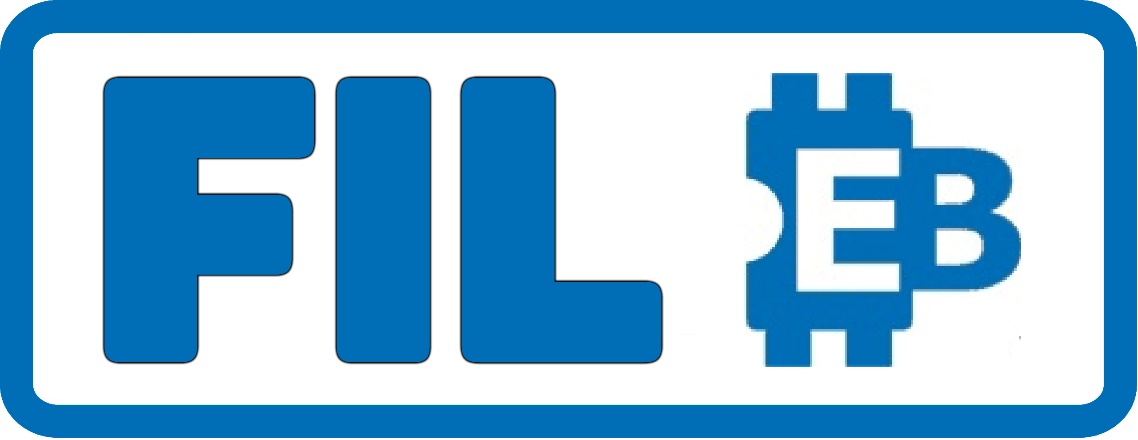
**STM32 Fast Initialization Library (FIL).**

Библиотека быстрой инициализации для микроконтроллеров STM32 на базе программной IDE EmBitz.

**Авторы: Назаров А.А., Гаранин Е.О.**

****

**Содержание**

|  |  |  |
| --- | --- | --- |
|  | [Введение](#Введение) | 3 |
| **1.** | [Структура библиотеки FIL](#Раздел1) | 5 |
|  | **1.1** [Архитектурное строение](#Раздел11) | 5 |
|  | **1.2** [Рекомендации по установке и список поддерживаемых устройств](#Раздел12) | 6 |
|  | **1.2.1** Версия для начинающих | 6 |
|  | **1.2.2** Версия для опытных пользователей | 6 |
|  | **1.3** [История разработки](#Раздел13) | 6 |
| **2.** | Конфигурация библиотеки | 6 |
|  | **2.1** Файл-карта портов | 6 |
|  | **2.2** Минимальная версия конфигурации | 6 |
|  | **2.3** Создание своей конфигурации | 6 |
| **3.** | Список API функций и макросов библиотеки FIL | 6 |
|  | **3.1** Регламентация названий | 6 |
|  | **3.2** Опциональные правила использования | 6 |
|  | **3.3** Перечень макросов конфигураций | 6 |
|  | **3.4** Перечень API функций | 6 |
|  | **3.4.1** Функции RCC (Reset Clock Configuration) | 6 |
|  | **3.4.2** Функции GPIO (General Purpose Input/Output) | 6 |
|  | **3.4.3** Функции TIM (Timer Interface Mode) | 6 |
|  | **3.4.4** Функции ADC (Analog-Digital Converter) | 6 |
|  | **3.4.5** Функции USART (Universal Asynchronous Receiver-Transmitter) | 6 |
|  | **3.4.6** Функции I2C (Inter Integrated Circuit) | 6 |
|  | **3.4.7** Функции DMA (Data Memory Access) | 6 |
|  | **3.4.8** Функции Регулятора | 6 |
|  | **3.4.9** Функции EXTI (External Interrupts) | 6 |
|  | **3.4.10** Функции SPI (Serial Peripheral Interface) | 6 |
|  | **3.4.11** Функции DAC (Digital-Analog Converter) | 6 |
|  | **3.4.12** Функции CAN (Controller Area Network) | 6 |
| **4.** | Использование операционной системы реального времени для микроконтроллеров | 6 |
| **5.** | Помощь в поддержке продукта | 6 |
|  | Листинг А | 6 |
|  | Листинг Б | 6 |
|  | Листинг В | 6 |
|  | Список литературы | 6 |

**Введение**

Популярность микроконтроллеров Atmega (Arduino series) в любительских проектах, а также некоторых профессиональных, обосновывается простотой использования и уменьшении времени разработки полномасштабного и функционального алгоритма управления. Однако, для дальнейшего углубленного изучения принципа функционирования микроконтроллерных систем среда не предоставляет набор инструментария и функций. Пользователь погружаясь в дальнейшее изучение сталкивается с трудностями восприятия нового стиля написания программ через регистровые переменные, что увеличивает время обучения и проектирования алгоритмов.

Одним из популярных аналогов являются микроконтроллеры компании ST Microelectronics – STM8, STM32. Данные линейки контроллеров имеют обширную периферию, а доступ к изменению регистровых параметров позволяет разработчику настроить под себя систему управления, обмена данными и иными функциями. Общение через регистры низкоуровневым стилем программирования для начинающего компенсируется познанием базовых принципов и нюансов при инициализации и отладке программного кода. Данный подход не эффективен применительно к начинающим программировать данные микроконтроллеры. Поэтому сотрудниками Ресурсного центра робототехники (РЦР) Донского государственного технического университета (ДГТУ) была выдвинута идея создания полномасштабной библиотеки обучения и плавного перехода с высокого на низкий уровень проектирования алгоритмов.

Альфа версия библиотеки Fast Initialization Library (FIL) вышла в октябре 2021 года, имея в своем функционале инициализацию малого участка периферии. Поддержка продукта осуществляется и по сей день. За развитием проекта можно следить через платформу Github[1].

В данном документе приведены основные сведения для взаимодействия с библиотекой быстрой инициализации микроконтроллеров STM32. В основных разделах содержится информация по управляющим командам, а также примеры их использования. Приведена структура и принцип работы обучающей платформы, на основе которой демонстрируется функционал библиотеки. Документ дополняется по мере увеличения функционала библиотеки.

Все файлы и программный код библиотеки не претендуют на авторские права, любой желающий может использовать её в своих проектах. Приветствуется распространение и отзыв обратной связи о функциональности.

Для этого просьба обращаться по адресу - г.Ростов-на-Дону, пл.Гагарина 1, ДГТУ, 2 корпус, 2-205.

Также возможен вариант отправки письма на почту –

[sasha.sanya.nazarov@yandex.ru](mailto:sasha.sanya.nazarov@yandex.ru)

garanin1392@gmail.com

1. **Структура библиотеки FIL**

**1.1** **Архитектурное строение**

Библиотека имеет структуру, изображенную на рисунке 1.1. В ней есть участок пользовательского кода, написанный или планируемый программистом, файл-карта портов текущего микроконтроллера, конфигурационный файл с определениями режимов работы каждого периферийного участка, интегрированный в IDE EmBitz файл с определениями библиотеки CMSIS, позволяющий обращаться к адресам ключевых участков через словесные интерпретации и файлы библиотеки FIL, представленные двумя фрагментами: линкер файлом и определениями основных функций (FIL API).

Библиотека использует определения конфигурационного файла и карты портов, чтобы предоставить к использованию необходимые функции, по умолчанию доступ к библиотеке через пользовательское пространство недоступен. После инициализации периферии линкер связывает необходимые файлы, в которых хранятся API функции. Определения регистровой библиотеки CMSIS необходимы для работы библиотеки, обычно они встроены в большинство IDE и имеют большой список поддерживаемых контроллеров ST.

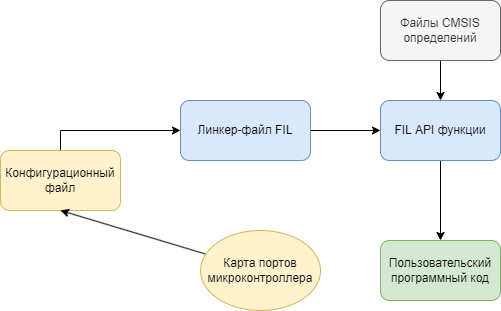
****

Рисунок 1.1 – архитектурное строение и взаимодействие с библиотекой FIL.

**1.2** **Рекомендации по установке и список поддерживаемых устройств**

**1.2.1 Версия для начинающих**

Установка начинается с создание нового проекта, согласно рисунку 1.1 и выбора архитектуры и названия проекта (рисунок 1.2).

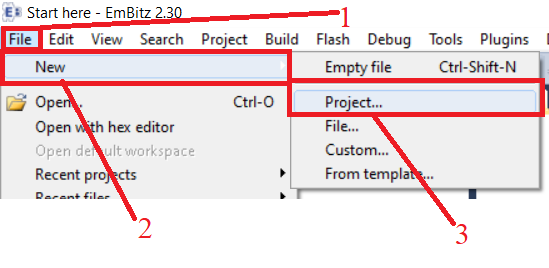


Рисунок 1.1 – Создание проекта.

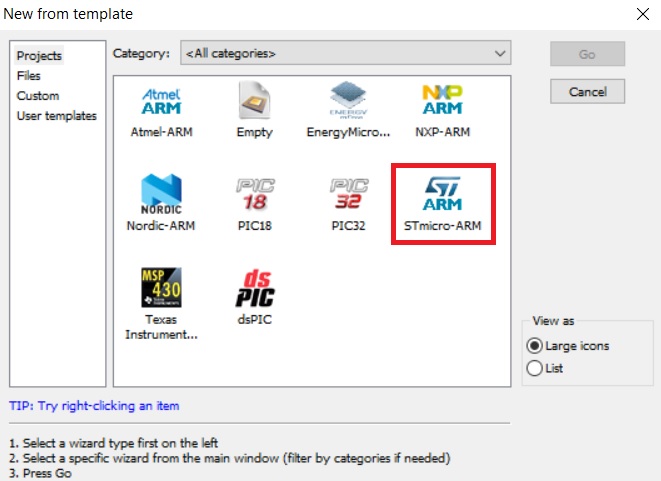
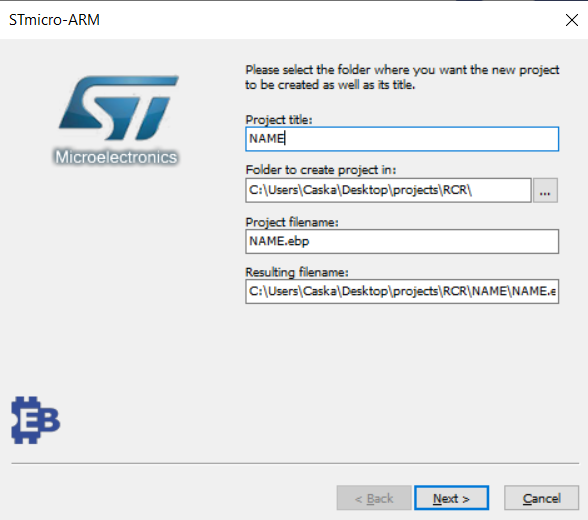
 

Рисунок 1.2 – Выбор линейки контроллера и названия проекта.

Далее потребуется выбрать микроконтроллер, с которым предстоит работать ([смотри список поддерживаемых устройств](#Поддержка)). Инструкция приведена на рисунках 1.3. На остальных окнах, не приведенных в текущей документации просто нажимать далее.

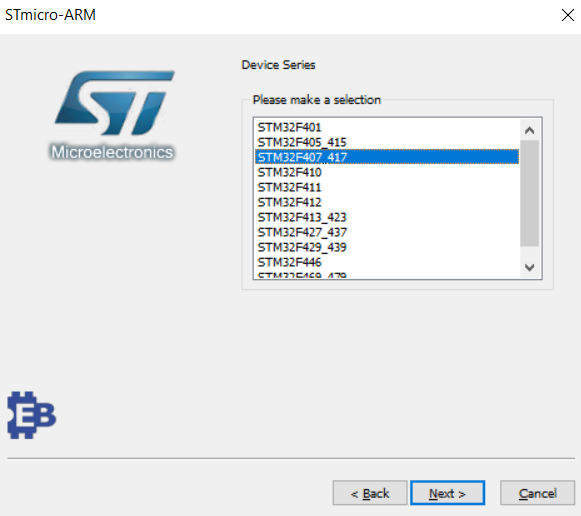
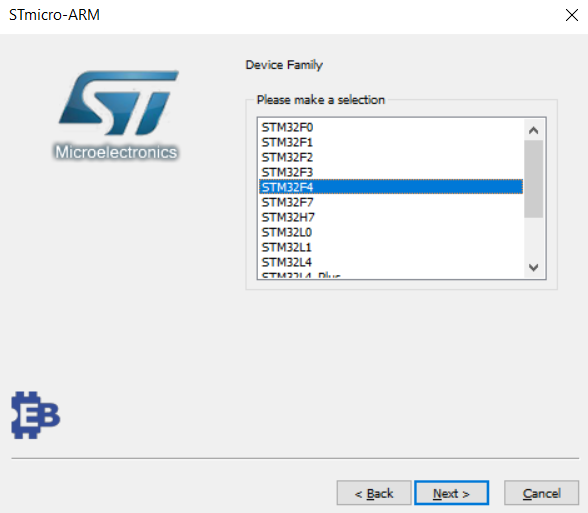


Рисунок 1.3 – выбор линейки микроконтроллера.

Далее, отключаем встроенные библиотеки SPL и HAL (рисунок 1.4). Потребуется только CMSIS определения, которые в любом случае будут добавлены в новых проект.

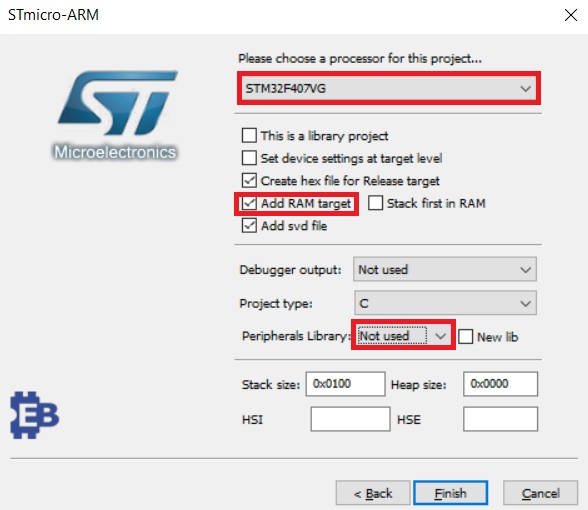


Рисунок 1.4 – Выбор модели, отключение встроенных библиотек.

Скопируйте файлы библиотеки FIL и любые другие файлы, необходимые для работы (рисунок 1.5). Итоговый вид проекта из проводника должен быть как на рисунке 1.6.

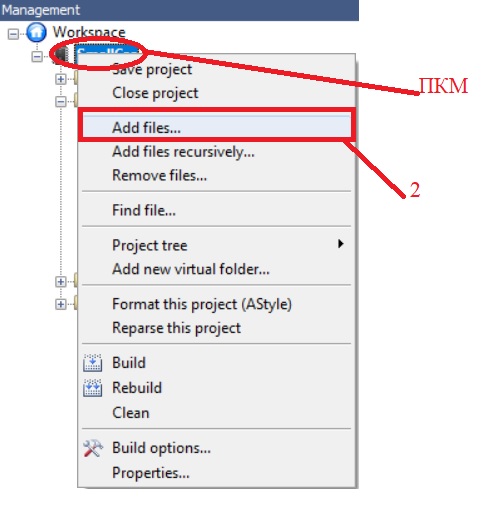


Рисунок 1.5 – Добавление новых файлов.

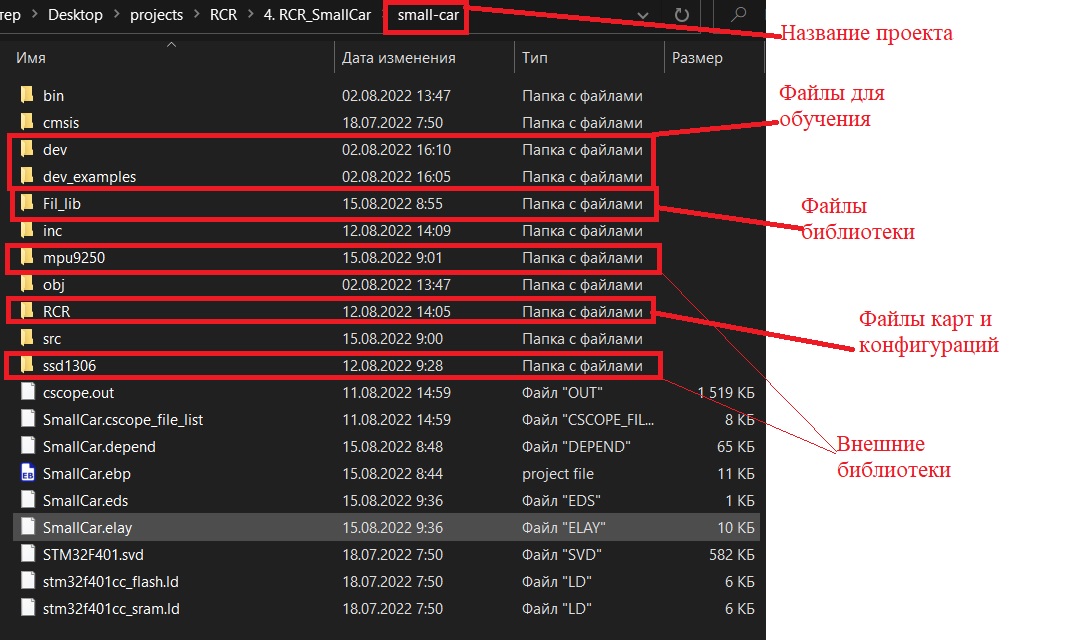


Рисунок 1.6 – добавление в пространство проекта файлов (проводник Windows).

Необходимо перейти в файл main.h и добавить подключение конфигурационного файла через *#include* как на рисунке 1.7.

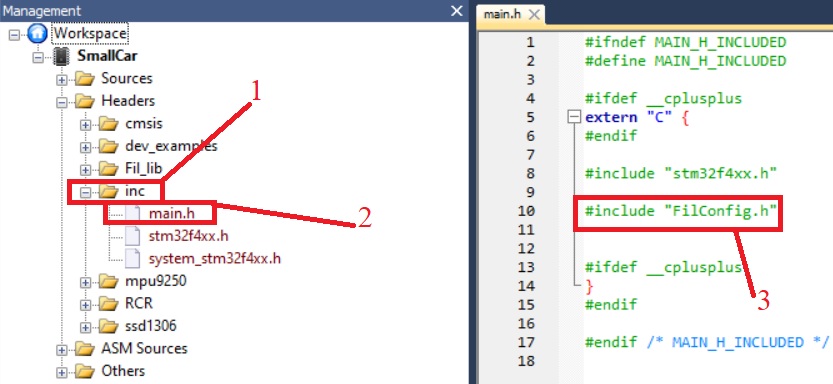


Рисунок 1.7 – Добавление библиотеки в проект.

**1.2.2 Версия для опытных пользователей**

На данный момент версия библиотеки имеет несколько особенностей, которые необходимо учитывать перед непосредственной работой: Карта портов микроконтроллера, с примером которой можно ознакомиться в Листинге С и на рисунке 1.2, содержит в себе базовые определения из библиотеки FIL, таким образом получается замкнутый цикл, где конфигурация для библиотеки использует малую долю себя, чтобы расширить пространство предоставляемых функций. Данная опция облегчает создание проекта пользователем и ускорение времени разработки программного кода, однако потребовалось оставить участок GPIO открытым для доступа.

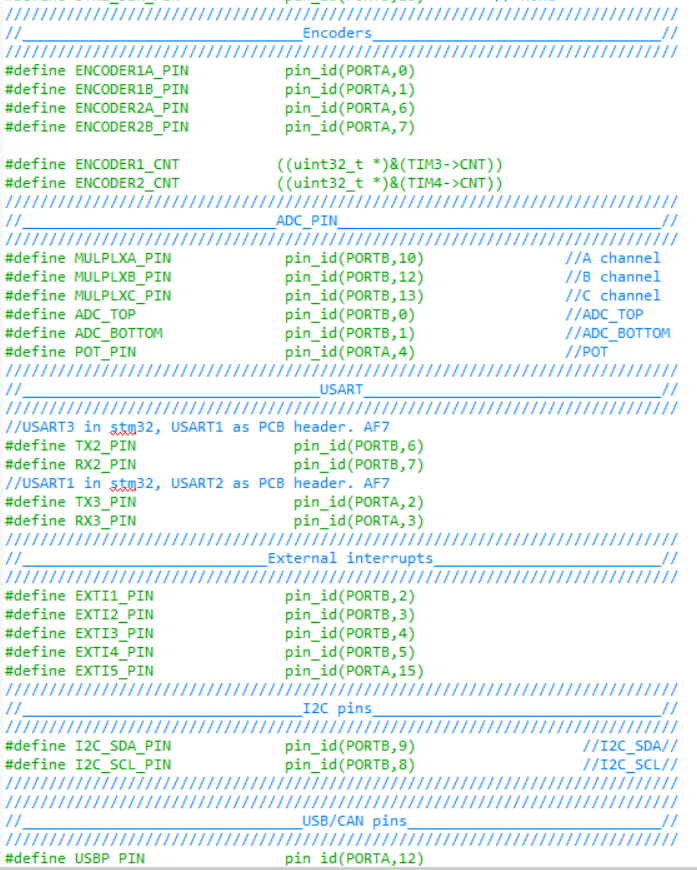


Рисунок 1.2 – пример написания карты портов микроконтроллера.

Несмотря на наличие большого количества предоставляемых функций, библиотека может содержать недоработки и ошибки, связанные с недостаточной проработанностью и защищенностью участков кода. Если таковые нашлись в явном виде, и они мешают разработке, просьба сообщить по контактам, приведенным во введении.

Список поддерживаемых контроллеров:

- ARM Cortex-M4 (STM32F407VG, STM32F411VE, STM32F401CC);

*Примечание 1.1: Список будет дополняться по мере продвижения разработки. Проект библиотеки протестирован в среде EmBitz v2.30. В будущих версиях планируется протестировать библиотеку в среде Keil uVision. Альтернативный путь – попытка запуска и интеграции с более старыми версиями EmBitz (1.11 – 2.10).*

Для установки потребуется проделать следующие шаги:

1) Скопировать файлы библиотеки через GitHub или любым другим способом;

2) Создать проект в IDE EmBitz, настроив его под текущую линейку используемого контроллера (без подключения встроенных библиотек SPL, HAL);

3) Подключить линкер файл библиотеки (рисунок 1.3). Подключение сделать в файле main.h;



4) Использовать готовые конфигурации или создать свою. Конфигурация представляет собой файл с настройками и картой портов. Для простоты хранения и подключения они хранятся в файле BoardSelection.h, структура которого изображена на рисунке 1.3.



Рисунок 1.3 Файл с проектами.

Для создания своей конфигурации необходимо установить аргумент \_\_configUSEBoards равным 4 и добавить в пустое пространство (*Development Board №4*) свои файлы. О написании своих конфигурационных файлов будет рассказано в следующих разделах.

**1.3** **История разработки**

13.10.2021 – начата работа над библиотекой.

23.10.2021 – разработаны функции инициализации для таймеров. Конфигурация портов ввода-вывода была использована заблаговременно, использована версия, разработанная в Ресурсном Центре Робототехники предположительно в 2013-2015 годах.

25.10.2021 – появление версии 1.0. Включала в себя инициализацию через ручное включение секторов кода, было разделение на уровни Default, Advanced и Developer.

31.01.2022 – добавление функций для работы с интерфейсом USART/UART. Некоторые изменения в таймерах и портах ввода-вывода.

02.02.2022 – добавлен файл-линкер FilConfig.h и добавлены исправления, не позволяющие конкретным API быть использованным без включения через линкер.

04.03.2022 – добавлены файлы настройки АЦП и контроллера прямого доступа к памяти(DMA). Исправление в файлах TIM.h, FilConfig.h.

13.05.2022 – изменена конфигурация линкера: теперь не нужно вручную изменять его настройки, для этого потребуется конфигурационный файл.

25.07.2022 – приезд после полевого тестирования библиотеки, добавлена поддержка и возможность подключить операционную систему микроконтроллеров FreeRTOS.

28.07.2022 – исправление работы функций, добавлена поддержка микроконтроллеров STM32F401CC и STM32F411VE.

11.08.2022 – добавление инициализации интерфейса I2C. Разработана функция сканирования шины и поиска устройств.

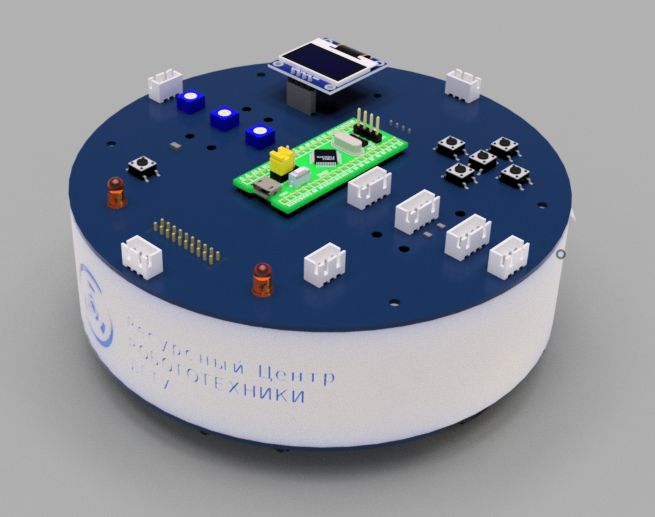
12.08.2022 – обновление документации по проекту.

**Начальный уровень разработчика \_Default**

Раздел предназначен для ознакомления только начавшим обучение программированию микроконтроллеров STM32. В сопровождении с этим документом предложены несколько типовых задач, решив которые, станет возможным переход на продвинутый уровень доступа.

* 1. **Структура обучающей платформы**

С целью разработки и последующей отладки программного кода РЦР ДГТУ предоставляет обучающую платформу, представляющую собой двухколесного робота. Обучаясь на данной платформе, разработчик сможет научиться программировать микроконтроллеры STM32 и продемонстрировать закрепленные знания выполнив несколько типовых заданий.



Данная платформа поддерживает несколько типовых функций: вывод информации на экран, программирование кнопок управления, светодиодов, управление исполнительными электроприводами.

Для успешной работы с роботом необходимо знать обозначения портов, чтобы можно было к ним обращаться. Все они указаны в файле Headers\lib\RCR DSTU\RCR\_PinsBoard.h

* 1. **Список предоставляемых команд начального уровня**

На таблице 2.1 приведен общий список команд для разработчика, которые определены интуитивно понятными словесными конструкциями. Использование данного уровня не потребует углубленных знаний и используется для начального ввода в программирование микроконтроллеров.

Таблица 2.1 – список общих управляющих команд начального уровня.

|  |  |  |
| --- | --- | --- |
| **Название команды** | **Что делает** | **Какие параметры потребует** |
| ClocksInit | Инициализация периферии для работы с платой обучения | - |
| LedInit | Определение нужного количества светодиодов в обычном режиме | - |
| Set\_led(led)**\*** | Включает выбранный светодиод | led – номер светодиода, который необходимо включить |
| PWM\_init(a) | Определение нужного количества светодиодов в специальном режиме работы (ШИМ) | a – количество необходимых светодиодов для режима ШИМ |
| PinVal(button) | Проверка состояний кнопок | Номер кнопки, которую необходимо проверить |
| SetPWM(MOTOR,DUTY)**\*** | Установка заполнения на выбранный светодиод. Значение заполнения задается от 0 до 100% | led – номер необходимого светодиода;  duty – требуемое заполнение на выбранном светодиоде. |
| MotorsInit | Инициализация двигателей |  |
| ButtonInit(NUMBER)\* | Определение нужного количества кнопок | Номер необходимой кнопки либо ALL |
| EncodersInit | Инициализация энкодеров |  |
| MoveTo(STATE,SPEED)\* | Запуск двигателей | HOLD - остановка  FORWARD – движение вперед  BACKWARD – движение назад  TURN\_LEFT – разворот налево  TURN\_RIGHT – разворот направо |
| GetEncoderData(NUMBER)\* | Вытащить значения 'нкодера |  |
| CleatEncoderData()\* | Отчистить данные энкодера |  |
| SetPin | Повышение логического уровня элемента |  |
| ResetPin | Понижение логического уровня элемента |  |
| ADCInit | Инициализация аналого-цифрового преобразователя (АЦП) для работы с переменным резистором и датчиком линии | - |
| **\* ̶** Имеется защита от неверного указания параметров | | |

Приведенного списка команд достаточно для выполнения практических задач начального уровня.

* 1. **Примеры использования команд начального уровня**

Для того, чтобы приступить к разработке сначала рассмотрим основной функционал программы EmBitz:

Когда у нас уже написан код необходимо сохранить

При разработке программного кода начинающему следует использовать правильный порядок написания программного кода. Для этого рассмотрим примитивные примеры решения практических задач:

**Пример 1**

*Используя инструменты начального уровня, напишите алгоритм включения светодиода.*

Для решения этой задачи был написан код приведенный ниже.

#include “main.h”

int main(void)

{

Clocks\_init;

Led\_init();

SetPin(LED\_PIN);

While(1) {}

}

Как видно из примера, первой и главной командой является строка Clocks\_init;, которая разблокирует дальнейшую настройку, если строку не включать то последующие действия не окажут никакого эффекта на микроконтроллер. Далее следует функция определения светодиодов Led\_init();. Функция имеет защиту от переопределения и неопределенности аргументов (см. примечание 2.1). Завершающей выполнение задачи функция SetPin(); включает светодиод. Команда ResetPin(); является противоположностью команды SetPin();, для решения текущей задачи не подходит, потому что выключает светодиод, когда нужно его включить.

*Примечание 2.1 – Функции Led\_init()* *и PWM\_init() защищены от переопределения. Это значит, что комбинированное использование одной и другой в программе приведет к тому, что первая выполняется и действует ровно до того момента, пока не определится вторая.*

Добавим возможность затухания светодиода через некоторое, установленное нами, время. Для этого в файл INTERRUPTS.c в конец кода добавим

void TIM5\_IRQHandler(void) {

globalTime++;

if(globalTime == 50) {

ResetPin(LED\_PIN);

}

ResetTimSR(Tim5);

Регулируя значение globalTime меняем длительность горения светодиода (значение 50 соответствует 10 секундам)

**Пример 2**

*Используя инструменты начального уровня, напишите алгоритм включения двигателей..*

int main(void)

{

ClocksInit;

MotorsInit;

MoveTo(FORWARD, 0.5);

while(1)

{

}

}

В данном примере мы устанавливаем два параметра: FORWARD –направление движения (в таблице 2.1 указаны остальные варианты), 0.5 – скорость движения (устанавливается в пределах от 0 до 1).

Добавьте код и запустите программу. Понаблюдайте за тем, как движется робот. Одинаковые ли скорости у обоих колес?

Любая пара двигателей, даже одной и той же модели и из одной партии, будет работать по-разному. Поэтому даже если на первый взгляд кажется, что двигатели работают с одинаковой скоростью, на больших оборотах или более продолжительной дистанции эта разница будет значительной. Так же существует целый ряд других помех, не позволяющих нашей платформе двигаться строго прямо. Это и неровность поверхности, помехи и недостаточно одинаковое напряжение на моторах.

В этой ситуации нам на помощь может прийти регулятор – это система для управления движением платформы. Она сравнивает сигнал на входе и на выходе системы. Полученная разница называется *ошибкой* и уже относительно неё система изменяет входные параметры для получения необходимых результатов.

Рассмотрим на конкретном примере: платформа движется по линии, за которой следят специальные датчики, установленные на платформе. Мы подаем одинаковый сигнал на оба мотора и спустя некоторое время траектория отклоняется левее заданного курса. Это засекают датчики, передают в устройство управления, и оно изменяет подаваемый сигнал так, чтобы левое колесо двигалось быстрее, а правое медленнее. Так траектория выравнивается, и платформа движется так, как нам необходимо.

Существует несколько принципов управления, но чаще всего используется ПИД-управление (Пропорционально-интегрально-дифференциальное управление). Оно собирает воедино все 3 других принципа, для получения наилучшего результата.

Каждый из элементов регулятора выполняет свою задачу и оказывает свое специфическое воздействие на функционирование системы: пропорциональный закон отвечает за настоящее (реагирует на текущую ошибку), дифференциальный – за будущее (реагирует на тенденцию изменения ошибки), а интегральный – за прошлое (накапливая предыдущие ошибки и сглаживая высокочастотные шумы).

Рассмотрим принципы управления немного детальнее.

Пропорциональное управление.

То, какую ошибку нам выдаст датчик, прямо пропорционально действует на входное воздействие. Для реализации просто возьмем усилитель, который будет усиливать сигнал ошибки в kp раз. Главное преимущество данного принципа в том, что он прост и быстро реагирует. Основным минусом является ограниченная точность и перерегулирование.

Пропорционально-дифференциальное управление.

В некоторых ситуациях, например, на извилистой дороге, нам важна реакция не только на ошибку, но и на скорость ее изменения. Для этого нам нужно как бы предсказывать следующие изменения ошибки, чтобы успевать реагировать. То есть если мы знаем, что ошибка увеличивается (это и есть скорость изменения ошибки), то нам необходимо увеличивать управляющее воздействие. В данной ситуации нам может помочь *дифференциатор.*

Задачей дифференциатора является умножение разности выходного сигнала в данный момент и предыдущего значения на постоянный коэффициент. Главным преимуществом данного принципа является его быстродействие. Но так же существуют и недостатки в виде чувствительности к шумам и небольшой точности.

Интегральное управление.

Чтобы решить проблему с шумами воспользуемся *интегрирующим звеном (интегратором)*

Интегратор складывает(интегрирует) сигнал ошибки. Управляющий сигнал в каждый момент времени пропорционален интегралу ошибки. Благодаря этому регулятор реагирует лишь на длительные отклонения, а шумы сглаживает. Ошибки есть всегда, и простое их накопление может сделать систему нестабильной, поэтому данный принцип работает только в паре с пропорциональным управлением. Преимуществом пропорционально-интегрального управления является то, что он стабильнее в установившемся режиме, чем П-управление и при определенном соотношении коэффициентов в переходном. Недостатком является то, что в переходных режимах меньшее быстродействие большая колебательность.

Соединив все эти принципы получим ПИД-управление. Которое соединяет в себе все преимущества и недостатки законов.

Функция ПИД-регулятора выглядит так

// Параметры пропорционального звена

float kp = 10; // Коэффициент пропорционального звена

// Параметры интегратора

float ki = 0.001; // Коэффициент интегрального звена

#define iMin -0.2 // Минимальное значение интегратора

#define iMax 0.2 // Максимальное значение интегратора

float iSum = 0; // Сумма ошибок (значение, накопленное в интеграторе)

// Параметры дифференциатора

float kd = 1; // Коэффициент дифференциального звена

float old\_y = 0; // Предыдущее значение сигнала

float PIDctl(float error, float y)

{ float up, ui, ud;

// Пропорциональная компонента

up = kp\*error;

// Интегральная компонента

iSum = iSum+error; // Накапливаем (суммируем)

if(iSum<iMin) iSum = iMin; // Проверяем граничные значение

if(iSum>iMax) iSum = iMax;

ui = ki\*iSum;

// Дифференциальная компонента

ud = kd\*(y-old\_y);

old\_y = y;

return up+ui+ud;

}

**Продвинутый уровень разработчика \_Advanced**

В данном разделе приложено описание продвинутого уровня доступа, открывающий возможность частичного использования библиотеки FIL. Основной отличительной чертой данной уровня от начального является возможность инициализации большей части интерфейсов с помощью упрощенно задаваемых команд. Однако использование продвинутого уровня потребует от разработчика задавать больше входных параметров, производить базовые расчеты для инициализации необходимой периферии.