параметр speed влияет на то, с какой частотой происходит вызов функции отображения режима. Частота вызова влияет на скорость затухания светодиодов и как следствие на общую скорость режима. Параметр scale определяет, сколько светодиодов зажигается за один проход. Внешне это прореживает активные светодиоды.

1. Заключение

В третьей главе был описан весь процесс разработки декоративного светильника. Работа светильника основана на ленте светодиодов RGB. В процессе разработки были изучены возможности использования сторонних библиотек в разработке программного обеспечения. Использование библиотеки работы со светодиодной намного упростило и ускорило процесс разработки светильника. Использованная библиотека позволяет работать с различными видами светодиодных лент. Для использования выбранной библиотеки осталось только правильно настроить все параметры. Этого не получилось бы без хорошей документации на выбранную библиотеку. Можно сделать вывод, что при разработке собственной библиотеки необходимо уделить особое внимание качественному описанию интерфейса библиотеки. Без этого использование библиотеки другими программистами станет просто невозможным.

В конструировании корпуса помогло использование различных доступных технических материалов. Корпус изделия был собран из пластиковой муфты, а блок питания представляет собой простое зарядное устройство от мобильного телефона. Рассеивателем света послужил лист бумаги формата А3.

Большое внимание было уделено программному обеспечению светильника. Текст программы написан с необходимыми комментариями, а названия всех функций и переменных соответствуют их функциональному назначению.

При помощи одной кнопки управления было реализовано переключение и настройка режимов, а также сохранение настроек.

ВЫВОДЫ И РЕКОМЕНДАЦИИ

В данной работе были рассмотрены компоненты и подходы к созданию устройств декоративного освещения. В работе представлены наиболее распространенные на рынке компоненты для построения устройств декоративного освещения, такие как платы Arduino, построенные на базе микроконтроллеров AVR и светодиодные ленты, состоящие из RGB светодиодов.

В ходе работы были проанализированы различные архитектуры микропроцессоров и способы организации аппаратного обеспечения для достижения лучших результатов с точки зрения экономичности и простоты разработки.

В результате работы был сконструирован декоративный светильник, управляемый микроконтроллером(AtMega32U4) на плате Arduino(Leonardo) и использующий ленты RGB светодиодов в качестве источника света. Готовый светильник оправдал ожидания – яркости достаточно для демонстрации и в дневное время, а реализованные световые эффекты позволяют оценить потенциальные возможности светодиодных лент.

На базе полученных результатов можно рекомендовать использование светодиодных лент, управляемых платами Arduino, для изготовления декоративных светильников с необычными световыми эффектами.

.

БИБЛИОГРАФИЯ

БИБЛИОГРАФИЯ

[1] Таненбаум Э., Остин Т., Архитектура компьютера. 6-е изд. - СПб.:Питер, 2017, стр. 69

[2] Arduino Commnication, https://www.tutorialspoint.com/arduino/arduino\_communication.htm, (посещено 05.06.2019)

[3] Arduino Leonardo, <http://arduino.ru/Hardware/ArduinoBoardLeonardo>

[4] Arduino Program Structure, https://www.tutorialspoint.com/arduino/arduino\_program\_structure.htm

[5] Arduino I/O Functions, https://www.tutorialspoint.com/arduino/arduino\_io\_functions.htm

[6] Arduino Pulse Width Modulation, https://www.tutorialspoint.com/arduino/arduino\_pulse\_width\_modulation.htm

[7] Arduino Interrupts, https://www.tutorialspoint.com/arduino/arduino\_interrupts.htm

[8] Восприятие цвета мозгом www.tikkurila.ru/dlya\_professionalov/tsveta/vvedenie\_v\_teoriyu\_tsveta/vospriyatie\_tsveta\_mozgom

[9] Цветовая модель RGB https://ru.wikipedia.org/wiki/RGB

[10] О цветовых пространствах, https://habr.com/ru/post/181580/

[11] Цветовая модель CMYK,https://ru.wikipedia.org/wiki/CMYK

[12] Language Reference https://www.arduino.cc/reference/en

[13] Writing a Library for Arduino, https://www.arduino.cc/en/Hacking/LibraryTutorial

[14] Цветовая температура, <http://www.ylati.ru/czvetovaya_temperatura.html>

[15] Адресная светодиодная лента и её подключение к Arduin, https://ledjournal.info/spravochnik/adresnaja-svetodiodnaja-lenta.html

Приложение 1. Основной код программного обеспечения

|  |  |
| --- | --- |
|  |  |

**void** setup() {

//Настройка ленты

FastLED.addLeds<WS2812B, LED\_PIN, COLOR\_ORDER>(leds, NUM\_LEDS);

FastLED.setBrightness(BRIGHTNESS);

//Установка максимального тока блока питания

**if** (CURRENT\_LIMIT > 0)

FastLED.setMaxPowerInVoltsAndMilliamps(5, CURRENT\_LIMIT);

FastLED.clear();//Установка всех значений в ноль

FastLED.show();//запись значений в светодиодную ленту для отображения

//

//настройка параметров кнопки

touch.setStepTimeout(100);

touch.setClickTimeout(500);//интервал между кликами

//

//Функция RandomSeed() инициализирует генератор псевдослучайных чисел.

randomSeed(analogRead(0));

//

//Инициализация последовательного порта

Serial.begin(9600);

  #ifdef DEBUG\_PRINT

delay(1000);

Serial.println();

Serial.println("LED lamp start");

  #endif

//

//Чтение Настроек из EEPROM

// если было сохранение настроек, то восстанавливаем их из EEPROM

**if** (EEPROM.read(0) == 102) {

currentMode = EEPROM.read(1);

**for** (byte x = 0; x < MODE\_AMOUNT; x++) {

modes[x].brightness = EEPROM.read(x \* 3 + 11); // (2-10 байт - резерв)

modes[x].speed = EEPROM.read(x \* 3 + 12);

modes[x].scale = EEPROM.read(x \* 3 + 13);

}

}

}

**void** loop() {

effectsTick();//Менеджер эффектов

buttonTick();//Отработка нажатий на кнопку

}

Приложение 2. Менеджер эффектов

|  |  |
| --- | --- |
|  |  |

**void** effectsTick() {

uint32\_t CurrentTime = millis() - effTimer;

**if** (ONflag && (CurrentTime >= modes[currentMode].speed)) {

effTimer = millis();

**switch** (currentMode) {

**case** SPARKLES\_MODE:

sparklesRoutine();

**break**;

**case** FILL\_MODE:

fillRoutine();

**break**;

**case** RNBOW\_HORION\_MODE:

rainbowHorizontal();

**break**;

**case** RNBOW\_VERTIC\_MODE:

rainbowVertical();

**break**;

**case** COLOR\_S\_MODE:

colorsRoutine();

**break**;

**case** COLOR\_MODE:

colorRoutine();

**break**;

**case** SNOWFALL\_MODE:

snowRoutine();

**break**;

**case** MATRIX\_MODE:

matrixRoutine();

**break**;

**case** WHITE\_LAMP\_MODE:

whiteLamp();

**break**;

**default**:

whiteLamp();

**break**;

}

FastLED.show();

}

}

ДЕКЛАРАЦИЯ ОБ ОТВЕТСТВЕННОСТИ

Нижеподписавшаяся, заявляю под личную ответственность, что материалы, представленные в магистерской диссертации, являются результатом личных научных исследований и разработок. Осознаю, что в противном случае, буду нести ответственность в соответствии с действующим законодательством.

Касимова Марина

25.05.2020