МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ
РОССИЙСКОЙ ФЕДЕРАЦИИ
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ
ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«САМАРСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ
УНИВЕРСИТЕТ ИМЕНИ АКАДЕМИКА С.П. КОРОЛЕВА»
(Самарский университет)

Институт информатики и кибернетики

Кафедра лазерных и биотехнических систем

Пояснительная записка к курсовому проекту

”Носимый монитор ЭКГ”

Выполнил студент группы 6364-120304D: _____ Репик В.И.

Руководитель проекта: _____ Корнилин Д.В.

Работа защищена с оценкой: _____

Самара 2023

ЗАДАНИЕ

Разработать монитор активности и отслеживания падений со следующими параметрами:

– Амплитуда сигнала от 0.5 мВ до 4 мВ;

– Диапазон частот 0.05 Гц до 40 Гц;

– Погрешность регистрации амплитуды и частоты 1%;

– Передача данных по интерфейсу Bluetooth;

– Предусмотреть возможность сохранения данных на встроенном носителе в течение суток;

– Питание батарейное;

РЕФЕРАТ

Пояснительная записка: 38 страниц, 20 рисунков, 9 источников, 1 приложение.

НОСИМЫЙ МОНИТОР ЭКГ, МИКРОКОНТРОЛЛЕР, BLUETOOTH, STM32WB, АЛГОРИТМ, ADS1293

В курсовом проекте разработаны структурная и принципиальная схемы монитора ЭКГ, с использованием интегральной AFE микросхемы ADS1293. Был осуществлен выбор микроконтроллера со встроенным блоком Bluetooth. Разработан алгоритм анализа данных и программа на языке Си, реализующая его.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Автоматизация различных процессов на базе интеллектуальных систем невозможна в современном мире без использования устройств такого типа, как микроконтроллер. Многофункциональные, компактные микроконтроллеры применяются во многих современных приборах, бытовом оборудовании, прочих инженерно-технических объектах, а также в медицинской диагностике.

Согласно статистике сердечно-сосудистые заболеваний являются причиной смерти 17,9 млн человек в год. Именно поэтому мониторинг и диагностика состояния сердечно-сосудистой системы человека является такой важной задачей. Самым простым методом диагностики является ЭКГ.

В данном курсовом проекте рассматривается проектирование носимого монитора ЭКГ, автономной системы, позволяющий вести непрерывный мониторинг показателей сердечно-сосудистой системы человека. В процессе проектирования были выбраны микросхема ADS1293 и микроконтроллер STM32WB55RCV6 со встроенным модулем Bluetooth, а также была написана программа управления на языке Си.

1 РАЗРАБОТКА СТРУКТУРНОЙ СХЕМЫ УСТРОЙСТВА

Структурная схема устройства представлена на рисунке 1.

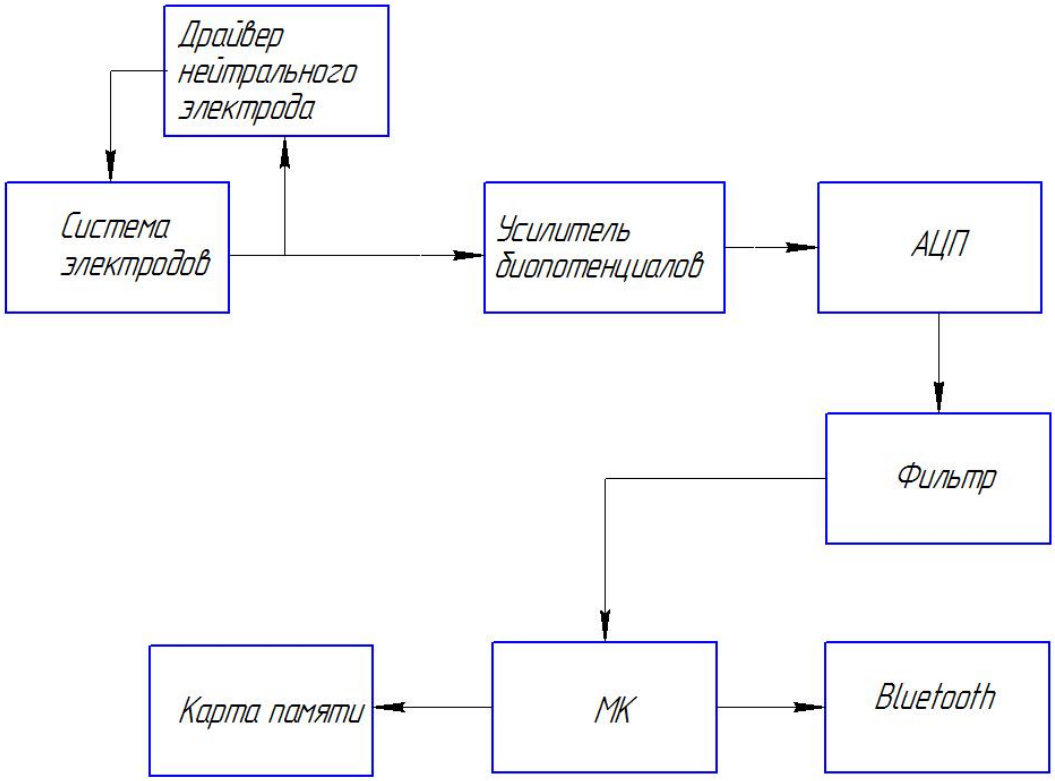

Рисунок 1 – Структурная схема устройства

Принцип работы устройства основан на регистрации разности биопотенциалов, возникающей под действием электрической активности сердца. Система электродов позволяет эту разность измерить. Сигнал будет содержать помимо полезной составляющей синфазную помеху. Для ее устранения используется драйвер нейтрального электрода. Сигнал с электродов с 1 по 3 усредняется, инвертируется в цепи обратной связи и подается на тело человека через электрод 4, тем самым подавляя синфазную составляющую. Так как разность биопотенциалов очень мала по своему значению, то для дальнейшего анализа ее необходимо усилить. Для этого следующим шагом служит усилитель биопотенциалов.

Усиленный сигнал преобразуется в цифровой посредством АЦП, про-

ходит через фильтр и поступает на микроконтроллер для обработки. Так же с микроконтроллера данные передаются по Bluetooth, передача данных осуществляется по таймеру. Предусмотрена запись данных на карту памяти.

Система электродов, драйвер нейтрального электрода, усилитель, АЦП и цифровой фильтр интегрированы в AFE микросхему ADS1293.

Данные поступают в микроконтроллер, где они проходят первичную обработку,

Так же, данные передаются по модулю Bluetooth, интегрированному в микроконтроллер. Передача данных запускается по таймеру. На устройстве есть LED-индикатор, который сигнализирует о передаче пакета данных.

Все элементы схемы питаются от литий-полимерного аккумулятора, имеющего номинальное напряжение 3.7 В, и DC-DC преобразователя, встроенного в микроконтроллер, который стабилизирует напряжение до уровня 3.3 В, необходимого всем элементам устройства.

2 РАЗРАБОТКА ПРИНЦИПИАЛЬНОЙ СХЕМЫ УСТРОЙСТВА

2.1 Разработка(аналоговой части

Для получения кардиосигнала возможны несколько подходов - проектирование собственных схемотехнических решений на основе дискретных компонентов, или использование Analog Front End (AFE) микросхем, например, ADS1293 от Texas Instruments.

Микросхема ADS1293 предназначена для измерения биопотенциалов, в таких медицинских приборах, как портативные электрокардиографы с батарейным питанием, холтеровские мониторы и аппаратура беспроводного мониторинга пациентов. [1]

ADS1293 способна поддерживать от одного до пяти каналов, что позволяет существенно сократить габариты, энергопотребление и полную стоимость масштабируемых измерительных медицинских систем. Каждый канал ADS1293 может быть независимо запрограммирован на работу со специальными (отличными от других) частотой выборки и полосой пропускания.

На основании анализа функциональных возможностей и технических характеристик AFE ADS1293, можно сделать вывод о том, что кроме очевидных преимуществ по габаритам в сравнении с аналоговой частью холтеровских мониторов на дискретных операционных усилителях (ОУ) и аналого-цифровых преобразователях (АЦП), гибридная интегральная схема (ИС) обладает достаточно низким энергопотреблением даже в активном режиме, высоким соотношением сигнал/шум и достаточным динамическим диапазоном для решения задач холтеровского мониторирования. Как следствие высокой интеграции аналоговой части и АЦП, а так же цифровой подсистемы первичной обработки квантованного сигнала использование AFE позволило существенно упростить схемные решения монитора ЭКГ, уменьшить габариты и продолжительность автономной работы от батареи той же емкости.

Структурная схема акселерометра из даташита ADS1293 [2] приведена на рисунке 2.

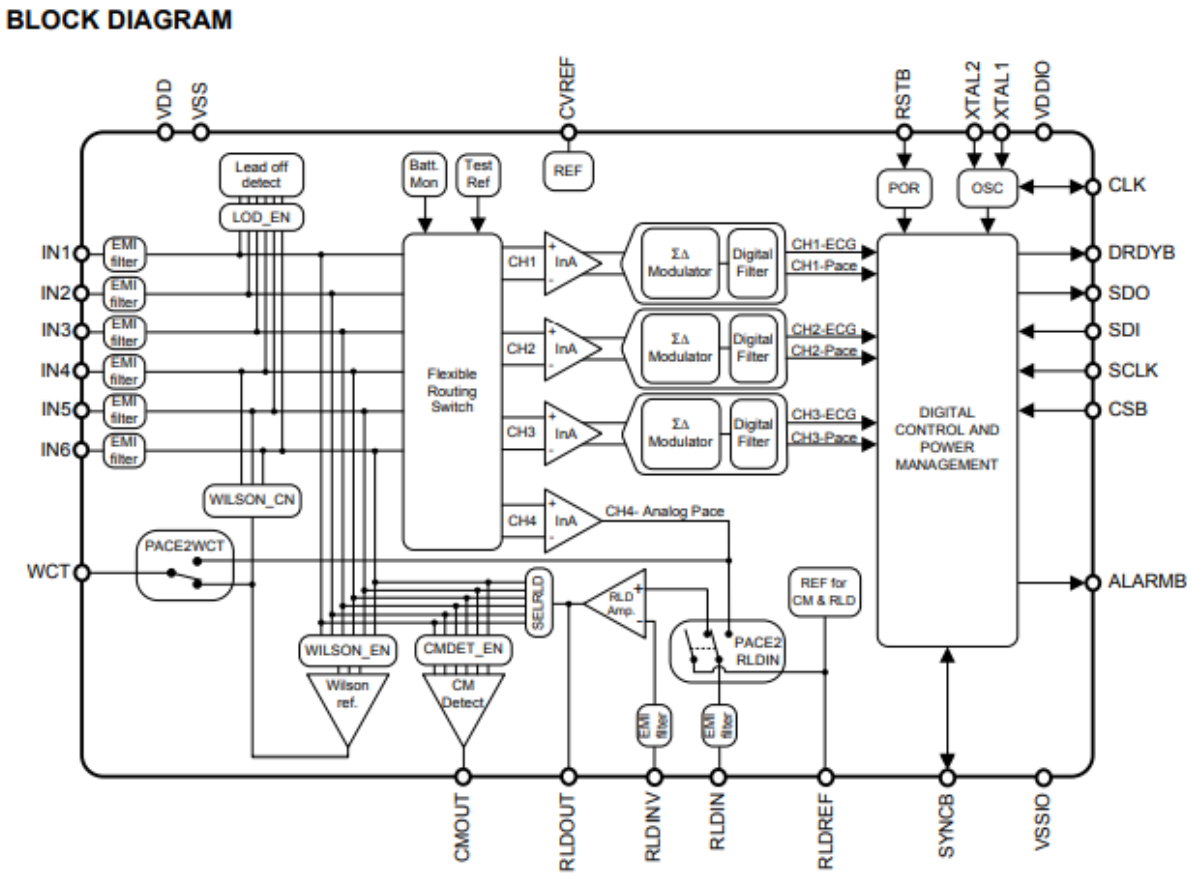

Рисунок 2 – Структурная схема ADS1293

Оцифрованный аналоговый сигнал от ADS1293 передается микроконтроллеру посредством Serial Periphial Interface (SPI).

Так же, в даташите приведена рекомендованная схема включения для трехэлектродной схемы(рисунок 3).

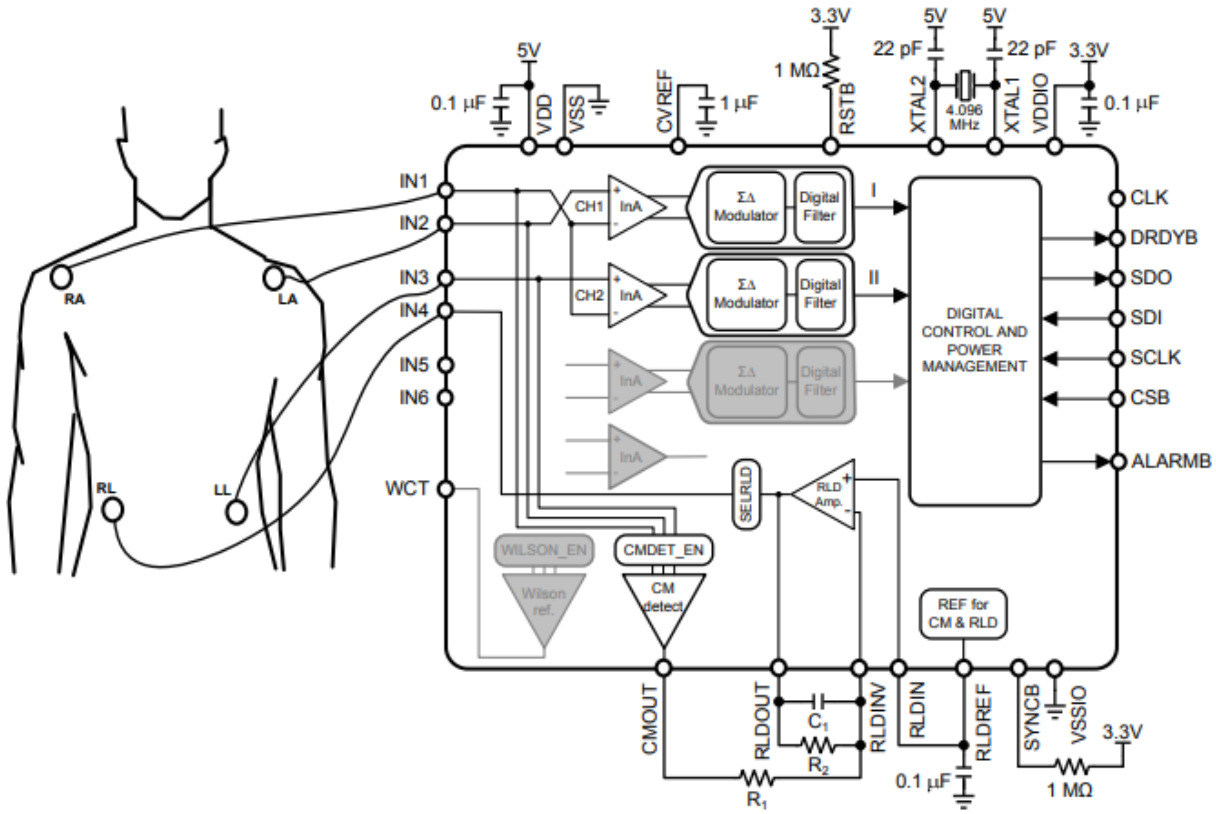

Figure 32. 3-Lead ECG Application

Рисунок 3 – Трехэлектродная схема включения

Нумерация и назначение выводов ADS1293 приведены ниже (рисунки 4, 5).

CONNECTION DIAGRAM

28-PIN LLP
(TOP VIEW)

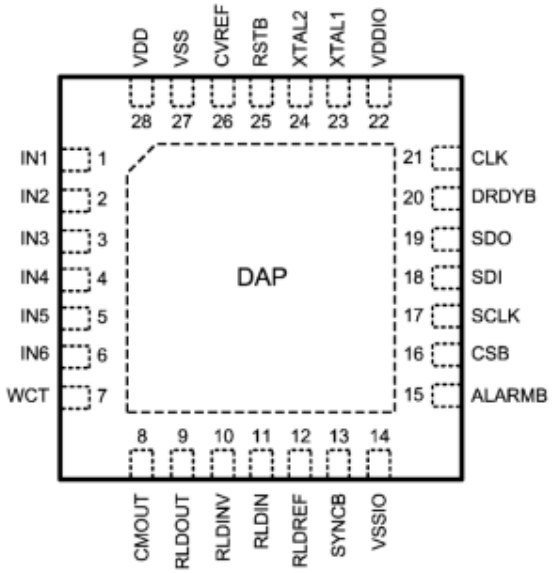

Рисунок 4 – Нумерация выводов

Table 2. Pin Descriptions

PIN		TYPE	FUNCTION
NO.	NAME		
1 - 6	IN1 - IN6	Analog Input	Electrode input signals
7	WCT	Analog Output	Wilson reference output or analog pace channel output
8	CMOUT	Output	Common-mode detector output
9	RLDOUT	Analog Output	Right leg drive amplifier output
10	RLDINV	Analog Input	Right leg drive amplifier negative input
11	RLDIN	Analog I/O	Right leg drive amplifier positive input or analog pace channel output
12	RLDREF	Analog Output	Internal right leg drive reference
13	SYNCB	Digital I/O	Sync bar; multiple-chip synchronization signal input or output
14	VSSIO	Digital Supply	Digital input/output supply ground
15	ALARMB	Digital Output	Alarm bar
16	CSB	Digital Input	Chip select bar
17	SCLK	Digital Input	Serial clock
18	SDI	Digital Input	Serial data input
19	SDO	Digital Output	Serial data output
20	DRDYB	Digital Output	Data ready bar
21	CLK	Digital I/O	Internal clock output or external clock input
22	VDDIO	Digital Supply	Digital input/output supply
23	XTAL1	Digital Input	External crystal for clock oscillator
24	XTAL2	Digital Input	External crystal for clock oscillator
25	RSTB	Digital Input	Reset bar
26	CVREF	Analog I/O	External cap for internal reference voltage
27	VSS	Analog Supply	Power supply ground
28	VDD	Analog Supply	Positive power supply
	DAP		No connect

Рисунок 5 – Назначение выводов

2.2 Выбор микроконтроллера

С учетом технического задания микроконтроллер должен обладать следующими свойствами:

– Интерфейс для работы с микросхемой ADS1293: SPI;

– Интерфейс для работы с внешней флеш-памятью: SPI или I^2C;

– Для передачи данных по Bluetooth: встроенный стек протокола Bluetooth;

– Малое энергопотребление;

– Свободные выводы для подключения индикатора и выводов прерываний от ADS1293;

Для решения задачи был выбран микроконтроллер STM32WB55RCV6 фирмы ST Microelectronics [3].STM32WB55 содержит два производительных ядра ARM-Cortex:

– ядро ARM® -Cortex® M4 (прикладное), работающее на частотах до 64 МГц, для пользовательских задач имеется модуль управления памятью, модуль плавающей точки, инструкции ЦОС (цифровой обработки сигналов), графический ускоритель (ART accelerator);

– ядро ARM®-Cortex® M0+ (радиоконтроллер) с тактовой частотой 32 МГц, управляющее радиотрактом и реализующее низкоуровневые функции сетевых протоколов;

Данный микроконтроллер включает в себя все необходимые периферийные устройства, такие как интерфейсы передачи данных SPI,необходимый для подключения к акселерометру, и радиоконтроллер с поддержкой Bluetooth.

Основные характеристики:

– типовое энергопотребление 50 мкА/МГц (при напряжении питания 3 В);

– потребление в режиме останова 1,8 мкА (радиочасть в режиме ожидания (standby));

– потребление в выключенном состоянии (Shutdown) менее 50 нА;

– диапазон допустимых напряжений питания 1,7…3,6 В (встроенный DC-DC–преобразователь и LDO-стабилизатор);

– рабочий температурный диапазон -40…105°C.

Структурная схема микроконтроллера приведена на рисунке 7, а назначение выводов портов корпуса на рисунке 6.

Figure 11. STM32WB55Rx VFQFPN68 pinout(1)(2)

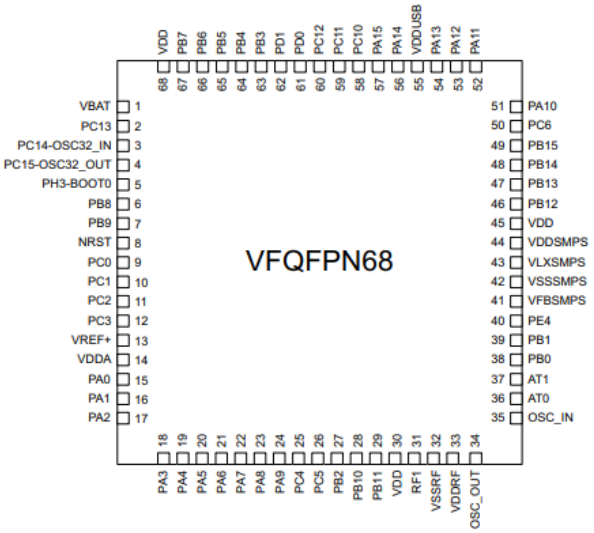

Рисунок 6 – Назначение выводов

Figure 2. STM32WB35xx block diagram

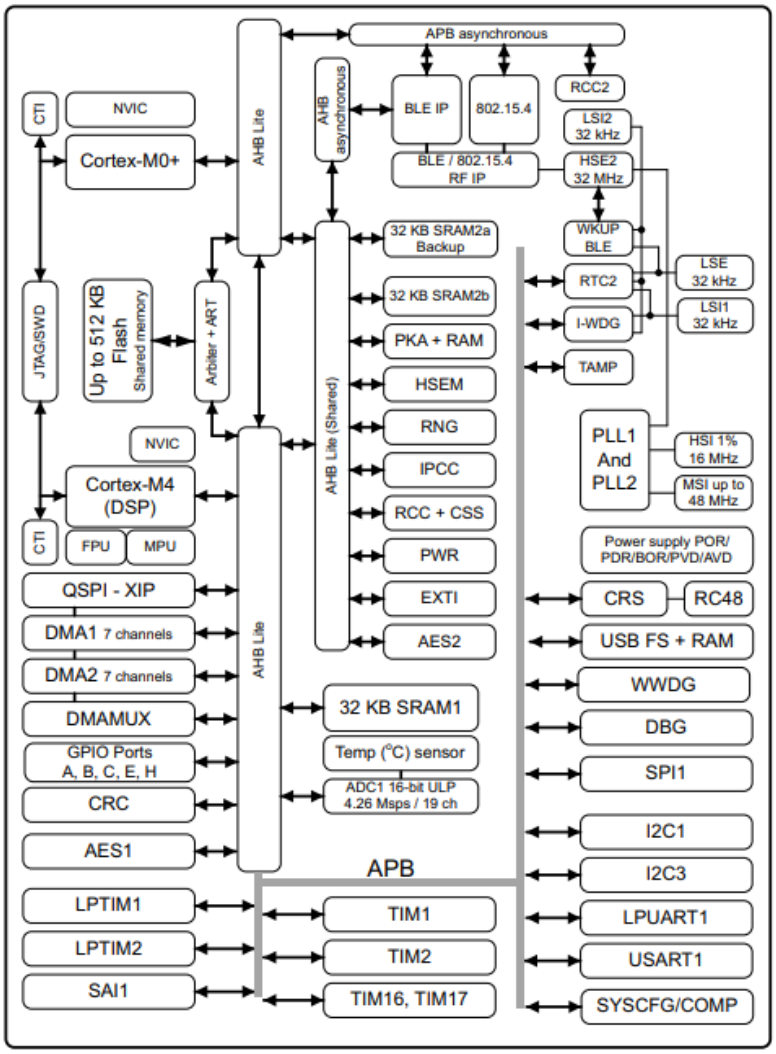

Рисунок 7 – Структурная схема

Подключение будет осуществляться согласно типовой схеме из Application note [4](рисунок 8).

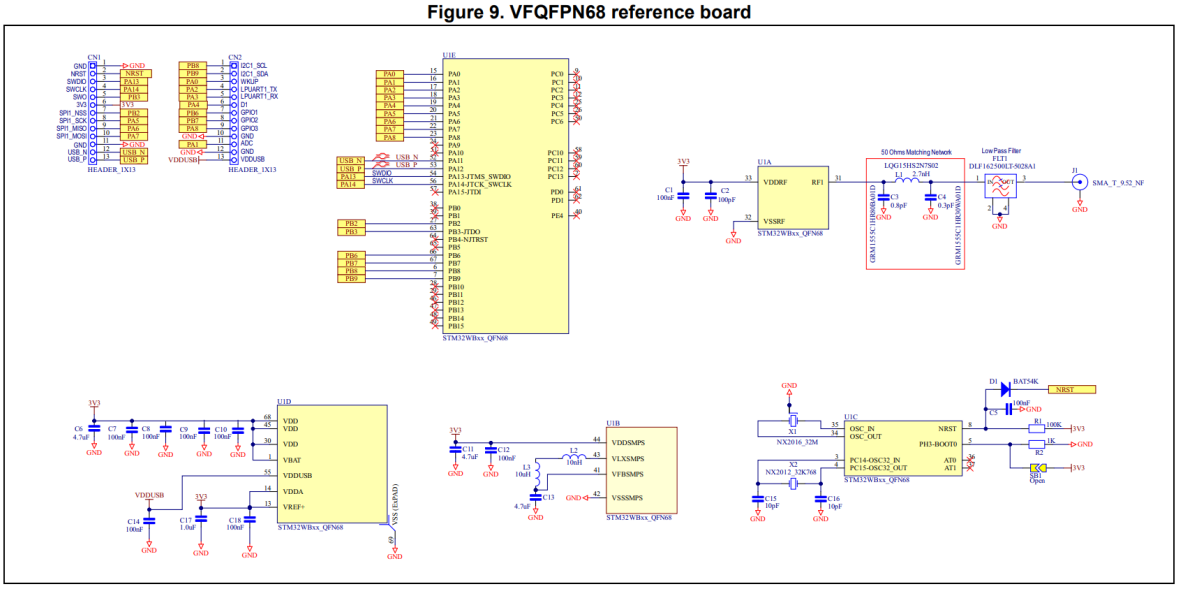

Рисунок 8 – Типовая схема подключения STM32WB55

2.3 Выбор встроенного носителя

В качестве носителя информации выберем последовательную FLASH-память серии W25Q [9]. Данная последовательная память может быть различной ёмкости — 8, 16, 32, 64, 128, 256 Мбит и т. д. Подключается такая память по интерфейсу SPI, а также по многопроводным интерфейсам Dual SPI, Quad SPI и QPI. Мы же пока будем подключим данную микросхему по обычному интерфейсу SPI.

Краткие основные характеристики W25Q:

– Потребляемая мощность и температурный диапазон:

– Напряжение питания 2.7…3.6 В

– Типичный потребляемый ток: 4 мА (активный режим), <1 мкА (в режиме снижения мощности)

– Рабочий температурный диапазон -40°C…+85°C.

Гибкая архитектура с секторами размером 4 кбайт:

– Посекторное стирание (размер каждого сектора 4 кбайт)

– Программирование от 1 до 256 байт

– До 100 тыс. циклов стирания/записи

– 20-летнее хранение данных

Максимальная частота работы микросхемы:

– 104 МГц в режиме SPI

– 208/416 МГц — Dual / Quad SPI

Также микросхема существует в различных корпусах, но в большинстве случаев распространён корпус SMD SO8. Распиновка микросхемы следующая(рисунок 9).

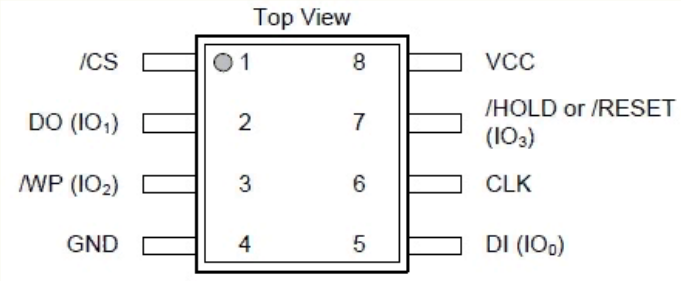

Рисунок 9 – Распиновка W25Q128

Описание выводов из [9](рисунок 10).

3.3 Pin Description SOIC / VSOP 208-mil, WSON 6x5-mm / 8x6-mm

PIN NO.	PIN NAME	I/O	FUNCTION
1	/CS	I	Chip Select Input
2	DO (IO1)	I/O	Data Output (Data Input Output 1)(1)
3	/WP (IO2)	I/O	Write Protect Input (Data Input Output 2)(2)
4	GND		Ground
5	DI (IO0)	I/O	Data Input (Data Input Output 0)(1)
6	CLK	I	Serial Clock Input
7	/HOLD or /RESET (IO3)	I/O	Hold or Reset Input (Data Input Output 3)(2)
8	VCC		Power Supply

Рисунок 10 – Описание выводов W25Q128

К микроконтроллеру подключается по стандартному интерфейсу SPI.

2.4 Блок питания

Питание схемы будет осуществляться с помощью аккумулятора LP-310-233350 [6] и DC-DC преобразователя LM3671 [5]. Аккумулятор литий-полимерный LP-310-233350 имеет номинальную емкость 310 мАч, номинальное напряжение 3,7 В, вес 8г. Длина: 50±1 мм. Ширина: 33±1 мм. Толщина: 2,3±1 мм.

DC-DC преобразователь LM3671MF с фиксированным выходным напряжением 3,3 В. Типичный ток покоя 16 мкА, типичный ток в выключенном состоянии - 0.01 мкА, максимальная нагрузка по току 600 мА.

Подключение DC-DC преобразователя будет будет осуществляться согласно типовой схеме из Data Sheet [5] (рисунок 11)

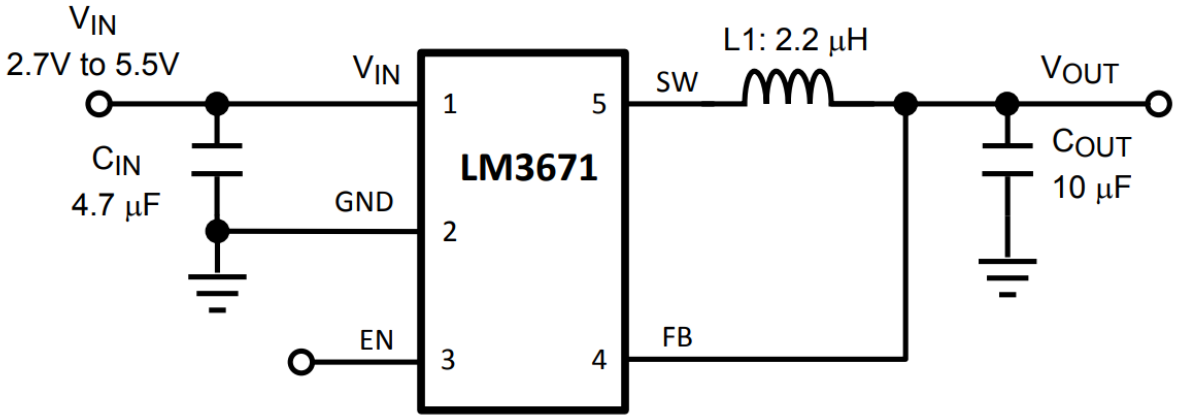

Рисунок 11 – Типовая схема включения DC-DC–преобразователя

3 РАЗРАБОТКА ПРОГРАММЫ

3.1 Разработка алгоритма

Проанализируем задание, учитывая ранее описанное. Необходимо получать данные от AFE микросхемы ADS1293 по интерфейсу SPI, записывать их во флеш-память W25Q128. Так же данные передаются по интерфейсу Bluetooth.

Для работы программы необходимо для начала разработать алгоритм. Алгоритм нашего устройства представлен на рисунке ??.

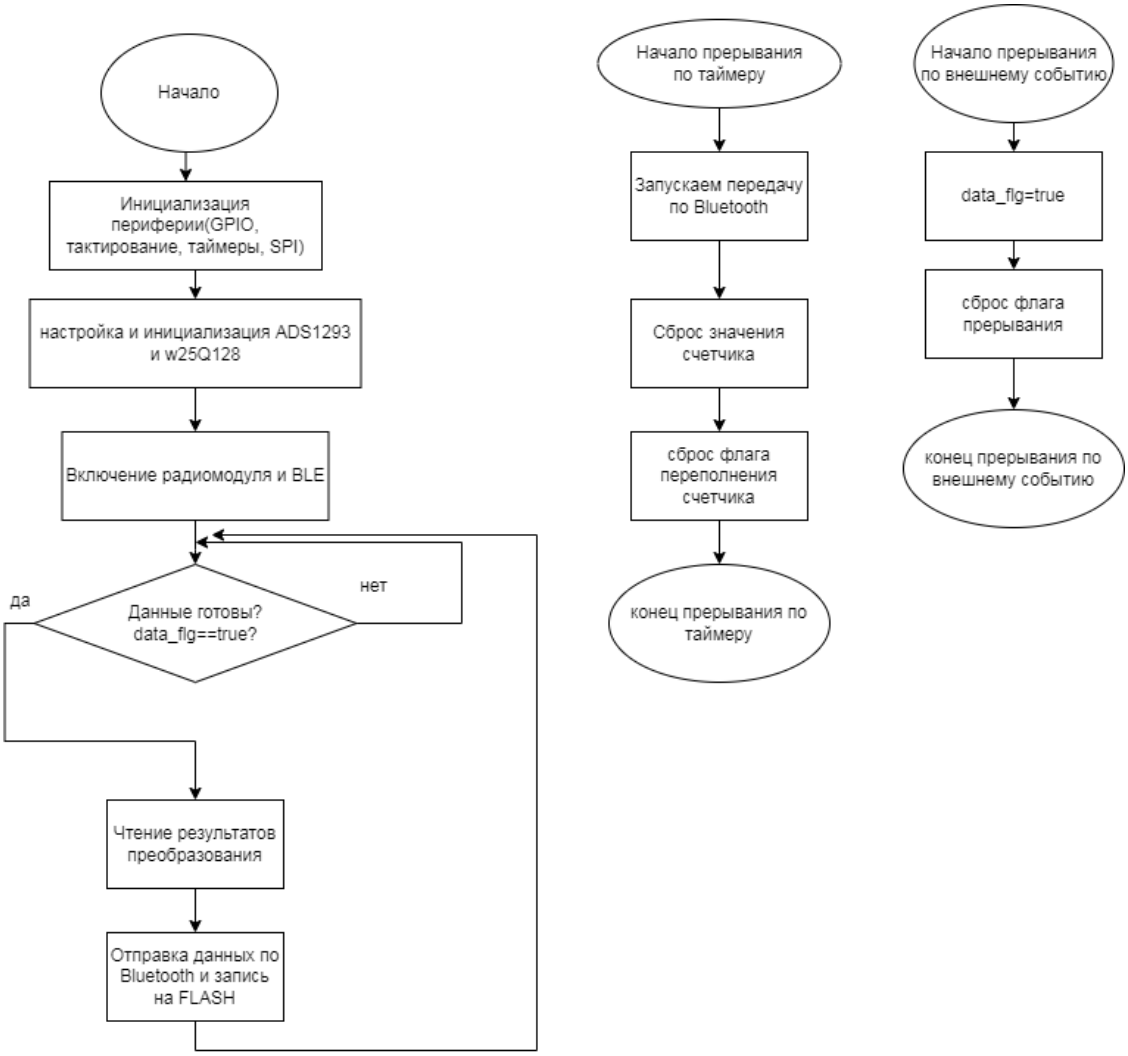

Рисунок 12 – Алгоритм работы устройства

3.2 Разработка кода

3.2.1 Выбор программного обеспечения

Для разработки ПО под STM32 можно использовать различные IDE. Самые популярные — IAR, Keil, Coocox (Eclipse). Мы же пойдем по пути, который с недавних пор абсолютно бесплатно и в полном объеме предоставляет сама ST.

STM32CubeIDE – многофункциональное средство разработки, являющееся частью экосистемы STM32Cube от компании STMicroelectronics. STM32CubeIDE – платформа разработки C/C++ с IP-конфигурацией, генерацией и компиляцией кода и способностью прошивки микроконтроллеров STM32. Программное обеспечение построено на платформе ECLIPSE™/CDT и пакетов программ GCC для разработки, а также отладчика GDB для прошивки микроконтроллера.

Какие плюсы у данного ПО: абсолютно бесплатно, нет ограничения по размеру кода, есть неплохой отладчик, простая установка и настройка. Так же, стоит отметить, что данная платформа кроссплатформенная - есть версии для Windows, Linux и даже MacOS. Ознакомиться с STM32CubeIDE можно в [7]

3.2.2 Инициализация периферии

В STM32CubeIDE встроен STM32CubeMx – программный продукт, позволяющий при помощи достаточно понятного графического интерфейса произвести настройку любой имеющейся на борту микроконтроллера периферии. Подробнее об этом можно прочитать в [8]

В нашем случае нужно создать проект, выбрать микроконтроллер и подключить периферию - таймер, SPI, тактирование. Все это настраивается в графическом интерфейсе.

Сначала в настройках Reset and Clock Controller(RCC) подключаем кварцевые резонаторы, как показано на рисунке 13.

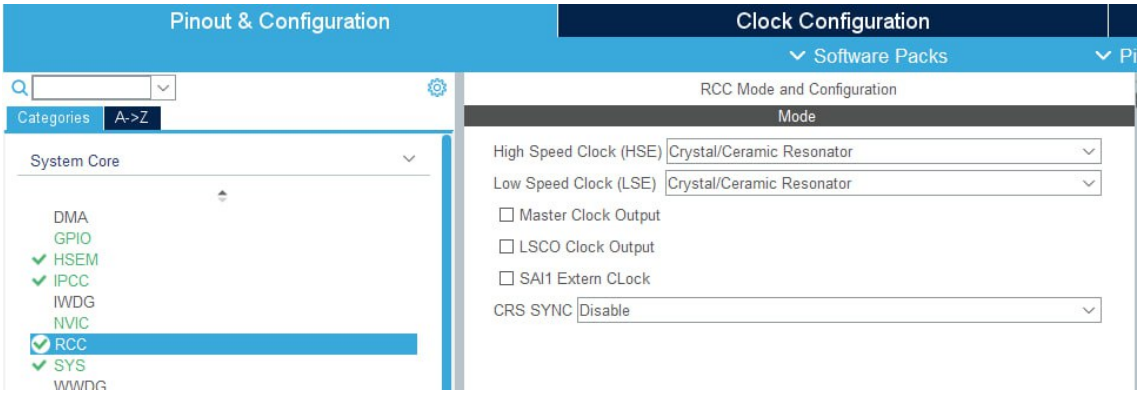

Рисунок 13 – Настройки RCC

Затем подключим порты ввода-вывода и настроим их как внешний источник прерываний, как показано на рисунке 14.

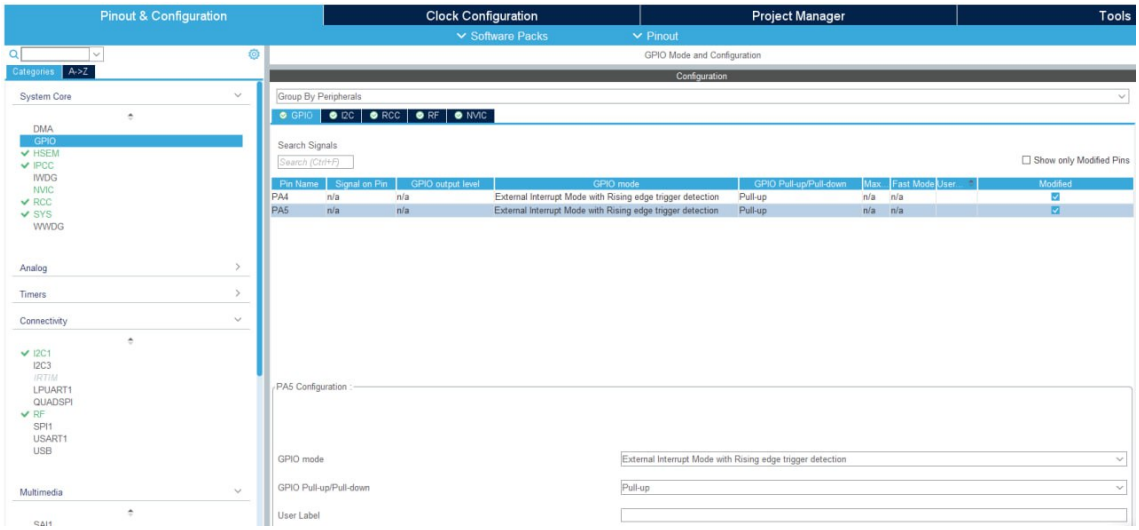

Рисунок 14 – Настройки портов ввода-вывода

Затем подключим порты ввода-вывода и настроим их как внешний источник прерываний, как показано на рисунке 15.

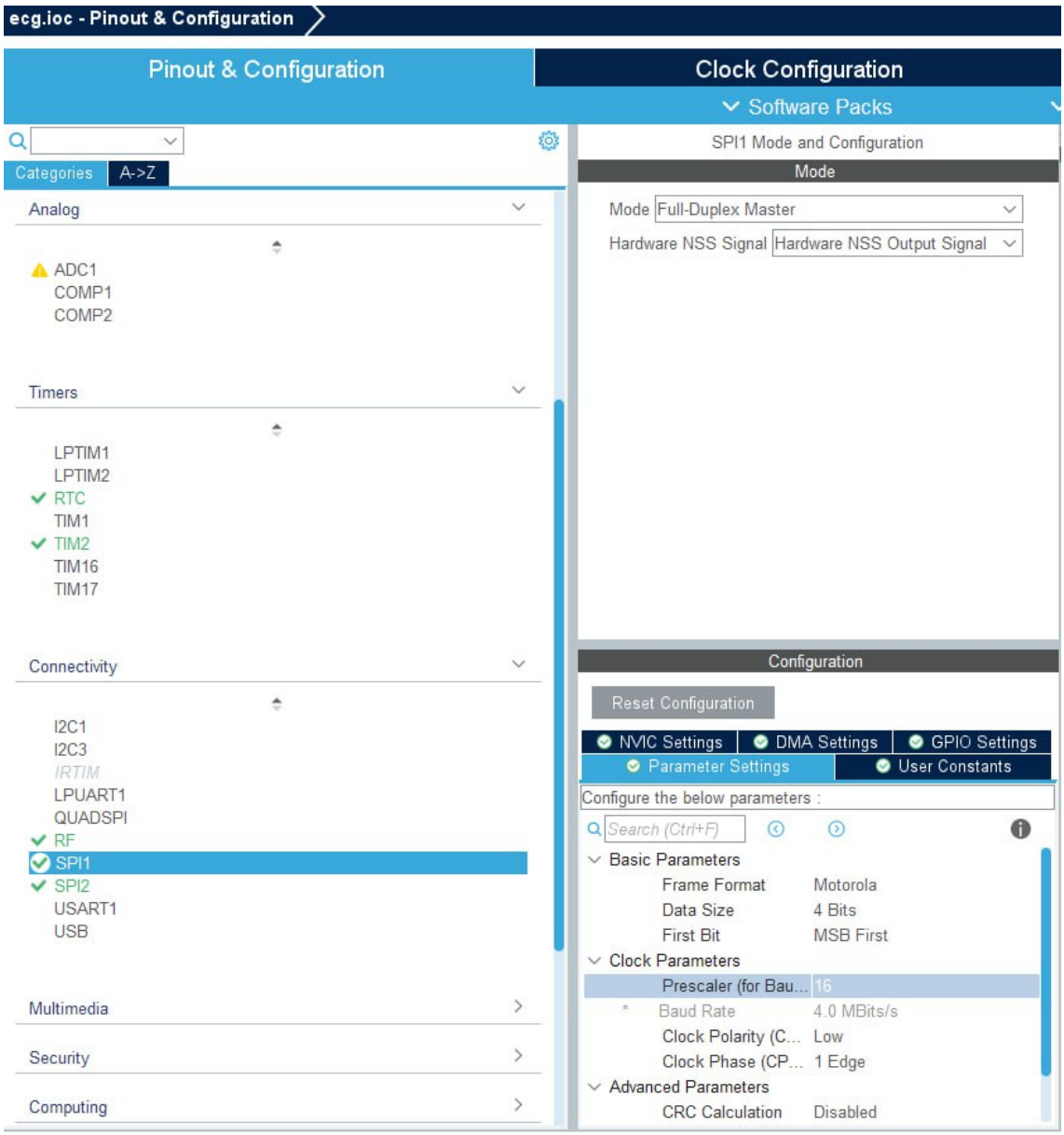

Рисунок 15 – Настройки портов ввода-вывода

После этого можно настроить тактирование на вкладке Clock Configuration, как показано на рисунке 16.

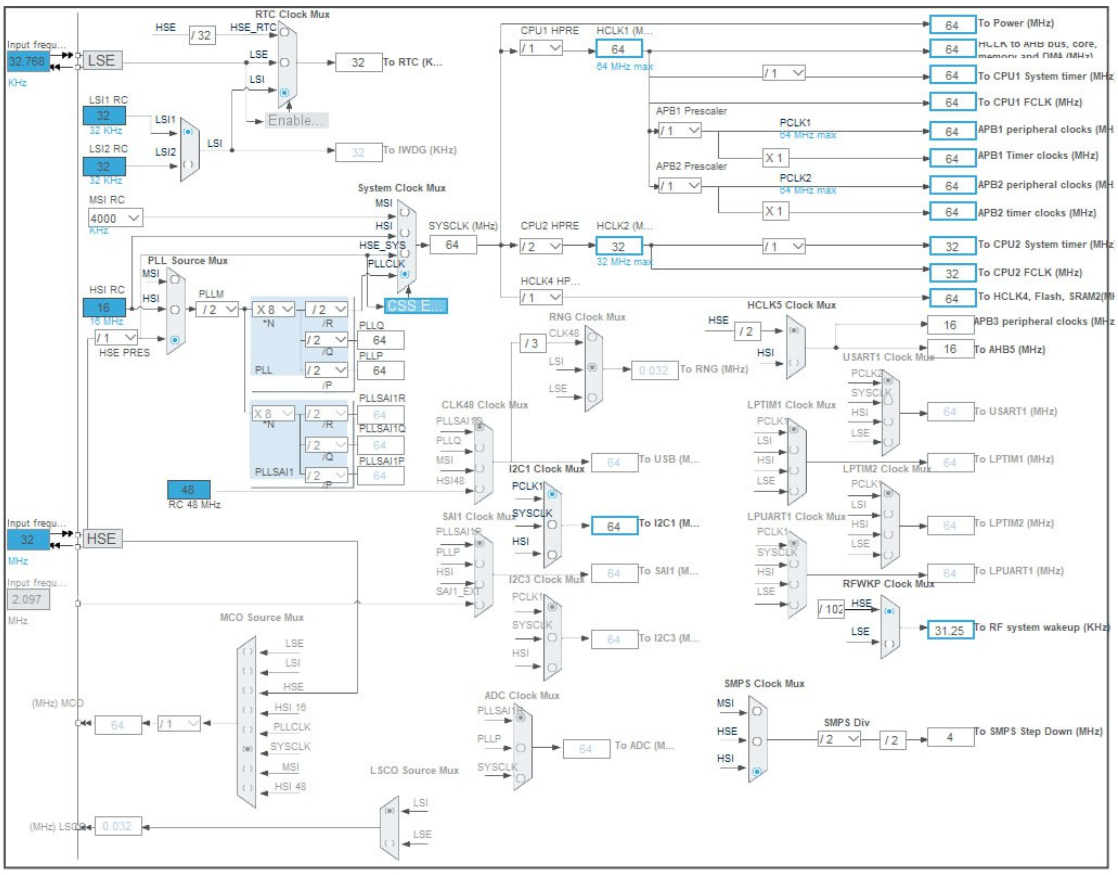

Рисунок 16 – Настройки тактирования

Так же включаем таймер - он необходим для того, чтобы

Для включения стека Bluetooth необходимо активировать Inter-Process Communication Controller(IPCCC), Hardware Semaphore (HSEM) (необходим для синхронизации процессов, запущеных на разных ядрах), как показано на рисунках 17, 18, включить Radio System(RF),как показано на рисунке 19.

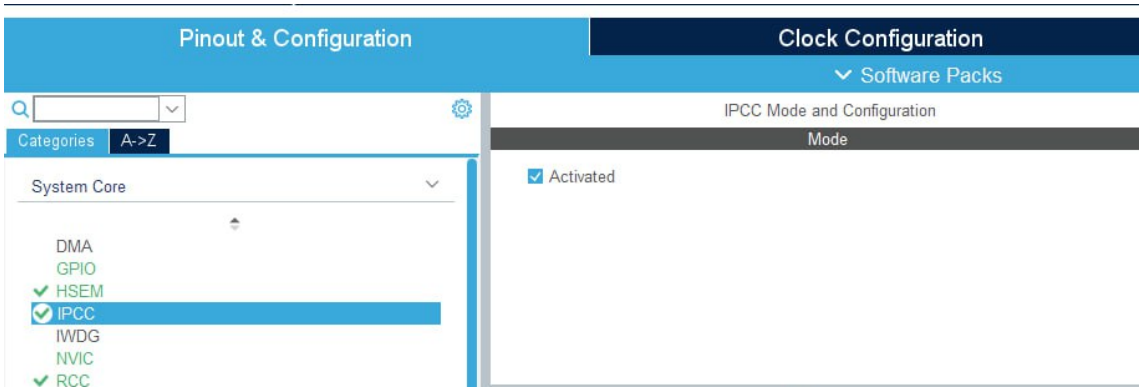

Рисунок 17 – Активируем IPCC

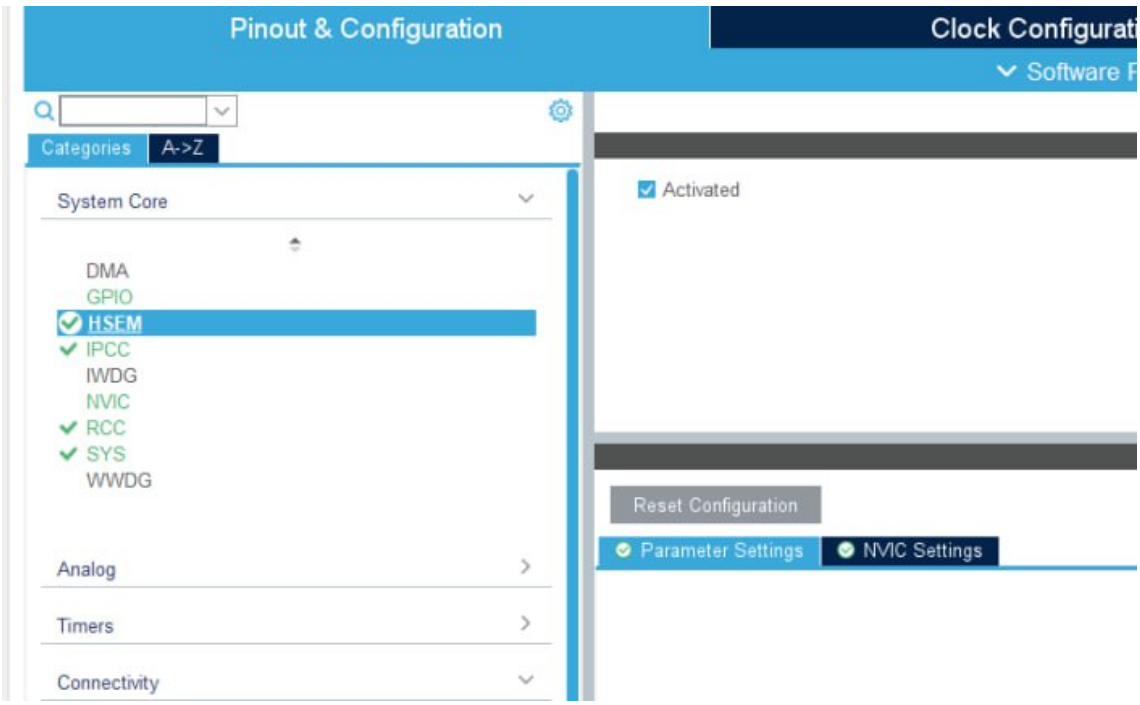

Рисунок 18 – Активируем HSEM

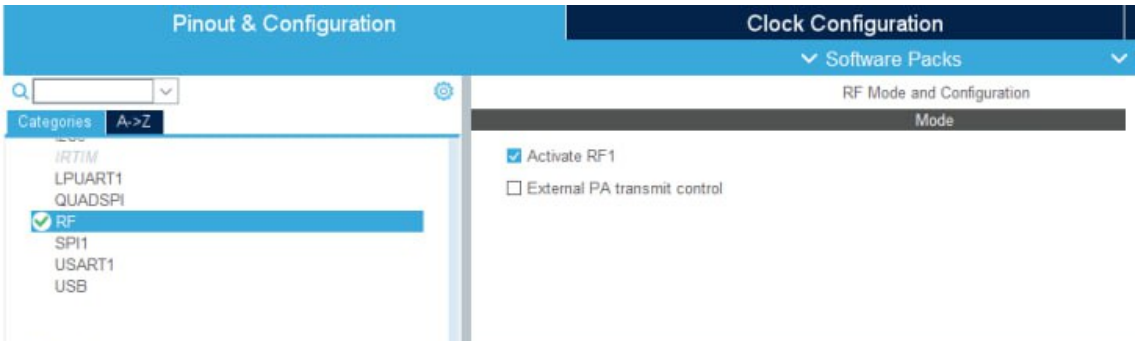

Рисунок 19 – Активируем RF

Теперь разблокирована вкладка STM32WPAN, где нужно включить стек Bluetooth. В настройках указываем, что конечное устройство будет являться сервером(то есть транслировать данные другим устройствам). Настройки показаны на рисунке 20.

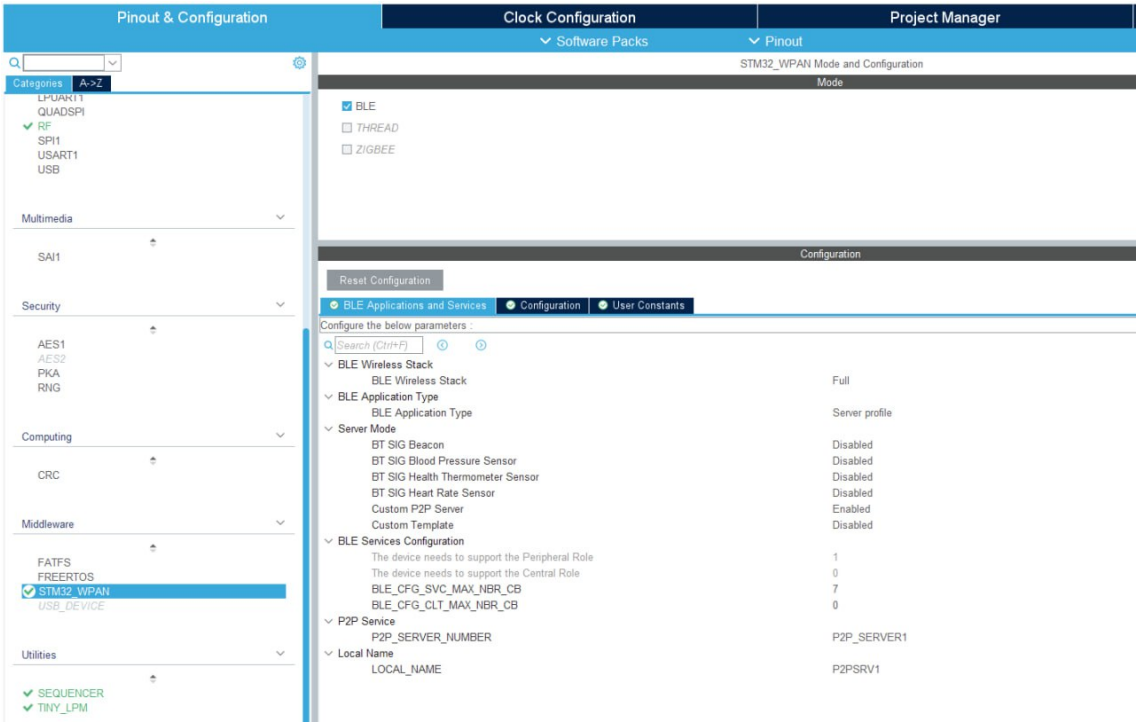

Рисунок 20 – Активируем Bluetooth

Дальше все настройки будут происходить в коде. После активации стека Bluetooth в директории проекта появляется множество файлов, отвечающих за функции радиоядра и сервисы Bluetooth.

Стоит отметить, что в данной линейке микропроцессоров используется еще не полноценная операционная система реального времени, но уже вполне функциональный диспетчер задач - Sequensor.

Для запуска этого диспетчера надо добавить строку в файл main.c вызов диспетчера задач(UTIL_SEQ_Run), а так же подключить библиотеки, содержащие все необходимые функции.

```
/* USER CODE BEGIN Includes */
#include "stm32_seq.h"  //подключаем  библиотеку Sequensor
/* USER CODE END Includes */


/* USER CODE BEGIN PV */
uint8_t send_ble[2] = {0,0};
/* USER CODE END PV */

/* USER CODE BEGIN WHILE */
while (1)
{
  UTIL_SEQ_Run(UTIL_SEQ_DEFAULT);  //запускаем  его в стандартном режиме.
```

```
/* USER CODE END WHILE */
}
```

Активируем прерывания от Inter-Process Communication Controller

```
/* USER CODE BEGIN 1 */
void RTC_WKUP_IRQHandler(void)
{
 HW_TS_RTC_Wakeup_Handler();
}

void IPCC_C1_TX_IRQHandler(void)
{
 HW_IPCC_Tx_Handler();

 return;
}

void IPCC_C1_RX_IRQHandler(void)
{
 HW_IPCC_Rx_Handler();
 return;
}
/* USER CODE END 1 */
```

В файле app_ble.c уже автоматически прописано соединение клиента с сервером, описаны оснновные функции инициализации и настроек. Можно изменить этот файл, добавив свой функционал. Добавим простое включение и выключение светодиода при активной передаче данных. Файл генерируется большой, в приложении не приводим, отметим лишь то место, куда добавляем собственный код. Светодиод так же подключаем с помощью графического интерфейса, и присваиваем ему метку LED_Pin, а порту присваиваем LED_GPIO_Pin.

```
/* USER CODE BEGIN RADIO_ACTIVITY_EVENT*/
HAL_GPIO_WritePin(LED_GPIO_Port, LED_Pin, GPIO_PIN_SET); // включение светодиода
HAL_Delay(5); //небольшая задержка
HAL_GPIO_WritePin(LED_GPIO_Port, LED_Pin, GPIO_PIN_RESET); //отключение светодиода
/* USER CODE END RADIO_ACTIVITY_EVENT*/
```

В файле p2p_server_app.c описаны функции, занимающиеся именно протоколом Bluetooth. Так же можно изменить этот файл, добавив свой функционал.

```
/* USER CODE BEGIN PTD */
unsigned char inputMessage = 0; //локальная переменная
uint8_t local[2] = {0,0};//массив для передачи
/* USER CODE END PTD */

/* USER CODE BEGIN PD */
void P2PS_Send_Notification(void);//прототип функции, отправляющей данные по Bluetooth
/* USER CODE END PD */

/* USER CODE BEGIN P2PS_STM_WRITE_EVT */
inputMessage = pNotification->DataTransfered.pPayload[1]; //присваиваем перерменной значение из элемента структуры
/* USER CODE END P2PS_STM_WRITE_EVT */
g
/* USER CODE BEGIN FD_LOCAL_FUNCTIONS*/
void P2PS_Send_Notification(void)
{
// формируем посылку
local[1] = send_ble[0];
local[0] = 1;
P2PS_STM_App_Update_Char(P2P_NOTIFY_CHAR_UUID, (uint8_t *)(&local));//отправляем данные по Bluetooth.
}
/* USER CODE END FD_LOCAL_FUNCTIONS*/

/* USER CODE BEGIN P2PS_APP_Init */
UTIL_SEQ_RegTask( 1<< CFG_TASK_RECEIVE_UART_ID, UTIL_SEQ_RFU, P2PS_Send_Notification);
/* USER CODE END P2PS_APP_Init */
```

ЗАКЛЮЧЕНИЕ

В данном курсовом проекте рассмотрены принципы разработки устройств на базе микроконтроллеров. Был разработан носимый монитор ЭКГ. Данные передаются по интерфейсу Bluetooth и записываются на карту памяти. В процессе работы были разработаны структурная и принципиальная схемы устройства, были проведены необходимые расчёты для получения заданной погрешности, осуществлен выбор микроконтроллера и вспомогательных компонентов схемы.

Конфигуратор кода STM32CubeIDE предоставляет все необходимые библиотеки для реализации устройства, а также обеспечивает необходимые настройки микроконтроллера перед началом реализации алгоритма основной программы. Далее был разработан алгоритм программы и текст программы на языке Си.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1 Беляев, А. О. Анализ аналоговых характеристик микросхемы ADS1293 для применения в медицинской технике / А. О. Беляев, В. В. Кириенко // Инженерный вестник Дона. – 2014. – № 3(30). – С. 67. – EDN TFXFTD

2 Data Sheet на AFE микросхему ADS1293 [Электронный ресурс]. URL:https://radioaktiv.ru/ds/ti/snas602b.pdf (Дата обращения: 6.05.2023)

3 Data Sheet на микроконтроллер STM32WB55CCU6 [Электронный ресурс]. URL:https://www.st.com/resource/en/datasheet/stm32wb55cc.pdf (Дата обращения: 10.05.2023)

4 Application note на микроконтроллеры серии STM32WB [Электронный ресурс]. URL:https://www.st.com/resource/en/application_note/an5165-development-of-rf-hardware-using-stm32wb-microcontrollers-stmicroelectronics.pdf (Дата обращения: 11.05.2023)

5 Data Sheet на DC-DC преобразователь LM3671/-Q1 [Электронный ресурс]. URL:https://static.chipdip.ru/lib/091/DOC001091994.pdf (Дата обращения: 16.05.2023)

6 Спецификация на Li-pol аккумулятор LP-310-233350 [Электронный ресурс]. URL:https://static.chipdip.ru/lib/412/DOC005412828.pdf (Дата обращения: 11.05.2023)

7 STM32CubeIDE - Integrated Development Environment for STM32 [Электронный ресурс]. URL:https://www.st.com/en/development-tools/stm32cubeide.html (Дата обращения: 2.05.2023)

8 STM32. Быстрый старт с STM32CubeMx [Электронный ресурс]. URL:https://microtechnics.ru/stm32cube-sozdanie-proekta/ (Дата обращения: 2.05.2023)

9 Data Sheet на последовательную FLASH память W25Q128FV [Электронный ресурс]. URL:https://www.winbond.com/resource-files/w25q128fv%20rev.l%2008242015.pdf (Дата обращения: 2.05.2023)

ОПРЕДЕЛЕНИЯ, ОБОЗНАЧЕНИЯ И СОКРАЩЕНИЯ

AFE	–	Analog Front End	8, 17
SPI	–	Serial Periphial Interface	9
АЦП	–	Аналого-цифровой преобразователь	8
ИС	–	интегральная схема	8
ОУ	–	операционный усилитель	8
ЭКГ	–	электрокардиограмма	8

ПРИЛОЖЕНИЕ А

Основная программа

```
/* Includes ------------------------------------------------------------------*/
#include "main.h"

/* Private includes ----------------------------------------------------------*/
/* USER CODE BEGIN Includes */
#include "ads1293.h"
#include "w25q.h"
#include "stdbool.h"
/* USER CODE END Includes */

#define ads_hspi &hspi1
 IPCC_HandleTypeDef hipcc;

RTC_HandleTypeDef hrtc;

SPI_HandleTypeDef hspi1;
SPI_HandleTypeDef hspi2;

TIM_HandleTypeDef htim2;

uint8_t databuffer[256]; // 256 байт = 1 страница памяти
uint32_t buffer[3]; //массив для одного измерения
uint8_t send_ble[2]={0,0};
bool data_flg = false;

void SystemClock_Config(void);
void PeriphCommonClock_Config(void);
static void MX_GPIO_Init(void);
static void MX_IPCC_Init(void);
static void MX_RF_Init(void);
static void MX_RTC_Init(void);
static void MX_SPI1_Init(void);
static void MX_SPI2_Init(void);
static void MX_TIM2_Init(void);

int main(void)
{
  HAL_Init();
  MX_APPE_Config();

  SystemClock_Config();

  PeriphCommonClock_Config();

   MX_IPCC_Init();
  MX_GPIO_Init();
  MX_RF_Init();
  MX_RTC_Init();
  MX_SPI1_Init();
  MX_SPI2_Init();
  MX_TIM2_Init();
  ADS1293_INIT(); //инициализация ADS1293
  W25qxx_Init();//инициализация флешки

  MX_APPE_Init();
```

```
 ADS1293_start_conv();// запуск преобразования


 while (1)
 {
  MX_APPE_Process();

     if(data_flg==true)//если данные готовы(о чем сообщит прерывание от DRDYB)
     {
          ADS1293_stream_read(buffer);//читаем данные с ADS1293 в массив buffer

          //с ads приходит 24 битные данные в количестве 3 штук = 9 байт за один проход
          //на флешку пишем постранично, передавая данные побайтово

          for (int i=0;i<=28;i++){//пока количество снятий показаний меньше 28, новые данные записываются в буферн
               //там они ждут, пока накопятся достаточное количество для записи на флеш память(записываем стра
               databuffer[0+3*i] = (buffer[0] & 0x000000ff);
               databuffer[1+3*i] = (buffer[0] & 0x0000ff00) >> 8;
               databuffer[2+3*i] = (buffer[0] & 0x00ff0000) >> 16;
               databuffer[3+3*i] = (buffer[0] & 0xff000000) >> 24;

               databuffer[4+3*i] = (buffer[1] & 0x000000ff);
               databuffer[5+3*i] = (buffer[1] & 0x0000ff00) >> 8;
               databuffer[6+3*i] = (buffer[1] & 0x00ff0000) >> 16;
               databuffer[7+3*i] = (buffer[1] & 0xff000000) >> 24;


               databuffer[8+3*i] = (buffer[2] & 0x000000ff);
               databuffer[9+3*i] = (buffer[2] & 0x0000ff00) >> 8;
               databuffer[10+3*i] = (buffer[2] & 0x00ff0000) >> 16;
               databuffer[11+3*i] = (buffer[2] & 0xff000000) >> 24;

               if(i>28){//если превысили число измерений, то
                    i=0;//сбрасываем счетчик
                    for (int numberPage=0; numberPage<=w25qxx.PageCount; numberPage=numberPage+256){
                         //записываем данные на флешку,
                         //пока numberPage номер страницы меньше чем число страниц на всей флешке

                          uint8_t clear = W25qxx_IsEmptyPage(0, 40);//в clear записывается 1, если чистая страница, 0 ес

                          if(clear==1) W25qxx_WritePage(databuffer, numberPage, 0, 256);//если чисто пишем 256 байт


                    }
               }
          }
     }

  }
}

void TIM2_IRQHandler(void) //прерывание по таймеру
{

     P2PS_Send_Notification();

     __HAL_TIM_SET_COUNTER(&htim2, 0); //Сбрасываем значение счетчика для дальнейшего счета
     HAL_TIM_IRQHandler(&htim2); //Сброс флагов таймера
}
```

```
void EXTI4_15_IRQHandler(void) //Прерывание по приходу импульса с DRDYB
{

      data_flg = true;

      HAL_GPIO_EXTI_IRQHandler(GPIO_PIN_5|GPIO_PIN_6); //сброс флага прерывания
}

void SystemClock_Config(void)
{
 RCC_OscInitTypeDef RCC_OscInitStruct = {0};
 RCC_ClkInitTypeDef RCC_ClkInitStruct = {0};
 HAL_PWR_EnableBkUpAccess();
 __HAL_RCC_LSEDRIVE_CONFIG(RCC_LSEDRIVE_LOW);
 __HAL_PWR_VOLTAGESCALING_CONFIG(PWR_REGULATOR_VOLTAGE_SCALE1);
 RCC_OscInitStruct.OscillatorType = RCC_OSCILLATORTYPE_HSI|RCC_OSCILLATORTYPE_HSE
                    |RCC_OSCILLATORTYPE_LSE|RCC_OSCILLATORTYPE_MSI;
 RCC_OscInitStruct.HSEState = RCC_HSE_ON;
 RCC_OscInitStruct.LSEState = RCC_LSE_ON;
 RCC_OscInitStruct.HSIState = RCC_HSI_ON;
 RCC_OscInitStruct.MSIState = RCC_MSI_ON;
 RCC_OscInitStruct.HSICalibrationValue = RCC_HSICALIBRATION_DEFAULT;
 RCC_OscInitStruct.MSICalibrationValue = RCC_MSICALIBRATION_DEFAULT;
 RCC_OscInitStruct.MSIClockRange = RCC_MSIRANGE_6;
 RCC_OscInitStruct.PLL.PLLState = RCC_PLL_ON;
 RCC_OscInitStruct.PLL.PLLSource = RCC_PLLSOURCE_MSI;
 RCC_OscInitStruct.PLL.PLLM = RCC_PLLM_DIV1;
 RCC_OscInitStruct.PLL.PLLN = 32;
 RCC_OscInitStruct.PLL.PLLP = RCC_PLLP_DIV2;
 RCC_OscInitStruct.PLL.PLLR = RCC_PLLR_DIV2;
 RCC_OscInitStruct.PLL.PLLQ = RCC_PLLQ_DIV2;
 if (HAL_RCC_OscConfig(&RCC_OscInitStruct) != HAL_OK)
 {
  Error_Handler();
 }

 RCC_ClkInitStruct.ClockType = RCC_CLOCKTYPE_HCLK4|RCC_CLOCKTYPE_HCLK2
                    |RCC_CLOCKTYPE_HCLK|RCC_CLOCKTYPE_SYSCLK
                    |RCC_CLOCKTYPE_PCLK1|RCC_CLOCKTYPE_PCLK2;
 RCC_ClkInitStruct.SYSCLKSource = RCC_SYSCLKSOURCE_PLLCLK;
 RCC_ClkInitStruct.AHBCLKDivider = RCC_SYSCLK_DIV1;
 RCC_ClkInitStruct.APB1CLKDivider = RCC_HCLK_DIV1;
 RCC_ClkInitStruct.APB2CLKDivider = RCC_HCLK_DIV1;
 RCC_ClkInitStruct.AHBCLK2Divider = RCC_SYSCLK_DIV2;
 RCC_ClkInitStruct.AHBCLK4Divider = RCC_SYSCLK_DIV1;

}

void PeriphCommonClock_Config(void)
{
 RCC_PeriphCLKInitTypeDef PeriphClkInitStruct = {0};
 PeriphClkInitStruct.PeriphClockSelection = RCC_PERIPHCLK_SMPS|RCC_PERIPHCLK_RFWAKEUP;
 PeriphClkInitStruct.RFWakeUpClockSelection = RCC_RFWKPCLKSOURCE_LSE;
 PeriphClkInitStruct.SmpsClockSelection = RCC_SMPSCLKSOURCE_HSE;
 PeriphClkInitStruct.SmpsDivSelection = RCC_SMPSCLKDIV_RANGE0;

}
```

```
static void MX_IPCC_Init(void)
{
  hipcc.Instance = IPCC;
  if (HAL_IPCC_Init(&hipcc) != HAL_OK)
  {
    Error_Handler();
  }
}

static void MX_RTC_Init(void)
{
  hrtc.Instance = RTC;
  hrtc.Init.HourFormat = RTC_HOURFORMAT_24;
  hrtc.Init.AsynchPrediv = CFG_RTC_ASYNCH_PRESCALER;
  hrtc.Init.SynchPrediv = CFG_RTC_SYNCH_PRESCALER;
  hrtc.Init.OutPut = RTC_OUTPUT_DISABLE;
  hrtc.Init.OutPutPolarity = RTC_OUTPUT_POLARITY_HIGH;
  hrtc.Init.OutPutType = RTC_OUTPUT_TYPE_OPENDRAIN;
  hrtc.Init.OutPutRemap = RTC_OUTPUT_REMAP_NONE;
}

static void MX_SPI1_Init(void)
{
  hspi1.Instance = SPI1;
  hspi1.Init.Mode = SPI_MODE_MASTER;
  hspi1.Init.Direction = SPI_DIRECTION_2LINES;
  hspi1.Init.DataSize = SPI_DATASIZE_4BIT;
  hspi1.Init.CLKPolarity = SPI_POLARITY_LOW;
  hspi1.Init.CLKPhase = SPI_PHASE_1EDGE;
  hspi1.Init.NSS = SPI_NSS_HARD_OUTPUT;
  hspi1.Init.BaudRatePrescaler = SPI_BAUDRATEPRESCALER_16;
  hspi1.Init.FirstBit = SPI_FIRSTBIT_MSB;
  hspi1.Init.TIMode = SPI_TIMODE_DISABLE;
  hspi1.Init.CRCCalculation = SPI_CRCCALCULATION_DISABLE;
  hspi1.Init.CRCPolynomial = 7;
  hspi1.Init.CRCLength = SPI_CRC_LENGTH_DATASIZE;
  hspi1.Init.NSSPMode = SPI_NSS_PULSE_ENABLE;
}

static void MX_SPI2_Init(void)
{
  hspi2.Instance = SPI2;
  hspi2.Init.Mode = SPI_MODE_MASTER;
  hspi2.Init.Direction = SPI_DIRECTION_2LINES;
  hspi2.Init.DataSize = SPI_DATASIZE_4BIT;
  hspi2.Init.CLKPolarity = SPI_POLARITY_LOW;
  hspi2.Init.CLKPhase = SPI_PHASE_1EDGE;
  hspi2.Init.NSS = SPI_NSS_HARD_OUTPUT;
  hspi2.Init.BaudRatePrescaler = SPI_BAUDRATEPRESCALER_2;
  hspi2.Init.FirstBit = SPI_FIRSTBIT_MSB;
  hspi2.Init.TIMode = SPI_TIMODE_DISABLE;
  hspi2.Init.CRCCalculation = SPI_CRCCALCULATION_DISABLE;
  hspi2.Init.CRCPolynomial = 7;
  hspi2.Init.CRCLength = SPI_CRC_LENGTH_DATASIZE;
  hspi2.Init.NSSPMode = SPI_NSS_PULSE_ENABLE;
  if (HAL_SPI_Init(&hspi2) != HAL_OK)
  {
    Error_Handler();
  }
```

```
}
static void MX_TIM2_Init(void)
{
  TIM_ClockConfigTypeDef sClockSourceConfig = {0};
  TIM_MasterConfigTypeDef sMasterConfig = {0};
  htim2.Instance = TIM2;
  htim2.Init.Prescaler = 64000-1;
  htim2.Init.CounterMode = TIM_COUNTERMODE_UP;
  htim2.Init.Period = 3000;
  htim2.Init.ClockDivision = TIM_CLOCKDIVISION_DIV1;
  htim2.Init.AutoReloadPreload = TIM_AUTORELOAD_PRELOAD_DISABLE;
}

static void MX_GPIO_Init(void)
{
  GPIO_InitTypeDef GPIO_InitStruct = {0};

  /* GPIO Ports Clock Enable */
  __HAL_RCC_GPIOC_CLK_ENABLE();
  __HAL_RCC_GPIOB_CLK_ENABLE();
  __HAL_RCC_GPIOA_CLK_ENABLE();

  /*Configure GPIO pin : PC3 */
  GPIO_InitStruct.Pin = GPIO_PIN_3;
  GPIO_InitStruct.Mode = GPIO_MODE_IT_RISING;
  GPIO_InitStruct.Pull = GPIO_PULLUP;
  HAL_GPIO_Init(GPIOC, &GPIO_InitStruct);

}

void Error_Handler(void)
{
  __disable_irq();
  while (1)
  {
  }
}
```

библиотека функций ads1293.c

```
#include <stdint.h>
#include "stdlib.h"
#include "ads1293.h"
#include "main.h"
SPI_HandleTypeDef ads_hspi;

void cs_low_ads(){
        HAL_GPIO_WritePin(GPIOA, CS_ADS_Pin, GPIO_PIN_RESET); //CS установлен в ноль
        };
void cs_high_ads(){
                HAL_GPIO_WritePin(GPIOA, CS_ADS_Pin, GPIO_PIN_SET); //CS установлен в единицу
                };


void ADS1293_write(uint8_t address, uint8_t value){
      uint8_t data[2];
      uint8_t command_field = address & ADS1293_WRITE_BIT;//формируем command field
      data[0] = command_field;
      data[1] = value; //байт, который запишем в регистр
```

```
	cs_low_ads();
	HAL_SPI_Transmit(&ads_hspi, data, 2, 100);
	//передача двух байт, первый - адрес регистра,
	//второй - то, что хотим записать
	cs_high_ads();
	HAL_Delay(10);
};
uint8_t ADS1293_read(uint8_t address){
	uint8_t command_field = address | ADS1293_READ_BIT;
	uint8_t read_val;
	cs_low_ads();
	HAL_SPI_Transmit(&ads_hspi, &command_field, 1, 100);
	HAL_SPI_Receive(&ads_hspi, &read_val, 1, 100);
	cs_high_ads();
	return read_val;
};

void ADS1293_INIT(){
	//power-down - выкл, stand-by - выкл, start conversion выкл
	ADS1293_write(ADS1293_CONFIG_REG,0b000);
	//POS1=IN2, NEG1=IN1
	ADS1293_write(ADS1293_FLEX_CH1_CN_REG,0b00010001);
	//POS2=IN3, NEG2=IN1
	ADS1293_write(ADS1293_FLEX_CH2_CN_REG,0b00011001);
	// POS3=IN5, NEG3=IN6
	ADS1293_write(ADS1293_FLEX_CH3_CN_REG,0b00101110);
	//common-mode detector -вкл на IN1,IN2,IN3
	ADS1293_write(ADS1293_CMDET_EN_REG,0b00000111);
	//подкл IN4 к RLD control
	ADS1293_write(ADS1293_RLD_CN_REG,0b0000100);
	//подключаем Wilson control
	ADS1293_write(ADS1293_WILSON_EN1_REG,0b001);
	ADS1293_write(ADS1293_WILSON_EN2_REG,0b010);
	ADS1293_write(ADS1293_WILSON_EN3_REG,0b011);
	ADS1293_write(ADS1293_WILSON_CN_REG,0b01);

	//используем внешний кварцевый резонатор
	ADS1293_write(ADS1293_OSC_CN_REG,0b100);


	//настраиваем цифровой фильтр
	ADS1293_write(ADS1293_R2_RATE_REG,0b1000);

	ADS1293_write(ADS1293_R3_RATE1_REG,0b100);
	ADS1293_write(ADS1293_R3_RATE2_REG,0b100);
	ADS1293_write(ADS1293_R3_RATE3_REG,0b100);

	//data ready bar на channel 1 ECG
	ADS1293_write(ADS1293_DRDYB_SRC_REG,0b1000);

	//вкл чтение с CH1 CH2 CH3, вкл status data loop read-back
	ADS1293_write(ADS1293_CH_CNFG_REG,0b1110001);

};

void ADS1293_start_conv(){
	ADS1293_write(ADS1293_CH_CNFG_REG,0b001);
}

void ADS1293_stop_conv(){
```

```
    ADS1293_write(ADS1293_CH_CNFG_REG,0b000);
}


void ADS1293_stream_read(uint32_t* databuffer){

    uint32_t result = 0x000000000000000000000000;
     ////////// Read lead I
     result=ADS1293_read(ADS1293_DATA_CH1_ECG_H_REG);
     result=result<<8;
     result|=ADS1293_read(ADS1293_DATA_CH1_ECG_M_REG);
     result=result<<8;
     result|=ADS1293_read(ADS1293_DATA_CH1_ECG_L_REG);
     databuffer[0]=result;

     ////////// Read lead II
     result=ADS1293_read(ADS1293_DATA_CH2_ECG_H_REG);
     result=result<<8;
     result|=ADS1293_read(ADS1293_DATA_CH2_ECG_M_REG);
     result=result<<8;
     result|=ADS1293_read(ADS1293_DATA_CH2_ECG_L_REG);
     databuffer[1]=result;

     //////Read lead V1
     result=ADS1293_read(ADS1293_DATA_CH3_ECG_H_REG);
     result=result<<8;
     result|=ADS1293_read(ADS1293_DATA_CH2_ECG_M_REG);
     result=result<<8;
     result|=ADS1293_read(ADS1293_DATA_CH2_ECG_L_REG);
     databuffer[2]=result;


}
```

заголовочный файл ads1293.c

```
#include "stm32wbxx_hal.h"
#ifndef SRC_ADS1293_H_
#define SRC_ADS1293_H_



#define ADS1293_CONFIG_REG            (0x00)  /* Main Configuration */

#define ADS1293_FLEX_CH1_CN_REG       (0x01)  /* Flex Routing Swich Control for Channel 1 */
#define ADS1293_FLEX_CH2_CN_REG       (0x02)  /* Flex Routing Swich Control for Channel 2 */
#define ADS1293_FLEX_CH3_CN_REG       (0x03)  /* Flex Routing Swich Control for Channel 3 */
#define ADS1293_FLEX_PACE_CN_REG      (0x04)  /* Flex Routing Swich Control for Pace Channel */
#define ADS1293_FLEX_VBAT_CN_REG      (0x05)  /* Flex Routing Swich Control for Battery Monitoriing */

#define ADS1293_LOD_CN_REG            (0x06)  /* Lead Off Detect Control */
#define ADS1293_LOD_EN_REG            (0x07)  /* Lead Off Detect Enable */
#define ADS1293_LOD_CURRENT_REG       (0x08)  /* Lead Off Detect Current */
#define ADS1293_LOD_AC_CN_REG         (0x09)  /* AC Lead Off Detect Current */

#define ADS1293_CMDET_EN_REG          (0x0A)  /* Common Mode Detect Enable */
#define ADS1293_CMDET_CN_REG          (0x0B)  /* Commond Mode Detect Control */
```

```
#define ADS1293_RLD_CN_REG          (0x0C)  /* Right Leg Drive Control */

#define ADS1293_WILSON_EN1_REG      (0x0D)  /* Wilson Reference Input one Selection */
#define ADS1293_WILSON_EN2_REG      (0x0E)  /* Wilson Reference Input two Selection */
#define ADS1293_WILSON_EN3_REG      (0x0F)  /* Wilson Reference Input three Selection */
#define ADS1293_WILSON_CN_REG       (0x10)  /* Wilson Reference Input Control */

#define ADS1293_REF_CN_REG          (0x11)  /* Internal Reference Voltage Control */

#define ADS1293_OSC_CN_REG          (0x12)  /* Clock Source and Output Clock Control */

#define ADS1293_AFE_RES_REG         (0x13)  /* Analog Front-End Frequency and Resolution */
#define ADS1293_AFE_SHDN_CN_REG     (0x14)  /* Analog Front-End Shutdown Control */
#define ADS1293_AFE_FAULT_CN_REG    (0x15)  /* Analog Front-End Fault Detection Control */
#define ADS1293_AFE_DITHER_EN_REG   (0x16)  /* Enable Dithering in Signma-Delta */
#define ADS1293_AFE_PACE_CN_REG     (0x17)  /* Analog Pace Channel Output Routing Control */

#define ADS1293_ERROR_LOD_REG       (0x18)  /* Lead Off Detect Error Status */
#define ADS1293_ERROR_STATUS_REG    (0x19)  /* Other Error Status */
#define ADS1293_ERROR_RANGE1_REG    (0x1A)  /* Channel 1 Amplifier Out of Range Status */
#define ADS1293_ERROR_RANGE2_REG    (0x1B)  /* Channel 1 Amplifier Out of Range Status */
#define ADS1293_ERROR_RANGE3_REG    (0x1C)  /* Channel 1 Amplifier Out of Range Status */
#define ADS1293_ERROR_SYNC_REG      (0x1D)  /* Synchronization Error */

#define ADS1293_R2_RATE_REG         (0x21)  /* R2 Decimation Rate */
#define ADS1293_R3_RATE1_REG        (0x22)  /* R3 Decimation Rate for Channel 1 */
#define ADS1293_R3_RATE2_REG        (0x23)  /* R3 Decimation Rate for Channel 2 */
#define ADS1293_R3_RATE3_REG        (0x24)  /* R3 Decimation Rate for Channel 3 */
#define ADS1293_P_DRATE_REG         (0x25)  /* 2x Pace Data Rate */
#define ADS1293_DIS_EFILTER_REG     (0x26)  /* ECG Filter Disable */
#define ADS1293_DRDYB_SRC_REG       (0x27)  /* Data Ready Pin Source */
#define ADS1293_SYNCOUTB_SRC_REG    (0x28)  /* Sync Out Pin Source */
#define ADS1293_MASK_DRDYB_REG      (0x29)  /* Optional Mask Control for DRDYB Output */
#define ADS1293_MASK_ERR_REG        (0x2A)  /* Mask Error on ALARMB Pin */

#define ADS1293_ALARM_FILTER_REG    (0x2E)  /* Digital Filter for Analog Alarm Signals */
#define ADS1293_CH_CNFG_REG         (0x2F)  /* Configure Channel for Loop Read Back Mode */

#define ADS1293_DATA_STATUS_REG     (0x30)  /* ECG and Pace Data Ready Status */
#define ADS1293_DATA_CH1_PACE_H_REG (0x31)  /* Channel1 Pace Data High [15:8] */
#define ADS1293_DATA_CH1_PACE_L_REG (0x32)  /* Channel1 Pace Data Low [7:0] */

#define ADS1293_DATA_CH2_PACE_H_REG (0x33)  /* Channel2 Pace Data High [15:8] */
#define ADS1293_DATA_CH2_PACE_L_REG (0x34)  /* Channel2 Pace Data Low [7:0] */

#define ADS1293_DATA_CH3_PACE_H_REG (0x35)  /* Channel3 Pace Data High [15:8] */
#define ADS1293_DATA_CH3_PACE_L_REG (0x36)  /* Channel3 Pace Data Low [7:0] */

#define ADS1293_DATA_CH1_ECG_H_REG  (0x37)  /* Channel1 ECG Data High [23:16] */
#define ADS1293_DATA_CH1_ECG_M_REG  (0x38)  /* Channel1 ECG Data Medium [15:8] */
#define ADS1293_DATA_CH1_ECG_L_REG  (0x39)  /* Channel1 ECG Data Low [7:0] */

#define ADS1293_DATA_CH2_ECG_H_REG  (0x3A)  /* Channel2 ECG Data High [23:16] */
#define ADS1293_DATA_CH2_ECG_M_REG  (0x3B)  /* Channel2 ECG Data Medium [15:8] */
#define ADS1293_DATA_CH2_ECG_L_REG  (0x3C)  /* Channel2 ECG Data Low [7:0] */

#define ADS1293_DATA_CH3_ECG_H_REG  (0x3D)  /* Channel3 ECG Data High [23:16] */
#define ADS1293_DATA_CH3_ECG_M_REG  (0x3E)  /* Channel3 ECG Data Medium [15:8] */
#define ADS1293_DATA_CH3_ECG_L_REG  (0x3F)  /* Channel3 ECG Data Low [7:0] */
```

```
#define ADS1293_REVID_REG               (0x40)  /* Revision ID */
#define ADS1293_DATA_LOOP_REG           (0x50)  /* Loop Read Back Address */


#define ADS1293_READ_BIT                (0x80)
#define ADS1293_WRITE_BIT               (0x7F)




void cs_low_ads();
void cs_high_ads();
void ADS1293_write(uint8_t address, uint8_t value);
uint8_t ADS1293_read(uint8_t address);
void ADS1293_INIT();
void ADS1293_start_conv();
void ADS1293_stop_conv();
void ADS1293_stream_read(unsigned long int* databuffer);




#endif /* SRC_ADS1293_H_ */
```