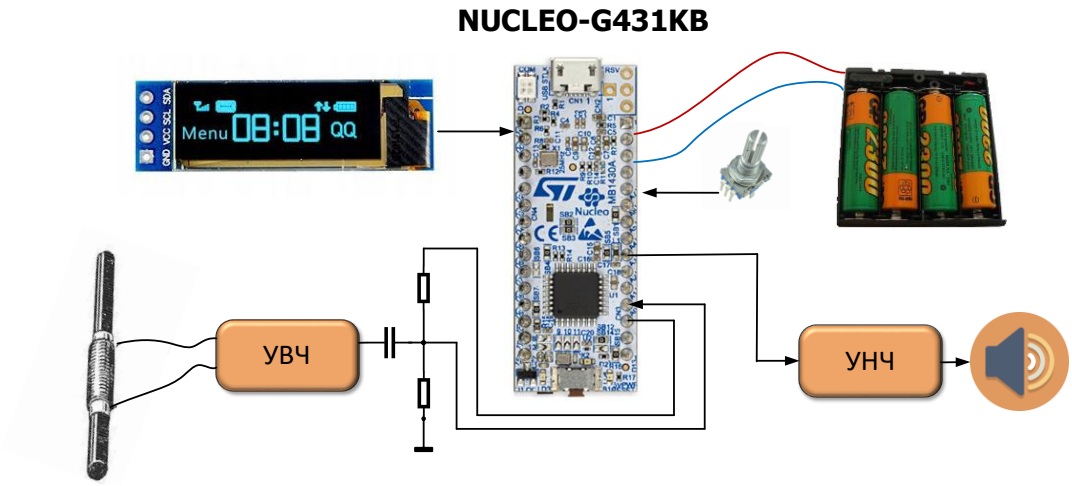
Простой цифровой радиоприемник на базе микроконтроллера STM32G4 своими руками



Микроконтроллеры компании STMicrielectronics широко применяются в различных сферах профессиональной разработки и технического творчества. Невысокая стоимость, доступность демонстрационных плат, развитый набор периферии, достаточно высокая производительность, низкое энергопотребление и развитая экосистема разработки делает создание различных устройств на основе данных контроллеров простым и увлекательным делом.

С выходом в серию семейства STM32G я наконец не выдержал и решил попробовать реализовать одну давнюю идею: собрать простое радиоприемное устройство диапазона LW/MW с минимальным количеством внешних электронных компонентов. Конструкция и программное обеспечение должны быть достаточно простыми, чтобы можно было на пальцах объяснить новичку основы цифровой обработки сигналов, не углубляясь в теорию.

Для реализации проекта была выбрана доступная демонстрационная плата NUCLEO-G431KB, содержащая контроллер STM32G431KBT6 (flash 128KB, ram 32KB, тактовая частота процессора до 170 MHz), минимально необходимую обвязку и интегрированный отладчик/программатор STLINK-V3E. Особенно ценным является наличие в контроллере 12 разрядного аналого-цифрового преобразователя ADC с частотой дискретизации до 4 MHz, что теоретически позволяет обрабатывать сигналы с частотами до 2 MHz.

Излагать материал я буду от простого к сложному в виде описания последовательных экспериментов, постепенно усложняя программное обеспечение. Компиляция программ выполнялась в Windows 10 с использованием бесплатной интегрированной среды разработки STM32CubeIDE от компании STMicrielectronics (https://www.st.com/en/development-tools/stm32cubeide.html). Исходные коды экспериментальных проектов находятся по адресу https://github.com/OlegDyakov/simple-radio.

Для разработки программ использовались следующие официальные документы от компании STMicrielectronics:

- «RM0440, Reference manual STM32G4 Series advanced Arm ® -based 32-bit MCUs» - подробное описание функциональных блоков контроллеров серии STM32G4.

- «DS12589 Rev 2 - Datasheet STM32G431x6, STM32G431x8, STM32G431xB» - описание контроллеров линейки STM32G431x6, STM32G431x8, STM32G431xB.

- «PM0214 Rev 10 - STM32 Cortex ® -M4 MCUs and MPUs programming manual» - руководство программиста.

Для выполнения экспериментов был создан исследовательский стенд (рис. 1), который в последствии позволил изготовить реально работающий радиоприемник начального уровня. В принципе, эксперименты можно не повторять, они предназначены для объяснения принципов работы программного обеспечения и получения характеристик с реальной платы.

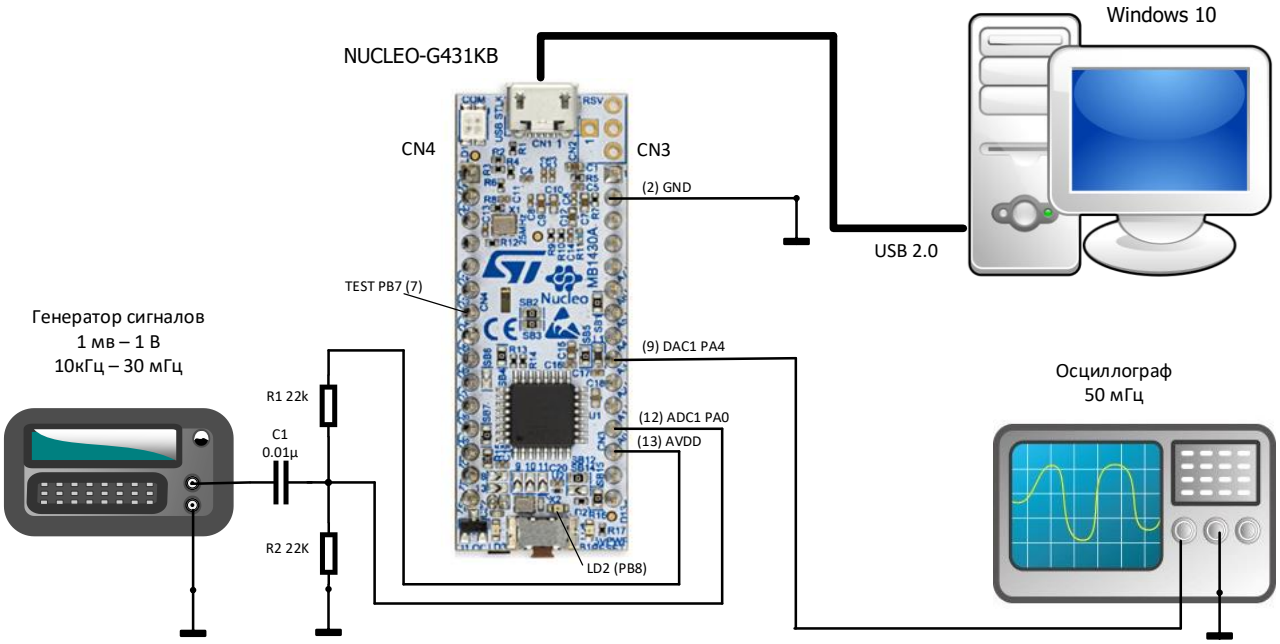


Рисунок 1. Схема исследовательского стенда NUCLEO-G431KB

1. Структура радиоприемника

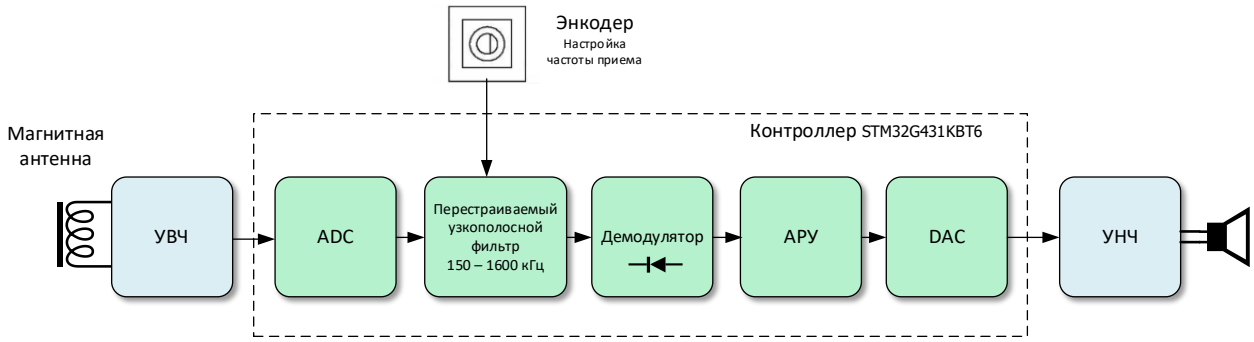


Рисунок 2. Структурная схема радиоприемника

Радиоприемник построен по классической схеме приемника прямого усиления и предназначен для приема АМ радиостанций, вещающих в диапазонах длинных и средних волн (LW и MW). Сигнал с ферритовой магнитной антенны поступает на усилитель высокой частоты (УВЧ). Далее сигнал оцифровывается в аналого-цифровом преобразователе (ADC) и подается на перестраиваемый

цифровой узкополосной полосовой фильтр, обеспечивающий настройку на нужную несущую частоту радиостанции. Выделенный сигнал передается на АМ демодулятор, реализованный по принципу диодного детектора. Для уменьшения эффекта «замирания сигнала» после демодулятора стоит система автоматической регулировки усиления (АРУ). Полученный цифровой звуковой сигнал поступает на цифро-аналоговый преобразователь (DAC), далее - на усилитель низкой частоты (УНЧ) и динамик (наушники).

2. Базовый проект

Для разработки программного обеспечения для контроллеров STM32 существует большое количество очень хороших готовых профессиональных программных пакетов. К сожалению, новичкам достаточно сложно сразу осмыслить сложный мир профессиональных средств разработки. К тому же стандартные библиотеки (CMSIS, HAL) закрывают от программиста механизмы работы с функциональными модулями контроллера.

У разработчика имеется довольно подробное описание функциональных модулей контроллера, например: "RM0440 Reference manual STM32G4 Series advanced Arm ® -based 32-bit MCUs", содержащее более 2000 страниц. Как правило, программисты применяют стандартные библиотеки от производителей контроллера, которые, мягко говоря, довольно сильно изолируют разработчика от управления аппаратными блоками. Если что-то работает не так или нам нужно "выжать" из железки максимум, приходиться «лезть» в код библиотеки, который написан на профессиональном С, предназначен для обеспечения переносимости на разные контроллеры и системы разработки и довольно сложен для понимания людям, которые делают первые шаги в embedded программировании. Хотелось быть «поближе к транзисторам», поэтому был разработан простой базовый проект на С, который является основой для реализации всех экспериментов, описанных в данной статье.

Экспериментальные проекты носят исследовательский характер и не претендуют на демонстрацию профессиональных подходов при разработке встроенного программного обеспечения.

Рассмотрим простейшую программу, написанную на языке С, которая выполняет переключение светодиода LD2 и логических уровней на выводе CN4-7 (PB7) демонстрационной платы NUCLEO-G431KB. Вывод PB7 мы будем использовать для измерения с помощью осциллографа времени выполнения различных функций в экспериментальных проектах. Проект простейшей программы находится в папке 01_Minimal.

Проект состоит из следующих файлов:

system_init.c	инициализация регистров контроллера и оперативной памяти при старте,
vectors.c	таблица прерываний,
gpio.c	настройка ножек контроллера,
main.c	основной цикл программы,
stm32g431.h	описание регистров контроллера, используемых в проекте,
stm32f4.ld	скрипт программы компоновщика.

Начнем анализ проекта с файла скрипта компоновщика «stm32f4.ld». Этот файл предназначен для предоставления информации компоновщику, каким образом организована основная память контроллера (FLASH, RAM), как необходимо размещать секции (области данных, с которыми работает компилятор С) в физической памяти контроллера, а также какую функцию программы необходимо вызывать сразу после рестарта.

Компилятор языка С размещает различные типы данных в разных стандартных секциях памяти:

.text - исполняемый код программы,

.rodata - константы,
.data - глобальные переменные, имеющие начальные константные значения (например int a=5;),
.bss - неинициализированные глобальные переменные (например int b;) .

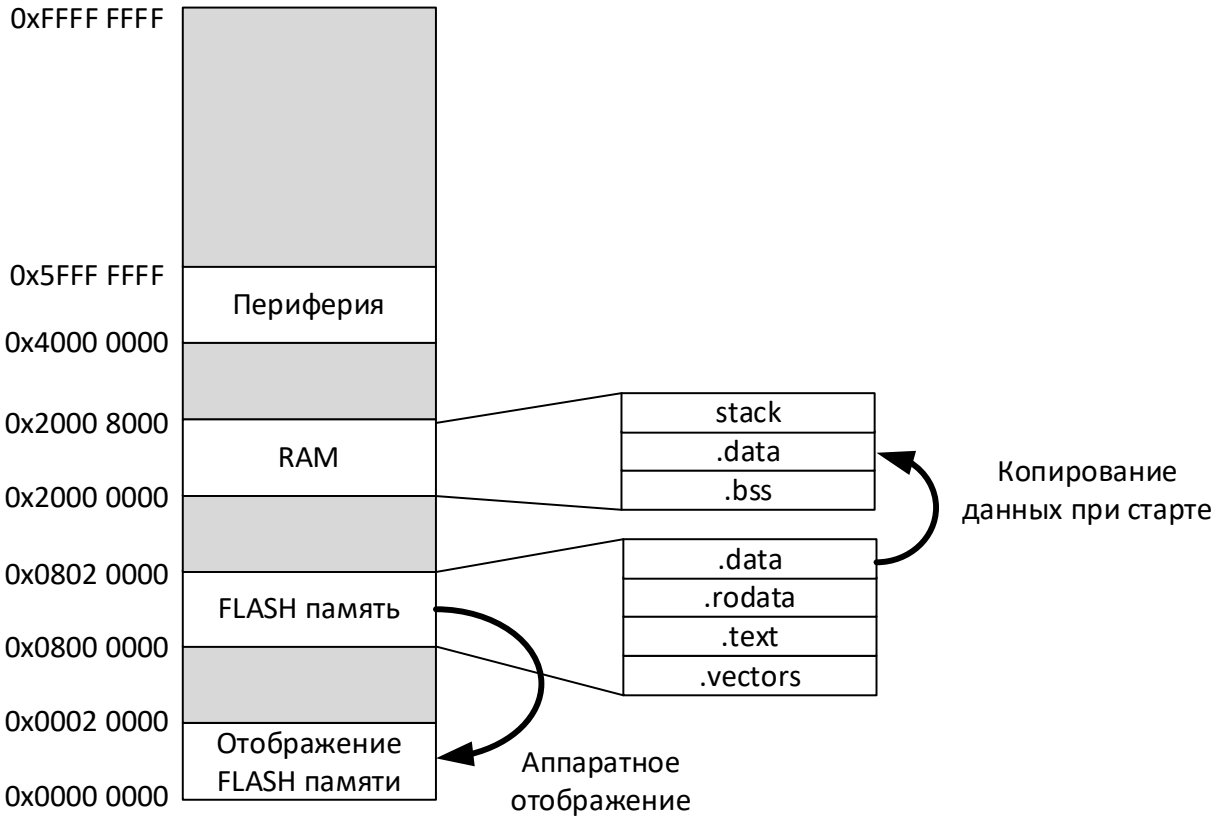


Рисунок 3. Распределение памяти

Схематично распределение памяти контроллера, а также размещение секций кода и данных программы представлены на рисунке 3. Здесь указаны только те области адресного пространства контроллера, которые имеют отношение к нашему проекту.

Кроме описания памяти и определения схемы расположения секций, в файле скрипта задаются значения констант, отображающих начальные и конечные адреса секций с данными, соответственно:

_sdata_ram, _edata_ram для секции .data, располагающейся в RAM;
_sdata_flash, _edata_flash для секции .data, располагающейся во FLASH;
_sbss, _ebss для секции. bss;
_svector, _evector для секции .vectors;
_estack начальный адрес стека.

Данные константы используются при инициализации памяти в коде программы, расположенном в файле system_init.c.

Контроллер STM32G431KBT6 является достаточно развитой системой на кристалле (SoC), содержащей около 50 функциональных модулей, которые управляются через регистры. Для программного обеспечения регистры доступны как 32 битные ячейки памяти, расположенные в разделе адресного пространства «Периферия» (рисунок 3). Описание всех регистров контроллера (названия регистров, адреса регистров, название и расположение битовых полей внутри регистра) занимает более 13 тысяч

строк (стандартный файл stm32g431xx.h из пакета CMSIS). К счастью, все устроено таким образом, что после аппаратного перезапуска в контроллере активизируется минимально необходимое количество блоков. Если неиспользуемые модули специально не включать, то они не будут оказывать никакого влияния на процесс работы программы. В проектах будем использовать усеченный вариант файла описания - stm32g431.h, который содержит определения управляющих регистров тех функциональных модулей, которые необходимы для реализации программ - экспериментов.

Функциональные модули в адресном пространстве контроллера представлены в виде последовательности 32 битных регистров (ячеек памяти). Согласно спецификации, для каждого модуля определен базовый адрес (адрес первого регистра в блоке) и смещение. Например, у модуля GPIOB базовый адрес GPIOB_BASE = 0x48000400. Смещение управляющего регистра GPIOB_ODR (output data register) равен 0x14, т.е. абсолютный адрес равен значению GPIOB_ODR = GPIOB_BASE + 0x14 = 0x48000414.

```
/*********************** GPIO (General-purpose I/Os) ********************************/

#define GPIOB_BASE     (0x48000400)
#define GPIOB_MODER    (*(volatile uint32_t *) (GPIOB_BASE + 0x00))
#define GPIOB_ODR      (*(volatile uint32_t *) (GPIOB_BASE + 0x14))
```

В файле stm32g431.h определены макросы, которые позволяют сделать более наглядной работу с регистрами контроллера.

```
//Макросы, упрощающие работу с регистрами, где
//REG - адрес регистра
//BIT - 32 битное слово - биты для которых будет выполнена данная функция.
//Например: INVERT(GPIOB_ODR,0x6)  проинвертирует биты 1 и 2 в регистре GPIOB_ODR

#define SET_BIT(REG, BIT)     ((REG) |= (BIT))
#define CLEAR_BIT(REG, BIT)   ((REG) &= ~(BIT))
#define INVERT_BIT(REG, BIT)  ((REG) ^= (BIT))
#define READ_BIT(REG, BIT)    ((REG) & (BIT))
#define CLEAR_REG(REG)        ((REG) = (0x0))
#define READ_REG(REG)         ((REG))
#define WRITE_REG(REG, VAL)   ((REG) = (VAL))
#define MODIFY_REG(REG, CLEARMASK, SETMASK)  WRITE_REG((REG), (((READ_REG(REG)) &
       (~(CLEARMASK))) | (SETMASK)))
```

Теперь перейдем к обсуждению последовательности действий контроллера при отработке аппаратного рестарта.

Контроллер имеет внутренний RC тактовый генератор на 16MHz (HSI RC 16). Сразу после рестарта выходной сигнал с данного генератора используется в качестве тактового сигнала для вычислительного ядра и периферийных модулей.

При отработке аппаратного рестарта, после выполнения ряда настроечных процедур, ядро контроллера загружает содержимое ячейки 0x00000000 в регистр SP (указатель стека), а содержимое ячейки 0x00000004 в регистр PC (счетчик команд). Другими словами, ячейка 0x00000000 должна содержать адрес начала стека, а ячейка 0x00000004 адрес, с которого начинается выполнение кода программы при рестарте (Reset handler’s address).

Область памяти в начале адресного пространства является виртуальной (0x00000000 - 0x00080000), т.е. в зависимости от настройки метода загрузки микроконтроллера, в данную область могут отображаться различные модули памяти: внутренний FLASH, RAM, системный ROM, внешний FLASH. Метод загрузки определяется распайкой ножки PB8 контроллера. Для платы NUCLEO-G431KB по умолчанию загрузка выполняется из внутреннего FLASH. Это означает, что ячейки FLASH памяти, расположенные по адресам 0x08000000 – 0x08020000, также доступны по адресам 0x00000000 – 0x00020000. Таким образом, для данного режима загрузки можно сказать, что микроконтроллер стартует с первого адреса FLASH памяти т.е. с адреса 0x08000000.

Как мы видели из файла «stm32f4.ld», первые 256 слов во внутреннем флэше занимает таблица векторов прерывания (секция. vectors). Элементы таблицы векторов прерывания содержат адреса

специальных функций (обработчиков), которые автоматически вызываются при возникновении прерываний от различных модулей микроконтроллера или в случае аппаратных ошибок. За исключением начального элемента таблицы, который содержит адрес вершины стека.

Определение таблицы векторов прерывания находится в файле «vectors.c».

```
__attribute__ ((section(".vectors"),used)) //Таблицу размещаем в секции .vectors
uint32_t * vectors[]  = {
  (uint32_t *) &_estack,               //Вершина стека
  (uint32_t *) start_up,               //Обработчик reset (точка входа)
  (uint32_t *) nmi_handler,            //Обработчик немаскируемого прерывания
  (uint32_t *) hardfault_handler       //Обработчик прерывания аппаратной ошибки
};
```

Как видно из кода таблицы, в базовом проекте определены только первые три обработчика, необходимые для работы проекта. Остальные элементы таблицы мы будем заполнять в следующих проектах по мере необходимости.

Функции nmi_handler и hardfault_handler обрабатывают аварийные ситуации. Они переводят вычислительное ядро в глухой цикл.

Функция start_up находится в файле «system_init.c» и выполняет действия по инициализации RAM, необходимые для работы программы, скомпилированной с языка С. Сначала выполняется копирование данных секции .data из FLASH в RAM, затем выполняется запись нулей в секцию. bss.

Для настройки линий PB7 и PB8 на режим логического выхода используется функция GpioInit, расположенная в файле gpio.c.

```
void GpioInit(void)
{
  // Подать тактовый сигнал на модуль GPIOA
  SET_BIT(RCC_AHB2ENR, RCC_AHB2ENR_GPIOBEN);

  //Включить PIN8 на вывод (светодиод LD2)
  MODIFY_REG(GPIOB_MODER,  GPIO_MODER_MODE8, 1<<GPIO_MODER_MODE8_Pos);

  //Включить PIN7 на вывод
  MODIFY_REG(GPIOB_MODER,  GPIO_MODER_MODE7, 1<<GPIO_MODER_MODE7_Pos);
}
```

Основные минимальные настройки (по умолчанию) загружаются в управляющие регистры на аппаратном уровне при рестарте контроллера. Для реализации функции переключения светодиода необходимо настроить модуль GPIO (General-purpose I/O). Для этого первым делом нужно подать тактовый сигнал на данный модуль. Эта операция выполняется через модуль RCC (Reset and clock control).

Для того чтобы точно определить период цикла переключения светодиода с помощью осциллографа, дополнительно к PB8 также настроим на вывод линию PB7 и будем изменять значения на данных выводах одновременно.

Основной цикл программы находится в файле «main.c».

Измеренное осциллографом время между двумя переключениями на линии PB7 составляет 686 мс. Это время в основном расходуется на выполнение цикла:

```
while( i ) i--;
```

Ассемблерный код, сгенерированный компилятором для данной строки, имеет следующий вид:

```
a:    ldr    r3, [r7, #4]  // Загрузить переменную i в регистр r3. (2 такта)
```

```
subs    r3, #1          // Вычесть 1 из регистра r3. (1 такт)
str     r3, [r7, #4]    // Сохранить регистр r3 в переменную i. (2 такта)
ldr     r3, [r7, #4]    // Загрузить переменную i в регистр r3. (2 такта)
cmp     r3, #0          // Сравнить регистр r3 с 0 (1 такт)
bne.n   a               // Если не равно перейти на метку a: (1 такт +
                        // 2 такта перестройка конвейера команд)
```

Данный код можно увидеть в файле «01_Minimal.list» в папке 01_Minimal\Debug\ проекта.

В комментариях для каждой ассемблерной инструкции указаны выполняемые действия и время выполнения в тактах. Общее время одного цикла равно 11 тактам. При частоте внутреннего RC генератора 16 MHz период одного такта составляет 62,5 нс. Время выполнения одной итерации цикла составляет 62,5 x 11 = 687,5 нс. На миллион итераций будет затрачено приблизительно 687 мс, что практически совпадает с результатом, полученным с помощью осциллографа при измерении сигнала на ножке PB7 (рис. 4). Данная техника оценки времени выполнения функций будет использоваться в следующих экспериментах.

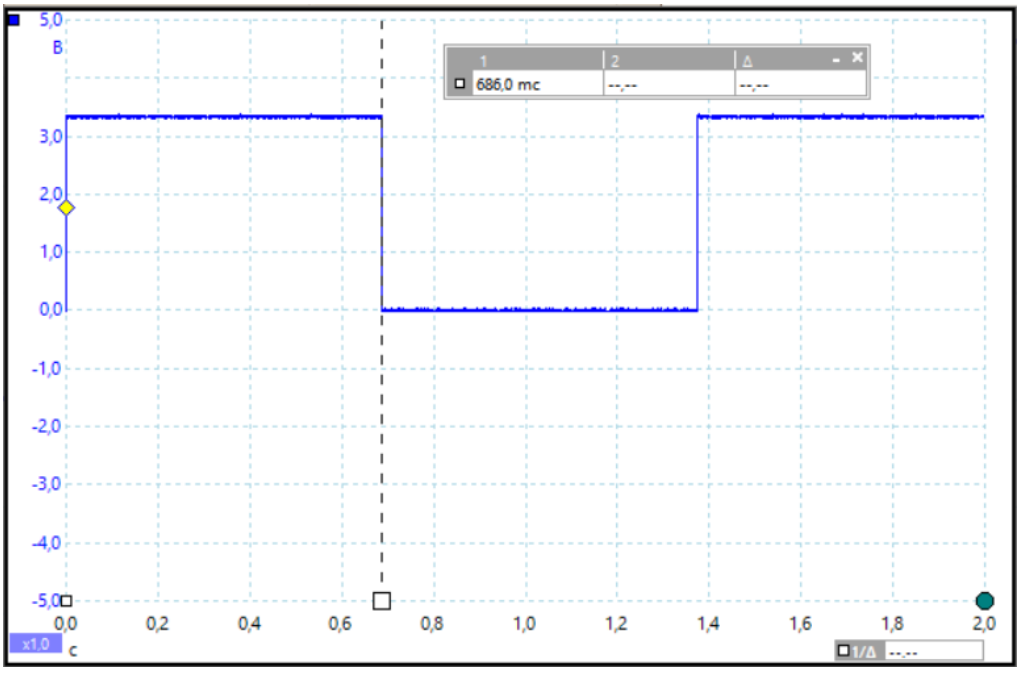


Рисунок 4. Сигнал на выводе PB7 (DC) при тактовой частоте 16 MHz

3. Подключение внешнего кварцевого резонатора

Стабильность внутреннего RC генератора микроконтроллера составляет приблизительно 1%, что недостаточно для поддержания точной настройки радиоприемника. Демонстрационная плата NUCLEO-G431KB содержит кварцевый резонатор на 24MHz, который обеспечивает стабильность порядка 0,01%. Для физического подключения кварцевого резонатора к микроконтроллеру необходимо запаять две перемычки под номером 9 и 10, как показано на рисунке 5.

Рисунок 5. Перемычки для подключения кварцевого резонатора

Также неплохо сразу удалить перемычки 2 и 3, которые нам будут мешать в дальнейшем при подключении энкодера. Удаление перемычек лучше выполнять с помощью паяльного фена.

Тактовая частота вычислительного ядра микроконтроллера STM32G431KBT6 может достигать 170 MHz. Для подключения кварцевого резонатора к внутренним схемам микроконтроллера и увеличения тактовой частоты до 170 MHz необходимо выполнить настройку следующих модулей:

- Power control (PWR),
- Embedded Flash memory (FLASH),
- Reset and clock control (RCC).

Выполняются перечисленные операции в функции SystemClock_Config() из файла «system_init.c».

```
void SystemClock_Config(void)
{

  //Включить режим повышения выходного напряжения главного регулятора до 1,28 вольт
  CLEAR_BIT(PWR_CR5, PWR_CR5_R1MODE);

  //Установка задержки чтения FLASH памяти до 4-х тактов
  FLASH_ACR=FLASH_ACR_DBG_SWEN  | FLASH_ACR_DCEN | FLASH_ACR_ICEN | FLASH_ACR_LATENCY_4WS ;
  while((FLASH_ACR & 0xf) != FLASH_ACR_LATENCY_4WS );

  //Подключение внешнего кварцевого резонатора (HSE ON)
  SET_BIT(RCC_CR, RCC_CR_HSEON);
  while(READ_BIT(RCC_CR, RCC_CR_HSERDY) != (RCC_CR_HSERDY));

  //Настройка PLL, используемого для SYSCLK домена
  //PLLM=6 (значение поля 5), PLLN=85 (значение поля 85), PLLR=2 (значение поля 0)
```

```
  MODIFY_REG(RCC_PLLCFGR, RCC_PLLCFGR_PLLSRC | RCC_PLLCFGR_PLLM| RCC_PLLCFGR_PLLN |
RCC_PLLCFGR_PLLR, RCC_PLLCFGR_PLLSRC_HSE | 5 <<RCC_PLLCFGR_PLLM_Pos | (85 <<
RCC_PLLCFGR_PLLN_Pos) | 0<<RCC_PLLCFGR_PLLR_Pos);

//Включение PLL
  SET_BIT(RCC_CR, RCC_CR_PLLON);
  while(READ_BIT(RCC_CR, RCC_CR_PLLRDY) != (RCC_CR_PLLRDY));

  //Включение выхода PLL, используемого для  SYSCLK
  SET_BIT(RCC_PLLCFGR, RCC_PLLCFGR_PLLREN);

  //Подключение выхода PLL в качестве источника  SYSCLK
  MODIFY_REG(RCC_CFGR, RCC_CFGR_SW, RCC_CFGR_SW_PLL);
}
```

Повышение тактовой частоты до максимального значения 170 MHz требует принятия специальных мер по настройке внутренних блоков.

Во-первых, необходимо увеличить напряжение питание ядра микроконтроллера с 1,2 вольт до 1,28 вольт (boost mode). Управление питанием внутренних блоков микроконтроллера осуществляет специальный модуль Power control (PWR).

Во-вторых, следует увеличить количество тактов ожидания чтения данных из FLASH памяти. FLASH память является относительно медленным устройством и позволяет считывать информацию без дополнительных тактов ожидания со скоростью до 30 MHz. Если тактовая частота ядра повышается до 170 MHz, то необходимо увеличить количество тактов ожидания до 4.

Для повышения тактовой частоты с 24MHz от кварцевого резонатора до 170 MHz используется блок PLL (phase-locked loop). Настройка PLL выполняется с помощью регистра RCC_PLLCFGR (PLL configuration register). Для настройки частоты используется три поля: PLLM, PLLN, PLLR:

$$Fclk = ((Finput/PLLM)*PLLN)/PLLR .$$

В нашем случае PLLM=6, PLLN=85, PLLR=2 и тактовая частота равна 170 = ((24/6)*85)/2.

После внесения всех перечисленных выше изменений, измеренное осциллографом на линии PB7 время между двумя переключениями составляет 64,7 мс, что соответствует 64,7 нс на одну итерацию цикла, или 64,7/11 = 5,88 нс на такт (что приблизительно соответствует 170 MHz тактовой частоты). То есть все настройки выполнены правильно.

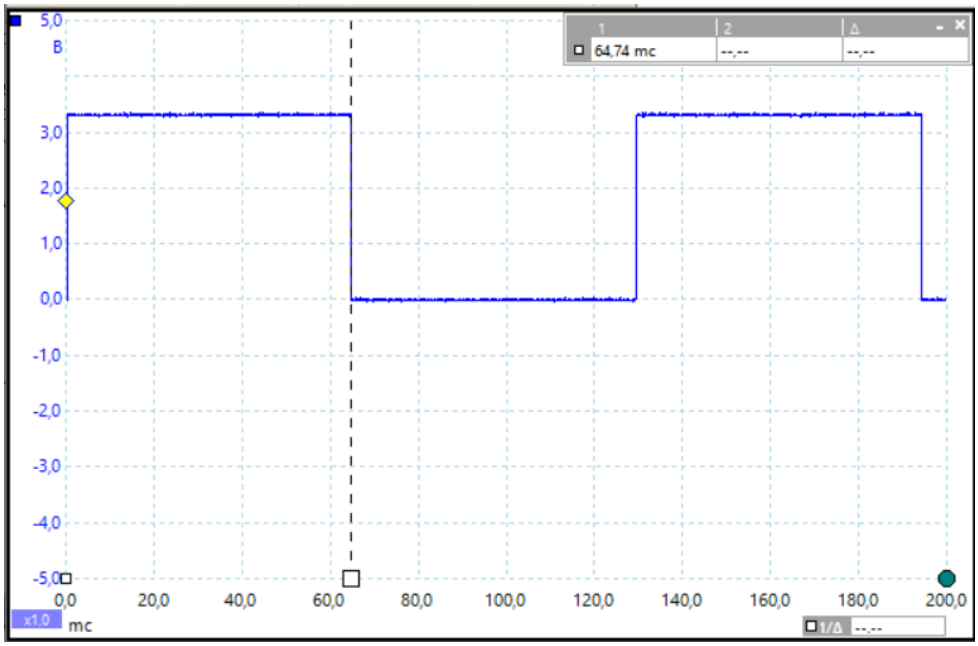


Рисунок 6. Сигнал на выводе PB7 (DC) при тактовой частоте 170 MHz

Базовый проект с подключенным внешним кварцевым генератором и увеличенной тактовой частотой процессора до 170 MHz содержится в папке 02_HSE.

4. Таймер TIM2

Для работы радиоприемника нам необходимо подать на ADC внутренний тактовый сигнал дискретизации, частота и стабильность которого в конечном счете будут определять точность настройки на радиостанцию. Обычно генератор тактовой частоты реализуется с помощью одного из таймеров контроллера. Будем использовать таймер общего назначения TIM2.

Для демонстрации работы настоим таймер на частоту 2 Гц (период 500 мс). Для этого нам необходимо записать в регистр TIM2_ARR (Auto-reload register) число K, которое определяет количество периодов входной тактовой частоты таймера (170 MHz), помещающихся в одном периоде выходной частоты таймера. Более точно: Fвых = Fтакт/(K+1). Если нам необходима частота 2 Гц, то коэффициент K должен быть равен 85000000 – 1. По умолчанию таймер работает в режиме прямого циклического счета (считает от 0 до K). При достижении значения K, счетчик таймера сбрасывается в 0, генерируется аппаратный сигнал "Counter overflow" и счет вновь повторяется от 0 до K. В нашем случае сигнал "Counter overflow" будет вырабатываться каждые 500 мс.

Настроим таймер TIM2 таким образом, чтобы при генерации сигнала "Counter overflow" вызывался обработчик прерывания - функция TIM2_IRQHandler, в которой будем одновременно переключать значение сигналов на выводах PB7 (Test) и PB8 (LD2) контроллера.

Все описанные действия выполняются в функциях, расположенных в файле «tim2.c» (проект 03_TIM2).

```
void TIM2_Init(void)
{

  SET_BIT(RCC_APB1ENR1, RCC_APB1ENR1_TIM2EN); // Подаем на TIM2 тактовую частоту
  TIM2_ARR = 85000000 - 1;                    // Загружаем Auto-reload register
                                              // f = 170 000 000/(K+1), 2Hz -> (85000000 - 1)

  SET_BIT(NVIC_ISER0, (1 << 28));          // Разрешить в NVIC прерывание #28 (TIM2)
  SET_BIT(TIM2_DIER, TIM_DIER_UIE);        // Разрешить прерывание по переполнению таймера
  SET_BIT(TIM2_CR1, TIM_CR1_CEN);          // Включить таймер
}


void TIM2_IRQHandler(void)
{
       CLEAR_BIT(TIM2_SR, TIM_SR_UIF);       // Сброс флага переполнения
       INVERT_BIT(GPIOB_ODR, GPIO_ODR_OD7);  // Инвертировать (PB7)
       INVERT_BIT(GPIOB_ODR, GPIO_ODR_OD8);  // Инвертировать LD2 (PB8)

}
```

Особо следует отметить процедуру подключения обработчика прерывания для таймера TIM2. По таблице 97 (STM32G4 Series vector table) документа *"RM0440 Reference manual STM32G4 Series advanced Arm-based 32-bit MCUs"* находим, что прерывание для устройства TIM2 имеет номер 28. Для настройки прерывания необходимо выполнить следующие действия:

1. Разрешить генерацию прерывания от устройства TIM2 (#28) в функциональном модуле NVIC (Nested vectored interrupt controller).

2. Поместить адрес функции обработчика прерывания в таблицу векторов прерывания (массив uint32_t *vectors[] в файле «vectors.c») в элемент массива под номером 28+16.

3. Разрешить прерывание по сигналу "Counter overflow" в регистре TIM2_DIER (DMA/Interrupt enable register).

После компиляции и загрузки программы в контроллер сигнал на выводе PB7 должен иметь следующий вид:

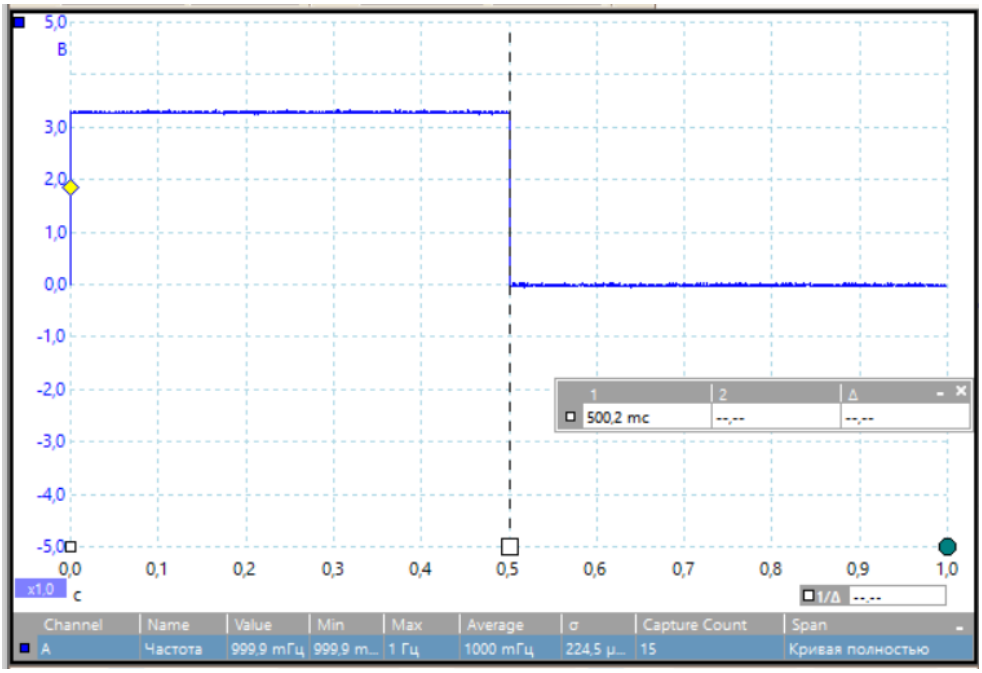


Рисунок 7. Сигнал на выводе PB7 (DC) при K = 85000000 - 1

5. DAC

В следующем примере подключим к проекту цифро-аналоговый преобразователь (DAC) и будем выводить в него данные по прерыванию от TIM2.

Проект программы находится в папке 04_TIM2_DAC. В данный проект добавлен файл «dac.c».

```
//Выход DAC1 на PA4

void DAC1_Init(void)
{

	SET_BIT(RCC_AHB2ENR, RCC_AHB2ENR_DAC1EN );  // Подать тактовый сигнал на DAC1
	SET_BIT(DAC1_CR, 1);                        // Включить выход DAC1
	DAC1_DHR12R1 = 0;                           // Записать 0 в регистр данных DAC1

}
```

Файл содержит одну короткую функцию инициализации модуля DAC1. В данном случае нам даже не нужно настраивать вывод PA4, так как он настроен на аналоговый режим по умолчанию.

Будем выводить информацию в DAC по прерыванию от TIM2. Для этого смодифицируем файл «tim2.c».

```
void TIM2_Init(void)
{

  SET_BIT(RCC_APB1ENR1, RCC_APB1ENR1_TIM2EN); // Подаем на TIM2 тактовую частоту
  // Загружаем Auto-reload register  f = 170 000 000/(TIM2_ARR + 1) (1 MHz -> 170 - 1)
  TIM2_ARR = 170 - 1;
  SET_BIT(NVIC_ISER0, (1 << 28));          // Разрешить в NVIC прерывание #28 (TIM2)
  SET_BIT(TIM2_DIER, TIM_DIER_UIE);        // Разрешить прерывание по переполнению таймера
  SET_BIT(TIM2_CR1, TIM_CR1_CEN);          // Включить таймер
}


void TIM2_IRQHandler(void)
{

  // Переменная I увеличивается на 1 при каждом вызове обработчика прерывания
  static int   I=0;

  CLEAR_BIT(TIM2_SR, TIM_SR_UIF);   // Сброс флага переполнения
  DAC1_DHR12R1 = I++;  // Запись в регистр данных DAC 12-ти младших разрядов переменной I

}
```

Устанавливаем период генерации таймера TIM2 равным 1 мкс. Модифицируем обработчик прерывания TIM2_IRQHandler. Добавляем в файл «main.c» вызов функции инициализации DAC1 - DAC1_Init(). После компиляции и загрузки программы 04_TIM2_DAC получаем на выходе DAC (вывод PA4) следующий сигнал (рисунок 8).

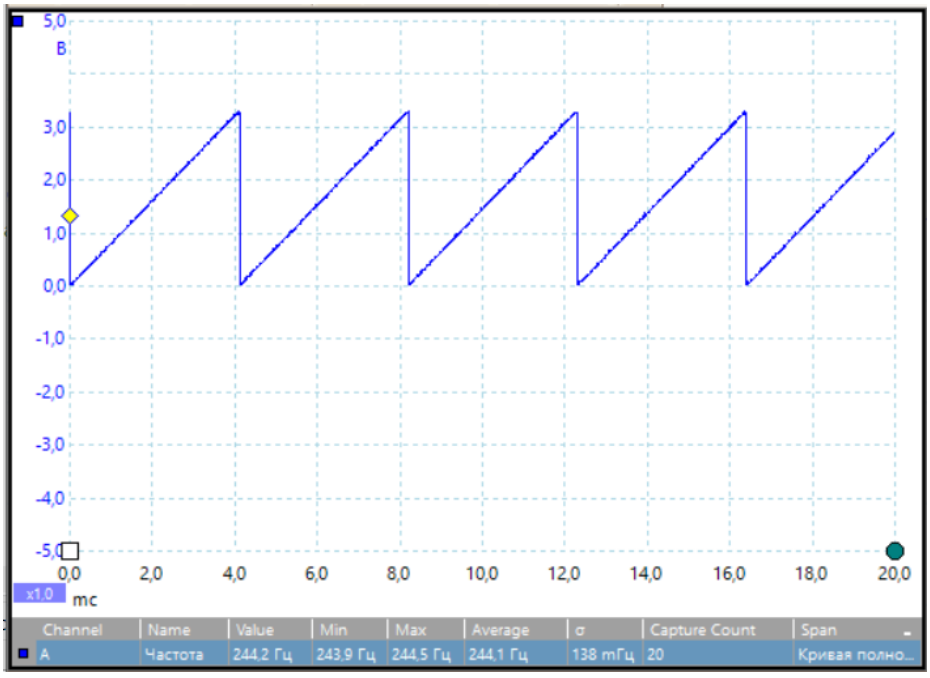


Рисунок 8. Сигнал на выходе DAC (DC), период таймера TIM2 - 1 мкс

Частота полученного пилообразного сигнала равна 244.1 Hz (колонка Average внизу картинки). Разрядность DAC равна 12 битам, то есть устройство может вывести 4096 уровней напряжения в диапазоне от 0 до 3,3 вольт. При записи в 32-х разрядный регистр данных DAC1_DHR12R1, старшие разряды с 12 по 31 отбрасываются. Таким образом в обработчике прерывания циклически перебираются значения DAC1_DHR12R1 от 0 до 4095 с интервалом одного шага в 1 мкс. Период полного перебора равен 4096 мкс, то есть частота равна 1/0.004096 = 244.14 Hz, что согласуется с измерением.

6. Генератор синусоидального сигнала

Теперь попробуем создать генератор синусоидального сигнала.

Различные колебательные системы: механические, электрические, тепловые, гидравлические, и т.д., описываются идентичными математическими выражениями. Как писали классики: «Единство природы обнаруживается в поразительной аналогичности дифференциальных уравнений, относящихся к разным областям явлений».

Поэтому для создания генератора будем использовать модель пружинного маятника, как наиболее наглядную.

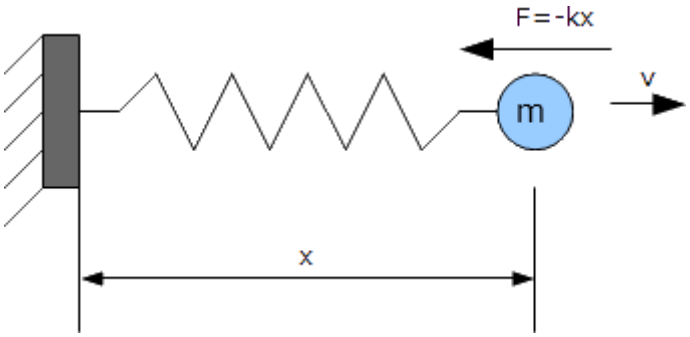


Рисунок 9. Движение пружинного маятника

Из курса физики средней школы известно, что ускорение равно:

$$a = F/m \quad (1).$$

По закону Гука:

$$F = - kX \quad (2).$$

Подставив второе выражение в первое получим:

$$a = - (k/m)*X.$$

Обозначим k/m, как K. В этом случае предыдущее выражение примет вид:

$$a = - K*X \quad (3).$$

Ускорение - это вторая производная координаты по времени, в связи с чем выражение (3) можно записать:

$$X''(t) = - K*X(t), \text{ или } X''(t) + K*X(t) = 0 \quad (4).$$

Уравнение (4) является дифференциальным уравнением второго порядка, частным решением которого является функция:

$$X(t) = A*Sin(2\pi*f*t) \quad (5),$$

где A – амплитуда, f – частота. Действительно, если взять вторую производную от выражения (5), получим:

$$X''(t) = - A*(2\pi*f)^2*Sin(2\pi*f*t) \quad (6).$$

Подставив (5) и (6) в (4) получим:

$$- A*(2\pi*f)^2*Sin(2\pi*f*t) = -K* A*Sin(2\pi*f*t).$$

Сократив справа и слева одинаковые сомножители получим:

$$(2\pi*f)^2 = K \quad (7),$$

То есть, при $K = (2\pi*f)^2$ выражение (7) является тождеством. Таким образом пружинный маятник колеблется по синусоидальному закону, при этом частота колебаний на основании выражения (7) равна:

$$f = \frac{\sqrt{K}}{2\pi} \quad (8).$$

Для моделирования пружинного маятника, заменим в выражении (4) вторую производную X″(t) дискретным эквивалентом. Сделаем это следующим образом.

По определению производной:

$$f'(t) = \lim_{\Delta t \to 0} \frac{\Delta f}{\Delta t} .$$

Если взять достаточно малое $\Delta t = (t^+ - t)$, то

$$f'(t) \approx \frac{\Delta f}{\Delta t} , \text{ или}$$

$$f'(t) \approx f \text{ , где } t^+ \text{ - это следующий отсчет времени.}$$

То есть для выражения X″(t) = -K*X(t) можно записать :

$$((X(t^+) - X(t))/ds) - (X(t) - X(t^-))/ds))/ds \approx -K*X(t) \quad (9),$$

где t - текущий отсчет времени, t^+ - следующий отсчет времени, t^- - предыдущий отсчет времени, ds - шаг дискретизации. Преобразовав выражение (9), можно записать:

$$(X(t^+) - X(t)) - (X(t) - X(t^-)) = -K*ds^2*X(t) \quad (10),$$

Обозначим $V1 = (X(t^+) - X(t))$ и $V0 = (X(t) - X(t^-))$. При этом выражение (10) примет вид:

$$V1 = V0 - K*ds^2*X(t) \quad (11).$$

Из формулы $V1 = (X(t^+) - X(t))$ следует, что:

$$X(t^+) = X(t) + V1 \quad (12).$$

Выражения (11) и (12) можно использовать для генерации дискретного эквивалента функции $A*Sin(2\pi*f*t)$ «налету», итерационно. Если обозначить $R = K*ds^2$, то эквивалентом выражений (11) и (12) на языке С будет запись:

```
V -= X*R;
X += V;
```

При этом на основании выражения (7) R можно вычислить по формуле:

$$R = (2\pi f \times ds)^2 \quad (13).$$

где f – частота генерируемого сигнала в герцах, ds – период дискретизации в секундах.

Изменим код обработчика прерывания в файле «tim2.c»:

```
#define ds  0.000001      // Период дискретизации в секундах
#define PI  3.1415926     // Число ПИ
#define PW  0.031415926   // PI*f*ds при f = 10000 Hz


void TIM2_IRQHandler(void)
{

    static float R = 4 * PW * PW; //R=(2*PW)^2
    static float V=0;
    static float X = 4096/3.3 * 0.5; //Начальное значение - амплитуда сигнала 0.5 V
    static float S=1500; //Смещение DAC

    CLEAR_BIT(TIM2_SR, TIM_SR_UIF);  //Сброс флага прерывания по переполнению таймера

    //Осциллятор
    V -= X*R;
    X += V;

    DAC1_DHR12R1 = X + S;

}
```

Здесь мы используем математику с плавающей запятой, поэтому нам необходимо включить сопроцессор FPU. Это делается добавлением одной строчки в функцию start_up() файла «system_init.c»:

```
    /* Запуск FPU */
    SCB_CPACR |= ((3 << (10*2))|(3 << (11*2)));  /* set CP10 and CP11 Full Access */
```

DAC поддерживает только положительные целые значения от 0 до 4095. Переменная X принимает значения приблизительно от -620.0 до +620.0. Для того чтобы перенести значения X в положительную область, перед выводом в DAC будем прибавлять к X смещение S, равное 1500.

После компиляции и загрузки программы на выходе DAC можно наблюдать следующий сигнал (рисунок 10).

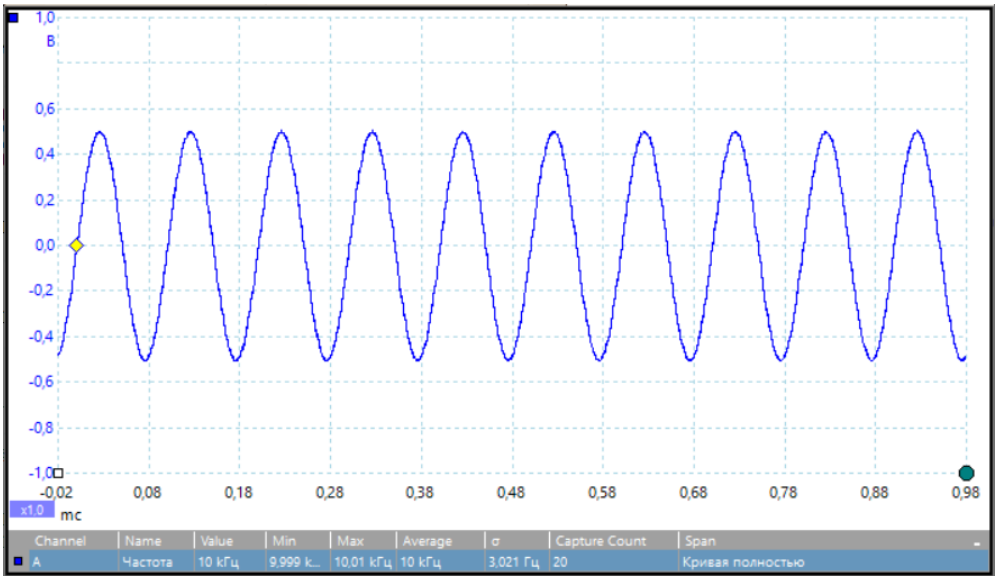


Рисунок 10. Сигнал на выходе DAC, период таймера TIM2 - 1 мкс, f = 10 KHz, THD = 0,05%

К сожалению, на частотах, приближающихся к частоте дискретизации, формула (13) перестает давать точный результат для вычисления R. Это объясняется тем, что замена производной X″(t) разностным уравнением (9) становиться достаточно грубым приближением. Например, на рисунке 11 приведена осциллограмма выходного сигнала при значении f в формуле (13) равной 200 KHz. Как видно из рисунка, реальный сигнал имеет частоту 216 KHz.

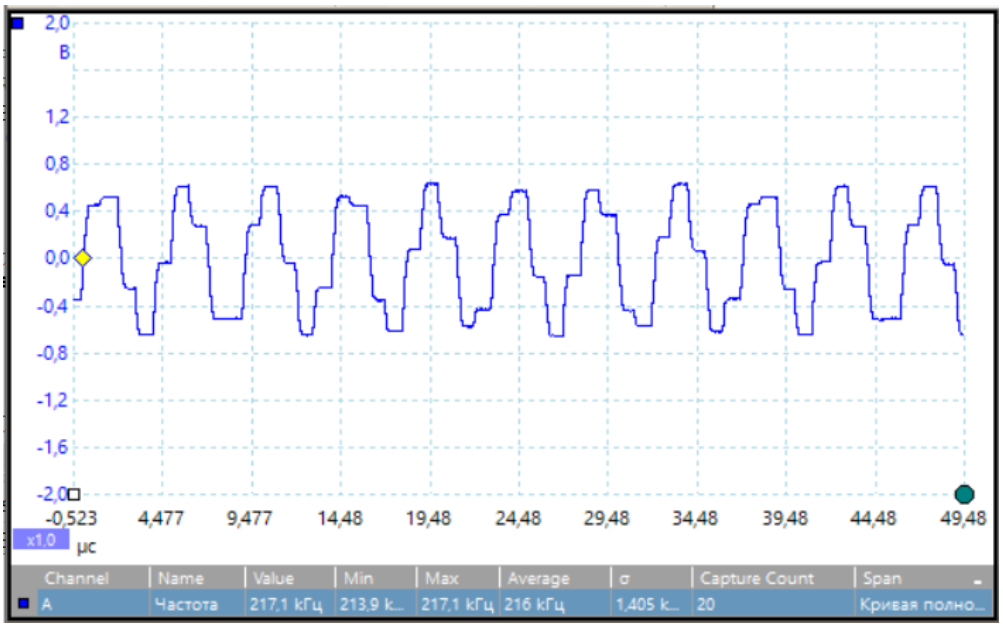


Рисунок 11. Сигнал на выходе DAC, период таймера TIM2 - 1 мкс, f = 200 KHz

Более точная формула для вычисления величины R в зависимости от частоты имеет вид:

$$R = (2*\sin(PI*W))^2 \quad (14),$$

где W - относительная частота равная f*ds. При этом W должна быть меньше 0.5 (частота Найквиста).

Проект программы по генерации синусоидального сигнала находится в папке 05_TIM2_DAC_SIN.

7. Подключение ADC

Блок-схема экспериментальной программы работы с АЦП достаточно проста: активируется ADC1 и DAC1, таймер TIM2 используется в качестве генератора частоты дискретизации для ADC, по прерыванию от ADC (окончание преобразования) сигнал с выхода ADC записывается в DAC.

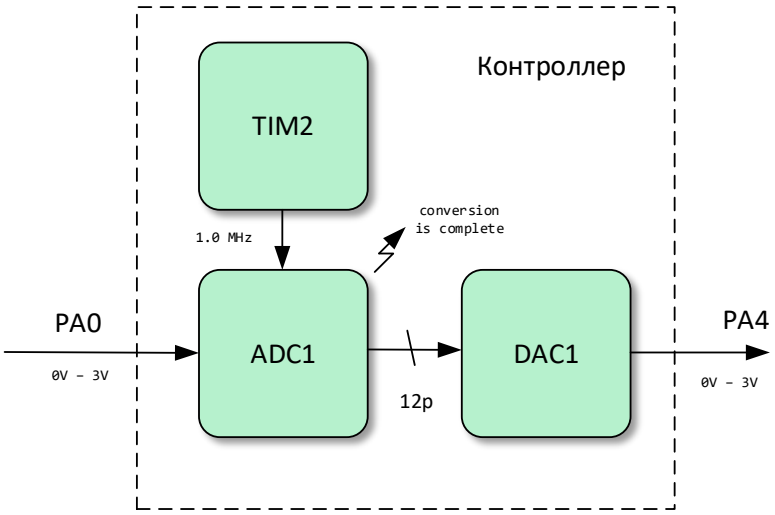


Рисунок 12. Структурная схема программы ADC -> DAC

Проект программы находится в папке 04_ADC_DAC. В проект добавлен файл «adc.c» , который состоит из двух функций: функции инициализации, и функции обработчика прерывания от ADC.

```
//Вход ADC1 на PA0
void ADC1_init()
{

  //Настройка тактирования ADC
  SET_BIT(RCC_CCIPR, 1 << 29 ); //Выбрать системный clock в качестве тактового сигнала для ADC
  SET_BIT(RCC_AHB2ENR, 1 << 13 ); // Подать тактовый сигнал на модуль ADC12
  SET_BIT(ADC12_COMMON_CCR, 2 << 16 ); // Выбрать в качестве тактового сигнала   adc_hclk/2

  //Настройка режима ADC
  ADC1_CR = 0;    // Сброс всех бит в ADC1_CR (ADEN = 0 для обеспечения конфигурирования)
  SET_BIT(ADC1_CR, 1 << 28);  // Включить  опорное напряжение ADC

  // Запуск преобразования от TIM2_TRG0 (канал 11)(биты 5- 11), перезапись разрешена (бит 12)
  ADC1_CFGR = (1 << 12)+(1 << 10)+ (11 << 5);
  ADC1_SQR1 = (1 << 6);   // Выбрать входной канал 1 (PA0)

  //Настройка прерывания
  ADC1_IER = (1 << 3);   // Разрешить прерывание по окончанию преобразования в ADC
  SET_BIT(NVIC_ISER0, (1 << 18));       // Разрешить в NVIC прерывание #18 (ADC1)

  SET_BIT(ADC1_CR, (1 << 0) | (1 << 2)); // Включить ADC1 и начать преобразование
  SET_BIT(TIM2_CR1, TIM_CR1_CEN); //Запуск TIM2 - старт генерации сигнала дискретизации для ADC1

}

void ADC1_IRQHandler(void)
{
    SET_BIT(ADC1_ISR, (1 << 3)); // Сброс флага переполнения в ADC1
    DAC1_DHR12R1 = ADC1_DR;      // Записать данные из ADC в DAC

}
```

Как и в случае с DAC, специально настраивать вывод PA0 нет необходимости, так как аналоговый режим выбирается по умолчанию после сброса контроллера. Настройка прерывания выполняется таким же образом, как для таймера TIM2 в предыдущем примере. Прерывание происходит по окончании процесса преобразования текущего отсчета в ADC.

Некоторым изменениям подвергся файл «tim2.c». Вместо генерации прерывания теперь таймер работает в режиме мастера, то есть выдает сигнал “Counter overflow” на внутренние линии

контроллера для использования в качестве триггера старта преобразования в ADC. Таким образом событие "Counter overflow" задает частоту дискретизации (Fd) входного сигнала.

Если собрать исследовательский стенд в соответствие с рисунком 1 и подать переменное напряжение от генератора на вход ADC, то на выходе DAC можно увидеть сигнал, представленный на рисунке 13. На всех осциллограммах данного раздела вход осциллографа работал в режиме AC.

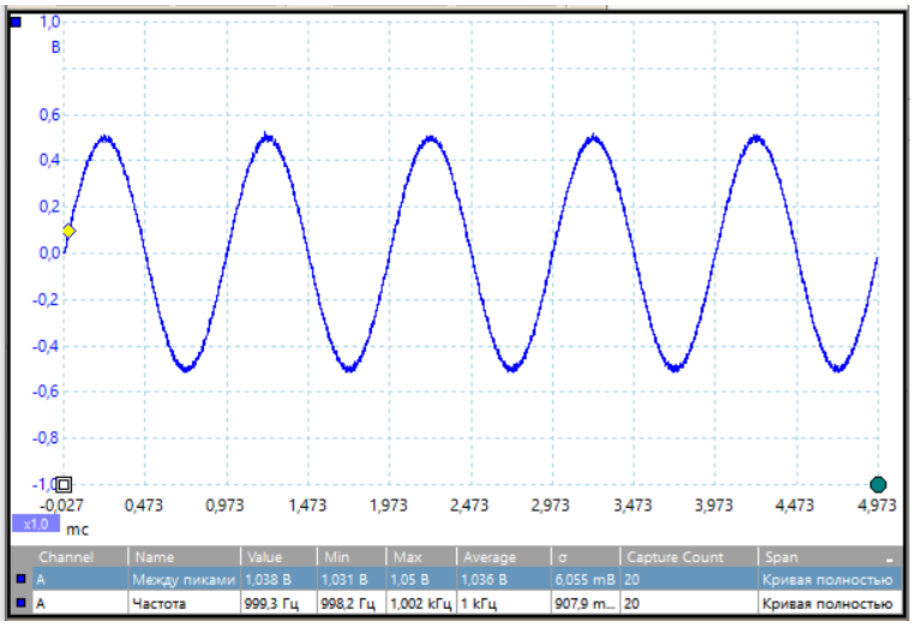


Рисунок 13. Сигнал на выходе DAC, частота входного сигнала ADC 1 KHz, амплитуда 0,5 V

При увеличении частоты генератора картинка будет меняться, на частоте 200 KHz будут отчетливо видны интервалы дискретизации сигнала (рисунок 14).

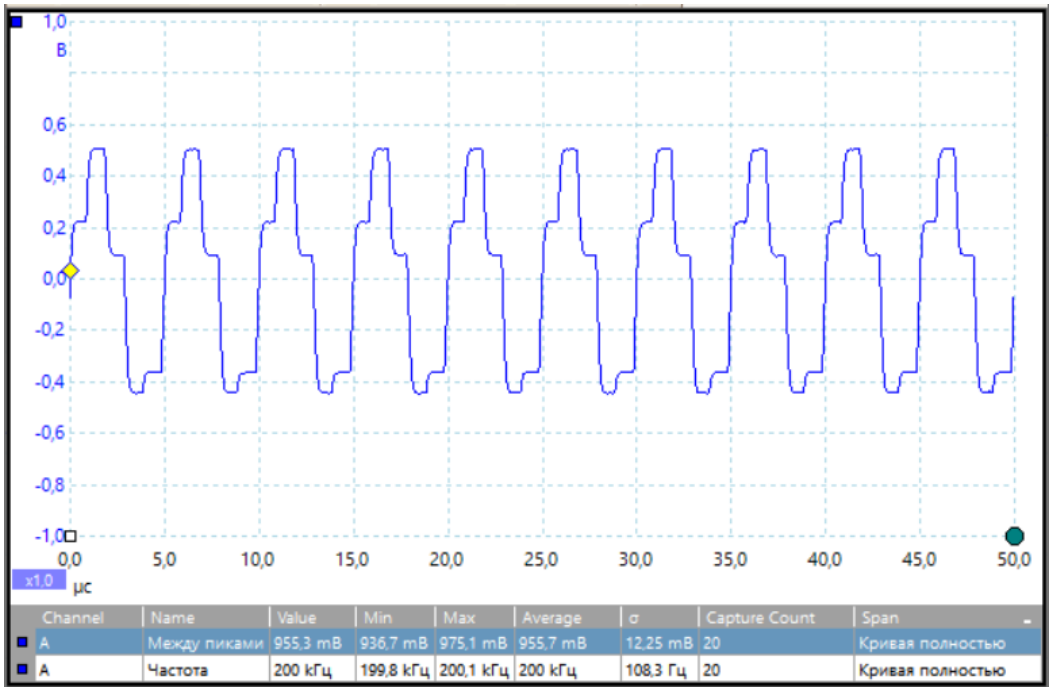


Рисунок 14. Сигнал на выходе DAC, частота входного сигнала ADC 200 KHz, амплитуда 0,5 V

При увеличении частоты входного сигнала до значения Fn = Fd/2 = 500 KHz (частота Найквиста/Котельникова) сигнал на выходе DAC примет следующий вид (рисунок 15).

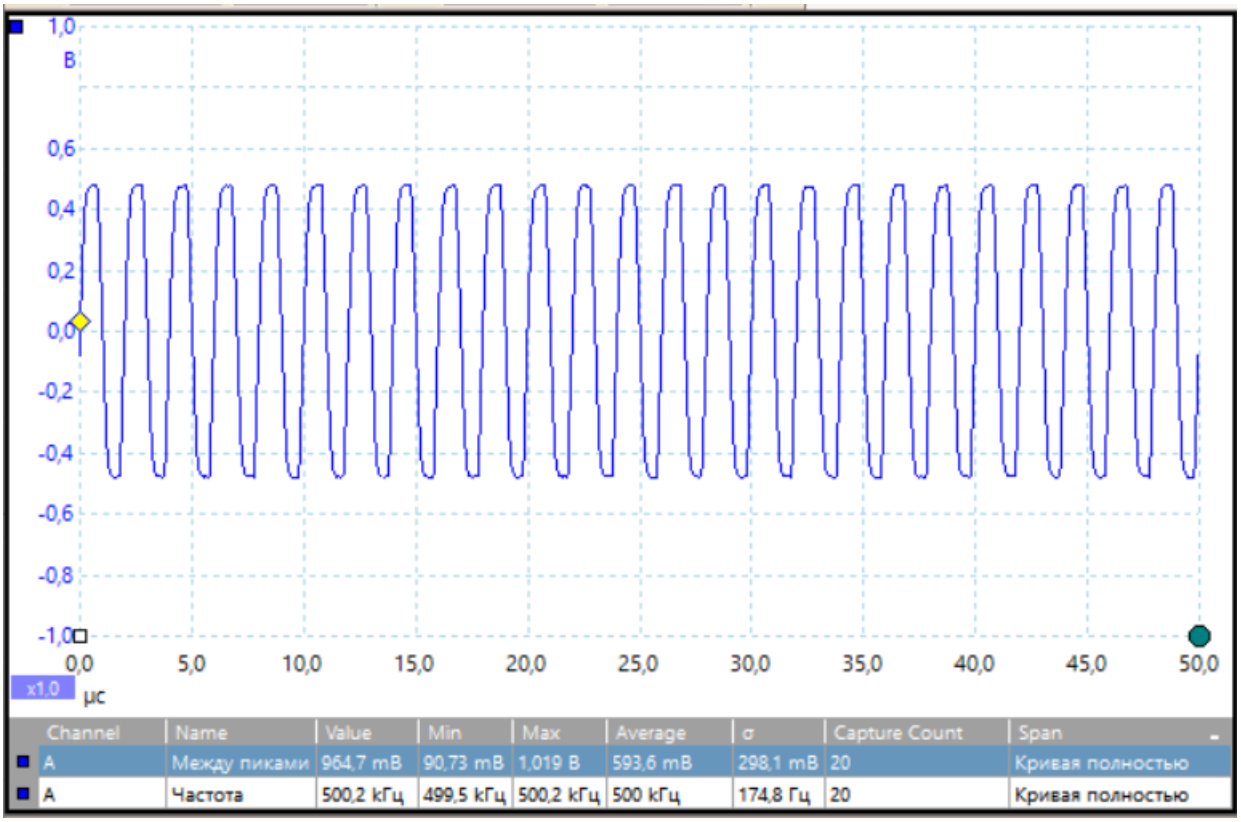


Рисунок 15. Сигнал на выходе DAC, частота входного сигнала ADC 500 KHz, амплитуда 0,5 V

Это максимальная частота входного синусоидального сигнала, для которой выходной сигнал будет сохранять точную информацию о частоте. При дальнейшем увеличении частоты входного сигнала Fin частота выходного сигнала Fout будет изменяться в соответствии с графиком, представленным на рисунке 16.

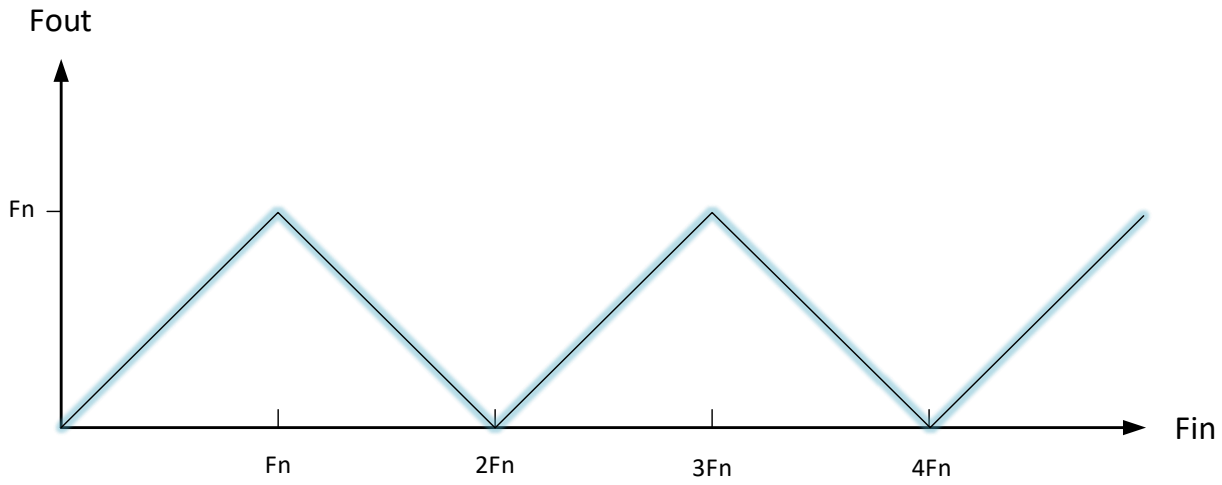


Рисунок 16. Частота сигнала на выходе DAC (Fout) при увеличении

частоты сигнала на входе ADC (Fin) больше чем Fn

График соответствует следующим выражениям:

Если Fin % Fd < Fn, то Fout = Fin % Fn; (15)

Если Fin % Fd >= Fn, то Fout = Fn – (Fin % Fn); (16)

где % - остаток от деления; Fin - частота входного сигнала, подаваемого на ADC; Fout - частота выходного сигнала с DAC; Fd - частота дискретизации; Fn - частота Найквиста, равная Fd/2.

В качестве иллюстрации можно привести осциллограмму, изображенную на рисунке 17. В данном случае частота входного сигнала равна 3900 KHz, частота дискретизации -1000 KHz и соответственно частота Найквиста - 500 KHz. Осциллограф показывает частоту на выходе приблизительно 100 KHz. Проверяем условие Fin % Fd = 3900 % 1000 = 900. Полученный результат больше частоты Найквиста, поэтому выбираем выражение (16). Подставив значения в формулу (16), мы получим Fout = 500 – (3900 % 500) = 500 – 400 = 100 KHz.

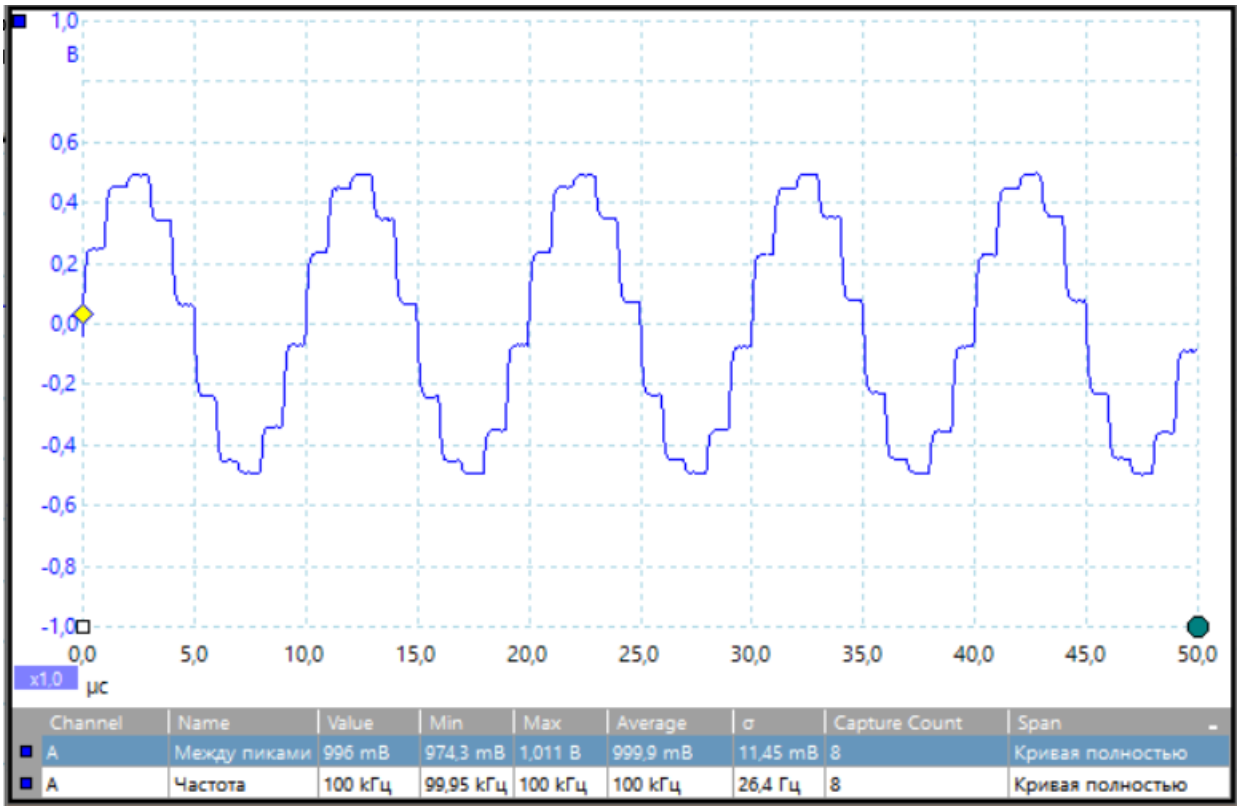


Рисунок 17. Сигнал на выходе DAC, частота входного сигнала ADC 3900 KHz, амплитуда 0,5 V

Максимальное значение частоты входного сигнала определяется частотными характеристиками блока выборки и хранения ADC. Измерения показывают, что схема выборки и хранения контроллера STM32G431KBT6 работает вполне удовлетворительно вплоть до частот порядка 100 MHz. Например, так выглядит осциллограмма сигнала с частотой 65.1 MHz (рисунок 18).

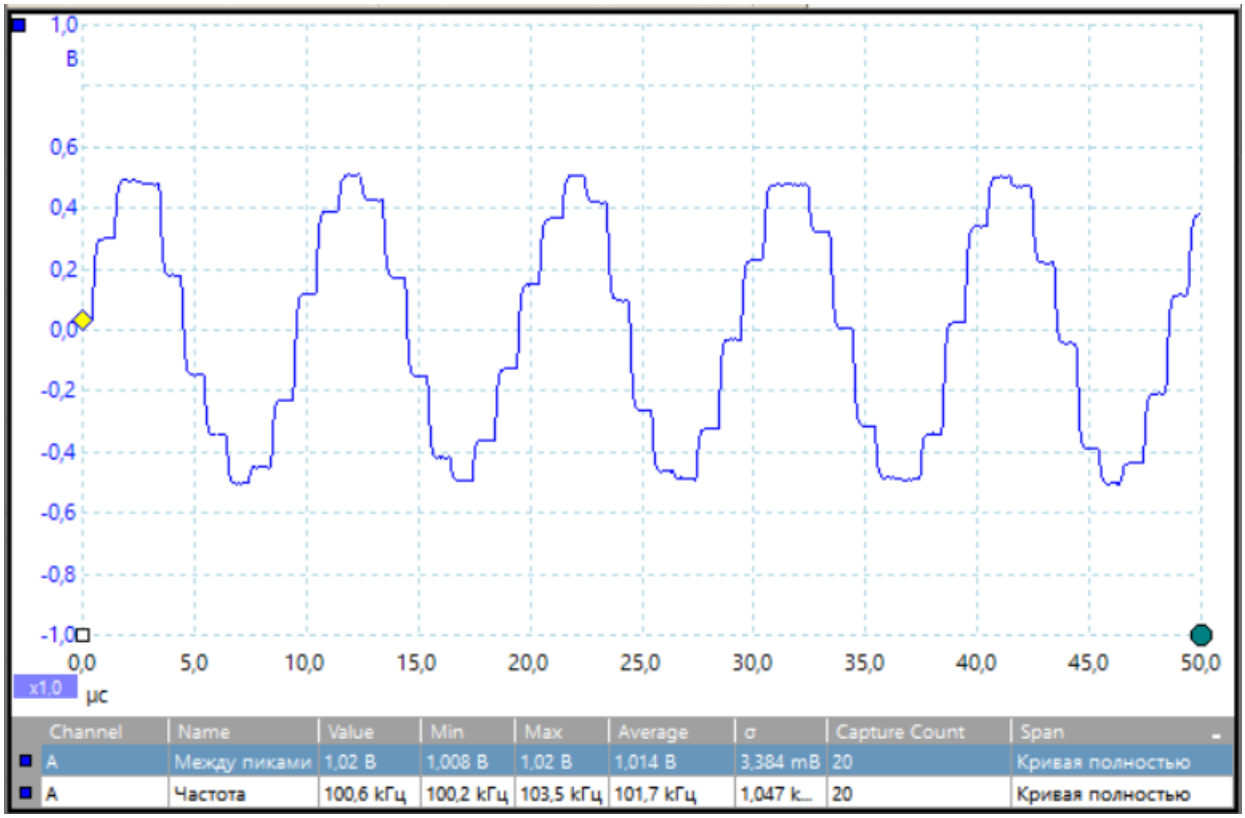


Рисунок 18. Сигнал на выходе DAC , частота входного сигнала 65.1 MHz, амплитуда 0,5 V

Данное обстоятельство определяет два следствия: 1) Входные цепи приемника, подключенные к ADC, нужно тщательно оберегать от просачивания высокочастотных внедиапазонных сигналов. 2) При соответствующем проектировании входных цепей радиоприемника можно обеспечить прием сигналов радиостанций SW и даже FM диапазонов (!).

8. Перестраиваемый полосовой цифровой фильтр

Структурная схема проекта для исследования фильтра представлена на рисунке 19.

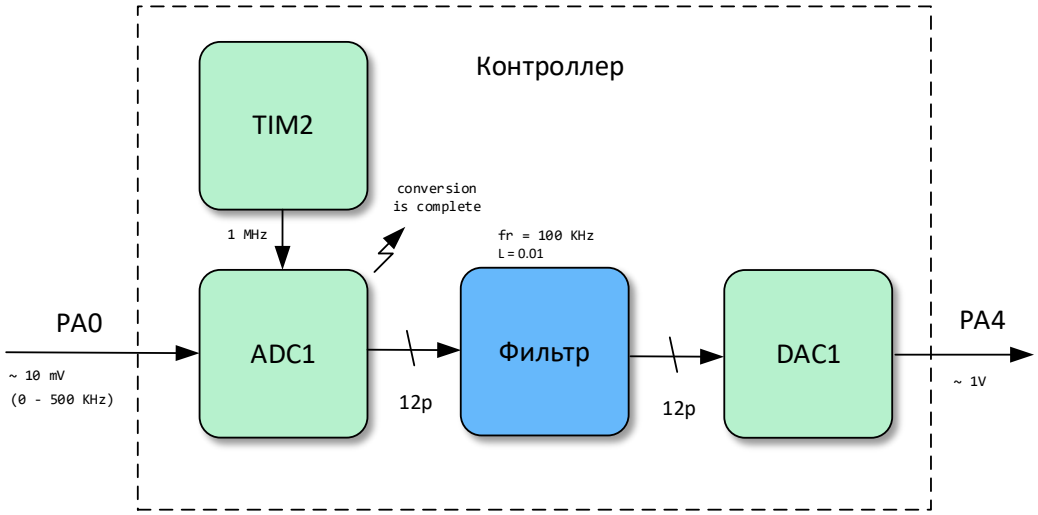


Рисунок 19. Структурная схема программы ADC -> Фильтр –> DAC

Мы будем использовать простейший цифровой фильтр, построенный на основе синусоидального генератора, описанного в разделе 6. Для того чтобы математическую модель идеального осциллятора, реализованного на основе пружинного маятника, превратить в фильтр, необходимо в модель осциллятора ввести потери. Поступим следующим образом - на каждом шаге вычисления нового значения X будем уменьшать это значение на некоторую небольшую величину, которая зависит от предыдущего значения X:

```
X += ADC;
V-=X*R;              (17)
X+=V – X*L.
```

Потери определяются величиной L. Чем меньше значение L, тем меньше потери.

Перед выполнением очередного вычисления новых величин V и X необходимо прибавить значение выходного сигнала от ADC к X.

Интересно отметить, что для полученного фильтра полоса пропускания dF практически не зависит от частоты fr (частоты резонанса фильтра) и по уровню 0.7 приблизительно равна величине:

$$dF \approx 0.18 * fd * L,$$

где fd – частота дискретизации. Это очень ценное свойство для перестраиваемых по частоте полосовых фильтров.

Фильтр обеспечивает «усиление» сигнала, т.е. ведет себя подобно обычному колебательному контуру. Выходной сигнал можно определить по формуле:

$$U_{вых} \approx U_{вх}/L.$$

В данном случае величина 1/L аналогична добротности Q обычного LC колебательного контура.

Обозначим Wr = fr*ds. Более точная формула при Wr > 0.1:

$$U_{вых} \approx U_{вх}(1 + Sin(PI*Wr)^2)/L$$

Обобщая сказанное, можно записать:

$R \approx (2*Sin(PI*Wr))^2$ (настройка частоты фильтра, погрешность не больше 0.5%);

$dF \approx 0.18 * fd * L$ (настройка полосы пропускания фильтра);

Uвых/Uвх ≈ (1 + Sin(PI*Wr)^2)/L (определение коэффициента передачи);

Проект с примером фильтра находится в папке 07_ADC_DAC_FIL. Параметры фильтра следующие:

- Частота дискретизации fd = 1 MHz (ds = 1 мкс);
- Центральная частота полосы пропускания фильтра fr = 100 KHz (Wr = 0.1);
- Потери L = 0.01.

Рассчитываем по формулам:

- R для центральной частоты полосы пропускания Wr = 0.1, R = 0,3819660;
- полоса пропускания dF = 0.18*1000000*0.01 = 1800 Hz;
- коэффициент передачи Uвых/Uвх = (1 + Sin(PI*0.1)^2)/0.01 = 110.

Код фильтра находится в функции-обработчике прерывания ADC1_IRQHandler файла «adc.c».

```
void ADC1_IRQHandler(void)
{

    //Параметры фильтра на частоту 100 KHz
    //R ≈ (2*Sin(PI*Wr))^2, где Wr = Fr/Fd, Fr = 100 KHz, Fd = 1000 KHz
    static float R = 0.381965999;
    static float L = 0.01;

    //Начальные значения
    static float X = 0;
    static float V = 0;

    //Смещение ADC и DAC
    static float   S = 1900;

    SET_BIT(ADC1_ISR, (1 << 3)); // Сброс флага переполнения в ADC1

    //Фильтр
    X += (int)ADC1_DR - S;
    V -= X * R;
    X += V - X * L;

    //Запись в DAC
    DAC1_DHR12R1 = X + S;

}
```

Функция **ADC1_IRQHandler** вызывается с интервалом 1 мкс, т.е. в данном случае частота дискретизации fd равна 1 MHz.

После загрузки программы в контроллер измеряем характеристики сигнала на выходе DAC.

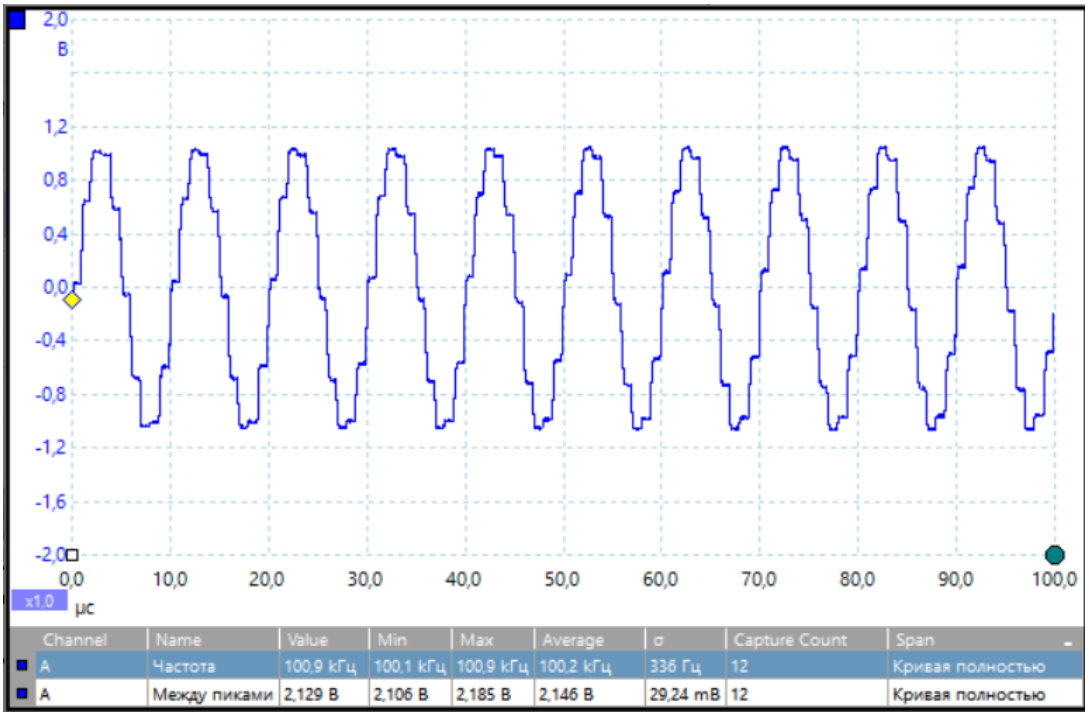


Рисунок 20. Сигнал на выходе DAC на резонансной частоте фильтра 100.2 KHz (расчет fr = 100KHz, L = 0.01), амплитуда входного сигнала 10 mV, амплитуда выходного сигнала 1.07 V (расчет 1.1 V)

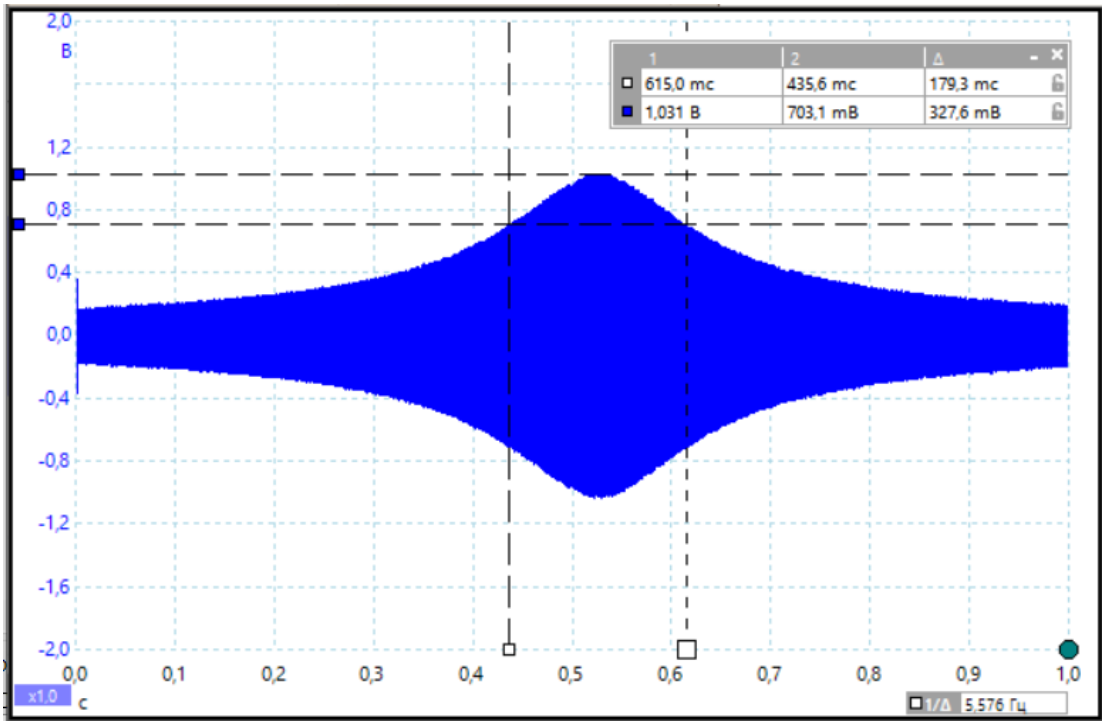


Рисунок 21. Амплитудно-частотная характеристика фильтра L = 0.01. Девиация частоты 10KHz (95 – 105 KHz), Ширина полосы пропускания (по уровню 0.7) 1.8 KHz (расчет 1.8 KHz)

Как видно из диаграмм, измеренные характеристики фильтра совпадают с расчетными величинами с достаточной для практического применения точностью. Заметим, что данный фильтр сохраняет

работоспособность до частот, достигающих значений Wr = 0,47, а для частот Wr <0,3 обеспечивает приемлемый уровень шумов.

Если потери L установить равным 0.001, мы получим результат, приведенный ниже.

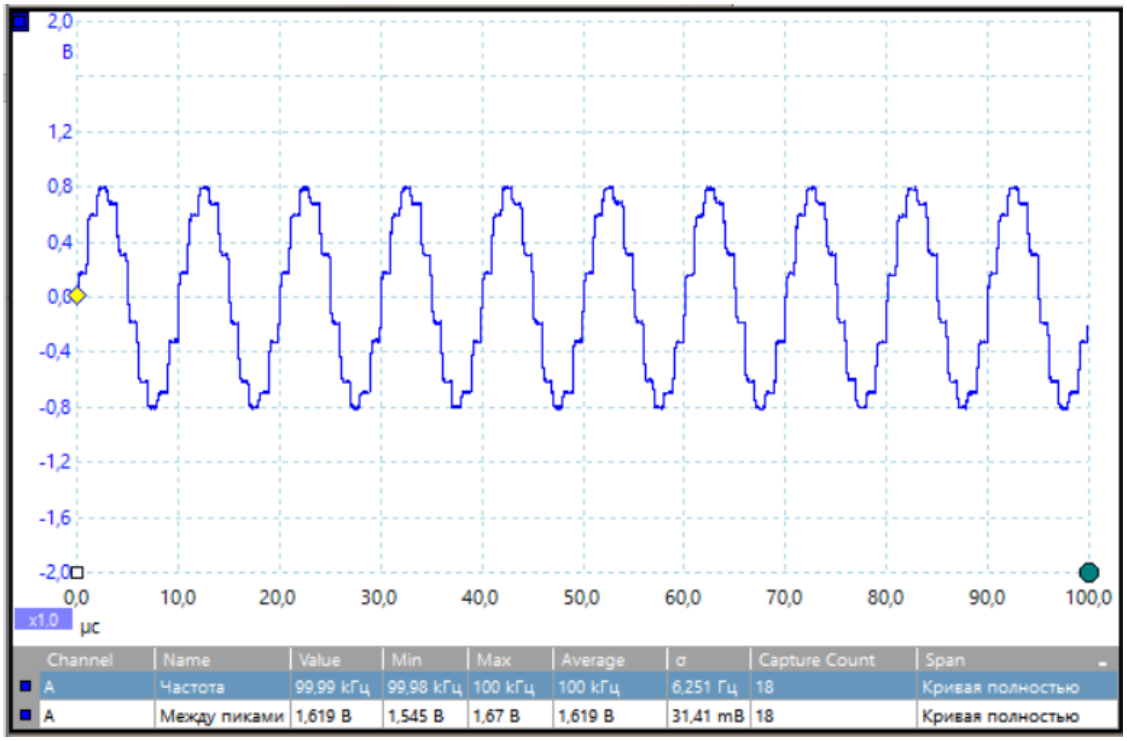


Рисунок 22. Сигнал на выходе DAC на резонансной частоте фильтра 100 KHz (расчет fr = 100KHz, L = 0.001), амплитуда входного сигнала 1 mV, амплитуда выходного сигнала 0,8 V (расчет 1.1 V)

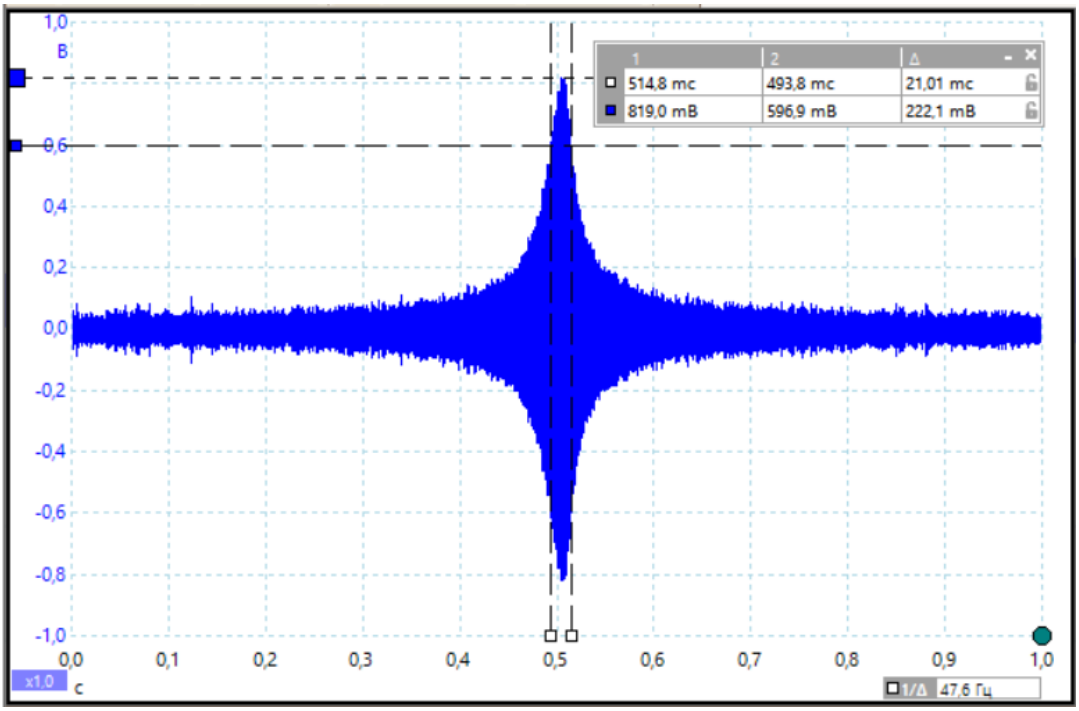


Рисунок 23. Амплитудно-частотная характеристика фильтра L = 0.001. Девиация частоты 10KHz (95 – 105 KHz), ширина полосы пропускания (по уровню 0.7) 210 Hz (расчет 180 Hz)

Мы получили впечатляющий результат: фильтр в связке с 12-ти разрядным АЦП сохраняет работоспособность при амплитуде входного сигнала 1 mV. Данное значение сопоставимо с величиной одного интервала квантования АЦП (3.3V / 4096 = 0.8 mV). При этом фильтр эквивалентен LC контуру с добротность порядка Q=500.

Улучшить параметры фильтрации вне полосы пропускания возможно за счет каскадного соединения нескольких фильтров (выход одного на вход следующего). Для выбранного контроллера STM32G431KBT6 за время интервала дискретизации 1 мкс программе, написанной на C, удается обсчитать два каскада фильтра.

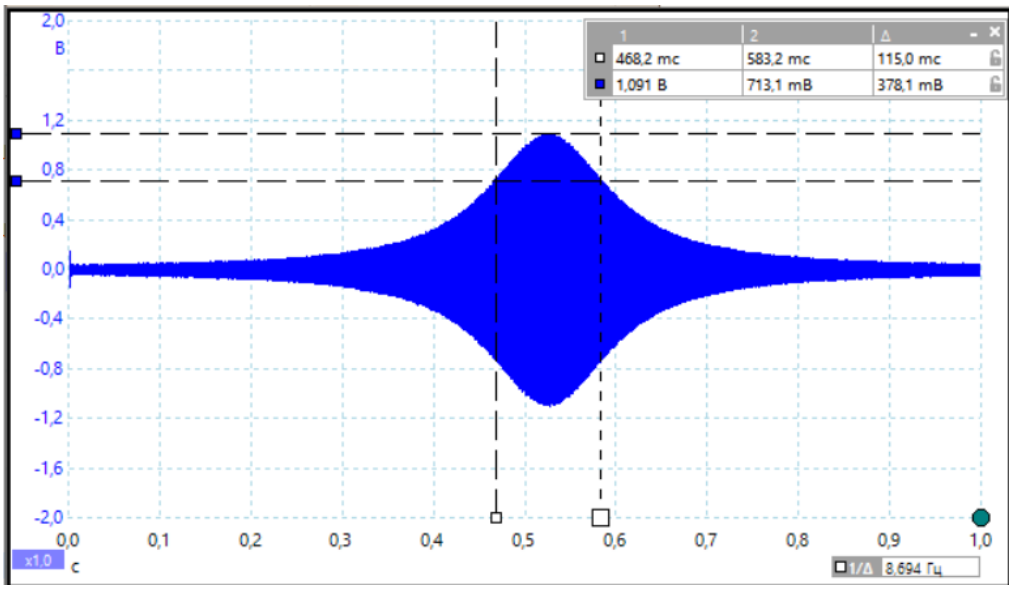


Рисунок 24. Амплитудно-частотная характеристика двухкаскадного фильтра L = 0.01, Fr = 100 KHz, девиация частоты 10KHz (95 – 105 KHz).

Для двухкаскадного фильтра при расстройке частоты входного сигнала на 10KHz происходит уменьшение амплитуды выходного сигнала где-то в 50 раз (33dB), что вполне достаточно для создания радиоприемника среднего уровня.

9. Детектор

Структурная схема проекта для исследования детектора представлена на следующем рисунке.

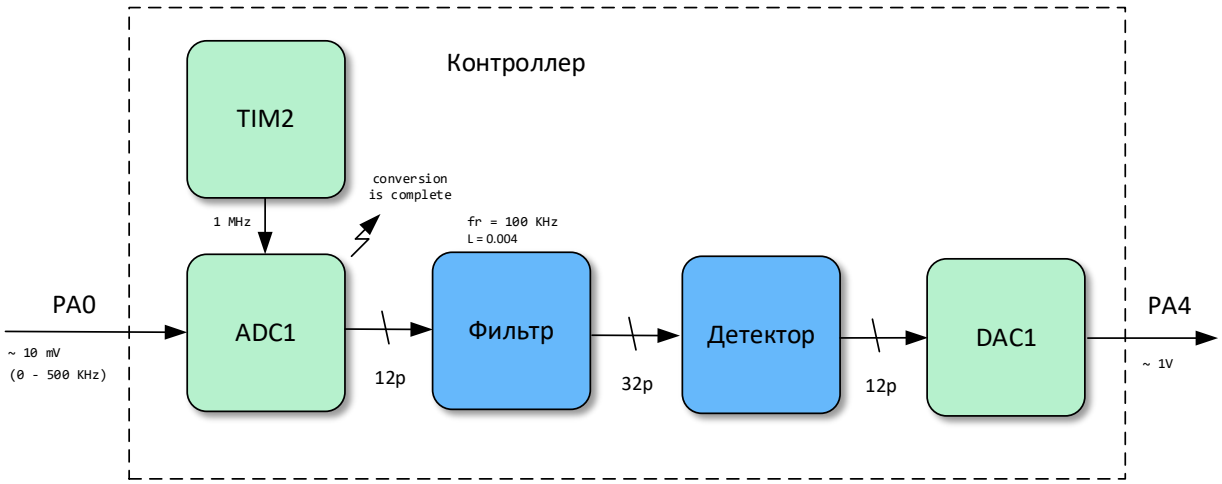


Рисунок 25. Структурная схема программы ADC -> Фильтр -> Детектор –> DAC

Модель детектора содержит функцию вычисления абсолютного значения величины (модуля) и простейшего фильтра низкой частоты.

В качестве модели фильтра низкой частоты будем использовать модель интегрирующей RC цепи.

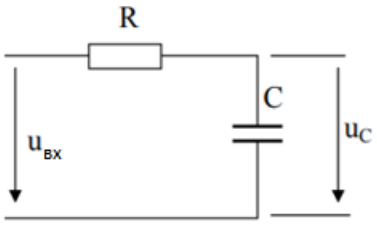


Рисунок 26. Интегрирующая RC цепь

На основании второго закона Кирхгофа можно записать:

$$u_{вх} = u_R(t) + u_C(t), \text{ или } u_{вх} = R \times i(t) + u_C(t), \quad (18)$$

где i(t) - ток в цепи.

Также известно, что

$$i(t) = C \times u'_C(t) \quad (19).$$

Подставив (19) в (18) получим:

$$u_{вх} = RCu'_C(t) + u_C(t).$$

Заменим $u'_C(t)$ на дробь (из определения производной):

$$u'_C(t) \approx \frac{u_C(t1)-u_C(t0)}{t1-t0}$$ или

$$u_{\text{вх}} = RC\frac{u_C(t1)-u_C(t0)}{ds} + u_C(t0)\ , (20)$$

где ds это время дискретизации.

Заменим дробь ds/RC на переменную K. Тогда выражение (20) можно записать:

$$u_C(t1) = u_C(t0) + \left(u_{\text{вх}} - u_C(t0)\right) \times K \qquad (21).$$

«Точка излома -3 дБ» для фильтра низкой частоты (ФНЧ), построенного на основе RC цепочки, достигается при частоте

$$f = \frac{1}{2\pi RC} \quad .$$

Подставив K, получим $f = \frac{K}{2\pi ds}$. Учитывая, что частота дискретизации fd равна 1/ds, можно записать:

$$K = 2\pi\frac{f}{fd} \quad (22).$$

На языке С реализация интегрирующей цепочки будет иметь вид:

```
U += (I – U) * K;
```

где U – выходной сигнал, I - входной сигнал, K рассчитывается на основании формулы (22).

Тестовый проект находится в папке 08_ADC_DAC_FIL_DET.

Код фильтра и детектора находится в обработчике прерывания ADC1_IRQHandler файла «adc.c».

```
#define FD   1000 //Частота дискретизации KHz
#define FR   100  //Частота фильтра KHz
#define FLPF 1    //Частота перегиба ФНЧ KHz
static float R;   //Параметр фильтра, зависящий от частоты


void ADC1_IRQHandler(void)
{

    //Параметр, определяющий полосу пропускания фильтра dF
    //L  ≈  5.6 * Fd/FD
    static float L = 0.004;

    //Начальные значения переменных фильтра
    static float X = 0;
    static float V = 0;

    //Параметр интегрирующей цепочки
    static float K = 2*M_PI*FLPF/FD; //K = 2PI*f/fd
    //Начальные значения переменных детектора
    static float U = 0;
    float D;
    int out;

    //Смещение для ADC и DAC
    static int     S_ADC = 2000;
    static int     S_DAC = 100;

    // Сброс флага переполнения в ADC1
    SET_BIT(ADC1_ISR, (1 << 3));

    //Фильтр
```

```
    X += (int)ADC1_DR- S_ADC;
    V -= X * R;
    X += V - X * L;

    //Детектор
    D = fabs(X);
    U += (D - U) * K; // Интегрирующая цепь
    out = U + S_DAC;

    // Ограничитель, предотвращает переполнение DAC при сильном сигнале
    if( out > 4000) out = 4000;

    //Запись в DAC
    DAC1_DHR12R1 = out;
}
```

После загрузки программы в контроллер измеряем характеристики сигнала на выходе DAC.

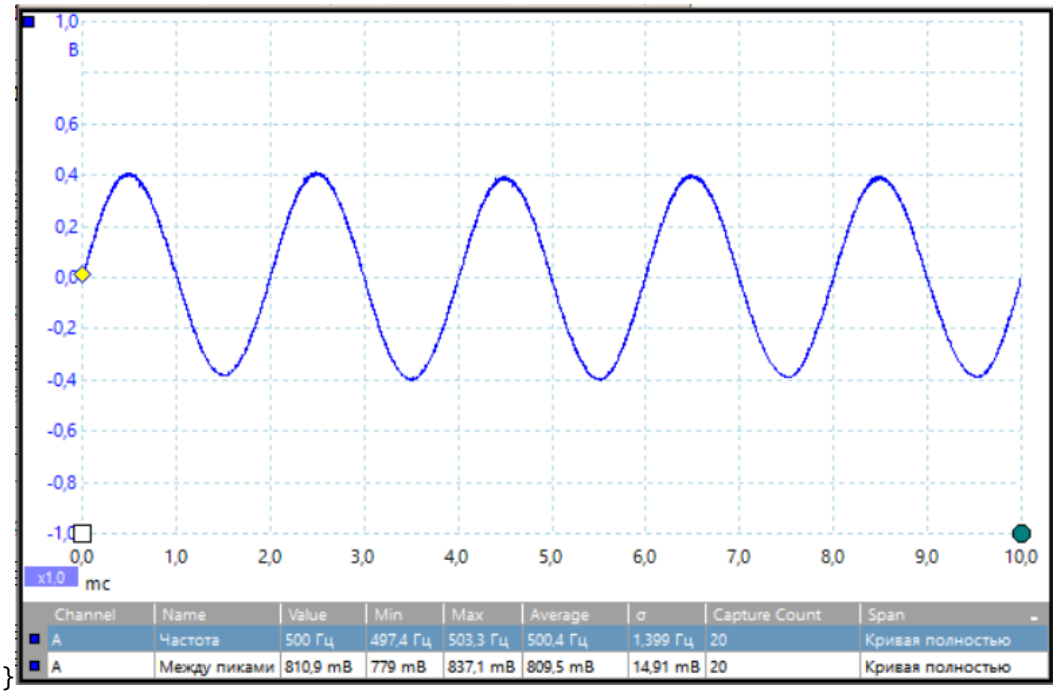


Рисунок 27. Сигнал на выходе DAC при подключенном детекторе (fr = 100 KHz, L = 0.004, частота среза ФНЧ 1KHz, амплитуда входного сигнала 10 mV, частота несущей 100 KHz, частота модулирующего сигнала 500 Hz, глубина модуляции 100%)

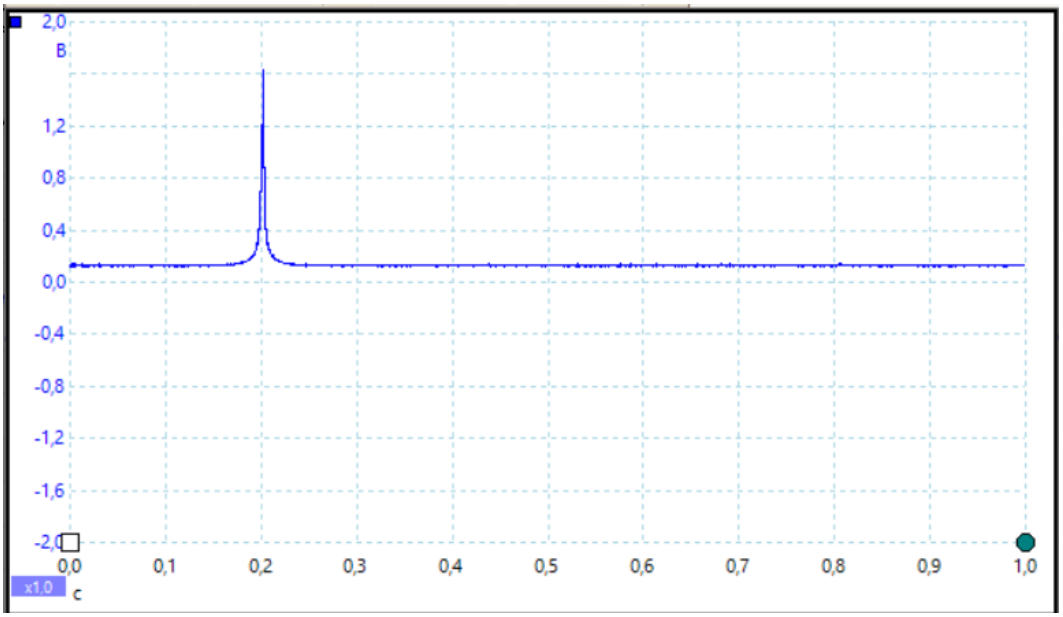


Рисунок 28. Амплитудно-частотная характеристика устройства L = 0.004, Fr = 100 KHz, амплитуда входного сигнала 10 mV, девиация частоты 500KHz (0 – 500 KHz).

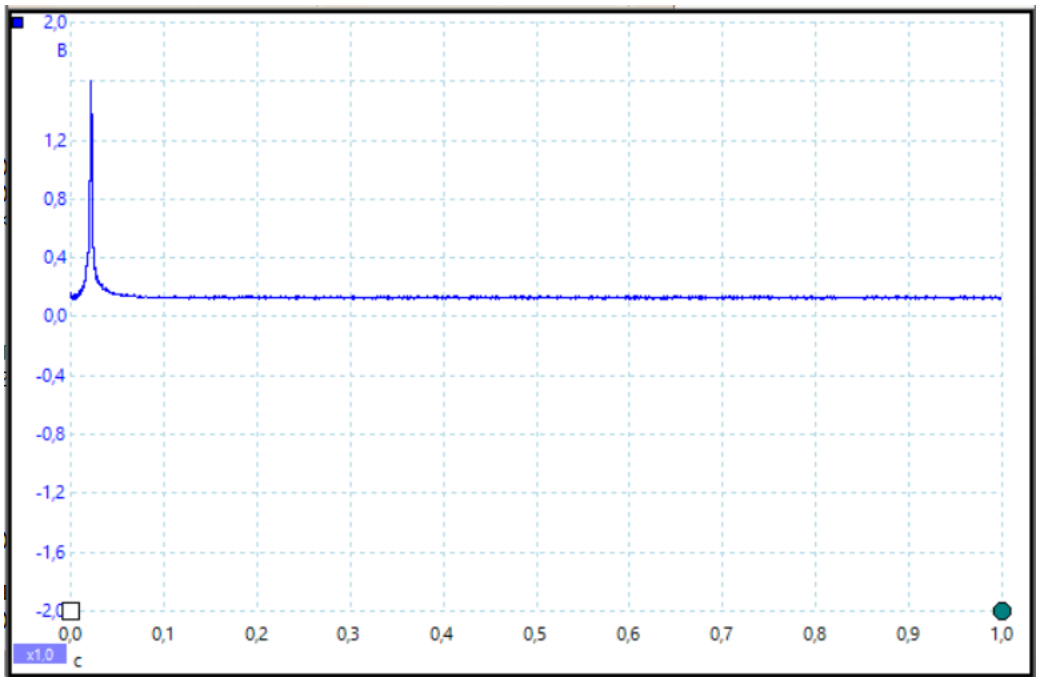


Рисунок 29. Амплитудно-частотная характеристика устройства L = 0.004, Fr = 10 KHz, амплитуда входного сигнала 10 mV, девиация частоты 500KHz (0 – 500 KHz).

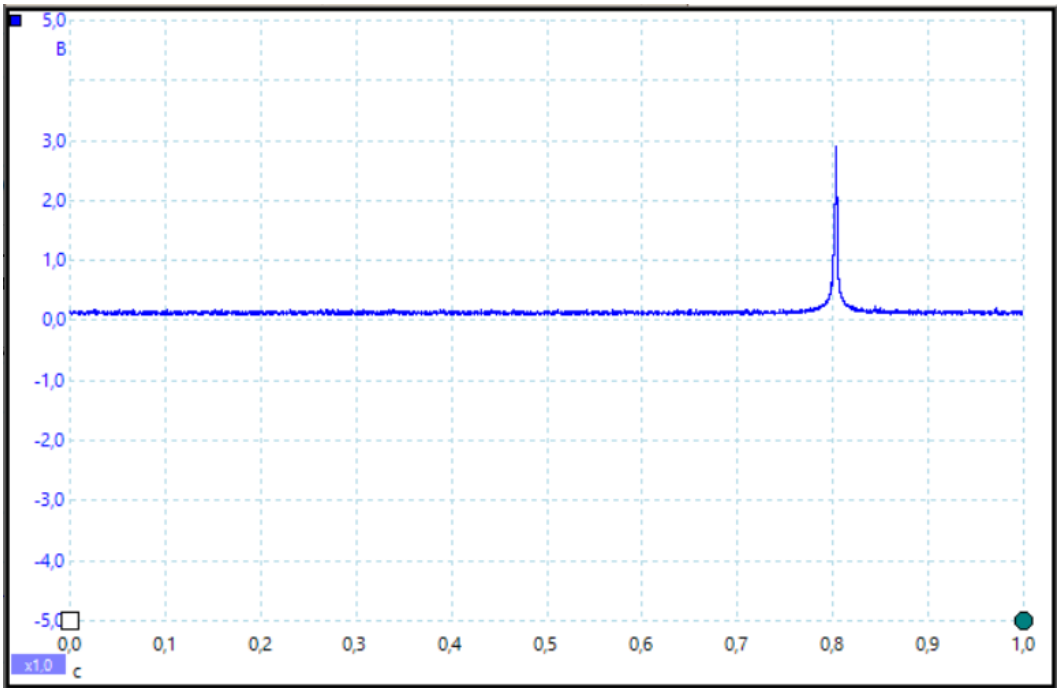


Рисунок 30. Амплитудно-частотная характеристика устройства L = 0.004, Fr = 400 KHz, амплитуда входного сигнала 6 mV, девиация частоты 500KHz (0 – 500 KHz).

Полученный макет является, по сути, приемником амплитудно-модулированных сигналов (AM) для диапазона частот 10 KHz – 400 KHz. Чувствительность устройства при отношении сигнал/шум 10дб составляет порядка 2 mV, селективность 26дб при расстройке 5 KHz, полоса пропускания 600 Hz по уровню 0.7. Это довольно неплохие характеристики.

К недостаткам устройства можно отнести относительно высокий уровень шума звукового сигнала в следствии ограниченного разрешения ADC и DAC (12 разрядов), относительно низкой частоты дискретизации (фильтр используется до частот Wr = 0.4, или всего 2.5 такта частоты дискретизации на период входного сигнала) и ограниченной разрядной сетки формата float32.

10. Настройка частоты – энкодер

Для изменения частоты настройки фильтра будем использовать специальное устройство - инкрементальный энкодер. Устроен он просто: две контактные группы замыкаются в нужном порядке в зависимости от направления вращения вала. Используемый энкодер EC11 содержит дополнительную кнопку (BUT), которая замыкается при нажатии на вал.

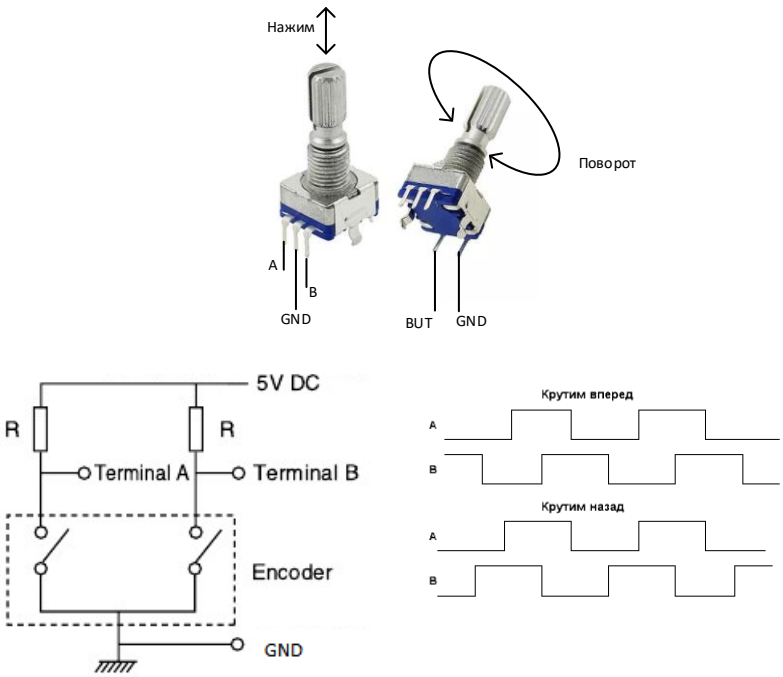


Рисунок 31. Работа инкрементального энкодера EC11

При вращении энкодера будем увеличивать/уменьшать частоту на 1 KHz на каждый «щелчок». При нажатии на кнопку энкодера (кнопка BUT) будем увеличивать частоту до ближайшего большего значения сотен. Например, если частота была 123 KHz, то после нажатия на кнопку BUT значение будет 200 KHz.

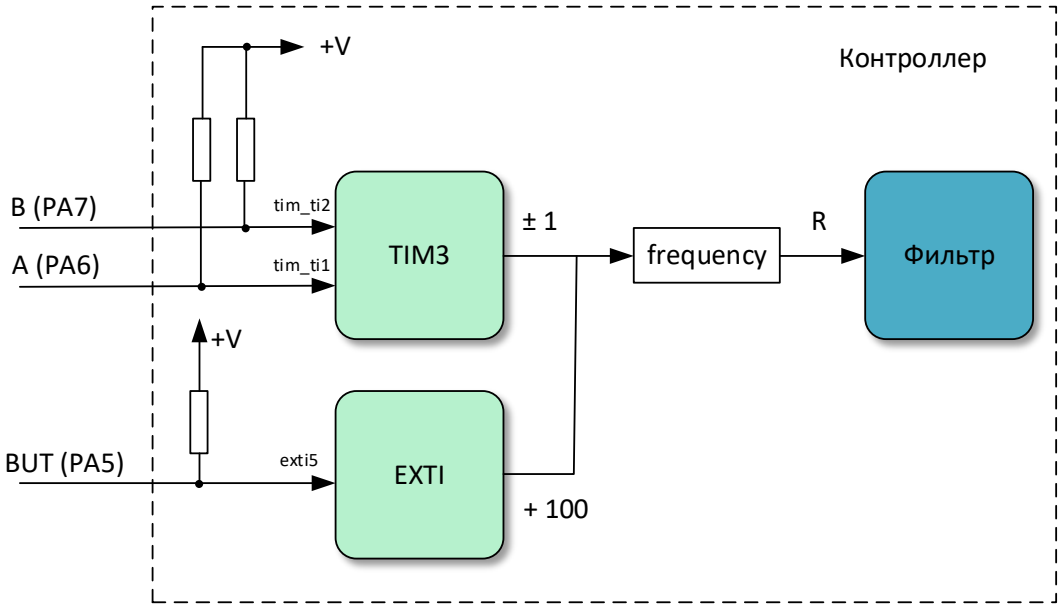


Рисунок 32. Структурная схема части программы работы с энкодером

Работа с инкрементальным энкодером является стандартной функцией для таймеров контроллера.

Выберем таймер TIM3 для обслуживания энкодера. Отрабатывать нажатие кнопки будем через контроллер внешних прерываний (EXTI).

Для обеспечения работы таймера TIM3 с энкодером нам нужно выполнить следующие шаги.

1. Настройка выводов контроллера PA6, PA7 на режим логического входа, с «подтяжкой» к положительной шине питания с помощью внутреннего резистора.

2. Настройка PA6 и PA7 на альтернативную функцию ввода (см. datasheet DS12589 Rev.2 Table 13. Alternate function), а именно подключение данных ножек через схемы внутренней коммутации ко входам tim_ti1 и tim_ti2 таймера TIM3, отвечающим за обработку сигналов от энкодера.

3. Настройка управляющих регистров таймера TIM3 на режим работы с энкодером.

4. Настройка прерывания от TIM3 по изменению содержимого счетчика таймера.

5. Создание функции-обработчика прерывания от таймера TIM3, которая будет модифицировать глобальную переменную «frequency».

Для обработки нажатия кнопки «BUT» через контроллер внешних прерываний выполняем следующие шаги.

1. Настройка вывода контроллера PA5 на режим логического входа с «подтяжкой» к положительной шине питания с помощью внутреннего резистора.

2. Настройка прерывания от блока EXTI при возникновении положительного фронта на ножке PA5 (канал внешнего прерывания EXTI5).

3. Создание функции-обработчика прерывания от блока EXTI по каналу EXTI9_5, которая будет увеличивать глобальную переменную «frequency» на 100 при каждом событии прерывания.

Работа счетчика таймера (регистр TIM3_CNT) в режиме обслуживания энкодера демонстрируется следующим рисунком.

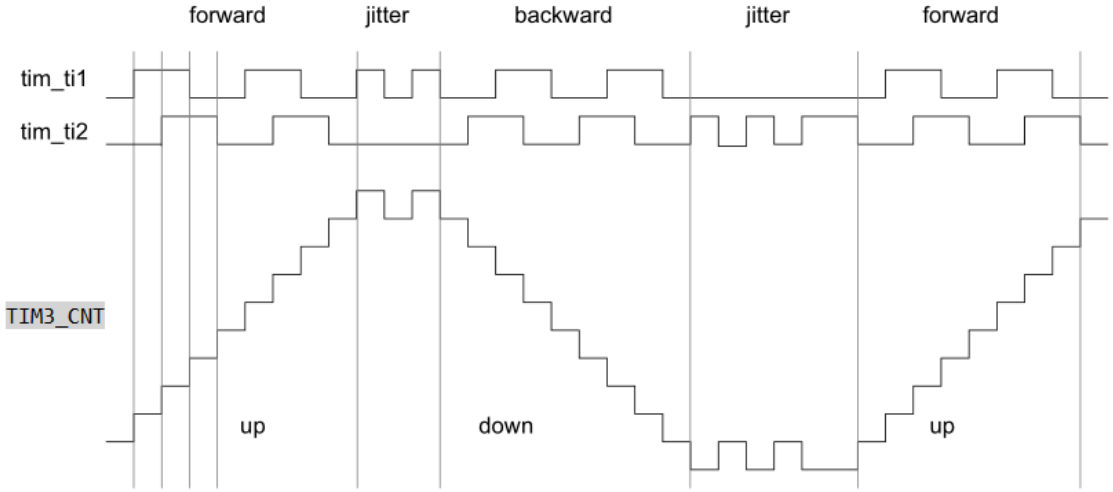


Рисунок 33. Работа счетчика таймера при обслуживании энкодера

Необходимо заметить, что значения регистра TIM3_CNT изменяются в диапазоне от 0 до значения, записанного в регистр TIM3_ARR (Auto-reload register) минус 1.

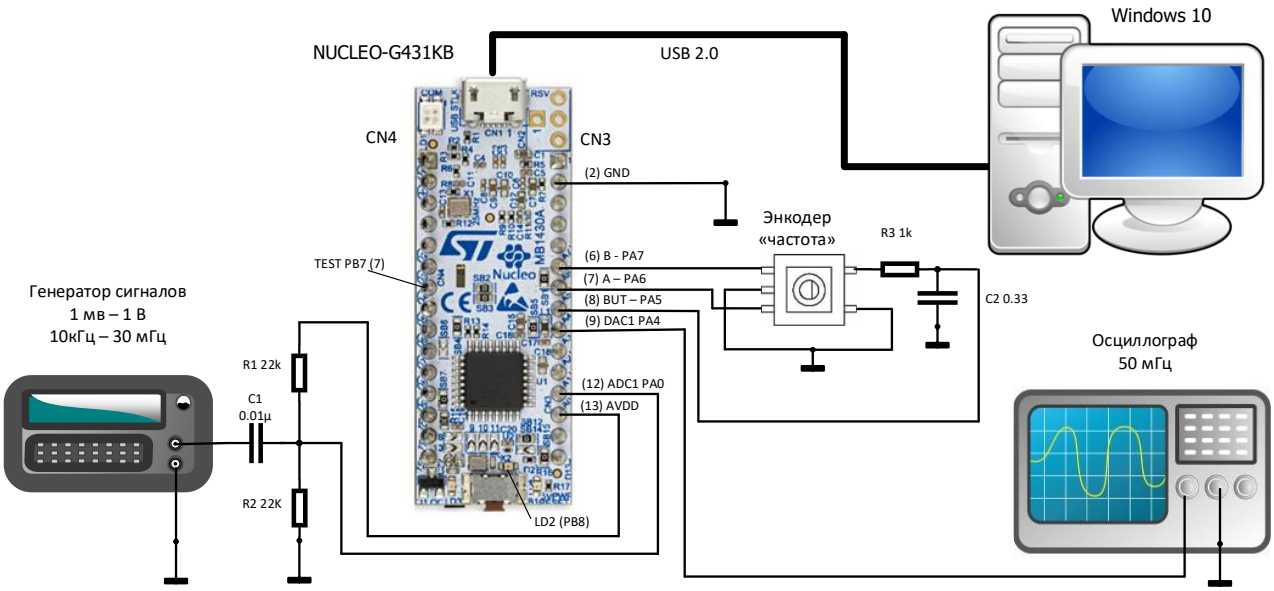


Рисунок 34. Схема исследовательского стенда с подключенным энкодером

На рисунке 34 представлена схема исследовательского стенда с подключенным энкодером. Для работы энкодера необходимо удалить перемычки 2 и 3, как указано на рисунке 5.

Проект работы с энкодером находится в папке 09_ADC_DAC_DET_FIL_DET_EN. В проект был добавлен файл «encoder.c», в котором находится функция инициализации энкодера и обработчики прерывания от TIM3 и EXTI5. Модификация глобальной переменной R, необходимой для задания центральной частоты полосы пропускания фильтра, выполняется каждый раз при вызове обработчика прерывания.

Для работы энкодера в проект введены следующие константы:

```
#define FREQ_MAX     400  // Максимальное значение частоты KHz
#define FREQ_MIN     100  // Минимальное значение частоты KHz
#define FREQ_START   100  // Начальное значение частоты KHz
```

Посмотреть результат работы программы можно с помощью осциллографа и генератора, включенного в режиме качания частоты.

11. OLED дисплей

Цифровая шкала настройки позволяет существенно повысить удобство использования радиоприемника. В проекте используется OLED графический индикатор на основе контроллера SSD1306. Индикатор имеет разрешение 128 x 32 точки и подключается к микроконтроллеру через интерфейс I2C.

Вывод алфавитно-символьной информации графического индикатора требует достаточно большого количества операций: генерация графического образа символа в памяти контроллера и побайтная передача образа символа в память дисплея по интерфейсу I2C.

Работу по передаче образа символа можно переложить на модуль прямого доступа к памяти (DMA) и таким образом разгрузить центральный процессор для задач обработки сигнала.

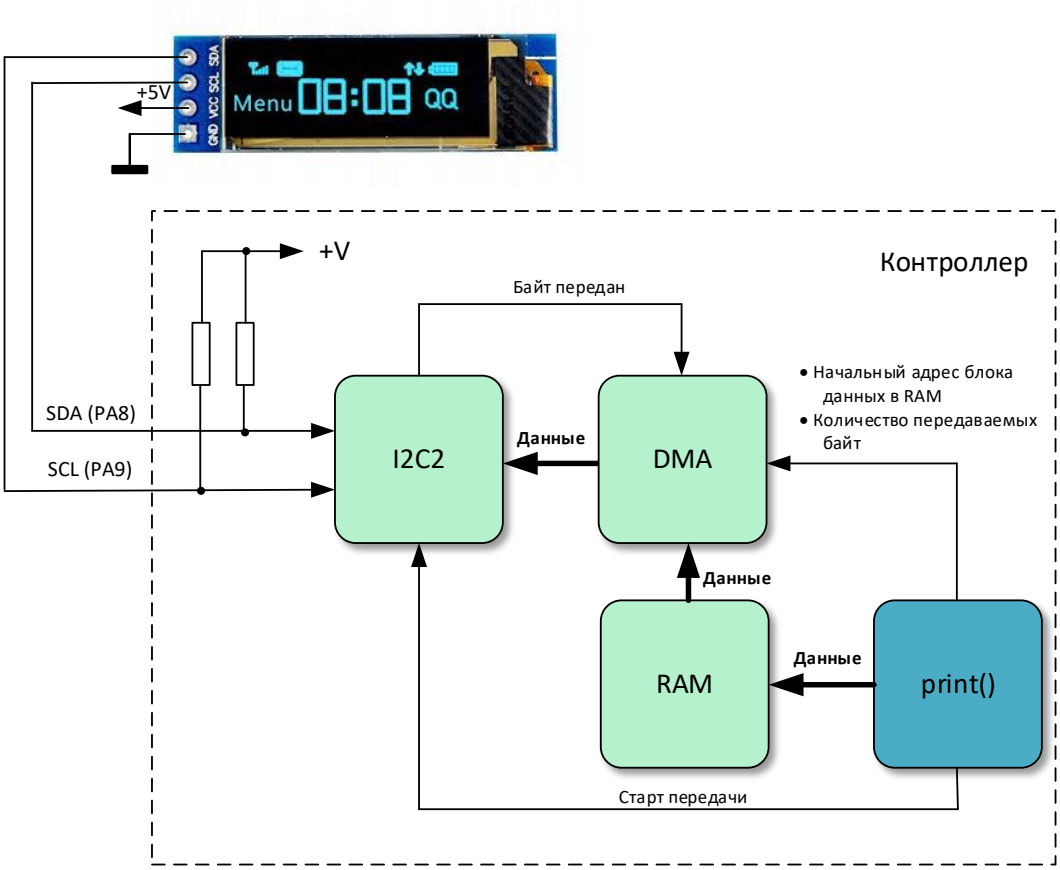


Рисунок 35. Схема передачи данных в OLED индикатор с использованием DMA

При вызове функции print() выполняется формирование в памяти контроллера многобайтной последовательности (команда), которую нужно передать по интерфейсу I2C в индикатор. Далее в модуль DMA записывается начальный адрес блока данных и количество байт для передачи, после чего в модуль I2C передается команда «Старт передачи». Далее процесс передачи блока данных по каналу I2C выполняется без участия основного процессора контроллера.

Тестовый проект с программой для стенда, схема которого представлена на рисунке 36, находится в папке 10_ADC_DAC_DET_FIL_DET_EN_OLED.

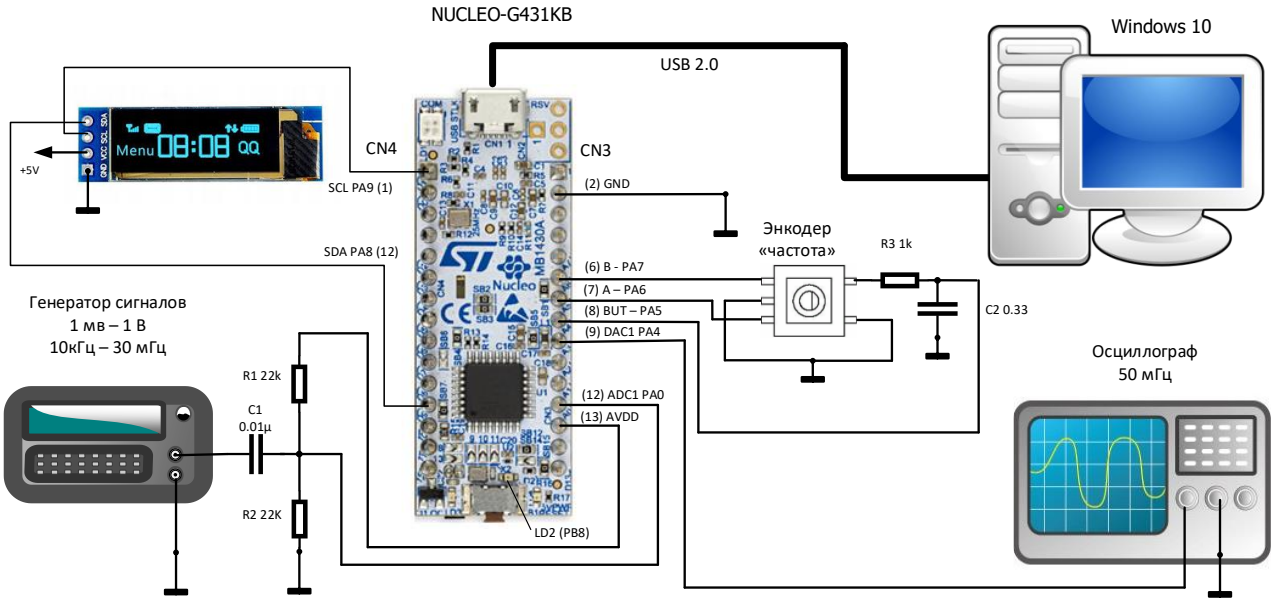


Рисунок 36. Схема исследовательского стенда с подключенным энкодером и OLED индикатором

12. Практическая схема радиоприемника

Схема простого радиоприемного устройства на базе отладочной платы NUCLEO-G431KB представлена на рисунке 37. Радиоприемник принимает сигналы в диапазоне длинных волн (LW) 150 – 280 KHz. На транзисторах T1 – T3 собран усилитель высокой частоты (УВЧ) с общим коэффициентом усиления порядка 100. Элементы L1, R3 и C3 образуют низкодобротный колебательный контур магнитной антенны, настроенный на середину диапазона. Первый каскад УВЧ собран по схеме дифференциального усилителя, который в значительной степени позволяет подавить синфазную помеху по входу и по линии питания. Подавление синфазной помехи является критически важным во время приема в городских условиях, при наличии большого количества бытовых электроприборов. Транзисторы T1 и T2 должны иметь по возможности одинаковые коэффициенты усиления по току. На транзисторе T3 собран второй каскад УВЧ, с выхода которого через разделительный конденсатор C6 высокочастотный сигнал подается на вход АЦП контроллера. Резисторы R9 и R10 образуют делитель напряжения, который выводит «нулевую» точку АЦП в середину диапазона значений, где-то около 2000.

Низкочастотный сигнал с выхода ЦАП поступает на фильтр низкой частоты (R11, C7, L2, C8), с частотой среза порядка 4 KHz, который служит для подавления шумов, возникающих за счет цифровой обработки сигнала. При некотором снижении качества фильтрации дроссель L2 можно заменить на резистор номиналом 1 ком. Выход фильтра подключен к усилителю мощности, собранному на транзисторе T4 по схеме эмиттерного повторителя. Наличие усилителя мощности позволяет подключить к устройству низкоомные наушники. Особенностью каскада является то, что он по постоянному току подключен к выходу ЦАП и не имеет отдельных цепей смещения. Выходное напряжение с выхода ЦАП изменяется от 0.6 до 3 вольт, что автоматически задает оптимальный режим работы каскада. При необходимости громкоговорящего приема сигнал с переменного резистора R13 можно подать на вход внешнего усилителя мощности.

Радиоприемник питается от четырех аккумуляторов AA с номинальным напряжением 1.2V.

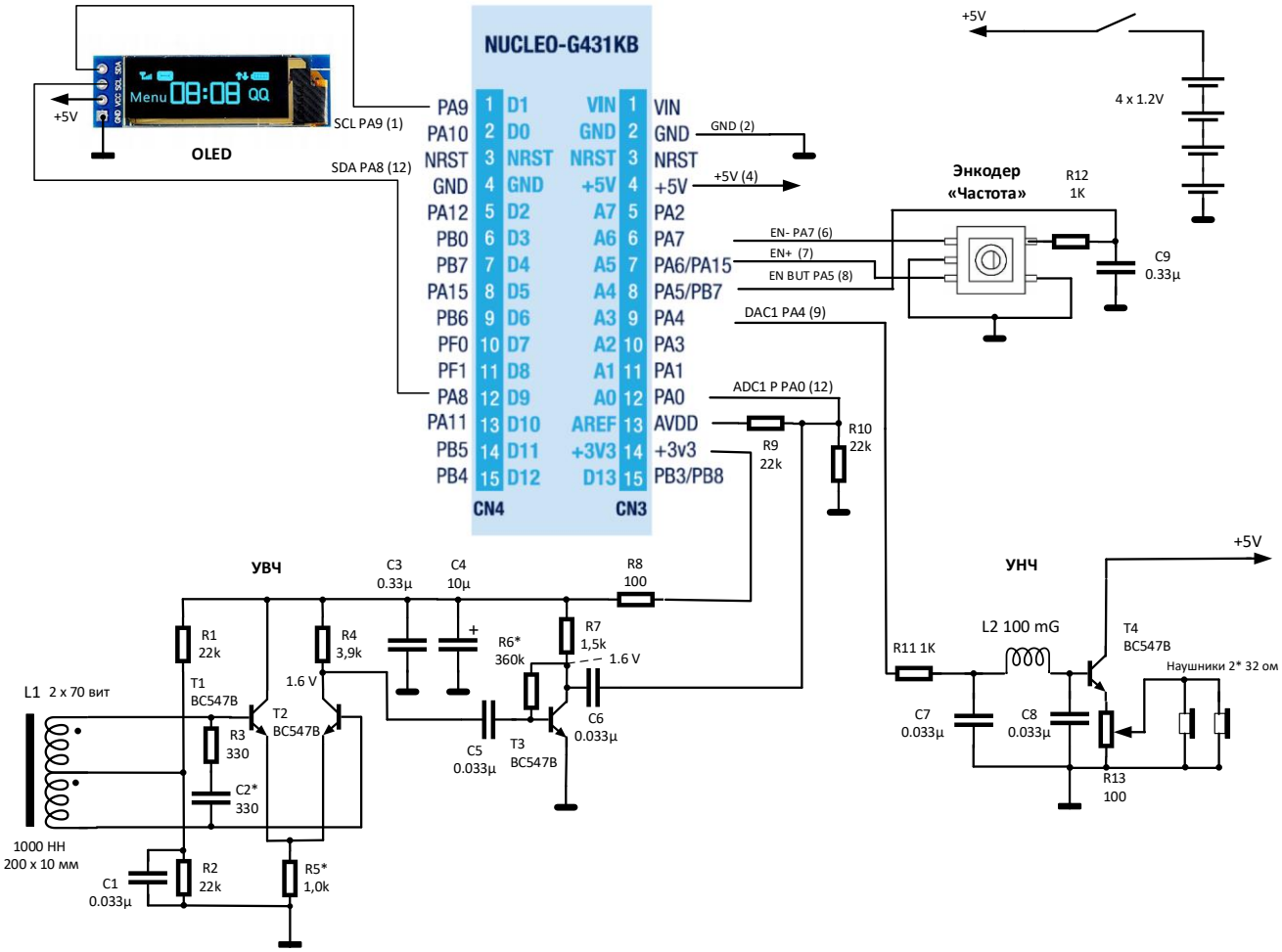


Рисунок 37. Схема радиоприемника диапазона LW

Проект с программой простейшего радиоприемника находится в каталоге 11_SIMPLE_RADIO.

Основное отличие от предыдущего проекта заключается в ведении автоматической регулировки уровня низкочастотного сигнала. Амплитуда выходного сигнала после детектора может меняться в диапазоне 90 дб. Динамический диапазон ЦАП составляет приблизительно 70 дб. Данное обстоятельство приводит к ограничению или полной потере выходного сигнала при приеме мощной радиостанции. Решением данной проблемы является автоматическое уменьшение уровня выходного сигнала при превышении среднего уровня некоторого порога, т.е. применение автоматической регулировки усиления (АРУ).

Программный код АРУ выглядит довольно просто:

```
OUT = U1;
//АРУ
U2  += (D - U2) * K2; // Интегрирующая цепь АРУ
if( U2 > AGCT)
       OUT = U1*(AGCT/U2);
```

Где U1 - это выход ФНЧ детектора. Мы используем еще одну интегрирующую цепь с частотой перегиба порядка 1 Hz. Если значение U2 меньше некоторого порога AGCT (порядка 1000), то сигнал OUT не меняется. В противном случае выходной сигнал уменьшается по формуле OUT = U1*(AGCT/U2). Дополнительным преимуществом использования АРУ является уменьшение эффекта замирания сигнала. Конечно, здесь речь идет о достаточно мощных радиостанциях.

К сожалению, в данной конфигурации программа работает на пределе производительности, в связи с чем пришлось уменьшить частоту дискретизации до 700 KHz. Для приема в длинноволновом диапазоне это не критично, и мне удавалось в Зеленограде (Московская область) в вечернее время на подоконнике окна многоэтажного дома достаточно уверенно принимать сигналы радиостанций на частотах 153 KHz (радио Румынии), 225 KHz (радио Варшава).

Необходимо заметить, что максимальная частота зависит не только от возможностей контроллера и разработанного программного кода, но и от используемого транслятора и его настроек. В проекте применялся Toolchain, поставляемый вместе с STM32CubeIDE (gnu-tools-for-stm32.9-2020-q2-update) при отключенной оптимизации (Optimization level: None –O0). При использовании компилятора и базовых библиотек от другого toolchain производительность может отличаться, иногда существенно.

Рисунок 38. Макет радиоприемника

По количеству мощных радиостанций диапазон средних волн 530 – 1600 KHz (MW) значительно более интереснее длинноволнового диапазона, но для достижения подобных «высот» нам необходимо увеличить производительность программы в 5 раз и поднять частоту дискретизации до 3500 KHz. Мы использовали далеко не все возможности микроконтроллера. В следующих главах рассмотрим возможности для «разгона» программы.

13. Захват сигнала с помощью DMA

Первое очевидное ускорение — это не тратить время на вызов обработчика прерывания при обработке каждого отсчета от ADC. Только на вызов прерывания требуется минимум 12 тактов процессора. К счастью, в арсенале контроллера имеется модуль прямого доступа к памяти (DMA). Записывать данные от ADC можно циклически в буфер памяти, расположенный в RAM, практически без участия процессного ядра. При заполнении буфера модуль DMA вызовет обработчик прерывания, который может запустить к исполнению код обработки сигнала сразу для большого количества отсчетов. Такая схема работы

позволяет параллельно заполнять текущий буфер памяти отсчетами от ADC с помощью DMA и обрабатывать предыдущий буфер с помощью ядра процессора.

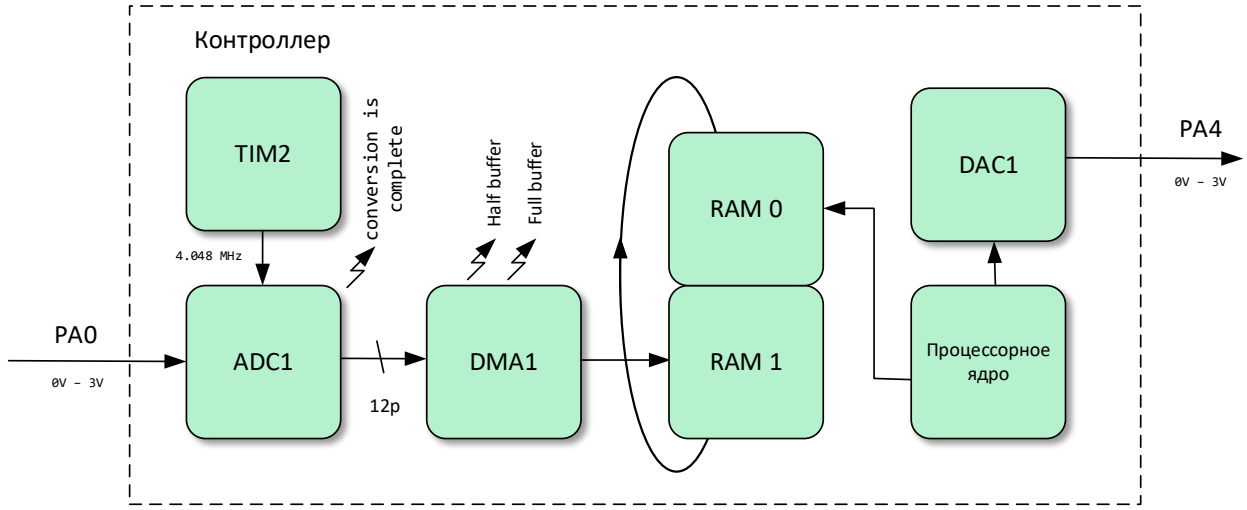


Рисунок 39. Схема записи отсчетов от ADC с использованием DMA

Проект программы, работающей по описанной схеме, находится в папке 12_SIMPLE_RADIO_DMA.

Изменений по сравнению с предыдущем проектом не так много, как может показаться на первый взгляд. В более ранних экспериментах мы просчитывали каждый отсчет ADC по прерыванию «Окончание преобразования в ADC» в обработчике ADC1_IRQHandler, в текущем эксперименте обрабатывается сразу 64 отсчета по прерываниям от DMA - «заполнена первая половина буфера» или «заполнена вторая половина буфера». Вся математика остается прежней и выполняется в обработчике DMA1_CH4_IRQHandler.

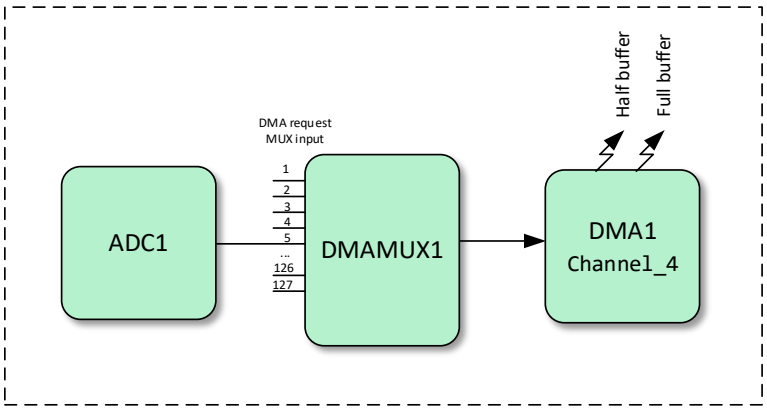


Рисунок 40. Коммутация запроса на обработку DMA

Подача запросов на передачу одного слова через DMA выполняется через специальный мультиплексор DMAMUX, который имеет 127 входов и может подключать запросы практически от всех устройств контроллера (рис. 40). Запрос на передачу отсчета от ADC подключен к линии номер 5.

Контроллер имеет два модуля DMA, каждый из которых содержит 6 независимых каналов. В нашем проекте мы используем DMA1 канал 4. Настройка мультиплексора DMAMUX1 и блока DMA1 выполняется в функции ADC1_init из файла «adc.c».

Еще одно изменение произошло в последовательности обработки отсчетов в обработчике DMA1_CH4_IRQHandler. Перестраиваемый полосовой фильтр работает внутри цикла перебора отсчетов из буфера, а ФНЧ и АРУ реализован вне цикла. Таким образом мы существенно разгружаем процессор в части обработки звукового сигнала.

Внесенные изменения позволили увеличить частоту дискретизации более чем в 2 раза с 700 KHz до 1600 KHz (константа FD в файле adc.c). Теперь имеется возможность настраивать фильтр на частоты 150 – 750 Khz, принимать сигналы диапазона LW и захватывать нижнюю часть MW диапазона.

14. Блок цифровой обработки сигнала (DSP) на ассемблере

Итак, для перекрытия всего MW диапазона необходимо увеличить скорость обработки сигнала минимум еще в 2 раза. Традиционным путем увеличения производительности программы является написание наиболее нагруженных частей кода на языке ассемблера. Такой критической секцией в нашем проекте является обработчик прерывания DMA1_CH4_IRQHandler.

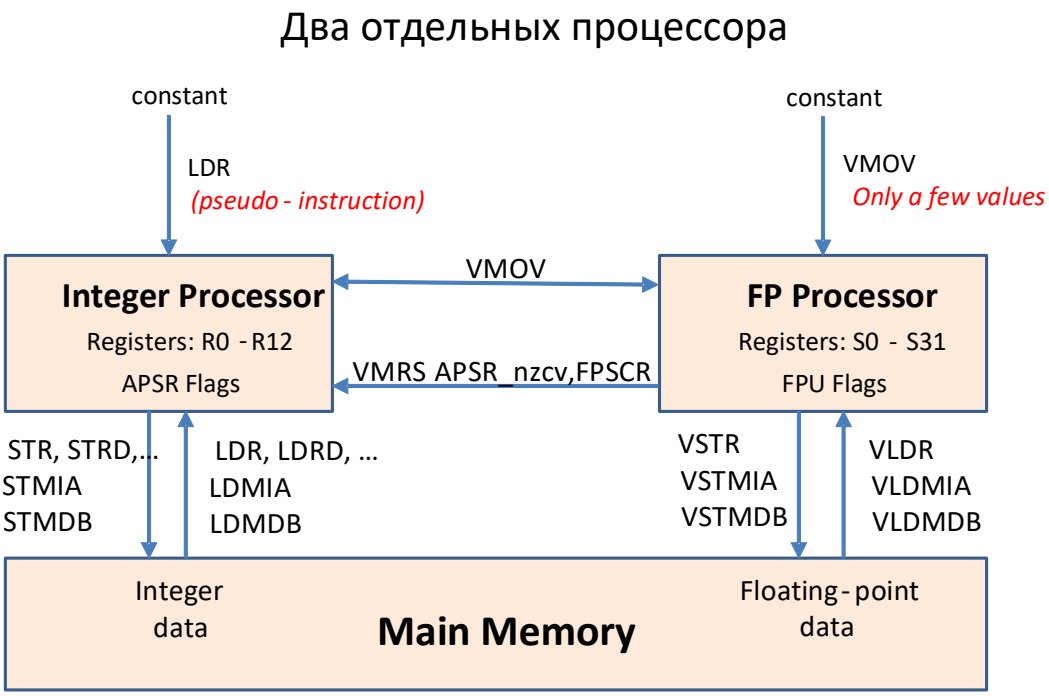


Рисунок 41. Обработка чисел с плавающей запятой (FP) в ядре контроллера

Вычислительное ядро контроллера содержит два процессора: один - для обработки целых чисел, другой - для чисел с плавающей запятой (рисунок 41). Сопроцессор для работы с плавающей точкой содержит 32 регистра. На рисунке 42 представлено распределение регистров сопроцессора при использовании стандартных функций языка С.

Рисунок 42. Регистры сопроцессора для работы с плавающей точкой

Для работы радиоприемника необходимо использовать 12 глобальных переменных с плавающей точкой:

- параметры фильтра - X, V, R, L, M;
- параметры интегрирующей цепочки детектора - U1, K1;
- параметры интегрирующей цепочки АРУ U2, K2;
- порог АРУ - AGCT;
- смещение ADC и DAC - S_ADC, S_DAC.

Для ускорения работы было бы не плохо хранить перечисленные переменные непосредственно в регистрах сопроцессора. Но имеется одно принципиальное препятствие – если мы используем в проекте какую-либо арифметику с плавающей запятой в функциях, написанных на языке С, то регистры сопроцессора могут быть произвольно переписаны и наши переменные будут изменены.

В текущем экспериментальном проекте мы полностью берем контроль над работой сопроцессора и отказываемся от использования плавающей точки в коде на С. Для этого нам нужно переписать две существующие функции:

- DMA1_CH4_IRQHandler – обработчик прерывания от DMA для ADC.
- KHZ_to_R - расчет параметра фильтра в зависимости от частоты R=(2*Sin(PI*W))^2.

Также необходимо добавить новую функцию dsp_init, которая записывает в регистры сопроцессора начальные значения при старте программы.

В предыдущем проекте в функции KHZ_to_R использовалась стандартная библиотечная функция языка С - sin(x). В новом проекте мы будем применять другой метод расчета R, а именно кусочно-линейную аппроксимацию, которая позволяет достичь даже несколько лучших результатов по точности настройки фильтра.

Проект находится в папке 13_SIMPLE_RADIO_DMA_ASM. Весь ассемблерный код помещен в файл «dsp.s». Не будем подробно останавливаться на программировании на ассемблере для ARM cortex M4 процессоров. Код достаточно подробно прокомментирован.

Использование ассемблера позволило увеличить скорость обработки практически в три раза. Теперь мы можем поднять частоту дискретизации до значения 3600 KHz и таким образом перекрыть весь MW диапазон.

Рассмотрим методику получения коэффициентов аппроксимации параметра R цифрового фильтра. Для определения корректирующей функции воспользуемся формулой расчета параметра фильтра:

$$R = (2\pi f \times ds)^2.$$

Нам необходимо раскомментировать строку под номером 100 в функции KHZ_to_R файла «dsp.s». Будем подавать на вход контроллера амплитудно-модулированный сигнал от генератора. Частоту сигнала будем менять от 100 KHz до 1600 KHz с шагом 100 KHz (схема стенда рис. 36). Для каждого значения входной частоты будем настраивать фильтр с помощью энкодера в резонанс и считывать показание частоты на OLED индикаторе. Результаты можно записать в виде таблицы (первых три столбца).

№	Входная частота Fr KHz	Выходная частота Frc KHz	A[i]	B[i]
0	100	100	1,000	0,00
1	200	199	0,990	1,00
2	300	297	0,980	3,00
3	400	392	0,950	12,00
4	500	485	0,930	20,00
5	600	574	0,890	40,00
6	700	658	0,840	70,00
7	800	738	0,800	98,00
8	900	812	0,740	146,00
9	1000	880	0,680	200,00
10	1100	941	0,610	270,00
11	1200	995	0,540	347,00
12	1300	1042	0,470	431,00
13	1400	1081	0,390	535,00
14	1500	1111	0,300	661,00
15	1600	1134	0,230	766,00

Таблица 1. Результат расчета коэффициентов линейной аппроксимации

Таким образом, чтобы настроить приемник на частоту, например, 1000 KHz нам необходимо подставить в формулу $R = (2\pi f \times ds)^2$ значение частоты 880 KHz.

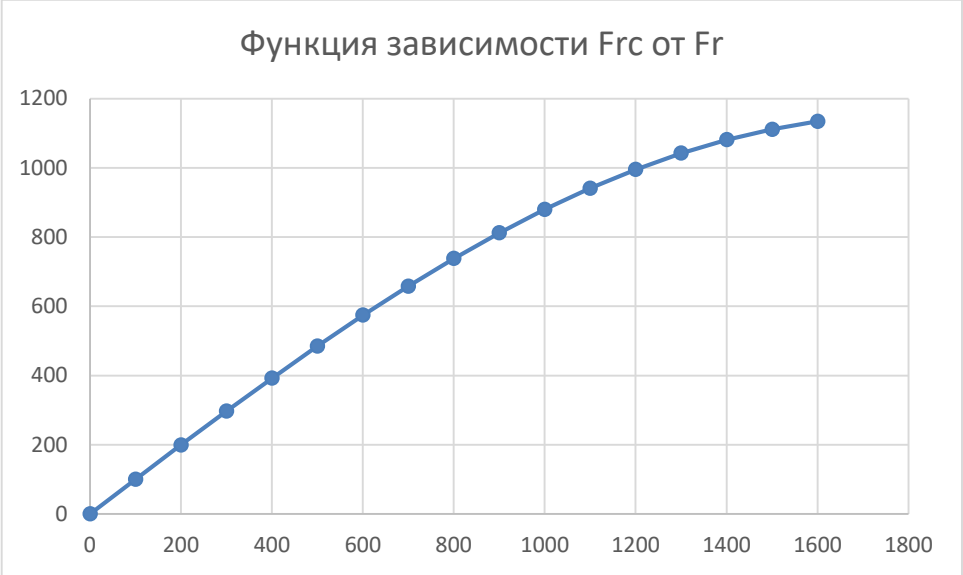


Рисунок 43. Кривая зависимости Frc от Fr

Для расчета промежуточных значений будем использовать формулу:

$$Frc = A[i]Fr + B[i] \quad (23),$$

где Frc - скорректированная частота, Fr – частота резонанса, i – индекс элемента массива коэффициентов линейной аппроксимации i = Fr % 100 (номер строки в таблице 1).

Значение A[i] и B[i] рассчитываются по формулам:

$$A[i] = (Frc[i+1] - Frc[i])/100;$$

$$B[i] = Frc[i] - A[i] \times Fr[i].$$

Пример расчета элементов массивов приведен в excel файле «Коррекция шкалы F I.xlsx», размещенном в каталоге текущего проекта. Полученные коэффициенты помещаем в массивы A[17] и B[17] файла «adc.c».

Теперь нужно закомментировать строчку номер 100 в функции KHZ_to_R файла «dsp.s» и проверить точность настройки. Измерения показали, что до частоты 1000 KHz погрешность настройки не превышает 1 KHz и на частотах от 1000 KHz до 1600 KHz не превышает 3 KHz.

Теперь можно проверить реальную скорость обработки блока в 64 отсчета ADC в функции DMA1_CH4_IRQHandler. Для этого раскомментируем строчки в данной функции, отвечающие за изменение сигнала на ножке PB7. Осциллограмма сигнала представлена на рисунке 44. Как видно из рисунка, время обсчета 64 отсчетов составляет приблизительно 10 мкс, а время записи сигнала в буфер (период между вызовами функции DMA1_CH4_IRQHandler) составляет приблизительно 17 мкс. Таким образом мы имеем достаточный запас по производительности и можем внести некоторые усовершенствования в программу.

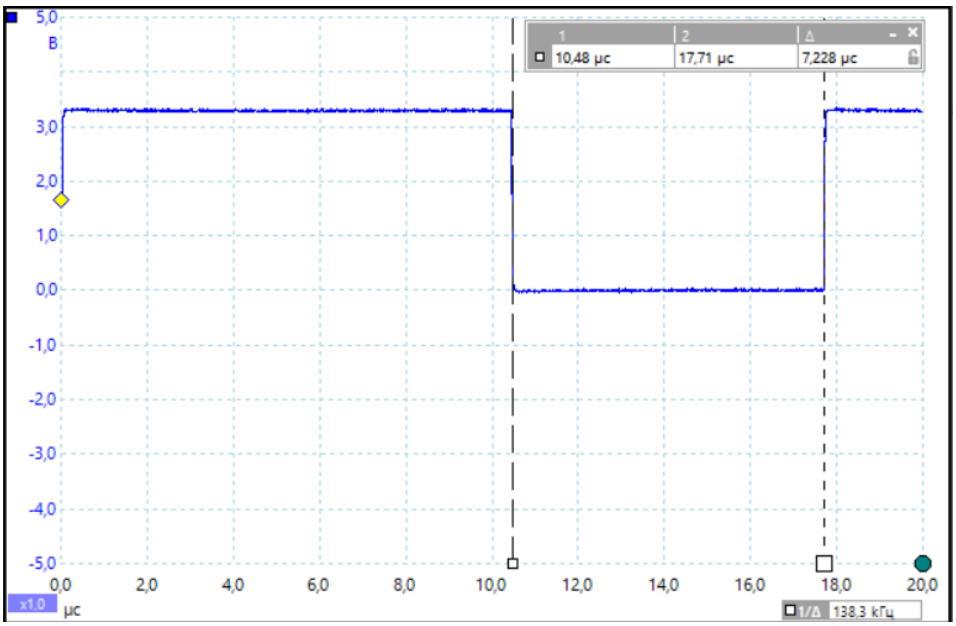


Рисунок 44. Время обработки 64 отсчетов (сигнал на ножке PB7) при частоте дискретизации 3600 KHz

Надо отметить, что в связи с расширением диапазона принимаемых сигналов схема приемника (рис. 37) претерпела небольшие изменения (рис. 45): 1) количество витков катушки L1 теперь должно быть 35 + 35 витков, 2) из схемы удалены элементы R3, C2.

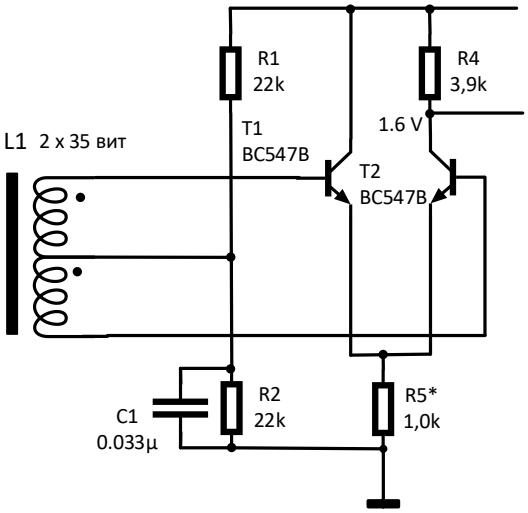


Рисунок 45. Модификация принципиальной схемы

15. Усовершенствованный радиоприемник

Увеличение скорости обработки позволило использовать еще один каскад фильтра в функции TIM4_IRQHandler и таким образом улучшить селективность радиоприемника.

Также был организован вывод относительной величины входного сигнала на OLED индикатор. В качестве уровня входного сигнала используется выходное значение интегрирующей цепочки блока АРУ (U2), деленное на 10. Вывод происходит раз в секунду по прерыванию от таймера TIM4. Величина «100» на индикаторе приблизительно соответствует 10 мВ на входе ADC1. Учитывая, что коэффициент

усиления УВЧ составляет около 100, одна единица выводимого уровня сигнала соответствует 1 мкВ сигнала от магнитной антенны.

Еще одно усовершенствование связано с изменением коэффициента компенсации передачи фильтра (M) в зависимости от частоты настройки. При изменении частоты настройки от низкочастотного конца диапазона к высокочастотному коэффициент передачи двухкаскадного фильтра увеличивается приблизительно 50 раз. Изменение происходит по нелинейному закону. Компенсация коэффициента передачи фильтра позволяет снизить уровень шумов в высокочастотной части диапазона.

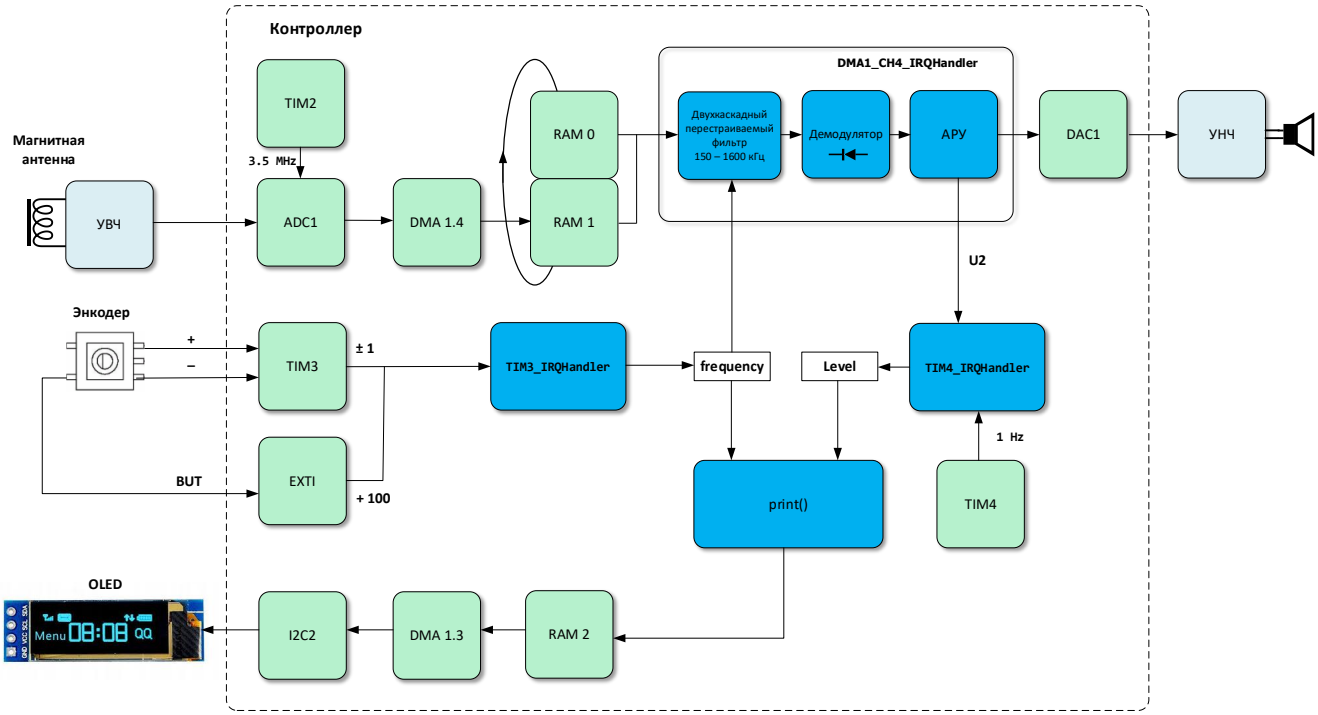


Рисунок 46. Обобщенная схема усовершенствованного радиоприемника

Финальная обобщенная схема радиоприемника представлена на рисунке 46. Проект находится в папке 14_SIMPLE_RADIO_DMA_ASM_II.

Хочется отметить, что весь проект содержит около 1000 строк кода с комментариями, занимает около 7KB флэша и 300 байт RAM.

Демонстрацию работы радиоприемника можно посмотреть по ссылке https://youtu.be/W8Ki2eaznIk.