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

”МОНИТОР АКТИВНОСТИ И ОТСЛЕЖИВАНИЯ ПАДЕНИЯ”

Выполнил студент группы 6364-120304D: _____ Краснов Д.Г.

Руководитель проекта: _____ Корнилин Д.В.

Работа защищена с оценкой: _____

Самара 2023

ЗАДАНИЕ

Разработать монитор активности и отслеживания падений со следующими параметрами:

– Датчик падений/движения/активности

– Диапазон регистрируемых ускорений от 2g до 8g;

– Частота обновления показаний 400 Гц;

– Передача данных по интерфейсу Bluetooth;

– Питание батарейное.

РЕФЕРАТ

Пояснительная записка: 42 страниц, 26 рисунков, источников, 1 приложение.

МОНИТОР АКТИВНОСТИ И ОТСЛЕЖИВАНИЯ ПАДЕНИЙ, МИКРОКОНТРОЛЛЕР, BLUETOOTH, АКСЕЛЕРОМЕТР, STM32WB, АЛГОРИТМ

В курсовом проекте разработаны структурная и принципиальная схемы монитора активности и отслеживания падений с датчиком на базе акселерометра, осуществлен выбор микроконтроллера с интегрированным блоком Bluetooth. Разработан алгоритм анализа данных и программа на языке Си, реализующая его.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Падения являются достаточно распространенной проблемой среди людей, в особенности пожилых, которая наносит существенный вред здоровью.

Падения являются серьезной проблемой общественного здравоохранения для пожилых людей во всем мире. Отчеты Всемирной организации здравоохранения показывают, что примерно 28-35% пожилых людей старше 65 лет страдают по крайней мере от одного падения в год, что приводит к травмам мышц или связок, переломам костей и травмам головы. Решением данной проблемы являются носимые детекторы падения. Носимые устройства позволяют осуществлять непрерывный мониторинг независимо от датчиков окружающей среды, что делает их повсеместными системами, которые собирают только пользовательские данные, способствуя расширению возможных сценариев использования. Кроме того, они используют простые датчики (акселерометры и гироскопы) с низким энергопотреблением.

В данном курсовом проекте рассматривается способ создания устройства на базе микроконтроллера, который сможет отслеживать активность и падения человека. В процессе были подобраны необходимые в задании микроконтроллер с интегрированным модулем Bluetooth, акселерометр, а также написана управляющая программа на языке Си.

1 РАЗРАБОТКА СТРУКТУРНОЙ СХЕМЫ УСТРОЙСТВА

Структурная схема устройства представлена на рисунке 1.

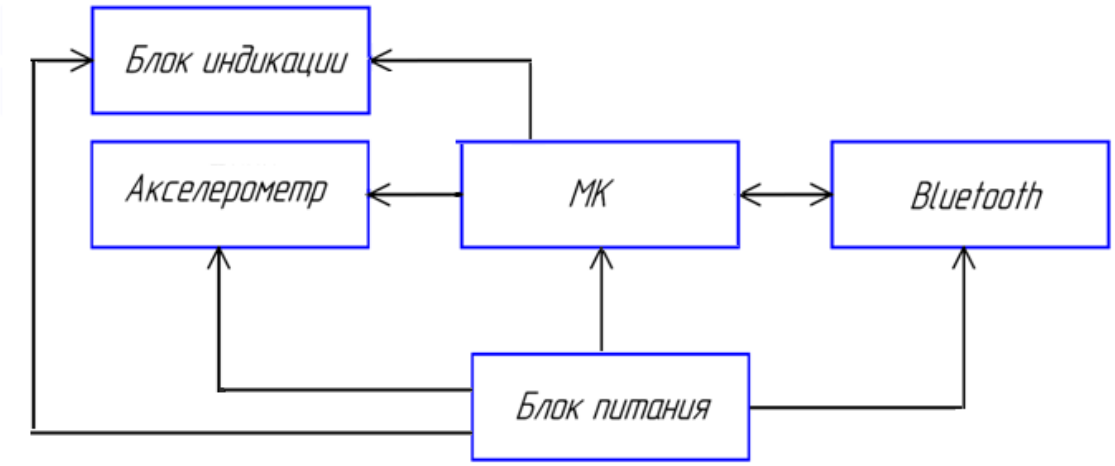


Рисунок 1 – Структурная схема устройства

Принцип работы устройства заключается в следующем. Трёхосевой акселерометр фиксирует ускорение по каждой из осей движения. Эти данные поступают в микроконтроллер, где проходят первичную обработку, и с помощью алгоритма на языке Си анализируются. В результате анализа можно выяснить характер движения человека, и то, происходит ли падение.

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

Все элементы схемы питаются от блока питания, который представляет собой литиевый аккумулятор, имеющий номинальное напряжение 3.7 В, и DC-DC преобразователя, который необходим для стабилизации напряжения на уровне 3.3 В, необходимого всем элементам устройства.

2 РАЗРАБОТКА ПРИНЦИПИАЛЬНОЙ СХЕМЫ УСТРОЙСТВА

Электрическая принципиальная схема представлена в приложении.

2.1 Выбор акселерометра

Стоит выяснить, как работают и устроены акселерометры. [2] Это датчики движения, входным сигналом которых является скорость и ускорение объекта. Отличительной особенностью данных устройств является их компактность и стоимость за счет налаженного производства микроэлектромеханических систем (МЭМС).

Основное применение датчики движения нашли в промышленности, а именно в авиации для определения положения летающего аппарата в пространстве и в строительстве. В медицине датчики движения используются редко, однако некоторые методики включают использование акселерометров.

Современные МЭМ акселерометры разделяют по физическому принципу детектирования ускорений, однако широкое распространение получили только 3 вида:

– Пьезоэлектрические, основой которых является пъезокристалл. Деформации кристалл приводят к появлению на нем разности потенциалов. Такие акселерометры имеют широкий диапазон частот и выдерживают значительные нагрузки. Однако пьезоэффект возникает только в момент деформации, что не позволяет измерять статические ускорения наподобие гравитационного. Также пьезоэлектрические акселерометры из-за значительного сопротивления пьезокристалла и малой разности потенциалов при деформации требуют высокоомного соединения со схемой.

– Пьезорезистивные своими характеристиками не сильно отличаются от ПЭА, имея столь же малую термостабильность и стабильность смещения. Однако получение полезного электрического сигнала происходит на

мостовой схеме с пьезорезистивными элементами, при этом нет необходимости использования высокоомного подключения. Также присутствует возможность самотестирования акселерометров и измерения статических нагрузок

– Емкостные – самый распространенный вид МЭМ акселерометров. Принцип действия заключается в измерении реакции измерительной ячейки, состоящей из сложного конденсатора с переменной емкостью на зондирующий сигнал. При измерении ускорения инерционная масса двигает нестатичную обкладку конденсатора, вследствие чего меняется емкость. При этом емкостные конденсаторы не имеют проблем, связанных с природой пьезоэффекта, а именно имеют конструкторскую легкость при подключении в цепь и возможность самотестирования. Также основными преимуществами является высокая термостабильность. Недостатком можно назвать сложность конструкции, однако при налаженном производстве это фактор не оказывает значительного влияния.

Таким образом, современные малогабаритные измерительные модули целесообразно конструировать с емкостными акселерометрами, за счет их стабильности отсутствия требований в схемах высокоомного подключения.

Согласно техническому заданию нам необходим акселерометр с диапазоном регистрируемых ускорений от 2g до 8g и возможностью выдачи показаний с частотой 400 Гц. Данным требованиям соответствует 3-осевой акселерометр ADXL345 [1], его основные характеристики представлены ниже.

– Тип датчика: цифровой, емкостной;

– Диапазон регистрируемых ускорений ±2g, ±4g, ±8g, ±16g;

– Частота обновления показаний: задается пользователем в диапазоне 0.1-3200 Гц;

– Сверхнизкое потребление: 23 мкА в режиме преобразования и 0.1 мкА в режиме ожидания;

– напряжение питания: 2-3.6В;

– Интерфейс цифрового вывода: I^2C, SPI;

– Разрядность: настраиваемая пользователем – 10 бит в диапазоне ±2g, 13 бит в остальных диапазонах.

Структурная схема акселерометра из даташита ADXL345 приведена на рисунке 2.

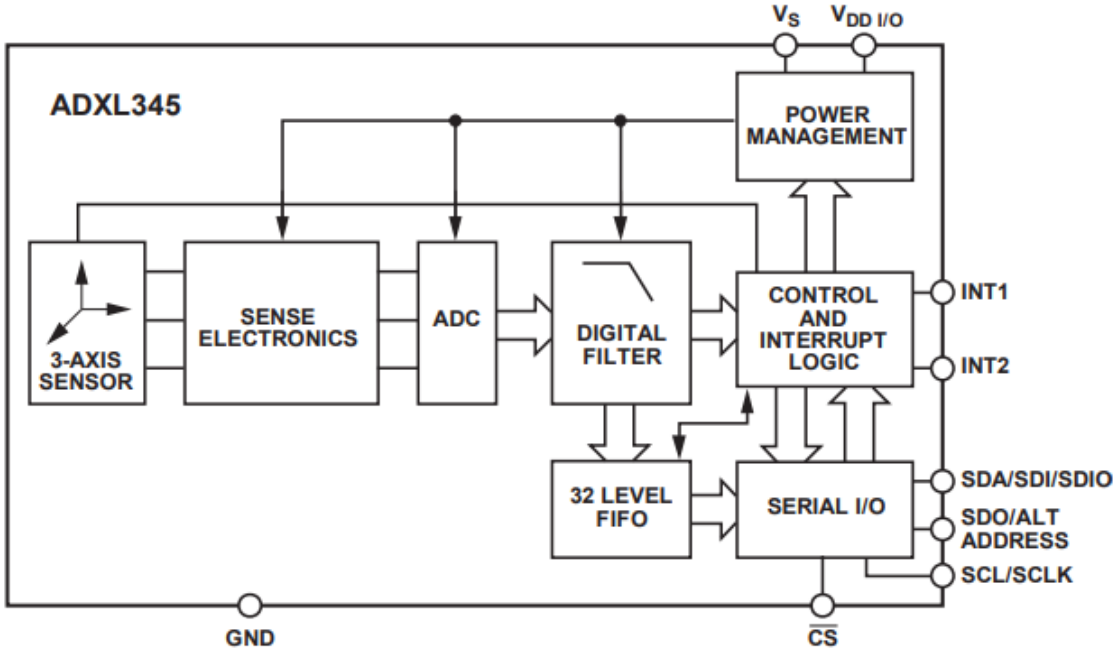
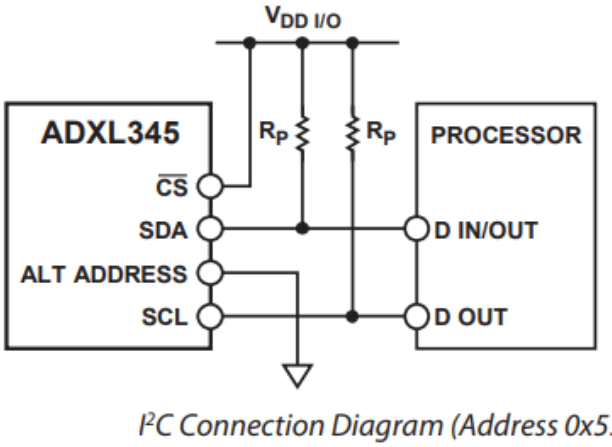


Рисунок 2 – Структурная схема акселерометра

Видно, что устройство состоит из 3-осевого ”сенсора”, представляющего собой несколько конденсаторов с нестатичными обкладками, ”чувствительной электроники”, аналого-цифрового преобразователя, цифрового фильтра, буфера FIFO для временного хранения результатов преобразования, контроллера питания и логического устройства, контролирующего работу акселерометра и логику прерываний. Устройство содержит выводы данных, соответствующие интерфейсам I^2C и SPI. Для связи с акселерометром мы будем использовать I^2C. Схема подключения представлена на рисунке 3.



I²C Connection Diagram (Address 0x53)

Рисунок 3 – Схема подключения акселерометра к микроконтроллеру по I^2C

Как видно из рисунка 3, для активации интерфейса I^2C необходимо подтянуть вывод $\overline{CS}$ к питанию.

Так же, в даташите приведена рекомендованная для минимизации шумов схема включения акселерометра(рисунок 4).

POWER SUPPLY DECOUPLING

A 1 µF tantalum capacitor (C_S) at V_S and a 0.1 µF ceramic capacitor ($C_{I/O}$) at $V_{DD\ I/O}$ placed close to the ADXL345 supply pins is recommended to adequately decouple the accelerometer from noise on the power supply. If additional decoupling is necessary, a resistor or ferrite bead, no larger than 100 Ω, in series with V_S may be helpful. Additionally, increasing the bypass capacitance on V_S to a 10 µF tantalum capacitor in parallel with a 0.1 µF ceramic capacitor may also improve noise.

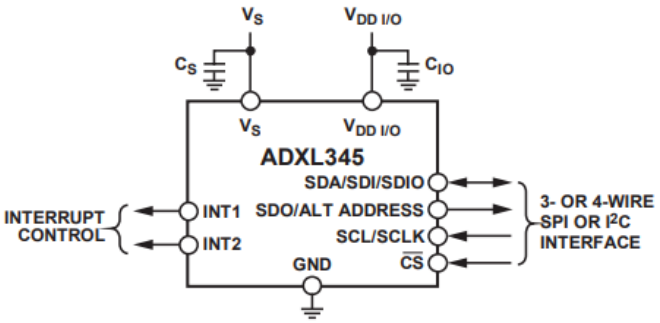


Рисунок 4 – Типовая схема включения акселерометра

Нумерация и назначение выводов ADXL345 приведено ниже (рисунки 5, 6).

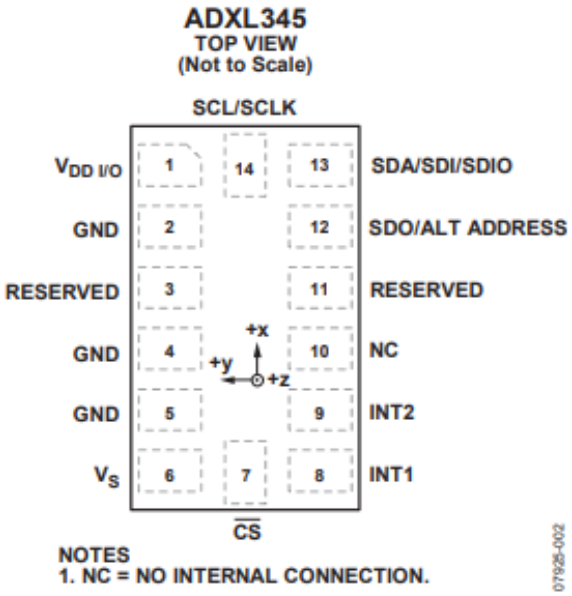


Figure 3. Pin Configuration (Top View)

Рисунок 5 – Нумерация выводов

Table 5. Pin Function Descriptions

Pin No.	Mnemonic	Description
1	$V_{DD\ I/O}$	Digital Interface Supply Voltage.
2	GND	This pin must be connected to ground.
3	RESERVED	Reserved. This pin must be connected to V_S or left open.
4	GND	This pin must be connected to ground.
5	GND	This pin must be connected to ground.
6	V_S	Supply Voltage.
7	$\overline{CS}$	Chip Select.
8	INT1	Interrupt 1 Output.
9	INT2	Interrupt 2 Output.
10	NC	Not Internally Connected.
11	RESERVED	Reserved. This pin must be connected to ground or left open.
12	SDO/ALT ADDRESS	Serial Data Output (SPI 4-Wire)/Alternate I^2C Address Select (I^2C).
13	SDA/SDI/SDIO	Serial Data (I^2C)/Serial Data Input (SPI 4-Wire)/Serial Data Input and Output (SPI 3-Wire).
14	SCL/SCLK	Serial Communications Clock. SCL is the clock for I^2C, and SCLK is the clock for SPI.

Рисунок 6 – Назначение выводов

2.2 Выбор микроконтроллера

С учетом технического задания микроконтроллер должен обладать следующими свойствами:

– Интерфейс для работы с микросхемой акселерометра: SPI или I^2C;

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

– Малое энергопотребление;

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

Для решения задачи был выбран микроконтроллер STM32WB35CCU6A фирмы ST Microelectronics [3].STM32WB35 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 I^2C,необходимый для подключения к акселерометру, и радиомодуль с поддержкой Bluetooth, диапазон питающего напряжения от 2 до 3,6 В. 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 8

Figure 2. STM32WB35xx block diagram

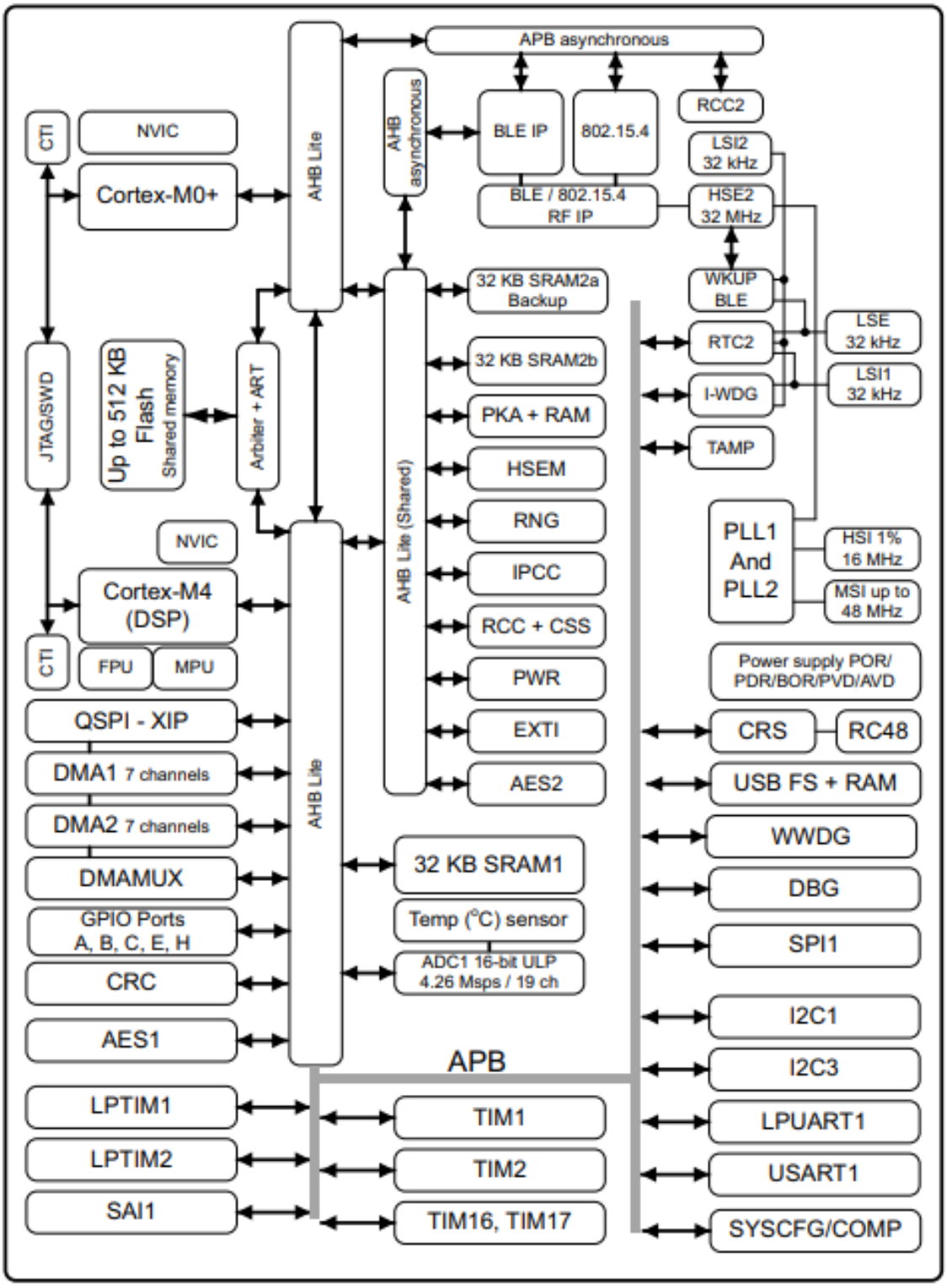


Рисунок 7 – Структурная схема

Figure 10. STM32WB55Cx and STM32WB35Cx UFQFPN48 pinout(1)(2)

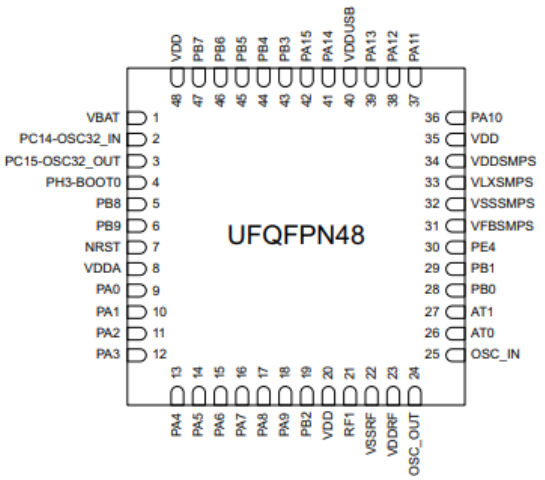


Рисунок 8 – Назначение выводов

Подключение будет осуществляться согласно типовой схеме из Application note [4](рисунок 9)

Figure 8. UFQFPN48 reference board

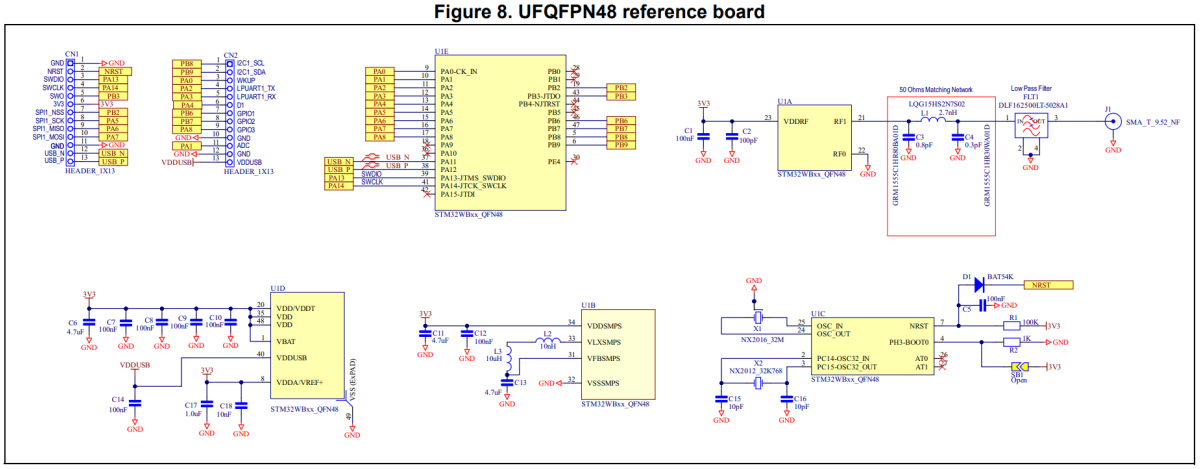


Рисунок 9 – Типовая схема подключения STM32WB35

2.3 Блок питания

Блок питания будет состоять из аккумулятора LP-130-232635 [5] и DC-DC преобразователя LM3671 [6]. Аккумулятор литий-полимерный LP-130-232635 имеет номинальную емкость 130 мАч, номинальное напряжение 3,7 В, вес 3г. Длина: 35±1 мм. Ширина: 26±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 [6] (рисунок 10)

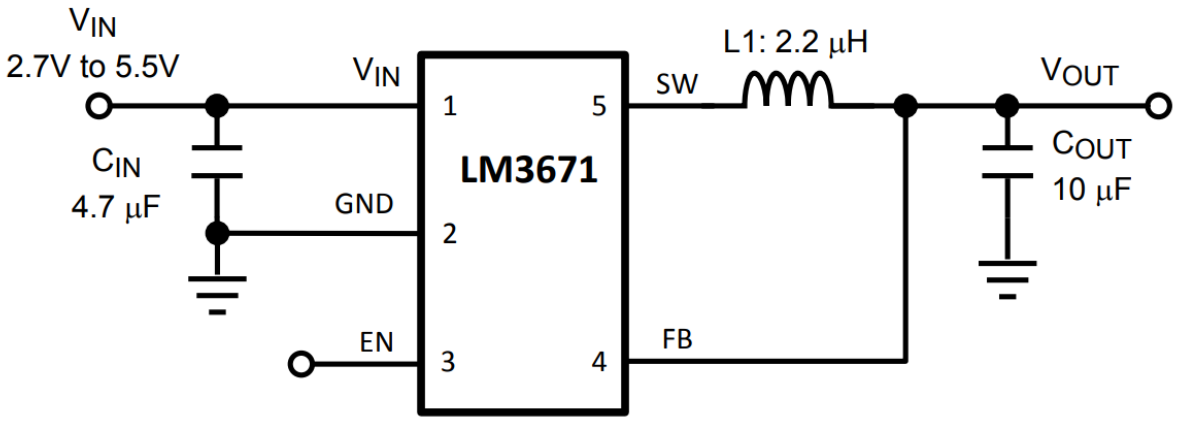


Рисунок 10 – Типовая схема включения DC-DC–преобразователя

3 РАЗРАБОТКА ПРОГРАММЫ

3.1 Разработка алгоритма

Сначала нужно понять, что необходимо для выполнения задач, поставленных в задании. Нужно проинициализировать необходимую переферию и настроить их в нужный режим работы, для чего нужно тщательно изучить даташиты на все компоненты.

Требуется обновлять данные с частотой 400Гц – для этого нужен таймер. Нужно каким-то образом обрабатывать полученные данные и отсылать результат по Bluetooth.

Для работы программы необходимо для начала разработать алгоритм. Алгоритм нашего устройства представлен на рисунке 11.

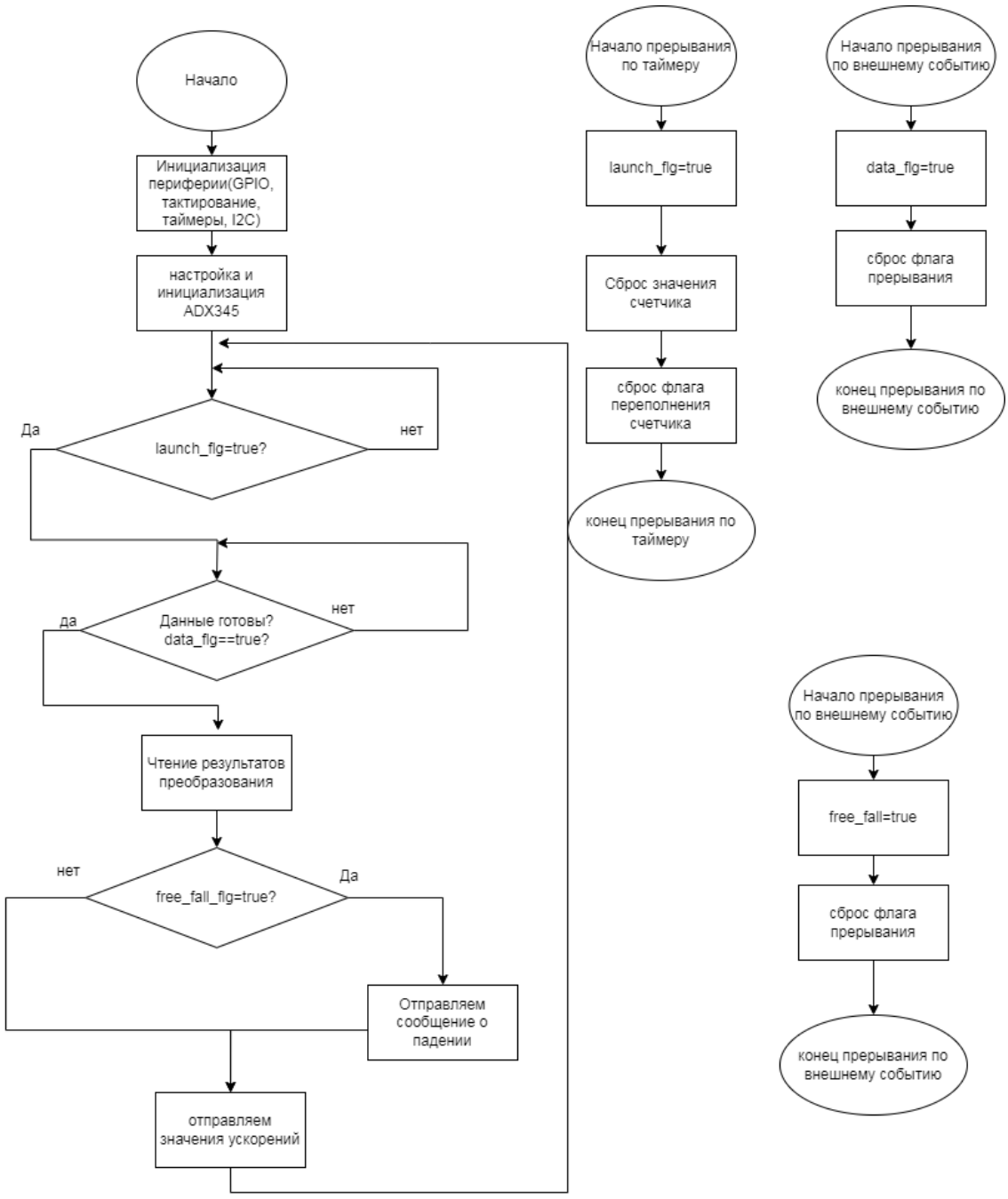


Рисунок 11 – Алгоритм работы устройства

Главное тело программы работает так - инициализирует всю необходимую переферию микроконтроллера, после чего проверяет launch_flg - флаг, который поднимается в прерывании от таймера каждые 2.5мс, что соответствует частоте обновления данных в 400Гц.

Так же есть три прерывания – по переполнению счетчика таймера, и два внешних - по изменению уровня на выводах акселерометра INT0 INT1, подключенных к выводам микроконтроллера PA4 и PA5.

Если флаг запуска launch_flg==true, то ждем, когда установится флаг data_flg(который устанавливается по готовности данных в акселерометре). После этого проверяем флаг свободного падения free_fall_flg==true. Если он активен - то помимо данных об ускорении пишем еще и сообщение о том, что произошло падение. Если free_fall_flg==false, то просто передаем данные по ускорению. Таким образом, для определения характера движения была использована гибкая система прерываний ADXL345. Подробнее о ней в следующем разделе.

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 переферии

В STM32CubeIDE встроен STM32CubeMx – программный продукт, позволяющий легко и непринужденно при помощи достаточно понятного графического интерфейса произвести настройку любой имеющейся на борту микроконтроллера периферии.

Предыстория создания CubeMx такова - ST имеют очень разнообразную линейку микроконтроллеров, тут и Cortex-M0, и Cortex-M0+, и Cortex-M3, и Cortex-M4. Соответственно, встает вопрос о каком-то едином наборе библиотек и едином инструменте для инициализации и конфигурирования всего этого многообразия. Вот для решения этих целей и был выпущен STM32CubeMx.

Суть концепции такова - создаем проект, выбираем микроконтроллер, и нам сразу же предлагается схема со всеми выводами выбранного контроллера. Нажимая на выводы и заходя в разнообразные меню, мы легко настраиваем как периферию, так и режимы работы каждого конкретного вывода. Сразу же очевидные плюсы - можно наглядно увидеть, какие выводы уже заняты, а какие еще свободны (в крупных проектах - более чем полезная функция). Подробнее об этом можно прочитать в [?]

В нашем случае нужно создать проект, выбрать микроконтроллер и подключить переферию - таймер, I^2C, тактирование. Все это настраивается в графическом интерфейсе.

Сначала в настройках Reset and Clock Controller(RCC) подключаем кварцевые резонаторы, как показано на рисунке 12.

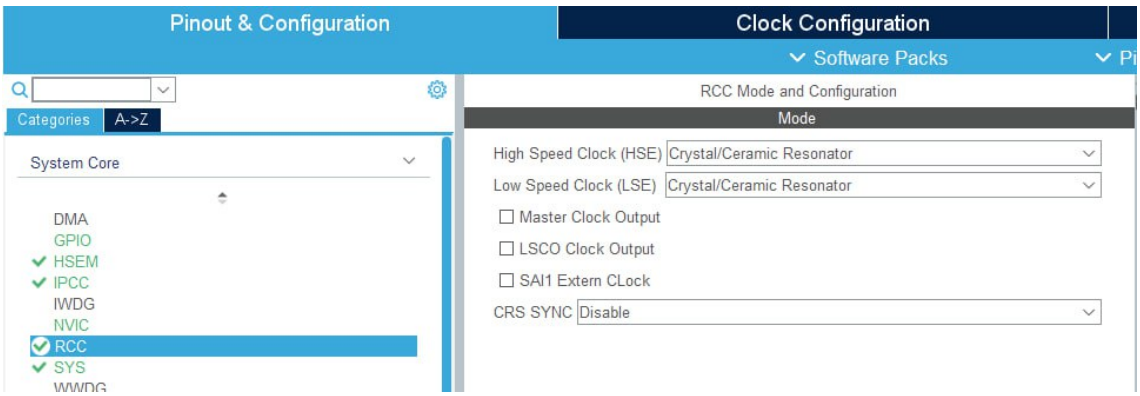


Рисунок 12 – Настройки RCC

Затем подключим интерфейс I^2C, как показано на рисунке 13.

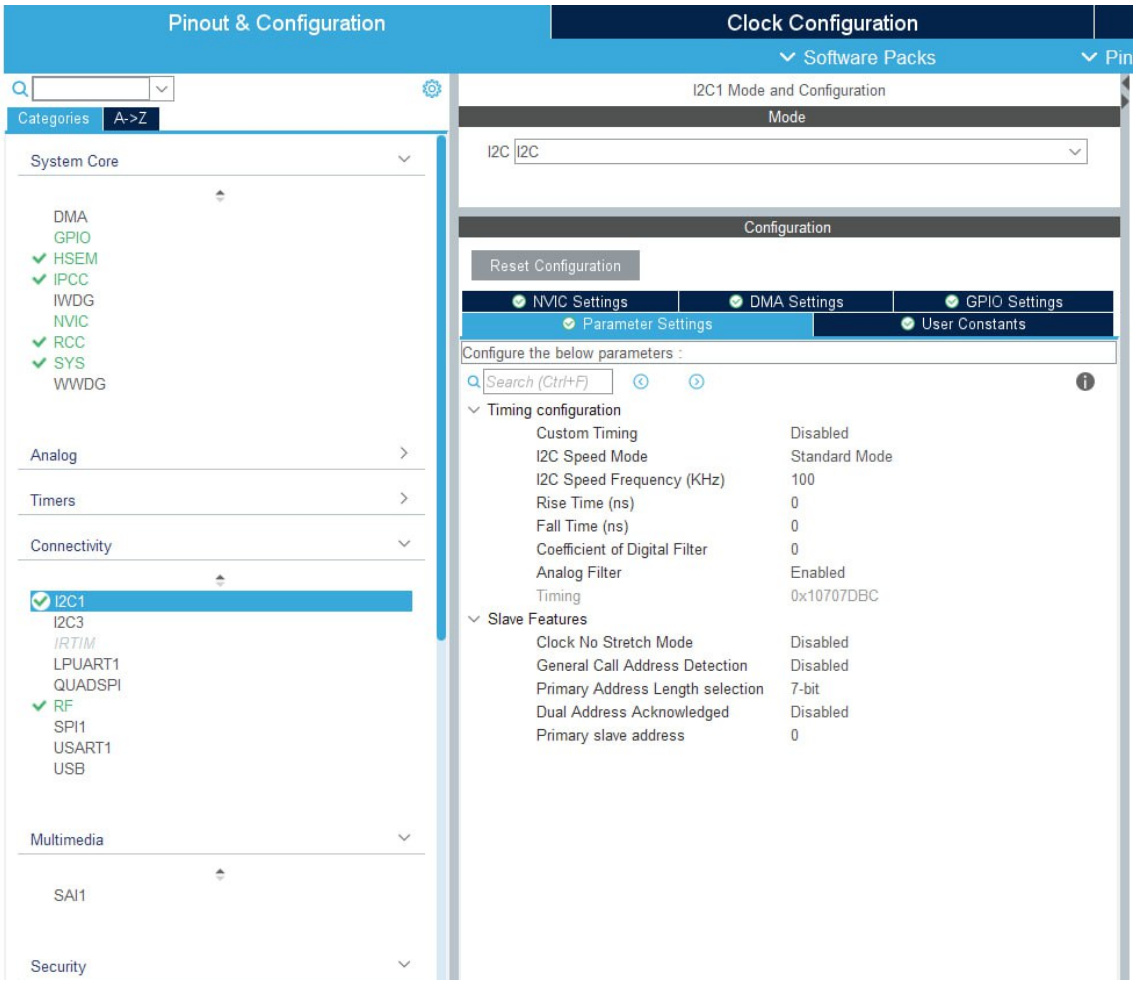


Рисунок 13 – Настройки I^2C

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

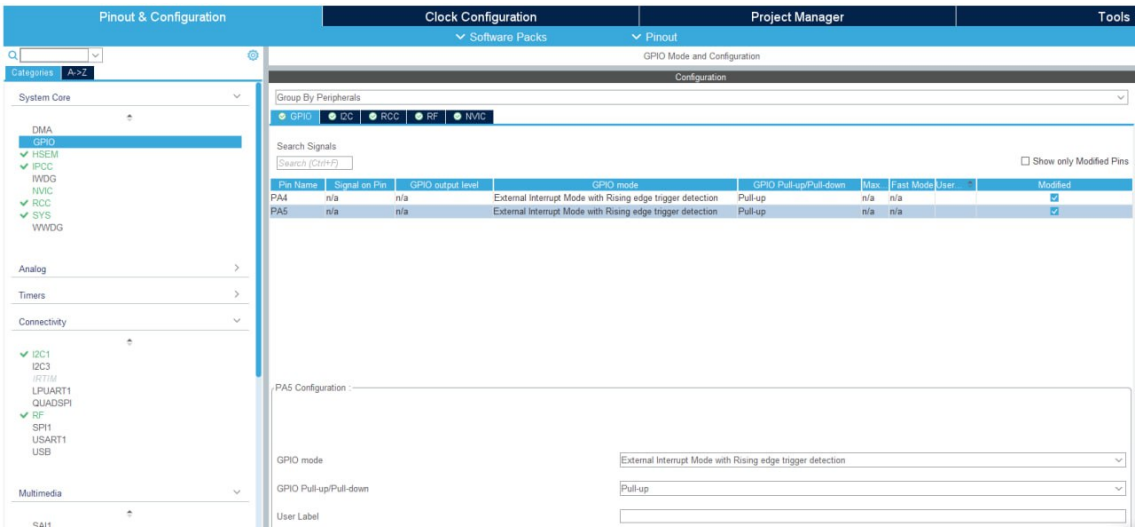


Рисунок 14 – Настройки портов ввода-вывода

После этого можно настроить тактирование на вкладке Clock Configuration, как показано на рисунке 15.

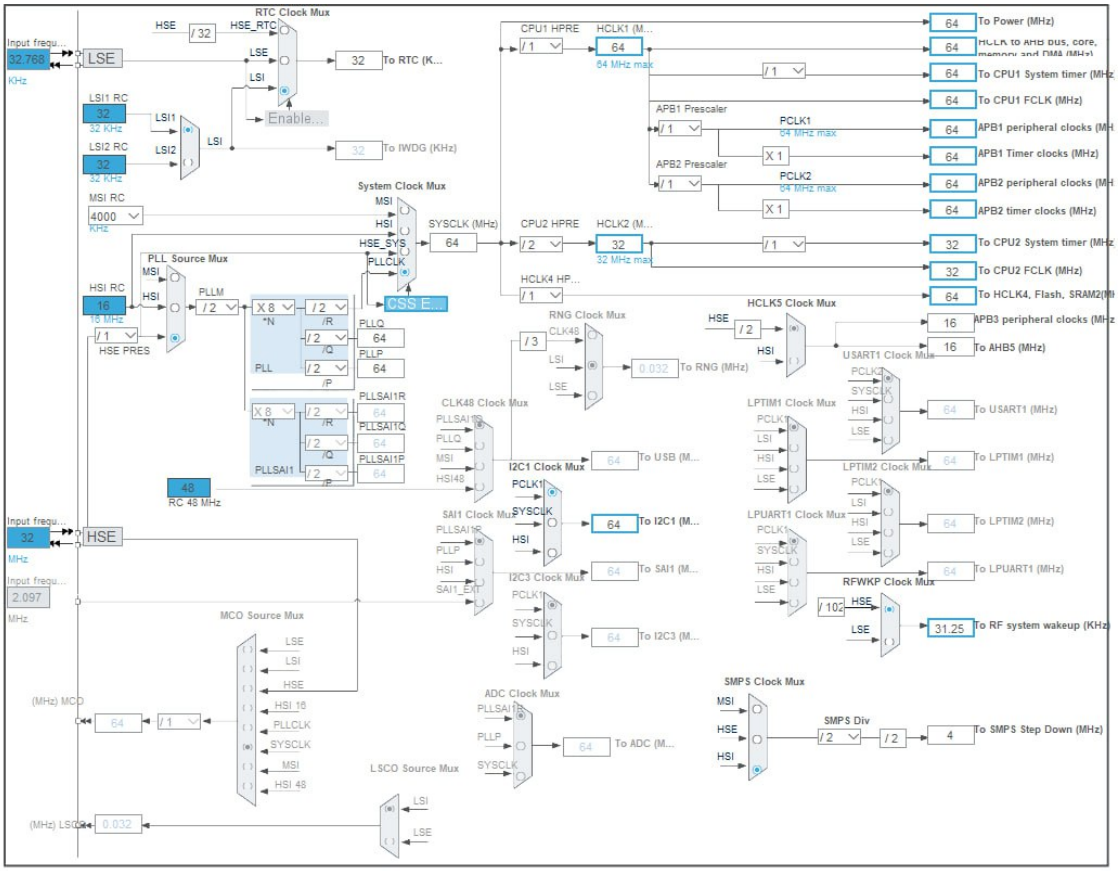


Рисунок 15 – Настройки тактирования

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 16, 17, включить Radio System(RF),как показано на рисунке 18.

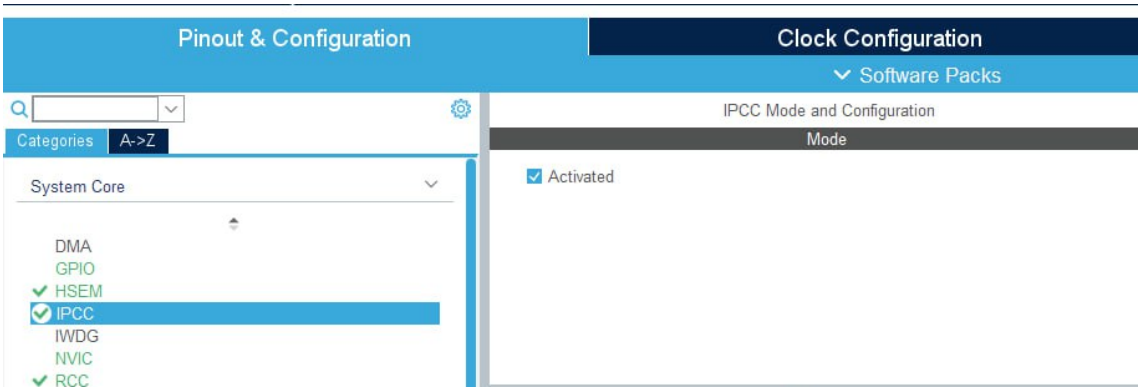


Рисунок 16 – Активируем IPCC

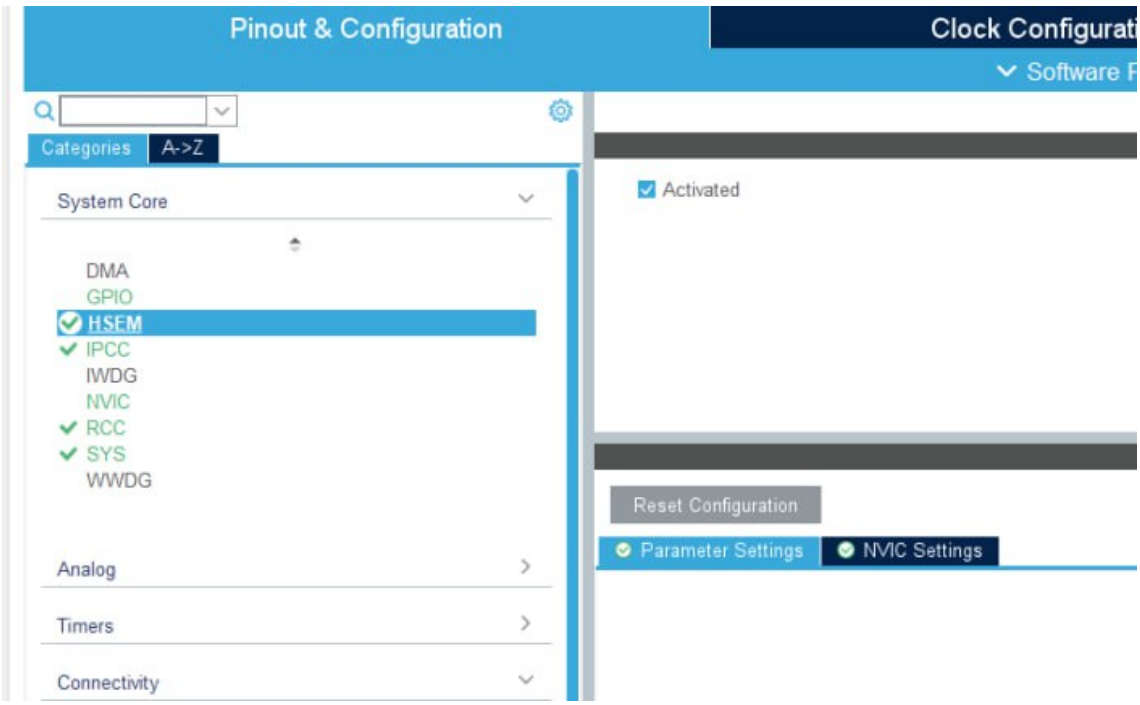


Рисунок 17 – Активируем HSEM

Рисунок 18 – 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 19.

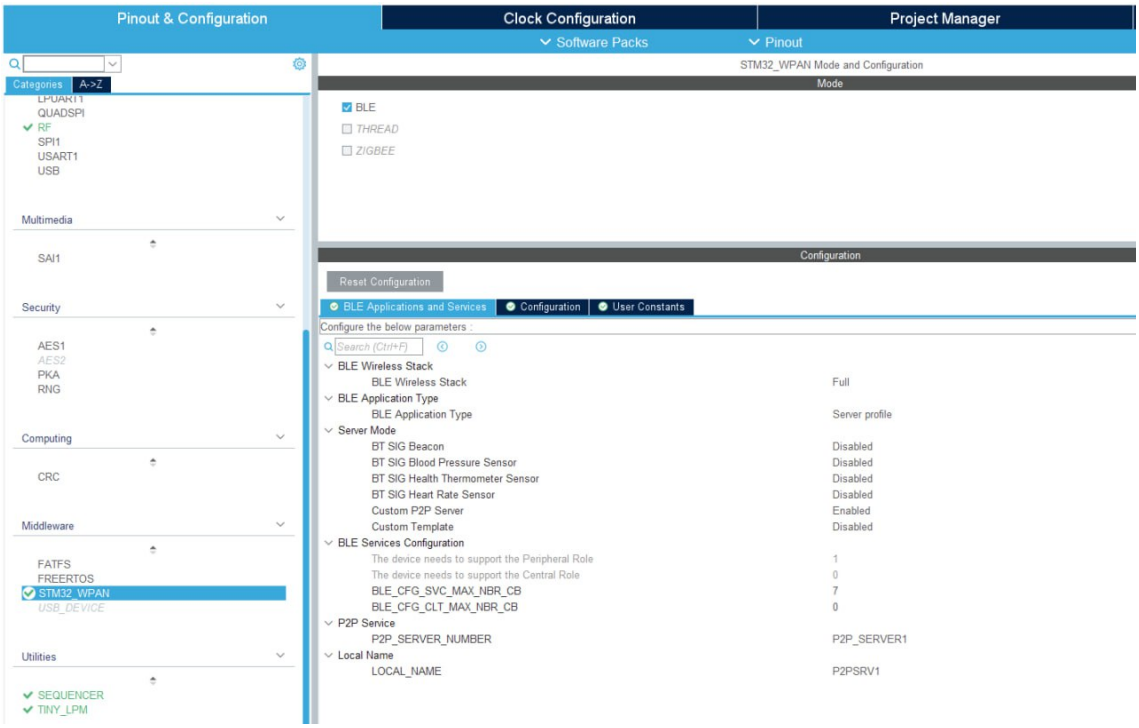


Рисунок 19 – Активируем Bluetooth

3.2.3 Инициализация ADXL345

Карта регистров представлена на рисунке 20.

REGISTER MAP

Table 19.

Address					
Hex	Dec	Name	Type	Reset Value	Description
0x00	0	DEVID	R	11100101	Device ID
0x01 to 0x1C	1 to 28	Reserved			Reserved; do not access
0x1D	29	THRESH_TAP	R/W	00000000	Tap threshold
0x1E	30	OFSX	R/W	00000000	X-axis offset
0x1F	31	OFSY	R/W	00000000	Y-axis offset
0x20	32	OFSZ	R/W	00000000	Z-axis offset
0x21	33	DUR	R/W	00000000	Tap duration
0x22	34	Latent	R/W	00000000	Tap latency
0x23	35	Window	R/W	00000000	Tap window
0x24	36	THRESH_ACT	R/W	00000000	Activity threshold
0x25	37	THRESH_INACT	R/W	00000000	Inactivity threshold
0x26	38	TIME_INACT	R/W	00000000	Inactivity time
0x27	39	ACT_INACT_CTL	R/W	00000000	Axis enable control for activity and inactivity detection
0x28	40	THRESH_FF	R/W	00000000	Free-fall threshold
0x29	41	TIME_FF	R/W	00000000	Free-fall time
0x2A	42	TAP_AXES	R/W	00000000	Axis control for single tap/double tap
0x2B	43	ACT_TAP_STATUS	R	00000000	Source of single tap/double tap
0x2C	44	BW_RATE	R/W	00001010	Data rate and power mode control
0x2D	45	POWER_CTL	R/W	00000000	Power-saving features control
0x2E	46	INT_ENABLE	R/W	00000000	Interrupt enable control
0x2F	47	INT_MAP	R/W	00000000	Interrupt mapping control
0x30	48	INT_SOURCE	R	00000010	Source of interrupts
0x31	49	DATA_FORMAT	R/W	00000000	Data format control
0x32	50	DATAX0	R	00000000	X-Axis Data 0
0x33	51	DATAX1	R	00000000	X-Axis Data 1
0x34	52	DATAY0	R	00000000	Y-Axis Data 0
0x35	53	DATAY1	R	00000000	Y-Axis Data 1
0x36	54	DATAZ0	R	00000000	Z-Axis Data 0
0x37	55	DATAZ1	R	00000000	Z-Axis Data 1
0x38	56	FIFO_CTL	R/W	00000000	FIFO control
0x39	57	FIFO_STATUS	R	00000000	FIFO status

Рисунок 20 – Описания регистров и их адреса

Для удобства перенесем эту карту в код:

```
#define DEVID_ID 0x00
#define THRESH_TAP 0x1D
#define OFSX 0x1E
#define OFSY 0x0F
#define OFSZ 0x20
#define DUR 0x21
#define Latent 0x22
#define Window 0x23
#define THRESH_ACT 0x24
#define THRESH_INACT 0x25
#define TIME_INACT 0x26
#define ACT_INACT_CTL 0x27
#define THRESH_FF 0x28
#define TIME_FF 0x29
#define TAP_AXES 0x2A
#define ACT_TAP_STATUS 0x2B
#define BW_RATE 0x2C
#define POWER_CTL 0x2D
#define INT_ENABLE 0x2E
#define INT_MAP 0x2F
#define INT_SOURCE 0x30
#define DATA_FORMAT 0x31
#define DATAX0 0x32
#define DATAX1 0x33
```

```
#define DATAY0                  0x34
#define DATAY1                  0x35
#define DATAZ0                  0x36
#define DATAZ1                  0x37
#define FIFO_CTL                  0x38
#define FIFO_STATUS             0x39
```

Далее необходимо эти регистры проинициализировать. Связь с акселерометром происходит по I^2C. Значит, перед тем как передать настройки регистров, нужно сформировать необходимый пакет и отправить его на шину I^2C. Для этого объявлена функция adxl_write.

```
void adxl_write(uint8_t address_reg, uint8_t value) {//запись в регистр
    uint8_t data[2];//массив для хранения посылки
    data[0] = address_reg;//сначала передаем адрес регистра в который будем читать
    data[1] = value; //Затем значение которое нужно записываем
    HAL_I2C_Master_Transmit(&hi2c1, adxl_addr, data, 2, timeout)  ;//отправляем массив с адресом и значением
}
```

Так же для инициализации объявлена функция adxl_init.

```
void adxl_init(void) {//инициализация акселерометра и настройка

    adxl_write(DATA_FORMAT, RANGE_8G);  //настраиваем диапазон +- 8g

    adxl_write(POWER_CTL, 0x00);  // выход из режима сна
    adxl_write(POWER_CTL, 0x08);  //включаем преобразование

    adxl_write(THRESH_FF, 0x06);  //настраиваем значение free fall treshold = 62.5mg*8=0.5g
    adxl_write(TIME_FF, 0x02);    //настраиваем время free fall
    //по двум параметрам выше срабатывает прерывание

    adxl_write(INT_ENABLE, FREE_FALL);    //включаем прерывание от free fall
    adxl_write(INT_MAP, FREE_FALL);  //назначаем его на вывод IN1

    adxl_write(INT_ENABLE, DATA_READY); //включаем прерывание по готовности данных
    adxl_write(INT_MAP, DATA_READY);  //назначаем его на вывод IN0

}
```

Разберем строки подробнее.

```
void adxl_init(void) {//инициализация акселерометра и настройка

    adxl_write(DATA_FORMAT, RANGE_8G);  //настраиваем диапазон +- 8g
}
```

Эта строка записывает в регистр DATA_FORMAT значение, соответствующее диапазону +-8g. Возможные диапазоны были так же предварительно объявлены.

```
void adxl_init(void) {
//инициализация акселерометра и настройка
#define RANGE_2G              0x00
#define RANGE_4G              0x01
#define RANGE_8G              0x02
#define RANGE_16G              0x03
}
```

Здесь и далее вся информация из [1], на рисунках 21 и 22 представлены необходимые фрагменты.

Register 0x31—DATA_FORMAT (Read/Write)

D7	D6	D5	D4	D3	D2	D1	D0
SELF_TEST	SPI	INT_INVERT	0	FULL_RES	Justify	Range	

Рисунок 21 – Описание регистра

Table 21. *g* Range Setting

Setting		
D1	D0	*g* Range
0	0	±2 *g*
0	1	±4 *g*
1	0	±8 *g*
1	1	±16 *g*

Рисунок 22 – Возможные значения для настройки

```
adxl_write(POWER_CTL, 0x00);  // выход из режима сна
adxl_write(POWER_CTL, 0x08);  //включаем преобразование
```

Данные две строчки сначала очищают регистр POWER_CTL, затем инициализируют включение режима преобразования. Регистр показан на рисунке 23

Register 0x2D—POWER_CTL (Read/Write)

D7	D6	D5	D4	D3	D2	D1	D0
0	0	Link	AUTO_SLEEP	Measure	Sleep	Wakeup	

Рисунок 23 – регистр POWER_CTL

```
adxl_write(THRESH_FF, 0x06);  //настраиваем значение free fall treshold = 62.5mg*8=0.5g
adxl_write(TIME_FF, 0x14);     //настраиваем время free fall =5ms*30=150ms
//по двум параметрам выше срабатывает прерывание
```

Данные две строчки записывают значения в регистры THRESH_FF и TIME_FF значения ускорения и времени, необходимых для того чтобы сработало прерывание по свободному падению. На рисунке 24 показано описание регистров.

Table 21. *g* Range Setting

Setting		
D1	**D0**	***g* Range**
0	0	±2 *g*
0	1	±4 *g*
1	0	±8 *g*
1	1	±16 *g*

Рисунок 24 – Описание регистров THRESH_FF и TIME_FF

```
adxl_write(INT_ENABLE, FREE_FALL);     //включаем прерывание от free fall
adxl_write(INT_MAP, FREE_FALL);  //назначаем его на вывод IN1

adxl_write(INT_ENABLE, DATA_READY); //включаем прерывание по готовности данных
adxl_write(INT_MAP, DATA_READY);  //назначаем его на вывод IN0
```

Эти строки включают прерывания по свободному падению и готовности данных и назначают их на соотвествующие выводы. Возможные для настройки прерывания показан на рисунке 25, про активацию необходимых прерываний и настройку выводов показано на рисунке 26

INTERRUPTS

The ADXL345 provides two output pins for driving interrupts: INT1 and INT2. Both interrupt pins are push-pull, low impedance pins with output specifications shown in Table 13. The default configuration of the interrupt pins is active high. This can be changed to active low by setting the INT_INVERT bit in the DATA_FORMAT (Address 0x31) register. All functions can be used simultaneously, with the only limiting feature being that some functions may need to share interrupt pins.

Interrupts are enabled by setting the appropriate bit in the INT_ENABLE register (Address 0x2E) and are mapped to either the INT1 pin or the INT2 pin based on the contents of the INT_MAP register (Address 0x2F). When initially configuring the interrupt pins, it is recommended that the functions and interrupt mapping be done before enabling the interrupts. When changing the configuration of an interrupt, it is recommended that the interrupt be disabled first, by clearing the bit corresponding to that function in the INT_ENABLE register, and then the function be reconfigured before enabling the interrupt again. Configuration of the functions while the interrupts are disabled helps to prevent the accidental generation of an interrupt before desired.

The interrupt functions are latched and cleared by either reading the data registers (Address 0x32 to Address 0x37) until the interrupt condition is no longer valid for the data-related interrupts or by reading the INT_SOURCE register (Address 0x30) for the remaining interrupts. This section describes the interrupts that can be set in the INT_ENABLE register and monitored in the INT_SOURCE register.

DATA_READY

The DATA_READY bit is set when new data is available and is cleared when no new data is available.

SINGLE_TAP

The SINGLE_TAP bit is set when a single acceleration event that is greater than the value in the THRESH_TAP register (Address 0x1D) occurs for less time than is specified in the DUR register (Address 0x21).

DOUBLE_TAP

The DOUBLE_TAP bit is set when two acceleration events that are greater than the value in the THRESH_TAP register (Address 0x1D) occur for less time than is specified in the DUR register (Address 0x21), with the second tap starting after the time specified by the latent register (Address 0x22) but within the time specified in the window register (Address 0x23). See the Tap Detection section for more details.

Activity

The activity bit is set when acceleration greater than the value stored in the THRESH_ACT register (Address 0x24) is experienced on any participating axis, set by the ACT_INACT_CTL register (Address 0x27).

Inactivity

The inactivity bit is set when acceleration of less than the value stored in the THRESH_INACT register (Address 0x25) is experienced for more time than is specified in the TIME_INACT register (Address 0x26) on all participating axes, as set by the ACT_INACT_CTL register (Address 0x27). The maximum value for TIME_INACT is 255 sec.

FREE_FALL

The FREE_FALL bit is set when acceleration of less than the value stored in the THRESH_FF register (Address 0x28) is experienced for more time than is specified in the TIME_FF register (Address 0x29) on all axes (logical AND). The FREE_FALL interrupt differs from the inactivity interrupt as follows: all axes always participate and are logically AND'ed, the timer period is much smaller (1.28 sec maximum), and the mode of operation is always dc-coupled.

Watermark

The watermark bit is set when the number of samples in FIFO equals the value stored in the samples bits (Register FIFO_CTL, Address 0x38). The watermark bit is cleared automatically when FIFO is read, and the content returns to a value below the value stored in the samples bits.

Рисунок 25 – Возможные прерывания

Register 0x2F—INT_MAP (R/$\overline{W}$)

D7	D6	D5	D4
DATA_READY	SINGLE_TAP	DOUBLE_TAP	Activity
D3	**D2**	**D1**	**D0**
Inactivity	FREE_FALL	Watermark	Overrun

Any bits set to 0 in this register send their respective interrupts to the INT1 pin, whereas bits set to 1 send their respective interrupts to the INT2 pin. All selected interrupts for a given pin are OR'ed.

Register 0x30—INT_SOURCE (Read Only)

D7	D6	D5	D4
DATA_READY	SINGLE_TAP	DOUBLE_TAP	Activity
D3	**D2**	**D1**	**D0**
Inactivity	FREE_FALL	Watermark	Overrun

Bits set to 1 in this register indicate that their respective functions have triggered an event, whereas a value of 0 indicates that the corresponding event has not occurred. The DATA_READY, watermark, and overrun bits are always set if the corresponding events occur, regardless of the INT_ENABLE register settings, and are cleared by reading data from the DATAX, DATAY, and DATAZ registers. The DATA_READY and watermark bits may require multiple reads, as indicated in the FIFO mode descriptions in the FIFO section. Other bits, and the corresponding interrupts, are cleared by reading the INT_SOURCE register.

Рисунок 26 – Настройка прерываний

На этом инициализацию устройства можно считать законченной.

Так же объявляется функция adxl_read для чтения данных с регистров акселерометра.

```
uint8_t adxl_read(uint8_t address_reg) {//чтение с регистра одного байта
	address_reg |= 0x80;  // маска для задания бита чтения
	uint8_t data[1]={0}; //переменная для прочитанных данных

	HAL_I2C_Master_Transmit(&hi2c1, adxl_addr, address_reg, 1, timeout);
	//посылаем адрес регистра, с которого хотим читать

	HAL_I2C_Master_Receive(&hi2c1, adxl_addr, data, 1, timeout);
	//читаем байт в переменную data
	return data;
}
```

Данный фрагмент сначала формирует адрес и по маске устанавливает бит, отвечающий за чтение, согласно интерфейсу I^2C. Затем объявляется переменная для хранения данных.

```
HAL_I2C_Master_Transmit(&hi2c1, adxl_addr, address_reg, 1, timeout);
```

передает на шину I^2C адрес устройства с которым хочет связаться в формате чтения, и передает один байт, в котором содержится адрес регистра.

Далее функция

```
HAL_I2C_Master_Receive(&hi2c1, adxl_addr, data, 1, timeout);
```

запрашивает у устройства один байт данных, и записывает его в переменную, которую функция потом вернет.

4 РАЗРАБОТКА СТРУКТУРНОЙ СХЕМЫ УСТРОЙСТВА

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1 Data Sheet на акселерометр ADXL345 [Электронный ресурс]. URL:https://static.chipdip.ru/lib/876/DOC011876534.pdf (Дата обращения: 15.05.2023)

2 Токарчук, Т. С. Особенности регистрации медико-биологических данных с применением акселерометрических датчиков / Т. С. Токарчук, Ю. О. Боброва // СПбНТОРЭС: труды ежегодной НТК. – 2019. – № 1(74). – С. 367-369.

3 Data Sheet на микроконтроллер STM32WB35CCU6 [Электронный ресурс]. URL:https://www.st.com/resource/en/datasheet/stm32wb35cc.pdf (Дата обращения: 16.05.2023)

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: 16.05.2023)

5 Спецификация на Li-pol аккумулятор LP-130-232635 [Электронный ресурс]. URL:https://static.chipdip.ru/lib/412/DOC005412824.pdf (Дата обращения: 16.05.2023)

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

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)

ОПРЕДЕЛЕНИЯ, ОБОЗНАЧЕНИЯ И СОКРАЩЕНИЯ

ПРИЛОЖЕНИЕ А

Основная программа

```
/* USER CODE BEGIN Header */
/* USER CODE END Header */
/* Includes ------------------------------------------------------------------*/
#include "main.h"

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

/* Private typedef -----------------------------------------------------------*/
/* USER CODE BEGIN PTD */
/* USER CODE END PTD */

/* Private define ------------------------------------------------------------*/
/* USER CODE BEGIN PD */
//карта регистров
#define DEVID_ID 0x00
#define THRESH_TAP 0x1D
#define OFSX 0x1E
#define OFSY 0x0F
#define OFSZ 0x20
#define DUR 0x21
#define Latent 0x22
#define Window 0x23
#define THRESH_ACT 0x24
#define THRESH_INACT 0x25
#define TIME_INACT 0x26
#define ACT_INACT_CTL 0x27
#define THRESH_FF 0x28
#define TIME_FF 0x29
#define TAP_AXES 0x2A
#define ACT_TAP_STATUS 0x2B
#define BW_RATE 0x2C
#define POWER_CTL 0x2D
#define INT_ENABLE 0x2E
#define INT_MAP 0x2F
#define INT_SOURCE 0x30
#define DATA_FORMAT 0x31
#define DATAX0 0x32
#define DATAX1 0x33
#define DATAY0 0x34
#define DATAY1 0x35
#define DATAZ0 0x36
#define DATAZ1 0x37
#define FIFO_CTL 0x38
#define FIFO_STATUS 0x39


//полоса частот и чвстота дискретизации
#define BWRATE_0_10 0x00
#define BWRATE_0_20 0x01
#define BWRATE_0_39 0x02
#define BWRATE_0_78 0x03
#define BWRATE_1_56 0x04
```

```
#define BWRATE_3_13        0x05
#define BWRATE_6_25        0x06
#define BWRATE_12_5        0x07
#define BWRATE_25            0x08
#define BWRATE_50            0x09
#define BWRATE_100           0x0A
#define BWRATE_200           0x0B
#define BWRATE_400           0x0C
#define BWRATE_800           0x0D
#define BWRATE_1600        0x0E
#define BWRATE_3200        0x0F

//диапазон
#define RANGE_2G           0x00
#define RANGE_4G           0x01
#define RANGE_8G           0x02
#define RANGE_16G            0x03



//прерывания
#define DATA_READY             0x80
#define SINGLE_TAP           0x40
#define DOUBLE_TAP         0x20
#define Activity             0x10
#define Inactivity           0x08
#define FREE_FALL            0x04
#define Watermark          0x02
#define Overrun              0x01


//



/* USER CODE END PD */

/* Private macro -------------------------------------------------------------*/
/* USER CODE BEGIN PM */
/* USER CODE END PM */

/* Private variables ---------------------------------------------------------*/
 I2C_HandleTypeDef hi2c1;

IPCC_HandleTypeDef hipcc;

RTC_HandleTypeDef hrtc;

/* USER CODE BEGIN PV */
uint8_t data_rec[6]; //массив для хранения данных при чтении
int16_t x, y, z;//переменные для хранения значений ускорения в необработанном формате
float xg, yg, zg; //переменные для значений ускорения пересчитанных в доли ускорения свободного падения

uint8_t adxl_addr = 0x53;
int timeout = 100;

bool upd_flg=false;
bool ff_flg=false;
bool launch_flg=false;
```

```
/* USER CODE END PV */

/* Private function prototypes -----------------------------------------------*/
void SystemClock_Config(void);
void PeriphCommonClock_Config(void);
static void MX_GPIO_Init(void);
static void MX_I2C1_Init(void);
static void MX_IPCC_Init(void);
static void MX_RF_Init(void);
static void MX_RTC_Init(void);
/* USER CODE BEGIN PFP */
/* USER CODE END PFP */

/* Private user code ---------------------------------------------------------*/
/* USER CODE BEGIN 0 */
void adxl_write(uint8_t address_reg, uint8_t value) {//запись в регистр
	uint8_t data[2];//массив для хранения посылки
	data[0] = address_reg;//сначала передаем адрес регистра в который будем читать
	data[1] = value; //Затем значение которое нужно записываем
	HAL_I2C_Master_Transmit(&hi2c1, adxl_addr, data, 2, timeout)  ;//отправляем массив с адресом и значением
}



uint8_t adxl_read(uint8_t address_reg) {//чтение с регистра одного байта
	address_reg |= 0x80;  // маска для задания бита чтения
	uint8_t data[1]={0}; //переменная для прочитанных данных

	HAL_I2C_Master_Transmit(&hi2c1, adxl_addr, address_reg, 1, timeout);//посылаем адрес регистра, с которого хотим

	HAL_I2C_Master_Receive(&hi2c1, adxl_addr, data, 1, timeout);//читаем байт в переменную data
	return data;
}



void adxl_init(void) {//инициализация акселерометра и настройка

	adxl_write(DATA_FORMAT, RANGE_8G);  //настраиваем диапазон +- 8g

	adxl_write(POWER_CTL, 0x00);  // выход из режима сна
	adxl_write(POWER_CTL, 0x08);  //включаем преобразование
	adxl_write(THRESH_FF, 0x04);  //настраиваем значение free fall treshold = 0.5g
	adxl_write(TIME_FF, 0x02);    //настраиваем время free fall
	//по двум параметрам выше срабатывает прерывание

	adxl_write(INT_ENABLE, FREE_FALL);    //включаем прерывание от free fall
	adxl_write(INT_MAP, FREE_FALL);  //назначаем его на вывод IN1

	adxl_write(INT_ENABLE, DATA_READY); //включаем прерывание по готовности данных
	adxl_write(INT_MAP, !DATA_READY);  //назначаем его на вывод IN0

}

/* USER CODE END 0 */

/**
  * @brief  The application entry point.
```

```
  * @retval int
  */
int main(void)
{
  /* USER CODE BEGIN 1 */
  /* USER CODE END 1 */

  /* MCU Configuration--------------------------------------------------------*/

  /* Reset of all peripherals, Initializes the Flash interface and the Systick. */
  HAL_Init();//подключение библиотеки HAl
  /* Config code for STM32_WPAN (HSE Tuning must be done before system clock configuration) */
  MX_APPE_Config(); //инициализация Bluetooth стека

  /* USER CODE BEGIN Init */
  /* USER CODE END Init */

  /* Configure the system clock */
  SystemClock_Config(); //инициализация тактирования

 /* Configure the peripherals common clocks */
  PeriphCommonClock_Config();

  /* IPCC initialisation */
   MX_IPCC_Init();//инициализация Inter-processor communication controller

  /* USER CODE BEGIN SysInit */
  /* USER CODE END SysInit */

  /* Initialize all configured peripherals */
  MX_GPIO_Init(); //инициализация портов ввода вывода
  MX_I2C1_Init(); //инициализация I2C
  MX_RF_Init(); //инициализация радиомодуля
  MX_RTC_Init(); //инициализация real time clock
  /* USER CODE BEGIN 2 */
  /* USER CODE END 2 */

  /* Init code for STM32_WPAN */
  MX_APPE_Init(); //
  adxl_init(); //инициализация и настройка акселерометра



  /* Infinite loop */
  /* USER CODE BEGIN WHILE */
  while (1)
  {
    /* USER CODE END WHILE */
    MX_APPE_Process(); //запуск функций, связанных со стеком Bluetooth

    if(launch_flg==true){
          uint16_t dataX = adxl_read(DATAX0)|(adxl_read(DATAX1)<<8);
          uint16_t dataY = adxl_read(DATAY0)|(adxl_read(DATAY1)<<8);
          uint16_t dataZ = adxl_read(DATAZ0)|(adxl_read(DATAZ1)<<8);


    }
```

```
    /* USER CODE BEGIN 3 */
  }
  /* USER CODE END 3 */
}

/**
  * @brief System Clock Configuration
  * @retval None
  */
void SystemClock_Config(void) //настройки тактирования(настраивались в кодогенераторе Cube)
{
  RCC_OscInitTypeDef RCC_OscInitStruct = {0};
  RCC_ClkInitTypeDef RCC_ClkInitStruct = {0};

  /** Configure the main internal regulator output voltage
  */
  __HAL_PWR_VOLTAGESCALING_CONFIG(PWR_REGULATOR_VOLTAGE_SCALE1);

  /** Initializes the RCC Oscillators according to the specified parameters
  * in the RCC_OscInitTypeDef structure.
  */
  RCC_OscInitStruct.OscillatorType = RCC_OSCILLATORTYPE_HSI|RCC_OSCILLATORTYPE_LSI1
                     |RCC_OSCILLATORTYPE_HSE;
  RCC_OscInitStruct.HSEState = RCC_HSE_ON;
  RCC_OscInitStruct.HSIState = RCC_HSI_ON;
  RCC_OscInitStruct.HSICalibrationValue = RCC_HSICALIBRATION_DEFAULT;
  RCC_OscInitStruct.LSIState = RCC_LSI_ON;
  RCC_OscInitStruct.PLL.PLLState = RCC_PLL_ON;
  RCC_OscInitStruct.PLL.PLLSource = RCC_PLLSOURCE_HSE;
  RCC_OscInitStruct.PLL.PLLM = RCC_PLLM_DIV2;
  RCC_OscInitStruct.PLL.PLLN = 8;
  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();
  }

  /** Configure the SYSCLKSource, HCLK, PCLK1 and PCLK2 clocks dividers
  */
  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;

  if (HAL_RCC_ClockConfig(&RCC_ClkInitStruct, FLASH_LATENCY_3) != HAL_OK)
  {
    Error_Handler();
  }

  /** Enables the Clock Security System
  */
  HAL_RCC_EnableCSS();
}
```

```
/**
  * @brief Peripherals Common Clock Configuration
  * @retval None
  */
void PeriphCommonClock_Config(void)//настройки тактирования для радиочасти (настраивались в кодогенераторе Cu
{
  RCC_PeriphCLKInitTypeDef PeriphClkInitStruct = {0};

  /** Initializes the peripherals clock
  */
  PeriphClkInitStruct.PeriphClockSelection = RCC_PERIPHCLK_SMPS|RCC_PERIPHCLK_RFWAKEUP;
  PeriphClkInitStruct.RFWakeUpClockSelection = RCC_RFWKPCLKSOURCE_HSE_DIV1024;
  PeriphClkInitStruct.SmpsClockSelection = RCC_SMPSCLKSOURCE_HSI;
  PeriphClkInitStruct.SmpsDivSelection = RCC_SMPSCLKDIV_RANGE1;

  if (HAL_RCCEx_PeriphCLKConfig(&PeriphClkInitStruct) != HAL_OK)
  {
    Error_Handler();
  }
  /* USER CODE BEGIN Smps */
  /* USER CODE END Smps */
}

/**
  * @brief I2C1 Initialization Function
  * @param None
  * @retval None
  */
static void MX_I2C1_Init(void) //настройки i2c (настраивались в кодогенераторе Cube)
{

  /* USER CODE BEGIN I2C1_Init 0 */
  /* USER CODE END I2C1_Init 0 */

  /* USER CODE BEGIN I2C1_Init 1 */
  /* USER CODE END I2C1_Init 1 */
  hi2c1.Instance = I2C1;
  hi2c1.Init.Timing = 0x10707DBC;
  hi2c1.Init.OwnAddress1 = 0;
  hi2c1.Init.AddressingMode = I2C_ADDRESSINGMODE_7BIT;
  hi2c1.Init.DualAddressMode = I2C_DUALADDRESS_DISABLE;
  hi2c1.Init.OwnAddress2 = 0;
  hi2c1.Init.OwnAddress2Masks = I2C_OA2_NOMASK;
  hi2c1.Init.GeneralCallMode = I2C_GENERALCALL_DISABLE;
  hi2c1.Init.NoStretchMode = I2C_NOSTRETCH_DISABLE;
  if (HAL_I2C_Init(&hi2c1) != HAL_OK)
  {
    Error_Handler();
  }

  /** Configure Analogue filter
  */
  if (HAL_I2CEx_ConfigAnalogFilter(&hi2c1, I2C_ANALOGFILTER_ENABLE) != HAL_OK)
  {
    Error_Handler();
  }

  /** Configure Digital filter
  */
```

```
  if (HAL_I2CEx_ConfigDigitalFilter(&hi2c1, 0) != HAL_OK)
  {
    Error_Handler();
  }
  /* USER CODE BEGIN I2C1_Init 2 */
  /* USER CODE END I2C1_Init 2 */

}

/**
  * @brief IPCC Initialization Function
  * @param None
  * @retval None
  */
static void MX_IPCC_Init(void)//настройки интерфейса связи процессорных ядер(настраивались в кодогенераторе Cube
{

  /* USER CODE BEGIN IPCC_Init 0 */
  /* USER CODE END IPCC_Init 0 */

  /* USER CODE BEGIN IPCC_Init 1 */
  /* USER CODE END IPCC_Init 1 */
  hipcc.Instance = IPCC;
  if (HAL_IPCC_Init(&hipcc) != HAL_OK)
  {
    Error_Handler();
  }
  /* USER CODE BEGIN IPCC_Init 2 */
  /* USER CODE END IPCC_Init 2 */

}

/**
  * @brief RF Initialization Function
  * @param None
  * @retval None
  */

/**
  * @brief RTC Initialization Function
  * @param None
  * @retval None
  */
static void MX_RTC_Init(void)//настройки часов реального времени(настраивались в кодогенераторе Cube)
{

  /* USER CODE BEGIN RTC_Init 0 */
  /* USER CODE END RTC_Init 0 */

  /* USER CODE BEGIN RTC_Init 1 */
  /* USER CODE END RTC_Init 1 */

  /** Initialize RTC Only
  */
  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;
  if (HAL_RTC_Init(&hrtc) != HAL_OK)
  {
    Error_Handler();
  }

  /** Enable the WakeUp
  */
  if (HAL_RTCEx_SetWakeUpTimer_IT(&hrtc, 0, RTC_WAKEUPCLOCK_RTCCLK_DIV16) != HAL_OK)
  {
    Error_Handler();
  }
  /* USER CODE BEGIN RTC_Init 2 */
  /* USER CODE END RTC_Init 2 */

}



/**
  * @brief RF Initialization Function
  * @param None
  * @retval None
  */
static void MX_RF_Init(void)
{

  /* USER CODE BEGIN RF_Init 0 */
  /* USER CODE END RF_Init 0 */

  /* USER CODE BEGIN RF_Init 1 */
  /* USER CODE END RF_Init 1 */
  /* USER CODE BEGIN RF_Init 2 */
  /* USER CODE END RF_Init 2 */

}

/**
  * @brief GPIO Initialization Function
  * @param None
  * @retval None
  */
static void MX_GPIO_Init(void) //настройки портов ввода вывода(настраивались в кодогенераторе Cube)
{
  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 pins : PA4 PA5 */
  GPIO_InitStruct.Pin = GPIO_PIN_4|GPIO_PIN_5;
  GPIO_InitStruct.Mode = GPIO_MODE_IT_RISING;
  GPIO_InitStruct.Pull = GPIO_NOPULL;
  HAL_GPIO_Init(GPIOA, &GPIO_InitStruct);

}
```

```
/* USER CODE BEGIN 4 */
/* USER CODE END 4 */

/**
  * @brief  This function is executed in case of error occurrence.
  * @retval None
  */
void Error_Handler(void)
{
  /* USER CODE BEGIN Error_Handler_Debug */
  /* USER CODE END Error_Handler_Debug */
}

#ifdef  USE_FULL_ASSERT
/**
  * @brief  Reports the name of the source file and the source line number
  *         where the assert_param error has occurred.
  * @param  file: pointer to the source file name
  * @param  line: assert_param error line source number
  * @retval None
  */
void assert_failed(uint8_t *file, uint32_t line)
{
  /* USER CODE BEGIN 6 */
  /* USER CODE END 6 */
}
#endif /* USE_FULL_ASSERT */
```