МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ
«Южно-Уральский государственный университет»
(национальный исследовательский университет)

Факультет «Высшая школа электроники и компьютерных наук»
Кафедра «Инфокоммуникационные технологии»

ПРОЕКТ ПРОВЕРЕН
Рецензент
_____
_____/_____/
«____»_____20__ г.

ДОПУСТИТЬ К ЗАЩИТЕ
Заведующий кафедрой
_____
_____/Даровских С.Н./
«____»_____20__ г.

«Прибор автоматизированного контроля»

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА
К ВЫПУСКНОМУ КВАЛИФИКАЦИОННОМУ ПРОЕКТУ
ЮУрГУ-Д.11.05.01.2017.283.00 ПЗ ВКП

Консультанты
Безопасность жизнедеятельности
_____/И.С. Окраинская/
«____»_____ 20__ г.

Руководитель проекта
Инженер – конструктор
ФГУП «ПСЗ»
_____/И.К. Куликов/
«____»_____ 20__ г.

Организационно - экономический раздел
_____/Р.Ш. Закиров/
«____»_____ 20__ г.

Автор проекта
студент группы КЭ-665
_____/М.В. Платонов/
«____»_____ 20__ г.

Нормоконтролер
_____/В.Д. Спицына/
«____»_____ 20__ г.

Челябинск 2017

АННОТАЦИЯ

Платонов М.В. Разработка прибора автоматизированного контроля. – Челябинск: ЮУрГУ, КЭ, 2017.Пояснительная записка выполнена на 149 листах, содержит 25 иллюстраций и 14 таблиц. Список литературы содержит 12 наименований. Графический материал состоит из 4 листов формата А1.

В данном дипломном проекте представлена разработка прибора автоматизированного контроля временных параметров электрических цепей, на 20 каналов. Основное назначение данного прибора – регистрация кратковременного размыкания(замыкания) цепи на пределе 1 мкс.

Цель работы – разработка функциональной, электрической принципиальной схемы прибора, электрической структурной схемы и сборочный чертежа, в соответствии с требованиями, выдвинутыми в техническом задании.

В процессе проектирования была разработана функциональная схема, выдвинуты требования к узлам прибора, произведен выбор элементной базы и его обоснование, разработана схема электрическая принципиальная и его конструкция. Устройство выполнено на современной элементной базе с применением передовых технологий в области электроники и схемотехники. Рассмотрены и учтены требования по безопасности жизнедеятельности, произведено экономическое обоснование дипломного проекта.

					ЮУрГУ-Д.11.05.01.2016.532.00 ПЗ		
Изм.	Лист	№ докум.	Подпись	Дата			
Разраб.		Платонов М.В.			Разработка прибора автоматизированного контроля	Лит.	Лист 3 / Листов
Провер.		Куликов И.К.				Д	
Н. Контр.		Спицина В.Д.				ЮУрГУ кафедра ИКТ	
Утверд.		Даровских С.Н.					

ОГЛАВЛЕНИЕ

ВВЕДЕНИЕ

В данной работе представлен процесс разработки прибора автоматизированного контроля временных параметров электрических цепей, на 20 каналов, предназначенные для регистрация кратковременного размыкания(замыкания) электрической цепи изделия. Прибор предназначен для лабораторных испытаний.

Данный класс испытаний имеет большое значение для оценки качества изготовления изделия. Это связано с тем, что многие устройства во время своей работы испытывают сильные механические воздействия, такие как вибрация, удары, линейные ускорения и т.д. Если эти негативные факторы проявляются во время работы изделия, то они могут существенно повлиять не только на механическую целостность изделия, но и на качество передачи электрических сигналов в цепях устройства. Особенно сильно это воздействие будет проявляться в местах установки соединительных элементов (например, жгутов). В местах установки соединителей, во время действия механических нагрузок, могут происходить кратковременные размыкания, или же замыкания, тех цепей, где это не предусмотрено. Эти явления могут привести к тому, что на электронную аппаратуру будут поступать не верные данные от других блоков устройства. В итоге, устройство либо выйдет из строя, либо не сможет корректно выполнять свои функции.

Весь процесс создания нового устройство можно разделить на несколько частей:

а) сбор необходимой информации и изучение предметной области;

б) разработка структурной схемы разрабатываемого устройства;

в) подбор соответствующих электронных компонентов;

г) разработка схемы для ПЛИС;

д) разработка программной части для ПЛИС и МК;

е) разработка принципиальной электрической схемы;

ж) разработка алгоритма работы прибора;

з) создание пояснительной записки и необходимого набора чертежей.

В дальнейшем все эти этапы будут рассмотрены в данной работе.

Данный прибор будет внедрен взамен действующему сейчас прибору ЕС6.

Недостатками существующего оборудования являются:

а) высокая трудоемкость изготовления и настройки;

б) имеет три канала на регистрацию размыкания;

в) имеет три канала на регистрацию замыкания.

Целью разработки нового прибора является устранение указанных недостатков.

Снижение трудоемкости изготовления и настройки можно обеспечить цифровой технологией обработки принимаемых сигналов и применением современных микросхем.

1 Анализ технического задания

Разрабатываемый прибор должен регистрировать размыкание(замыкание) электрической цепи по 20 каналам. Каналы должны работать как на регистрацию размыкания , так и на регистрацию замыкания. Время срабатывания на пределе одной микросекунды для регистрации размыкания не более 0,5 микросекунд. В свою очередь, время срабатывания на пределе одной микросекунды для регистрации замыкания не более 1 микросекунды.

Прибор, с целью снижение трудоемкости изготовления, будет выполнен на современной элементной базе. Элементная база будет включать в себя цифровую обработку сигнала, поступаемого с испытуемой схемы. Так как прибор будет регистрировать 20 каналов одновременно необходимо подключить функциональные блоки, которые будут оповещать о срабатывании канала.

Аппаратная часть разрабатываемого устройства будет включать в себя:

а) кнопка включение/отключения прибора;

б) 20 входных каналов;

в) разъем питания на 24 вольта;

г) один LCD дисплей, на котором будет выводиться номер срабатываемого канала на размыкание(замыкание);

а) динамик, будет срабатывать при размыкании(замыкании) канала и будет работать в течении 20 секунд;

б) светодиод, который будет срабатывать одновременно с высвечиванием номера канала на дисплее, над каждым каналом будет располагаться свой светодиод.

в) кнопка сброса, будет отвечать за сброс мигание светодиода и звукового оповещения.

Прибор должен считать одну микросекунду как на регистрацию размыкания , так и на регистрацию замыкания. Для этого расчета будет применяться программируемая логическая интегральная схема (ПЛИС).

Далее, чтобы связать функциональные блоки, которые будут оповещать об срабатывании канала, необходимо применение микроконтроллера. Микроконтроллер (МК) будет принимать рассчитанные, с помощью логики, сигналы, которые будут свидетельствовать о размыкании(замыкании) канала на испытуемой схеме. Затем МК передает сигналы на срабатывание функциональных блоков.

Так как разрабатываемый прибор работает в лабораторных условиях необходимо быстро и точно рассчитать время в ПЛИС и передать рассчитанный сигнал на микроконтроллер. Существуют параллельная передача данных и последовательная передача данных со всевозможными интерфейсами. Последовательная передача данных обычно используется на более дальних расстояниях и значительно отстает по скорости передачи по сравнению с параллельной передачей данных. Ввиду быстродействия и маленького расстояния передачи, при разработки данного устройства, используется параллельная передача данных.

Программная часть для ПЛИС должна выполнять следущие действия:

а) рассчитывать одну микросекунду для размыкания контактов схемы;

б) рассчитывать одну микросекунду для замыкания контактов схемы;

в) передать рассчитанные параметры в МК для оповещения срабатывания канала.

Программная часть для МК должна:

а) вывести на экран номер срабатываемого канала;

б) произвести звуковое оповещение;

в) зажечь и мигать светодиод своего канала, который сработал;

г) отвечать за кнопку отключения светодиода.

Прибор будет потреблять питание постоянного напряжения 24 вольта. Для того чтобы избежать эффект «ложного срабатывания» прибора, поставим фильтр нижних частот.

Структурная схема прибора изображена на рисунке 1.1:

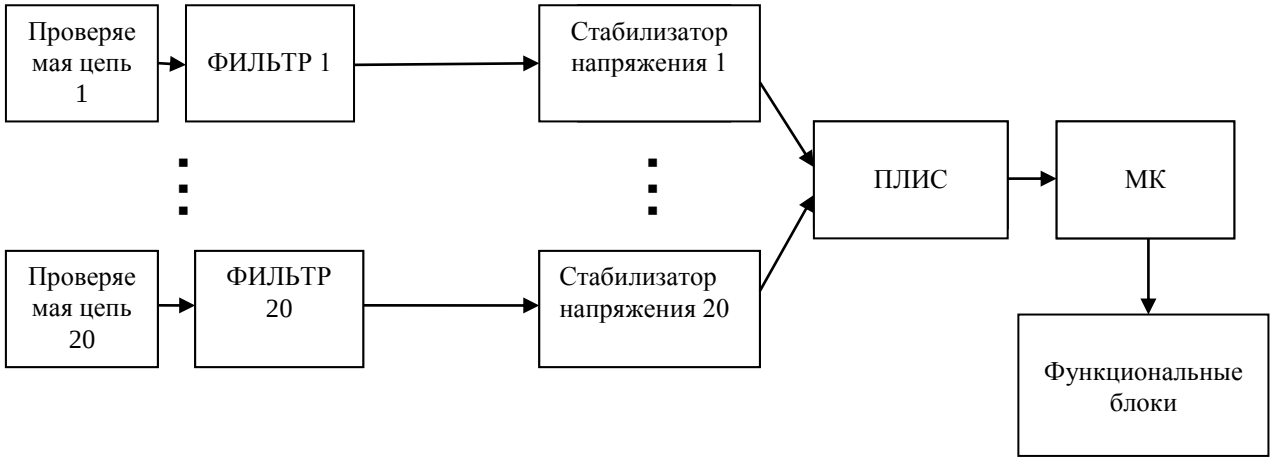


Рисунок 1.1 – Структурная схема прибора

2 Подбор элементной базы

2.1 Выбор ПЛИС

Программируемая логическая интегральная схема (ПЛИС, англ. Programmable logic device, PLD) — электронный компонент, используемый для создания цифровых интегральных схем. В отличие от обычных цифровых микросхем, логика работы ПЛИС не определяется при изготовлении, а задаётся посредством программирования (проектирования). Для программирования используются программатор и IDE (отладочная среда), позволяющие задать желаемую структуру цифрового устройства в виде принципиальной электрической схемы или программы на специальных языках описания аппаратуры: Verilog, VHDL, AHDL и др.

Достаточно много компаний в мире занято производством цифровых устройств на основе ПЛИС и использованием их в своих системах. Такие как: Xilinx, Altera, Lattice Semiconductor, Actel, Atmel, и т.д..Наиболее известные производители ПЛИС — Altera и Xilinx.

Компания Xilinx является одним из лидеров в области производства ПЛИС-микросхем. На данный момент у этой компании существует несколько серий выпускаемой аппаратуры для разного рода вычислений.

Virtex. Высокопроизводительные ПЛИС на основе FPGA, призванные заменить специализированные интегральные схемы при решениях различных ресурсоемких задач.

Spartan. Более дешевые и менее производительные ПЛИС FPGA, разработанные для использования в устройствах, рассчитанных на большие тиражи и невысокую стоимость комплектующих.

CoolRunner и XC9500. Серии ПЛИС типа CPLD, предназначенных для использования в различных портативных устройствах - мобильных телефонах, GPS-навигаторах, КПК и т.д. Для микросхем данного типа главными критериями является минимизация размеров и потребляемой мощности.

Компания Altera является основным конкурентом компании Xilinx, причем по всем основным направлениям. Главное из них - это производство ПЛИС как типа FPGA, так и типа CPLD. В мае 2008 года Altera представила новое семейство из серии Stratix высокопроизводительных микросхем типа FPGA - Stratix IV, работающих на 40-нм архитектуре. Для менее ресурсоемких задач компания Altera предлагает серию ПЛИС FPGA Cyclone, а в качестве компромисса между производительными Stratix и недорогими Cyclone - серию Arria. Для мобильных устройств выпускается серия Max на основе ПЛИС типа CPLD.

Сферы использования ПЛИС являются электронные устройства для решения следующих задач:

- цифровая обработка сигналов;
- работа видео- и аудиотехники;
- скоростная передача цифровой информации;
- криптографические и аналогичные им системы информационной безопасности и пр.

Основным критерием в выборе ПЛИС является наличие достаточного количества ножек для реализации одновременной работы на 20 каналов. Так же немаловажным критерием является стоимость на микросхему.

Архитектура семейства MAX II объединяющая различные узлы (массив программируемой логики, пользовательскую Flash-память, встроенный RC-генератор, встроенный линейный регулятор напряжения) позволяет снизить общую стоимость разрабатываемой системы.

Ввиду актуальности и низкой стоимости выберем ПЛИС семейства MAXII EPM240100C5. Буквы EPM означают семейство. 240 — количество логических блоков (LE). Бывают 240/570/1270/2210. После цифр могут быть буквы G или Z, означающие малопотребляющие, 1,8 В версии. Если их нет, то это обычный, 2,5…3,3 В версии. T100 означает TQFP-100 корпус, шаг ножек 0,5 мм. Кроме TQFP-100, есть ещё 144 пиновый вариант, для чипов с количеством блоков 570 или 1270. Буква C/I/A означает температурный диапазон, C — самый обычный, 0…85 градусов. Последняя цифра важна. Она определяет класс скорости. От трех, самый быстрый, до восьми, самый медленный. Классы 3…5 относятся к обычным версиям, 6…8 — к малопотребляющим. Класс скорости определяет тайминги и частоты счётчиков. Например, 16-битный счетчик может работать на 304 МГц на третьем классе и только на 201 МГц на пятом. В конце может быть буква N, означающая отсутствие свинца.

Характеристи ПЛИС:

Питание от 2,5 до 4,6 вольт (максимум, 3,0…3,6 рекомендуемое). До 304 МГц частоты. 8192 бит памяти. 80 пинов I/O, 25 мА максимальный ток, входное напряжение на пин от -0,7 В до VCC+0,7 В, Прошивается через JTAG. 240 логических блоков.

2.2 Выбор микроконтроллера

Микроконтроллер (англ. Micro Controller Unit, MCU) — микросхема, предназначенная для управления электронными устройствами. Типичный микроконтроллер сочетает на одном кристалле функции процессора и периферийных устройств, содержит ОЗУ и (или) ПЗУ. По сути, это однокристальный компьютер, способный выполнять относительно простые задачи. Отличается от микропроцессора интегрированными в микросхему устройствами ввода-вывода, таймерами и другими периферийными устройствами.

Микроконтроллеры объединяются в семейства. К одному семейству относят изделия, имеющие одинаковое ядро – совокупность таких понятий, как система команд, циклограмма работы ЦП, организация памяти программ и памяти данных, система прерываний и базовый набор периферийных устройств. Отличия между различными представителями одного семейства заключаются, в основном, в составе периферийных устройств и объеме памяти программ или данных. Наиболее важная особенность семейства – программная совместимость на уровне двоичного кода всех входящих него МК.

Известные семейства МК

PIC (Microchip)

PIC – микроконтроллеры Гарвардской архитектуры, производимые американской компанией MicrochipTechnologyInc. Название PIC является сокращением от PeripheralInterfaceController, что означает «периферийный интерфейсный контроллер».

В основу концепции PIC, единую для всех выпускаемых семейств, была положена RISC-архитектура (ReducedInstructionSetComputer – архитектура с сокращенным набором команд) с системой простых однословных команд, применение встроенной памяти программ и данных и малое энергопотребление.

В номенклатуре MicrochipTechnologyInc. представлен широкий спектр восьми, 16-и и 32-битных микроконтроллеров и цифровых сигнальных контроллеров под маркой PIC. Отличительной особенностью PIC-контроллеров является хорошая преемственность различных семейств. Это и программная совместимость (единая бесплатная среда разработки MPLAB IDE), и совместимость по выводам, по периферии, по напряжениям питания, по средствам разработки, по библиотекам и стекам наиболее популярных коммуникационных протоколов. Номенклатура насчитывает более 500 различных контроллеров со всевозможными вариациями периферии, памяти, количеством выводов, производительностью, диапазонами питания и температуры и т. д.

AVR (Atmel)

Концепция новых скоростных микроконтроллеров была разработана группой разработчиков исследовательского центра ATMEL в Норвегии, инициалы которых затем сформировали марку **AVR** (**A**lfBogen / **V**ergardWollan / **R**iscarchitecture). Первые микроконтроллеры AVR AT90S1200 появились в середине 1997 г. и быстро снискали расположение потребителей.

AVR-архитектура, на основе которой построены микроконтроллеры семейства AT90S, объединяет мощный гарвардский RISC-процессор с раздельным доступом к памяти программ и данных, 32 регистра общего назначения, каждый из которых может работать как регистр- аккумулятор, и развитую систему команд фиксированной 16-бит длины. Большинство команд выполняются за один машинный такт с одновременным исполнением текущей и выборкой следующей команды, что обеспечивает производительность до 1 MIPS на каждый МГц тактовой частоты.

Достоинства:

а) высокий показатель быстродействие/энергопотребление;

б) удобные режимы программирования;

в) широкая номенклатура;

г) доступность программно-аппаратных средств поддержки;

д) высокая нагрузочная способность выходов.

ARM (ARM Limited)

Архитектура **ARM** (AdvancedRISCMachine, AcornRISCMachine, усовершенствованная RISC-машина) – семейство лицензируемых 32-битных и 64-битных микропроцессорных ядер разработки компании **ARMLimited**. Компания занимается исключительно разработкой ядер и инструментов для них (компиляторы, средства отладки и т. п.), зарабатывая на лицензировании архитектуры сторонним производителям.

Данные процессоры имеют низкое энергопотребление, поэтому находят широкое применение во встраиваемых системах и доминируют на рынке мобильных устройств, для которых важно низкое энергопотребление.

Среди лицензиатов: AnalogDevices, Atmel, Xilinx, Altera, CirrusLogic (англ.), Intel (до 27 июня 2006 года), Marvell (англ.), NXP, STMicroelectronics, Samsung, MediaTek, MStar, Qualcomm, SonyEricsson, TexasInstruments, nVidia, Freescale, Миландр.

По результатам анализа техничего задания нам нужно подобрать микроконтроллер, который будет отвечать за управление функциональными блоками. Основными критериями подборки микроконтроллера являются:

– число разрядов,не менее восьми;

– наличие 20 входов, предназначенные для приема сигналов с ПЛИС;

– 20 выводов выделенные под зажигание светодиодов над каждым каналом, в случае срабатывания;

– один вывод под динамик;

– наличие таймер/счетчика для реализации звукового сигнала в устройстве;

– питание от 2..5 вольт.

С задачами, возложенными на контроллер, могут справиться современные восьми разрядные контроллеры. Остановимся подробней на фирмах ATMEL и Microchip. Контроллеры PIC фирмы Microchip обладают меньшей производительностью по сравнению с контроллерами AVR фирмы ATMEL. Контроллер PIC выполняет одну инструкцию за четыре такта, в то время как контроллер AVR выполняет большинство инструкций за 1..2 такта. Еще одним преимуществом контроллеров AVR является их меньшая стоимость, при одинаковых возможностях с контроллерами PIC. Поэтому остановимся на контроллерах AVR фирмы ATMEL. Наиболее точно вышеперечисленным требованиям удовлетворяет микроконтроллер Atmega 128.

Выбранный микроконтроллер имеет следующие технические характеристики:

– 32-х восьми разрядных регистров общего назначения, регистры управления встроенной периферией;

– Производительность до 16 миллионов операций в секунду при тактовой частоте 16 МГц;

– Износостойкость 128-ми кбайт внутрисистемно перепрограммируемой флэш-памяти: 1000 циклов запись/стирание;

– Опциональный загрузочный сектор с отдельной программируемой защитой;

– Встроенное статическое ОЗУ емкостью 4 кбайт;

– Опциональная возможность адресации внешней памяти размером до 64 кбайт;

– Два восьми разрядного. таймера-счетчика с раздельными предделителями и режимами сравнения;

– Два расширенных 16-ти разрядного таймера-счетчика с отдельными предделителями, режимами сравнения и режимами захвата;

– Счетчик реального времени с отдельным генератором;

Ввод-вывод и корпуса;

– 53 программируемые линии ввода-вывода;

– 64 вывода корпус TQFP.

– Рабочие напряжения

– 2,7 ...5,5 В для ATmega128L;

– 4,5…5,5 В для ATmega128;

2.3 Блок питания

По техническому заданию устройство получает напряжение питания постоянного тока равное 24 В. Это напряжение необходимо для питания

испутуемой схемы и для питания самого прибора. ПЛИС требует постоянного напряжения питания 3,3 В. Микроконтроллер постоянного напряжения питания от 2,7 до 5,5 В. Поэтому необходим преобразующий постоянное напряжение 24 В в напряжение 3,3 В. Для предотвращения попадания повышенного тока в цепь питания устройства необходимо применить предохранитель.

Для преобразования напряжения одного уровня в напряжение другого уровня часто применяют импульсные преобразователи напряжения с использованием индуктивных накопителей энергии. Такие преобразователи отличаются высоким КПД, иногда достигающим 95%, и обладают возможностью получения повышенного, пониженного или инвертированного выходного напряжения. В соответствии с этим известно три типа схем преобразователей: понижающие , повышающие и инвертирующие . Общими для всех этих видов преобразователей являются пять элементов: источник питания, ключевой коммутирующий элемент, индуктивный накопитель энергии (катушка индуктивно-сти, дроссель), блокировочный диод и конденсатор фильтра, включенный параллельно сопротивлению нагрузки. Включение этих пяти элементов в различных сочетаниях позволяет реализовать любой из трех типов импульсных преобразователей. Регулирование уровня выходного напряжения преобразователя осуществляется изменением ширины импульсов, управляющих работой ключевого коммутирующего элемента и, соответственно, запасаемой в индуктивном накопителе энергии. Стабилизация выходного напряжения реализуется путем использования обратной связи: при изменении выходного напряжения происходит автоматическое изменение ширины импульсов. Для стабилизации выходного напряжения импульсных стабилизаторов любого типа могут быть использованы обычные «линейные» стабилизаторы, но они имеют низкий КПД. В этой связи гораздо логичнее для стабилизации выходного напряжения импульсных преобразователей использовать импульсные же стабилизаторы напряжения, тем более, что осуществить такую стабилизацию совсем несложно. Импульсные стабилизаторы напряжения, в свою очередь,

подразделяются на стабилизаторы с широтно-импульсной моду-ляцией и на стабилизаторы с частотно-импульсной модуляцией. В первых из них изменяется длительность управляющих импульсов при неизменной частоте их следования. Во вторых, напротив, изменяется частота управляющих импульсов при их неизменной длительности. Встречаются импульсные стабилизаторы и со смешанным регулированием.

Итак, для питания МК , ПЛИС и lcd дисплея нам понадобиться понижающий импульсный преобразователь напряжения. Существует масса специализированных микросхем, например LM2574, LM2594, LM267x, LT1073, L4971, ST1S03, AS1333, ST1S03, ST1S06, ST1S09, ST1S10. Они существуют в разных корпусах для разных выходных напряжений и токов. Стоимость таких микросхем около 3 евро, однако нам требуется надежное и не дорогое решение. Рассмотрим микросхему MC34063 . MC34063 очень распространена, из ее характеристик можно отметить напряжение от 3 до 40 вольт, Максимальный ток 1,5А, частота преобразования 100KHz.

Для уменьшения пульсации на выходе преобразователя применим фильт нижних частот.

3 Разработка принципиальной схемы, программная часть ПЛИС

3.1 Задача, выбор САПРов

Из анализа технического задания программируемая логическая интегральная схема должна:

– рассчитывать одну микросекунду для размыкания контактов схемы;

– рассчитывать одну микросекунду для замыкания контактов схемы;

– передать рассчитанные параметры в МК для оповещения срабатывания канала.

Для того чтобы рассчитать одну микросекунду на регистрацию размыкание и замыкание контактов электрической схемы необходимо построить цифровую схему на выходе которой будут временные диаграммы показанные на рисунке 3.1:

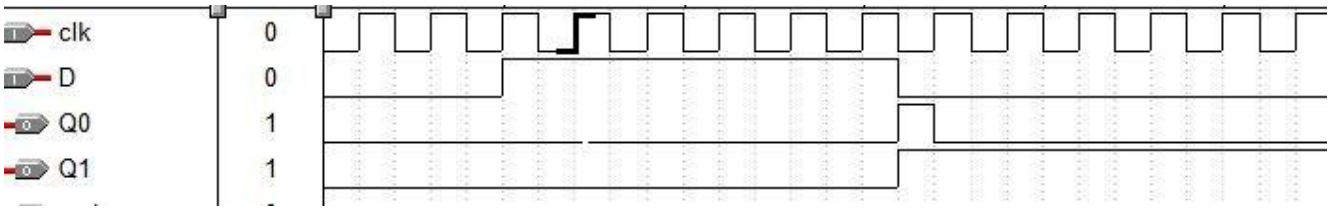


Рисунок 3.1 – Работа ПЛИС на регистрацию размыкания

На диаграмме используются следующие обозначения:

clk – тактовый сигнал;

D – входной сигнал;

Q0 – промежуточный выходной сигнал;

Q1 – результирующий выходной сигнал, передающийся в последствии на входы МК.

Идея заключается в том, что входной сигнал D сравнивается с тактовыйм сигналом, если в длительность сигнала D вошли пять переходов из нуля в еденицу тактового сигнала, то считается что сработал канал на регистрацию

замыкания. Промежуточный выходной сигнал показывает что сработал канал. В свою очередь Q0 преобразовывается в Q1 , для того чтобы микроконтроллер сделал последующие действия, а именно высветился на экран номер срабатываемого канала, загорелся светодиод над входом канала сопровождающий звуковым сигналом.

Для регистрации канала на размыкания используется аналогичный принцип, показанный на рисунке 3.2, обозначение входных и выходных сигналов от регистрации на замыкание ничем не отличается.

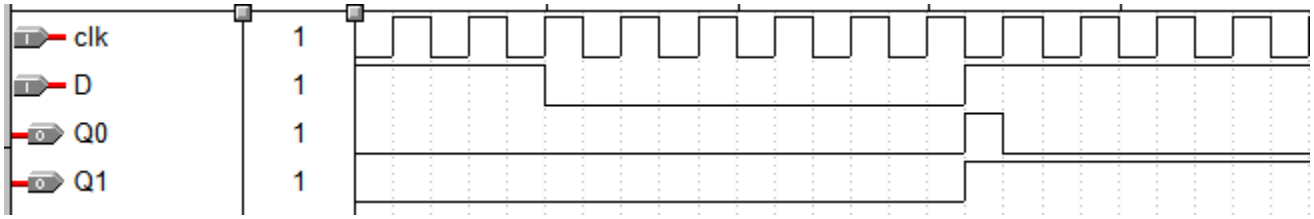


Рисунок 3.2 – Работа ПЛИС на регистрацию замыкания

Разработаем схему для регистрации размыкания для ПЛИС для одного канала.

Для реализации данной идеи воспользуемся сапром MAX+PLUSII.

САПР MAX+PLUS II предназначена для разработки цифровых устройств и предоставляет для решения этой задачи следующие инструменты:

а) средства описания проекта;
б) средства компиляции проекта;
в) средства верификации проекта;
г) средства программирования ПЛИС.

3.2 Разработка принципиальной схемы ПЛИС

Итак нам нужно построить цифровую схему , которая будет удовлетворять временной диаграмме показанной на рисунках 3.1, 3.2. Для этого обратимся к теории построения цифровых схем. Основными элементами с помощью которых строиться схема являются логические элементы такие как:

- OR;
- AND;
- NOT;
- NAND;
- NOR;
- XOR;
- XNOR;

Наиболее наглядно логическая функция характеризуется таблицей, называемой таблицей истинности. Талица истинности содержит всевозможные комбинации входных переменных X и соответствующие им значения функции Y. Основные элементы , их работа и предназначение расписаны в таблице 3.1

Количество комбинаций составляет 2^n, где n – число аргументов.

Таблица 3.1 – Вентили

<table>
<tr><th>Название элемента</th><th>Назначение элемента</th><th>Таблица истинности элемента</th></tr>
<tr><td>OR(ИЛИ)</td><td>Выполняет операцию логического сложения (дизъюнкцию). Обозначают эту операцию символом ∨ или знаком сложения (+). Функция Y=X1∨X2 принимает значение логической 1, если хотя бы одна переменная равна 1.</td><td>
<table>
<tr><th>X1</th><th>X2</th><th>Y</th></tr>
<tr><td>0</td><td>0</td><td>0</td></tr>
<tr><td>0</td><td>1</td><td>1</td></tr>
<tr><td>1</td><td>0</td><td>1</td></tr>
<tr><td>1</td><td>1</td><td>1</td></tr>
</table>
</td></tr>
</table>

Продолжение таблицы 3.1

<table>
<tr><th>Название элемента</th><th>Назначение элемента</th><th>Таблица истинности элемента</th></tr>
<tr><td>AND (И)</td><td>Выполняет операцию логического умножения (конъюнкцию). Такую операцию обозначают символом /\ или значком умножения (·). Если все входные переменные равны 1, то и функция Y=X1·X2 принимает значение логической 1. Если хотя бы одна переменная равна 0, то и выходная функция будет равна 0.</td><td><table><tr><th>X1</th><th>X2</th><th>Y</th></tr><tr><td>0</td><td>0</td><td>0</td></tr><tr><td>0</td><td>1</td><td>0</td></tr><tr><td>1</td><td>0</td><td>0</td></tr><tr><td>1</td><td>1</td><td>1</td></tr></table></td></tr>
<tr><td>NOT (НЕ)</td><td>выполняет операцию логического отрицания (инверсию). При логическом отрицании функция Y принимает значение противоположное входной переменной X. Эту операцию обозначают ¬.</td><td><table><tr><th>X</th><th>Y</th></tr><tr><td>1</td><td>0</td></tr><tr><td>0</td><td>1</td></tr></table></td></tr>
<tr><td>NAND (И НЕ)</td><td>Выполняет операцию логического умножения над входными переменными, а затем инвертирует полученный результат и выдаёт его на выход.</td><td><table><tr><th>X1</th><th>X2</th><th>Y</th></tr><tr><td>0</td><td>0</td><td>1</td></tr><tr><td>0</td><td>1</td><td>1</td></tr><tr><td>1</td><td>0</td><td>1</td></tr><tr><td>1</td><td>1</td><td>0</td></tr></table></td></tr>
<tr><td>NOR (ИЛИ НЕ)</td><td>Выполняет операцию логического сложения над входными переменными, а затем инвертирует полученный результат и выдаёт его на выход.</td><td><table><tr><th>X1</th><th>X2</th><th>Y</th></tr><tr><td>0</td><td>0</td><td>1</td></tr><tr><td>0</td><td>1</td><td>0</td></tr><tr><td>1</td><td>0</td><td>0</td></tr><tr><td>1</td><td>1</td><td>0</td></tr></table></td></tr>
<tr><td>XOR</td><td>Выполняет операцию логическое исключающее ИЛИ</td><td><table><tr><th>X1</th><th>X2</th><th>Y</th></tr><tr><td>0</td><td>0</td><td>0</td></tr><tr><td>0</td><td>1</td><td>1</td></tr><tr><td>1</td><td>0</td><td>1</td></tr><tr><td>1</td><td>1</td><td>0</td></tr></table></td></tr>
<tr><td>XNOR</td><td>Логическая операция исключающее И</td><td><table><tr><th>X1</th><th>X2</th><th>Y</th></tr><tr><td>0</td><td>0</td><td>1</td></tr><tr><td>0</td><td>1</td><td>0</td></tr><tr><td>1</td><td>0</td><td>0</td></tr><tr><td>1</td><td>1</td><td>1</td></tr></table></td></tr>
</table>

Международный стандарт обозначения элементов показан на рисунке 3.3:

Рисунок 3.3 – Международное обозначение логических элементов

Вернемся к рисунку 3.1 Для того чтобы входной сигнал D сравнивался с тактовыйм сигналом, т.е. если в длительность сигнала D войдут пять переходов из нуля в еденицу тактового сигнала, то считается что сработал канал на регистрацию замыкания. Нам необходимо воспользоваться счетчиком, который будет считать эти переходы из нуля в еденицу. Счетчики у нас строятся на D и T - триггерах. Обратимся к понятиям счетчик и триггер.

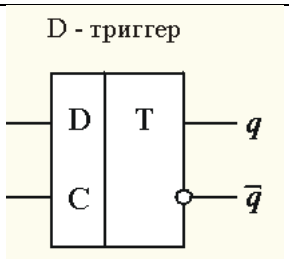
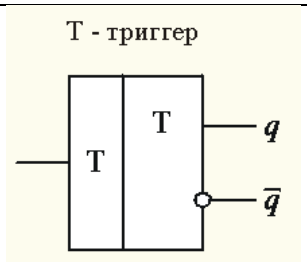
Счётчики используются для построения схем таймеров или для выборки инструкций из ПЗУ в микропроцессорах. Они могут использоваться как делители частоты в управляемых генераторах частоты (синтезаторах).

Триггеры (триггерные системы) - класс блоков, обладающих способностью длительно находиться в одном из двух устойчивых состояний и чередовать их под воздействием внешних сигналов. Каждое состояние триггера легко распознаётся по выходным сигналам.

Отличительной особенностью двоичного триггера как функционального устройства является свойство запоминания двоичной информации. Под памятью триггера подразумевают способность оставаться в одном из двух состояний и после прекращения действия переключающего сигнала. Приняв одно из состояний за «1», а другое за «0», можно считать, что триггер хранит (помнит) один разряд числа, записанного в двоичном коде.

Отличительной особенностью аналогового триггера (многоразрядного регистра) как функционального устройства является свойство запоминания аналоговой информации. Под памятью триггера подразумевают способность запоминать своё последнее состояние и после прекращения действия переключающего сигнала. Таким образом, можно считать, что триггер хранит (помнит) аналоговую величину.

Таблица 3.2 – Триггеры

Название элемента	Обозначние	Таблица истинности
D – триггер		D | Состояние: 0 | 1; 0 | 0 | 0; 1 | 1 | 1
T – триггер		T | Состояние: 0 | 1; 0 | 0 | 1; 1 | 1 | 0

Чтобы построить схему на логике необходимо восмпользоваться картами Карно.Карты Карно представляют собой графический способ минимизации переключательных (булевых) функций.

Для того чтобы счетчик у нас считал до пяти, нам необходимо взять натуральный логарифм от пяти. В итоге получаем, что нам необходимо взять 3 триггра для построения схемы для ПЛИС для одного канала.

Опираясь на построенные карты Карно и на выше изложенную теорию, построили схему показанную на рисунке 3.4.

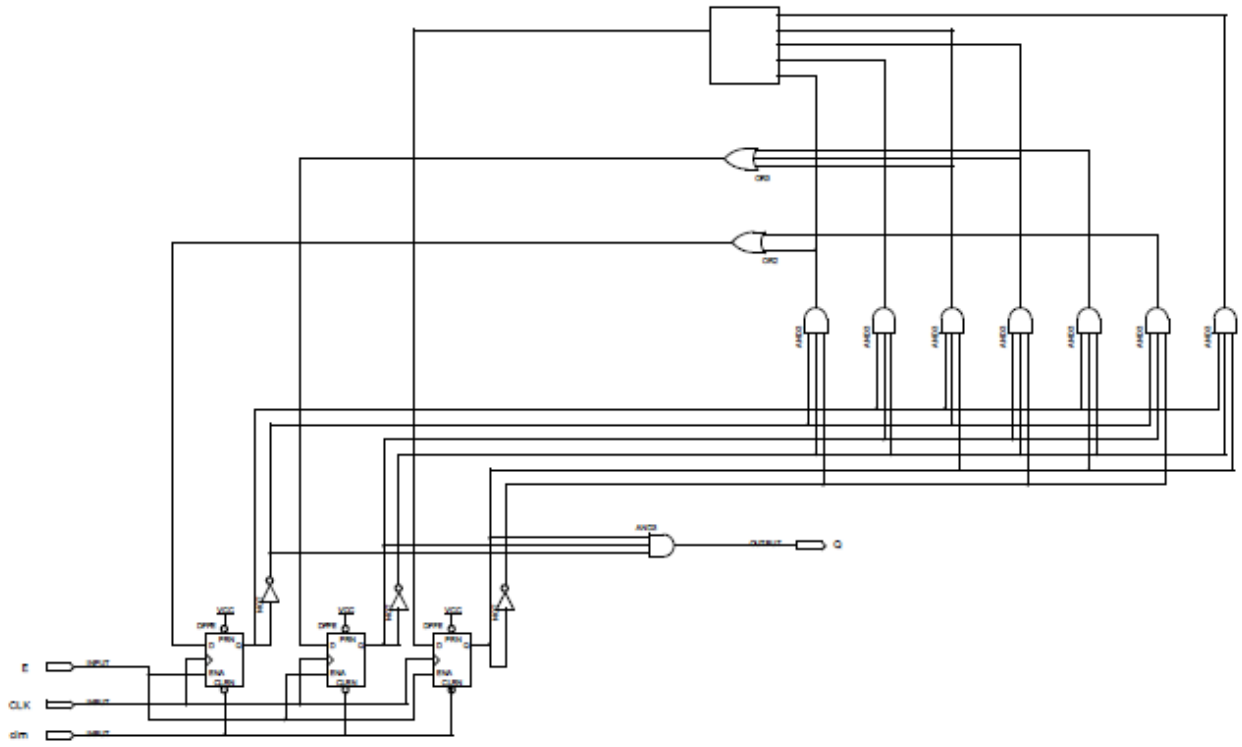


Рисунок 3.4 – Схема построенная в MAX PLUS II

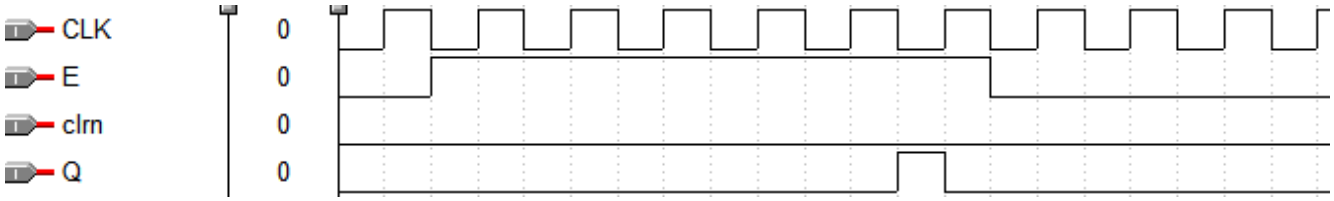


Рисунок 3.5 – Временные диаграммы работы схемы

Временные диаграммы работы схемы (Рисунок 3.5) показывают, что Карты Карно построены были верно. Схема работает правильно. На построенной схеме и временной диаграмме были использованы следующие обозначения:

CLK – тактовый сигнал;

E – входной сигнал;

Clrm – сигнал сброса;

Q – выходной сиглал.

3.3 Выбор тактовой частоты работы устройства

Выбор тактовой частоты работы устройства – это вопрос, который важно решать как с теоретической, так и с технической точки зрения. С точки зрения теории тактовая частота должна быть максимально возможной, что обеспечит максимальное быстродействие прибора. В тоже время необходимо учитывать один очень важный фактор: не существует электронных компонентов с идеальными временными характеристиками.

Работоспособность простейших компонентов электроники (резисторы, катушки индуктивности, конденсаторы, транзисторы и т.д.) либо абсолютно не зависит от тактовой частоты, либо эта зависимость не значительна. В то же время зависимость работоспособности более сложных компонентов электроники очень сильна. Это связано с тем, что в них происходят различные переходные процессы. Как следствие, сигнал, на выходе таких компонентов, можно считать правильным только по прошествии определенного времени, а именно времени переключения компонента.

Тем не менее, для выполнения своих функций, а главная из них – это регистрация размыканий, или замыканий, устройство должно работать с частотой, которая позволит выполнять эти условия.

Определить тактовую частоту можно с помощью формулы 3.3.

$$f_0 = \frac{1}{T\min}, \qquad (3.3)$$

где: f_0 – минимальная частота работы прибора;

T_{min} – минимальное время регистрируемого события.

Согласно техническому заданию разрабатываемое устройство должно регистрировать размыкания, длительность которых больше или равна 0,5 мкс, и замыкания, длительность которых больше или равна 1 мкс. Для расчета тактовой частоты необходимо выбрать минимальное из них. Это обусловлено тем, что

иначе устройство будет «пропускать» размыкания длительность которых менее 1 мкс, что не допустимо.

Поэтому для расчета тактовой частоты будет использоваться время размыкания. Подставив это значение в формулу 3.3, получим:

$$f_0 = \frac{1}{0{,}5 \cdot 10^{-6}} = 2 \cdot 10^6 \text{ Гц}$$

Минимальная частота работы прибора должна составлять 2 МГц. Но в процессе разработки устройства было решено увеличить тактовую частоту до 10 МГц. Это сделано для того, чтоб оценивать группы импульсов. Такая оценка необходима прежде всего потому, что если частота будет 2 МГц, то устройство может пропустить какое – либо событие в любом из портов. Это связано с тем, что размыкание или замыкание может произойти между тактовыми импульсами. Именно поэтому было решено увеличить тактовую частоту до 10 МГц. Это изменение позволит оценивать правильность 10 импульсов.

3.4 Разработка програмной части ПЛИС

В силу трудоемкости реализации данной схемы была выдвинута идея реализовать данную схему с помощью какого нибудь языка программирования. Существуют несколько языков, с помощью которых можно описать цифровую схему. Основные языки это AHDL, Verilog, VHDL. Остановимся поподробней на каждом языке и опишем основные плюсы и минусы каждого языка.

Язык описания аппаратуры AHDL (AlteraHardwareDescriptionLanguage) разработан фирмой Altera и предназначен для описания комбинационных и последовательностных логических устройств, групповых операций, цифровых автоматов (statemachine) и таблиц истинности с учётом архитектурных особенностей ПЛИС фирмы Altera. Он полностью интегрируется с системой автоматизированного проектирования ПЛИС MAX+PLUS II. Файлы описания аппаратуры, написанные на языке AHDL, имеют расширение .tdf (Textdesignfile). Для создания TDF-файла можно использовать как текстовый редактор системы

MAX+PLUS II, так и любой другой. Проект, выполненный в виде TDF-файла, компилируется, отлаживается и используется для формирования файла программирования или загрузки ПЛИС фирмы Altera.

Verilog, Verilog HDL (Verilog Hardware Description Language)— это язык описания аппаратуры, используемый для описания и моделирования электронных систем. Verilog HDL, не следует путать с VHDL (конкурирующий язык), наиболее часто используется в проектировании, верификации и реализации (например, в виде СБИС) аналоговых, цифровых и смешанных электронных систем на различных уровнях абстракции. Разработчики Verilog сделали его синтаксис очень похожим на синтаксис языка C, что упрощает его освоение. Verilog имеет препроцессор, очень похожий на препроцессор языка C, и основные управляющие конструкции «if», «while» также подобны одноимённым конструкциям языка C. Соглашения по форматированию вывода также очень похожи (см. printf). Следует отметить, что описание аппаратуры, написанное на языке Verilog (как и на других HDL-языках) принято называть программами, но в отличие от общепринятого понятия программы как последовательности инструкций, здесь программа задает структуру системы.

VHDL(Very high speed integrated circuits Hardware Description Language)-данный язык предназначен для описания проектируемых систем на схемотехническом уровне проектирования и замены классического подхода к схемотехническому проектированию на уровне отдельных элементов. Язык позволяет описывать цифровые ситемы на алгоритмическом уровне. При помощи специального программного обеспечения описание на языке VHDL преобразовывается в схему на уровне простейших элементов цифровой электроники. Описание на языке применятся VHDL как при проектировании заказных СБИС так и при проектировании цифровых систем на базе специальных устройств ПЛИС.

Поскольку язык описания VHDL все больше набирает популярность среди разработчиков , я буду разрабатывать программу на этом языке. Так же язык VHDL имеет множество преимуществ такие как:

Гибкость. Проект, описанный на VHDL, может быть легко настроен под конкретные задачи потребителя. Универсальный язык. VHDL - общепринятый язык для всех основных фирм - изготовителей микросхем ПЛИС, ПЛМ, заказных СБИС как стандартный язык для задания сложных проектов. Проектирование с VHDL - устойчивая тенденция в инженерной технологии. Существуют компиляторы, транслирующие VHDL-программы в эквивалентные им Verilog - программы.

Моделирование с учетом задержек. Фирмы - изготовители микросхем в своих САПР обеспечивают генерацию моделей результатов размещения и трассировки, описанных на VHDL.

Стандартное подключение блоков. Конструкции языка, такие как entity, port map, configuration, обеспечивают надежную и быструю стыковку блоков, разработанных разными фирмами и разработчиками, в различном сочетании.

Стандартное тестирование. На всех этапах разработки выполняется тестирование по одной методике одними и теми же тестами.

VHDL - стандарт будущего. Все новые САПР основаны на технологии трансляции описания ВУ на языке описания аппаратуры. Использование VHDL - гарантия того, что через 5 и 10 лет найдется САПР, поддерживающая старые разработки.

Структура программы

Процесс программирования на VHDL можно описать следующим образом:

Шаг 1: Включение в код используемых библиотек;

Шаг 2: Описание точек входа и точек выхода устройства;

Шаг 3: Описание точек входа и выхода элементов входящих в устройство;

Шаг 4: Описание архитектуры элементов входящих в устройство;

Шаг 5: Описание архитектуры всего устройства.

Для реализации схемы я буду использовать систему проектировая Quartus II. Программное обеспечение Quartus II предоставляет полный цикл для создания высокопроизводительных систем на кристалле. Quartus II объединяет в себе проектирование, синтез, размещение элементов, трассировку соединений и верификацию, связь с системами проектирования других производителей. Разработка систем на кристалле требует от разработчиков эффективной командной работы. Изменения в одной части проекта должно иметь минимальное влияние на других членов команды. Программное обеспечение Quartus II - это наиболее комплексная среда для разработки систем на кристалле SOPC, доступная в настоящее время. Quartus II включает в себя блочный метод разработки LogicLock. На рисунке 3.6 показаны временные диаграммы работы программы на языке VHDL.

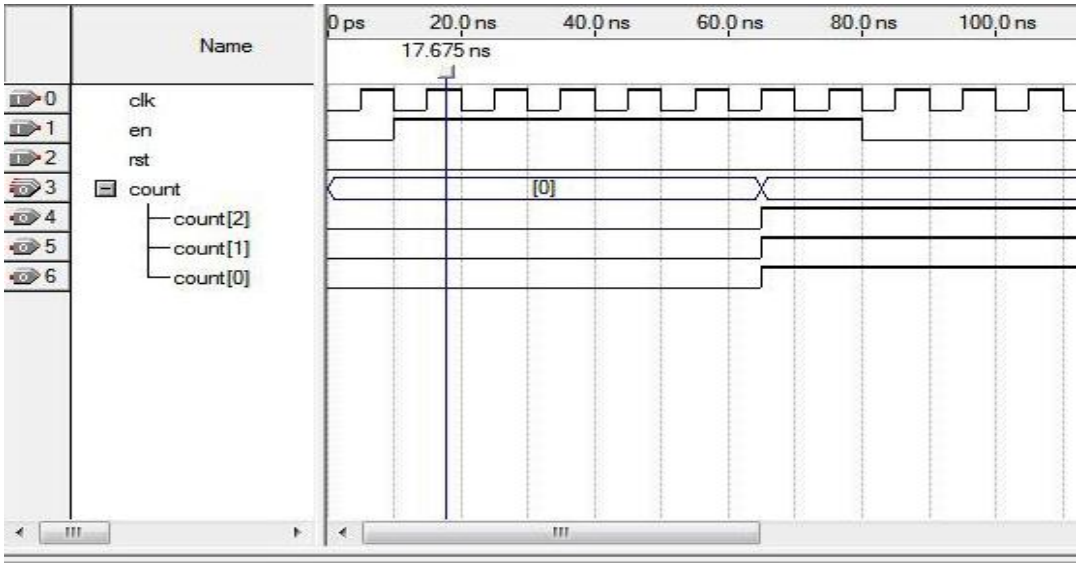


Рисунок 3.6 – Временные диаграммы работы программы

Обозначения используемые на временных диаграммах:

– Clk – тактовый сигнал;

– En – входной сигнал;

– Rst – сигнал сброса;

– Count – выходной сигнал, который в последствии передается на вход МК.

На рисунке 3.6 показаны временные диаграммы работы программы на языке VHDL. По временным диаграммам можно понять что схема работает именно так как нам и надо. Т.е. на вход у нас приходит какой то сигнал, на временной диаграмме это сигнал en, сравнивается с тактовым сигналом clk, высчитывается пять переходох и на выходе у нас появляется сигнал, который говорит нам о том, что канал сработал.

На рисунке 3.7 показан результат работы программы в случае , если входной сигнал при сравнении с тактовым окаждется меньше чем положено, т.е. если входной сигнал при сравнении с тактовым будет меньше, не будут входить 5 переходов из 0 в 1 , то на выходе ничего не будет, канал не сработает.

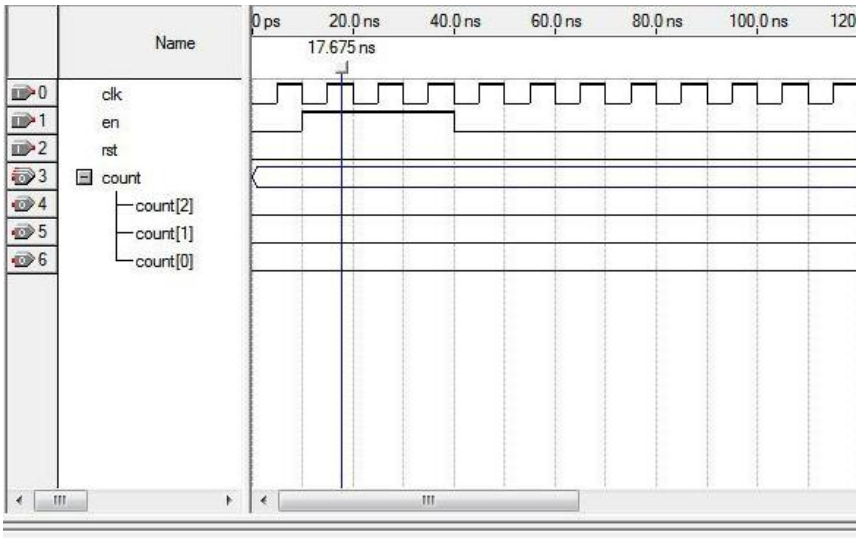


Рисунок 3.7 – Временные диаграммы работы программы

На рисунке 3.8 показана схема просимулированная программой Quartus. Для одного канала.

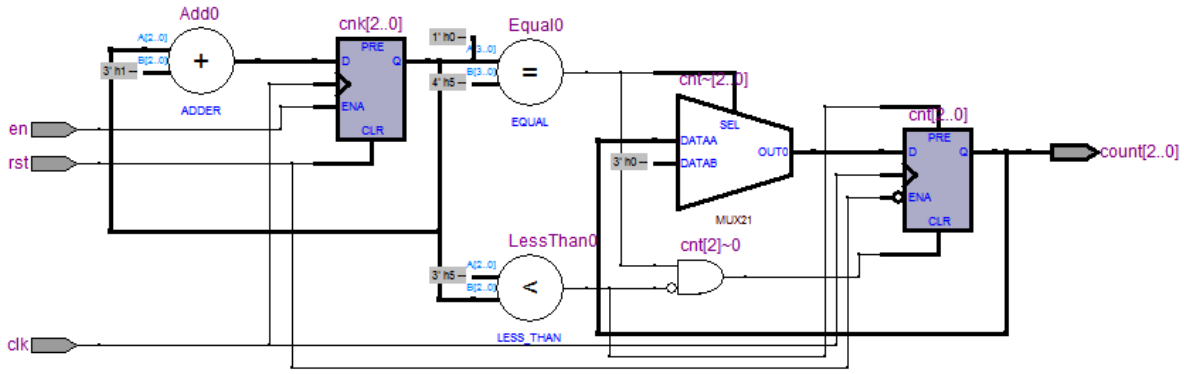


Рисунок 3.8 – Схема канала на регистрацию размыкания

Для того чтобы канал работал на регистрацию замыкания нам необходимо добавить элемент XOR, в результате у нас получиться окончательная схема для работы канала на регистрацию замыкани и размыкания на один канал показанная на рисунке 3.9.

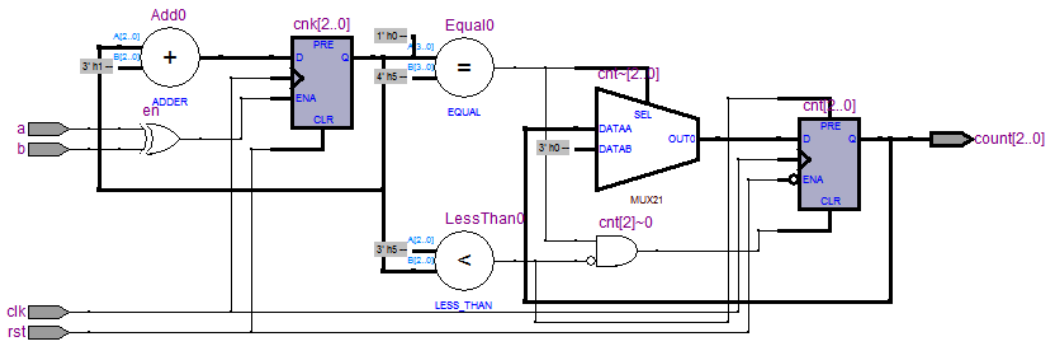


Рисунок 3.9 – Схема канала на регистрацию размыкания/замыкания

На рисунке 3.10 показана работа канала на регистрацию замыкания, принцип работы канала регистрацию замыкания точно такой же как на регистрацию размыкания. Т.е. счетчик считает 5 переходов из 0 в 1 у тактового сигнала. Сравнивает с входным сигналом, после , на выходе, появляется сигнал, передающийся на микроконтроллер. Сигнал b всегда равен еденице.

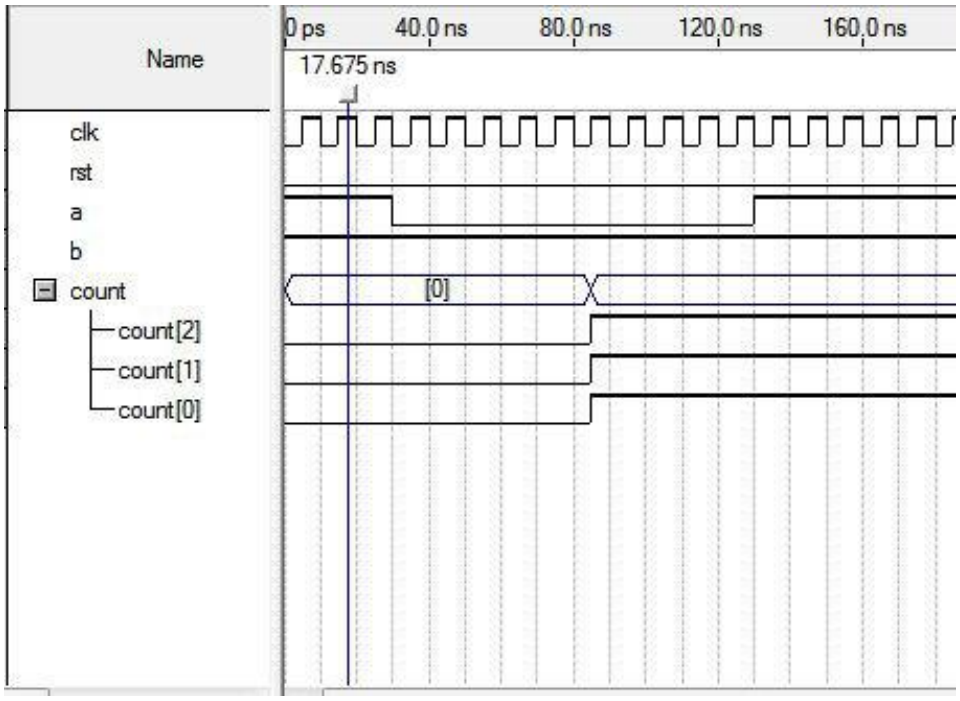


Рисунок 3.10 – Результат работы схемы на регистрацию замыкания.

4 Разработка программы для МК

4.1 Задача, выбор САПРов

По результату подбора элементов мы выбрали микроконтроллер ATMEGA 128 фирмы ATMEL. Напомню, что нам, по результатам анализа технического задания необходимо было, чтобы МК принимал обработанную информацию с ПЛИС и выводил эту информацию с помощью функциональных блоков. Т.е. с ПЛИС у нас приходит сигнал, МК в свою очередь должен принять сигнал, и просигнализировать, что канал сработал. Должен загореться светодиод, на экране появиться номер сработанного канала с сопровождением звукового сигнала длительностью не более 20 секунд.

Для реализации данной части проекта необходим пакет программ с помощью которых можно было бы написать программу для МК, и проссимлировать работу МК в реальных условиях. Воспользуемся САПРами:

- AtmelStudio 6.0, для написания программы для МК;
- Proteus 7 Professional, для симуляции работы МК.

AtmelStudio 6 является интегрированной средой для разработки и отладки Atmel ARM Cortex-M и Atmel AVR приложений. AtmelStudio 6 IDP - лёгкий в использовании инструмент для разработки, построения и отладки приложений, написанных на языке C/C++ или ассемблере. Интегрированная среда разработки AtmelStudio 6 значительно сокращает затраты на создание новых проектов, так как она бесплатна, предоставляет отладочные инструменты профессионального уровня и поставляется совместно с AtmelSoftwareFramework, который содержит 1600 примеров ARM и AVR проектов.

ProteusProfessional — пакет программ для автоматизированного проектирования электронных схем. Пакет представляет собой систему схемотехнического моделирования, базирующуюся на основе моделей электронных компонентов принятых в PSpice. Отличительной чертой пакета ProteusProfessional является возможность моделирования работы

программируемых устройств: микроконтроллеров, микропроцессоров, DSP и прочее. Дополнительно в пакет ProteusProfessional входит система проектирования печатных плат. ProteusProfessional может симулировать работу следующих микроконтроллеров: 8051, ARM7,ARM Cortex-M3, AVR, TexasInstruments, Motorola, PIC, BasicStamp. Библиотека компонентов содержит справочные данные.

Познакомимся подробней с микроконтроллерами AVR семейства Atmel.

4.2 Архитектура AVR микроконтроллера

Микроконтроллеры AVR имеют гарвардскую архитектуру (то есть программа и данные находятся в разных адресных пространствах) и систему команд, близкую к идеологии RISC. Процессор AVR имеет 32 восьми битных регистров общего назначения, объединённых в регистровый файл. В отличие от «идеального» RISC, регистры не абсолютно ортогональны:

Некоторые команды работают только с регистрами r16…r31. К ним относятся команды работающие с непосредственным операндом: ANDI/CBR, ORI/SBR, CPI, LDI, LDS(16-бит), STS(16-бит), SUBI, SBCI, а также SER и MULS;

Команды увеличивающие и уменьшающие 16-битное значение (в тех моделях, где они доступны) с непосредственным операндом (ADIW, SBIW) работают только с одной из пар r25:r24, r27:r26 (X), r29:r28 (Y), или r31:r30 (Z);

Команда копирования пары регистров (в тех моделях, где доступна) работает только с соседними регистрами начинающимися с нечётного (r1:r0, r3:r2, …, r31:r30);

Результат умножения (в тех моделях, в которых есть модуль умножения) всегда помещается в r1:r0. Также, только эта пара используется в качестве операндов для команды самопрограммирования (где доступна);

Некоторые варианты команд умножения принимают в качестве аргументов только регистры из диапазона r16…r23 (FMUL, FMULS, FMULSU, MULSU).

4.2.1 Система команд AVR микроконтроллеров

Система команд микроконтроллеров AVR весьма развита и насчитывает в различных моделях от 90 до 133 различных инструкций. Большинство команд занимает только 1 ячейку памяти (16 бит). Большинство команд выполняется за 1 такт. Всё множество команд микроконтроллеров AVR можно разбить на несколько групп:

- команды логических операций;
- команды арифметических операций и команды сдвига;
- команды операции с битами;
- команды пересылки данных;
- команды передачи управления;
- команды управления системой.

Управление периферийными устройствами осуществляется через адресное пространство данных. Для удобства существуют «сокращённые команды» IN/OUT.

4.2.2 Разновидности AVR микроконтроллеров

Стандартные семейства:

- tinyAVR (ATtinyxxx):

а) флеш-память до 16 Кб; SRAM до 512 б; EEPROM до 512 б;

б) число линий ввода-вывода 4..18 (общее количество выводов 6..32);

в) ограниченный набор периферийных устройств.

– megaAVR (ATmegaxxx):

а) флеш-память до 256 Кб; SRAM до 16 Кб; EEPROM до 4 Кб;

б) число линий ввода-вывода 23..86 (общее количество выводов 28-100);

в) аппаратный умножитель;

г) расширенная система команд и периферийных устройств.

– XMEGA AVR (ATxmegaxxx):

а) флеш-память до 384 Кб; SRAM до 32 Кб; EEPROM до 4 Кб;

б) четырёхканальный DMA-контроллер;

в) инновационная система обработки событий.

Как правило, цифры после префикса обозначают объём встроенной flash-памяти (в КБ) и модификацию контроллера. А именно, максимальная степень двойки, следующая за префиксом обозначает объём памяти, а оставшиеся цифры определяют модификацию (напр., ATmega128 — объём памяти 128 КБ; ATmega168 — объём памяти 16 КБ, модификация 8; ATtiny44 и ATtiny45 — память 4 КБ, модификации 4 и 5 соответственно).

На основе стандартных семейств выпускаются микроконтроллеры, адаптированные под конкретные задачи:

- со встроенными интерфейсами USB, CAN, контроллером LCD;
- со встроенным радиоприёмо-передатчиком — серии ATAxxxx, ATAMxxx;
- для управления электродвигателями — серия AT90PWMxxxx;
- для автомобильной электроники;
- для осветительной техники.

Кроме указанных выше семейств, ATMEL выпускает 32-разрядные микроконтроллеры семейства AVR32, которое включает в себя подсемейства

AT32UC3 (тактовая частота до 66 МГц) и AT32AP7000 (тактовая частота до 150 МГц).

4.2.3 Архитектура выбранного контроллера

Рассмотрим конкретно архитектуру выбранного микроконтроллера, а именно МК фирмы ATMEL, Atmega 128.

Ядро центрального процессорного устройства показана на рисунке 4.1:

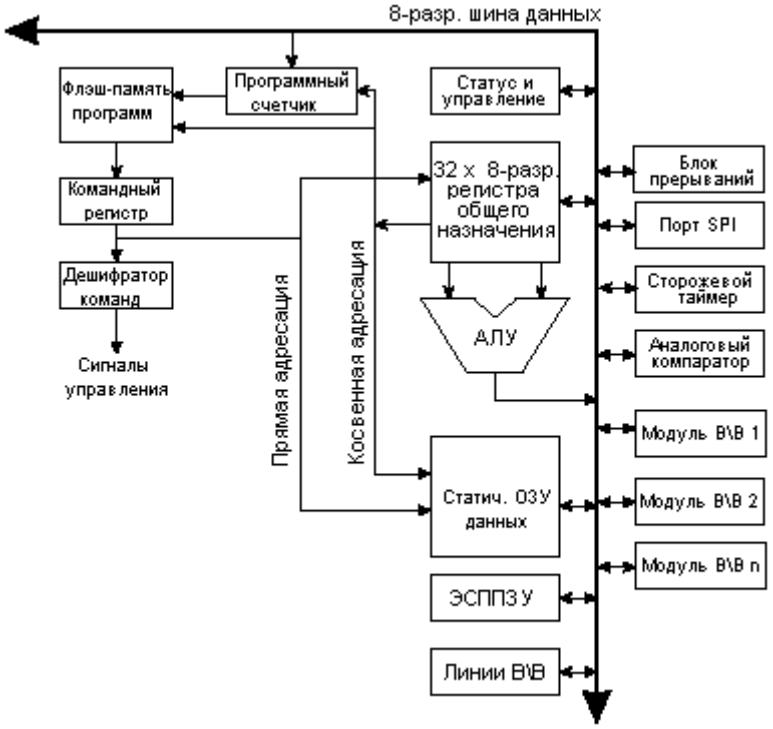


Рисунок 4.1 – Функциональная схема архитектуры МК

Ядро состоит, в первую очередь, из памяти программ и Арифметико-логического устройства(АЛУ) , блока управления (на диаграмме не показан) и программного счетчик. Также есть тактовый генератор, задающий импульсы относительно которых работают блоки микроконтроллера.

При старте микроконтроллера значение программного счетчика равно 0000 — это адрес первой команды в нашей памяти. Микроконтроллер считывает оттуда два байта (код команды и ее аргументы) и отдает на выполнение в декодер команд . Следующий шаг идет в зависмости от команды. Если это просто команда работы с какими-либо действиями, то они будут выполнены, а на следующем такте значение программного счетчика будет увеличено. Из

следующей пары ячеек памяти будут взяты еще два байта команды, и также отправлены на выполнение. Команда перехода выполняется следующим образом: в Программный счетчик загружается адрес указанный в команде (абсолютный переход) или его значение увеличивается не на 1, а на столько сколько нужно и на следующем такте микроконтроллер возьмет команду уже с нового адреса. Декодер команд считывает команду и выдает ее логике блока управления, который уже включает все остальные блоки, говоря им делать нужные действия в нужном порядке.

Вся математика и обработка делается посредством АЛУ. Это, своего рода, калькулятор. Он может складывать, вычитать, сравнивать, сдвигать разными способами, делить и умножать.

В качестве промежуточных операндов используются 32 ячейки — Оперативные регистры общего назначения РОН. Доступ к этим ячейкам самый быстрый, а число операций с их содержимым наиболее богатое.

ОЗУ

Кроме 32 регистров в микроконтроллере есть оперативная память. Оперативная память это несколько сотен ячеек памяти. От 64 байт до 4килобайт, в зависимости от модели. В этих ячейках могут храниться любые данные, а доступ к ним осуществляется через команды Load и Store. То есть нельзя взять, например, и прибавить к ячейке в памяти, скажем, единицу. Нам сначала сделать операцию Load из ОЗУ в РОН, потом в регистре прибавить нашу единицу и операцией Store сохранить ее обратно в память.

EEPROM

Долговременная память. Память которая не пропадает после выключения питания. Если Flash может содержать только код и константы, а писать в нее при выполнении ничего нельзя , то в ЕЕПРОМ можно сколько угодно писать и читать. Но в качестве оперативной памяти ее не используют. Дело в том, что цикл записи в EEPROM длится очень долго — миллисекунды. Чтение тоже не

быстрое. Число циклов перезаписи всего 100 000, что не очень много в масштабах работы оперативной памяти. ЕЕПРОМ используется для сохранения различных настроек, предустановок, собранных данных и так далее, что может потребоваться после включения птания и в основном на чтение.

Периферия

Порты ввода вывода —ножки взаимодействия контроллера с внешним миром. Именно порты обеспечивают управление с другими элементами схемы.

UART/USART приемопередатчик — последовательный порт. Работает по ассинхронному протоколу. Подходит для связи с компьютером и другими контроллерами.

Таймеры/счетчики — задача таймеров отсчитывать время. Сказал ему отсчитать 100 тактов процессора — он приступит и как досчитает подаст сигнал. Им же можно подсчитывать длительность входных сигналов, подсчитывать число входных импульсов.Подробное описание функций таймера есть в даташите

АЦП — аналоговый вход. Позволяет взять и замерить аналоговый сигнал. АЦП это своеобразный вольтметр.

SPI — еще один последовательный протокол, похожа на IIIC, но не позволяет организовывать сети. Работает только в режиме Мастер-Ведомый.

Аналоговый Компаратор — еще один аналоговый интерфейс. Но, в отличии от АЦП, он не замеряет, а сравнивает два аналоговых сигнала, выдавая результат A>B или A<B в двоичном виде.

PWM — ШИМ генератор. С помощью ШИМ генератора легко задать аналоговый сигнал. Например, менять яркость свечения светодиода или скорость вращения двигателя. Да мало ли куда его применить можно. Число каналов ШИМ разное от контроллера к контроллеру.

Взаимодействие ядра с периферией

Ядро одно на всех, периферия разная. Общение между ними происходит через память. Т.е. у периферии есть свои ячейки памяти — регистры периферии. У каждого периферийного устройства их не по одной штуки. В этих регистрах находятся биты конфигурации. В зависимости от того как эти биты выставлены в таком режиме и работает периферийное устройство. В эти же регистры нужно записывать данные которые мы хотим выдать, например, по последовательному порту, или считывать данные которые обработал АЦП. Для работы с периферией есть специальные команды IN и OUT для чтения из периферии в регистр РОН и записи из регистра РОН в периферию соответственно.

4.3 Дисплей, написание библиотеки

ЖК дисплей на основе микроконтроллера HD44780 является наиболее часто используемым в электроники. Мы можем его встретить в кофейных автоматах, часах, копирах, принтерах, роутерах и т.п. Также данный дисплей используется в LCD шилдах для Arduino. ЖК дисплей представляет из себя модуль, состоящий из микроконтроллера HD44780 разработанный фирмой Hitachi и непосредственно самим ЖК дисплеем. Микроконтроллер принимает команды и обрисовывает соответствующие символы на ЖК дисплее. Существует огромное количество разновидностей данного ЖК модуля, он может быть 1,2, четырех строчный с различным числом символов на строке, с подсветкой или без, с различным цветом подсветки и т.п. Объединяет их всех наличие микроконтроллера HD44780, зная команды которого позволит нам без проблем использовать в своих проектах ту или иную модификацию.

Для работы с дисплеями на основе HD44780 создано большое количество библиотек как на ассемблере так и на СИ, также для Arduino существует своя библиотека «LiquidCrystal».

Для изучения я решил не использовать наработки, а поработать с ним на «низком уровне», подергать его ножки самим, тем самым я получу

представление о его работе. Полученные навыки позволят мне самому написать библиотеку.

4.3.1 Подключение дисплея:

Все дисплеи на основе HD44780 имеет схожую распиновку, Распиновка дисплея показана на рисунке 4.2.

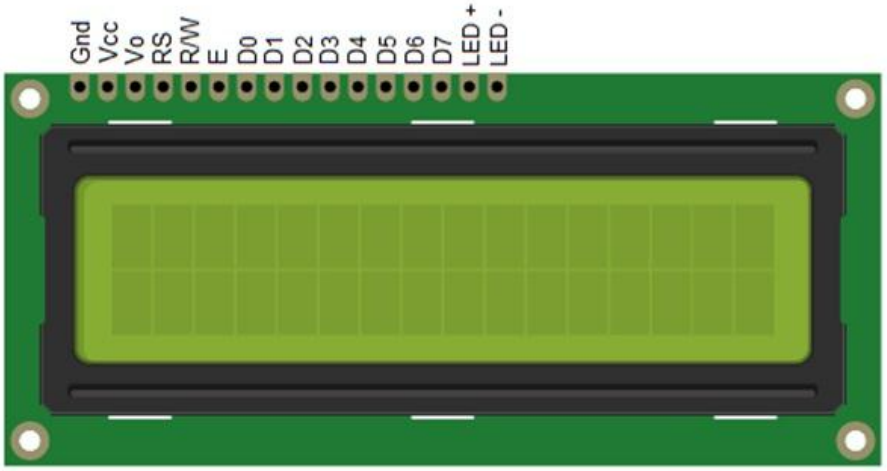


Рисунок 4.2 – Изображение дисплея

На рисунке 4.2 отображены следующие обозначения.

GND – земля (иногда пишут Vss)

Vcc – напряжение питания +5В (иногда пишут Vdd)

Vo – напряжение контрастности от 0В до +5В, данный вывод надо подключить к потенциометру, для регулировки

RS –вывод с помощью которого, дисплей определяет что в него поступает данные или команды

RW – вывод с помощью которого, дисплей определяет передавать или получать данные

E – линия синхронизации

D0 … D7 – шина команд/данных

LED + , LED — – выводы для питания подсветки

Дисплей может работать в двух режимах. В восьми разрядном (т.е. когда, для обмена информацией используются контакты от D0 до D7), данные пересылаются за один такт. В четырех разрядном (для обмена используются только контакты D4…D7), в этом случае данные пересылаются за два такта, сначала старшие четыре бита, потом младшие четыре бита.

Мы будем экономить выводы микроконтроллера и подключим наш дисплей для работы в четырех разрядном режиме как показано на рисунке 4.3.

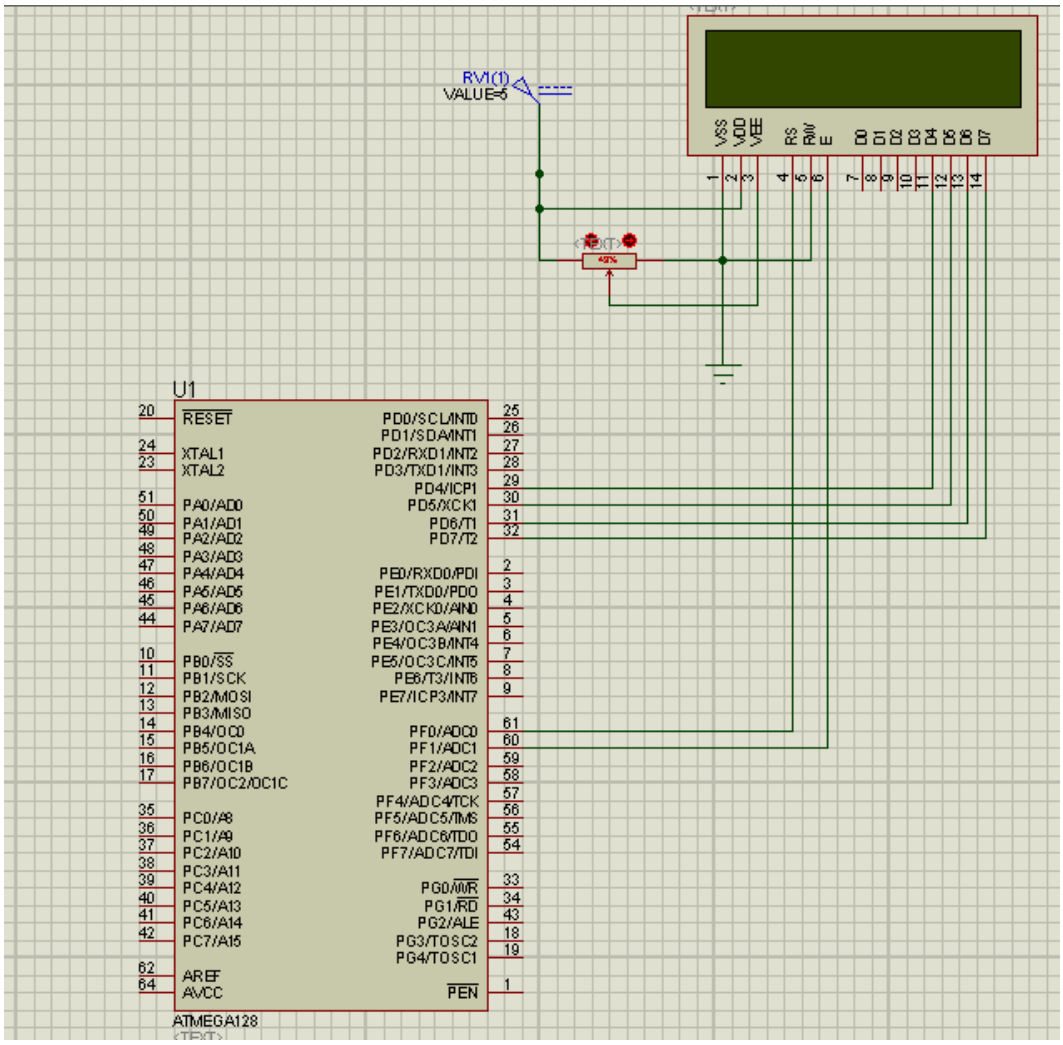


Рисунок 4.3 – Подключение дисплея

Выводы D4…D7, RS, E подключаются к любому цифровому порту микроконтроллера. Вывод RW мы использовать не будем, т.к. нам нет практической нужды принимать данные из ЖК, наша задача — только

передавать. По этому, свободные выводы RW, D0…D3 подключим к земле (по умолчанию). Заметим, что Ve подключается через реостат (можно через делитель напряжения), это подключение отвечает за контраст дисплея, в случае , если его не подключать, на экране ничего не появиться.

4.3.2 Обмен информации с LCD

Мы будем говорить о четырех разрядном режиме ЖК, поэтому для работы нам потребуется минимум 6 выводов микроконтроллера. Итак общение с ЖК происходит с помощью управляющих выводов:

RS – логическая единица ЖК принимает данные, логический ноль ЖК принимает команды;

С помощью линий данных:

D4…D7 — разряды идут от младшего к старшему

И с помощью вывода стробирующих импульсов: E

ЖК принимает информацию с помощью D4-D7, которая может быть данными (ASCII код выводимого символа) если на RS логическая единица или командой (очистить экран, перенести курсор и т.п.) если на RS логический ноль. Обмен информацией происходит по байтно, т.к. мы говорим четырех разрядном режиме, то микроконтроллер выставляет на D4…D7 логические единицы и логические нули, которые соответствуют старшему передаваемому полубайту, далее на E формируется стробирующий импульс, по заднем фронту которого ЖК считывает данные.

Далее микроконтроллер заново выставляет на D4…D7 логические единицы и логические нули, которые соответствуют младшему передаваемому полубайту и опять на E формируется стробирующий импульс, по заднем фронту которого ЖК считывает данные. После некоторой временной паузы (зависит от команды) цикл передачи байта данных или команды повторяется. Временные диаграммы работы показаны на рисунке 4.4.

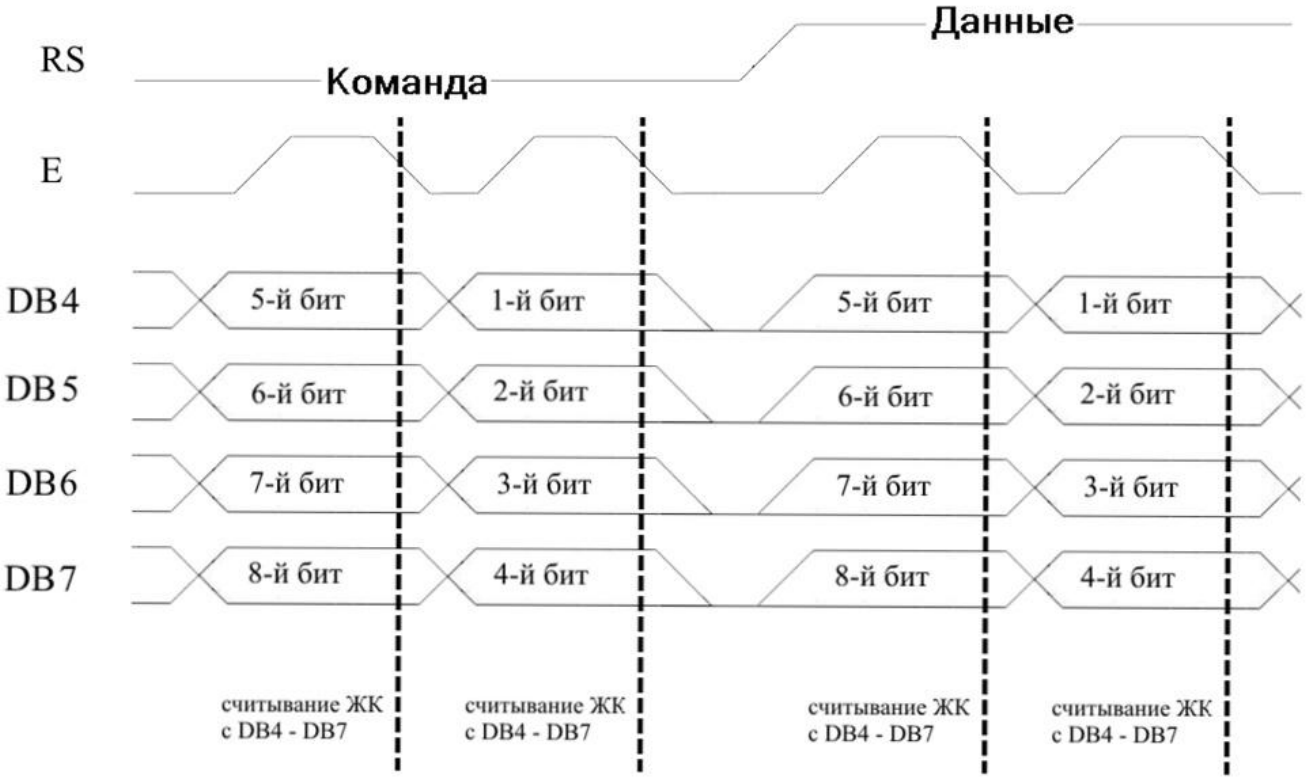


Рисунок 4.4 – Временные диаграммы работы дисплея

4.3.3 Команды дисплея

Т.к. мы не будем считывать с ЖК (вывод RW посадили на ноль), то, я рассмотрю команды для настройки и записи, которые для удобства я свел в таблицу 4.1 и пронумеровал.

Таблица 4.1 – Команды работы дисплея.

Команда	RS	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0	Макс. время (мкс)
1 Очистить дисплей и установить курсор в нулевую позицию	0	0	0	0	0	0	0	0	1	1600
2 Возврат курсора в нулевую позицию	0	0	0	0	0	0	0	1	–	1600

Таблица 4.1 –Продолжение.

Команда	RS	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0	Макс. время (мкс)
3 Направление сдвига курсора при записи следующего символа: ID = 1 сдвиг в право ID = 0 сдвиг в лево Разрешение сдвига экрана: S = 1 сдвиг разрешен S = 0 сдвиг запрещен	0	0	0	0	0	0	1	I/D	S	40
4 Включить/выключить дисплей: D = 0 дисплей вкл. D = 1 дисплей откл. C = 0 курсор виден C = 1 курсор не виден B = 0 курсор моргает C = 1 курсор не моргает	0	0	0	0	0	1	D	C	B	40
5 Двигать курсор: R/L = 0 сдвиг влево R/L = 1 сдвиг вправо	0	0	0	0	1	S/C	R/L	–	–	40
6 Устанавливаем разрядность шины дисплея: DL = 0 дисплей использует 4 х разрядную шину DL = 1 дисплей использует 8 ми разрядную шину N = 0 используют 1 ю строку N = 1 использует вторую строку F = 0 шрифт 5x10 F = 1 стандартный шрифт 5x7	0	0	0	1	DL	N	F	–	–	40
7 Установка адреса в памяти GGRAM в которую будем записывать или считывать 8 байт данных	0	0	1	ACG	ACG	ACG	ACG	ACG	ACG	40

Таблица 4.1 –Продолжение.

Команда	RS	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0	Макс. время (мкс)
8 Установка адреса в память DDRAM в которую будем записывать цифровой код символа	0	1	ADD	ADD	ADD	ADD	ADD	ADD	ADD	40
9 Запись данных в GGRAM или DDRAM (бит)		4	3	2	1					

В DDRAM памяти содержится ASCII коды символов которые в данный момент отображаются на экране, еще её называют видеопамять. Для 16-ти символьных двух строчных дисплеев её размер составляет 80 байт, т.е. по 40 байт на каждую строку. На дисплеи отображается только часть DDRAM, с 0x80 по 0x8F включительно для первой строки и с 0xC0 по 0xCF включительно для второй строки, остальную часть не видно.

Перенести черту видимости вправо или влево, можно с помощью команды №5. Для этого, нужно выставить S/C=1 и R/L в зависимости от направления, если вправо —R/L=1, если влево — R/L=0. Таким образом мы можем передвигать видимую часть по DDRAM, ширина её все равно останется равным 16-ти символам для первой строки и 16-ти символам для второй строки.

В дисплеи есть так называемый счетчик адреса или курсор, который начинает считать с 0x80 (в экране это левый верхний угол), при записи нового символа в DDRAM счетчик инкрементируется или декрементируется.

Помимо этого, с помощью команды №5 можно перемещать сам курсор, если выставить S/C=0 и R/L в зависимости от направления, если вправо —R/L=1, если влево — R/L=0.

Команда №1 служит для очистки памяти DDRAM, следовательно, всех текущих отображающихся символов и установки курсора в начальное положение, т.е. в верхний левый угол дисплея.

Команда №2 не очищает DDRAM, а только устанавливает курсор в начальное положение, т.е. в верхний левый угол дисплея.

Дисплей по умолчанию не знает, в какую сторону сдвигать курсор при записи, для этого существует команда №3. Если бит ID=1, то курсор после записи символа сдвигается вправо (инкрементируется), если ID=0 (декрементируется), то влево.

С помощью команды №4 мы можем включать и отключать отображение DDRAM памяти на дисплеи. Установив бит D=0, дисплей прекращает отображать символы, но в памяти DDRAM они остаются, при D=1 дисплей отображает содержимое DDRAM.

Команда №6 используется при инициализации ЖК, если бит DL=1, то, дисплей использует с DB0 по DB7, а при DL=0, только четыре вывода шины, т.е. с DB5 по DB7.

Бит N указывает сколько строк мы будем использовать (N=1 две строки, N=0 одну строку).

Разряд F позволяет указать размер шрифта, обычно используется 5x7 при F=1, но можно использовать и 5x10 при F=0 используя только одну строку (обычно, данный функционал не работает).

С помощью команды №8 можно указать в какую ячейку DDRAM мы будем записывать ASCII код символа. Она похожа на команду №5, т.к. в ней мы тоже управляем перемещением курсора, но только уже по адресам. Командой №8, мы можем использовать для своих нужд часть памяти DDRAM, которая не отображается на дисплеи.

Команда №9 переставляет собой просто данные. С её помощью производится запись ASCII кода символа в DDRAM память для его отображения на ЖК дисплеи. Для четырех разрядного режима, запись производится в два такта, сначала старший полубайт, потом младший полубайт.

В памяти CGROM храниться «битовое изображение» выводимых символов. На рисунке 4.6 показана таблица выводимых символов.

Рисунок 4.6 –Таблица выводимых символов

Т.е. когда мы посылаем в DDRAM код 0x53, микроконтроллер HD44780 ищет ячейку 0x53 в CGROM и отрисовывает символ в соответствие с первой и нулевой данной ячейки (не всегда микроконтроллер содержит в CGROM кириллицу, там могут быть латинские или японские символы)

Память CGRAM является частью CGROM. Главное её свойство заключается в том, что мы можем записывать в неё свои символы. Для этого нужно воспользоваться командой №7, в которой мы указываем адрес ячейки и далее командой №9 отправить 8 байт под ряд с «битовым изображением символа»

4.4 Подключение светодиода

4.4.1 Алгоритм программы:

– производим чтение порта A;
– проверяем PE0, если он равен нулю включаем алгоритм мигания;
– если PE0 равен единице выключаем алгоритм мигания и тушим светодиод;
– переходим к началу основного цикла(первый пункт);
– пишем алгоритм мигания светодиодом (зажигаем светодиод, пауза, гасим светодиод, пауза);
– переходим к началу алгоритма(первый пункт).

Проссимулируем работу экрана и мигание светодиода в программе Proteus. Кнопка инициализирует приход сигнала с микроконтроллера. При нажатии кнопки у нас начинает мигать светодиод красным цветом. Затем через пол секунды на экране появляется надпись какой канал сработал. В данном случая я проссимулировал работу первого канала. Симуляция работы дисплея и светодиода показана на рисунке 4.7.

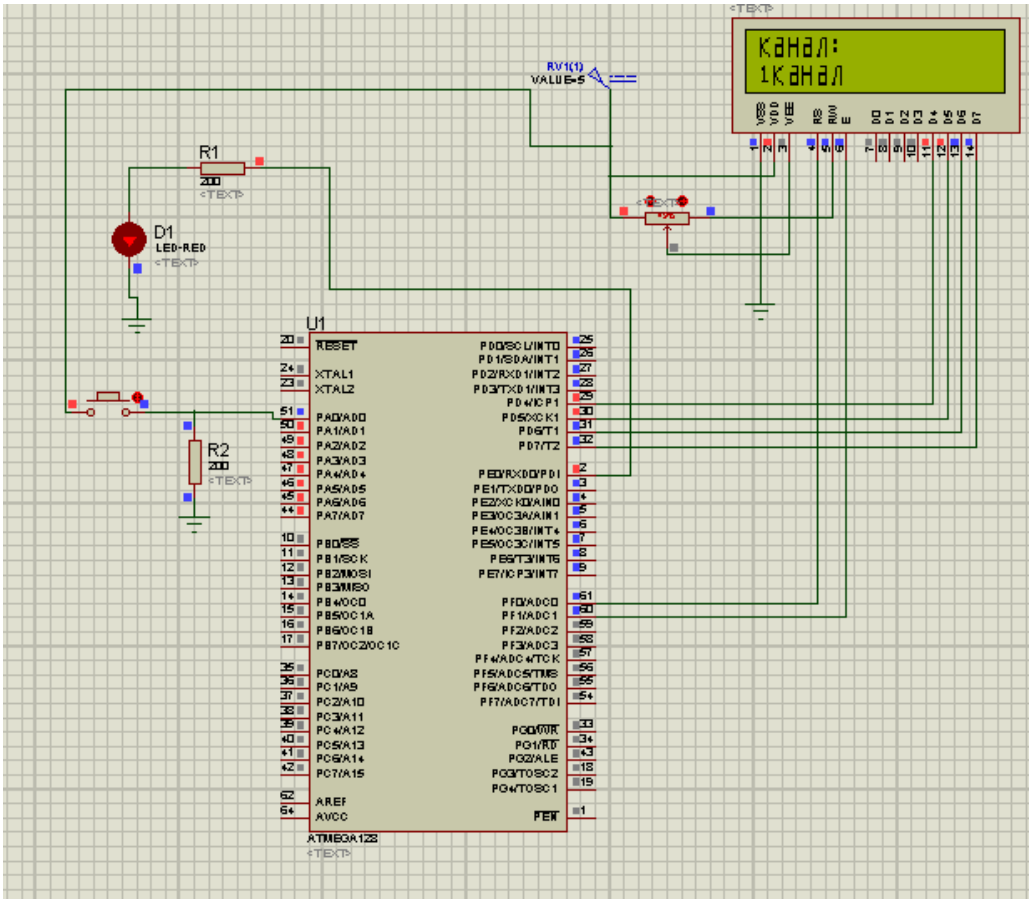


Рисунок 4.7 – Симуляция работы дисплея и светодиода

4.5. Подключение динамика

Формирование звука с помощью микроконтроллера не составляет большого труда. Достаточно взять за основу программу мигающего светодиода и подключить вместо него динамик, а в самой программе поменять константу задержки таким образом, чтобы частота на выходе повысилась до звукового диапазона. Человек может услышать звуки с частотой от 50Гц до 15кГц. Светодиод в одном из наших уроков мигает с частотой 4Гц, а если уменьшить время задержки в 1000 раз, то можно получить частоту на выходе 4Гц.

Задача заключается в то чтобы разработать программу, при помощи которой микроконтроллер Atmega128 будет воспроизводить простую однотональную мелодию.

Музыкальный ряд делится на октавы, каждая октава делится на 12 нот, это 7 основных(До, Ре, Ми, Фа, Соль, Ля, Си) и пять дополнительных(До диез, Ре диез, Фа диез, Соль диез, Ля диез). Частоты двух соседних нот отличаются

друг от друга в одинаковое количество раз, а частоты двух одноименных нот двух соседних октав отличаются в 2 раза.

Для формирования звука используем шестнадцатиразрядный таймер/счетчик1, он будет работать в режиме СТС (сброс при совпадении), для управления коэффициентом пересчета используем регистр ICR1. Режим СТС позволяет осуществлять непосредственное управление частотой сигнала. Для активизации этого режима биты WGM13 и WGM12 устанавливаем в единицу.

Для того чтобы в режиме СТС на выходе формировался периодический сигнал, необходимо настроить выход OC1A таким образом, чтобы при каждом совпадении сигнал на выходе менял свое значение на противоположное. Для этого установим бит COM1A1 в единицу и подключим к нему динамик,а также настраиваем порт PB1 на выход.

Частота сигнала на выходе OC1A определяется по формуле:

$$f_{OCnA} = \frac{f_{clk\_I/o}}{2 \times N \times (1 + OCRnA)}$$

где N - коэффициент пересчета предварительного делителя, вместо OCR1A в нашем случае ICR1.

Исходя из этой формулы расчитываем коэффициенты для ICR1, зная частоты основных нот. Так как для большинства разобраться с нотной портитурой не очень просто, названия нот я определил как в редакторе мелодий старого телефона Сименс. Коды таких мелодий еще можно найти в интернете и в дальнейшем поменять в исходном тексте. Также важно понятие как длительность тона - это время звучания одной ноты, она выражается долями от целой. В общем за проигрывание ноты у нас отвечает функция Play_LE.

Кроме нот любая мелодия содержит паузы, это промежуток времени когда ни один звук не звучит. за паузы отвечает функция Set_temp, она попросту в нужное время отключает таймер/счетчик.

Окончательная схема подключения для одного канала показа на рисунке 4.7.

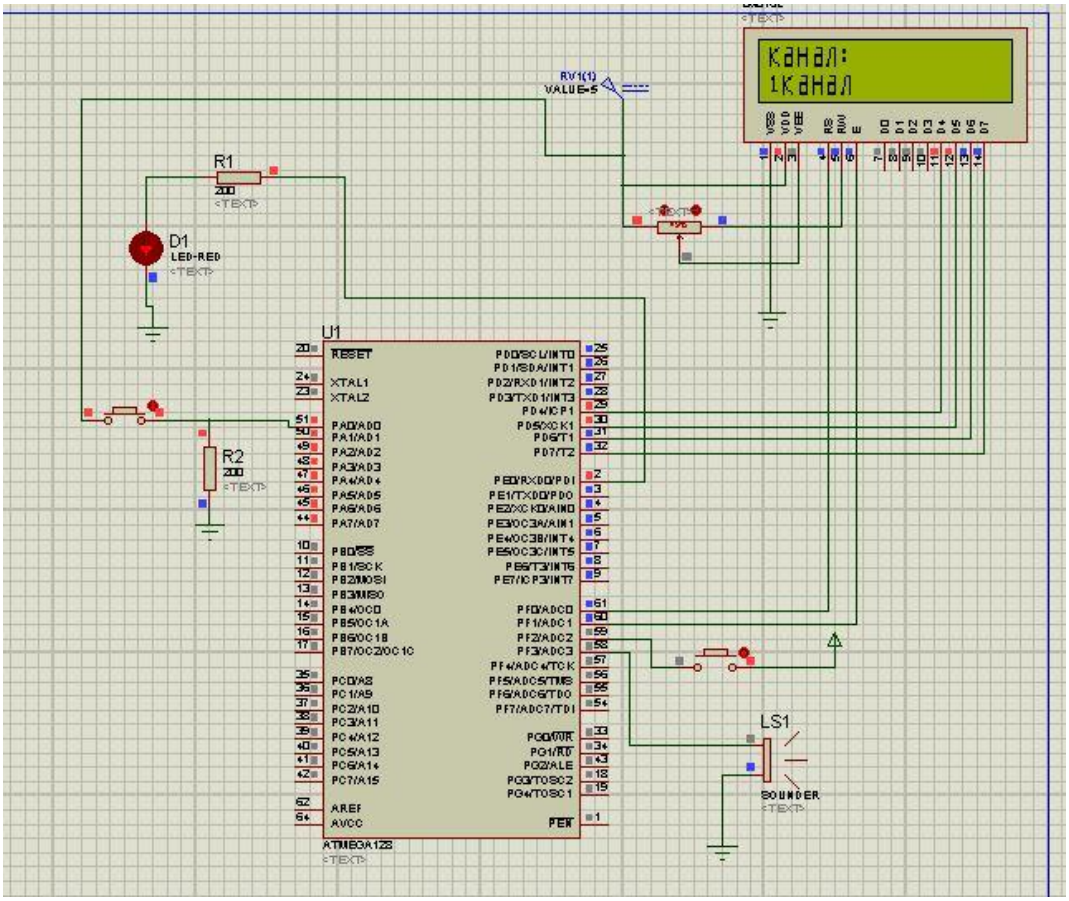


Рисунок 4.7 – Окончательная схема подключения функциональных блоков

5 Фильтр, стабилизатор, источник питания

5.1 Фильтр

Самые простые фильтры состоят из резисторов и конденсаторов. Эти простые RC–фильтры верхних или нижних частот обеспечивают пологие характеристики коэффициента передачи с наклоном 6 дБ/октава после точки, соответствующей значению коэффициента передачи —3 дБ. Можно построить полосовой фильтр, соединяя каскадно фильтры верхних и нижних частот; при этом характеристики такого фильтра опять же имеют пологие «склоны» с наклоном 6 дБ/октава. Для многих целей такие фильтры вполне подходят, особенно в тех случаях, когда сигнал, который должен быть подавлен, далеко сдвинут по частоте относительно желательной полосы пропускания. В качестве примеров можно указать шунтирование радиочастотных сигналов в схемах усиления звуковых частот, «блокирующие» конденсаторы для исключения постоянной составляющей и разделение модулирующей и несущей частот.

Изображенный на рисунке 5.1 двухполюсный фильтр широко применяется благодаря повышенной устойчивости и легкости регулировки. Он называется фильтром на основе метода переменных состояния. Среди прочих достоинств этой схемы существенна возможность путем коммутации выходов получать из одной схемы фильтры верхних и нижних частот, а также полосовой фильтр. Кроме того, частоту фильтра можно регулировать при неизменном значении добротности Q (или неизменной полосе пропускания – по выбору) характеристики в полосе пропускания. Несколько секций могут быть соединены каскадно для создания фильтров более высоких порядков.

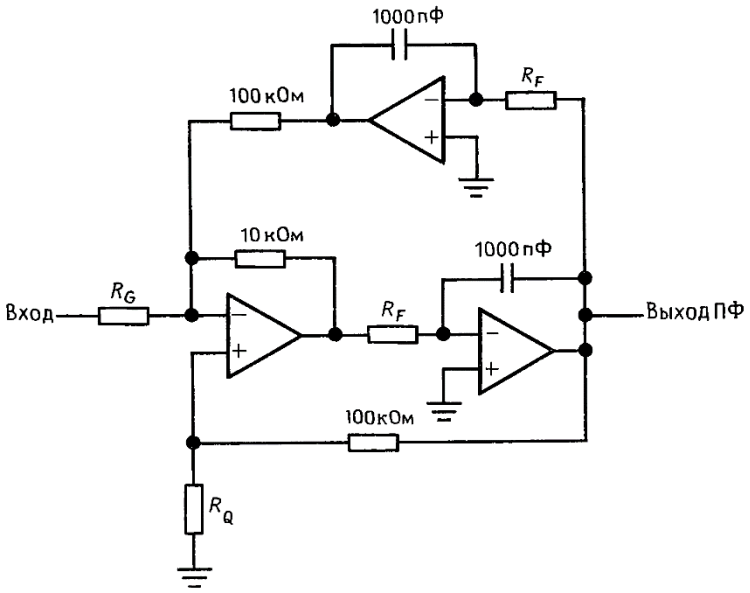


Рисунок 5.1 – Фильтр на основе метода переменных состояний

Несмотря на большое число схемных элементов, фильтр, построенный на основе метода переменных состояния, представляется наиболее удачной схемой для реализации (высокодобротных) полосовых фильтров. Он обладает низкой поэлементной чувствительностью, не предъявляет высоких требований к ширине полосы пропускания ОУ, а также прост в настройке. Например, в представленной на рисунке 5.1 схеме, используемой в качестве полосового фильтра, с помощью двух резисторов R_F устанавливается центральная частота полосы пропускания, в то время как резисторы R_Q и R_G совместно определяют добротность Q и коэффициент усиления в полосе пропускания.

Следовательно, можно сделать настраиваемый по частоте фильтр с фиксированной добротностью β при использовании в качестве резистора R_F двухсекционного переменного резистора (потенциометра). С другой стороны, переменным можно сделать резистор R_Q, при этом получается фильтр с фиксированной частотой и изменяемой добротностью Q.

На рисунке 5.2 изображена полезная модификация полосового фильтра на основе метода переменных состояния.

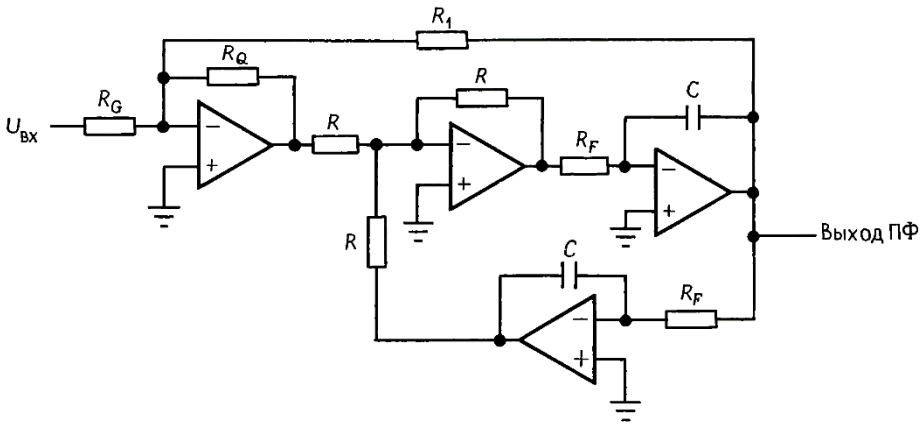


Рисунок 5.2 – Модификация полоскового фильтра на основе метода переменных состояний

Недостатком является использование в ней четырех ОУ, достоинство же заключается в возможности регулировать ширину полосы пропускания (т.е. добротность Q) без изменения коэффициента усиления в полосе. Действительно, как добротность Q, так и коэффициент усиления устанавливаются единственным резистором. Добротность Q, коэффициент усиления и центральная частота полосы пропускания полностью независимы и задаются следующими простыми соотношениями.

$$f_0 = \frac{1}{2 \cdot \pi \cdot R_f \cdot C} \qquad (5.1)$$

где: f_0 – частота на которую настраивается фильтр;

R_f – сопротивление резистора R_f;

C – емкость конденсатора C.

$$Q = R_1 \cdot R_Q, \qquad (5.2)$$

где: Q – добротность фильтра;

R_1 – сопротивление резистора R_1;

R_Q – сопротивление резистора R_Q.

$$G = R_1 \cdot R_G, \quad (5.3)$$

где: G – коэффициент усиления;

R_G – сопротивление резистора R_G.

R = 10 кОм.

В процессе разработки было принято использовать именно этот тип активного фильтра. Для его реализации используется микросхема КР1435УД4. Как заявлено производителем, эта схема имеет высокое входное сопротивление и пониженные нелинейные искажения. Так же, по сравнению с другими микросхемами серии КР1435, обладает высоким быстродействием – 10 В/мкс. Данная микросхема в своем корпусе содержит четыре операционных усилителя и имеет компоновку представленную в таблице 5.2.

Таблица 5.2 – Компоновка микросхемы КР1435УД4

Назначение вывода	Номер вывода			
Канал	1	2	3	4
Вход +	3	5	10	12
Вход –	2	6	8	13
Выход	1	7	9	14
U_{cc}+	4	4	4	4
U_{cc} –	11	11	11	11

Как уже было сказано выше, для организации полоскового фильтра одних ОУ не достаточно. Поэтому необходимо рассчитать значение всех компонентов схемы, приведенной на рисунке 5.2.

Из формулы 5.1 можно рассчитать значение сопротивления R_f и емкости C.

Примем R_f = 10 кОм, тогда емкость C составит:

$$C = \frac{1}{2 \cdot 3{,}14 \cdot 10 \cdot 10^3 \cdot 10 \cdot 10^6} = 1{,}6 \text{ нФ}.$$

По формуле 5.2, задав добротность Q = 100 и сопротивление R_1=10 кОм, определим сопротивление R_Q.

$$R_Q = 100 \cdot 10 \cdot 10^3 = 1 \cdot 10^6 = 1 \text{ МОм}.$$

Зная коэффициент усиления G = 1 и R_1 = 10 кОм, определим сопротивление R_G.

$$R_G = 1 \cdot 10 \cdot 10^3 = 10 \text{ кОм}.$$

В разрабатываемом устройстве было принято использовать чип – конденсаторы и чип – резисторы фирмы Yageo выполненные в корпусе типа 0805.

5.1.1 Блок «А»

На схеме электрической принципиальной можно видеть, что один участок схемы повторяется для всех программируемых портов. Для удобства чтения схемы этого участка условно было принято называть его блок «А». Его структура приведена на рисунке 5.3.

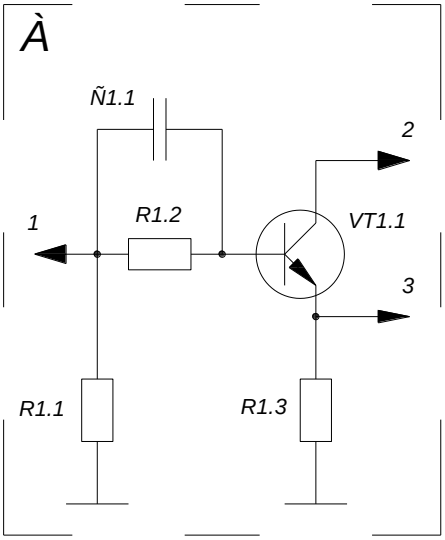


Рисунок 5.3 – Структура блока «А»

На рисунке видно, что данная схема в своем составе имеет такой элемент как транзистор.

Транзистор – это один из основных «активных» компонентов. Он представляет собой устройство, которое может усиливать входной сигнал по мощности. Увеличение мощности сигнала происходит за счет внешнего источника питания. Отметим, что увеличение амплитуды сигнала не является в данном случае определяющим. Так, например, повышающий трансформатор – «пассивный» компонент, такой же, как резистор или конденсатор, обеспечивает усиление по напряжению, но не может усилить сигнал по мощности. Устройства, которые обладают свойством усиления по мощности, характеризуются способностью к генерации, обусловленной передачей выходного сигнала обратно на вход.

Изобретателей транзистора когда–то заинтересовала именно способность устройства усиливать сигнал по мощности. Для начала они соорудили с помощью транзистора усилитель звуковых частот для громкоговорителя и убедились, что на выходе сигнал больше, чем на входе.

Транзистор является неотъемлемой частью всякой электронной схемы, начиная от простейшего усилителя или генератора до сложнейшей цифровой вычислительной машины. Интегральные схемы (ИС), которые в основном заменили схемы, собранные из дискретных транзисторов, представляют собой совокупности транзисторов или других компонентов, построенные на едином кристалле полупроводникового материала.

Обязательно следует разобраться в том, как работает транзистор, даже если вам придется пользоваться в основном интегральными схемами. Дело в том, что, для того чтобы собрать электронное устройство из интегральных схем и подключить его к внешним цепям, необходимо знать входные и выходные характеристики каждой используемой ИС. Кроме того, транзистор служит основой построения межсоединений, как внутренних (между ИС), так и внешних. И наконец, иногда (и даже довольно часто) случается, что подходящей ИС промышленность не выпускает и приходится прибегать к схемам, собранным из дискретных компонентов. Транзисторы сами по себе очень интересны, и ознакомление с их работой доставит удовольствие.

Обычно изучая транзистор, пользуются его эквивалентной схемой и h–параметрами. На наш взгляд, такой подход сложен и надуман. И дело не только в том, что, глядя на мудреные уравнения, вы едва ли поймете, как работает схема, скорее всего вы будете иметь смутное представление о параметрах транзистора, их значениях и самое главное диапазонах изменения.

И наконец, несколько слов о принятых в инженерной практике условностях. Напряжение на выводе транзистора, взятое по отношению к потенциалу земли, обозначается буквенным индексом (К, Б или Э): например, Uк–это напряжение на коллекторе. Напряжение между выводами обозначается двойным индексом, например, $U_{БЭ}$–это напряжение между базой и эмиттером. Если индекс образован двумя одинаковыми буквами, то это–напряжение источника питания: Uкк–это напряжение питания (обычно положительное) коллектора, Uээ – напряжение питания (обычно отрицательное) эмиттера.

На рисунке 5.4 показан эмиттерный повторитель. Легко заметить, что в блоке «А» транзистор включен именно по этой схеме включения.

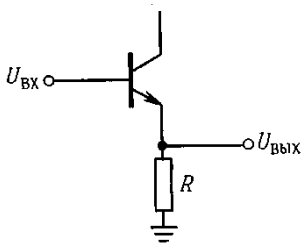


Рисунок 5.4 – Схема эмиттерного повторителя

Он назван так потому, что выходной сигнал снимается с эмиттера, напряжение на котором равно напряжению на входе (на базе) минус падение напряжения на диоде (на переходе база–эмиттер): $U_э = U_Б - 0,6$ В. Выходной сигнал по форме повторяет входной, но уровень его напряжения на 0,6–0,7 В ниже. Для приведенной схемы входное напряжение $U_{ВХ}$ должно составлять по крайней мере 0,6 В, иначе выходное напряжение будет равно потенциалу земли. Если к эмиттерному резистору подключить источник отрицательного напряжения, то входной сигнал может быть отрицательным. Следует отметить, что в эмиттерном повторителе отсутствует резистор в коллекторной цепи.

На первый взгляд эта схема может показаться бесполезной, но дело в том, что ее входной импеданс значительно больше, чем выходной. Из этого следует, что источник входного сигнала будет отдавать меньшую мощность, если нагрузку подключить к нему не непосредственно, а через эмиттерный повторитель. Поэтому обладающий внутренним импедансом источник (имеется в виду его эквивалентная схема) может через повторитель работать на нагрузку, которая обладает сравнимым или даже более низким импедансом, без потери амплитуды сигнала (эта потеря неизбежна при прямом включении из–за эффекта делителя напряжения). Иными словами, эмиттерный повторитель обеспечивает

усиление по току, хотя и не дает усиления по напряжению. Он также обеспечивает усиление по мощности.

Определим все компоненты схемы, представленной на рисунке 5.3. Как можно узнать из схемы электрической принципиальной, напряжение приложенное к коллектору транзистора составляет $U_{кк}$ = 12В, напряжение приложенное к базе $U_Б$ = 5В, ток базы $I_Б$ = 0,1мА.

Сопротивление R1.1 можно определить по формуле 5.4.

$$R1.1 = \frac{U_Á}{I_á} \tag{5.4}$$

Получим:

$$R1.1 = \frac{5}{0{,}1 \times 10^{-3}} = 50 \text{ кОм.}$$

Ток коллектора $I_к$ можно определить по формуле 5.5.

$$I_к = h_{21} \cdot I_Б, \tag{5.5}$$

где: h_{21} – это коэффициент передачи по току.

Для расчета примем его равным 30, что соответствует параметрам транзистора КТ315А. Тогда ток $I_к$ будет равен:

$$I_к = 30 \cdot 0{,}1 \cdot 10^{-3} = 3 \text{ мА.}$$

Для упрощения расчетов примем $I_к = I_э$. Зная ток эмиттера и напряжение на коллекторе $U_{кк}$ = 12В, определим значение R1.3 по формуле 5.6.

$$R1.3 = \frac{U_{КК}}{I_Э} \tag{5.6}$$

$$R1.3 = \frac{12}{3 \times 10^{-3}} = 4 \text{ кОм}.$$

Остается определить значение сопротивления R1.2. Эта задача решается при помощи известного соотношения приведенного в формуле 5.7.

$$R1.2 = 0{,}1 \cdot h_{21} \cdot R_{Э} \qquad (5.7)$$

Подставив все необходимые данные в эту формулу, получим значение сопротивления R1.2.

$$R1.2 = 0{,}1 \cdot 30 \cdot 4 \cdot 10^{3} = 12 \text{ кОм}$$

Конденсатор С используется в данной схеме для ускорения протекания процессов в транзисторе. Это необходимо для того чтобы уменьшить различия между поступающим сигналом и сигналом на выходе схемы с точки зрения временных характеристик. Данный конденсатор выбирается исходя из значения емкости p – n перехода транзистора. Зная этот факт, емкость конденсатора составит 0,16 пФ.

5.2 Блок «В», стабилизатор

Блок «В», также как и блок «А», повторяется для каждого порта. Функциональной нагрузкой данного блока является прием сигналов, проходящих через испытуемую цепь, и передача их на программируемую логическую интегральную схему. В своем составе он имеет ранее описанный активный фильтр. Структура блока Б представлена на рисунке 5.5.

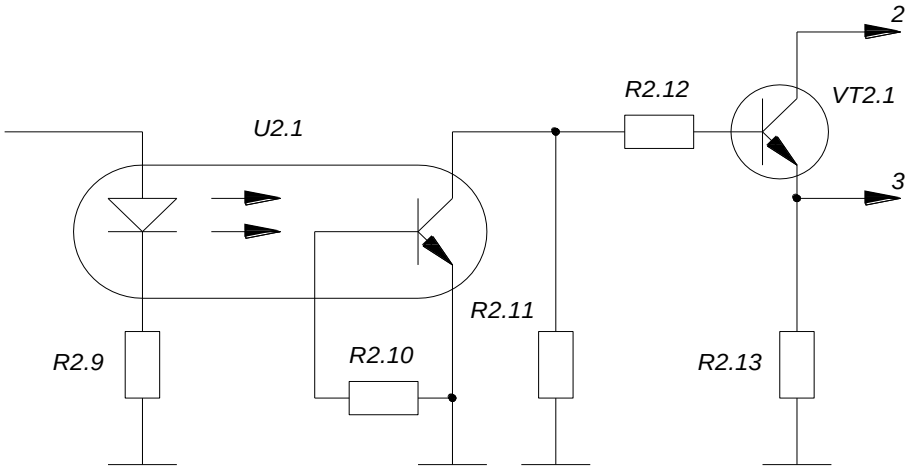


Рисунок 5.5 – Структура блока «В»

В этой схеме резистор R2.9 является токоограничивающим. Это связано с тем, что в случае его отсутствия, при открытии диода оптопары U2.1 произойдет короткое замыкание, так как сопротивление диода в открытом состоянии очень мало. Данный резистор можно рассчитать по формуле 5.8.

$$R2.9 = \frac{U}{I}, \qquad (5.8)$$

где: U – напряжение в цепи;

I – ток в цепи.

Так как активный фильтр собран таким образом, что обеспечивает коэффициент усиления по напряжению равный 1, то напряжение цепи будет равно напряжению поступающему с блока А, то есть 12В. Ток в свою очередь так же не изменяется и составляет 3мА. Зная эти параметры определим значение резистора R1.3.

$$R2.9 = \frac{12}{3 \cdot 10^{-3}} = 4\,\text{кОм}.$$

Для разрабатываемого устройства была выбрана оптопара 4N38 фирмы Toshiba. Данный компонент имеет коэффициент передачи по току 10%. Это значит, что ток на базе транзистора U2.1 составит 0.3 мА.

Резисторы R2.10 и R2.11 идентичны. Значение их сопротивления определяются формулой, аналогичной формуле 5.8.

$$R2.10 = R2.11 = \frac{12}{0{,}3 \cdot 10^{-3}} = 40\ \text{кОм}.$$

Ток коллектора, сопротивление базы и сопротивление эмиттера можно определить по ранее описанным формулам 5.4, 5.6 и 5.7 соответственно. Для данного участка схемы используется тот же транзистор, что и в блоке А.

$$I_к = 30 \cdot 0{,}3 \cdot 10^{-3} = 9\ \text{мА}.$$

$$R2.13 = \frac{U_{КК}}{I_Э} = \frac{5}{9 \cdot 10^{-3}} = 0.5\ \text{кОм}.$$

$$R2.12 = 0{,}1 \cdot 30 \cdot 0{,}5 \cdot 10^{3} = 1{,}5\ \text{кОм}.$$

5.3 Источник питания

В качестве источника питания было принято использовать понижающий трансформатор с тремя вторичными обмотками. Каждая из обмоток соответствует определенному номиналу напряжения: 3.3В – для питания микроконтроллера и ПЛИС; 5В – для питания индикатора.

В техническом задании оговорено, что потребляемая мощность не должна превышать 60Вт. Для расчета источника питания так же важно определить мощность, которую устройство будет потреблять от сети переменного тока.

5.3.1 Расчет потребляемой мощности

Для определения мощности потребляемой устройством в целом необходимо изначально определить мощность потребляемую каждым устройством в отдельности. Потребителями мощности в данном устройстве выступают:

а) Микроконтроллер;

б) Программируемая логическая интегральная схема;

в) Операционные усилители;

г) Транзисторы;

д) Индикатор.

Для определения потребляемой мощности можно воспользоваться формулой 5.9.

$$P = U \cdot I, \qquad (5.9)$$

где: U – напряжение питания;

I – потребляемый ток.

Воспользовавшись этой формулой применительно к каждому из элементов схемы, получим значения мощности потребляемой каждым из элементов.

Микроконтроллер:

$$P = 3.3 \cdot 60 \cdot 10^{-3} = 198\,\text{мВт.}$$

Программируемая логическая схема:

$$P = 3.3 \cdot 120 \cdot 10^{-3} = 396\,\text{мВт.}$$

Транзистор:

$$P = 12 \cdot 1 \cdot 10^{-3} = 12\,\text{мВт.}$$

Операционный усилитель:

$$P = 5 \cdot 11 \cdot 10^{-3} = 55\,\text{мВт.}$$

Индикатор:

$$P = 5 \cdot 4 \cdot 10^{-3} = 20\,\text{мВт.}$$

Получив эти данные можно рассчитать суммарную потребляемую мощность. При этом нужно учесть, что транзисторов в схеме 40, а микросхем с операционными усилителями 20.

$$P_{сум} = 198 + 396 + 20 + 12 \cdot 40 + 55 \cdot 4 \cdot 20 = 5494 \text{ мВт.}$$

5.3.2 Расчет трансформатора

Зная мощность потребляемую устройством, можно рассчитать мощность трансформатора.

$$P_{тр} = 1{,}25 \cdot P_{сум} \qquad (5.10)$$

$$P_{тр} = 1{,}25 \cdot 5494 \cdot 10^{-3} \approx 8 \text{ Вт.}$$

Определив мощность трансформатора, рассчитаем значение тока, текущего в первичной обмотке.

$$I_1 = \frac{P_{тр}}{U_1} \qquad (5.11)$$

где: I_1 – ток текущий через первую обмотку;

U_1 – напряжение на первичной обмотке трансформатора.

$$I_1 = \frac{8}{220} = 0{,}03 \text{ А.}$$

Рассчитаем необходимую площадь поперечного сечения сердечника магнитопровода.

$$S = 1.3 \cdot P_{тр} \qquad (5.12)$$

где: S – площадь поперечного сечения магнитопровода.

$$S = 1.3 \cdot 11 = 14{,}3 \text{ см}^2.$$

Определим количество витков первичной (сетевой) обмотки.

$$W_1 = \frac{50 \cdot U_1}{S} \qquad (5.13)$$

где: W_1 – число витков первичной обмотки.

$$W_1 = \frac{50 \cdot 220}{14{,}3} = \frac{11000}{14{,}3} \approx 770 \text{ витков.}$$

Определим количество витков вторичных (обмотки 2, 3 и 4) обмоток.

$$W_n = \frac{55 \cdot U_n}{S} \qquad (5.14)$$

где: W_n – число витков обмотки n;

U_n – напряжение обмотки n.

$$W_2 = \frac{55 \cdot U_2}{S} = \frac{55 \cdot 12}{14{,}3} = 46 \text{ витков.}$$

$$W_3 = \frac{55 \cdot U_3}{S} = \frac{55 \cdot 5}{14{,}3} = 20 \text{ витков.}$$

$$W_4 = \frac{55 \cdot U_4}{S} = \frac{55 \cdot 3{,}3}{14{,}3} = 17 \text{ витков.}$$

Определим диаметр проводов обмоток трансформатора. Это можно сделать исходя из значения тока обмотки и значений таблицы 5.3.

Таблица 5.3 – Диаметр провода обмотки

$I_{обм}$, мА	Менее 25	25 – 60	60 – 100	100 – 160	160 – 250	250 – 400	400 – 700	700 – 1000
d,	0,1	0,15	0,2	0,25	0,3	0,4	0,5	0,6

мм								

Для простоты конструкции оценка диаметра провода производится из значения максимального из всех токов обмоток – ток в четвертой обмотке I_4 = 180 мА. Из этого следует, что в качестве провода для обмоток трансформатора был выбран провод ПЭВТЛ – 2 диаметр которого составляет 0,3 мм.

В качестве магнитопровода трансформатора выбран сердечник кольцевого типа B64290 – A(K)58 – N30.

5.3.3 Схема источника питания

Принципиальная схема источника питания приведена на рисунке 5.6.

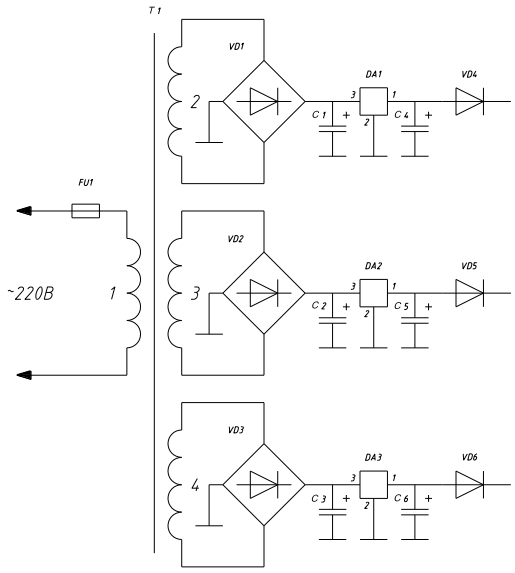


Рисунок 5.6 – Принципиальная схема источника питания

Как видно из рисунка, в цепь первичной обмотки для предотвращения выхода из строя разрабатываемого устройства из – за скачков напряжения питающей сети, включен предохранитель FU ВП4 – 1. Данный элемент выйдет из строя, прекратив протекание тока в цепи, при токе первичной обмотки превышающим значение 50 мА.

Цепи вторичных обмоток идентичны как по строению, так и по элементной базе. В цепь каждой из обмоток установлен диодный мост КЦ405А. Этот элемент обеспечивает отсечение отрицательной полуволны синусоидального сигнала, поступающего с обмоток трансформатора. Таким образом, происходит преобразование переменного тока в постоянный.

Конденсаторы С1 – С6 (К50 – 29) и аналоговые микросхемы DA1 – DA3 (КР1170ЕН5) обеспечивают стабилизацию напряжения, а так же снижение пульсаций.

На выходе схемы установлены импульсные диоды VD3 – VD6 (КД522Б). Функция этих элементов схемы заключается в устранении возможных обратных токов.

6 Организационно-экономический раздел.

В данном разделе приводится экономический анализ и сетевое планирование разрабатываемого устройства. Оно заключается в выявлении оценки экономического эффекта от производства проектируемой единицы продукции. В процессе выполнения данного раздела выделены и рассчитаны основные этапы экономического планирования, по итогам которого построен сетевой график, наглядно отражающий время выполнения работ. Также проведен расчет сметы производства и себестоимости готового изделия.

В конце раздела подведены итоги экономического анализа, в которых отображены наиболее удачные варианты для запуска в дальнейшем серийной продукции разработки.

6.1 Анализ аналогов, сравнение технических параметров

Разрабатываемый прибор автоматизированной регистрации является модификацией существующего прибора ЕС6, который имеет устаревшую технологию изготовления, при этом регистрация проходит по 3 каналам регистрирует замыкание и по 3-м каналам происходит регистрация размыкания. Прибор ЕС6 был изготовлен в 1979 году, элементная база прибора сильно устарела, схема построена полностью на аналоговых компонентах. Логика построена на реле. В разрабатываемом приборе, по сравнению с предшествующим прибором ЕС6 , элементная база будет сделана на современной элементной базе, с использованием всевозможных микросхем. А именно с использованием программируемой логической микросхемы, которая используется для логики. Так же будет использован микроконтроллер для индикации срабатывания канала. В приборе ЕС6 методом индикации служила одна лампочка над каналом. В разрабатываемом приборе используется загорание светодиода над срабатываемом канале, звуковая индикация , так же будет на дисплее отображаться какой канал сработал в данный момент . Так же основным

преимуществом разрабатываемого прибора будет то, что прибор будет работать на 20 каналов, при этом канал сам будет определять на что он работает. Либо на регистрацию замыкания, либо на регистрацию размыкния.

В настоящее время на рынке аналогов данного прибора нет.

Понятия параметров сетевого планирования

Сетевой анализ является методом анализа сроков начала и окончания частей проекта, который позволяет связать выполнение различных работ и процессов во времени, получив прогноз общей продолжительности реализации всего проекта. С его помощью можно выделить основные наиболее рациональные пути развития этапов производства от идеи до готовой единицы продукции и получить объективную оценку этих параметров при выбранном варианте структуры работ и распределения ресурсов.

Существуют различные методы построения сетевого планирования:

Детерминированные сетевые методы

а) Диаграмма Ганта с дополнительным временным люфтом 10-20 %;

б) Метод критического пути (МКП).

Вероятностные сетевые методы

Неальтернативные

а) Метод статистических испытаний (метод Монте-Карло);

б) Метод оценки и пересмотра планов (PERT).

Альтернативные

а) Метод графической оценки и анализа (GERT).

Сетевое планирование и управление содержит три основных этапа: структурное планирование, календарное планирование и оперативное управление. Оно начинается с разбиения исследуемого проекта на определенные шаги, необходимые для достижения конечной цели

разработки. Затем определяются оценки продолжительности работ и строится сетевой график, который позволяет, прежде всего, оценить временные характеристики проекта и входящих в него работ.

Сетевой график состоит из элементов двух видов – работ и событий, и позволяет в наглядной форме представить временную структуру проекта. Он отображает взаимосвязи между работами внутри проекта и порядок их выполнения. График является направленным графом, в котором каждая работа представляется ориентированной дугой, а каждое событие – узлом. Событие в свою очередь определяется как момент времени, когда завершается одна работа и начинается другая.

В процессе разработки проекта, при построении сетевого графика учитываются основные правила его построения:

- перед построением графика в технологической последовательности устанавливаются правила выполнения начала и окончания работ;
- сетевая модель строится от начала к окончанию при прямом расчете и в обратном направлении при обратном расчете;
- в сетевой модели отсутствуют "тупиковые" события;
- в сетевой модели отсутствуют замкнутые контуры - пути, соединяющие события, связанные сами с собой.

Особое значение в сетевом графике имеют критические работы. Работа считается критической, если задержка ее начала приводит к задержке срока окончания проекта в целом. Некритическая работа отличается тем, что промежуток времени между ее ранним началом и поздним окончанием больше ее фактической продолжительности. Другими словами, любая некритическая работа имеет резерв времени, что позволяет максимально оптимизировать её выполнение. Поэтому в данной дипломной работе в экономических расчетах используется метод критического пути. Он исходит из того, что длительность операций можно оценить с достаточно высокой степенью точности и определенности. Основным достоинством метода критического пути является

возможность манипулирования сроками выполнения задач, не лежащих на критическом пути.

Задачи лежащие на критическом пути имеют нулевой резерв времени выполнения и в случае изменения их длительности изменяются сроки всего проекта. В связи с этим при разработке критические задачи требуют более тщательного контроля, в частности, своевременного выявления проблем и рисков, влияющих на сроки их выполнения и, следовательно, на сроки выполнения проекта в целом.

Календарное планирование по выбранному МКП требует определенных входных данных. После их ввода производится процедура прямого и обратного прохода по сети и вычисляется выходная информация.

6.2 Анализ этапов разработки, построение сетевого графика

Перед построением модели сетевого графика рассчитываются временные параметры всех составляющих его компонентов.

Для полного построения необходимо произвести два вида расчетов: прямой и обратный. При прямом расчете вычисляются ранние сроки выполнения каждого события. При обратном вычислении они производятся в обратной последовательности, что позволяет вычислить поздние сроки событий. Также на сетевом графике отмечаются номера событий и разница во времени между ранними и поздними сроками.

Прямой расчет – определение минимально возможного времени реализации проекта начинается с работ, не имеющих предшественников. В ходе него определяется *ES* (ранний старт) и *EF* (ранний финиш). Ранние начала и ранние окончания работ определяются последовательно, слева направо по графику, то есть от исходного события сети к завершающему.

Основными формулами для расчетов являются:

$$ES_0 = 0, \tag{6.1}$$

$$EF = ES + CONT, \tag{6.2}$$

где: ES_0 - нулевой ранний старт,

ES - ранний старт события,

EF - ранний финиш события,

CONT -продолжительность события.

Ожидаемое время выполнения работы ($t_{ожид.}$) рассчитывается по двухоценочной методике, исходя из минимальной ($t_{мин.}$) и максимальной ($t_{макс.}$) оценок продолжительности работы. При этом предполагается, что минимальная оценка соответствует наиболее благоприятным, а максимальная - наиболее неблагоприятным условиям работы.

Ниже приведены основные формулы прямого расчета. Ожидаемая продолжительность каждой работы рассчитывается по формуле:

$$t_{ожид.} = 0{,}6\ t_{мин.} + 0{,}4\ t_{макс.}. \quad (6.3)$$

Среднеквадратическое отклонение (δ_{ij}) продолжительности выполнения работ в двухоценочной методике определяется по формуле:

$$\delta_{ij} = 0{,}2 \cdot (t_{ij_{MAX}} - t_{ij_{MIN}}). \quad (6.4)$$

Дисперсия определяется по формуле:

$$D_{ij} = 0{,}04 \cdot (t_{ij_{MAX}} - t_{ij_{MIN}})^2. \quad (6.5)$$

Если при прямом расчете возникают ситуации выполнения двух или более параллельных процессов, то ранний срок свершения следующего промежуточного события вычисляется по функции максимума:

$$t_p = \max(t_{pi} + t_{ij}), \quad (6.6)$$

где: t_{ij} - ожидаемая продолжительность работы,

t_{pi} - ранний срок свершения события, непосредственно предшествующего данному.

Обратный расчет. Определяются *LS* (поздний старт), *LF* (поздний финиш) и *R* (резерв). Поздние начала и поздние окончания определяются в обратном порядке – от завершающегося события графика к исходящему, то есть справа налево.

Основными формулами для расчетов являются:

$$EF_N = LF_N, \tag{6.7}$$

$$LS_i = LF_i - CONT, \tag{6.8}$$

где: LF_N - поздний финиш конечного события,

LS - поздний старт события,

LF - поздний финиш события,

CONT -продолжительность события.

После прохождения по сетевому графику в прямом и обратном направлениях должно выполняться условие $ES_0=LS_0$. Это означает, что для завершающего события никакие резервы времени не планируются. Если при обратном расчете возникают ситуации выполнения двух или более параллельных процессов, то поздний срок свершения следующего промежуточного события вычисляется по функции минимума:

$$t_s = \min(t_{si} - t_{ij}), \tag{6.9}$$

где: t_{si} - ожидаемая продолжительность работы,

t_{ij} - поздний срок свершения события непосредственно следующего за данным промежуточным событием.

Резерв времени свершения события - это промежуток времени, на который может быть отсрочено событие без нарушения сроков разработки в целом. Он образуется у событий, для которых поздний срок свершения события больше раннего срока, то есть имеется свободный временной промежуток:

$$R_i = t_s - t_p, \tag{6.10}$$

Если $t_s = t_p$, то есть поздний срок свершения события равен раннему сроку свершения события, то такое событие не имеет резерва времени и это событие относится к критическому пути.

Результаты расчетов временных параметров событий сетевого графика приведены в таблице 6.1.

После построения сетевого графика проводится его всесторонний анализ, с тем чтобы в дальнейшем принять меры по его оптимизации. Определить степень трудности выполнения в срок каждой группы работ некритического пути можно с помощью коэффициента напряженности работ. Коэффициентом напряженности $K_н$ работы называется отношение продолжительности несовпадающих (заключенных между одними и теми же событиями) отрезков пути, одним из которых является путь максимальной продолжительности, проходящий через данную работу, а другим – критический путь:

$$K_H = \frac{t(L_{\max}) - t'_{кр}}{t_{кр} - t'_{кр}} \qquad (6.11)$$

где: $t(L_{max})$ – продолжительность максимального пути, проходящего через работу;

$t_{кр}$ – продолжительность отрезка рассматриваемого пути, совпадающего с критическим путем;

$t'_{кр}$ – продолжительность критического пути.

Таблица 6.1 - Проведение работ по разработке прибора

Номер работы	Наименование работы	Продолжительность, дн.			Исполнители, чел.		
		мин.	макс.	ожид.	руков.	инж.	лабор.
0-1	Получение технического задания на разработку	1	2	1	1	1	1
0-2	Выбор литературных источников	3	4	3	0	0	1
0-3	Поиск и анализ существующих аналогов	1	4	3	1	1	1
1-4	Определение стадий и этапов разработки прибора	2	3	3	1	1	1
3-4	Разработка структурной схемы прибора	1	5	4	0	1	0
2-4	Подбор элементной базы	2	5	4	0	1	0
4-5	Разработка идеи реализации ПЛИС	3	6	5	0	0	1
4-6	Схемотехническая часть реализации ПЛИС	10	20	15	0	0	1
4-7	Программная часть реализации ПЛИС	10	14	12	0	1	0
5-7	Подбор индикации прибора	3	6	3	0	1	0
6-7	Структура программы для МК	2	4	3	0	0	1
7-8	Чертеж схемы	12	15	13	0	1	1
7-9	Трассировка платы	8	11	9	0	1	1
7-10	Разработка корпуса прибора	2	10	8	0	0	1
8-10	Отладка программы МК	3	5	4	0	0	1
9-10	Отладка программы ПЛИС	3	5	4	0	0	1

Таблица 6.1 – Продолжение

Номер работы	Наименование работы	Продолжительность, дн.			Исполнители, чел.		
		мин.	макс.	ожид.	руков.	инж.	лабор.
10-11	Выявление достоинств и недостатков алгоритмов, оптимизация их работы	4	6	5	0	1	0
10-12	Оформление программной документации	2	3	2	1	0	1
11-12	Анализ литературы организационно-экономического раздела	2	5	4	0	0	1
12-13	Разработка и анализ сетевого графика	2	3	2	0	1	0
12-14	Расчет прибыли и экономического эффекта разработки	3	5	4	1	1	1
13-15	Оформление организационно-экономического раздела	4	5	4	0	0	1
12-15	Анализ литературы раздела БЖД	3	4	3	0	0	1
14-15	Разработка мер безопасности человека от негативных факторов на производстве	2	5	3	1	1	0
15-16	Оформление раздела БЖД	4	5	4	0	0	1
16-17	Оформление технической документации	3	5	4	1	1	1

Таблица 6.1 – Продолжение

Номер работы	Наименование работы	Продолжительность, дн.			Исполнители, чел.		
		мин.	макс.	ожид.	руков.	инж.	лабор
16-18	Оформление пояснительной записки	2	6	5	0	0	1
17-18	Оформление графических документов дипломного проекта	6	8	7	1	1	1
18-19	Проверка и сдача проекта	5	7	6	1	0	1

Коэффициент сложности сетевого графика представляет собой отношение количества работ сетевого графика к количеству событий и определяется по формуле:

$$K_{сложн} = n_{раб} / n_{соб}, \quad (6.12)$$

где K_c – коэффициент сложности сетевого графика;

$n_{раб}$ – количество работ, ед.;

$n_{соб}$ – количество событий, ед.

Сетевые графики, имеющие коэффициент сложности от 1,0 до 1,5, являются простыми, от 1,51 до 2,0 – средней сложности, более 2,1 – сложными. Для данного графика коэффициент сложности не превышает 1,5, график является простым:

$$K_{сложн} = 29/20=1,45. \quad (6.13)$$

Согласно приведенной выше таблице и расчетным данным, приводится сетевой график с рассчитанными значениями затрат по времени на каждую стадию разработки, он приведен на отдельном листе А1, который прилагается к пояснительной записке.

Экономические расчеты локализуются по этапам подготовки таким образом, чтобы они максимально способствовали нахождению оптимальных проектных решений, при этом взаимосвязь между техническими и

экономическими расчетами должна поддерживаться на продолжении всего цикла проектирования. Необходимость применения тех или иных экономических расчетов зависит целиком от характера проекта и от его особенностей. В экономических расчетах приходится оперировать многочисленными стоимостными и натуральными показателями. Применение новой техники, предназначенной для изготовления определенной продукции или выполнения определенной работы, должно привести к сокращению затрат на единицу выпускаемой продукции или выполняемой работы. Поэтому исходным стоимостным показателем является экономия (годовая) от снижения себестоимости, ожидаемая в результате внедрения новой техники или другого мероприятия, исчисленная в масштабе годового выпуска продукции или годового объема выполняемых работ по плану.

Ее можно считать по следующей формуле:

$$Э_г = C_1 - C_2, \qquad (6.14)$$

где: $Э_г$ – годовая экономия;

C_1 – себестоимость годового выпуска продукции (работ) до внедрения мероприятия;

C_2 – то же, после внедрения мероприятия.

На основе расчета годовой экономии в дальнейшем определяется экономическая эффективность капитальных вложений. При составлении сметы затрат на производство и плановой калькуляции, учитывается только та сумма экономии, которая может быть получена с момента внедрения до конца планируемого года.

Для определения экономической целесообразности новой техники или внедрение мероприятий необходимо рассчитать необходимые капитальные вложения. В простом случае, когда действующие производственные фонды без

изменения их состава и стоимости дополнительно оснащаются техническими средствами, нужно сопоставить дополнительные капитальные вложения на эти средства с ожидаемой экономией и рассчитать срок окупаемости или коэффициент экономической эффективности капитальных вложений по следующим формулам:

$$T = \frac{К_д}{C_1 - C_2}, \quad (6.15)$$

$$\varepsilon = \frac{C_1 - C_2}{К_д} = \frac{1}{T}, \quad (6.16)$$

где: Т – срок окупаемости капитальных вложений;

$К_д$ – сумма дополнительных капитальных вложений, необходимых для внедрения мероприятия;

ε – коэффициент экономической эффективности.

Наряду со стоимостными показателями экономическая эффективность новой техники и различных мероприятий измеряются так же и натуральными показателями: повышение производительности труда (увеличение выработки продукции на одного работающего); сокращение трудоемкости изготовления продукции в нормо – часах; количество высвобождаемых рабочих; экономия материалов в результате уменьшения норм расхода; количество высвобождаемого оборудования и производственных площадей; сокращение брака; улучшение условий труда. Размер экономии в прямых затратах труда и материалов рассчитывается сначала на единицу продукции или выполняемой работы путем сопоставлением норм до и после внедрения, а затем на весь объем выпуска продукции, предусмотренный по годовому плану.

Определение всех показателей эффективности нужно не только для выяснения экономической целесообразности внедрения новой техники или мероприятий, но и для правильного отражения результатов этого внедрения в соответствующих

показателях плана по производству, труду и заработной плате, материально – техническому снабжению, себестоимости.

Общие положения прямой методики определения экономической эффективности капитальных вложений исходят из того, что различные отрасли и звенья народного хозяйства связаны между собой взаимными поставками орудий и предметов труда. В связи с этим внедрение новой техники и другие мероприятия в одном звене приводят к изменению экономических показателей и в других, связанных с ним, звеньям. Поэтому необходимо определить народнохозяйственную эффективность внедрения новой техники и других мероприятий. Создание новых и усовершенствование существующих изделий затрагивает экономические показатели не только производства, но и эксплуатации их в самых различных областях науки, техники и народного хозяйства.

Если в результате создания нового или усовершенствование существующего изделия изменяется схема или конструкция по сравнению с предшествующим образцом, но не изменяются его эксплуатационные свойства (или они не поддаются количественному измерению), то это затрагивает только показатели производства. В таком случае экономический эффект определяется на месте изготовления изделия путем сопоставления приведенных затрат по сравниваемым вариантам по формуле:

$$Э_т = (C_1 + \xi_н \cdot К_1) - (C_2 + \xi_н \cdot К_2), \quad (6.17)$$

где: $Э_т$ – годовой экономический эффект;

C_1 и C_2 – себестоимость годового выпуска продукции до и после внедрения в производство новых или усовершенствованных изделий;

$К_1$ и $К_2$ – капитальные вложения или производственные фонды до и после внедрения в производство новых или усовершенствованных изделий;

$\varepsilon_н$ – нормативный коэффициент экономической эффективности.

Одним из самых важных вопросов, который встает перед разработчиками радиоэлектронной аппаратуры и экономистами, работающими в этой области, является вопросы экономически обоснованной цены на изделие. Выделяют два основных этапа ценообразования на предприятии. Этап первый – определение базовой цены, то есть цены без скидок, наценок, транспортных, страховых, сервисных компонентов. Этап второй заключается в определении цены с учетом вышеуказанных компонентов. Существует пять основных методов определения базовой цены любого устройства, которые можно использовать изолированно либо в различных комбинациях друг с другом:

а) метод полных издержек;

б) метод стоимости изготовления;

в) метод определения цены на базе сокращенных затрат;

г) метод рентабельности инвестиций;

д) метод маркетинговых оценок.

Для определения цены измерителя скорости пули наиболее рациональным методом будет метод стоимости изготовления.

6.3 Расчет оптовой цены устройства

Себестоимость и оптовая цена изделия на этапе его проектирования может быть определена различными методами, такими как:

а) Затратный метод или метод «средние затраты + прибыль»;

б) Параметрический метод;

в) Ценообразование на основе текущих цен;

г) Ценообразование на основе анализа безубыточности и обеспечения целевой прибыли;

д) Прочие.

На начальных этапах проектирования новой продукции ее себестоимость рассчитывается приближенными методами на основе укрупненных нормативов:

рассчитываем прямые затраты (основные материалы, покупные комплектующие изделия, заработную плату производственных рабочих), а косвенные затраты начисляются определенным процентом к заработной плате производственных рабочих (общепроизводственные, общехозяйственные, коммерческие расходы), определяемые на основе базового предприятия (ПСЗ г. Трехгорный).

6.4 Расчет покупных комплектующих

Стоимость покупных комплектующих изделий, расходуемых на одно изделие, определяется по формуле:

$$Р_{комп} = N_{комп} \cdot Ц_{комп}, \quad (6.19)$$

где: $N_{комп}$ – количество покупных комплектующих изделий, данного наименования;

$Ц_{комп}$ – цена единицы комплектующих изделий.

Расчет стоимости покупных изделий проведем в таблице 6.2.

Таблица 6.2 – Расчет стоимости покупных изделий для изготовления устройства регистрации кратковременных замыканий и размыканий электрических цепей изделий

Наименование материалов	Цена, руб.	Норма на единицу	Стоимость, руб.
Диодный мост КЦ405А	5,00	3	15,00
Диод КД522Б	5,00	3	15,00
Конденсатор Yageo C0805	0,90	60	54,00
Конденсатор 50 – 29	14,00	6	84,00
Микросхема ATMEGA128	450,00	1	450,00
Микросхема EPM240T100C5 – 1	250,00	1	250,00

Таблица 6.2 – Продолжение

Наименование материалов	Цена, руб.	Норма на единицу	Стоимость, руб.
Микросхема DV – 16236	936,00	1	936,00
Микросхема КР1435УД4	170,00	20	3400,00
Микросхема КР1170ЕН5	15,00	3	45,00
Печатная плата	1027,50	1	1027,50
Резистор Yageo R0805	1,10	512	563,20
Разъем DIN 41612 – 32x2	59,00	1	59,00
Светодиод BL-L102URC	15,00	20	300,00
Разъем 3 – 151	15,00	1	15,00
Предохранитель ВП4 – 1	3,54	1	3,54
Провод ПЭВТЛ – 2	250,00	1	250,00
Транзистор КТ315А	1,50	64	96,00
Ферритовый сердечник B64290 – A(K)58 – N30	26,00	1	26,00
Тактовая кнопка 0613HIM-130G-G (TC-A109)	37,00	1	37,00
Корпус наборный UKM	1348,50	1	1348,50
Итого:			8974,04

6.5 Расчет заработной платы основных рабочих

Для расчета величины заработной платы производственных рабочих необходимо определить норму времени, необходимую для выполнения определенной работы (комплекса операций), т.е. провести техническое нормирование комплекса работ по изготовлению устройства регистрации кратковременных замыканий(размыканий) электрических цепей изделий. В

связи с преобладающим в настоящее время в радио и приборостроении серийным и мелкосерийным характером производства нормативы трудоемкости разрабатываются, главным образом, на комплексы приемов. Норма на комплекс приемов рассчитывается при условии наличия маршрутного технологического процесса.

Техническая норма времени включает в себя норму подготовительно – заключительного времени Тпз и норму штучного времени Тшт. Штучное время состоит из оперативного времени Топ и времени обслуживания рабочего места Тоб, времени перерыва на отдых и личные надобности Тот. Оперативное время Топ состоит из основного Тос и вспомогательного Твс и равно их сумме. В мелкосерийном и серийном производстве время обслуживания рабочего места и время на отдых рассчитывают в процентах от оперативного времени. Норму штучного времени рассчитаем по формуле:

$$T_{шт} = T_{об} \cdot \left(1 + \frac{K}{100}\right), \qquad (6.20)$$

где: К – коэффициент, учитывающий отношение времени на обслуживание рабочего места и личные надобности к оперативному времени в процентах. Коэффициент определяют по таблице нормативов.

В большинстве случаев однородные технологические операции состоят из одних и тех же структурных элементов, что дает возможность применять укрупненные нормативы штучного времени. Укрупненные нормативы содержат расчетные величины оперативного или неполного штучного времени. Точность нормативов и степень их укрупнения соответствует типу производства. К примеру, для серийного производства она равна 10%, а для мелкосерийного – 15%. Под точностью нормативов понимают выраженное в процентах предельное отклонение, которое может получиться между нормой времени,

установленной по нормативам, и временем, полученным на основании хронометражных данных или технического расчета. Нормативы времени оформляются в виде особых таблиц, в которых указана продолжительность тех или иных элементов нормы в зависимости от влияющих на нее факторов. Для примера рассмотрим порядок нормирования некоторых работ, наиболее часто встречающихся в радиоаппаратостроении:

Нормирование слесарно – сборочных работ.

Сборочные процессы в радиоаппаратостроении характеризуются рядом особенностей зависящих от конфигурации деталей, технологического процесса сборки, способа соединения деталей и пр. В условиях мелкосерийного производства большое значение имеют слесарные работы, заключающиеся в ручной подгонке, доделке и т.д. Штучное время сборки рассчитывается по формуле:

$$T_{шт} = (t1 + t2 + t3 + ... + t_n) \cdot (1 + K), \quad (6.21)$$

где: t1… tn –оперативное время на каждую операцию (комплекс операций).

Нормирование электромонтажных работ.

В условиях мелкосерийного производства на монтажные работы разрабатываются укрупненные нормативы. В данных работах, в основном, преобладает ручной труд, поэтому оперативное время при нормировании не разделяется на основное и вспомогательное. Штучное время на электромонтажные работы рассчитывается по формуле:

$$T_{шт} = (T_{оп} + T_{и}) \cdot (1 + K), \quad (6.22)$$

где: $T_{и}$ – время на то, чтобы взять и отложить инструмент.

Аналогично рассчитывают нормативы времени на другие операции, применяемые в радиоаппаратостроении: работы по намотке катушек, изготовление деталей из пластмасс, штамповочные работы, станочные работы, промывочные работы, операции по обдувке и т. п.

Таблицы для расчета нормативов трудоемкости приведены в справочном пособии «Техническое нормирование труда в приборостроении» и «Справочник нормировщика». Все данные по проведенным расчетам целесообразно снести в следующую таблицу 6.3:

Таблица 6.3 – Укрупненный расчет штучного времени на изготовление единицы изделия

Содержание основных операций	Время на един. мин.	Кол – во единиц	Время операц. мин.
1 Резка проводов на заготовки	0,09	14	1,26
2 Снятие изоляции с одножильных проводов	0,14	14	1,96
3 Лужение концов проводов	0,08	14	1,12
4 Формовка выводов проводов	0,43	898	386,14
5 Установка элементов на плату	0,04	728	29,12
7 Электромонтажная	0,29	962	278,98
8 Промывка	5,7	1	5,7
9 Обдувка	0,08	1	0,08
10 Покрытие платы лаком	4,92	1	4,92

Итого:	709,28

Время на настройку и проверку работоспособности прибора определяется по формуле:

$$T' = (10\% \div 25\%) \cdot T_{опер} \text{ мин.} \quad (6.23)$$

Принимаем время настройки и проверки работоспособности прибора равным 12% от $T_{опер}$. Следовательно:

$$T' = 0{,}12 \cdot 718{,}9 = 86{,}27 \text{ мин.}$$

Точное время для изготовления устройства определяется по формуле:

$$T_{шт} = (T_0 + T') \cdot \left(1 + \frac{К}{100}\right), \quad (6.24)$$

где: К – коэффициент, учитывающий величину времени на обслуживание рабочего места и личные надобности.

Штучное время на изготовление прибора составляет:

$$T_{шт} = (718{,}9 + 86{,}27) \cdot \left(1 + \frac{10}{100}\right) = 885{,}69 \text{ мин.}$$

Для мелкосерийного производства точность получения нормативов составляет 15%, таким образом, время на изготовление прибора составляет:

$$T_{шт}' = 885{,}69 \cdot 1{,}15 = 1018{,}55 \text{ мин.}$$

Подготовительно – заключительная норма времени ($T_{пз}$), укрупнено принимаем равной 75% от Тшт’. Время затрачиваемое на подготовительные и заключительные операции составляет:

$$T_{пз} = 1018{,}55 \cdot 0{,}75 = 763{,}9 \text{ мин.}$$

Техническая норма времени определяется по формуле:

$$. T_н = T_{пз} + T_{шт}' \quad (6.25)$$

Техническая норма времени составляет:

$$T_н = 763,9 + 1018,55 = 1781,45 \text{ мин.}$$

Величина производственной зарплаты определяется по часовым тарифным ставках, премиальным надбавкам, применяемым на базовом предприятии ПСЗ г. Трехгорный. В данном случае применяется сдельно – премиальная система оплаты труда и прямая зарплата рассчитывается по формуле:

$$P_{сд} = \frac{(T_н \cdot T_{ст})}{60}, \quad (6.26)$$

где: $P_{сд}$ – заработная плата производственных рабочих на единицу продукции;

$T_{ст}$ – средняя часовая тарифная ставка основных рабочих ($T_{ст}$=20,15 руб.).

Заработная плата основных рабочих составляет:

$$P_{сд} = \frac{(1718,45 \cdot 20,15)}{60} = 592,27 \text{ руб.}$$

Премиальная надбавка (П) составляет 40% от заработной платы основных рабочих и равна:

$$П = 0,4 \cdot 598,27 = 239,31 \text{ руб.}$$

Районный коэффициент (РК) составляет 20% от заработной платы основных рабочих с премиальной надбавкой и равен:

$$РК = 0,2 \cdot (592,27 + 239,31) = 116,32 \text{ руб.}$$

Дополнительная заработная плата (Д) составляет 20% от заработной платы основных рабочих с учетом премиальной надбавки и районного коэффициента и равна:

$$Д = 0,2 \cdot (598,27 + 239,31 + 116,32) = 198,58 \text{ руб.}$$

Тогда фонд заработной платы основных рабочих (ФЗП) определяется по формуле:

$$ФЗП = Р_{сд} + П + РК + Д. \qquad (6.27)$$

Фонд заработной платы основных рабочих равен:

$$ÔÇÏ = 598,27 + 239,31 + 116,32 + 198,58 = 1143,48 \text{ руб.}$$

Ставка страховых взносов составляет 30% от фонда заработной платы основных рабочих.

Ставка страховых взносов равна:

$$ССВ = 0,30 \cdot 286,32 = 343,05 \text{ руб.}$$

6.6 Расчет себестоимости

Рассчитаем косвенные затраты на производство устройства регистрации кратковременных замыканий(размыканий) электрических цепей изделия в таблице 6.4.

Таблица 6.4 – Косвенные затраты на производство продукции

Наименование косвенных затрат	Процент от ФЗП, %	ФЗП, руб.	Величина затрат, руб.
Общепроизводственные	192	1143,48	2195,49

расходы			
Общехозяйственные расходы	135	1143,48	1543,70
Коммерческие расходы	92	1143,48	1052,00

Составим калькуляцию оптовой цены прибора автоматизированного контроля электрических цепей изделий. Сведем в таблицу 6.5.

Таблица 6.5 – Калькуляция оптовой цены прибора автоматизированного контроля электрических цепей изделий

Наименование статей калькуляции	Сумма, (руб.)	Структура (уд. вес в %)
Комплектующие изделия	8974,04	54
Фонд заработной платы	1143,48	5
Ставка страховых взносов	343,05	1
Общепроизводственные расходы	2195,49	9
Цеховая себестоимость	12422,11	69
Общехозяйственные расходы	1543,70	6
Заводская себестоимость	13965,81	76
Коммерческие расходы	1052,00	4
Полная себестоимость	15017,81	80
Прибыль	6754,45	20
Оптовая цена	19772,26	100

Прибыль составляет 20% от полной себестоимости или 6754,45 рубля.

6.7 Расчет срока окупаемости

Для расчета срока окупаемости разрабатываемого устройства изначально необходимо сравнить качественные и количественные показатели нового и старого устройства. Это сравнение приведено в таблице 6.5.

Таблица 6.5 – Качественные и количественные показатели устройств

Наименование показателя	ЕС6	Разрабатываемое устройство
Количество одновременно проверяемых цепей, шт.	3	20
Потребляемая мощность, Вт.	60	8
Максимальное время работы, ч.	8	8

Основное различие, принципиально важное для производства, заключается в количестве одновременно проверяемых цепей. Дальнейшие расчеты будут проводиться исходя из этого условия.

Как видно из таблицы 6.5 разрабатываемое устройство по количеству одновременно проверяемых цепей в 7 раз превосходит возможности старого устройства. Это значит, что для выполнения одних и тех же объемов работ понадобится 7 устройств ЕС6.

Для определения срока окупаемости определим затраты на использование обоих устройств в течении года. Эти затраты можно определить воспользовавшись формулой 6.26.

$$C_n = (C_{кВт} \times P + C_{раб} \cdot T_{раб}) \cdot N \cdot T \cdot D_{раб}, \quad (6.26)$$

где: С – затраты на использование устройства n;

$C_{кВт}$ – средняя стоимость одного кВт\ч;

P – потребляемая мощность в кВт\ч;

$C_{раб}$ – стоимость работ оператора в час;

$T_{раб}$ – время на подключение трех цепей;

N – количество устройств для проверки 32 цепей;

T – максимальное время работы;

$D_{раб}$ – количество рабочих дней.

Определим затраты на использование старого устройства:

$$C_1 = (4 \cdot 0{,}06 + 70 \cdot 0{,}008) \cdot 10 \cdot 8 \cdot 249 = 14576{,}00 \text{ руб.}$$

Определим затраты на использование нового устройства:

$$C_1 = (4 \cdot 0{,}008 + 70 \cdot 0{,}008) \cdot 1 \cdot 8 \cdot 249 = 1179{,}26 \text{ руб.}$$

Определим норму амортизации для нового устройства:

$$К_Д = К + К \cdot 0{,}083, \quad (6.27)$$

где: К – себестоимость изделия.

$$К_Д = 19772{,}26 + 19772{,}26 \cdot 0{,}083 = 21413{,}36 \text{ руб.}$$

Воспользовавшись формулой 6.28 определим срок окупаемости:

$$T = \frac{21413{,}36}{14576\text{-}1179{,}264} = 1{,}59, (\text{года}) \quad (6.28)$$

7 Безопасность жизнедеятельности

Разрабатываемый прибор автоматизированного контроля предназначен для регистрации кратковременного размыкание(замыкание) электрических цепей изделий. Прибор используется в лабораторный условиях, с использованием вибрационных стендов.

В данном разделе описываются основные опасности, которые могут возникнуть в процессе эксплуатации прибора, их воздействие на организм человека, а также меры их устранения.

7.1 Анализ опасных и вредных производственных факторов

Эксплуатация прибора автоматизированного контроля электрических цепей изделий производится в соответствии с ИОТ 002-2001 «Инструкция по охране труда при эксплуатации электроустановок до 1000 В».

К работам по эксплуатации прибора автоматизированного контроля электрических цепей изделий допускаются лица, прошедшие медицинский осмотр и инструктаж по охране труда.

Лица, допущенные к эксплуатации прибора автоматизированного контроля электрических цепей изделий, должны соблюдать правила внутреннего трудового распорядка, установленные режимы труда и отдыха.

При эксплуатации прибора автоматизированного контроля электрических цепей изделий возможно воздействие на работающих следующих опасных производственных факторов:

а) Поражение электрическим током при прикосновении к токоведущим частям;

б) Неисправности изоляции или зануления.

В процессе эксплуатации устройства персонал должен соблюдать правила использования средств индивидуальной защиты, соблюдать правила личной гигиены, содержать в чистоте рабочее место.

7.2 Анализ условий эксплуатации проектируемого устройства.

Помещения в отношении опасности поражения людей электрическим током классифицируются:

а) Помещения с повышенной опасностью.

б) Особо опасные помещения.

в) Помещения без повышенной опасности.

Помещения, в которых эксплуатируется прибор автоматизированного контроля электрических цепей изделий, относятся к помещениям с повышенной опасностью, характеризующихся наличием одного из следующих условий, создающих повышенную опасность:

а) Сырости или токопроводящей пыли (относительная влажность более 75%).

б) Токопроводящих полов (металлических, земляных, железобетонных, кирпичных и т.д.).

в) Высокой температуры (выше 35°С).

г) Возможности одновременного прикосновения человека к металлическим корпусам электрооборудования с одной стороны и к металлическим конструкциям зданий, технологическим аппаратам, механизмам и т.д., имеющим соединение с землей, с другой стороны.

Помещения, в которых эксплуатируется прибор автоматизированного контроля электрических цепей изделий, характеризуются наличием условия (пункта 4), создающего повышенную опасность.

К защитным мерам от опасности прикосновения к токоведущим частям электроустановок относятся: изоляция, ограждение, блокировка, пониженные напряжения, электрозащитные средства, сигнализация и плакаты.

Все изолирующие защитные средства подвергаются электрическим испытаниям, определенным напряжением в определенные сроки, оговоренные правилами, и могут применяться только при наличии на них штампа

электротехнической лаборатории об их испытании. Защитные средства с просроченным сроком проверки применять запрещается и они должны быть изъяты с рабочих мест.

Основными называются средства, изоляция которых длительно выдерживает рабочее напряжение электроустановок и, которые позволяют прикасаться к токоведущим частям, находящимися под напряжением.

При эксплуатации прибор автоматизированного контроля электрических цепей изделий используются:

а) указатели напряжения;

б) диэлектрические перчатки;

в) плакаты и знаки безопасности.

Электрозащитными средствами следует пользоваться по их прямому назначению в электроустановках напряжением не выше того, на которое они рассчитаны.

Перед работой персонал должен:

а) проверить их на исправность;

б) отсутствие внешних повреждений;

в) срок годности по штампу.

Для предупреждения персонала о наличии напряжения или его отсутствия в электроустановках применяется звуковая или световая сигнализация.

Наряду с применением технических методов и средств электробезопасности, важное значение для снижения электротравматизма имеет четкая организация эксплуатации электроустановок и электросетей, профессиональная подготовка работников, сознательная производственная и трудовая дисциплина.

7.3 Микроклимат производственных помещений

При эксплуатации прибора автоматизированного контроля электрических цепей изделий выполняемые работы в соответствии СанПиН 2.2.4.548-96 относятся к категории лёгких работ IIа, с затратами энергии 140-174 ккал/ч , сопровождающиеся незначительным физическим напряжением (работа в закрытом помещении, сидя за столом, перенос устройства на расстояние не более 5 м).

При обеспечении оптимальных и допустимых показателей микроклимата в холодное время года следует применять средства защиты рабочих мест от радиационного охлаждения и от остекленных поверхностей оконных проемов, в теплый период года – от попадания прямых солнечных лучей.

7.4 Производственное освещение

При организации производственного освещения рабочего места, на котором осуществляется эксплуатация прибора автоматизированного контроля электрических цепей изделий, необходимо обеспечить равномерное распределение яркости на рабочей поверхности и окружающих предметах. Перевод с ярко освещенной на слабо освещенную поверхность вынуждает глаз переадаптироваться, что ведет к утомлению зрения и соответственно к снижению производительности труда. Неправильное направление света на рабочее место может создать резкие тени, блики и дезориентировать работающего.

Производственные помещения для эксплуатации прибора автоматизированного контроля электрических цепей изделий должны иметь естественное и искусственное освещение, светлую окраску потолка, стен и оборудования. Освещенность должна быть не менее 200-300 лк.

Искусственное освещение в производственных помещениях должно осуществляться системой общего равномерного освещения (допускается использование местного освещения, предназначенного для освещения рабочей зоны расположения элементов).

7.5 Вибрационная безопасность

В соответствии с ГОСТ 24346-80 (СТ СЭВ 1926-79) «Вибрация. Термины и определения» под вибрацией понимается движение точки или механической системы, при котором происходит поочередное возрастание и убывание во времени значений по крайней мере одной координаты. Причиной возбуждения вибраций являются возникающие при работе машин и агрегатов неуравновешенные силовые воздействия. Наличие дисбаланса приводит к появлению неуравновешенных сил, вызывающих вибрацию.

Основными параметрами вибрации, происходящей по синусоидальному закону, являются: частота, амплитуда смещения, скорость, ускорение, период колебания (время, в течение которого совершается одно полное колебание).

В производственных условиях почти не встречается вибрации в виде простых гармонических колебаний. При работе машин и оборудования обычно возникает сложное колебательное движение, которое является апериодическим или квазипериодическим, имеющим импульсный или толчкообразный характер. В нашем приборе вибрацию создает лабораторный испытательный стенд, который под действием вибрации проверяет размыкание(замыкание) цепи.

Причинами вибрации могут быть неправильная установка и эксплуатация машин и оборудования, неравномерный износ отдельных узлов.

Вибродемпфирование производится с помощью использования композиционных материалов: сталь - алюминий, сталь - медь, а также пластмасс, древесины или резины. Широкое распространение получили вибродемпфирующие покрытия, которые в зависимости от величины динамического модуля упругости подразделяются на *жесткие* и *мягкие.* Первые эффективны в области низких частот, вторые - высоких.

К мягким вибродемпфирующим покрытиям относятся мягкие пластмассы, резины, пенопласт и др. В приборе автоматизированного контроля используется мягкое вибродемпфирующее покрытие в виде прорезиненных ножек.

7.6 Внешний вид прибора

Прибор автоматизированного контроля электрических цепей изделий соответствует требованиям ГОСТ 12.2.007.0–75 ССБТ Изделие электротехническое.

В приборе автоматизированного контроля используется:

– рабочая изоляция токоведущих частей;

– безопасное сверхнизкое напряжение в электрических цепях;

– элементы для осуществления защитного зануления металлических нетоковедущих частей изделия, которые могут оказаться под напряжением (при нарушении изоляции, режима работы изделия и т.п.);

Устанавливаются пять классов защиты: 0; 0I; I; II; III. В приборе автоматизированного контроля электрических цепей изделий используется первый (I) класс защиты, к которому относятся изделия, имеющие по крайней мере рабочую изоляцию и элемент для зануления. Т.к. прибор имеет провод для присоединения к источнику питания, этот провод должен иметь зануляющую жилу и вилку с зануляющим контактом.

Прибор автоматизированного контроля электрических цепей изделий не является источником шума , вибрации, электромагнитных полей, теплового оптического рентгеновского излучения.

Электрическая схема прибора исключает возможность его самопроизвольного включения и отключения. Расположение и соединение частей прибора выполнены с учетом удобства и безопасности наблюдения за изделием при выполнении сборочных работ, проведении осмотра, испытаний и обслуживания.

Конструкция прибора исключает возможность неправильного присоединения его сочленяемых токоведущих частей при монтаже изделий у потребителя. Конструкция штепсельных розеток и вилок для напряжений выше 42 В отличается от конструкции розеток и вилок для напряжений 42 В и менее.

Прибор оборудован надписями и табличками. Предупредительные сигналы, надписи и таблички применяются для указания на: включенное состояние изделия, наличие напряжения, пробой изоляции, режим работы изделия, запрет доступа внутрь изделия без принятия соответствующих мер, действие аппаратов защиты. Знаки, используемые при выполнении предупредительных табличек и сигнализации, выполняются по ГОСТ 12.4.024-74* и размещаются на изделиях в местах, удобных для обзора.

Требования к защитному занулению. В приборе автоматизированного контроля используется болт. Элемент для присоединения зануляющего проводника выполнен из металла, стойкого в отношении коррозии, и контактная часть не имеет поверхностной окраски. Размещен на изделии в безопасном и удобном для подключения зануляющего проводника месте.

Органы управления на рабочей поверхности прибора размещены с учетом требований ГОСТ 12.2.032-78 ССБТ "Рабочее место при выполнении работ сидя. Общие эргономические требования". Наиболее важные органы управления прибора располагаются в ближней зоне рабочего.

Конструкция органов управления учитывает:

а) требуемую точность и скорость движений при осуществлении управления, а также частоту использования органа управления;

б) допустимые динамические и статические нагрузки на двигательный аппарат человека;

в) антропометрические характеристики двигательного аппарата человека;

г) необходимость быстрого распознавания органов управления, формирования и закрепления навыков по управлению.

Места возможных контактов органов управления с руками работающего выполнены из нетоксичных и нетеплопроводных материалов. Их форма и

размеры обеспечивают надежный захват руками. Все тумблеры выполнены в однозначном исполнении. При этом состоянию “ВКЛЮЧЕНО” соответствует верхнее положение переключателя, “ВЫКЛЮЧЕНО” - нижнее. Пользование органами ручного управления и регулировки в последовательности, отличной от установленной в технической документации, сопровождается включением УЗО.

Элементы индикации и отображения информации располагаются интуитивно понятно, исходя из удобства функционирования, снабжаются поясняющими надписями и рисунками, указывающими их назначение или выполняемую функцию.

Для кнопок и разъемов выполняются поясняющие рисунки и соответствующие надписи. Для обозначения состояния питания предусмотрен индикатор, указывающий состояние устройства: «ВКЛЮЧЕН/ВЫКЛЮЧЕН».

Основным средством отображения информации служит дисплей, отображаемые символы и графические знаки на котором должны быть понятны любому работающему со прибором человеку. Для этого дисплей имеет аппаратные настройки, которые настраиваются для каждого обучаемого отдельно перед началом работы. Также может изменяться разрешение и размер текста, если имеются соответствующие отклонения от нормы зрения у испытуемого. Дисплей устанавливается в центре прибора, для того, чтобы при работе не возникало потребности в повороте головы и дополнительной концентрации зрения. Границы дисплея располагаются в вертикальной плоскости под углами не превышающими ±30° от нормальной линии взгляда и в горизонтальной плоскости под углом ±30° от сагиттальной плоскости, что обеспечивает комфортные условия при работе с ним.

Компоновка лицевой панели прибора соответствует требованиям расположения элементов электрических схем и выполняется согласно ГОСТ 12.2.049-80 «Оборудование производственное. Общие эргономические требования».

ЗАКЛЮЧЕНИЕ

После проведения комплекса работ по разработке прибора автоматизированного контроля электрических цепей изделий был создан комплект конструкторской документации необходимой для производства данного устройства.

При разработке схемы электрической принципиальной был произведен расчет и подбор различных элементов схемы. Путем подбора резисторов и конденсаторов, входящих в состав активного фильтра, произведена настройка активного фильтра на частоту полезного сигнала, что позволило избежать возможного влияния помех на правильность работы схемы. Так же, подбором резисторов блока «А» проедена настройка режима работы транзистора по напряжению и по току.

После подбора элементной базы была разработана программная часть для ПЛИС и МК. Далее была построена схема электрическая принципиальная, при помощи средств САПР AutoCAD 2014. Так же были построены алгоритм работы прибора на один канал, структурная схема прибора и сетевой график.

Экономические расчеты, проведенные по окончанию разработки печатной платы, позволили определить как затраты на разработку и создание нового устройства, так и срок окупаемости устройства. Подсчитав затраты на изготовление нового устройства его цена составляет 19772,26 рублей. По результатам расчетов устройство полностью окупится через 1,59 года, или 1 год 6 месяцев.

В конце работы были определены возможные опасные и вредные производственные факторы способные повлиять на жизнь и здоровье оператора прибора автоматизированного контроля электрических цепей изделия. Исходя из данных о возможных физических нагрузках, была определена категория работ. По физическим нагрузкам работа с прибором автоматизированного контроля электрических цепей изделия относится к работам категории I. По зрительной нагрузке – 3-й разряд.

БИБЛИОГРАФИЧЕСКИЙ СПИСОК

1) AVR. Учебный курс, – http://easyelectronics.ru/.

2) Основы языка VHDL, – http://allhdl.ru/.

3) Алексенко, А.Г. Применение прецизионных аналоговых микросхем. – 2 – е изд. – М.: Радио и связь, 1985. – 256 с.: ил.

4) Брежнева, К.М., Гантман, Е.И., Т.И. Давыдова, Т.И. Транзисторы для аппаратуры широкого применения: Справочник. – М.: Радио и связь,1981. – 656 с.: ил.

5) Платан Активные компоненты: Справочник. – 2011. – 107 с.

6) Платан Оптоэлектроника: Справочник. – 2011. – 199 с.

7) Платан Пассивные компоненты: Справочник. – 2011. – 164 с.

8) Бородянко, В.Н, Дубовицкий, Г.П., Коголь ,И.М. Электротехника: Учебно – методический комплекс – Offline версия 2.2. – Челябинск, 2008. – 3202 файлов.

9) Харотов, В.Я. Микроконтроллеры AVR. Практикум для начинающих. – М.: Изд – во МГТУ им Н.Э. Баумана, 2007 – 240 с.: ил.

10) Хоровиц, П., Хилл, У. Искусство схемотехники, Т. 1. – 4 – е изд. – М.: Мир, 1993. – 413 с.: ил.

11) Хоровиц, П., Хилл, У. Искусство схемотехники, Т. 2. – 4 – е изд. – М.: Мир, 1993. – 371 с.: ил.

12) Хоровиц, П., Хилл, У. Искусство схемотехники, Т. 3. – 4 – е изд. – М.: Мир, 1993. – 367 с.: ил.

13) Ши, Р.Ф. Расчет транзисторных цепей. – М.: Энергия, 1964. – 264 с.: ил.

ПРИЛОЖЕНИЕ А. Программа работы ПЛИС

```
-- Подключаем стандартные библиотеки
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_unsigned.all;
use IEEE.std_logic_arith.all;
---------------------------------------------
entity nextlevel1 is – задаем блок  nextlev1
port (a1, b1,  clk,  rst1 : in std_logic;  -- Обозначаем порты на вход
count : out std_logic_vector (2 downto 0)); -- Обозначаем порты на выход
--ex1:out std_logic_vector (2 downto 0)); -- Обозначаем порты на выход
end nextlevel1; -- завершаем nextlevel1
---------------------------------------------
architecture behav1 of nextlevel1 is – называем архитектуру behav1 и пишем архитектуру
nextlevel1
signal cnk1: std_logic_vector (2 downto 0);  -- задаем тип сигналу cnk1 std_logic_vector
signal cnt1: std_logic_vector (2 downto 0);  -- задаем тип сигналу cnt1 std_logic_vector
signal q1 : std_logic_vector (2 downto 0); -- задаем тип сигналу q1 std_logic_vector
signal en1 : std_logic; -- задаем тип сигналу en1 std_logic
--signal count1 :  std_logic_vector (2 downto 0); -- задаем тип счетчику
--signal exq1 :  std_logic_vector (2 downto 0);
---------------------------------------------
begin -- переход
process (q1 , cnt1 , cnk1 , clk1 , en1 , rst1, a1, b1) --, exq1  , count) – работаем с сигналами
описываемые процессом
begin -- переход
---------------------------------------------
 En1 <= a1 xor b1; -- задаем дополнительный вход
if (rst 1= '1') then – в случаее если сигнал сброса = '1'
cnk1 <= (others1 => '0'); -- тогда на выходе 0
elsif (rising_edge(clk)) then
if (en1 = '1') then
cnk1 <= cnk1 + '1'; -- прописываем счетчик
end if;
```

```
if (cnk1 = "0101") then
cnt1 <= (others => '0');
end if;
end if;
-----------------------------------------------
if (q1 = "0101") then -- задаем значаение до скольки должен считать счетчик
cnt <= (others => '0');
end if;
if (q 1> "0101") then
cnt <= (others => '1');
end if;
-----------------------------------------------
--if (cnt1 = "1111")then -- задаем значение при котором на выходе появл лог. еденица
--exq <= (others => '1');
--end if;
-----------------------------------------------
end process;
count1 <= cnt1; -- записываем результаты на определенные сигналы
q1 <= cnk1;
--ex1<=exq;
end behav1;


-----------------------------------------------
entity nextlevel1 is – задаем блок  nextlev1
port (a1, b1,  clk,  rst1 : in std_logic;  -- Обозначаем порты на вход
count : out std_logic_vector (2 downto 0)); -- Обозначаем порты на выход
--ex1:out std_logic_vector (2 downto 0)); -- Обозначаем порты на выход
end nextlevel1; -- завершаем nextlevel1
-----------------------------------------------
architecture behav1 of nextlevel1 is – называем архитектуру behav1 и пишем архитектуру
nextlevel1
signal cnk1: std_logic_vector (2 downto 0);  -- задаем тип сигналу cnk1 std_logic_vector
signal cnt1: std_logic_vector (2 downto 0);  -- задаем тип сигналу cnt1 std_logic_vector
signal q1 : std_logic_vector (2 downto 0); -- задаем тип сигналу q1 std_logic_vector
```

```
signal en1 : std_logic; -- задаем тип сигналу en1 std_logic
--signal count1 :  std_logic_vector (2 downto 0); -- задаем тип счетчику
--signal exq1 :  std_logic_vector (2 downto 0);
-----------------------------------------------
begin -- переход
process (q1 , cnt1 , cnk1 , clk1 , en1 , rst1, a1, b1) --, exq1  , count) – работаем с сигналами
описываемые процессом
begin -- переход
-----------------------------------------------
 En1 <= a1 xor b1; -- задаем дополнительный вход
if (rst 1= '1') then – в случаее если сигнал сброса = '1'
cnk1 <= (others1 => '0'); -- тогда на выходе 0
elsif (rising_edge(clk)) then
if (en1 = '1') then
cnk1 <= cnk1 + '1'; -- прописываем счетчик
end if;
if (cnk1 = "0101") then
cnt1 <= (others => '0');
end if;
end if;
-----------------------------------------------
if (q1 = "0101") then -- задаем значаение до скольки должен считать счетчик
cnt <= (others => '0');
end if;
if (q 1> "0101") then
cnt <= (others => '1');
end if;
-----------------------------------------------
--if (cnt1 = "1111")then -- задаем значение при котором на выходе появл лог. еденица
--exq <= (others => '1');
--end if;
-----------------------------------------------
end process;
count1 <= cnt1; -- записываем результаты на определенные сигналы
```

```
q1 <= cnk1;
--ex1<=exq;
end behav1;
-----------------------------------------------
entity nextlevel1 is – задаем блок  nextlev1
port (a1, b1,  clk,  rst1 : in std_logic;  -- Обозначаем порты на вход
count : out std_logic_vector (2 downto 0)); -- Обозначаем порты на выход
--ex1:out std_logic_vector (2 downto 0)); -- Обозначаем порты на выход
end nextlevel1; -- завершаем nextlevel1
-----------------------------------------------
architecture behav1 of nextlevel1 is – называем архитектуру behav1 и пишем архитектуру
nextlevel1
signal cnk1: std_logic_vector (2 downto 0);  -- задаем тип сигналу cnk1 std_logic_vector
signal cnt1: std_logic_vector (2 downto 0);  -- задаем тип сигналу cnt1 std_logic_vector
signal q1 : std_logic_vector (2 downto 0); -- задаем тип сигналу q1 std_logic_vector
signal en1 : std_logic; -- задаем тип сигналу en1 std_logic
--signal count1 :  std_logic_vector (2 downto 0); -- задаем тип счетчику
--signal exq1 :  std_logic_vector (2 downto 0);
-----------------------------------------------
begin -- переход
process (q1 , cnt1 , cnk1 , clk1 , en1 , rst1, a1, b1) --, exq1  , count) – работаем с сигналами
описываемые процессом
begin -- переход
-----------------------------------------------
 En1 <= a1 xor b1; -- задаем дополнительный вход
if (rst 1= '1') then – в случаее если сигнал сброса = '1'
cnk1 <= (others1 => '0'); -- тогда на выходе 0
elsif (rising_edge(clk)) then
if (en1 = '1') then
cnk1 <= cnk1 + '1'; -- прописываем счетчик
end if;
if (cnk1 = "0101") then
cnt1 <= (others => '0');
end if;
```

```
end if;
-----------------------------------------
if (q1 = "0101") then -- задаем значаение до скольки должен считать счетчик
cnt <= (others => '0');
end if;
if (q 1> "0101") then
cnt <= (others => '1');
end if;
-----------------------------------------
--if (cnt1 = "1111")then -- задаем значение при котором на выходе появл лог. еденица
--exq <= (others => '1');
--end if;
-----------------------------------------
end process;
count1 <= cnt1; -- записываем результаты на определенные сигналы
q1 <= cnk1;
--ex1<=exq;
end behav1;
-----------------------------------------
entity nextlevel1 is – задаем блок  nextlev1
port (a1, b1,  clk,  rst1 : in std_logic;  -- Обозначаем порты на вход
count : out std_logic_vector (2 downto 0)); -- Обозначаем порты на выход
--ex1:out std_logic_vector (2 downto 0)); -- Обозначаем порты на выход
end nextlevel1; -- завершаем nextlevel1
-----------------------------------------
architecture behav1 of nextlevel1 is – называем архитектуру behav1 и пишем архитектуру
nextlevel1
signal cnk1: std_logic_vector (2 downto 0);  -- задаем тип сигналу cnk1 std_logic_vector
signal cnt1: std_logic_vector (2 downto 0);  -- задаем тип сигналу cnt1 std_logic_vector
signal q1 : std_logic_vector (2 downto 0); -- задаем тип сигналу q1 std_logic_vector
signal en1 : std_logic; -- задаем тип сигналу en1 std_logic
--signal count1 :  std_logic_vector (2 downto 0); -- задаем тип счетчику
--signal exq1 :  std_logic_vector (2 downto 0);
-----------------------------------------
```

```
begin -- переход
process (q1 , cnt1 , cnk1 , clk1 , en1 , rst1, a1, b1) --, exq1  , count) – работаем с сигналами описываемые процессом
begin -- переход
-----------------------------------------------
 En1 <= a1 xor b1; -- задаем дополнительный вход
if (rst 1= '1') then – в случаее если сигнал сброса = '1'
cnk1 <= (others1 => '0'); -- тогда на выходе 0
elsif (rising_edge(clk)) then
if (en1 = '1') then
cnk1 <= cnk1 + '1'; -- прописываем счетчик
end if;
if (cnk1 = "0101") then
cnt1 <= (others => '0');
end if;
end if;
-----------------------------------------------
if (q1 = "0101") then -- задаем значаение до скольки должен считать счетчик
cnt <= (others => '0');
end if;
if (q 1> "0101") then
cnt <= (others => '1');
end if;
-----------------------------------------------
--if (cnt1 = "1111")then -- задаем значение при котором на выходе появл лог. еденица
--exq <= (others => '1');
--end if;
-----------------------------------------------
end process;
count1 <= cnt1; -- записываем результаты на определенные сигналы
q1 <= cnk1;
--ex1<=exq;
end behav1;
-----------------------------------------------
```

```
entity nextlevel1 is – задаем блок  nextlev1
port (a1, b1,  clk,  rst1 : in std_logic;  -- Обозначаем порты на вход
count : out std_logic_vector (2 downto 0)); -- Обозначаем порты на выход
--ex1:out std_logic_vector (2 downto 0)); -- Обозначаем порты на выход
end nextlevel1; -- завершаем nextlevel1
---------------------------------------------
architecture behav1 of nextlevel1 is – называем архитектуру behav1 и пишем архитектуру
nextlevel1
signal cnk1: std_logic_vector (2 downto 0);  -- задаем тип сигналу cnk1 std_logic_vector
signal cnt1: std_logic_vector (2 downto 0);  -- задаем тип сигналу cnt1 std_logic_vector
signal q1 : std_logic_vector (2 downto 0); -- задаем тип сигналу q1 std_logic_vector
signal en1 : std_logic; -- задаем тип сигналу en1 std_logic
--signal count1 :  std_logic_vector (2 downto 0); -- задаем тип счетчику
--signal exq1 :  std_logic_vector (2 downto 0);
---------------------------------------------
begin -- переход
process (q1 , cnt1 , cnk1 , clk1 , en1 , rst1, a1, b1) --, exq1  , count) – работаем с сигналами
описываемые процессом
begin -- переход
---------------------------------------------
 En1 <= a1 xor b1; -- задаем дополнительный вход
if (rst 1= '1') then – в случаее если сигнал сброса = '1'
cnk1 <= (others1 => '0'); -- тогда на выходе 0
elsif (rising_edge(clk)) then
if (en1 = '1') then
cnk1 <= cnk1 + '1'; -- прописываем счетчик
end if;
if (cnk1 = "0101") then
cnt1 <= (others => '0');
end if;
end if;
---------------------------------------------
if (q1 = "0101") then -- задаем значаение до скольки должен считать счетчик
cnt <= (others => '0');
```

```
end if;
if (q 1> "0101") then
cnt <= (others => '1');
end if;
-----------------------------------------------
--if (cnt1 = "1111")then -- задаем значение при котором на выходе появл лог. еденица
--exq <= (others => '1');
--end if;
-----------------------------------------------
end process;
count1 <= cnt1; -- записываем результаты на определенные сигналы
q1 <= cnk1;
--ex1<=exq;
end behav1;
-----------------------------------------------
entity nextlevel1 is – задаем блок  nextlev1
port (a1, b1,  clk,  rst1 : in std_logic;  -- Обозначаем порты на вход
count : out std_logic_vector (2 downto 0)); -- Обозначаем порты на выход
--ex1:out std_logic_vector (2 downto 0)); -- Обозначаем порты на выход
end nextlevel1; -- завершаем nextlevel1
-----------------------------------------------
architecture behav1 of nextlevel1 is – называем архитектуру behav1 и пишем архитектуру
nextlevel1
signal cnk1: std_logic_vector (2 downto 0);  -- задаем тип сигналу cnk1 std_logic_vector
signal cnt1: std_logic_vector (2 downto 0);  -- задаем тип сигналу cnt1 std_logic_vector
signal q1 : std_logic_vector (2 downto 0); -- задаем тип сигналу q1 std_logic_vector
signal en1 : std_logic; -- задаем тип сигналу en1 std_logic
--signal count1 :  std_logic_vector (2 downto 0); -- задаем тип счетчику
--signal exq1 :  std_logic_vector (2 downto 0);
-----------------------------------------------
begin -- переход
process (q1 , cnt1 , cnk1 , clk1 , en1 , rst1, a1, b1) --, exq1  , count) – работаем с сигналами
описываемые процессом
begin -- переход
```

```
-----------------------------------------------
 En1 <= a1 xor b1; -- задаем дополнительный вход
if (rst 1= '1') then – в случаее если сигнал сброса = '1'
cnk1 <= (others1 => '0'); -- тогда на выходе 0
elsif (rising_edge(clk)) then
if (en1 = '1') then
cnk1 <= cnk1 + '1'; -- прописываем счетчик
end if;
if (cnk1 = "0101") then
cnt1 <= (others => '0');
end if;
end if;
-----------------------------------------------
if (q1 = "0101") then -- задаем значаение до скольки должен считать счетчик
cnt <= (others => '0');
end if;
if (q 1> "0101") then
cnt <= (others => '1');
end if;
-----------------------------------------------
--if (cnt1 = "1111")then -- задаем значение при котором на выходе появл лог. еденица
--exq <= (others => '1');
--end if;
-----------------------------------------------
end process;
count1 <= cnt1; -- записываем результаты на определенные сигналы
q1 <= cnk1;
--ex1<=exq;
end behav1;
-----------------------------------------------
entity nextlevel1 is – задаем блок  nextlev1
port (a1, b1,  clk,  rst1 : in std_logic;  -- Обозначаем порты на вход
count : out std_logic_vector (2 downto 0)); -- Обозначаем порты на выход
--ex1:out std_logic_vector (2 downto 0)); -- Обозначаем порты на выход
```

```
end nextlevel1; -- завершаем nextlevel1
-----------------------------------------
architecture behav1 of nextlevel1 is – называем архитектуру behav1 и пишем архитектуру nextlevel1
signal cnk1: std_logic_vector (2 downto 0);  -- задаем тип сигналу cnk1 std_logic_vector
signal cnt1: std_logic_vector (2 downto 0);  -- задаем тип сигналу cnt1 std_logic_vector
signal q1 : std_logic_vector (2 downto 0); -- задаем тип сигналу q1 std_logic_vector
signal en1 : std_logic; -- задаем тип сигналу en1 std_logic
--signal count1 :  std_logic_vector (2 downto 0); -- задаем тип счетчику
--signal exq1 :  std_logic_vector (2 downto 0);
-----------------------------------------
begin -- переход
process (q1 , cnt1 , cnk1 , clk1 , en1 , rst1, a1, b1) --, exq1  , count) – работаем с сигналами описываемые процессом
begin -- переход
-----------------------------------------
 En1 <= a1 xor b1; -- задаем дополнительный вход
if (rst 1= '1') then – в случаее если сигнал сброса = '1'
cnk1 <= (others1 => '0'); -- тогда на выходе 0
elsif (rising_edge(clk)) then
if (en1 = '1') then
cnk1 <= cnk1 + '1'; -- прописываем счетчик
end if;
if (cnk1 = "0101") then
cnt1 <= (others => '0');
end if;
end if;
-----------------------------------------
if (q1 = "0101") then -- задаем значаение до скольки должен считать счетчик
cnt <= (others => '0');
end if;
if (q 1> "0101") then
cnt <= (others => '1');
end if;
```

```
-----------------------------------------------
--if (cnt1 = "1111")then -- задаем значение при котором на выходе появл лог. еденица
--exq <= (others => '1');
--end if;
-----------------------------------------------
end process;
count1 <= cnt1; -- записываем результаты на определенные сигналы
q1 <= cnk1;
--ex1<=exq;
end behav1;
-----------------------------------------------
entity nextlevel1 is – задаем блок  nextlev1
port (a1, b1,  clk,  rst1 : in std_logic;  -- Обозначаем порты на вход
count : out std_logic_vector (2 downto 0)); -- Обозначаем порты на выход
--ex1:out std_logic_vector (2 downto 0)); -- Обозначаем порты на выход
end nextlevel1; -- завершаем nextlevel1
-----------------------------------------------
architecture behav1 of nextlevel1 is – называем архитектуру behav1 и пишем архитектуру
nextlevel1
signal cnk1: std_logic_vector (2 downto 0);  -- задаем тип сигналу cnk1 std_logic_vector
signal cnt1: std_logic_vector (2 downto 0);  -- задаем тип сигналу cnt1 std_logic_vector
signal q1 : std_logic_vector (2 downto 0); -- задаем тип сигналу q1 std_logic_vector
signal en1 : std_logic; -- задаем тип сигналу en1 std_logic
--signal count1 :  std_logic_vector (2 downto 0); -- задаем тип счетчику
--signal exq1 :  std_logic_vector (2 downto 0);
-----------------------------------------------
begin -- переход
process (q1 , cnt1 , cnk1 , clk1 , en1 , rst1, a1, b1) --, exq1  , count) – работаем с сигналами
описываемые процессом
begin -- переход
-----------------------------------------------
 En1 <= a1 xor b1; -- задаем дополнительный вход
if (rst 1= '1') then – в случаее если сигнал сброса = '1'
cnk1 <= (others1 => '0'); -- тогда на выходе 0
```

```
elsif (rising_edge(clk)) then
if (en1 = '1') then
cnk1 <= cnk1 + '1'; -- прописываем счетчик
end if;
if (cnk1 = "0101") then
cnt1 <= (others => '0');
end if;
end if;
-----------------------------------------
if (q1 = "0101") then -- задаем значаение до скольки должен считать счетчик
cnt <= (others => '0');
end if;
if (q 1> "0101") then
cnt <= (others => '1');
end if;
-----------------------------------------
--if (cnt1 = "1111")then -- задаем значение при котором на выходе появл лог. еденица
--exq <= (others => '1');
--end if;
-----------------------------------------
end process;
count1 <= cnt1; -- записываем результаты на определенные сигналы
q1 <= cnk1;
--ex1<=exq;
end behav1;
-----------------------------------------
entity nextlevel1 is – задаем блок  nextlev1
port (a1, b1,  clk,  rst1 : in std_logic;  -- Обозначаем порты на вход
count : out std_logic_vector (2 downto 0)); -- Обозначаем порты на выход
--ex1:out std_logic_vector (2 downto 0)); -- Обозначаем порты на выход
end nextlevel1; -- завершаем nextlevel1
-----------------------------------------
architecture behav1 of nextlevel1 is – называем архитектуру behav1 и пишем архитектуру
nextlevel1
```

```
signal cnk1: std_logic_vector (2 downto 0);  -- задаем тип сигналу cnk1 std_logic_vector
signal cnt1: std_logic_vector (2 downto 0);  -- задаем тип сигналу cnt1 std_logic_vector
signal q1 : std_logic_vector (2 downto 0); -- задаем тип сигналу q1 std_logic_vector
signal en1 : std_logic; -- задаем тип сигналу en1 std_logic
--signal count1 :  std_logic_vector (2 downto 0); -- задаем тип счетчику
--signal exq1 :  std_logic_vector (2 downto 0);
-----------------------------------------------
begin -- переход
process (q1 , cnt1 , cnk1 , clk1 , en1 , rst1, a1, b1) --, exq1  , count) – работаем с сигналами
описываемые процессом
begin -- переход
-----------------------------------------------
 En1 <= a1 xor b1; -- задаем дополнительный вход
if (rst 1= '1') then – в случаее если сигнал сброса = '1'
cnk1 <= (others1 => '0'); -- тогда на выходе 0
elsif (rising_edge(clk)) then
if (en1 = '1') then
cnk1 <= cnk1 + '1'; -- прописываем счетчик
end if;
if (cnk1 = "0101") then
cnt1 <= (others => '0');
end if;
end if;
-----------------------------------------------
if (q1 = "0101") then -- задаем значаение до скольки должен считать счетчик
cnt <= (others => '0');
end if;
if (q 1> "0101") then
cnt <= (others => '1');
end if;
-----------------------------------------------
--if (cnt1 = "1111")then -- задаем значение при котором на выходе появл лог. еденица
--exq <= (others => '1');
--end if;
```

```
-----------------------------------------------
end process;
count1 <= cnt1; -- записываем результаты на определенные сигналы
q1 <= cnk1;
--ex1<=exq;
end behav1;
-----------------------------------------------
entity nextlevel1 is – задаем блок  nextlev1
port (a1, b1,  clk,  rst1 : in std_logic;  -- Обозначаем порты на вход
count : out std_logic_vector (2 downto 0)); -- Обозначаем порты на выход
--ex1:out std_logic_vector (2 downto 0)); -- Обозначаем порты на выход
end nextlevel1; -- завершаем nextlevel1
-----------------------------------------------
architecture behav1 of nextlevel1 is – называем архитектуру behav1 и пишем архитектуру
nextlevel1
signal cnk1: std_logic_vector (2 downto 0);  -- задаем тип сигналу cnk1 std_logic_vector
signal cnt1: std_logic_vector (2 downto 0);  -- задаем тип сигналу cnt1 std_logic_vector
signal q1 : std_logic_vector (2 downto 0); -- задаем тип сигналу q1 std_logic_vector
signal en1 : std_logic; -- задаем тип сигналу en1 std_logic
--signal count1 :  std_logic_vector (2 downto 0); -- задаем тип счетчику
--signal exq1 :  std_logic_vector (2 downto 0);
-----------------------------------------------
begin -- переход
process (q1 , cnt1 , cnk1 , clk1 , en1 , rst1, a1, b1) --, exq1  , count) – работаем с сигналами
описываемые процессом
begin -- переход
-----------------------------------------------
 En1 <= a1 xor b1; -- задаем дополнительный вход
if (rst 1= '1') then – в случаее если сигнал сброса = '1'
cnk1 <= (others1 => '0'); -- тогда на выходе 0
elsif (rising_edge(clk)) then
if (en1 = '1') then
cnk1 <= cnk1 + '1'; -- прописываем счетчик
end if;
```

```
if (cnk1 = "0101") then
cnt1 <= (others => '0');
end if;
end if;
-----------------------------------------------
if (q1 = "0101") then -- задаем значаение до скольки должен считать счетчик
cnt <= (others => '0');
end if;
if (q 1> "0101") then
cnt <= (others => '1');
end if;
-----------------------------------------------
--if (cnt1 = "1111")then -- задаем значение при котором на выходе появл лог. еденица
--exq <= (others => '1');
--end if;
-----------------------------------------------
end process;
count1 <= cnt1; -- записываем результаты на определенные сигналы
q1 <= cnk1;
--ex1<=exq;
end behav1;
-----------------------------------------------
entity nextlevel1 is – задаем блок  nextlev1
port (a1, b1,  clk,  rst1 : in std_logic;  -- Обозначаем порты на вход
count : out std_logic_vector (2 downto 0)); -- Обозначаем порты на выход
--ex1:out std_logic_vector (2 downto 0)); -- Обозначаем порты на выход
end nextlevel1; -- завершаем nextlevel1
-----------------------------------------------
architecture behav1 of nextlevel1 is – называем архитектуру behav1 и пишем архитектуру
nextlevel1
signal cnk1: std_logic_vector (2 downto 0);  -- задаем тип сигналу cnk1 std_logic_vector
signal cnt1: std_logic_vector (2 downto 0);  -- задаем тип сигналу cnt1 std_logic_vector
signal q1 : std_logic_vector (2 downto 0); -- задаем тип сигналу q1 std_logic_vector
signal en1 : std_logic; -- задаем тип сигналу en1 std_logic
```

```
--signal count1 :  std_logic_vector (2 downto 0); -- задаем тип счетчику
--signal exq1 :  std_logic_vector (2 downto 0);
-----------------------------------------
begin -- переход
process (q1 , cnt1 , cnk1 , clk1 , en1 , rst1, a1, b1) --, exq1  , count) – работаем с сигналами
описываемые процессом
begin -- переход
-----------------------------------------
 En1 <= a1 xor b1; -- задаем дополнительный вход
if (rst 1= '1') then – в случаее если сигнал сброса = '1'
cnk1 <= (others1 => '0'); -- тогда на выходе 0
elsif (rising_edge(clk)) then
if (en1 = '1') then
cnk1 <= cnk1 + '1'; -- прописываем счетчик
end if;
if (cnk1 = "0101") then
cnt1 <= (others => '0');
end if;
end if;
-----------------------------------------
if (q1 = "0101") then -- задаем значаение до скольки должен считать счетчик
cnt <= (others => '0');
end if;
if (q 1> "0101") then
cnt <= (others => '1');
end if;
-----------------------------------------
--if (cnt1 = "1111")then -- задаем значение при котором на выходе появл лог. еденица
--exq <= (others => '1');
--end if;
-----------------------------------------
end process;
count1 <= cnt1; -- записываем результаты на определенные сигналы
q1 <= cnk1;
```

```
--ex1<=exq;
end behav1;
-----------------------------------------
entity nextlevel1 is – задаем блок  nextlev1
port (a1, b1,  clk,  rst1 : in std_logic;  -- Обозначаем порты на вход
count : out std_logic_vector (2 downto 0)); -- Обозначаем порты на выход
--ex1:out std_logic_vector (2 downto 0)); -- Обозначаем порты на выход
end nextlevel1; -- завершаем nextlevel1
-----------------------------------------
architecture behav1 of nextlevel1 is – называем архитектуру behav1 и пишем архитектуру
nextlevel1
signal cnk1: std_logic_vector (2 downto 0);  -- задаем тип сигналу cnk1 std_logic_vector
signal cnt1: std_logic_vector (2 downto 0);  -- задаем тип сигналу cnt1 std_logic_vector
signal q1 : std_logic_vector (2 downto 0); -- задаем тип сигналу q1 std_logic_vector
signal en1 : std_logic; -- задаем тип сигналу en1 std_logic
--signal count1 :  std_logic_vector (2 downto 0); -- задаем тип счетчику
--signal exq1 :  std_logic_vector (2 downto 0);
-----------------------------------------
begin -- переход
process (q1 , cnt1 , cnk1 , clk1 , en1 , rst1, a1, b1) --, exq1  , count) – работаем с сигналами
описываемые процессом
begin -- переход
-----------------------------------------
 En1 <= a1 xor b1; -- задаем дополнительный вход
if (rst 1= '1') then – в случаее если сигнал сброса = '1'
cnk1 <= (others1 => '0'); -- тогда на выходе 0
elsif (rising_edge(clk)) then
if (en1 = '1') then
cnk1 <= cnk1 + '1'; -- прописываем счетчик
end if;
if (cnk1 = "0101") then
cnt1 <= (others => '0');
end if;
end if;
```

```
-----------------------------------------------
if (q1 = "0101") then -- задаем значаение до скольки должен считать счетчик
cnt <= (others => '0');
end if;
if (q 1> "0101") then
cnt <= (others => '1');
end if;
-----------------------------------------------
--if (cnt1 = "1111")then -- задаем значение при котором на выходе появл лог. еденица
--exq <= (others => '1');
--end if;
-----------------------------------------------
end process;
count1 <= cnt1; -- записываем результаты на определенные сигналы
q1 <= cnk1;
--ex1<=exq;
end behav1;
-----------------------------------------------
entity nextlevel1 is – задаем блок  nextlev1
port (a1, b1,  clk,  rst1 : in std_logic;  -- Обозначаем порты на вход
count : out std_logic_vector (2 downto 0)); -- Обозначаем порты на выход
--ex1:out std_logic_vector (2 downto 0)); -- Обозначаем порты на выход
end nextlevel1; -- завершаем nextlevel1
-----------------------------------------------
architecture behav1 of nextlevel1 is – называем архитектуру behav1 и пишем архитектуру
nextlevel1
signal cnk1: std_logic_vector (2 downto 0);  -- задаем тип сигналу cnk1 std_logic_vector
signal cnt1: std_logic_vector (2 downto 0);  -- задаем тип сигналу cnt1 std_logic_vector
signal q1 : std_logic_vector (2 downto 0); -- задаем тип сигналу q1 std_logic_vector
signal en1 : std_logic; -- задаем тип сигналу en1 std_logic
--signal count1 :  std_logic_vector (2 downto 0); -- задаем тип счетчику
--signal exq1 :  std_logic_vector (2 downto 0);
-----------------------------------------------
begin -- переход
```

```
process (q1 , cnt1 , cnk1 , clk1 , en1 , rst1, a1, b1) --, exq1  , count) – работаем с сигналами описываемые процессом
begin -- переход
-----------------------------------------
 En1 <= a1 xor b1; -- задаем дополнительный вход
if (rst 1= '1') then – в случаее если сигнал сброса = '1'
cnk1 <= (others1 => '0'); -- тогда на выходе 0
elsif (rising_edge(clk)) then
if (en1 = '1') then
cnk1 <= cnk1 + '1'; -- прописываем счетчик
end if;
if (cnk1 = "0101") then
cnt1 <= (others => '0');
end if;
end if;
-----------------------------------------
if (q1 = "0101") then -- задаем значаение до скольки должен считать счетчик
cnt <= (others => '0');
end if;
if (q 1> "0101") then
cnt <= (others => '1');
end if;
-----------------------------------------
--if (cnt1 = "1111")then -- задаем значение при котором на выходе появл лог. еденица
--exq <= (others => '1');
--end if;
-----------------------------------------
end process;
count1 <= cnt1; -- записываем результаты на определенные сигналы
q1 <= cnk1;
--ex1<=exq;
end behav1;
-----------------------------------------
entity nextlevel1 is – задаем блок  nextlev1
```

```
port (a1, b1,  clk,  rst1 : in std_logic;  -- Обозначаем порты на вход
count : out std_logic_vector (2 downto 0)); -- Обозначаем порты на выход
--ex1:out std_logic_vector (2 downto 0)); -- Обозначаем порты на выход
end nextlevel1; -- завершаем nextlevel1
-----------------------------------------------
architecture behav1 of nextlevel1 is – называем архитектуру behav1 и пишем архитектуру nextlevel1
signal cnk1: std_logic_vector (2 downto 0);  -- задаем тип сигналу cnk1 std_logic_vector
signal cnt1: std_logic_vector (2 downto 0);  -- задаем тип сигналу cnt1 std_logic_vector
signal q1 : std_logic_vector (2 downto 0); -- задаем тип сигналу q1 std_logic_vector
signal en1 : std_logic; -- задаем тип сигналу en1 std_logic
--signal count1 :  std_logic_vector (2 downto 0); -- задаем тип счетчику
--signal exq1 :  std_logic_vector (2 downto 0);
-----------------------------------------------
begin -- переход
process (q1 , cnt1 , cnk1 , clk1 , en1 , rst1, a1, b1) --, exq1  , count) – работаем с сигналами описываемые процессом
begin -- переход
-----------------------------------------------
 En1 <= a1 xor b1; -- задаем дополнительный вход
if (rst 1= '1') then – в случаее если сигнал сброса = '1'
cnk1 <= (others1 => '0'); -- тогда на выходе 0
elsif (rising_edge(clk)) then
if (en1 = '1') then
cnk1 <= cnk1 + '1'; -- прописываем счетчик
end if;
if (cnk1 = "0101") then
cnt1 <= (others => '0');
end if;
end if;
-----------------------------------------------
if (q1 = "0101") then -- задаем значаение до скольки должен считать счетчик
cnt <= (others => '0');
end if;
```

```
if (q 1> "0101") then
cnt <= (others => '1');
end if;
-----------------------------------------------
--if (cnt1 = "1111")then -- задаем значение при котором на выходе появл лог. еденица
--exq <= (others => '1');
--end if;
-----------------------------------------------
end process;
count1 <= cnt1; -- записываем результаты на определенные сигналы
q1 <= cnk1;
--ex1<=exq;
end behav1;
-----------------------------------------------
entity nextlevel1 is – задаем блок  nextlev1
port (a1, b1,  clk,  rst1 : in std_logic;  -- Обозначаем порты на вход
count : out std_logic_vector (2 downto 0)); -- Обозначаем порты на выход
--ex1:out std_logic_vector (2 downto 0)); -- Обозначаем порты на выход
end nextlevel1; -- завершаем nextlevel1
-----------------------------------------------
architecture behav1 of nextlevel1 is – называем архитектуру behav1 и пишем архитектуру
nextlevel1
signal cnk1: std_logic_vector (2 downto 0);  -- задаем тип сигналу cnk1 std_logic_vector
signal cnt1: std_logic_vector (2 downto 0);  -- задаем тип сигналу cnt1 std_logic_vector
signal q1 : std_logic_vector (2 downto 0); -- задаем тип сигналу q1 std_logic_vector
signal en1 : std_logic; -- задаем тип сигналу en1 std_logic
--signal count1 :  std_logic_vector (2 downto 0); -- задаем тип счетчику
--signal exq1 :  std_logic_vector (2 downto 0);
-----------------------------------------------
begin -- переход
process (q1 , cnt1 , cnk1 , clk1 , en1 , rst1, a1, b1) --, exq1  , count) – работаем с сигналами
описываемые процессом
begin -- переход
-----------------------------------------------
```

```
 En1 <= a1 xor b1; -- задаем дополнительный вход
if (rst 1= '1') then – в случаее если сигнал сброса = '1'
cnk1 <= (others1 => '0'); -- тогда на выходе 0
elsif (rising_edge(clk)) then
if (en1 = '1') then
cnk1 <= cnk1 + '1'; -- прописываем счетчик
end if;
if (cnk1 = "0101") then
cnt1 <= (others => '0');
end if;
end if;
-----------------------------------------------
if (q1 = "0101") then -- задаем значаение до скольки должен считать счетчик
cnt <= (others => '0');
end if;
if (q 1> "0101") then
cnt <= (others => '1');
end if;
-----------------------------------------------
--if (cnt1 = "1111")then -- задаем значение при котором на выходе появл лог. еденица
--exq <= (others => '1');
--end if;
-----------------------------------------------
end process;
count1 <= cnt1; -- записываем результаты на определенные сигналы
q1 <= cnk1;
--ex1<=exq;
end behav1;
-----------------------------------------------
entity nextlevel1 is – задаем блок  nextlev1
port (a1, b1,  clk,  rst1 : in std_logic;  -- Обозначаем порты на вход
count : out std_logic_vector (2 downto 0)); -- Обозначаем порты на выход
--ex1:out std_logic_vector (2 downto 0)); -- Обозначаем порты на выход
end nextlevel1; -- завершаем nextlevel1
```

```
----------------------------------------------
architecture behav1 of nextlevel1 is – называем архитектуру behav1 и пишем архитектуру nextlevel1
signal cnk1: std_logic_vector (2 downto 0);  -- задаем тип сигналу cnk1 std_logic_vector
signal cnt1: std_logic_vector (2 downto 0);  -- задаем тип сигналу cnt1 std_logic_vector
signal q1 : std_logic_vector (2 downto 0); -- задаем тип сигналу q1 std_logic_vector
signal en1 : std_logic; -- задаем тип сигналу en1 std_logic
--signal count1 :  std_logic_vector (2 downto 0); -- задаем тип счетчику
--signal exq1 :  std_logic_vector (2 downto 0);
----------------------------------------------
begin -- переход
process (q1 , cnt1 , cnk1 , clk1 , en1 , rst1, a1, b1) --, exq1  , count) – работаем с сигналами описываемые процессом
begin -- переход
----------------------------------------------
 En1 <= a1 xor b1; -- задаем дополнительный вход
if (rst 1= '1') then – в случаее если сигнал сброса = '1'
cnk1 <= (others1 => '0'); -- тогда на выходе 0
elsif (rising_edge(clk)) then
if (en1 = '1') then
cnk1 <= cnk1 + '1'; -- прописываем счетчик
end if;
if (cnk1 = "0101") then
cnt1 <= (others => '0');
end if;
end if;
----------------------------------------------
if (q1 = "0101") then -- задаем значаение до скольки должен считать счетчик
cnt <= (others => '0');
end if;
if (q 1> "0101") then
cnt <= (others => '1');
end if;
----------------------------------------------
```

```
--if (cnt1 = "1111")then -- задаем значение при котором на выходе появл лог. еденица
--exq <= (others => '1');
--end if;
-----------------------------------------------
end process;
count1 <= cnt1; -- записываем результаты на определенные сигналы
q1 <= cnk1;
--ex1<=exq;
end behav1;
-----------------------------------------------
entity nextlevel1 is – задаем блок  nextlev1
port (a1, b1,  clk,  rst1 : in std_logic;  -- Обозначаем порты на вход
count : out std_logic_vector (2 downto 0)); -- Обозначаем порты на выход
--ex1:out std_logic_vector (2 downto 0)); -- Обозначаем порты на выход
end nextlevel1; -- завершаем nextlevel1
-----------------------------------------------
architecture behav1 of nextlevel1 is – называем архитектуру behav1 и пишем архитектуру
nextlevel1
signal cnk1: std_logic_vector (2 downto 0);  -- задаем тип сигналу cnk1 std_logic_vector
signal cnt1: std_logic_vector (2 downto 0);  -- задаем тип сигналу cnt1 std_logic_vector
signal q1 : std_logic_vector (2 downto 0); -- задаем тип сигналу q1 std_logic_vector
signal en1 : std_logic; -- задаем тип сигналу en1 std_logic
--signal count1 :  std_logic_vector (2 downto 0); -- задаем тип счетчику
--signal exq1 :  std_logic_vector (2 downto 0);
-----------------------------------------------
begin -- переход
process (q1 , cnt1 , cnk1 , clk1 , en1 , rst1, a1, b1) --, exq1  , count) – работаем с сигналами
описываемые процессом
begin -- переход
-----------------------------------------------
 En1 <= a1 xor b1; -- задаем дополнительный вход
if (rst 1= '1') then – в случаее если сигнал сброса = '1'
cnk1 <= (others1 => '0'); -- тогда на выходе 0
elsif (rising_edge(clk)) then
```

```
if (en1 = '1') then
cnk1 <= cnk1 + '1'; -- прописываем счетчик
end if;
if (cnk1 = "0101") then
cnt1 <= (others => '0');
end if;
end if;
-----------------------------------------
if (q1 = "0101") then -- задаем значаение до скольки должен считать счетчик
cnt <= (others => '0');
end if;
if (q 1> "0101") then
cnt <= (others => '1');
end if;
-----------------------------------------
--if (cnt1 = "1111")then -- задаем значение при котором на выходе появл лог. еденица
--exq <= (others => '1');
--end if;
-----------------------------------------
end process;
count1 <= cnt1; -- записываем результаты на определенные сигналы
q1 <= cnk1;
--ex1<=exq;
end behav1;
-----------------------------------------
entity nextlevel1 is – задаем блок  nextlev1
port (a1, b1,  clk,  rst1 : in std_logic;  -- Обозначаем порты на вход
count : out std_logic_vector (2 downto 0)); -- Обозначаем порты на выход
--ex1:out std_logic_vector (2 downto 0)); -- Обозначаем порты на выход
end nextlevel1; -- завершаем nextlevel1
-----------------------------------------
architecture behav1 of nextlevel1 is – называем архитектуру behav1 и пишем архитектуру
nextlevel1
signal cnk1: std_logic_vector (2 downto 0);  -- задаем тип сигналу cnk1 std_logic_vector
```

```
signal cnt1: std_logic_vector (2 downto 0);  -- задаем тип сигналу cnt1 std_logic_vector
signal q1 : std_logic_vector (2 downto 0); -- задаем тип сигналу q1 std_logic_vector
signal en1 : std_logic; -- задаем тип сигналу en1 std_logic
--signal count1 :  std_logic_vector (2 downto 0); -- задаем тип счетчику
--signal exq1 :  std_logic_vector (2 downto 0);
-----------------------------------------------
begin -- переход
process (q1 , cnt1 , cnk1 , clk1 , en1 , rst1, a1, b1) --, exq1  , count) – работаем с сигналами описываемые процессом
begin -- переход
-----------------------------------------------
 En1 <= a1 xor b1; -- задаем дополнительный вход
if (rst 1= '1') then – в случаее если сигнал сброса = '1'
cnk1 <= (others1 => '0'); -- тогда на выходе 0
elsif (rising_edge(clk)) then
if (en1 = '1') then
cnk1 <= cnk1 + '1'; -- прописываем счетчик
end if;
if (cnk1 = "0101") then
cnt1 <= (others => '0');
end if;
end if;
-----------------------------------------------
if (q1 = "0101") then -- задаем значаение до скольки должен считать счетчик
cnt <= (others => '0');
end if;
if (q 1> "0101") then
cnt <= (others => '1');
end if;
-----------------------------------------------
--if (cnt1 = "1111")then -- задаем значение при котором на выходе появл лог. еденица
--exq <= (others => '1');
--end if;
-----------------------------------------------
```

```
end process;
count1 <= cnt1; -- записываем результаты на определенные сигналы
q1 <= cnk1;
--ex1<=exq;
end behav1;
-----------------------------------------
entity nextlevel1 is – задаем блок  nextlev1
port (a1, b1,  clk,  rst1 : in std_logic;  -- Обозначаем порты на вход
count : out std_logic_vector (2 downto 0)); -- Обозначаем порты на выход
--ex1:out std_logic_vector (2 downto 0)); -- Обозначаем порты на выход
end nextlevel1; -- завершаем nextlevel1
-----------------------------------------
architecture behav1 of nextlevel1 is – называем архитектуру behav1 и пишем архитектуру
nextlevel1
signal cnk1: std_logic_vector (2 downto 0);  -- задаем тип сигналу cnk1 std_logic_vector
signal cnt1: std_logic_vector (2 downto 0);  -- задаем тип сигналу cnt1 std_logic_vector
signal q1 : std_logic_vector (2 downto 0); -- задаем тип сигналу q1 std_logic_vector
signal en1 : std_logic; -- задаем тип сигналу en1 std_logic
--signal count1 :  std_logic_vector (2 downto 0); -- задаем тип счетчику
--signal exq1 :  std_logic_vector (2 downto 0);
-----------------------------------------
begin -- переход
process (q1 , cnt1 , cnk1 , clk1 , en1 , rst1, a1, b1) --, exq1  , count) – работаем с сигналами
описываемые процессом
begin -- переход
-----------------------------------------
 En1 <= a1 xor b1; -- задаем дополнительный вход
if (rst 1= '1') then – в случаее если сигнал сброса = '1'
cnk1 <= (others1 => '0'); -- тогда на выходе 0
elsif (rising_edge(clk)) then
if (en1 = '1') then
cnk1 <= cnk1 + '1'; -- прописываем счетчик
end if;
if (cnk1 = "0101") then
```

```
cnt1 <= (others => '0');
end if;
end if;
-----------------------------------------------
if (q1 = "0101") then -- задаем значаение до скольки должен считать счетчик
cnt <= (others => '0');
end if;
if (q 1> "0101") then
cnt <= (others => '1');
end if;
-----------------------------------------------
--if (cnt1 = "1111")then -- задаем значение при котором на выходе появл лог. еденица
--exq <= (others => '1');
--end if;
-----------------------------------------------
end process;
count1 <= cnt1; -- записываем результаты на определенные сигналы
q1 <= cnk1;
--ex1<=exq;
end behav1;
-----------------------------------------------
entity nextlevel1 is – задаем блок  nextlev1
port (a1, b1,  clk,  rst1 : in std_logic;  -- Обозначаем порты на вход
count : out std_logic_vector (2 downto 0)); -- Обозначаем порты на выход
--ex1:out std_logic_vector (2 downto 0)); -- Обозначаем порты на выход
end nextlevel1; -- завершаем nextlevel1
-----------------------------------------------
architecture behav1 of nextlevel1 is – называем архитектуру behav1 и пишем архитектуру
nextlevel1
signal cnk1: std_logic_vector (2 downto 0);  -- задаем тип сигналу cnk1 std_logic_vector
signal cnt1: std_logic_vector (2 downto 0);  -- задаем тип сигналу cnt1 std_logic_vector
signal q1 : std_logic_vector (2 downto 0); -- задаем тип сигналу q1 std_logic_vector
signal en1 : std_logic; -- задаем тип сигналу en1 std_logic
--signal count1 :  std_logic_vector (2 downto 0); -- задаем тип счетчику
```

```
--signal exq1 :  std_logic_vector (2 downto 0);
-----------------------------------------------
begin -- переход
process (q1 , cnt1 , cnk1 , clk1 , en1 , rst1, a1, b1) --, exq1  , count) – работаем с сигналами описываемые процессом
begin -- переход
-----------------------------------------------
 En20 <= a20 xor b20; -- задаем дополнительный вход
if (rst20= '1') then – в случаее если сигнал сброса = '1'
cnk20 <= (others1 => '0'); -- тогда на выходе 0
elsif (rising_edge(clk)) then
if (en1 = '1') then
cnk20 <= cnk20 + '1'; -- прописываем счетчик
end if;
if (cnk20 = "0101") then
cnt1 <= (others => '0');
end if;
end if;
-----------------------------------------------
if (q20 = "0101") then -- задаем значаение до скольки должен считать счетчик
cnt <= (others => '0');
end if;
if (q 20> "0101") then
cnt <= (others => '1');
end if;
-----------------------------------------------
--if (cnt20 = "1111")then -- задаем значение при котором на выходе появл лог. еденица
--exq1 <= (others => '1');
--end if;
-----------------------------------------------
end process;
count1 <= cnt1; -- записываем результаты на определенные сигналы
q20 <= cnk20;
--ex20<=exq;
```

```
end behav20
```

;ПРИЛОЖЕНИЕ Б. Программа работы МК

Библиотека для LCD

```
#ifndef HD44780_Config
#define HD44780_Config

#define uchar unsigned char
#define Sbit(reg,bit) (reg|=(1<<bit))
#define Cbit(reg,bit) (reg&=~(1<<bit))
#define CheckBit(reg,bit) (reg&(1<<bit))
#define FirstStr_StartPosition_DDRAM_Addr 0x80
#define SecondStr_StartPosition_DDRAM_Addr 0xC0

/*
 *указываем разрядность шины данных дисплея (1-8 бит, 0-4 бита)
 */
#define Data_Length 0

/*
 *указываем количество строк дисплея (1-2 строки, 0-1 строка)
 */
#define NumberOfLines 1

/*
 *указываем размер шрифта дисплея (1–шрифт 5x10 точек, 0–шрифт 5x7 точек)
 */
#define Font 1

/*
 *указываем порт МК, к которому подключена линия E
 *указываем номер бита порта МК, к которому подключена линия E
 */
#define PORT_Strob_Signal_E PORTF
#define PIN_Strob_Signal_E 1

/*
 *указываем порт МК, к которому подключена линия RS
 *указываем номер бита порта МК, к которому подключена линия RS
 */
#define PORT_Strob_Signal_RS PORTF
#define PIN_Strob_Signal_RS 0

/*
 *порты ввода-вывода шины данных дисплея D4-D7
 *номер бита порта ввода-вывода шины данных дисплея D4-D7
 */
#if Data_Length==0        //4-х проводная шина данных
  #define PORT_bus_4 PORTD
  #define PIN_bus_4 4
  #define PORT_bus_5 PORTD
```

```
  #define PIN_bus_5 5
  #define PORT_bus_6 PORTD
  #define PIN_bus_6 6
  #define PORT_bus_7 PORTD
  #define PIN_bus_7 7
/*
 *указываем порты ввода-вывода шины данных дисплея D0-D7
 *указываем номер бита порта ввода-вывода шины данных дисплея D0-D7
 */
#elif Data_Length==1      //8-ми проводная шина данных
  #define PORT_bus_0 PORTD
  #define PIN_bus_0 0
  #define PORT_bus_1 PORTD
  #define PIN_bus_1 1
  #define PORT_bus_2 PORTD
  #define PIN_bus_2 2
  #define PORT_bus_3 PORTD
  #define PIN_bus_3 3
  #define PORT_bus_4 PORTD
  #define PIN_bus_4 4
  #define PORT_bus_5 PORTD
  #define PIN_bus_5 5
  #define PORT_bus_6 PORTD
  #define PIN_bus_6 6
  #define PORT_bus_7 PORTD
  #define PIN_bus_7 7
#endif

/*
 *функция инициализации дисплея. Должна вызываться первой.
 */
void LCD_Init(void);

/*
 *функция команды полной очистки DDRAM-памяти дисплея и установка курсора в
 *нулевую позицию
 */
void LCD_Full_Clean(void);

/*
 *функция команды установки курсора и положения дисплея в нулевую позицию
 */
void LCD_CursorPosition_ToStart(void);

/*
 *функция команды направления сдвига курсора после записи и разрешения сдви-
 *га дисплея вместе с курсором:
 *I_D(Increment/Decrement)-направление сдвига курсора после записи (I_D=1-сдвиг вправо,
I_D=0–сдвиг влево);
 *S(Shift)–разрешение сдвига дисплея вместе с курсором (S=1-сдвиг разрешен, S=0-сдвиг
запрещен);
```

```
 */
void LCD_AutoMovCurDispDirect(uchar I_D, uchar S);

/*
 *функция команды включения-выключения дисплея, включение-выключение курсора
 *и его мигания:
 *D(Display)–включение дисплея (D=1-дисплей включен, D=0-дисплей отключен)
 *C(Cursor)-видимость курсора (C=1–видимый курсор, C=0–погашенный курсор);
 *B(Blink)-мигание курсора (B=1–курсор мигает, B=0–курсор не мигает);
 */
void LCD_DisplEnable_CursOnOffBlink(uchar D, uchar C, uchar B);

/*
 *функция команды перемешения дисплея/курсора, направления перемешения:
 *S_C(Screen/Cursor)–перемещение дисплея/курсора (S_C=1–перемещается дисплей, S_C=0–
перемещается курсор)
 *R_L(Right/Left)-направление перемещения дисплея/курсора (R_L=1–перемещение вправо,
R_L=0–перемещение влево).
 */
void LCD_MovingCurDispDirection(uchar S_C, uchar R_L);

/*
 *функция вывода символов на дисплей. Параметры:
 *Addr-код символа (адрес в памяти знакогенератора)
 *Str-номер строки, в которой нужно вывести символ
 *Cursor-позиция символа в строке
 */
void LCD_Show(uchar Addr, uchar Str, uchar Cursor);

/*
 *функция записи пользовательских символов в память знакогенератора CGRAM
 *дисплея. Диапазон адресов параметра Addr: 0x00-0x07 (8 пользовательских симво-
 *лов). 1 символ занимает 8 байт данных
 */
void LCD_UserSymbolsWrite(uchar Addr, uchar *data);

/*
 *функция управляющих сигналов RS, E и шины данных. Вызывается для передачи дан-
 *ных дисплею. В качестве параметра RS передаётся 0 или 1. Если 0-команда, 1-данные.
 */
void BusLinesState(uchar *data, uchar RS);

#endif
```

Описание библиотеки для LCD:

```
#include "HD44780_Config.h"

#include <util/delay.h>
```

```
#include <avr/io.h>

#if Data_Length==1

  uchar *data_bus_port[8]={(uchar*)&PORT_bus_0, (uchar*)&PORT_bus_1,
(uchar*)&PORT_bus_2, (uchar*)&PORT_bus_3,
        (uchar*)&PORT_bus_4, (uchar*)&PORT_bus_5, (uchar*)&PORT_bus_6,
(uchar*)&PORT_bus_7};
  uchar data_bus_pin[8]={PIN_bus_0, PIN_bus_1, PIN_bus_2, PIN_bus_3, PIN_bus_4, PIN_bus_5,
PIN_bus_6, PIN_bus_7};

  void BusLinesState(uchar *data, uchar RS)
  {
        uchar time=40;

        for (uchar i=0; i<8; i++)
        if (CheckBit(*data,i)) Sbit(*data_bus_port[i],data_bus_pin[i]);
        else Cbit(*data_bus_port[i],data_bus_pin[i]);

        if (RS==0)
        {
              Sbit(PORT_Strob_Signal_E,PIN_Strob_Signal_E);
              _delay_us(time);
              Cbit(PORT_Strob_Signal_E,PIN_Strob_Signal_E);
              _delay_us(time);
        }
        else
        if (RS==1)
        {
              Sbit(PORT_Strob_Signal_RS,PIN_Strob_Signal_RS);
              Sbit(PORT_Strob_Signal_E,PIN_Strob_Signal_E);
              _delay_us(time);
              Cbit(PORT_Strob_Signal_E,PIN_Strob_Signal_E);
              Cbit(PORT_Strob_Signal_RS,PIN_Strob_Signal_RS);
```

```
                _delay_us(time);
        }
        else
        if (RS==2)
        {
                Sbit(PORT_Strob_Signal_E,PIN_Strob_Signal_E);
                _delay_ms(5);
                Cbit(PORT_Strob_Signal_E,PIN_Strob_Signal_E);
                _delay_ms(5);
        }
}

void LCD_Init(void)
{
        //6
        uchar bus_data=0b100000;

        if (Data_Length==1) Sbit(bus_data,4);
        else Cbit(bus_data,4);
        if (NumberOfLines==1) Sbit(bus_data,3);
        else Cbit(bus_data,3);
        if (Font==1) Sbit(bus_data,2);
        else Cbit(bus_data,2);
        BusLinesState(&bus_data,0);
}

#elif Data_Length==0

uchar *data_bus_port[4]={(uchar*)&PORT_bus_4, (uchar*)&PORT_bus_5,
(uchar*)&PORT_bus_6, (uchar*)&PORT_bus_7};
uchar data_bus_pin[4]={PIN_bus_4, PIN_bus_5, PIN_bus_6, PIN_bus_7};

void BusLinesState(uchar *data, uchar RS)
{
```

```
uchar time=40;
uchar HB, LB;
uchar *temp;

HB=*data>>4;
LB=*data&~(1<<7)&~(1<<6)&~(1<<5)&~(1<<4);

for (uchar z=0; z<2; z++)
{
        if (z==0) temp=&HB;
        else
        if (z==1) temp=&LB;

        for (uchar i=0; i<4; i++)
        if (CheckBit(*temp,i)) Sbit(*data_bus_port[i],data_bus_pin[i]);
        else Cbit(*data_bus_port[i],data_bus_pin[i]);

        if (RS==0)
        {
                Sbit(PORT_Strob_Signal_E,PIN_Strob_Signal_E);
                _delay_us(time);
                Cbit(PORT_Strob_Signal_E,PIN_Strob_Signal_E);
                _delay_us(time);
        }
        else
        if (RS==1)
        {
                if (z==0)
                Sbit(PORT_Strob_Signal_RS,PIN_Strob_Signal_RS);
                Sbit(PORT_Strob_Signal_E,PIN_Strob_Signal_E);
                _delay_us(time);
                Cbit(PORT_Strob_Signal_E,PIN_Strob_Signal_E);
                if (z==1)
                Cbit(PORT_Strob_Signal_RS,PIN_Strob_Signal_RS);
```

```
                _delay_us(time);
            }
            else
            if (RS==2)
            {
                Sbit(PORT_Strob_Signal_E,PIN_Strob_Signal_E);
                _delay_ms(5);
                Cbit(PORT_Strob_Signal_E,PIN_Strob_Signal_E);
                _delay_ms(5);
            }
        }
}

void LCD_Init(void)
{
        //6
        uchar bus_data=0b100000;
        uchar empty_Init=0x33;

        for (uchar i=0; i<2; i++)
        {
            if (i==0)
            BusLinesState(&empty_Init,0);
            else
            if (i==1)
            {
                empty_Init--;
                BusLinesState(&empty_Init,0);
            }
        }

        if (Data_Length==1) Sbit(bus_data,4);
        else Cbit(bus_data,4);
        if (NumberOfLines==1) Sbit(bus_data,3);
```

```
            else Cbit(bus_data,3);
            if (Font==1) Sbit(bus_data,2);
            else Cbit(bus_data,2);

            BusLinesState(&bus_data,0);
    }

#endif

void LCD_Full_Clean(void)//1
{
        uchar bus_data=0b1;
        BusLinesState(&bus_data,2);
        Sbit(PORT_Strob_Signal_E,PIN_Strob_Signal_E);
        _delay_ms(5);
        Cbit(PORT_Strob_Signal_E,PIN_Strob_Signal_E);
        _delay_ms(5);
}

void LCD_CursorPosition_ToStart(void)//2
{
        uchar bus_data=0b10;
        BusLinesState(&bus_data,2);
        Sbit(PORT_Strob_Signal_E,PIN_Strob_Signal_E);
        _delay_ms(5);
        Cbit(PORT_Strob_Signal_E,PIN_Strob_Signal_E);
        _delay_ms(5);
}

void LCD_AutoMovCurDispDirect(uchar I_D, uchar S)//3
{
        uchar bus_data=0b100;

        if (I_D==1) Sbit(bus_data,1);
```

```
        else Cbit(bus_data,1);
        if (S==1) Sbit(bus_data,0);
        else Cbit(bus_data,0);
        BusLinesState(&bus_data,0);
}

void LCD_DisplEnable_CursOnOffBlink(uchar D, uchar C, uchar B)//4
{
        uchar bus_data=0b1000;

        if (D==1) Sbit(bus_data,2);
        else Cbit(bus_data,2);
        if (C==1) Sbit(bus_data,1);
        else Cbit(bus_data,1);
        if (B==1) Sbit(bus_data,0);
        else Cbit(bus_data,0);
        BusLinesState(&bus_data,0);
}

void LCD_MovingCurDispDirection(uchar S_C, uchar R_L)//5
{
        uchar bus_data=0b10000;

        if (S_C==1) Sbit(bus_data,3);
        else Cbit(bus_data,3);
        if (R_L==1) Sbit(bus_data,2);
        else Cbit(bus_data,2);
        BusLinesState(&bus_data,0);
}

void LCD_UserSymbolsWrite(uchar Addr, uchar *data)
{
        uchar bus_data=0b1000000;
```

```
	bus_data|=Addr*8;
	BusLinesState(&bus_data,0);

	for (uchar i=0; i<8; i++)
	{
		BusLinesState(data,1);
		data++;
	}
}

void LCD_Show(uchar Addr, uchar Str, uchar Cursor)
{
	uchar x=0;

	if (Str==1)
	{
		x=FirstStr_StartPosition_DDRAM_Addr+Cursor;

		BusLinesState(&x,0);
		BusLinesState(&Addr,1);
	}
	else
	if (Str==2)
	{
		x=SecondStr_StartPosition_DDRAM_Addr+Cursor;

		BusLinesState(&x,0);
		BusLinesState(&Addr,1);
	}
}
```

Основная программа:

```
#define F_CPU 1000000
#include <avr/io.h>
```

```
#include "HD44780_Config.h"
#include <util/delay.h>


        uchar UserSymbol[4][8]={
        {0x11, 0x12, 0x14, 0x18, 0x18, 0x14, 0x12, 0x11} , // К
        {0x7, 0x1, 0x1, 0xF, 0x9, 0x9, 0x9, 0xF},          // a
        {0x11, 0x11, 0x11, 0x1F, 0x11, 0x11, 0x11, 0x11},  // H
        {0x3, 0x5, 0x5, 0x5, 0x5, 0x9, 0x9, 0x19}}  ;       // Л

        //0x3A это двоеточие
        // 0x31 это еденица


uchar Word1[]={0x00,0x01,0x02,0x01,0x03,0x3A};
uchar Word2[]={0x31};



int main()
{

        DDRB=0b00000000; PORTB=0b00000000;// установлено на входы
        DDRC=0b00000000; PORTC=0b00000000;// установлено на входы
        DDRF=0b00000011; PORTF=0b00000000;
        DDRD=0b11111111; PORTD=0b00000000;
        DDRA = 0b00000000; // порт D на вход
        PORTA = 0b11111111; // подключаем внутренние подтяг. резисторы
        DDRE = 0b11111111; // порт B на выход
```

```
    PORTE = 0b00000000; // устанавливаем нули на выходе




    LCD_Init();
    LCD_DisplEnable_CursOnOffBlink(1,0,0);

while(1)// выводим текст на LCD
{
        if(PINA&(1 << PA0)) // Проверяем нажатие кнопки
    {
        for (uchar i=0; i<4; i++)
        LCD_UserSymbolsWrite(i,&UserSymbol[i][0]);



        for (uchar i=0; i<6; i++){
            LCD_Show(Word1[i],1,i);
        LCD_Show(Word2[i],2,i);}
    while(1){// загораем светодиод
        if(PINA&(1 << PA0)) // Проверяем нажатие кнопки
        {
    PORTE &= ~(1 << PE0); // гасим светодиод
    _delay_ms(20); // задержка 20 мс
    PORTE |= (1 << PE0); // зажигаем светодиод
    _delay_ms(20); // задержка 20мс
        }
    else
```

```
        {
                PORTE &= ~(1 << PE0); // гасим светодиод
                _delay_ms(20); // задержка 20 мс
                PORTE |= (1 << PE0); // зажигаем светодиод
                _delay_ms(20); // задержка 20мс
        }}
        }
        else
        {
                PORTE &= ~(1 << PE0); // Гасим светодиод
        }

}}
#define G5      1275 // Соль
#define A5      1135 // Ля
#define B5      1011 // Си
#define D6      850 // Ре
#define E6      757 // Ми

#define LE32    1*3 // 1/32 Длительность тона
#define LE16    2*3 // 1/16
#define LE16D   3*3
#define LE16T   2*2
#define LE8     4*3 // 1/8
#define LE8D    6*3
#define LE8T    4*2
#define LE4     8*3 // 1/4
#define LE4D    12*3
#define LE4T    8*2
```

```
#define LE2     16*3 // 1/2
#define LE2D    24*3
#define LE1     32*3 // 1

void Delay_us(unsigned char time_us) // функция задержки в us
{ register unsigned char i;

        for(i = 0; i < time_us; i++)   // 4 цикла
        { asm (" PUSH  R0 ");       // 2 цикла
                asm (" POP   R0 ");       // 2 цикла
                // 8 циклов = 1 us для 8MHz
        }
}

void Delay_ms(unsigned int time_ms) // функция задержки в ms
{ register unsigned int i;

        for(i = 0; i < time_ms; i++)
        { Delay_us(250);
                Delay_us(250);
                Delay_us(250);
                Delay_us(250);
        }
}

unsigned char temp;

void Set_temp(unsigned char number)   // функция установки темпа и паузы
{
```

```
	temp = number;			// установка темпа
	TCCR1A = (1 << COM1A0);		// CTC режим, использование OC1A
	TCCR1B = (1 << WGM13)|(1 << WGM12);	// звук выключен
}

void Play_LE(unsigned int sound, unsigned int LE) // функция проигрывания ноты
{
	ICR1 = sound;			// установка ICR1
	TCNT1 = 0x0000;			// очистка Timer/Counter1
	TCCR1B |= (1 << WGM13)|(1 << WGM12)	// CTC режим
	|(1 << CS10);	// предделитель = 8

	Delay_ms(LE*temp*7);		// длительность ноты

	TCCR1B = (1 << WGM13)|(1 << WGM12); // звук выключен
}

int main(void)
{
	DDRB |= (1 << PB5);
	PORTB = 0x00;

	Delay_ms(50);

	Set_temp(3);	// установка темпа
	Play_LE(Dis3,LE8);
	Play_LE(Cis3,LE8);
	Play_LE(Fis3,LE4);
	Play_LE(Fis3,LE4);
```

```
        Delay_ms(1000);
        while(1);
}
```