Министерство образования и науки Российской Федерации
Федеральное государственное автономное образовательное учреждение
высшего образования
«Российский государственный профессионально-педагогический университет»

РАЗРАБОТКА ЛАБОРАТОРНОГО СТЕНДА ПО ПРОГРАММИРОВАНИЮ МИКРОКОНТРОЛЛЕРА STM

Выпускная квалификационная работа
по направлению подготовки 44.03.04 Профессиональное обучение (по отраслям)
профилю подготовки «Энергетика»
профилизации «Энергохозяйство предприятий, организаций, учреждений и энерго-сберегающие технологии»

Идентификационный код ВКР: 150

Екатеринбург 2017

Министерство образования и науки Российской Федерации
Федеральное государственное автономное образовательное учреждение высшего образования
«Российский государственный профессионально-педагогический университет»
Институт инженерно-педагогического образования
Кафедра электрооборудования и энергоснабжения

К ЗАЩИТЕ ДОПУСКАЮ:
Заведующая кафедрой ЭС
_____ А.О. Прокубовская
«____»_____2017 г.

ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА

РАЗРАБОТКА ЛАБОРАТОРНОГО СТЕНДА ПО ПРОГРАММИРОВАНИЮ МИКРОКОНТРОЛЛЕРА STM

Исполнитель:
студент(ка) группы ЭС-402 _____ (подпись) А.Ю. Иванин

Руководитель:
ст. преподаватель кафедры ЭС _____ (подпись) А.А. Емельянов

Нормоконтролер:
ст. преподаватель кафедры ЭС _____ (подпись) Т.В. Лискова

Екатеринбург 2017

АННОТАЦИЯ

Выпускная квалификационная работа выполнена на 61 страницах, содержит 40 рисунков,10 таблиц, 26 источников литературы, а также 5 приложений на 39 страницах.

Ключевые слова: МИКРОКОНТРОЛЛЕР, ПРОГРАММИРОВАНИЕ, ЯЗЫК ПРОГРАММИРОВАНИЯ, ЛАБОРАТОРНЫЙ СТЕНД, ШИРОТНО – ИМПУЛЬСНАЯ МОДУЛЯЦИЯ.

Иванин А. Ю., Разработка лабораторного стенда по программированию микроконтроллера STM: выпускная квалификационная работа / А. Ю. Иванин; Рос. гос. проф. – пед. ун-т, Ин-т инж. – пед. образования, Каф. Электрооборудования и энергоснабжения. – Екатеринбург, 2017. – 60 с.

Краткая характеристика содержания ВКР:

1. Тема выпускной квалификационной работы «Разработка лабораторного стенда по программированию микроконтроллера STM». В работе рассмотрен процесс программирования микроконтроллеров STM32.

2. Цель работы: разработать лабораторный стенд по программированию микроконтроллера STM32.

3. В ходе выполнения выпускной квалификационной работы были рассмотрены: базовые понятия о микроконтроллерах, основные отличия микроконтроллеров фирмы STMicroelectronics – STM32 от микроконтроллеров другой фирмы, особенности программирования микроконтроллеров STM32. Был разработан лабораторный стенд, также был разработан комплекс лабораторных работ.

4. Специализированного оборудования для изучения автоматики в электроприводе в учебных заведениях практически нет, данный лабораторный стенд был создан с целью показать обучаемым в реальности то, что они видели в книгах по электроприводу.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

В век научно-технического прогресса основной упор образования должен ставиться на развитие технического мышления у обучаемых. Необходимо обеспечить применение теоретических знаний на практике, именно на практике обучаемые смогут развить необходимые им в дальнейшем навыки и компетенции.

Использование обучающего оборудования в процессе обучения, даёт возможность закрепить полученные на теоретических уроках знания. Лабораторные стенды позволяют проводить работы близкие к реальной производственной деятельности, это придает процессу обучения особый стиль и вызывает больший интерес у студентов.

В современном мире системы управления электроприводом строятся в основном на микроконтроллерах. Микроконтроллер представляет собой систему «на одном кристалле», при минимальной стоимости имеет в себе массу периферийных устройств. Программирование микроконтроллера означает написание алгоритма с помощью специального языка программирования и запись его в память контроллера.

В качестве выпускной квалификационной работы был разработан лабораторный стенд по изучению возможностей программирования микроконтроллеров фирмы STMicroelectronics – STM32. Специализированного оборудования для изучения автоматики в электроприводе в учебных заведениях практически нет, данный лабораторный стенд был создан с целью показать обучаемым в реальности то, что они видели в книгах по электроприводу. Лабораторные работы, которые студенты смогут на нем проводить будут посвящены исследованиям широтно-импульсной модуляции при помощи данных микроконтроллеров. Исследования ШИМ и общее знакомство с микропроцессорной техникой даст возможность обучаемым лучше ориентироваться в вопросах о системах управления электроприводом.

Актуальность темы обусловлена:

– развитием микропроцессорной техники в качестве систем управления;

– применением частотно-регулируемого электропривода во всех отраслях промышленности;

– развитием полупроводниковой преобразовательной техники;

– потребностью в высококвалифицированном техническом персонале;

– необходимостью применения учебной техники в процессе подготовки студентов.

Объект исследования: микроконтроллер STM32.

Предмет исследования: лабораторный стенд по программированию микроконтроллера STM.

Цель: разработка лабораторного стенда по программированию микроконтроллера STM32.

Задачи:

– провести анализ литературы по микроконтроллерам фирмы STMicroelectronics;

– провести обзор элементной базы и схемотехнических решений;

– разработать программное обеспечение для микроконтроллеров;

– изготовить лабораторный стенд;

– разработать комплекс лабораторных работ по программированию микроконтроллеров.

1. БАЗОВЫЕ ПОНЯТИЯ О МИКРОКОНТРОЛЛЕРАХ

Микроконтроллер (рисунок 1) – микросхема, которая позволяет управлять различными электронными устройствами, содержит в себе ПЗУ (постоянное запоминающие устройство) и ОЗУ (оперативное запоминающие устройство). По сути микроконтроллер является однокристальным компьютером, способным выполнять простые операции. Имеет в наличии интегрированные порты ввода-вывода, таймеры и другие периферийные устройства.

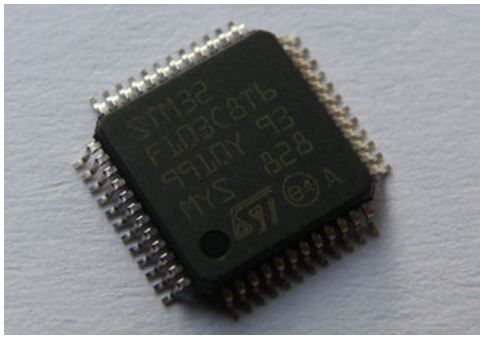


Рисунок 1 – Микроконтроллер фирмы STMicroelectronics

Программное обеспечения для микроконтроллеров пишутся на специальных языках программирования (обычно на языке Ассемблера или Си) и записываются в память контроллера при помощи программатора (рисунок 2). Программатор это, аппаратно-программное устройство, предназначенное для считывания и записи в постоянное запоминающее устройство.

Для хранения исполняемой программы в микроконтроллере используется память FLASH, так же называемая памятью программ. FLASH – память является энергозависимой и может быть перезаписана около десяти тысяч раз. Данная память является подобием жесткого диска персонального компьютера.

Для хранения различных изменяющихся данных, которые используются во время выполнения программ, используется оперативная память, так же называемая RAM. Эта память может быть записана и стерта любое количество раз, но данные сохраняются в ней до тех пор, пока микроконтроллер подключен

к питанию. Является подобием оперативной памяти персонального компьютера.

Рисунок 2 – Программатор фирмы STMicroelectronics

Для хранения редко-изменяющихся данных (и сохранения данных во время того как микроконтроллер отключен) используется память EEPROM. В некоторых микроконтроллерах данный тип памяти отсутствует, поэтому приходится эмулировать его программно.

При проектировании микроконтроллеров приходится соблюдать компромисс между размерами и стоимостью с одной стороны и гибкостью, и производительностью с другой. Для разных приложений оптимальное соотношение этих и других параметров может различаться очень сильно. Поэтому существует огромное количество типов микроконтроллеров, отличающихся архитектурой процессорного модуля, размером и типом встроенной памяти, набором периферийных устройств, типом корпуса и т. д. В отличие от обычных компьютерных микропроцессоров, в микроконтроллерах часто используется гарвардская архитектура памяти, то есть раздельное хранение данных и команд в ОЗУ и ПЗУ соответственно [15].

Микроконтроллеры часто содержат встроенные периферийные устройства, которые позволяют использовать одну маленькую микросхему вместо большого количества отдельных устройств.

Неполный список периферийных устройств, которые могут использоваться в микроконтроллерах, включает в себя:

– универсальные цифровые порты, которые можно настраивать как на ввод, так и на вывод;

– различные интерфейсы ввода-вывода, такие, как UART, I²C, SPI, CAN, USB, IEEE 1394, Ethernet;

– аналого-цифровые и цифро-аналоговые преобразователи;

– компараторы;

– широтно-импульсные модуляторы (ШИМ-контроллер);

– таймеры;

– контроллеры бесколлекторных двигателей, в том числе шаговых;

– контроллеры дисплеев и клавиатур;

– радиочастотные приемники и передатчики;

– массивы встроенной FLASH-памяти;

– встроенные тактовый генератор и сторожевой таймер.

Для продолжения темы микроконтроллеров необходимо ввести ряд определений:

1. Порты ввода – вывода (GPIO) – группы выводов микроконтроллеров, предназначенных для общения микроконтроллера с «внешним микром».

2. UART, I^2C, SPI, CAN, USB – различные интерфейсы ввода-вывода, предназначенные для передачи данных.

3. АЦП – аналогово – цифровой преобразователь, предназначен для измерения значений входящих напряжений и преобразования их в цифровую информацию.

4. ЦАП – цифро – аналоговый преобразователь, предназначен для преобразования входящего цифрового сигнала в выходящий аналоговый сигнал.

5. Таймер – позволяет очень точно отмерять интервалы времени, генерировать выходной ШИМ – сигнал, подсчитывать период входного сигнала и т.д.

6. Watchdog – сторожевой таймер, предназначен для контроля над зависанием системы.

7. DMA – модуль прямого доступа к памяти, позволяет передавать данные между устройствами не используя ресурсы процессора.

Применение микроконтроллеров в современном мире очень широко, так как используется достаточно мощное вычислительное устройство с широким спектром возможностей, построенного на всего одной микросхеме это в значительной степени снижает энергопотребление и стоимость устройств, построенных на его базе.

Микроконтроллеры используют для управления различными электронными устройствами и отдельными блоками:

– в вычислительной технике – контроллеры для жестких/гибких дисков, в материнских платах, калькуляторах;

– электронике и разнообразных устройствах бытовой техники, в которой используется электронные системы управления – микроволновых печах, стиральных машинах, телефонах, различных роботах, в системах «Умный дом».

Микроконтроллеры также нашли широкое применение в промышленности:

– устройства промышленной автоматики – от программируемого реле до ПЛК;

– системы управления станками.

1.1. Микроконтроллеры STM32

В данной разработке лабораторного стенда используются микроконтроллеры STM32. Выбор пал именно на эти контроллеры из-за наличия в них интегрированной периферии – расширенных таймеров, которые позволяют формировать ШИМ – сигнал для управления 3-фазными двигателями, а именно такой ШИМ является целью исследований в лабораторных работах.

Микроконтроллеры STM32 выполнены на базе ядра ARM Cortex. На первый взгляд разница между микроконтроллерами STM32 и микроконтроллерами другой фирмы не видна, набор встроенных УВВ (устройств ввода-вывода) та-

кие как АЦП, таймеры общего назначения, I^2C, SPI, CAN, USB и часы реального времени имеют и самые обыкновенные микроконтроллеры. Но если рассматривать каждое из этих УВВ более детально станет очевидно, что в STM32 они устроены гораздо сложнее. Например, АЦП – аналогово – цифровой преобразователь имеет разрядность в 12 бит, а также имеет встроенный датчик температуры и поддерживает несколько режимов преобразования входных данных. Таймеры оснащены блоками захвата и сравнения и могут быть использованы как отдельно, так и синхронно, что позволяет создавать большие массивы таймеров. В STM32, как было сказано ранее, имеются, так называемые, расширенные таймеры (advanced timers), их используют для управления электродвигателями. Для этого у них предусмотрено шесть комплементарных ШИМ-выводов с программируемым мертвым временем (dead-time) [20].

В отличие от других микроконтроллеров в STM32 предусмотрен модуль DMA – прямой доступ к памяти, каждый канал данного модуля может быть использован для передачи данных между регистрами любого из УВВ и запоминающими устройствами.

Еще одним плюсом микроконтроллеров STM32 является сочетание характеристик малого энергопотребления и довольно высокой производительности. Они способны работать всего лишь от 2В-ого источника питания на тактовой частоте 72Мгц и потреблять в активном состоянии всего лишь 36мА, если же использовать поддерживаемые ядром Cortex экономные режимы работы, то можно снизить энергопотребление до ничтожных 2мА.

За безопасность в данном типе микроконтроллеров отвечают два сторожевых таймера (watchdog), которые позволяют в случае ошибки исполнения программы перезагрузить микроконтроллер автоматически и продолжить выполнение.

При разработке лабораторного стенда использовались микроконтроллеры STM32F103C8T6 (рисунок 1) и STM32F407VGT6 (рисунок 3).

Рисунок 3 – Микроконтроллер STM32F407VGT6

Микроконтроллер STM32F103C8T6 имеет 64Кб Flash – памяти и тактовую частоту ядра 72МГц. Выполнен на базе процессора ARM Cortex M3 и имеет 32-х битную архитектуру. Имеет в наличии один расширенный таймер TIM1 с помощью которого можно генерировать 3-х фазный ШИМ сигнал.

Микроконтроллер STM32F404VGT6 имеет 1Мб Flash- памяти и тактовую частоту ядра 192Мгц. В отличие от предыдущего STM32F4VGT6 выполнен на базе процессора ARM Cortex M4, который так же имеет 32-х битную архитектуру. В наличие имеется уже два расширенных таймера, а также двухканальный ЦАП, который будет использоваться в исследованиях.

1.2. Особенности программирования микроконтроллеров STM32

Программирование микроконтроллеров STM32 сводится к написанию алгоритма программы на языке программирования и записи исполняемой программы в память контроллера. Для написания программ будет использоваться язык С (Си). С (Си) – компилируемый статически типизированный язык программирования общего назначения, разработанный в 1969—1973 годах сотрудником Bell Labs Деннисом Ритчи как развитие языка Би.

Вначале код пишется на понятном для человека языке в среде программирования IDE, затем с помощью компилятора переводится в, так называемый, машинный код и записывается с помощью программатора в память микроконтроллера.

Для программирования STM32 на языке С существует множество платных и бесплатных сред разработок. Также существуют компиляторы для различных языков. Но в данной разработке используется язык программирования Си, следовательно, и компилятор будет использоваться для этого языка.

Для того чтобы научиться разрабатывать программное обеспечение для микроконтроллеров, которые будут использованы в лабораторных работах, необходимо научиться общаться с ними на одном языке [17].

Комментарии

Комментарии необходимы для пояснения кода программы и не являются частью кода. Комментарии бывают многострочные и однострочные. Однострочные и многострочные комментарии подробнее рассмотрены в таблице 1.

Таблица 1 – Виды комментариев

Комментарии	
//Однострочный комментарий	Если поместить // на любую строку, то компилятор проигнорирует весь текст после этого символа на данной строке. Обычно используется для пояснения определенной строки кода.
/*Многострочный комментарий*/	Многострочные комментарии начинаются с /* и заканчиваются */. Многострочные комментарии необходимы при раскрытии целого блока кода, например описание программы.

Переменные

Все константы и операторы располагаются в FLASH – памяти (ПЗУ, RAM, память программ), содержимое данной памяти остается неизменной в процессе работы программы. Для хранения данных, которые могут быть изменены в процессе выполнения программы используется оперативная память (ОЗУ, RAM, память данных), в данной памяти так же располагаются все переменные. Переменная – поименованная, либо адресуемая иным способом область памяти, адрес которой можно использовать для осуществления доступа к данным и изменять значение в ходе выполнения программы. Для ввода переменной в языке С используются следующие ключевые слова:

1. char – создает переменную размером 1 байт (8 бит), которая может принимать 256 значений.

2. int – вещественный тип, размер создаваемой переменной будет зависеть от архитектуры используемого микроконтроллера, для STM32 размер будет равен 32 бита, и будет занимать в памяти микроконтроллера 4 байта.

3. short – вещественный тип, размер переменной 16 бит, это означает что переменная может принимать 65536 значений и занимает в памяти 2 байта.

4. long – вещественный тип, размер переменной 32 бита, это означает что переменная может принимать 2^{32} значений и занимает в памяти 4 байта.

5. float – переменная используемая для операций с числами с плавающей запятой, занимает 4 байта. На действие с данным типом переменной необходимо затратить больше тактов процессора, то есть больше времени.

Для числовых переменных существуют модификаторы, которые указывают на наличие знака у переменной.

1. unsigned – указывает на то, что переменная не будет иметь знака то есть будет принимать только положительные значения, например *unsigned char* создаст переменную с диапазоном значений от 0 до 255.

2. signed – указывает на то, что переменная будет иметь знак, например *signed char* создаст переменную с диапазоном значений от -128 до +128.

Для хранения строковых данных, а также для хранения большого количества однотипных значений удобно использовать массивы. Массив, как и переменная имеет тип, а также имеет свое уникальное имя. В отличии от переменной при определении массива необходимо указать количество входящих в него элементов. В дальнейшем к целому массиву можно обращаться просто, написав его имя, а к определенному элементу массива можно обратится по индексу элемента. Индекс элемента в массиве — это порядковый номер элемента, начиная с нуля.

Имена переменных, функций и констант в С должны выбираться исходя из следующих правил:

1. Имена должны начинаться с букв латинского алфавита (a – z, A – Z), либо со знака нижнего подчеркивания «_».

2. В именах переменных, констант, а также функций разрешено использовать только буквы латинского алфавита, цифры (0 – 9), а также символ нижнего подчеркивания «_», другие символы использовать запрещено.

3. В языке С регистр букв имеет значение, это означает что применяемые буквы нижнего регистра (a – z) и буквы верхнего регистра (A – Z) различны. Например, имена переменных: *name, Name, NaMe* – различны.

4. Имена переменных, констант, функций могут иметь длину в соответствии со стандартом ANSIC не превышающую 32 символа.

5. Идентификаторы для новых объектов не должны совпадать с ключевыми словами языка, а также с названиями библиотечных функций.

Если необходимо записать значение переменной в шестнадцатеричном виде, то используется знак «0x» (например, 0xAAA), если необходимо создать переменную в двоичном виде, то используется знак «0b» (например, 0b1001).

Различные виды переменных представлены в таблице 2.

Таблица 2 – Некоторые виды переменных

Введенная переменная	Область применения переменной
unsigned char name_value1;	Данная переменная может использоваться когда, заранее известно что значение будет изменятся например в диапазоне от 0 до 100, не превышает размерность 1 байт, а также имеет только положительные значения.
unsigned short name_value2;	Данная переменная может использоваться тогда когда, заранее известно что значение будет изменятся например в диапазоне от 50 до 5000, не превышает размерность 2 байт, а также имеет только положительные значения
signed int name_value3;	Данная переменная подходит для диапазона значений например от -30 до +6000, имеет размерность 4 байта
unsigned int name_array[100];	Данный массив представляет собой набор из ста переменных типа *unsigned int* с именем *name_array.*

Диапазон возможных значений для разных типов переменных представлен в таблице 3.

Таблица 3 – Диапазон значений для различных переменных

Тип переменной	Диапазон значений для unsigned	Диапазон значений для signed
char	От 0 до 255	От -128 до +127
short	От 0 до 65535	От -32768 до +32767
int (для Cortex – M3)	От 0 до 4294967295	От – 2147483648 до +2147483647
long	От 0 до 4294967295	От – 2147483648 до +2147483647
long long	От 0 до 18446744073709551615	От -223372036854775808 до +223372036854775807

Операторы

Для совершения операций над переменными существуют операторы действия:

1. «+» – оператор математического сложения, например, *a+b*, складывает две переменные *a* и *b*.
2. «-» – оператор математического вычитания, например, *a-b*.
3. «*» – оператор математического умножения, например, *a*b*, перемножает две переменные *a* и *b*.
4. «/» – оператор математического деления, например, *a/b*.
5. «%» – оператор остатка от деления, определяет остаток от деления, например, *a%b, 6/4 =1.5*, остаток от деления *0.5*, значит при *6%4 = 0.5*.
6. «=» – оператор присваивания значения, например, *a=b*, присваивает переменной *a* значение переменной *b*.
7. «<» – операция сравнение – меньше.
8. «>» – операция сравнение – больше.
9. «<=» – операция сравнение – меньше либо равно.
10. «>=» – операция сравнение – больше либо равно.
11. «==» – математическая операция сравнения – равно.
12. «!=» – математическая операция сравнения – не равно.

13. «++» – инкремент, например, *a++*, инкремент числа *a*, увеличение на единицу.

14. «--» – декремент, например, *a--*, декремент числа *a*, уменьшение на единицу.

15. «<<» – операция побитовый сдвиг влево, например, *a<<b*, сдвигает значение *a* влево, на количество бит *b, 23<<1,* число *23* в двоичной системе счисления имеет вид *00010111,* сдвигается на *1* ***влево***, получается *0 0010 111****0****,* что соответствует числу *46* в десятичной системе счисления.

16. «>>» – операция побитовый сдвиг вправо, например, *a>>b*, сдвигает значение *a* вправо, на количество бит *b, 23>>1,* число *23* в двоичной системе счисления имеет вид *00010111,* сдвигается на *1* ***вправо***, получается ***0****000 1011,* что соответствует числу *11* в десятичной системе счисления.

17. «&» – операция поразрядовое логическое «И», например, *a&b*. Также данная операция называется логическим умножением.

18. «|» – операция поразрядовое логическое «ИЛИ», например, *a|b*. Также данная операция называется логическим сложением.

19. «~» – операция логическое «НЕ», например, *~a*, инвертирует значение переменной a, то есть логические нули всех разрядов становятся логическими единицами, а логические единицы становятся нулями.

Логические операторы

Язык программирования не является полноценным без использования логических операторов. В языке С существует четыре основных оператора: *if, while, for, switch*.

Оператор *if* производит условный переход к выполнению различных частей программ. Оператор имеет определенную структуру (рисунок 4).

```
if(условие)
{
Код выполняемый при выполнении "условия"
}

else
{
Данный код выполняется если "Условие" не выполнено
      }
```

Рисунок 4 – Структура оператора *if*

В круглых скобках после *if* указывается условие, при выполнении которого будет исполнен код, который заключен между первыми фигурными скобками. Если условие не выполняется, то будет исполнятся код, который заключен между фигурными скобками после *else*. Наличие *else* необязательно, если не требуется выполнение кода в случае не выполнения условия, то *else* можно не писать.

В качестве условия очень часто указывают выражения сравнения, например, *a == b* или *a!=b*. Фигурные скобки означают начало и конец кода.

Если необходимо использовать сразу несколько условий, то такие условия объединяют операторами логических связей «&&» – логическое «И», «||» – логическое «ИЛИ».

Для создания циклов, дающих возможность многократно повторять один и тот же код до достижения какого-то условия, существую циклы *while* и *for*.

Синтаксис цикла *while* очень прост, в круглых скобках, которые следуют сразу же за словом *while*, указывается условие, пока данное условие будет выполняется, будет и выполняется код, заключенный в фигурные скобки после *while* (рисунок 5).

```
while(условие)
{

код выполняется, пока выполняется условие

}
```

Рисунок 5 – Структура цикла *while*

Цикл *while* необходимо применять с осторожность, так как возможно создать ситуацию, когда условия цикла не будет выполнено никогда, что приве-

дет к зависанию программы. Более безопасно использовать цикл *for*. Данный цикл предназначен для выполнения заранее заданного количества повторений кода (рисунок 6). Синтаксис написания цикла *for* следующий: *for*(задание начального условия; условие; операция по изменению условия).

```
unsigned char i;
      //Переменная для условия в цикле for
      for(i=0; i<=10; i++)
      {
      a++;
      //Данный код будет выполняться пока i не станет равной 10
      }
```

Рисунок 6 – Структура цикла *for*

Функции

Очень часто программа должна выполнять одну и ту же последовательность действий, при достаточном повторе таких действий и с целью улучшения читаемости кода, эти последовательности создают в виде отдельной группы. Когда необходимо снова выполнить данную последовательность, программу просто направляют к данному блоку. Такой обособленный блок кода в языке С называется функцией.

Основными предпосылками для создания функций являются:

– наличие одинаковых, часто повторяющихся действий;

– желание структурировать программу в виде отдельных блоков для удобства прочтения кода.

Использование функций имеет существенный минус, например, при переходе к функции и возврате из нее микроконтроллер вынужден сохранять некоторые данные, например, адрес программы, а это неминуемо скажется на скорости расчета.

Любая функция имеет свой уникальный заголовок, список передаваемых ей переменных, тип возвращаемой переменной, и, самое основное, это тело функции, с содержанием ее кода. Имя функции выбирается по тем же правилам что и имя для переменной. Общий вид функции представлен на рисунке 7.

```
возвращаемый_тип_переменной function_name(передаваемый_тип_переменной)
{


}
```

Рисунок 7 – Общий вид функции

Если функция не должна принимать и/или возвращать данные, то в качестве типа данных необходимо указать *void*. Любая функция может возвращать только одну переменную, но принимать может множество переменных, указанных в скобках через запятую. Пример функции приведен на рисунке 8.

```
void delay(unsigned int n)
{
        while(n)
        {
                n--;
        }
}
```

Рисунок 8 – Пример функции с *void*

Для того, чтобы иметь возможность вызывать функции, расположенные как выше, так и ниже места вызова. Для этого в самом начале файла необходимо прописать имена функций (объявить функции) с указанием передаваемых и возвращаемых им типов данных.

Заголовочные файлы

В больших программах, появляется множество различных функций, для удобства вызова этих функций их прототипы выносят в отдельные файлы, которые называются заголовочными. Заголовочный файл имеет расширение «.h» и подключается к файлу исходного кода (с расширение «.c») с помощью директивы *#include<>*. Для удобства читаемости кода обычно создают несколько файлов с исходным кодом (расширение «.c») и соответствующие им заголовочные файлы с расширением «.h».

Макросы (препроцессорные директивы)

При использовании в различных частях кода множества постоянно повторяющихся констант, либо других небольших фрагментов кода, таких как, например, переопределение выводов микроконтроллера, для упрощения изме-

нения этих мест, а также для улучшения читаемости кода используют препроцессорную директиву *#define*.

Общий вид макроса:

#define имя_макроса заменяемая строка

Команда *#define* используется для организации замены по всему файлу. Другими словами, *#define* сообщает компилятору, что необходимо заменить «имя_макроса» на «заменяемая строка».

Предпроцессорная директива *#define* может использовать логические условия, подобно *«if…else»*. Форма записи логических условий представлена на рисунке 9.

```
#ifdef ADC //если макрос ADC был определен ранее
//Действия выполняемые, если ADC был определен ранее
#else //если ADC не был определен ранее
//Действия выполняемые, если ADC не был определен ранее
#endif // Конец условия
```

Рисунок 9 – Логические условия для макросов

Помимо *#ifdef* также используются и другие условия:

#ifndef – подобно *#ifdef*, но выполняется если указанный макрос не был ранее определен;

#if – подобно *if* сравнивает выполнение логического условия для макросов.

Программисты, пишущие программы на языке С, использующие макросы, обычно в качестве идентификаторов используются символы верхнего регистра. Если разработчик будет следовать данному правилу, то тот, кто будет читать его программу сразу же поймет где будет происходить макрозамена.

Для того чтобы понять особенности программирования STM32 необходимо подробно разобрать один пример. Для примера будет выбрана самая простая программа, также называемая «Hello world! для микроконтроллеров» – зажечь светодиод, расположенный на отладочной плате. Для данного примера

используется отладочная плата на базе микроконтроллера STM32F103C8T6 (рисунок 10).

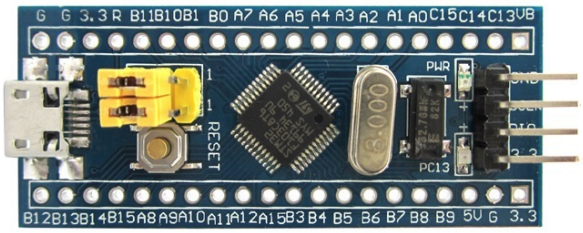


Рисунок 10 – Отладочная плата на базе STM32F103C8T6

Плата имеет 32 вывода для «общения» микроконтроллера с внешним миром, которые могут быть сконфигурированы как на вход, так и на выход. Имеется также кнопка сброса, которая позволяет прекратить выполнение программы и начать всё заново. Также плата имеет интерфейсы Micro-USB и SWD для программатора. Данная отладочная плата также имеет светодиод, который подключен к порту C выводу 13.

Для загрузки рабочей программы в микроконтроллер по интерфейсу SWD будет использоваться программатор ST-LINK V2 (рисунок 11) для микроконтроллеров STM32.

Рисунок 11 – Программатор ST-LINK V2

Для изучения основ программирования необходимо ввести некоторые определения:

– библиотека – сборник подпрограмм, которые используются для разработки программного обеспечения;

– структура – специальная конструкция, позволяющая содержать в себе множество переменных различного типа;

– регистр процессора – блок ячеек памяти, который образует сверхбыструю оперативную память;

– прерывание – сигнал от программного или аппаратного обеспечения, сообщающий процессору о наступлении какого-нибудь события, прерывания бывают внутренними и внешними.

Программа состоит из функций (подпрограмм), их может быть сколько угодно много, но всего они должны быть обязательно написаны в основной функции *main()*, так как данная функция является входной точкой, что находится в этой функции. Общий вид функции *main()* представлен на рисунке 12.

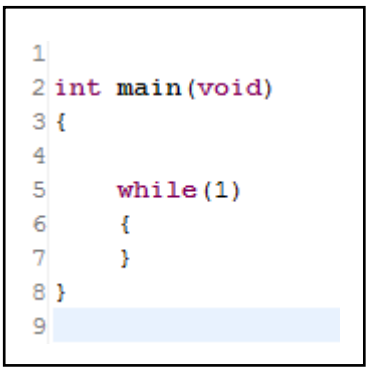


Рисунок 12 – Функция *main()*

Между фигурными скобками заключаются блоки программ. В функции main представлены два блока программ: первый от самой функции, в нем будут располагаться все подпрограммы разработанного кода; второй блок принадлежит бесконечному циклу *while*.

Разработка любого кода начинается с выбора и подключения библиотек. Для того чтобы зажечь светодиод необходимы всего две библиотеки *RCC* и *GPIO*.

Библиотека *RCC* используется для управления тактовым генератором микроконтроллера. Тактовый генератор микроконтроллера позволяет подавать напряжение на порты ввода – вывода и другую периферию, так как одной из особенностей микроконтроллеров STM32 является выключенная периферия для экономии энергии.

Библиотека *GPIO* используется для управления GPIO (General Ports Input – Output) портами ввода – вывода. Позволяет настраивать выводы микро-

контроллера на вход или на выход, считывать состояние выводов, получать входные данные и т.д.

Заголовочные файл – файл, содержимое которого автоматически добавляется препроцессором в исходный текст. По сложившейся традиции в заголовочных файлах размещают библиотечные функции. Подключаются с помощью директивы *#include* <>, где между «<>» должно быть записано название заголовочного файла. Пример подключения заголовочных файлов представлен на рисунке 13.

```
1 #include <stm32f10x_rcc.h>//библиотека для тактового генератора
2 #include <stm32f10x_gpio.h>//библиотека для GPIO
3 int main(void)
4 {
5
6     while(1)
7     {
8     }
9 }
```

Рисунок 13 – Пример подключения заголовочных файлов

Когда подключение заголовочных файлов завершено, необходимо приступить к разработке самого кода. Вначале необходимо включить тактирование порта, на котором находится светодиод, на самой плате указано что это порт С. Тактирование портов в данном микроконтроллере включается функцией *RCC_APB2PeriphClockCmd(),* APB2 – это шина к которой подключены все порты ввода – вывода данного микроконтроллера. Функция *RCC_APB2PeriphClockCmd()* имеет два аргумента, первый аргумент — это периферия, которую собираемся включить (например, *RCC_APB2Periph_GPIOC*, так включается тактирование порта С), второй аргумент — это включение или отключение тактирования и может принимать значения либо *ENABLE*, либо *DISABLE* соответственно.

После включения тактирования порта необходимо настроить вывод 13, как выход. Для этого нужно создать и заполнить структуру, понятие структура было введено ранее. Для создания структуры портов ввода – вывода используется ключевое слово *GPIO_InitTypeDef*. Затем структура должна быть заполне-

на полями, для данного микроконтроллера чтобы определить вывод необходимо заполнить всего три поля:

1) GPIO_Mode – определяет в качестве чего будет настроен вывод, может принимать значения:

а) **GPIO_Mode_AIN** — аналоговый вход;

б) **GPIO_Mode_IN_FLOATING** — дифференциальный вход;

в) **GPIO_Mode_IPD** — вход с подтягивающим резистором к земле;

г) **GPIO_Mode_IPU** — вход с подтягивающим резистором к питанию;

д) **GPIO_Mode_Out_OD** — выход с открытым стоком;

е) **GPIO_Mode_Out_PP** — выход двумя состояниями;

ж) **GPIO_Mode_AF_OD** — выход с открытым стоком для альтернативных функций. Используется в случаях, когда выводом должна управлять периферия, прикрепленная к данному выводу порта (например, вывод Tx USART1 и т.п.);

з) **GPIO_Mode_AF_PP** — выход для альтернативных функций с двумя состояниями;

2) GPIO_Pin – определяет номер вывода;

3) GPIO_Speed – определяет скорость тактирования вывода.

Для включения светодиода нужно настроить вывод как выход, для этого необходимо в поле *GPIO_Mode* указать значение *GPIO_Mode_Out_PP***.** В поле *GPIO_Pin* необходимо указать значение *GPIO_Pin_13*, так как светодиод находится на 13 выводе. В поле *GPIO_Speed* необходимо указать значение *GPIO_Speed_2MHz* такой частоты достаточно для того чтобы зажечь светодиод.

После заполнения всех полей необходимо инициализировать структуру с помощью функции *GPIO_Init()*, эта функция передают значения полей структуры в порт GPIO. Функция *GPIO_Init()* имеет два аргумента, первый аргумент – наименование порта в который будет отсылаться структура, в данном случае это *GPIOC*, второй аргумент — это наименование структуры перед которой обязательно ставится знак «&». Знак «&» называется указателем на адрес памяти.

После инициализации структуры необходимо записать бит в вывод 13 порта B, для включения светодиода. Делается это функцией *GPIO_SetBits()*: функция имеет два аргумента, первый аргумент — это наименование порта в данном случае это *GPIOC,* второй аргумент это номер вывода, в данном случае это *GPIO_Pin_13*. После этого необходимо сбросить бит функцией *GPIO_ResetBits()*: функция имеет два аргумента, таких же как их предыдущая. Обе этих функции записываются в бесконечном цикле *while(1)*. Полный код программы представлен на рисунке 14.

После этого программу необходимо скомпилировать, а затем загрузить в память микроконтроллера.

По такому принципу строятся все программы для микроконтроллеров STM32. Так же как был включен светодиод можно инициализировать любое периферийное устройство, входящее в состав микроконтроллера, используя стандартные библиотеки. Описание функций для любого другого периферийного устройства можно найти в заголовочных файлах с расширением «.h».

```
#include <stm32f10x_rcc.h>//библиотека для тактового генератора
#include <stm32f10x_gpio.h>//библиотека для GPIO
int main(void)
{
//функция включения тактирования
RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOC, ENABLE);
//Создание структуры gpio
GPIO_InitTypeDef gpio;
//Заполнение полей структуры
gpio.GPIO_Mode = GPIO_Mode_Out_PP;
gpio.GPIO_Pin = GPIO_Pin_13;
gpio.GPIO_Speed = GPIO_Speed_2MHz;
//Инициализация структуры с указанием на адрес в памяти gpio
GPIO_Init(GPIOC, &gpio);
    while(1)
    {
        //Запись бита в порт
        GPIO_SetBits(GPIOC, GPIO_Pin_13);
        //Сброс бита
        GPIO_ResetBits(GPIOC, GPIO_Pin_13);
    }
}
```

Рисунок 14 – Полный код программы для включения светодиода

1.3. Основные сведения о широтно – импульсной модуляции

ШИМ (широтно – импульсная модуляция) – сигнал, который представляет собой последовательность прямоугольных импульсов, которые характеризу-

ются периодом – T и длительностью «единицы» – d, в цифровом виде ШИМ представляет собой череду импульсов с двумя возможными значениями «единицей» и «нулем». ШИМ представлен на рисунке 15.

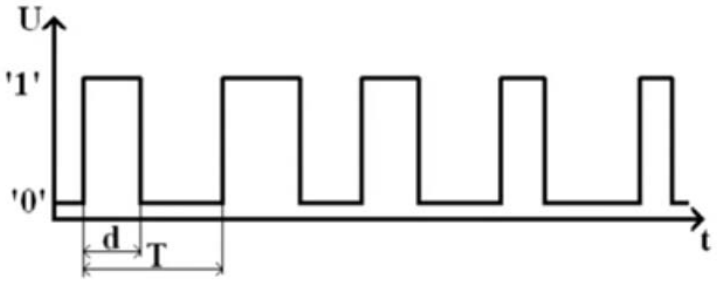


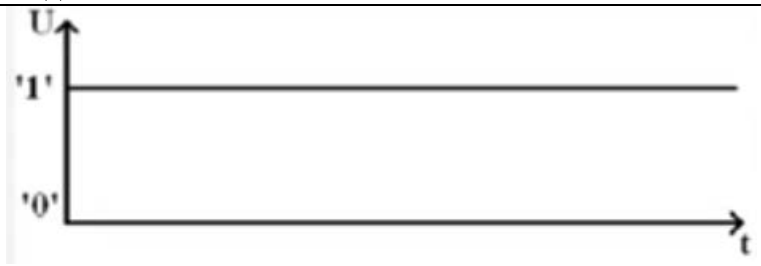
Рисунок 15 – ШИМ

Отношение длительности «единицы» – d к периоду – T называется скважностью ШИМ и обозначается буквой S:

$$S = \frac{d}{T} \quad (1)$$

В зависимости от значения S, ШИМ принимает различный вид, виды сигналов приведены в таблице 4.

Таблица 4 – Виды сигналов при различной скважности

Скважность	Вид сигнала
1	
0	
0.5	

При S = 1, ШИМ представляет собой чистую логическую единицу. При S = 0, ШИМ представляет собой чистый логический ноль. Если S = 0.5, то ШИМ – сигнал представляет собой идущие через равные промежутки прямоугольники, такой сигнал называется – меандр. Эти значения представлены в качестве примера, могут существовать и другие значения, их диапазон лежит в пределах от 0 до 1.

В микроконтроллерах используется генерация ШИМ при помощи таймеров. На рисунке 16 представлена генерация ШИМ при помощи таймера [19].

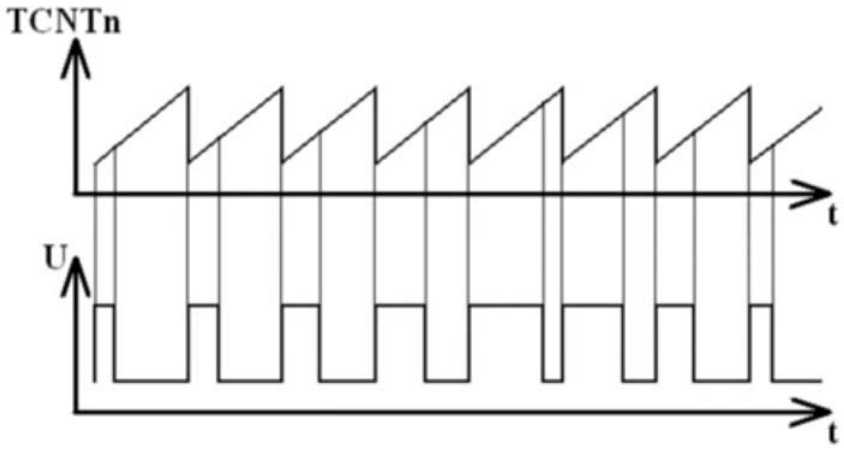


Рисунок 16 – Генерация ШИМ при помощи таймера

Пилообразный сигнал формируется счетчиком таймера. Как видно из рисунка 16, как только счетчик начинает считать в ШИМ – сигнале выставляется единица, когда счетчик досчитывает до нужного значения единица сбрасывается в ноль и так далее. Если менять значение, до которого должен досчитать счетчик, то будет изменяться скважность ШИМ, таким образом, происходит формирование ШИМ – сигнала с помощью таймера.

2. ОПИСАНИЕ ЛАБОРАТОРНОГО СТЕНДА

Для проведения лабораторных работ был разработан учебный лабораторный стенд с использованием отладочных плат на базе микроконтроллеров STM32. Внешний вид стенда представлен на рисунке 17.

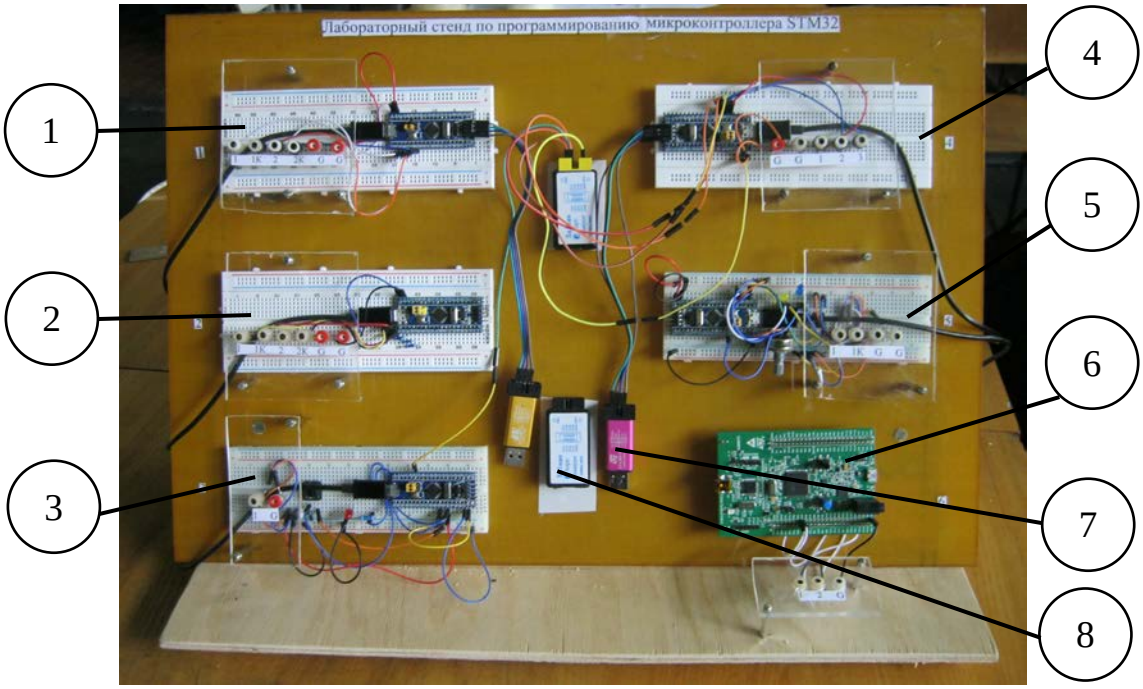


Рисунок 17 – Лабораторный стенд

Учебный лабораторный стенд предназначен для исследования возможностей микроконтроллеров STM32, изучения принципов программирования ШИМ – сигналов.

Компоненты лабораторного стенда:

1. Фрагмент лабораторного стенда по изучению программирования синусоидальных ШИМ – сигналов, сдвинутых на 90°. Фрагмент выполнен в виде макетной платы без пайки, с установленной на ней отладочной платой на базе микроконтроллера STM32F103C8T6. Также к стенду на металлических стойках прикреплено оргстекло, на нем расположены гнезда для подключения цифрового осциллографа. Используется в лабораторной работе № 1.

2. Фрагмент лабораторного стенда по изучению программирования управляющих сигналов транзисторной стойки с генерацией мертвого времени

(DeadTime). Так же, как и предыдущий, данный фрагмент выполнен в виде макетной платы без пайки, с установленной на ней отладочной платой на базе микроконтроллера STM32F103C8T6. Также к стенду на металлических стойках прикреплено оргстекло, на нем расположены гнезда для подключения цифрового осциллографа. Используется в лабораторной работе № 2.

3. Фрагмент лабораторного стенда по изучению программирования регулирования скважности ШИМ – сигнала с помощью кнопки. Данный фрагмент выполнен в виде макетной платы без пайки, с установленной на ней отладочной платой на базе микроконтроллера STM32F103C8T6, помимо микроконтроллера на макетной плате установлена кнопка при нажатии на которую изменяется выходной сигнал, а также установлен светодиод, который наглядно показывает изменение скважности импульса. Также к стенду на металлических стойках прикреплено оргстекло, на нем расположены гнезда для подключения цифрового осциллографа. Используется в лабораторной работе № 3.

4. Фрагмент лабораторного стенда по изучению программирования трехфазных синусоидальных ШИМ – сигналов, сдвинутых на 120°. Выполнен в виде макетной платы без пайки, с установленной на ней отладочной платой на базе микроконтроллера STM32F103C8T6. Также к стенду на металлических стойках прикреплено оргстекло, на нем расположены гнезда для подключения цифрового осциллографа. Используется в лабораторной работе № 4.

5. Фрагмент лабораторного стенда по изучению программирования регулирования ШИМ – сигнала с помощью потенциометра с генерацией мертвого времени (DeadTime). Фрагмент выполнен в виде макетной платы без пайки, с установленной на ней отладочной платой на базе микроконтроллера STM32F103C8T6, помимо микроконтроллера на макетной плате закреплен потенциометр при вращении которого изменяется скважность выходных импульсов. Также к стенду на металлических стойках прикреплено оргстекло, на нем расположены гнезда для подключения цифрового осциллографа. Используется в лабораторной работе № 5.

6. Фрагмент лабораторного стенда по изучению программирования синусоидального и пилообразного сигналов. В данном фрагменте используется отладочная плата STM32F4 – DISCOVERY (рисунок 18) на базе микроконтроллера STM32F407VGT6. Плата размещена без использования макетной платы. Гнезда для подключения цифрового осциллографа размещены на оргстекле. Используется в лабораторной работе № 6.

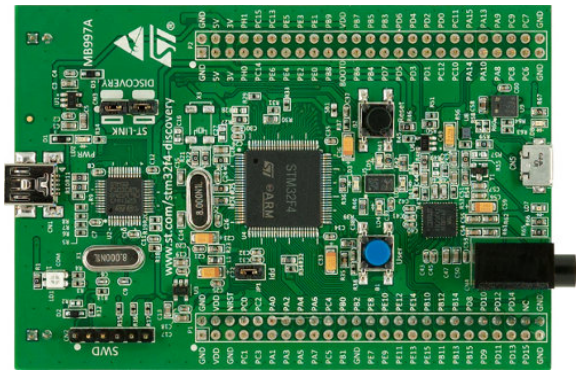


Рисунок 18 – Отладочная плата STM32F4 – DISCOVERY

7. Программатор ST – LINK/V2. Используется для загрузки рабочего кода в микроконтроллеры, может быть подключен к любому из микроконтроллеров, расположенных на стенде. Также может быть использован для отладки программ.

8. Логический анализатор. для того чтобы снимать выходные сигналы с выводов плат, на лабораторном стенде расположены два 8 – ми канальных логических анализатора Saleae logic (рисунок 19). Логический анализатор — это электронное устройство, способное записывать и отображать последовательность цифровых сигналов, с помощью специализированного программного обеспечения для персонального компьютера возможно проводить анализ цифровых сигналов, поступающих с выводов микроконтроллера. К логическому анализатору возможно подключать одновременно восемь выводов контроллера.

Рисунок 19 – Логический анализатор Saleae logic

2.1. Описание среды программирования

Для данного проекта была использована среда программирования CooCox CoIDE. Это свободная интегрированная среда разработки, ориентированная на микроконтроллеры на базе ARM Cortex. По сути CoIDE является текстовым редактором для кода, компилятор для языка Си (GCC for ARM) необходимо устанавливать и подключать отдельно. Основным преимуществом данной среды разработки является то, что благодаря специальному хранилищу, при подключенном интернете, пользователь имеет доступ ко множеству библиотек и примеров. Общий вид окна среды программирования представлен на рисунке 20.

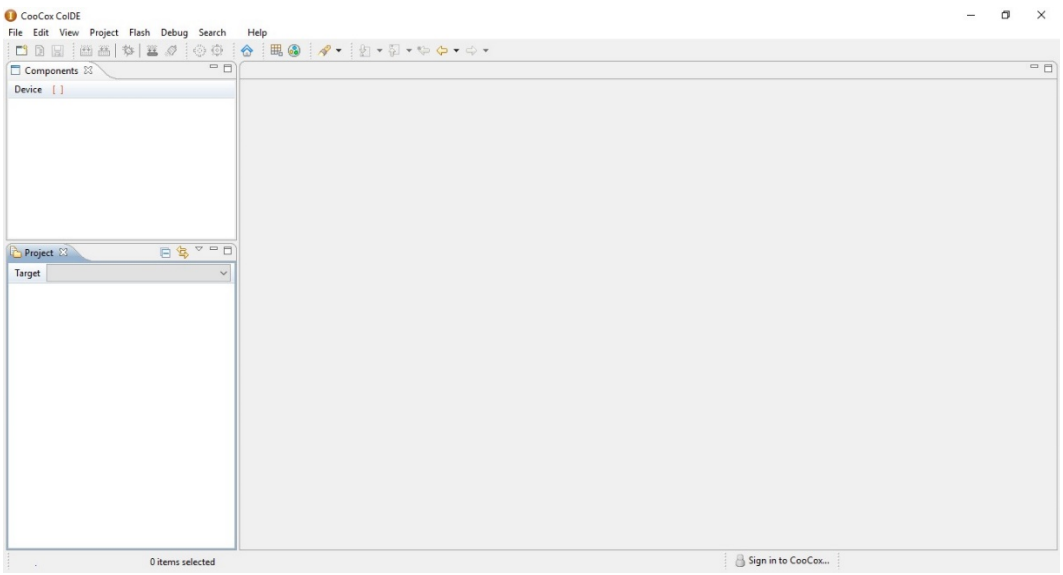


Рисунок 20 – Общий вид CooCox CoIDE

Для работы с CooCox CoIDE необходимо ознакомиться с панелью инструментов (рисунок 21).

Рисунок 21 – Панель инструментов

Основные инструменты, с которыми работает пользователь:

– создать новый проект (New project);

– создать новый файл (New file);

– сохранить проект (Save);

– собрать проект, скомпилировать (Build);

– начать отладку устройства (Start Debug);

– загрузка кода в Flash – память (Download Code To Flash);

– репозиторий, хранилище, где находятся подключаемые библиотеки (Repository).

Основное пространство окна занимает текстовый редактор кода (рисунок 22), в нем пользователь набирает и редактирует свои коды. В текстовом редакторе присутствует указание на ошибки, если код набран неверно, появляется надпись об этом. Также имеется очень удобная система автодополнения, которая, при написании некоторой последовательности символов, предлагает разработчику дополнить текст чтобы получить необходимые функции в программировании.

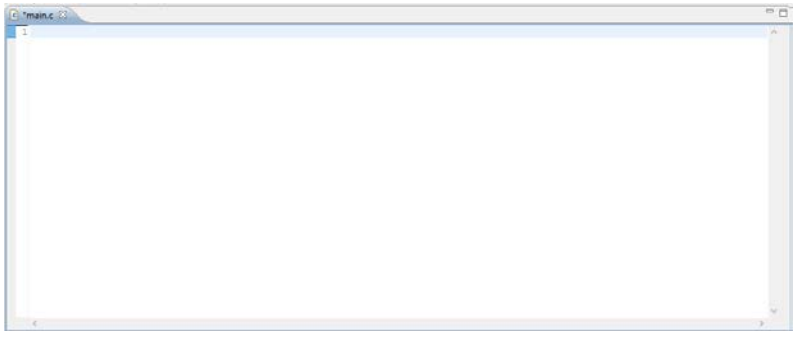


Рисунок 22 – Текстовой редактор кода CoIDE

Слева от редактора кода находится панель компонентов проекта и панель файлов, входящих в проект (рисунок 23). В панели компонентов отображаются все подключенные к проекту библиотеки, а также к каждой из библиотек мож-

но посмотреть примеры программ в интернете. В панели файлов отображаются все файлы, которые находятся в папке проекта и подключены к нему, можно подключать и создавать новые файлы.

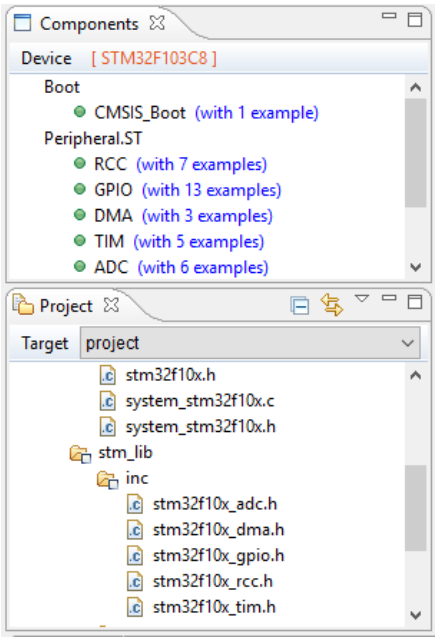


Рисунок 23 – Панель компонентов и панель файлов

Очень важной частью окна является консоль (рисунок 24). В консоли отображаются все действия, которые производит пользователь: компиляция, загрузка кода, отладка и так далее.

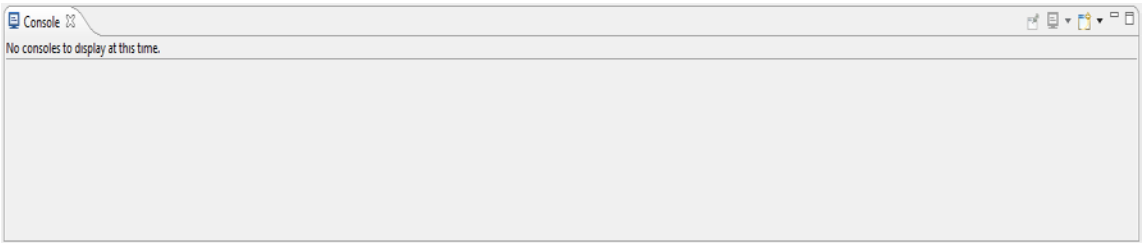


Рисунок 24 – Консоль

2.2. Разработка комплекса лабораторных работ

Данная разработка предназначена для студентов профиля «Энергетика» направления подготовки 44.03.04 Профессиональное обучение (по отраслям) [16]. Может применяться в рамках дисциплины «Интегрированные инженерные сети». В процессе выполнения комплекса лабораторных работ, обучающиеся

познакомятся с применением микроконтроллеров в качестве систем управления, которые могут применятся в таких актуальных, на сегодняшний день, технологиях как «умный» дом. Также данный комплекс лабораторных работ может применятся в дисциплине «Электрические машины» для изучения систем управления электроприводом.

2.3. Указания по выполнению лабораторной работы 1

Тема «Программирование двухфазного генератора с синусоидальными ШИМ – сигналами со сдвигом 90°»

Цель: программирование ШИМ – сигналов, сдвинутых на 90°.

Оборудование:

– учебный лабораторный стенд;

– персональный компьютер;

– двухканальный цифровой осциллограф.

Краткие теоретические сведения:

Сдвиг сигналов в данной лабораторной работе производится с помощью прерывания, которое генерируется системным таймером (SysTick). Системный таймер является частью ядра Cortex – M3, это 24 – битный вычитающий счетчик, с функциями автоперезагрузки и генерации прерывания по завершению счета. Данный таймер работает на тактовой частоте процессора.

Для инициализации системного таймера используется функция *SysTick_Config()*, в аргументах (скобках) данной функции указывается количество тактов, которое следует считать таймеру. После инициализации системного таймера, важной частью является описание функции прерывания (обработчика прерывания). Все функции прерывания можно найти в файле *startup_stm32f10x_md.c* [24]. На рисунке 25 представлен общий вид функции прерывания для системного таймера.

```
11 void SysTick_Handler(void)
12 {
13
14
15 }
```

Рисунок 25 – Функция прерывания *SysTick*

Во время прерывания процессор приостанавливает свою текущую активность, сохраняя своё состояние и переходит к выполнению обработчика прерывания. В обработчике прерывания будет записан код, в результате которого будут получены синусоидальные напряжение, сдвинутые на 90°.

Содержание и порядок выполнения работы:

1. На рабочем столе компьютера найти и двойным щелчком левой кнопки мыши на значке CooCox CoIDE запустить среду программирования.

2. После запуска CooCox CoIDE в строке меню нажать:

Project -> New Project.

3. В появившемся окне в поле Project name, ввести имя своему будущему проекту на английской раскладке, после этого нажать Next>.

4. Выбрать поле с надписью Chip, после этого нажать Next>.

5. Появится окно с выпадающими списками различных фирм микроконтроллеров (рисунок 26), необходимо открыть список ST, затем из выпадающего списка открыть подсписок STM32F103x, после этого найти микроконтроллер STM32F103C8 выбрать его левым щелчком мыши и нажать Finish.

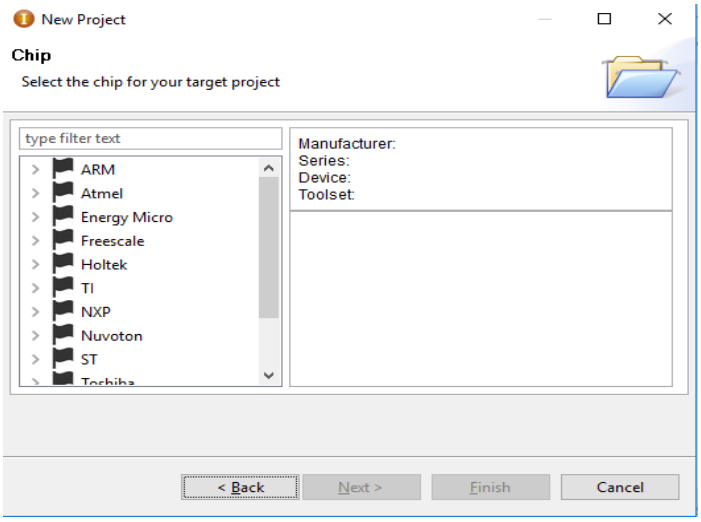


Рисунок 26 – Выбор микроконтроллера

6. После проделанных действий появится главное окно с репозиторием для выбора необходимых для проекта библиотек, подключить следующие библиотеки:

а) RCC – для управления тактовым генератором;

б) GPIO – для управления портами ввода – вывода;

в) TIM – для управления таймерами.

7. После выбора необходимых библиотек, в панели файлов выбрать файл *main.c*, весь код будет находится здесь.

8. С помощью директивы *#include* <> подключить заголовочные файлы (рисунок 27).

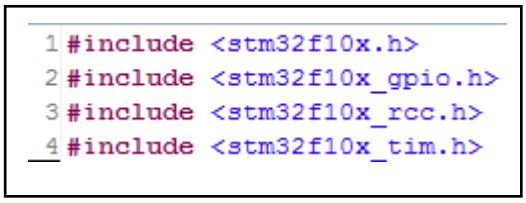


Рисунок 27 – Подключенные библиотеки

9. Ввести все структуры, которые будут использоваться в коде, а также ввести массив для построения синусоидальной ШИМ (Рисунок 28). Массив имеет тип *uint16_t*, что означает, что данный массив не имеет отрицательных значений, а также числа, входящие в этот массив, могут принимать значения в диапазоне от 0 до 65535.

```
//-------Ввод структур-------------
GPIO_InitTypeDef gpio;
GPIO_InitTypeDef gpio_1;
GPIO_InitTypeDef gpio_2;
GPIO_InitTypeDef gpio_3;
TIM_TimeBaseInitTypeDef TIM_Base_1;
TIM_OCInitTypeDef TIM_PWM;
TIM_BDTRInitTypeDef bdtr;
uint16_t sinPWM[125]={
        240,255,270,285,300,314,328,342,
        356,369,381,393,404,415,425,434,
        443,450,457,463,468,472,476,478,
        480,480,480,478,476,472,468,463,
        457,450,443,434,425,415,404,393,
        381,369,356,342,328,314,300,285,
        270,255,240,225,210,195,180,166,
        152,138,124,111,99,87,76,65,
        55,46,37,30,23,17,12,8,
        4,2,0,0,0,2,4,8,
        12,17,23,30,37,46,55,65,
        76,87,99,111,124,138,152,166,
        180,195,210,225,240,
        240,255,270,285,300,314,328,342,
        356,369,381,393,404,415,425,434,
        443,450,457,463,468,472,476,478
};
```

Рисунок 28 – Ввод структур и массива синуса

10. Для удобства восприятия кода, программа была разделена на несколько подпрограмм (функций), вначале необходимо ввести и заполнить функцию *void initRcc(void)* (рисунок 29) для включения тактирования всех используемых периферийных устройств. Данная функция имеет тип *void*.

```
//--------Функция включения тактирования переферии---------------
void initRCC(void)
{
    RCC->APB2ENR |= RCC_APB2ENR_IOPBEN | RCC_APB2ENR_IOPAEN
            | RCC_APB2ENR_TIM1EN | RCC_APB2ENR_AFIOEN;
    RCC->AHBENR |= RCC_AHBENR_DMA1EN;
}
```

Рисунок 29 – Функция включения тактирования

11. Ввести и заполнить функцию инициализации всех периферийных устройств *void initAll(void)*, данная функция, как и предыдущая также имеет тип *void*. Но так как она имеет большой объем необходимо разбить ее на участки:

а) назначение портов ввода – вывода (рисунок 30). Все выводы назначены как альтернативные функции с двумя состояниями;

```
//------Назначение GPIO (Портов ввода-вывода)-------------
GPIO_StructInit(&gpio);
gpio.GPIO_Mode = GPIO_Mode_AF_PP;
gpio.GPIO_Pin = GPIO_Pin_8;
gpio.GPIO_Speed = GPIO_Speed_50MHz;
GPIO_Init(GPIOA, &gpio);
//-------------------------------------
GPIO_StructInit(&gpio_1);
gpio_1.GPIO_Mode = GPIO_Mode_AF_PP;
gpio_1.GPIO_Pin = GPIO_Pin_9;
gpio_1.GPIO_Speed = GPIO_Speed_50MHz;
GPIO_Init(GPIOA, &gpio_1);
//-------------------------------------
GPIO_StructInit(&gpio_2);
gpio_2.GPIO_Mode = GPIO_Mode_AF_PP;
gpio_2.GPIO_Pin = GPIO_Pin_13;
gpio_2.GPIO_Speed = GPIO_Speed_50MHz;
GPIO_Init(GPIOB, &gpio_2);
//---------------------------------
GPIO_StructInit(&gpio_3);
gpio_3.GPIO_Mode = GPIO_Mode_AF_PP;
gpio_3.GPIO_Pin = GPIO_Pin_14;
gpio_3.GPIO_Speed = GPIO_Speed_50MHz;
GPIO_Init(GPIOB, &gpio_3);
```

Рисунок 30 – Назначение GPIO в функции initAll()

б) инициализация таймера TIM1 (рисунок 31):

```
//-----Назначение TIM1----------------
TIM_TimeBaseStructInit(&TIM_Base_1);
TIM_Base_1.TIM_Prescaler = 0;//предделитель частоты 0
TIM_Base_1.TIM_CounterMode = TIM_CounterMode_Up;//таймер считает вверх
TIM_Base_1.TIM_Period = 480;
TIM_Base_1.TIM_ClockDivision = TIM_CKD_DIV1;
TIM_TimeBaseInit(TIM1,&TIM_Base_1);
TIM_Cmd(TIM1, ENABLE);
```

Рисунок 31 – Инициализация таймера

в) инициализация ШИМ (рисунок 32):

```
TIM_OCStructInit(&TIM_PWM);
TIM_PWM.TIM_OCMode = TIM_OCMode_PWM1;
TIM_PWM.TIM_OutputState = TIM_OutputState_Enable;//Включение каналов ШИМ
TIM_PWM.TIM_OutputNState = TIM_OutputNState_Enable;//Включение комплементарных каналов ШИМ
TIM_OC1Init(TIM1, &TIM_PWM);//Первый канал ШИМ
TIM_OC1PreloadConfig(TIM1, TIM_OCPreload_Enable);
TIM_OC2Init(TIM1, &TIM_PWM);//Второй канал ШИМ
TIM_OC2PreloadConfig(TIM1, TIM_OCPreload_Enable);
```

Рисунок 32 – Инициализация ШИМ

г) необходимо инициализировать системный таймер, для этого написать функцию *SysTick_Config()*, в аргументах у данной функции указывается количество тактов, необходимо указать 1200, в итоге должно получиться *SysTick_Config(1200)*.

12. Заполнить обработчик прерывания *SysTick_Handler* (рисунок 33), в котором будет проходить процесс записи значений массива в регистры сравнения таймера.

```
106 void SysTick_Handler(void)// Функция прерывания(системный таймер)
107 {
108     t1++;
109     if(t1>=100)
110     {t1=0;}
111     GPIO_ResetBits(GPIOA, GPIO_Pin_8 | GPIO_Pin_9 | GPIO_Pin_10);//Сброс битов в ранее настроенных выводах
112     TIM1->CCR1 = (sinPWM[t1+25]);//Запись данных из массива sinPWM в регистр сравнения CCR1 таймера TIM1
113     TIM1->CCR2 = (sinPWM[t1]);//Запись данных из массива sinPWM в регистр сравнения CCR2
114
115 }
116
117
```

Рисунок 33 – Описание функции прерывания

13. После того как все функции были введены и заполнены нужно объявить их перед функцией *int main()*, как это делается представлено на рисунке 34.

```
33 void initRCC(void);
34 void initAll(void);
35 int main(void)
36 {
37
```

Рисунок 34 – Объявление функций

14. Записать данные функции между фигурными скобками в *main()* (рисунок 35), цикл *while*, в этом проекте останется пустым. После этого код можно считать завершенным.

```
35 int main(void)
36 {
37
38     initRCC();
39     initAll();
40
41     while(1)
42     {
43
44     }
45 }
```

Рисунок 35 – Функция *main()*

15. После написания кода программы его необходимо скомпилировать, для этого в панели инструментов нужно нажать «Build», проект будет соби-

раться. В случае успешной компиляции в консоли появится надпись «BUILD SUCCESSFUL», а также будет указан размер программы. Если же в коде присутствуют ошибки тогда в консоли будут указаны где именно находятся эти ошибки, а также появится надпись «BUILD FAILED».

16. После завершения компиляция, последним этапом станет загрузка рабочей программы в микроконтроллер (прошивка). Для этого нужно, через специальный кабель (удлинитель USB), подключить программатор, расположенный на лабораторном стенде к компьютеру. После подключения, в панели инструментов нажать «Download Code to Flash» и дождаться окончания загрузки, в случае удачной загрузки в консоли появятся надписи: Erase: Done; Program: Done; Verify: Done. Если существуют проблемы с подключением платы к компьютеру, то появится надпись «Error: Connect failed, check config and cable connection», необходимо проверить кабель, к которому подключено устройство;

17. Для проверки работоспособности программы необходимо подключить двухканальный цифровой осциллограф «GWINSTEK GDS – 71062», к выводам на лабораторном стенде. Щупы осциллографа присоединяются к клеммам. Первый канал осциллографа необходимо подключить к клемме «G» и «1», предварительно настроив масштаб 2В на осциллографе. Второй канал осциллографа необходимо подключить к клемме «G» и «2».

Описание клемм представлено на рисунке 36.

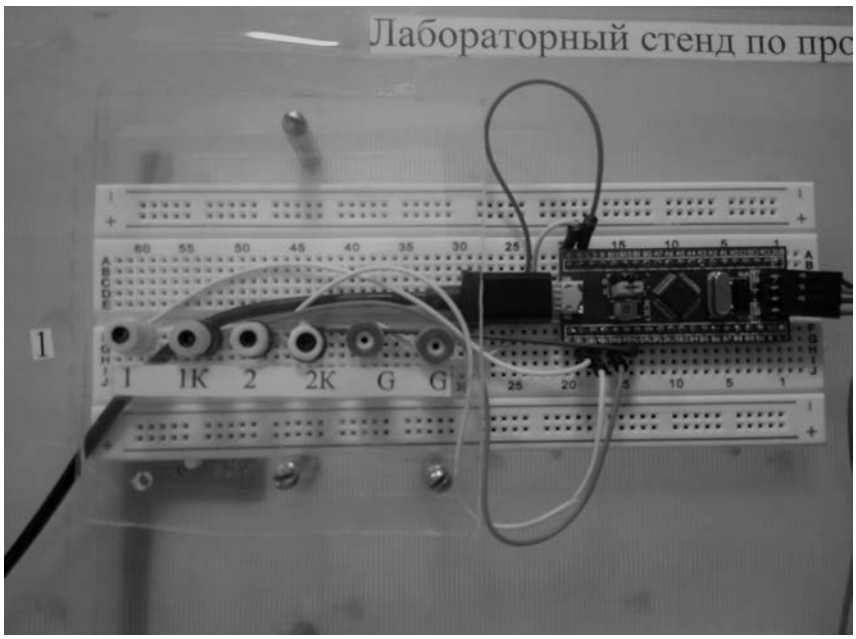


Рисунок 36 – Лабораторный стенд для исследования сдвига двух сигналов на 90°

1 – вывод первого синусоидального сигнала, 1К – вывод первого комплементарного сигнала, 2 – вывод второго синусоидального сигнала, 2К – вывод второго комплементарного сигнала, G – земля.

Задание: в соответствии с назначенным преподавателем вариантом, необходимо к каждому значению массива синусов *sinPWM* прибавить соответствующее значение из таблицы 5, скомпилировать полученный проект, загрузить рабочую программу в микроконтроллер, получить результат и продемонстрировать его преподавателю.

Таблица 5 – Значения для таблицы синусов

Вариант	1	2	3	4
Значение	50	60	70	80

Содержание отчёта:

1. Название и цель работы.
2. Код с измененными параметрами.
3. Компоновка элементов лабораторной установки, название входящих в нее элементов, в том числе основных узлов микроконтроллера.
4. Выводы о проделанной работе.

5. Подготовиться к устному опросу по лабораторной работе.

Вопросы для самоконтроля:

1. Что такое компиляция?
2. Дайте определение термину «прерывание».
3. Объясните, как в данном коде организован сдвиг сигналов на определенный электрический угол.

2.4. Характеристика лабораторной работы 2

Тема «Программирование управляющих сигналов транзисторной стойки с генерацией «мертвого» времени (DeadTime)»

Цель: программирование ШИМ – сигналов с настройкой «мертвого времени».

Описание лабораторной работы:

В процессе выполнения данной лабораторной работы, обучающиеся научатся владеть технологией разработки программного обеспечения для микроконтроллеров STM32. Познакомятся с методом генерации мертвого времени. Также вживую увидят результаты проделанной работы, что позволит наглядно убедиться в правильности выполнения задания.

Краткие теоретические сведения:

«Мертвое» время (DeadTime) – задержка по времени положительных фронтов управляющих сигналов для исключения аварийных ситуаций в стойках. Стойка (рисунок 37) — это основной элемент силовой схемы, состоит из двух последовательно соединённых транзисторов и обратных диодов, соединённых параллельно с ними.

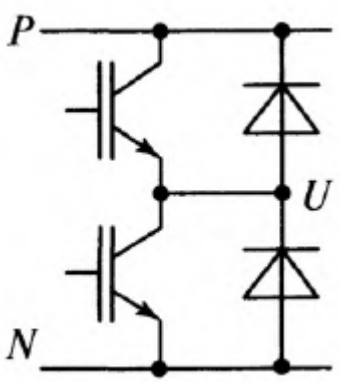


Рисунок 37 – Один из вариантов исполнения стойки

Управление стойкой происходит с помощью ШИМ, обычно для стойки с двумя транзисторами скважность задается только для верхнего ключа, а нижний ключ работает в комплементарном (со–зависимом) режиме (рисунок 38), то есть когда верхний ключ включен, то нижний выключен и наоборот, кода нижний ключ включен, верхний должен быть выключен. Такой комплементарный способ управления применяется в большинстве преобразователей. Микроконтроллер STM32F103C8T6 имеет в своем наборе таймер TIM1, с помощью которого можно назначить комплементарные выводы для генерации ШИМ – сигнала, то есть программисту необходимо задать скважность ШИМ только для верхнего ключа, а на соответствующем выводе микроконтроллера аппаратно сформируется комплементарный сигнал для нижнего ключа.

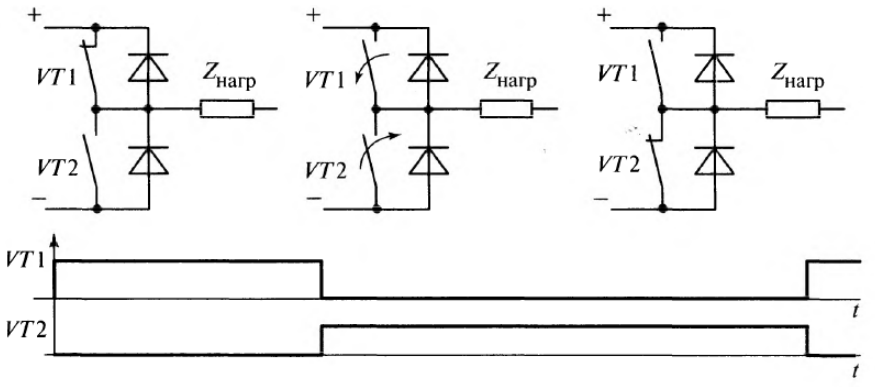


Рисунок 38 – Комплементарное управление транзисторами в стойке

Особое внимание следует обратить на момент выключения верхнего транзистора и включения нижнего. На практике время срабатывания транзисторов отлично от нуля и возможна ситуация, когда один транзистор уже успел

включиться, а другой еще не успел выключится это приводит к короткому замыканию между положительным и отрицательным контактами стойки. Ток, который возникает в таком аварийном режиме называют «сквозным». Для предотвращения таких ситуаций используют генерацию «мертвого» времени, то есть происходит смещение фронтов сигналов (рисунок 39) и возникают паузы в управлении, которые гарантируют безопасное включение и отключение транзисторов в стойке.[1]

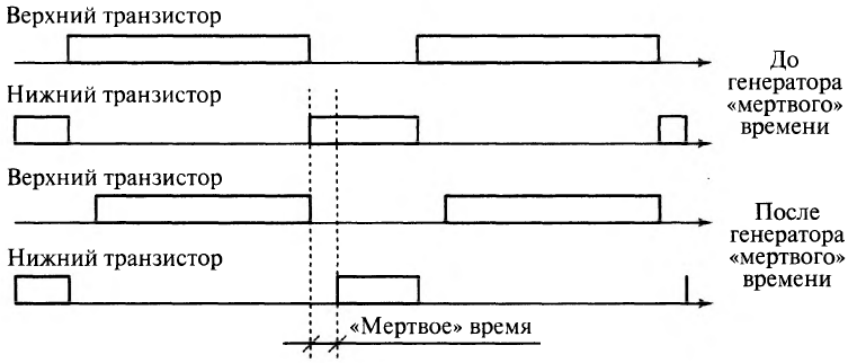


Рисунок 39 – Генерация «мертвого» времени

Содержание и порядок выполнения работы:

Представлено в ПРИЛОЖЕНИИ А.

Задание: в соответствии с назначенным преподавателем вариантом необходимо изменить значение поля *TIM_DeadTime* структуры *BDTR* на значение из таблицы 6, скомпилировать полученный проект, загрузить рабочую программу в микроконтроллер и продемонстрировать результат.

Таблица 6 – Значения для поля TIM_DeadTime

Вариант	1	2	3	4
Значение	20	30	40	50

Содержание отчёта:

1. Название и цель работы.
2. Код с измененными параметрами.
3. Компоновка элементов лабораторной установки, название входящих в нее элементов, в том числе основных узлов микроконтроллера.

4. Выводы о проделанной работе.
5. Подготовиться к устному опросу по лабораторной работе.

Вопросы для самоконтроля:

1. Что такое мертвое время (DeadTime)?
2. Что такое сквозной ток?
3. Поясните как происходит управление стойкой при помощи ШИМ.
4. Поясните способ комплементарного управления стойкой.
5. Укажите в каком поле структуры *BDTR*, указывается значение мертвого времени?

2.5. Характеристика лабораторной работы 3

Тема «Изменение скважности ШИМ – сигнала с помощью кнопки (button)»

Цель: написать программу для изменения скважности ШИМ – сигнала с помощью кнопки.

Описание лабораторной работы:

В процессе выполнения данной лабораторной работы, обучающиеся научатся владеть технологией разработки программного обеспечения для микроконтроллеров STM32. Познакомятся с особенностями инициализации такого периферийного устройства как порт ввода – вывода. Научатся использовать выводы микроконтроллера для снятия входных данных, в данном случае с кнопки. Знакомство с портами ввода – вывода наиболее важная часть в процессе изучения микроконтроллеров, так как именно они используется для взаимодействия контроллера с внешним миром. После проделанной работы, обучающиеся будут лучше ориентироваться в вопросах о разработках систем управления для «умного» дома, электропривода и так далее.

Краткие теоретические сведения:

Порты ввода – вывода (GPIO) это основной элемент любого микроконтроллера, они используются для «общения» микроконтроллера с внешним миром. Устройство портов ввода – вывода представлено на рисунке 40.

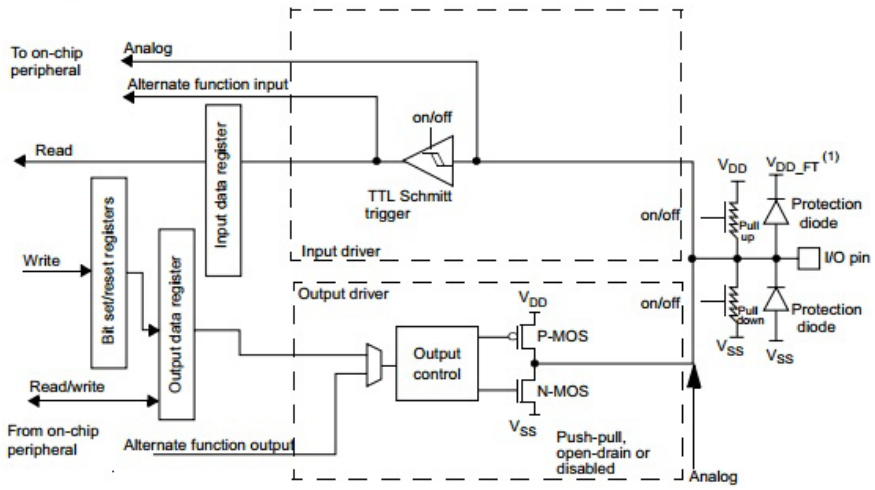


Рисунок 40 –Устройство портов ввода – вывода (GPIO)

Как видно из рисунка 40, выводы микроконтроллера можно сконфигурировать как на вход, так и на выход:

1. Входной драйвер (input driver) – позволяет сконфигурировать порт микроконтроллера как вход (аналоговый вход, альтернативная функция), с возможностью чтения входных данных, включает в себя:

а) триггер Шмитта (TTL Schmitt trigger);

б) стягивающий резистор (pull – down) –подключен к выводу G (земля), позволяет получить на выводе низкое напряжение (логический ноль);

в) подтягивающий резистор (pull – up) – подключен к питанию микроконтроллера, позволяет получить на выводе высокое напряжение (логическую единицу);

г) защитный диод (protection diode) – защищает микроконтроллер от перенапряжения.

2. Выходной драйвер (output driver) – позволяет сконфигурировать порт микроконтроллера как выход, с возможностью чтения и записи выходных данных, также возможна конфигурация порта в виде выхода альтернативной функ-

ции (например, для генерации ШИМ – сигналов). Выходной драйвер включает в себя:

а) управление выходом (output control);

б) транзисторы (P – MOS и N – MOS), используются для определения на выходе микроконтроллере высокого напряжения (логической единицы), либо низкого напряжения (логического нуля);

в) защитный диод (protection diode) – защищает микроконтроллер от перенапряжения.

В данной лабораторной работе необходимо подключать кнопку. Кнопка – это устройство при нажатии, на которое происходит замыкание контактов. Для подключения кнопки к микроконтроллеру необходимо сконфигурировать порт микроконтроллера как вход и включить чтение входных данных.

Содержание и порядок выполнения работы:

Представлено в ПРИЛОЖЕНИИ Б.

Задание: в соответствии с назначенным преподавателем вариантом, необходимо изменить значение макроса *TIM_PULSE_BUTTON_ON* на значение из таблицы 7, скомпилировать проект, загрузить рабочую программу в микроконтроллер, результат продемонстрировать.

Таблица 7 – Значения для макроса

Вариант	1	2	3	4
Значения	1600	1700	1800	2000

Содержание отчёта:

1. Название и цель работы.
2. Код с измененными параметрами.
3. Компоновка элементов лабораторной установки, название входящих в нее элементов, в том числе основных узлов микроконтроллера.
4. Выводы о проделанной работе.
5. Подготовиться к устному опросу по лабораторной работе.

Вопросы для самоконтроля:

1. Что такое кнопка?
2. Напишите фрагмент кода считывания данных с вывода микроконтроллера.
3. Для чего используется подтягивающий к питанию резистор?
4. Напишите фрагмент кода для инициализации вывода микроконтроллера как «входа».

2.6. Характеристика лабораторной работы 4

Тема «Программирование трехфазного генератора с синусоидальными ШИМ – сигналами со сдвигом 120°»

Цель: программирование ШИМ – сигналов, сдвинутых на 120°.

Описание лабораторной работы:

В процессе выполнения данной лабораторной работы, обучающиеся научатся владеть технологией разработки программного обеспечения для микроконтроллеров STM32. Познакомятся с одним из методов формирования трехфазных синусоидальных ШИМ – сигналов, сдвинутых на 120°, узнают о таком периферийном устройстве как DMA – прямой доступ к памяти, также узнают, как использовать его чтобы получать выходные сигналы заданной формы. Данная лабораторная работа позволит обучающимся глубже понять принципы работы частотных преобразователей и формируемых в них ШИМ – сигналов.

Краткие теоретические сведения:

В основе данной лабораторной работы лежит принцип формирования синусоидальных ШИМ – сигналов, сдвинутых на определенных электрический угол. Таблицы для синусов можно рассчитать по формулам 2.

$$\begin{cases} \gamma_a = U_* sin(\omega_0 t); \\ \gamma_b = U_* sin\left(\omega_0 t + \frac{2\pi}{3}\right); \\ \gamma_b = U_* sin\left(\omega_0 t - \frac{2\pi}{3}\right), \end{cases} \quad (2)$$

где U_* – относительное значение амплитуды напряжения.

Для расчета данных таблиц можно использовать любой расширенный калькулятор, например, можно воспользоваться средствами MS Excel.

Для передачи значений из массивов синусов в регистры сравнения таймера, используются каналы DMA. DMA – это прямой доступ к памяти, минуя процессор, то есть возможно передавать огромные массивы данных, при этом процессор будет «отдыхать». Это аппаратный модуль, которые располагается на системной шине и имеет доступ ко всей памяти и ко всем периферийным устройствам микроконтроллера.

Для того чтобы запустить DMA необходимо:

1. Включить модуль и канал DMA.
2. Указать адрес источника и адрес получателя.
3. Задать некоторые параметры передачи.

После этого необходимо только дать устройству DMA команду «Старт» и передача данных начнется. Независимо от объема данных эта операция происходит без участия процессора. У микроконтроллера STM32F103C8T6 в наличии имеется один 7 – канальный модуль DMA.

Возможны три направления передачи данных по DMA:

1. Периферия – память, для приема данных и складирование их в буфер.
2. Память – периферия, для передачи данных из буфера.
3. Память – память, копирование блока данных из одного участка памяти в другой.

Содержание и порядок выполнения работы:

Представлено в ПРИЛОЖЕНИИ В.

Задание: в соответствии с назначенным преподавателем вариантом, необходимо изменить значение поля *TIM_Prescaler* структуры

TIM_TimeBaseInitTypeDef, на значение из таблицы 8, скомпилировать проект, загрузить рабочую программу в микроконтроллер, результат продемонстрировать.

Таблица 8 – значения для поля TIM_Prescaler

Вариант	1	2	3	4
Значения	5	10	15	20

Содержание отчёта:

1. Название и цель работы.
2. Код с измененными параметрами.
3. Компоновка элементов лабораторной установки, название входящих в нее элементов, в том числе основных узлов микроконтроллера.
4. Выводы о проделанной работе.
5. Подготовиться к устному опросу по лабораторной работе.

Вопросы для самоконтроля:

1. Объясните, как в данном коде организован сдвиг сигналов на определенный электрический угол.
2. Что такое DMA?
3. Назовите направления передачи DMA.
4. По каким формулам строятся таблицы для ШИМ – сигналов?

2.7. Характеристика лабораторной работы 5

Тема «Регулирование скважности сигнала при помощи аналогового потенциометра с настройкой «мертвого» времени (DeadTime)»

Цель: использовать АЦП (аналогово – цифровой преобразователь) для снятия данных с потенциометра для регулирования скважности сигнала.

Описание лабораторной работы:

В процессе выполнения данной лабораторной работы, обучающиеся научатся владеть технологией разработки программного обеспечения для микроконтроллеров STM32 с использованием аналоговых датчиков, в данном слу-

чае потенциометра. Обучающиеся также познакомятся с основами тактирования микроконтроллера и выбором множителя частоты, а также предделителей, это очень важная часть, так как тактирование является основой, от настройки тактирования зависит работа периферийных устройств. Также обучающиеся увидят зависимость мертвого времени от частоты тактирования. Данная работа дает глубокое понимание основных аспектов частотного – регулирования электропривода, который нашел свое применение во многих областях.

Краткие теоретические сведения:

Настройка тактирования всегда начинается с выбора кварцевого резонатора. По умолчанию контроллер тактируется от *внутреннего кварцевого резонатора (HSI)* на 8 МГц, эта частота не может быть изменена. Но микроконтроллер STM32F103C8T6 может работать на частоте до 72 МГЦ, поэтому на отладочной плате установлен *внешний кварцевый резонатор (HSE)* на 8МГц при инициализации которого, частоту можно изменить, умножив ее на некий коэффициент. PLL – множитель частоты, коэффициент на который умножается частота внешнего кварцевого резонатора. В данном случае максимально возможный коэффициент 9. Далее настраиваются делители частоты на шинах AHB, APB1, APB2.

Данные с аналоговых датчиков, в данном случае с потенциометра всегда снимаются с помощью АЦП. АЦП – аналогово – цифровой преобразователь, это устройство, которое преобразует входной аналоговый сигнал в выходной сигнал в дискретной (цифровой) форме. Выполнен на основе компаратора. Компаратор – устройство сравнения, электронная схема, принимающая на свои входы два аналоговых сигнала, сравнивает их и в результате выдается сигнал высшего уровня напряжения.

В микроконтроллере STM32F103C8T6 в наличии имеются два 12-ти разрядных АЦП. Данные с АЦП можно считывать в буфер при помощи DMA, что реализовано в данной работе. Также запуск АЦП можно организовать по триггеру, в качестве триггера, например, можно использовать таймер, то есть АЦП будет запускаться, когда таймер досчитает до определенного значения. Данное

АЦП является 18-ти канальным, шестнадцать внешних каналов и два внутренних.

АЦП является одним из самых важных периферийных устройств микроконтроллера, так как природа вокруг нас не дискретна, а непрерывна, поэтому всевозможные датчики обычно выдают аналоговый сигнал и для перевода его в цифровой вид используется аналогово – цифровой преобразователь.

Содержание и порядок выполнения работы:

Представлено в ПРИЛОЖЕНИИ Г.

Задание: в соответствии с назначенным преподавателем вариантом необходимо изменить значение поля *TIM_DeadTime* структуры *BDTR* на значение из таблицы 9, скомпилировать полученный проект, загрузить рабочую программу в микроконтроллер и продемонстрировать результат.

Таблица 9 – Значения для поля TIM_DeadTime

Вариант	1	2	3	4
Значение	150	170	200	255

Содержание отчёта:

1. Название и цель работы.
2. Код с измененными параметрами.
3. Компоновка элементов лабораторной установки, название входящих в нее элементов, в том числе основных узлов микроконтроллера.
4. Выводы о проделанной работе.
5. Подготовиться к устному опросу по лабораторной работе.

Вопросы для самоконтроля:

1. Что такое АЦП?
2. Для чего используется PLL?
3. Объясните в чем отличие внутреннего и внешнего кварцевого резонатора в микроконтроллере STM32?
4. Какое максимальное значение множителя частоты может быть для микроконтроллера, используемого в лабораторной работе?

5. Каким образом организовано снятие данных с потенциометра в данной лабораторной работе?

2.8. Характеристика лабораторной работы 6

Тема «Программирование синусоидального и пилообразного сигнала с помощью цифро – аналогового преобразователя (ЦАП)»

Цель: использовать цифро – аналоговый преобразователь (ЦАП) для генерации синусоидального и пилообразного сигнала.

Описание лабораторной работы:

В процессе выполнения данной лабораторной работы, обучающиеся научатся владеть технологией использования ЦАП для генерации аналоговых сигналов заданной формы. Познакомятся с особенностями настройки цифро – аналогового преобразователя, узнают об использовании ЦАП совместно с триггером (в данном случае с таймером), а также совместно с DMA.

Краткие теоретические сведения:

ЦАП (DAC) – цифро – аналоговый преобразователь, устройство для преобразования входного дискретного (обычно двоичного) кода в аналоговый сигнал. ЦАП являются интерфейсом между дискретным цифровым миром и реальным аналоговым.

ЦАП в STM32F407VGT6 имеет следующие характеристики:

– напряжение от 0 до 3.3В;

– аппаратная генерация шума и треугольных импульсов;

– два независимых канала;

– возможно переключатся между 8 и 12-ти разрядным режимами с выравниванием битов по левому или по правому краю.

ЦАП использует два вывода отладочной платы PA4 и PA5, именно эти выводы задействованы в данной лабораторной работе.

ЦАП в STM32 можно использовать как генератор псевдослучайных чисел, в таком случае на цифровом осциллографе можно будет наблюдать шум. Также можно задавать амплитуду треугольных импульсов.

Синусоидальный сигнал аппаратно получить нельзя, поэтому рекомендуется использовать ЦАП совместно с DMA, для отправки значений синуса в регистр ЦАП. Также синусоидальный сигнал можно сгенерировать, используя прерывания и описать в обработчике прерывания запись значений массива синуса в регистр ЦАП [22].

Содержание и порядок выполнения работы:

Представлено в ПРИЛОЖЕНИИ Д.

Задание: в соответствии с назначенным преподавателем вариантом, необходимо изменить значение поля *DAC_LFSRUnmask_TriangleAmplitude* в структуре ЦАП для генерации пилообразных сигналов на значение из таблицы 10, проект скомпилировать, загрузить рабочую программу в микроконтроллер. Результат работы продемонстрировать.

Таблица 10 – Значения для генерации пилообразного сигнала

Вариант	1	2
Значение	DAC_TriangleAmplitude_15	DAC_TriangleAmplitude_31
Вариант	3	4
Значение	DAC_TriangleAmplitude_127	DAC_TriangleAmplitude_511

Содержание отчёта:

1. Название и цель работы.
2. Код с измененными параметрами.
3. Компоновка элементов лабораторной установки, название входящих в нее элементов, в том числе основных узлов микроконтроллера.
4. Выводы о проделанной работе.
5. Подготовиться к устному опросу по лабораторной работе.

Вопросы для самоконтроля:

1. Что такое ЦАП?
2. Какую разрядность имеет ЦАП в STM32F407VGT6?

3. Что такое триггер?

4. Каким образом в данном коде реализована генерация синусоидального сигнала?

ЗАКЛЮЧЕНИЕ

Результатом данной выпускной квалификационной работы стало создание учебного лабораторного стенда по программированию микроконтроллеров STM32.

Программирование микроконтроллеров довольно специфичная тема, требующая не только теоретической подготовки, а также постоянных практических упражнений. Невозможно изучить программирование опираясь лишь на сведения, полученные из всевозможных теоретических источников, без постоянных практических занятий навыки программирования развиваться не будут.

Результат проделанной работы можно разделить на три крупных этапа: теоретический, конструктивный и методический.

На первом этапе были рассмотрены базовые сведения о микроконтроллерах, также было сказано почему в данной разработке участвует именно микроконтроллеры фирмы STMicroelectronics. Было проведено знакомство с особенностями данных микроконтроллеров, сказано об их основных отличиях от микроконтроллеров других фирм. Также один подраздел посвящен основам программирования микроконтроллеров STM32. Рассмотрен способ генерации ШИМ – сигналов с помощью таймеров, которые являются часть микроконтроллеров.

Результатом второго этапа стало создание лабораторного стенда по программированию микроконтроллеров STM32. На нем располагаются шесть микроконтроллеров данной фирмы. Стенд был разработан с целью проведения на нем лабораторных работ для изучения программирования микроконтроллеров STM32.

На третьем этапе был разработан комплекс лабораторных работ. В комплекс лабораторных работ входят шесть работ по программированию микроконтроллеров, которые можно проводить на учебном лабораторном стенде с применением персонального компьютера. Для снятия сигналов используется

цифровой двухканальный осциллограф. Лабораторные работы позволяют ознакомиться с микропроцессорной техникой, изучить основы программирования.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Анучин А.С. Системы управления электроприводов: учебник для вузов. – Москва: Издательский дом МЭИ, 2015. – 373. с.: ил.

2. Васильев А. С., Основы программирования микроконтроллеров. – Санкт – Петербург: Университет ИТМО, 2016. – 95 с.

3. Водовозов А.М., Микроконтроллеры для систем автоматики: учебное пособие/ А. М. Водовозов. Изд. 3-е доп. и перераб. – Москва.: Инфра-Инженерия, 2016. – 164 с.: ил.; табл.

4. Гусев В.Г. Электроника и микропроцессорная техника: учебник. – 6-е издание – Москва: КНОРУС, 2013. – 800 с.

5. Дастин Э. Внедрение, управление и автоматизация / Э. Дастин, Д. Рэшка, Д. Пол; Пер. с англ. М. Павлов. – Москва: Лори, 2013. - 567 с.

6. Джозеф Ю., Ядро Cortex - M3 компании ARM. Полное руководство/ Джозеф Ю; пер. с англ. А. В. Евстифеева. – М.: Додэка-XXI, 2012. – 552с.: ил.

7. Клеменс Б. Язык С в XXI веке/пер. с английского А. А. Слинкина. – Москва.: ДМК Пресс, 2015. – 365 с.: ил.

8. Кузин, Александр Владимирович, Микропроцессорная техника: учебник: для студентов образовательных учреждений среднего профессионального образования, обучающихся по группам специальностей "Информатика и вычислительная техника", "Электротехника" / А. В. Кузин, М. А. Жаворонков. - 7-е изд., стер. - Москва: Академия, 2013. – 303 с.

9. Магда Ю. С., Программирование и отладка C/C++ приложений для микроконтроллеров ARM. – Москва.: ДМК Пресс, 2012. – 168 с.: ил.

10. Новиков В.А., Электропривод в современных технологиях. Учебник для студентов вузов. Москва: Академия, 2014 – 480 с.

11. Онищенко Г.Б. Электрический привод: учебник для студ. высш. учеб. заведений / Г. Б. Онищенко. – Москва: Издательский центр «Академия», 2013. – 288 с.

12. Огородников И.Н. Микропроцессорная техника: введение в Cortex-M3: учеб. пособие/ И.Н. Огородников. – Екатеринбург: изд-во Урал. Ун-та, 2015. – 116 с.

13. Рюмик С.М. 1000 и одна микроконтроллерная схема: энциклопедия. – Москва: Додека XXI, 2015. – 356 с.

14. Страуструп Б. Язык программирования C++, Специальное издание, Пер. с англ. – М.: Издательство Бином, 2015 г. – 1136 с.: ил.

15. Торгаев С.Н., Основы микропроцессорной техники: микроконтроллеры STM8S: учебное пособие/С.Н. Торгаев, И.С. Мусоров, Д.С. Чертихина и др.; Томский политехнический университет. – Томск: Изд-во Томского политехнического университета, 2014. – 130 с.

16. Федеральный государственный образовательный стандарт высшего образования по направлению подготовки 44.03.04 Профессиональное обучение (по отраслям), утвержденный Приказом Министерства образования и науки Российской Федерации от 01 октября 2015 г. №1085.

17. ARM – это просто [Электрон. ресурс]: научная статья: СМИ Сайт – Паяльник «cxem.net», 1999 – 2017. – Режим па: http://cxem.net/mc/mc131.php, свободный. – Загл. с экрана.

18. Таймеры общего назначения и продвинутые [Электрон. ресурс]: интернет – урок: RoboCraft, 2009 – 2017. – Режим доступа: http://robocraft.ru/blog/ARM/739.html, свободный. – Загл. с экрана.

19. Генерация ШИМ в STM32 [Электрон. ресурс]: интернет – урок: easystm32, 2012. – Режим доступа: http://easystm32.ru/for-beginners/35-pwm-in-stm32, свободный. – Загл. с экрана.

20. Brown G., Discovering The STM32 Microcontroller: work is covered by the Creative Commons Attibution-NonCommercial- ShareAlike 3.0 Unported (CC BY-NC-SA 3.0) license, 2013.

21. Datasheet STM32F103x8, STM32F103xB, DocID13587 Rev 17, 2015

22. Reay D., Digital signal processing using the ARM Cortex – M4: Published by John Wiley & Sons, Inc., Hoboken, New Jersey Published simultaneously in Canada, 2016.

23. Reference manual, STM32F101xx, STM32F102xx, STM32F103xx, STM32F105xx and STM32F107xx advanced ARM® -based 32-bit MCUs, RM0008, 2015.

24. Martin T., The Designer's Guide to the Cortex – M: The Boulevard, Langford Lane, Kidlington, Oxford, OX5 1GB 225 Wyman Street, Waltham, MA 02451, USA, 2013.

25. Noviello C., Mastering STM32, a step-by-step guide to the most complete ARM Cortex – M platform, using a free and powerful development environment based on Eclipse and GCC, 2015 – 2016.

26. Yiu J., The Definitive Guide to ARM Cortex – M0 and Cortex – M0+ Processors: The Boulevard, Langford Lane, Kidlington, Oxford, OX5 1GB 225 Wyman Street, Waltham, MA 02451, USA, 2015.