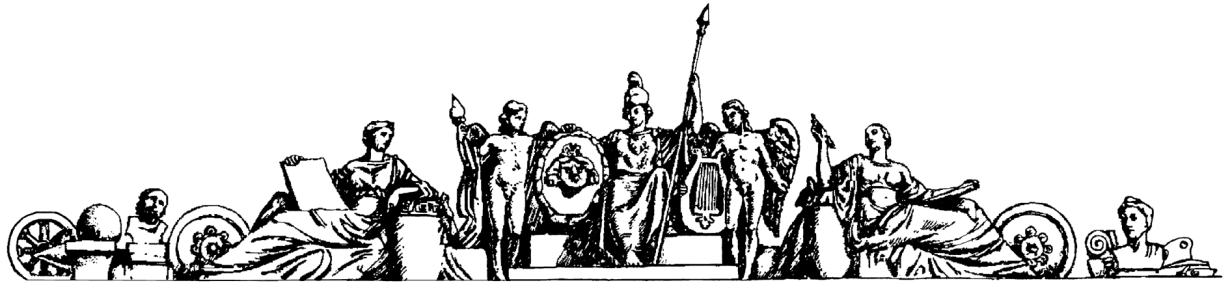
**Государственное образовательное учреждение высшего профессионального образования**

**«Московский государственный технический университет имени Н.Э. Баумана»**

**(МГТУ им. Н.Э. Баумана)**

**Факультет “СПЕЦИАЛЬНОЕ МАШИНОСТРОЕНИЕ”**

**Кафедра “СМ-7”**

**Методические указания к лабораторному практикуму по курсу «Интеллектуальные системы управления мехатронными и робототехническими системами»**

**(микропроцессорные системы управления)**

*Москва, 2022*

Рассмотрены содержание курса лабораторных работ, состав лабораторного стенда, принципы работы микропроцессорной системы управления на основе микроконтроллера STM32F030K6T6 и часть этапов разработки программного обеспечения для неё.

Для студентов, изучающих курс «Интеллектуальные системы управления мехатронными и робототехническими системами».

[РЕКОМЕНДАЦИИ ПО ВЫПОЛНЕНИЮ РАБОТЫ 3](#_Toc111453390)

[ВВЕДЕНИЕ 6](#_Toc111453391)

[СОДЕРЖАНИЕ ЛАБОРАТОРНОГО ПРАКТИКУМА 8](#_Toc111453392)

[МЕТОДИЧЕСКОЕ ОБЕСПЕЧЕНИЕ И ОБОРУДОВАНИЕ 9](#_Toc111453393)

[ОПИСАНИЕ ДОПОЛНИТЕЛЬНОГО ОБОРУДОВАНИЯ 12](#_Toc111453394)

[СОДЕРЖАНИЕ ЛАБОРАТОРНЫХ РАБОТ 15](#_Toc111453395)

[Лабораторная работа №1. Создание проекта в среде разработки. Использование портов ввода/вывода. Прерывания и их использование. Использование таймеров 15](#_Toc111453396)

[Лабораторная работа №2. Знакомство с модулем АЦП и формирование ШИМ 33](#_Toc111453397)

[Лабораторная работа №3. Построение разомкнутой системы управления двигателем постоянного тока 46](#_Toc111453398)

[Лабораторная работа №4. Построение замкнутой системы управления двигателем постоянного тока. Проверка показателей качества замкнутой системы 50](#_Toc111453399)

[Лабораторная работа №5. Исследование нейросетевого регулятора 52](#_Toc111453400)

# РЕКОМЕНДАЦИИ ПО ВЫПОЛНЕНИЮ РАБОТЫ

1. Для усвоения материала и успешного выполнения лабораторных работ **необходимо** использование соответствующей документации. Используемая документация:

* *Datasheet* на микроконтроллер (файл STM32F0.pdf) – техническая документация на *конкретный микроконтроллер*. Используется как справочная информация.
* *Reference Manual* (файл RM0360.pdf ) – подробная инструкция большого объема с описанием работы МК. Главный документ для освоения микроконтроллера. В отличие от datasheet, инструкции пишут для широкой линейки МК, в них содержится много информации о периферии, отсутствующей в конкретной модели.
* Принципиальные схемы стенда.

1. Datasheet STM32F0.pdf, Reference Manual RM0360.pdf и принципиальные схемы лабораторных стендов прилагаются к данным МУ. Дополнительные ссылки на документацию по микроконтроллеру STM32F030K6T6:

<https://www.farnell.com/datasheets/2554494.pdf>

https://www.st.com/resource/en/reference\_manual/dm00091010-stm32f030x4-x6-x8-xc-and-stm32f070x6-xb-advanced-arm-based-32-bit-mcus-stmicroelectronics.pdf

1. Рекомендуется использовать сочетания «горячих» клавиш для упрощения работы с кодом и текстом: ***Ctrl + F*** для быстрого поиска строки в коде или тексте, ***Ctrl + S*** для быстрого сохранения изменений в файле, ***Ctrl + Z*** для отмены(отката) последнего изменения в файле, ***Ctrl + A*** для выделения всего текста файла. Данные сочетания применимы для Windows.
2. Перед написанием кода программ для первых лабораторных обязательно изучите приложение А и Б, где приведен типовой листинг программ, а так же даны подробные комментарии.
3. Внимательно читайте ошибки, которые выдает вам компилятор. Например следующий вывод говорит о наличии кириллицы в названии переменных или функций(частая проблема буквы С):

[cc] D:\STM\main.c:55:22: error: stray '\321' in program

[cc] 55 | GPIO\_ToggleBits(GPIOС, GPIO\_Pin\_15);

[cc] | ^

[cc] D:\STM\main.c:55:18: error: 'GPIO' undeclared (first use in this function); did you mean 'GPIOC'?

[cc] 55 | GPIO\_ToggleBits(GPIOС, GPIO\_Pin\_15);

[cc] | ^~~~

[cc] | GPIOC

1. Нумерация элементов в языке С начинается с 0, а не с 1. Поэтому для записи некоторых значений удобно использовать следующие конструкции:

timer\_init.TIM\_Period = 1000 - 1;

Данная запись устанавливает период TIM равным 1000, а не 999 (999 – это порядковый номер значения 1000).

# ВВЕДЕНИЕ

Во всем мире разработкам технологий интеллектуального управления уделяется все большее внимание. Повышенный интерес к интеллектуальным системам управления (ИСУ) объясняется рядом причин.

Первая из них состоит в том, что традиционные технологии уже не могут обеспечить повышение качества управления, поскольку не могут учитывать всех неопределенностей, воздействующих на систему. Неопределенности системы отражают неточности модели объекта, причем как параметрические, так и структурные. Неопределенности входных сигналов отражают различную природу внешних возмущений, действующих на объект и регулятор. Совершенствование известных традиционных алгоритмов управления не всегда дает желаемый результат. Это объясняется как сложностью самих алгоритмов, так и трудностями их реализации на цифровой технике с учетом условий обеспечения устойчивости дискретной системы управления.

Вторая причина связана с тем, что в настоящее время назрела целесообразность использования преимуществ интеллектуальных технологий управления. При этом необходимо говорить о реальности применения существующей элементной базы для создания определенных классов ИСУ, относительная простота которых связана с обработкой ограниченного набора знаний в конкретной предметной области. При этом естественно возникает и требует специального исследования целый комплекс вопросов: о составе и оптимальных объемах знаний, о выборе формы их представления и способах формирования и т. д. Проблема создания новой элементной базы, например, нейросетевых структур, нечетких контроллеров и т. д., специально ориентированных на поддержку интеллектуальных технологий обработки информации и управления, остается крайне актуальным и самостоятельным направлением исследований.

Другая причина связана с тем, что дальнейшее развитие интеллектуальных технологий управления как на исполнительном уровне (интеллектуальный привод, интеллектуальный мехатронный модуль и т д.), так и на уровне организации целесообразных действий и поведения позволяет обеспечить создание принципиально нового поколения машин, обладающих высокими техническими характеристиками и функциональными возможностями.

Интеллектуальное управление развивается по следующим направлениям:

— интеллектуальное управление промышленными объектами и производственными системами,

— создание систем интеллектуального управления подвижными объектами различного назначения и транспортными средствами наземного, подводного и воздушного базирования,

— разработка средств и методов управления интеллектуальными роботами специального, промышленного, медицинского, бытового и других применений,

— разработка и создание специализированных аппаратных средств для систем интеллектуального управления

Для реализации систем интеллектуального управления необходимо использование вычислительных комплексов. В лабораторной работе в качестве вычислительного комплекса будет рассмотрен микроконтроллер в составе отладочной платы для управления исполнительным электроприводом. Функциональная схема электропривода замкнутого привода с микропроцессором в контуре управления показана на рис.1.

|  |
| --- |
|  |
| Рис.1 Структура замкнутого привода. |

# СОДЕРЖАНИЕ ЛАБОРАТОРНОГО ПРАКТИКУМА

**Цель лабораторных работ:** знакомство с этапами разработки исполнительных приводов для мехатронных и робототехнических систем с возможностью использования интеллектуальной системы.

**Основные задачи:**

1. Изучение принципов работы микропроцессорной системы управления на основе микроконтроллером.
2. Реализация АЦП и ШИМ на основе данного микроконтроллера.
3. Построение разомкнутой системы управления, снятие ЛАЧХ и ЛФЧХ, проведение анализа полученных характеристик.
4. Построение замкнутой системы управления, оценка ее параметров и проведения коррекции.
5. Синтез корректирующего устройства для полученной замкнутой системы в соответствии с требованиями технического задания, выдаваемого преподавателем.
6. Проведение анализа полученной замкнутой скорректированной системы на соответствие условиям технического задания.
7. Попытка реализации нейросетевого регулятора.

В соответствии с этими задачами практикум состоит из четырех обязательных лабораторных работ и одной факультативной лабораторной работы. Длительность каждой лабораторной работы четыре академических часа.

# МЕТОДИЧЕСКОЕ ОБЕСПЕЧЕНИЕ И ОБОРУДОВАНИЕ

Оборудование:

1. Персональный компьютер с установленной средой программирования и соответствующими драйверами.
2. Лабораторный стенд привода с микропроцессорным управлением изображен на (Рис 2).

|  |
| --- |
|  |
| Рис.2 Лабораторный стенд |

В состав лабораторного стенда входят:

* 1. Электродвигатель постоянного тока ДПМ25Н1-04, его параметры указаны в таблице 1

|  |  |
| --- | --- |
| Таблица 1 | |
| Номинальные параметры двигателя ДПМ25Н1-04 | |
| Номинальное напряжение: |  |
| Номинальная частота вращения |  |
| Номинальный ток |  |
| Номинальный момент |  |
| Пусковой момент |  |
| Сопротивление обмоток якоря |  |
| Момент инерции якоря |  |
| Электрическая постоянная времени |  |
| Электромеханическая постоянная времени |  |

* 1. Редуктор с передаточным отношением равным 133 и К.П.Д. 69%
  2. Прецизионный потенциометрический датчик ПТП-11, его основные технические параметры указаны в таблице 2

|  |  |
| --- | --- |
| Таблица 2 | |
| Основные параметры прецизионного потенциометрического датчика ПТП-11 | |
| Конструктивное исполнение | одинарный |
| Напряжение питания | +3 В |
| Номинальная величина сопротивления | 0,5 кОм |
| Мощность рассеивания | 1 Вт |
| Функциональная зависимость | прямая |
| Класс точности | I класс ±60" |
| Момент трогания ротора (20°C) | ≤ 1х10-3 Н·м |
| Момент инерции ротора | 3,0х10-7 кг·м² |
| Допустимое осевое смещение вала, мм | ±0,02 |
| Допустимое радиальное смещение вала, мм | ±0,02 |

Плата STM32F0 (рис. 3) предназначена для ознакомления с возможностями 32-битного МК на основе ARM-архитектуры, а также для реализации собственных устройств и приложений с использованием аппаратного обеспечения платы.

|  |
| --- |
| Снимок |
| Рис. 3. Внешний вид платы STM32F0 Discovery |

***Плата STM32F0 оснащена:***

* микроконтроллером STM32F030K6T6 с ядром ARM Cortex-M0 тактовой частотой 48 МГц, 32 Кб Flash-памяти, 4 Кб RAM в корпусе LQFP32;
* отладчиком ST-Link/V2 для отладки и программирования МК;
* питанием платы через USB или от внешнего источника питания 5 В;
* датчиком движения ST MEMS LIS302DL и выходами цифрового акселерометра по трем осям; - датчиком звука ST MEMS MP45DT02;
* звуковым ЦАП CS43L22;
* восемью светодиодами: LD1 (красный/зеленый) для USB-подключения, LD2 (красный) для питания 3.3 В, четыре пользовательские светодиода: LD3 (оранжевый), LD4 (зеленый), LD5 (красный), LD6 (синий) и два светодиода для USB On-The-Go – LD7 (зеленый) и LD8 (красный)
* двумя кнопками (для перепрограммирования пользователем и для перезапуска).

Оснащение отладочной платы включает большое количество периферийных устройств, что позволяет реализовывать на ней примеры различной сложности.

* 1. Усилитель мощности – LMD18200. Необходим для обеспечения управления двигателем. Представляет собой интегральную схему, состоящую из полного транзисторного моста и управляющей логической схемы. Имеет защиту от перенапряжения перегрева и сверхтока. Может питать двигатели напряжением до 55В и током до 3А. В лабораторном стенде напряжение питания составляет 27В.
  2. Восемь светодиодов.
  3. Операционный усилитель. Двуполярный сигнал с генератора, проходя через операционный усилитель, преобразуется в однополярный для согласования с уровнем, допустимым для микроконтроллера, входного сигнала.

1. Компьютеризированная измерительная лаборатория на основе ПК-осциллографов и генераторов фирмы Velleman.

# ОПИСАНИЕ ДОПОЛНИТЕЛЬНОГО ОБОРУДОВАНИЯ

**PC SCOPE 1GS/s – 50 MHz (PCS500)**

Двухканальный цифровой запоминающий осциллограф, спектральный анализатор и регистратор неустановившихся процессов (Рис. 4).

Это устройство имеет следующие преимущества:

* + внешний триггер с запуском по нарастанию и спаду сигнала;
  + улучшенная входная чувствительность;
  + пониженный уровень шумов в канале вертикального отклонения;
  + улучшенная временная развертка осциллографа;
  + возможность отображения суммы и разности сигналов на входах и инвертированного сигнала;
  + возможность отображения периодических сигналов с частотой стробирования до 1 ГГц;
  + расширенный до 25 МГц частотный диапазон спектроанализатора.

|  |
| --- |
|  |
| Рис. 4. Внешний вид приставки PCS500 |

Основные характеристики сведены в таблицу 3.

|  |  |
| --- | --- |
|  | Таблица 3 |
| Основные технические характеристики цифрового осциллографа PCS500 | |
| **Параметры** | **Тип прибора PCS500** |
| Число каналов | 2 |
| Входной импеданс | 1 МOм/30 пф |
| Разрядность, бит | 8 |
| Чувствительность | 5 мВ...15 В/дел. |
| Максимальное входное напряжение, В | 100 |
| Диапазон частот, МГц | 0...50 |
| Неравномерность АЧХ, дБ | ±3 |
| Погрешность отсчета,% | 2,5 |
| Временная развертка | 20 нс...100 мс/дел. |
| Частота стробирования | 1,25 кГц...50 МГц |
| Максимальная частота стробирования, ГГц | 1 |
| Длина записи | 4096 |
| Напряжение питания, В | 9...10 |
| Вес, г | 490 |
| Режимы входов | AC, DC, Ground |
| Интерполяция | Линейная или сглаженная |
| Маркеры | Для напряжения и частоты |

**PC FUNCTION GENERATOR 0.01 Hz to 1 MHz (PCG10/8016)**

Виртуальный ПК-генератор. Уникальная особенность генератора – его совместимость с ПК-осциллографами Velleman PCS500 для создания измерительного комплекса с расширенными возможностями отображения данных на дисплее. Внешний вид изображен на рис 5.

|  |
| --- |
|  |
| Рис. 5. Внешний вид приставки PCG10/8016 |

Основные характеристики указаны в таблице 4.

|  |  |  |
| --- | --- | --- |
|  | | Таблица 4 |
| Основные технические характеристики генератора PCS500/K8016 | | |
| **Параметр** | **Тип прибора PCG10/K8016** | |
| Диапазон частот: | 0.01 Гц - 1 МГц | |
| Источник питания: | адаптер 12 В/800 мА (PS1208) | |
| Разреш. при измер. частоты: | 0,01% | |
| Вертикальное разрешение: | 8 бит (0.4 % от полной шкалы) | |
| Диапазон амплитуды: | 100 мВ - 10 В при нагрузке 600 Ом | |
| Отклонение от нуля: | от -5В до +5В max (0.4% от полн. шкалы) | |
| Макс. частота дискретизации: | 32 МГц | |
| Коэффициент гармоник: | менее 0,08% | |
| Выходной импеданс: | 50 Ом | |
| Размеры: | 235х165х47 мм | |

# СОДЕРЖАНИЕ ЛАБОРАТОРНЫХ РАБОТ

## Лабораторная работа №1. Создание проекта в среде разработки. Использование портов ввода/вывода. Прерывания и их использование. Использование таймеров

**Цель работы:** ознакомиться с процедурой созданием проектов для МПС стенда, рассмотреть их структуру. На примере стенда изучить принципы работы с портами ввода/вывода и организации их взаимодействия; ознакомиться с понятием прерывания, возможностями, которые они предоставляют; научиться использовать таймеры для выполнения действий в определённые временные интервалы.

Основные задачи лабораторной работы №1:

1. Создание проекта в среде CoIDE.
2. Создание программы на языке С.
3. Отладка программы в среде CoIDE.
4. Знакомство с модулем ввода\вывода и таймерами
5. Выполнение задания 1.

**Создание проекта в среде CoIDE**

Запускаем программу CoIDE. После запуска программы открывается окно приветствия (Рис. 6.).

|  |
| --- |
| рис.1.jpg |
| Рис.6 Окно приветствия. |

Создаем новый проект, используя меню «Project» - «New Project» (Рис. 7)

|  |
| --- |
| рис.2.jpg |
| Рис.7 Создание нового проекта |

В открывшемся окне Project (Рис.8) задаем имя проекта и путь к папке, в которой будет располагаться проект, сняв при этом галочку «Use default path». Затем нажать «Next >».

Примечание: проект необходимо расположить по такому пути, ***не содержащему символов кириллицы***: D:\STM\Student\_projects\<SM7-\*\*M >\<Your\_Name>\

|  |
| --- |
| рис.3.jpg |
| Рис.8 Окно создания проекта |

В следующем окне Model (Рис.9) выбрать кружок с надписью «Chip» и нажать «Next >».

|  |
| --- |
| рис.4.jpg |
| Рис.9 Выбор программируемого устройства. |

В следующем окне Chip (Рис.10) выбрать, используемый микроконтроллер (STM32F030K6T6), раскрыв вкладки ST->STM32F0x-> STM32F030K6T6. Затем нажать кнопку «Finish».

|  |
| --- |
| G:\Выбор STM32.PNG |
| Рис.10 Выбор микроконтроллера |

После выполнения указанных действий получаем окно созданного проекта в CoIDE ( Рис .11).

|  |
| --- |
|  |
| Рис.11 Окно созданного проекта |

Создание проекта на этом завершено.

**Создание первой программы в CoIDE**

***Задание 1:***

Необходимо написать программу, в результате выполнения которой, МПС будет изменять состояние светодиода, подключенного к выводу Х, 1 раз в Y миллисекунд с помощью Z таймера.(См. таблицу 5)

|  |  |  |  |
| --- | --- | --- | --- |
| Таблица 5 | | | |
| Условия задания 1 | | | |
| № компьютера | Вывод X | Длительность задержки Y | Таймер Z |
| 1 | PB3 | 100 | TIM17 |
| 2 | PB4 | 200 | TIM16 |
| 3 | PB5 | 300 | TIM15 |
| 4 | PB7 | 400 | TIM14 |

Ниже будет приведен пример, реализующий мигание светодиода, подключенного к выводу 15 порта C (PC15), с задержкой 500 миллисекунд на таймере TIM14.

***Подключение к проекту необходимых библиотек.***

В результате выполнения п.1 окно программы должно выглядеть, согласно рис.11. Для подключения библиотек к проекту необходимо во вкладке «Peripherals» открытой закладки Repository установить галочки напротив интересующих библиотек, в рамках данного примера - напротив ***«RCC»(Reset and Clock control)***, ***«GPIO»(general-purpose I/O)***. Галочки напротив ***«CMSIS BOOT»*** и ***«M0 CMSIS Core»*** будут установлены автоматически, так как они содержат описания регистров ядра и периферии, стартовой инициализации микроконтроллера и др. Библиотеки необходимо использовать для упрощения настройки и использования периферии микроконтроллера. Результат выполнения изображен на рис 12.

|  |
| --- |
|  |
| Рис.12 Выбор библиотек |

В правой части основного окна CoIDE расположена вкладка «Help», где можно посмотреть краткую справку по выбранной библиотеке, и функциям содержащихся в этих библиотеках.

Примечание: если на названии вкладки нажать правую кнопку мыши и из выпадающего меню выбрать пункт «Maximize», то выбранная вкладка будет открыта на весь экран, для восстановления окон вкладок по умолчанию необходимо в том же меню выбрать пункт «Restore».

Во вкладке «Project»(левая нижняя часть экрана) представлена структура проекта, рис. 13.

|  |  |
| --- | --- |
|  | Первый уровень раскрывающегося меню - это папка проекта (название), входящие в проект элементы показаны ниже со смещением. |
| Второй и последующие уровни раскрывающегося меню - это папки, содержащиеся в проекте. |
| Заголовочные файлы и файлы библиотек, могут располагаться, как в основной папке проекта, так и в подпапках. |
| Основной файл проекта main.c, т.е. в нем по умолчанию расположена функция int main() |
| Рис.13 Структура проекта. | |

Чтобы открыть любой файл проекта, щелкните по нему два раза. После этого в основном окне появится новая вкладка с интересующим файлом.

Откройте файл main.c, на экране появится окно с текстом файла main.c (Рис.14)

|  |
| --- |
| Рис.9.jpg |
| Рис.14 Открытие main.c |

***Программа включения светодиода. Тактирование.***

Библиотеки загружены в папку проекта, но необходимо подключить их к исполняемой программе. Вставьте команду препроцессора перед функцией main:

#include "stm32\_lib/inc/stm32f0xx\_rcc.h"

Так как по умолчанию вSTM32F030K6T6 все периферийные блоки отключены от тактового генератора, то необходимо включить тактирование. Для этого необходимо определить, к какой системной шине принадлежит интересующий периферийный блок. Так как необходимо управлять выводом микроконтроллера P***C***15, то необходимо включить PORT***C***. Согласно datasheet (файл STM32F0.pdf , стр. 11 Рис. 1 Block diagram) PORTC подключен к системной шине AHB. Следовательно, используем соответствующую функцию из библиотеки RCC:

RCC\_AHBPeriphClockCmd(RCC\_AHBPeriph\_GPIOC,ENABLE);

Первый аргумент этой функции - переменная с названием периферийного модуля, возможные значения этого аргумента можно посмотреть либо в заголовочном файле **stm32f0xx\_rcc.h**, либо при наведении на функцию во всплывающем окне (Рис. 15). Удобный поиск по файлам и библиотекам можно осуществлять комбинацией клавиш ***Ctrl*** + ***F*** (Рис 16.).

|  |
| --- |
| Рис.10.jpg |
| Рис.15 Всплывающая подсказка.    Рис.16 Удобный поиск по файлу. |

Второй аргумент - команда ENABLE или DISABLE (включить, либо отключить).

Остальные функции, отвечающие за другие шины, перечислены в **stm32f0xx\_rcc.h** . Вставляем эту функцию в тело main перед циклом while(1).

***Программа включения светодиода. Настройка порта ввода-вывода.***

Для настройки интересующего вывода микроконтроллера как выхода необходимо, подключить заголовочный файл **stm32f0xx\_gpio.h**

#include "stm32\_lib/inc/stm32f0xx\_gpio.h"

Подключите данную библиотеку перед функцией main.

Затем необходимо создать структуру типа GPIO\_InitTypeDef, назовем её ledinit и вставим в тело функции main:

GPIO\_InitTypeDef ledinit;

Далее с помощью библиотечной функции заполним структуру стандартными значениями:

GPIO\_StructInit(&ledinit);

Согласно **stm32f0xx\_gpio.c** тело функции следующее:

**void** GPIO\_InitStruct(GPIO\_InitTypeDef\* GPIO\_InitStruct)

{

/\* Reset GPIO init structure parameters values \*/

GPIO\_InitStruct->GPIO\_Pin = GPIO\_Pin\_All;

GPIO\_InitStruct->GPIO\_Mode = *GPIO\_Mode\_IN*;

GPIO\_InitStruct->GPIO\_Speed = *GPIO\_Speed\_Level\_2*;

GPIO\_InitStruct->GPIO\_OType = *GPIO\_OType\_PP*;

GPIO\_InitStruct->GPIO\_PuPd = *GPIO\_PuPd\_NOPULL*;

}

Это означает, что все выводы (1 строчка функции), выбранного порта будут входами (2 вторая строчка функции), тактируются с частотой 2МГц (3 стр.), находятся в двухтактном состоянии (4 строчка) без подтяжки к линии питания или к «земле».

Далее необходимо изменить в проинициализированной структуре несколько элементов (15 вывод назначаем как выход):

ledinit.GPIO\_Pin = GPIO\_Pin\_15;

ledinit.GPIO\_Mode = GPIO\_Mode\_OUT;

С помощью функции GPIO\_Init загружаем заполненную структуру в регистры выбранного (1 аргумент функции) порта GPIOC:

GPIO\_Init(GPIOC, &ledinit);

Затем выставляем для порта GPIOC (1 аргумент функции) на его выводе GPIO\_Pin\_15 (2 аргумент функции) логическую единицу, что позволит включить светодиод:

GPIO\_SetBits(GPIOC, GPIO\_Pin\_15);

После выполнения указанных действий файл main.c будет иметь следующий вид (Рис. 17):

|  |
| --- |
|  |
| Рис. 17 Программа включения светодиода.  Сохраняем изменения в файле main.c сочетанием клавиш ***Ctrl*** + ***S*** . |

***Загрузка программы в микроконтроллер.***

В панели инструментов CoIDE расположены следующие иконки (Рис.18):

|  |  |
| --- | --- |
| Рис.12.jpg | 1-Создание проекта |
| 2-Создание нового файла |
| 3-Сохранение |
| 4-Скомпилировать проект |
| 5-Скомпилировать все файлы проекта |
| 6-Режим отладки(debugging) |
| 7- Загрузка программы в МПС |
| Рис.18 панели инструментов CoIDE | |

Скомпилируйте проект. Если компиляция прошла успешно (в окне Console появится сообщение «BUILD SUCCESSFUL»), загрузите программу в МПС. Светодиод LD6 (подключенный к выводу PC15) должен светиться.

***Включение таймера и работа с прерыванием. Подключение библиотек.***

Для подключения таймера потребуется загрузить в проект еще две библиотеки. Перейдите во вкладку «Repository» основного окна и установите галочку напротив «TIM» и «MISC».

Подключаем заголовочный файл **stm32f0xx\_tim.h**, для этого вставьте команду препроцессора вверху файла main.с:

#include "stm32\_lib/inc/stm32f0xx\_tim.h"

#include "stm32\_lib/inc/stm32f0xx\_misc.h"

В качестве таймера для создания задержки по заданию используем TIM14.

***Включение таймера и работа с прерыванием. Настройка таймера тактирования.***

Перед main создадим отдельную функцию отвечающую за настройку таймера тактирования. Назовем ее TimConfg. В main добавляем строчку TimConfg(); Данная функция ничего не принимает и ничего не выдает(тип void):

void TimConfg(void) // Настройка таймера.

{

TIM\_TimeBaseInitTypeDef Tim;

NVIC\_InitTypeDef NVIC\_I;

RCC\_APB1PeriphClockCmd(RCC\_APB1Periph\_TIM14, ENABLE); // Включаем тактирование TIM14.

// Выключаем предварительный делитель частоты таймера:

Tim.TIM\_ClockDivision = TIM\_CKD\_DIV1;

// Выключаем предварительный делитель частоты таймера.

Tim.TIM\_CounterMode = TIM\_CounterMode\_Up; // Значение в счетном регистре увеличивается.

Tim.TIM\_Prescaler = (SystemCoreClock / 1000) - 1; // Делим так чтобы на таймер шел 1 КГц.В данном случае происходит деление на 8.

Tim.TIM\_Period = 500 - 1;

// Переполнение таймера происходит каждые 500 милисек.

Tim.TIM\_RepetitionCounter = 0;

TIM\_TimeBaseInit(TIM14, &Tim);

// Инициализация таймера.

// Настройка прерывания от таймера.

NVIC\_I.NVIC\_IRQChannel = TIM14\_IRQn;

NVIC\_I.NVIC\_IRQChannelPriority = 3;

// Приоритет прерывания (0 - 3. 0 - наивысший, 3 - наименьший).

NVIC\_I.NVIC\_IRQChannelCmd = ENABLE;

// Разрешаем прерывание.

NVIC\_Init(&NVIC\_I);

TIM\_ITConfig(TIM14, TIM\_IT\_Update, ENABLE);

// Разрешаем прерывание таймера по переполнению.

TIM\_Cmd(TIM14, ENABLE); // Запуск таймера.

}

Аналогично блоку GPIOC надо определить к какой шине подключен TIM14. Согласно документации (файл STM32F0.pdf , стр. 11 Рис. 1 *Block diagram*) TIM14 подключен к шине APB, поэтому используем соответствующую функцию из библиотеки RCC для подачи тактирования:

RCC\_APB1PeriphClockCmd(RCC\_APB1Periph\_TIM14,ENABLE);

Так же создаем структуру типа TIM\_TimeBaseInitTypeDef Tim; В stm32f0xx\_tim.c можно посмотреть на тело данной функции:

**void** **TIM\_TimeBaseStructInit**(TIM\_TimeBaseInitTypeDef\* TIM\_TimeBaseInitStruct)

{

/\* Set the default configuration \*/

TIM\_TimeBaseInitStruct->TIM\_Period = 0xFFFFFFFF;

TIM\_TimeBaseInitStruct->TIM\_Prescaler = 0x0000;

TIM\_TimeBaseInitStruct->TIM\_ClockDivision = TIM\_CKD\_DIV1;

TIM\_TimeBaseInitStruct->TIM\_CounterMode = TIM\_CounterMode\_Up;

TIM\_TimeBaseInitStruct->TIM\_RepetitionCounter = 0x0000;

}

Нам интересен лишь предделитель и период таймера, которые необходимо изменить в соответствии с заданным вариантом. Предделитель используется для того, чтобы подать на таймер 1 КГц, преобразуя частоту от системной шины. Значение частоты системной шины можно посмотреть в файле system\_stm32f0xx.c (в нашем случае 48 МГц):

uint32\_t WEAK SystemCoreClock = 48000000;

Затем значение предделителя устанавливается таким, чтобы после него на таймер шла частота 1 КГц. Значение периода задается равным 499, чтобы таймер переполнялся 2 раза в секунду.  
 Затем настраивается прерывание. Название вектора прерывания (в примере TIM14\_IRQn)можно найти в REF\_MANUAL или в **stm32f0xx.h** строки 186-216 . Его приоритет устанавливается самым низким (3), но это не имеет особого значения, т. к. нет других прерываний от периферии. Функция NVIC\_Init() применяет параметры прерывания заданные в структуре NVIC\_I.  
 Далее разрешается в качестве источника прерываний переполнение таймера и запускается таймер. На этом настройка завершена. Обработчиком самого прерывания является функция TIM14\_IRQHandler, которую следует переопределить после функции main.

void TIM14\_IRQHandler(void)

// Обработчик прерывания от таймера.

{

GPIO\_ToggleBits(GPIOB, GPIO\_Pin\_15);

// Мигаем светодиодом.

TIM\_ClearITPendingBit(TIM14, TIM\_IT\_Update);

// Сброс флага прерывания по переполнению таймера.

}

Примечание:

Название функции-обработчика прерывания может быть отличным от TIM14\_IRQHandler, указанного в примере. Смотрим необходимое название в **startup\_stm32f0xx.s.**

Функция GPIO\_ToggleBits() изменяет состояние выходов на противоположное (для ряда МК идет по умолчанию в stm32f?xx\_gpio.h). Она не включена в список стандартных функций для нашего микроконтроллера, поэтому определим ее самостоятельно. Добавим следующий код перед функцией main.c:

// Функция инвертирует состояние вывода.

**void** **GPIO\_ToggleBits**(GPIO\_TypeDef\* GPIOx, uint16\_t GPIO\_Pin)

{

/\* Check the parameters \*/

assert\_param(IS\_GPIO\_ALL\_PERIPH(GPIOx));

GPIOx->ODR ^= GPIO\_Pin;

}

В итоге два раза в секунду будет происходить прерывание от таймера(мигание диода два раза в секунду).

После выполнения всех указанных действий скомпилируйте и загрузите программу в МПС. В результате выполнения программы светодиод должен мигать с необходимой частотой. Для проверки частоты переключения подключите осциллограф к нужному выводу МПС.

Листинг программы приведен в приложении А.

***Режим отладки.***

Для входа в режим отладки нажмите на соответствующую иконку («жук» 6 по порядку) в панели инструментов или сочетание клавиш Ctrl+F5. После выполнения указанных действий получаем окно отладки CoIDE (Рис. 19):

|  |
| --- |
| Рис.13.jpg |
| Рис.19 Режим отладки |

Элементы управления процессом отладки представлены на рис 20.

|  |  |
| --- | --- |
| Рис.14.jpg | 1- Показывает окно с дизассемблированным кодом программы и пошаговое его выполнение. |
| 2 - Сброс микроконтроллера |
| 3- Запуск программы на выполнение |
| 4- Остановка выполнения программы |
| 5- Выход из режима отладки |
| 6- Шаг со входом внутрь функции |
| 7- Шаг с пропуском операций внутри функции |
| 8- Шаг с остановкой при выходе из функции |
| 9- Выполнения программы до места, в которое установлен курсор |
| Рис.20 Элементы управления процессом отладки | |

Для установки точки останова необходимо два раза щелкнуть слева от номера строки. После установки появится красная точка напротив необходимой строки (Рис. 21).

|  |
| --- |
| Рис.15.jpg |
| Рис.21 Точка останова |

Установите точку останова в тело обработчика прерываний и убедитесь, что МПС заходит в обработчик прерываний.

В нижнем правом углу находится вкладка Variables. В неё можно добавлять переменные, значения которых необходимо контролировать.. Контролировать значения возможно только при остановке программы.

## Лабораторная работа №2. Знакомство с модулем АЦП и формирование ШИМ

Цель работы: ознакомиться с возможностями генерации ШИМ с помощью демонстрационной платы; научиться генерировать сигнал с заданными параметрами и использовать его для управления внешними устройствами; исследовать возможности использования АЦП.

Основные задачи лабораторной работы №2:

1. Создание ШИМ сигнала с помощью модуля таймера.
2. Знакомство с модулем АЦП.
3. Выполнение задания 2.

***Создание ШИМ сигнала с помощью модуля таймера***

Широтно-импульсная модуляция (ШИМ) – способ управления средним значением напряжения на нагрузке путем изменения скважности импульсов, управляемых ключом. В основном, микроконтроллеры позволяют генерировать цифровой сигнал ШИМ различной частоты.

Поскольку формирования сигнала ШИМ не является основной функцией выводов микроконтроллера, которые подключены к светодиодам, то необходимо выполнить соответствующую их конфигурацию. Для этого следует настроить выводы для выполнение альтернативных функций. Генерация ШИМ в них связана с использованием дополнительных режимов таймера.

Особенностью данных микроконтроллеров является то, что в предделитель и другие регистры можно записать любое значение, которое можно описать с помощью отведенного количества разрядов.

***Задание 2:***

Необходимо написать программу, в результате выполнения которой, МПС будет считывать сигнал задающего генератора с вывода PA1 и формировать сигнал пропорциональный заданию с использованием широтной-импульсной модуляции на выводе X с частой ШИМ (Y±5%) кГц при Z реализации. (См. таблицу 2)

|  |  |  |  |
| --- | --- | --- | --- |
| Таблица 2 | | | |
| Условие задания 2 | | | |
| № компьютера | Вывод X | Частота Y | Реализация Z |
| 1 | PB5 | 5 | Аппаратная |
| 2 | PB4 | 6 | Программная |
| 3 | PB6 | 7 | Аппаратная |
| 4 | PB7 | 8 | Программная |

Ниже будет приведен пример, реализующий считывание величины сигнала задающего генератора с вывода PA1 и формирование сигнала, пропорционального заданному сигналу, с использованием широтной-импульсной модуляции с частотой (9±0.45) кГц на выводе PB0 аппаратно, а на выводе PB1 программно.

***Подключение к проекту необходимых библиотек***

Необходимо к проекту подключить еще одну библиотеку, для этого:

1. Перейдите в закладку Repository во вкладку «Peripherals»
2. Установите галочку напротив «ADC»
3. Проверьте, что установлены галочки напротив «CMSIS BOOT», «M0 CMSIS Core», «RCC», «GPIO», «TIM», «MISC».

В результате этих действий структура проекта должна быть как на рис 21.

|  |
| --- |
|  |
| Рис.21 Новая структура проекта |

***Настройка порта ввода-вывода***

Для настройки интересующих выводов микроконтроллера необходимо подключить заголовочный файл stm32f0xx\_gpio.h

#include "stm32\_lib/inc/stm32f0xx\_gpio.h"

Вставьте команду препроцессора перед функцией main.

Так как в примере необходимо реализовать и аппаратную ШИМ и программную, то один выход потребуется настроить как выполняющий альтернативную аппаратную функцию, а другой как цифровой выход, соответственно. Также потребуется настроить один вывод как аналоговый вход для модуля АЦП.

***Назначение выводов, выполняющих альтернативные функции (необходимо для аппаратной ШИМ)***

Для данного блока реализуем отдельную функцию:

void GPIO\_Ap\_SHIM(void) {

}

Реализуем тело функции. Создаем структуру типа GPIO\_InitTypeDef, назовем её ledinit\_AF:

GPIO\_InitTypeDef ledinit\_AF;

С помощью библиотечной функции заполним структуру стандартными значениями:

GPIO\_StructInit(&ledinit\_AF);

Далее необходимо изменить в проинициализированной структуре несколько элементов:

ledinit\_AF.GPIO\_Pin = GPIO\_Pin\_0;

ledinit\_AF.GPIO\_Mode = GPIO\_Mode\_AF;

Назначаем 0 вывод, как выполняющий альтернативную функцию.

Теперь необходимо связать вывод с конкретной периферией. В документе STM32F0.pdf с стр.29 в ***Table 11*** указано, какие периферийные блоки имеют выходы на каждый из выводов. Согласно таблице, на вывод PB0 выходит ***TIM3\_CH3*** - 3 канал 3 таймера (зафиксируйте номер канала таймера, понадобится в дальнейшем). Чтобы подключить вывод PB0 к таймеру используйте функцию:

GPIO\_PinAFConfig(GPIOB, GPIO\_PinSource0, GPIO\_AF\_TIM3);

1 аргумент - название порта, 2 аргумент - номер вывода, 3 аргумент -указывает с чем будет соединен вывод (все возможные варианты указаны в stm32f0xx\_gpio.h строки с 235 по 338).

Инициализируем структуру

GPIO\_Init(GPIOB, &ledinit\_AF);

***Назначение вывода, выполняющего функцию цифрового выхода (необходимо для программной ШИМ)***

Для данного блока реализуем отдельную функцию:

void GPIO\_Progr\_SHIM(void)

{

}

Реализуем тело функции. Создаем структуру типа GPIO\_InitTypeDef, назовем её ledinit:

GPIO\_InitTypeDef ledinit;

С помощью библиотечной функции заполним структуру стандартными значениями.

GPIO\_StructInit(&ledinit);

Необходимо изменить в проинициализированной структуре несколько элементов:

ledinit.GPIO\_Pin = GPIO\_Pin\_1;

ledinit.GPIO\_Mode = GPIO\_Mode\_OUT;

Назначаем 13 вывод цифровым выходом.

GPIO\_Init(GPIOB, &ledinit);

Загружаем заполненную структуру в регистры порта.

***Назначение пина, выполняющего функцию аналогового входа***

Аналогично задается режим аналогового входа:

void GPIO\_Out(void)

{

GPIO\_InitTypeDef GPIO\_InitStructure;

GPIO\_StructInit(&GPIO\_InitStructure);

GPIO\_InitStructure.GPIO\_Pin = GPIO\_Pin\_1;

GPIO\_InitStructure.GPIO\_Mode = GPIO\_Mode\_AN;

GPIO\_InitStructure.GPIO\_PuPd = GPIO\_PuPd\_NOPULL;

GPIO\_Init(GPIOA, &GPIO\_InitStructure);

}

Где GPIO\_Mode\_AN показывает, что ножка PA1 будет работать в аналоговом режиме.

***Настройка таймера***

Для программной ШИМ подойдет любой таймер, кроме TIM6, TIM7, так как это базовые таймеры, не имеющие каналов сравнения.

Для настройки интересующих таймеров микроконтроллера необходимо подключить заголовочный файл stm32f0xx\_tim.h:

#include "stm32\_lib/inc/stm32f0xx\_tim.h"

Создаем отдельную функцию SHIM\_Conf и заполняем ее тело.

Создадим структуру типа TIM\_TimeBaseInitTypeDef, назовем её timer\_init:

TIM\_TimeBaseInitTypeDef timer\_init;

Затем заполним структуру стандартными значениями с помощью вызова библиотечной функции

TIM\_TimeBaseStructInit(&timer\_init);

Настроим таймер так, чтобы его ***переполнение*** происходило с частотой близкой к 9 кГц.

В проинициализированной структуре таймера необходимо изменить несколько элементов:

timer\_init.TIM\_Period = 111 - 1;

timer\_init.TIM\_Prescaler = (SystemCoreClock / 1000) - 1;

Предделитель устанавливаем так , что 1 тик таймера будет происходить с частотой 1 МГц.

Период равен 111, значит, переполнение таймера будет происходить с частотой 9.009 КГц.

Загружаем заполненную структуру в регистры выбранного таймера (1 аргумент функции).

TIM\_TimeBaseInit(TIM3, &timer\_init);

После команд включения библиотек необходимо вставить объявление глобальной переменной, она будет необходима для ***задания скважности ШИМ***:

volatile uint16\_t PWM\_val = 10;

Создадим структуру типа TIM\_OCInitTypeDef, назовем её tim\_OC\_init:

TIM\_OCInitTypeDef tim\_OC\_init;

Вставим в тело создаваемой функции. Это структура описывает взаимодействие таймера с выходами.

Затем заполним структуру стандартными значениями вызовом библиотечной функции

TIM\_OCStructInit(&tim\_OC\_init);

Согласно stm32f0xx\_tim.c тело функции следующее:

void TIM\_OCStructInit(TIM\_OCInitTypeDef\* TIM\_OCInitStruct)

{

/\* Set the default configuration \*/

TIM\_OCInitStruct->TIM\_OCMode = TIM\_OCMode\_Timing;

TIM\_OCInitStruct->TIM\_OutputState = TIM\_OutputState\_Disable;

TIM\_OCInitStruct->TIM\_OutputNState = TIM\_OutputNState\_Disable;

TIM\_OCInitStruct->TIM\_Pulse = 0x00000000;

TIM\_OCInitStruct->TIM\_OCPolarity = TIM\_OCPolarity\_High;

TIM\_OCInitStruct->TIM\_OCNPolarity = TIM\_OCPolarity\_High;

TIM\_OCInitStruct->TIM\_OCIdleState = TIM\_OCIdleState\_Reset;

TIM\_OCInitStruct->TIM\_OCNIdleState = TIM\_OCNIdleState\_Reset;

}

Это означает, что:

* канал сравнения таймера не оказывает влияния на вывод, связанный с таймером (1 стр. тела),
* связь с выводом отключена (2-3 стр. тела),
* 0 – значение, при котором происходит событие сравнения (4 стр. тела),
* состояние вывода после момента сравнения - высокое (5-6 стр. тела),
* начальное состояние вывода - низкое (8-9 стр. тела).

Все параметры с символом N (3,6,9 стр. тела) относятся к комплементарным выходам (только TIM1 и TIM8).

Необходимо изменить несколько значений в проинициализированной структуре:

tim\_OC\_init.TIM\_Pulse = PWM\_val;

tim\_OC\_init.TIM\_OCMode = TIM\_OCMode\_PWM1;

tim\_OC\_init.TIM\_OutputState = TIM\_OutputState\_Enable;

Это означает, что:

* значение, при котором происходит событие сравнения, равно скважности (1 стр. тела),
* режим таймера ШИМ (подробнее смотри Ref Manual) (2 стр. тела),
* связь с выводом включена (3 стр. тела).

Инициализируем структуру

TIM\_OC1Init(TIM3, &tim\_OC\_init);

Разрешаем прерывание. Название обработчика можно найти в файле startup\_stm32f0xx.s. Для разрешения прерывания заранее в теле функции создана переменная NVIC\_InitTypeDef NVIC\_I;

// Настройка прерывания от таймера.

NVIC\_I.NVIC\_IRQChannel = *TIM3\_IRQn*;

NVIC\_I.NVIC\_IRQChannelPriority = 3;

// Приоритет прерывания (0 - 3. 0 - наивысший, 3 - наименьший).

NVIC\_I.NVIC\_IRQChannelCmd = *ENABLE*;

// Разрешаем прерывание.

NVIC\_Init(&NVIC\_I);

Данный код аналогичен команде:

NVIC\_EnableIRQ(TIM3\_IRQn);

Далее необходимо включить генерацию прерывания по переполнению.

TIM\_ITConfig(TIM3, TIM\_IT\_Update, ENABLE);

*Для программного формирования ШИМ* необходимо включить генерацию прерывания по событию сравнения (для аппаратного нет необходимости):

TIM\_ITConfig(TIM3, TIM\_IT\_CC4, ENABLE);

Значения вторых аргументов данных функций можно найти в файле stm32f0xx\_tim.h строки 581-589.

Примечание: Обратите внимание, что TIM\_IT\_CC4 прерывание по событию сравнения только с четвертым каналом.

В конце функции включаем полученный таймер:

TIM\_Cmd(TIM3, ENABLE);

***Функции обработка прерывания.***

Обработчик для *программного* формирования ШИМ при переполнении таймера выставляет логическую единицу на выводе PB1, и логический ноль, если произошло событие сравнения:

void TIM3\_IRQHandler() {

if (TIM\_GetITStatus(TIM3, TIM\_IT\_Update) != RESET) {

TIM\_ClearITPendingBit(TIM3, TIM\_IT\_Update);

TIM\_SetCompare1(TIM3,PWM\_val);

if (PWM\_val > 0) GPIO\_SetBits(GPIOD, GPIO\_Pin\_11);

}

if (TIM\_GetITStatus(TIM3, TIM\_IT\_CC4) != RESET) {

TIM\_ClearITPendingBit(TIM3, TIM\_IT\_CC4);

if (PWM\_val < 109) GPIO\_ResetBits(GPIOB, GPIO\_Pin\_1);

}

}

Проверяем, по какому событию произошло прерывание (2 аргумент функции), возвращает RESET (0), если прерывания не было:

TIM\_GetITStatus(TIM3, TIM\_IT\_Update)

Очищаем флаг выбранного прерывания:

TIM\_ClearITPendingBit(TIM3, TIM\_IT\_Update);

Вызываем функцию, которая загружает в регистр таймера, выбранного в 1 аргументе функции, значение (2 аргумент функции), с которым будет идти сравнение (единица означает, что загрузка произойдет в регистр сравнения только первого канала):

TIM\_SetCompare1(TIM3,PWM\_val);

Два условия if необходимы для того, чтобы избежать неопределенности при одновременном срабатывании двух типов прерываний: событие сравнения и переполнение таймера.

Обработчик только для *аппаратного формирования ШИМ* выглядел бы следующим образом:

void TIM4\_IRQHandler() {

TIM\_ClearITPendingBit(TIM3, TIM\_IT\_Update);

TIM\_SetCompare1(TIM3,PWM\_val);

}

***Настройка АЦП***

Для настройки модуля АЦП необходимо, подключить заголовочный файл :

#include "stm32\_lib/inc/stm32f0xx\_adc.h"

Реализуем отдельную функцию ADC\_Conf(void) для настройки АЦП. Внутри функции создаем структуру АЦП:

ADC\_InitTypeDef ADC\_InitStructure;

Проинициализируем обе структуры:

ADC\_StructInit(&ADC\_InitStructure);

Данная структура описана в файле stm32f0xx\_adc.c.

Внутри структуры можно задать следующие параметры:

* АЦП будет настроен на получение результата в разрешении 12 бит ;
* режим последовательного измерения нескольких выходов выключен;
* режим непрерывного измерения выключен ;
* фронт, по которому будет начинаться преобразование, не выбран(событие сравнения канала таймера является событием для начала преобразования);
* данные выравнены по правому краю ;
* количество измерений 1 ;

Так как все настройки по умолчанию, удовлетворяют нашим требованиям, загрузим значение структуры в регистры:

ADC\_Init(ADC1, &ADC\_InitStructure);

Включаем прерывание по окончании преобразования:

ADC\_ITConfig(ADC1, ADC\_IT\_EOC, ENABLE);

Виды прерываний можно найти в REF\_MANUAL, а запись их в виде аргументов функции - в stm32f0xx\_adc.h .

Включаем модуль АЦП:

ADC\_Cmd(ADC1, ENABLE);

Разрешаем прерывание от АЦП, название обработчика можно найти в stm32f4xx.h :

NVIC\_EnableIRQ(ADC1\_IRQn);

Далее задаем номер модуля АЦП, номер канала, порядковый номер измеряемого канала (при измерении нескольких каналов показывает, каким по порядку будет измерен канал), время, в течение которого будет происходить преобразование:

ADC\_ChannelConfig(ADC1,ADC\_Channel\_4,ADC\_SampleTime\_7\_5Cycles);

Второй аргумент - номер канала, его можно найти в файле STM32F0.pdf (или stm32f0xx\_adc.h),последний аргумент - порядковый номер и возможные варианты последнего аргумента в (stm32f0xx\_adc.h).

Запускаем преобразование.

ADC\_SoftwareStartConv(ADC1);

Далее необходимо написать ***обработчик прерывания АЦП***. Обработчик копирует преобразованное значение в переменную, с коэффициентом, который обеспечивает при максимальном значении АЦП (4095) максимальное значение скважности ШИМ (110) и запускает новое преобразование.

void ADC\_IRQHandler() {

if (ADC\_GetITStatus(ADC1, ADC\_IT\_EOC) != RESET) {

ADC\_ClearITPendingBit(ADC1, ADC\_IT\_EOC);

PWM\_val = (110 \* (ADC\_GetConversionValue(ADC1)))/ 4095;

ADC\_RegularChannelConfig(ADC1, ADC\_Channel\_4, 1, ADC\_SampleTime\_239\_5Cycles);

ADC\_SoftwareStartConv(ADC1);

}

}

Примечание: ADC\_GetConversionValue(ADC1) - функция, которая получает преобразованное значение из модуля АЦП, указанного в аргументе функции.

***Тактирование модулей***

Как и в прошлой ЛР для тактирования интересующей периферии микроконтроллера необходимо подключить заголовочный файл stm32f0xx\_rcc.h

В завершении остаётся включить тактирование всех используемых модулей. Вставьте в начало функции main следующие строку:

SystemInit();

***В начале каждой функции*** необходимо указать какую периферию необходимо использовать:

RCC\_AHB1PeriphClockCmd(RCC\_AHB1Periph\_GPIOA, ENABLE);

RCC\_AHB1PeriphClockCmd(RCC\_AHB1Periph\_GPIOB, ENABLE);

RCC\_APB1PeriphClockCmd(RCC\_APB1Periph\_TIM3, ENABLE);

RCC\_APB2PeriphClockCmd(RCC\_APB2Periph\_ADC1, ENABLE);

Листинг программы приведен в приложении Б.

***Проверка***

Для проверки правильности работы подключите генератор сигнала к стенду. На экране будет видно, как скважность ШИМ изменяется пропорционально сигналу генератора. Для проверки правильности периода ШИМ в меню осциллографа необходимо включить курсоры и произвести измерение периода.

## Лабораторная работа №3. Построение разомкнутой системы управления двигателем постоянного тока

**Цель работы**: используя навыки, полученные на предыдущих работах, построить разомкнутую систему управления двигателем постоянного тока;

Основные задачи лабораторной работы №3:

* Измерение входного сигнала при помощи АЦП и запись полученного цифрового кода в регистр ШИМ;
* Контроль правильности работы ШИМ при помощи осциллографа;
* Реализация импульсного управления двигателем;
* Определение положительного/отрицательного направления вращения двигателя и коммутация этого сигнала на драйвер двигателя;
* Контроль скорости и направления вращения двигателя;
* Получение ЛФЧХ и ЛФЧХ разомкнутой системы;
* Построение математической модели разомкнутой системы в MATLAB.

-Получить техническое задание на разработку привода у преподавателя.

**Принцип работы системы:**

На вход модуля ADC1 (вывод PA1) микроконтроллера STM32 поступает синусоидальный сигнал управления с генератора сигналов в диапазоне от [-1.5B, +1.5В] предварительно инвертированный и поднятый по уровню на 1.5В операционным усилителем. Сигнал оцифровывается c разрешением 12 бит. Полученное значение с АЦП, лежащее в диапазоне [0, 4095] преобразуется в [0, NШИМmax] и поступает в модуль ШИМ (TIM3\_CH3, вывод PB0) в качестве скважности. ШИМ сигнал формируется c частотой от 5 кГц до 10 кГц, а направление вращения определяется относительно середины диапазона [0, 4095]. Знак сигнала поступает отдельно на вывод PE9. ШИМ генерирует меандр, поступающий на УМ, а с него непосредственно на якорную обмотку двигателя. Таким образом, осуществляется импульсное управление двигателем постоянного тока. Функциональная схема разомкнутой системы представлена на рис. 22.

Предварительно производится настройка сигнала нулевого уровня, поступающего с генератора специальным построечным резистором.

|  |
| --- |
|  |
| Рис. 22. Функциональная схема управления двигателем постоянного тока |

Для построения ЛАЧХ и ЛФЧХ разомкнутой системы необходимо измерять амплитуду входного сигнала и сигнала, снимаемого с датчика обратной связи.

По снятым данным рассчитать амплитуду и фазу передаточной функции разомкнутой системы в различных точках по формулам:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  |  | |  | |
| Без учета поправки | |  | |  |
| С учетом поправки | |  | |  |

*T,f –* период и частота входного сигнала;

, - амплитуды выходного и входного сигналов;

– амплитуда разомкнутой передаточной функции

*-* задержка между сигналами;

– фаза разомкнутой передаточной функции;

Как известно, в системах подобной двигателю постоянного тока (2 апериодических звена или колебательное звено) выходной сигнал запаздывает относительно входного, поэтому при измерении запаздывания между сигналами нужно учитывать эту особенность. Если измерять задержку между пиком входного сигнала и последующим пиком выходного сигнала, то поправку вносить не нужно (Рис.23). Если измерять задержку между пиком входного сигнала и предшествующим пиком выходного сигнала, то нужно внести поправку.

|  |
| --- |
|  |
| Рис. 23 Эксперимент без внесения поправки |

Заполнить таблицу (количество экспериментов можно изменять по усмотрение студента, частота не более 10 Гц):

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Таблица 13 | | | | | |
| Измеренные параметры | | | | | |
| № | Гц | Входной В | Выходной В | Задержка с | Поправка |
| 1 | 0,025 |  |  |  |  |
| 2 | 0,050 |  |  |  |  |
| 3 | 0,080 |  |  |  |  |
| 4 | 0,090 |  |  |  |  |
| 5 | 0,100 |  |  |  |  |
| 6 | 0,150 |  |  |  |  |
| 7 | 0,200 |  |  |  |  |
| 8 | 0,300 |  |  |  |  |
| 9 | 0,400 |  |  |  |  |
| 10 | 0,500 |  |  |  |  |
| 11 | 0,600 |  |  |  |  |
| 12 | 0,700 |  |  |  |  |
| 13 | 0,800 |  |  |  |  |
| 14 | 0,900 |  |  |  |  |
| 15 | 1,000 |  |  |  |  |
| 16 | 1,500 |  |  |  |  |
| 17 | 2,000 |  |  |  |  |
| 18 | 2,500 |  |  |  |  |
| 19 | 3,000 |  |  |  |  |
| 20 | 3,500 |  |  |  |  |
| 21 | 4,000 |  |  |  |  |
| 22 | 4,500 |  |  |  |  |
| 23 | 5,000 |  |  |  |  |
| 24 | 6,000 |  |  |  |  |
| 25 | 7,000 |  |  |  |  |

Затем произвести расчет по формулам и записать их в таблицу\*, согласно образцу

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Таблица 14 | | | | |
| Рассчитанные параметры | | | | |
| № | Период с | Частота рад/c | Амплитуда | Фаза |
| 1 | 40,00 | 0,157 | 13,98 | 68,94 |
| \*\*\* | | | | |
| 24 | 0,17 | 37,699 | -32,31 | 176,40 |
| 25 | 0,14 | 43,982 | -33,47 | 185,40 |

По полученным данным построить экспериментальную ЛАЧХ и сравнить её с полученной теоретически.

## Лабораторная работа №4. Построение замкнутой системы управления двигателем постоянного тока. Проверка показателей качества замкнутой системы

**Цель работы**: Синтезировать и реализовать систему управления электроприводом постоянного тока, отвечающую условиям технического задания:

Основные задачи лабораторной работы №4:

* Реализация корректирующего устройства в MATLAB;
* Реализация корректирующего устройства и замкнутой системы на лабораторном стенде;
* Оценка качества замкнутой системы на соответствие техническому заданию.

***Принцип работы системы:***

На вход первого модуля ADC1 (вывод PA1) микроконтроллера STM32 поступает синусоидальный сигнал управления с генератора сигналов в диапазоне от [-1.5B, +1.5В] предварительно инвертированный и поднятый по уровню на 1.5В операционным усилителем. На вход второго модуля ADC2 (вывод PA2) - сигнал положения ротора двигателя с датчика угла. Эти сигналы оцифровываются c разрешением 12 бит, после чего из сигнала управления вычитается сигнал угла по формуле: (data\_adc\_1 - data\_adc\_2) – получается значение ошибки управления. Вычисленная разность проходит через последовательное корректирующее устройство. Полученное значение, лежащее в диапазоне [0, 4095] преобразуется в [0, NШИМmax] и поступает в модуль ШИМ (TIM3\_CH3, вывод PB0) в качестве скважности. ШИМ сигнал формируется c частотой от 5 кГц до 10 кГц, а направление вращения определяется относительно середины диапазона [0, 4095]. Знак сигнала поступает отдельно на вывод PE9. Модуль ШИМ генерирует меандр, поступающий на УМ, а с него непосредственно на якорную обмотку двигателя. Таким образом, осуществляется импульсное управление двигателем постоянного тока. Функциональная схема разомкнутой системы представлена на рис. 23.

Предварительно производится настройка сигнала нулевого уровня, поступающего с генератора специальным построечным резистором.

|  |
| --- |
|  |
| Рис. 24. Функциональная схема управления двигателем постоянного тока |

## Лабораторная работа №5. Исследование нейросетевого регулятора

**Цель работы**: Синтезировать и реализовать нейросетевой регулятор для управления электроприводом постоянного тока, отвечающего условиям технического задания:

Основные задачи лабораторной работы №5:

* Выбрать тип нейросетевого регулятора;
* Определить архитектуру нейросетевого регулятора.
* Сформировать учебные, контрольные и проверочные выборки, соответствующие выбранной архитектуре системы.
* Обучить нейросетевой регулятор.
* Проверить качество работы нейросетевого регулятора на модели
* Реализовать нейросетевой регулятор на лабораторном стенде;
* Оценка качества системы с нейросетевым регуляторм на соответствие техническому заданию.

**Пояснение к выбору типа нейросетевого регулятора:**

Для реализации нейросетевого регулятора можно использовать один из предложенных типов:

* Подражающее нейроуправление.

Это системы нейроуправления, в которых нейросетевой регулятор обучается на примерах динамики обычного контроллера по обратной связи, построенного, например, на основе корректирующих устройств, синтезированных с помощью ЛАЧХ. После обучения нейронная сеть в точности воспроизводит функции исходного контроллера. В качестве примеров динамики контроллера может быть использована запись поведения человека-оператора. Обычный контроллер по обратной связи (или человек-оператор) управляет объектом управления в штатном режиме. Значения величин на входе и выходе контроллера протоколируются, и на основе протокола формируется обучающая выборка.

* Инверсное управление

В схеме обобщенного инверсного нейроуправления в качестве контроллера используется нейронная модель инверсной динамики объекта управления, называемая инверсный нейроэмулятор. Инверсный нейроэмулятор представляет собой нейронную сеть, обученную в режиме офлайн имитировать обратную динамику объекта управления на основе записанных траекторий поведения динамического объекта. Для получения таких траекторий, на объект управления в качестве управляющего сигнала подают некоторый случайный процесс. Значения управляющих сигналов и ответных реакций объекта протоколируют и на этой основе формируют обучающую выборку. В результате нейронная сеть по требуемому положению углового положения вала формирует управляющее воздействие на систему.

**Приложение А**

Программа для ознакомления с модулями RCC, GPIO, TIM, MISC.

**#include** "stm32\_lib/inc/stm32f0xx\_rcc.h"

**#include** "stm32\_lib/inc/stm32f0xx\_gpio.h"

**#include** "stm32\_lib/inc/stm32f0xx\_tim.h"

**#include** "stm32\_lib/inc/stm32f0xx\_misc.h"

**void** **TimConfg**(**void**) // Настройка таймера тактирования

{

RCC\_APB1PeriphClockCmd(RCC\_APB1Periph\_TIM14, *ENABLE*);

// Включаем тактирование TIM14.

TIM\_TimeBaseInitTypeDef Tim;

NVIC\_InitTypeDef NVIC\_I;

Tim.TIM\_ClockDivision = TIM\_CKD\_DIV1;

// Выключаем предварительный делитель частоты таймера.

Tim.TIM\_CounterMode = TIM\_CounterMode\_Up;

// Значение в счетном регистре увеличивается.

Tim.TIM\_Prescaler = (SystemCoreClock / 1000) - 1;

// Делим так чтобы на таймер шел 1 КГц.

Tim.TIM\_Period = 499;

// Таймер будет генерирвать событие 2 раза в секунду.

Tim.TIM\_RepetitionCounter = 0;

TIM\_TimeBaseInit(TIM14, &Tim); // Инициализация таймера.

// Настройка прерывания от таймера.

NVIC\_I.NVIC\_IRQChannel = *TIM14\_IRQn*;

NVIC\_I.NVIC\_IRQChannelPriority = 3;

// Приоритет прерывания (0 - 3. 0 - наивысший, 3 - наименьший).

NVIC\_I.NVIC\_IRQChannelCmd = *ENABLE*;

// Разрешаем прерывание.

NVIC\_Init(&NVIC\_I);

TIM\_ITConfig(TIM14, TIM\_IT\_Update, *ENABLE*);

// Разрешаем прерывание таймера по переполнению.

TIM\_Cmd(TIM14, *ENABLE*); // Запуск таймера.

}

// Функция инвертирует состояние вывода.

**void** **GPIO\_ToggleBits**(GPIO\_TypeDef\* GPIOx, uint16\_t GPIO\_Pin)

{

/\* Check the parameters \*/

assert\_param(IS\_GPIO\_ALL\_PERIPH(GPIOx));

GPIOx->ODR ^= GPIO\_Pin;

}

**void** **TIM14\_IRQHandler**(**void**) // Обработчик прерывания от таймера.

{

GPIO\_ToggleBits(GPIOB, GPIO\_Pin\_15);

// Мигаем светодиодом.

TIM\_ClearITPendingBit(TIM14, TIM\_IT\_Update);

// Сброс флага прерывания по переполнению таймера.

}

**int** **main**(**void**)

{

SystemInit();

//Назначение порта 15 как выхода

RCC\_AHBPeriphClockCmd(RCC\_AHBPeriph\_GPIOB,*ENABLE*);

GPIO\_InitTypeDef ledinit;

GPIO\_StructInit(&ledinit);

ledinit.GPIO\_Pin = GPIO\_Pin\_15;

ledinit.GPIO\_Mode = *GPIO\_Mode\_OUT*;

GPIO\_Init(GPIOB, &ledinit);

GPIO\_SetBits(GPIOB, GPIO\_Pin\_15);

TimConfg(); // Вызов функции настройки таймера

**while**(1)

{

}

}

**Приложение Б**

Программа для ознакомления с модулями ADC и таймером в режиме формирования ШИМ.

**#include** "stm32\_lib/inc/stm32f0xx\_rcc.h"

**#include** "stm32\_lib/inc/stm32f0xx\_gpio.h"

**#include** "stm32\_lib/inc/stm32f0xx\_tim.h"

**#include** "stm32\_lib/inc/stm32f0xx\_misc.h"

**#include** "stm32\_lib/inc/stm32f0xx\_adc.h"

// Определение скважности ШИМа

**volatile** uint16\_t PWM\_val = 10;

// Аппаратный выход PB0

**void** **GPIO\_Ap\_SHIM**(**void**)

{

RCC\_AHBPeriphClockCmd(RCC\_AHBPeriph\_GPIOB,*ENABLE*);

GPIO\_InitTypeDef ledinit\_AF;

GPIO\_StructInit(&ledinit\_AF);

ledinit\_AF.GPIO\_Pin = GPIO\_Pin\_0;

ledinit\_AF.GPIO\_Mode = *GPIO\_Mode\_AF*;

GPIO\_PinAFConfig(GPIOB, GPIO\_PinSource0, GPIO\_AF\_1);

GPIO\_Init(GPIOB, &ledinit\_AF);

}

// Программный выход PB1

**void** **GPIO\_Progr\_SHIM**(**void**)

{

RCC\_AHBPeriphClockCmd(RCC\_AHBPeriph\_GPIOB,*ENABLE*);

GPIO\_InitTypeDef ledinit;

GPIO\_StructInit(&ledinit);

ledinit.GPIO\_Pin = GPIO\_Pin\_1;

ledinit.GPIO\_Mode = *GPIO\_Mode\_OUT*;

GPIO\_Init(GPIOB, &ledinit);

}

// Аналоговый вход PA1

**void** **GPIO\_Out**(**void**)

{

RCC\_AHBPeriphClockCmd(RCC\_AHBPeriph\_GPIOA,*ENABLE*);

GPIO\_InitTypeDef GPIO\_InitStructure;

GPIO\_StructInit(&GPIO\_InitStructure);

GPIO\_InitStructure.GPIO\_Pin = GPIO\_Pin\_1;

GPIO\_InitStructure.GPIO\_Mode = *GPIO\_Mode\_AN*;

GPIO\_InitStructure.GPIO\_PuPd = *GPIO\_PuPd\_NOPULL*;

GPIO\_Init(GPIOA, &GPIO\_InitStructure);

}

// Конфигурация ШИМа аппаратная

**void** **SHIM\_Conf\_AP**(**void**)

{

RCC\_APB1PeriphClockCmd(RCC\_APB1Periph\_TIM3, *ENABLE*);

// Инициализация переменной для настройки прерывания

NVIC\_InitTypeDef NVIC\_I;

// Настройка переполнения таймера

TIM\_TimeBaseInitTypeDef timer\_init;

TIM\_TimeBaseStructInit(&timer\_init);

timer\_init.TIM\_Period = 111 - 1;

timer\_init.TIM\_Prescaler = (SystemCoreClock / 1000) - 1;

TIM\_TimeBaseInit(TIM3, &timer\_init);

// Задание скважности ШИМ

TIM\_OCInitTypeDef tim\_OC\_init;

TIM\_OCStructInit(&tim\_OC\_init);

tim\_OC\_init.TIM\_Pulse = PWM\_val;

tim\_OC\_init.TIM\_OCMode = TIM\_OCMode\_PWM1;

tim\_OC\_init.TIM\_OutputState = TIM\_OutputState\_Enable;

TIM\_OC1Init(TIM3, &tim\_OC\_init);

// Настройка прерывания от таймера.

NVIC\_I.NVIC\_IRQChannel = *TIM3\_IRQn*;

NVIC\_I.NVIC\_IRQChannelPriority = 3; // Приоритет прерывания (0 - 3. 0 - наивысший, 3 - наименьший).

NVIC\_I.NVIC\_IRQChannelCmd = *ENABLE*; // Разрешаем прерывание.

NVIC\_Init(&NVIC\_I);

// Включаем генерацию прерывания по переполнению

TIM\_ITConfig(TIM3, TIM\_IT\_Update, *ENABLE*);

// Включаем таймер

TIM\_Cmd(TIM3, *ENABLE*);

}

// Конфигурация ШИМа программная

**void** **SHIM\_Conf\_PROG**(**void**)

{

RCC\_APB1PeriphClockCmd(RCC\_APB1Periph\_TIM3, *ENABLE*);

// Инициализация переменной для настройки прерывания

NVIC\_InitTypeDef NVIC\_I;

// Настройка переполнения таймера

TIM\_TimeBaseInitTypeDef timer\_init;

TIM\_TimeBaseStructInit(&timer\_init);

timer\_init.TIM\_Period = 111 - 1;

timer\_init.TIM\_Prescaler = (SystemCoreClock / 1000) - 1;

TIM\_TimeBaseInit(TIM3, &timer\_init);

// Задание скважности ШИМ

TIM\_OCInitTypeDef tim\_OC\_init;

TIM\_OCStructInit(&tim\_OC\_init);

tim\_OC\_init.TIM\_Pulse = PWM\_val;

tim\_OC\_init.TIM\_OCMode = TIM\_OCMode\_PWM1;

tim\_OC\_init.TIM\_OutputState = TIM\_OutputState\_Enable;

TIM\_OC1Init(TIM3, &tim\_OC\_init);

// Настройка прерывания от таймера.

NVIC\_I.NVIC\_IRQChannel = *TIM3\_IRQn*;

NVIC\_I.NVIC\_IRQChannelPriority = 3; // Приоритет прерывания (0 - 3. 0 - наивысший, 3 - наименьший).

NVIC\_I.NVIC\_IRQChannelCmd = *ENABLE*; // Разрешаем прерывание.

NVIC\_Init(&NVIC\_I);

// Включаем генерацию прерывания по переполнению

TIM\_ITConfig(TIM3, TIM\_IT\_Update, *ENABLE*);

// Генератор прерывания по событию сравнения

TIM\_ITConfig(TIM3, TIM\_IT\_CC4, *ENABLE*);

// Включаем таймер

TIM\_Cmd(TIM3, *ENABLE*);

}

// Переопределение обработчика прерывания ШИМ

// для программной реализации

**void** **TIM3\_IRQHandler**()

{

**if** (TIM\_GetITStatus(TIM3, TIM\_IT\_Update) != *RESET*) {

TIM\_ClearITPendingBit(TIM3, TIM\_IT\_Update);

TIM\_SetCompare1(TIM3,PWM\_val);

**if** (PWM\_val > 0) GPIO\_SetBits(GPIOB, GPIO\_Pin\_1);

}

**if** (TIM\_GetITStatus(TIM3, TIM\_IT\_CC4) != *RESET*) {

TIM\_ClearITPendingBit(TIM3, TIM\_IT\_CC4);

**if** (PWM\_val < 109) GPIO\_ResetBits(GPIOB, GPIO\_Pin\_1);

}

}

// Переопределение обработчика прерывания ШИМ

// для аппаратной реализации

//void TIM3\_IRQHandler()

//{

// TIM\_ClearITPendingBit(TIM3, TIM\_IT\_Update);

// TIM\_SetCompare1(TIM3,PWM\_val);

//}

// Настройка модуля АЦП

**void** **ADC\_Conf**(**void**)

{

RCC\_APB2PeriphClockCmd(RCC\_APB2Periph\_ADC1, *ENABLE*);

// Создание структуры АЦП

ADC\_InitTypeDef ADC\_InitStructure;

// Инициализация структуры АЦП

ADC\_StructInit(&ADC\_InitStructure);

// Загрузка настроек по умолчанию

ADC\_Init(ADC1, &ADC\_InitStructure);

// Включаем прерывание по окончанию преобразования

ADC\_ITConfig(ADC1, ADC\_IT\_EOC, *ENABLE*);

// Включаем модуль

ADC\_Cmd(ADC1, *ENABLE*);

// Разрешаем прерывание

NVIC\_EnableIRQ(ADC1\_IRQn);

// Загрузка конфигурации

ADC\_ChannelConfig(ADC1, ADC\_Channel\_4, ADC\_SampleTime\_7\_5Cycles);

//Запуск преобразования

ADC\_StartOfConversion(ADC1);

}

// Переопределение обработчика прерывания АЦП

**void** **ADC\_IRQHandler**()

{

**if** (ADC\_GetITStatus(ADC1, ADC\_IT\_EOC) != *RESET*)

{

ADC\_ClearITPendingBit(ADC1, ADC\_IT\_EOC);

PWM\_val = (110 \* (ADC\_GetConversionValue(ADC1)))/ 4095;

ADC\_ChannelConfig(ADC1, ADC\_Channel\_4, ADC\_SampleTime\_239\_5Cycles);

ADC\_StartOfConversion(ADC1);

}

}

**int** **main**(**void**)

{

SystemInit();

GPIO\_Progr\_SHIM();

GPIO\_Out();

SHIM\_Conf\_PROG();

ADC\_Conf();

**while**(1)

{

}

}