

УДК 004.31 (075.8)
ББК 32.973.26-018
Б 73

Богаченков А.Н. Цифровые устройства и микропроцессоры [Электронный ресурс]: Методические указания по выполнению лабораторных работ. — М.: МИРЭА – Российский технологический университет, 2019. — 1 электрон. опт. диск (CD-ROM).

Методические указания содержат описания 3-х лабораторных работ, в которых изучаются аппаратные и программные средства процессоров с ядром ARM.

Материал предназначен для студентов очной формы обучения по направлениям: 11.03.01 «Радиотехника», 11.03.02 «Инфокоммуникационные технологии и системы связи», 11.03.03 «Конструирование и технология электронных средств» и специальности 11.05.01 «Радиоэлектронные системы и комплексы».

Материал может быть использован при изучении дисциплин «Цифровые устройства и микропроцессоры», «Цифровые устройства в телекоммуникациях», «Цифровые устройства и микропроцессоры в конструкциях электронных средств» студентами как очной, так и очно-заочной форм обучения, а также для самостоятельной работы при освоении базового курса кафедры.

Методические указания издаются в авторской редакции.

Рецензенты:

Удалов Александр Иванович, к.т.н., доцент кафедры телекоммуникаций и радиотехники РТУ МИРЭА

Системные требования:

Наличие операционной системы Windows, поддерживаемой производителем.

Наличие свободного места в оперативной памяти не менее 128 Мб.

Наличие свободного места в памяти постоянного хранения (на жестком диске) не менее 30 Мб.

Наличие интерфейса ввода информации.

Дополнительные программные средства: программа для чтения pdf-файлов (Adobe Reader).

Подписано к использованию по решению Редакционно-издательского совета

МИРЭА — Российский технологический университет.

Объем: 2.47 мб

Тираж: 10

© Богаченков А.Н., 2019

© МИРЭА – Российский технологический университет, 2019

Оглавление

ЛАБОРАТОРНАЯ РАБОТА № 1	4
Порядок выполнения	6
1. Ознакомление со средой разработки, операторами языка Си.....	6
2. Построение графиков функций стандартной библиотеки Си	10
3. Индивидуальное программирование вычислительных операций	12
Контрольные вопросы	15
ЛАБОРАТОРНАЯ РАБОТА № 2	17
Порядок выполнения	18
1. Ввод-вывод логических сигналов через параллельные порты	18
2. Использование таймеров для формирования временных интервалов и сигналов с широтно-импульсной модуляцией	19
3. Использование таймеров для генерации тональных сигналов	21
4. Использование таймеров для измерения временных интервалов	22
Контрольные вопросы	23
ЛАБОРАТОРНАЯ РАБОТА № 3	24
Порядок выполнения	24
1. Измерения с помощью аналого-цифрового преобразователя	24
2. Вывод сигналов с использованием ЦАП и аудиокодека	27
3. Дискретизация сигналов с помощью АЦП	28
Контрольные вопросы	30
СПИСОК ЛИТЕРАТУРЫ	32
ПРИЛОЖЕНИЕ. ИСХОДНЫЕ ТЕКСТЫ ПРОГРАММ	33
Тексты к лабораторной работе № 1	33
Тексты к лабораторной работе № 2	39
Тексты к лабораторной работе № 3	50

Лабораторная работа № 1

Общее знакомство с процессорами с ядром ARM Cortex-M4 и средой разработки программного обеспечения

Целью лабораторной работы является ознакомление: с основными характеристиками процессоров, использующих ядро ARM Cortex-M4, со средой разработки программного обеспечения, с форматами и диапазонами обрабатываемых чисел, особенностями выполнения операций на языке Си, формами представления выходной информации. Проверка работы производится в режиме симуляции и на отладочной плате.

Лабораторный макет

Внешний вид отладочной платы и основные компоненты, задействованные при выполнении лабораторного практикума, показаны на рис. 1.1.

Функциональная схема макета и его описание приведены в учебном пособии «Процессоры с ядром ARM в лабораторном практикуме и курсовом проектировании» [4].

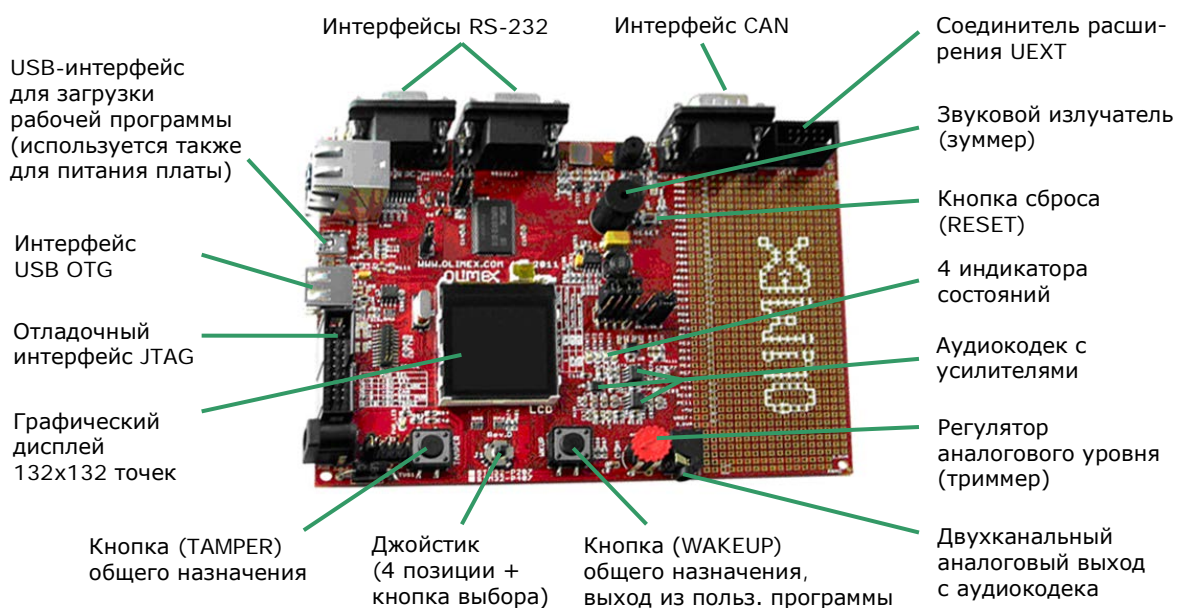


Рис. 1.1

Среда разработки

Для разработки программ для процессоров с ядром ARM используется интегрированная среда MDK-ARM μ Vision 5) компании Keil (www.keil.com). Программный пакет имеет в составе базу данных о всех выпущенных процес-

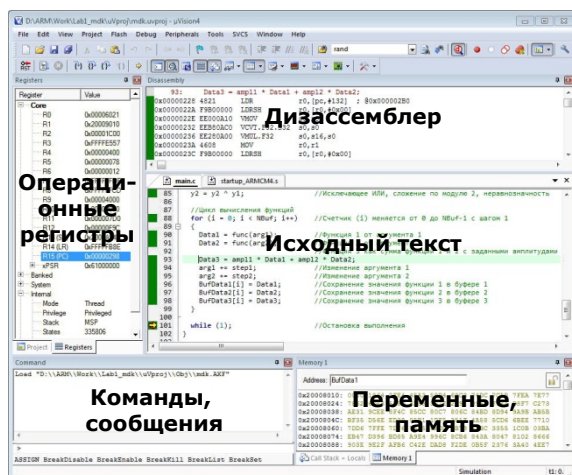
сорах с данным ядром, систему управления созданием проектов, интеллектуальный редактор для подготовки исходных текстов, библиотеки функций (математических, ввода-вывода, обработки сигналов и др.), компилятор-компоновщик программ на языках C/C++ и ассемблера, отладчик-симулятор, отладчик с использованием внутрисхемного эмулятора, расширения для оптимизации процесса проектирования, быстрого создания устройств с различными интерфейсами. Свободно распространяемая версия, не требующая лицензирования, имеет ограничение на объем скомпилированного кода (32 килобайта), а также еще на некоторые функции, не используемые в учебных проектах.

Пользовательский интерфейс программного пакета MDK-ARM имеет различный вид в зависимости от режима работы: производится работа с файлами проекта или отладка. Вид по умолчанию показан на рис. 1.2. При отладке может быть открыто большое число дополнительных окон: несколько областей памяти, стек, локальные переменные, графический анализатор, регистры ядра, регистры всех периферийных устройств, консольный вывод и др. Основные действия при работе в среде приведены в табл. 1.1.

Таблица 1.1. Основные действия в среде Keil μ Vision

Операция	«Горячая» клавиша	Кнопка	Меню
Компиляция проекта	F7		Project/Build
Настройки проекта	—		Project/Options for Target...
Переход в режим отладки, возврат из режима отладки	Ctrl+F5		Debug/Start/ Stop Debug Session
Вставка/удаление точки останова	F9		Debug/Insert/Remove Breakpoint
Шаговое выполнение с заходом в вызываемые функции	F11		Debug/Step
Шаговое выполнение без захода в вызываемые функции	F10		Debug/Step Over
Непрерывное выполнение	F5		Debug/Run
Остановка выполнения	—		Debug/Stop
Открытие окна логического анализатора	—		View / Analysis Windows / Logic Analyzer

Режим подготовки проекта



Режим отладки

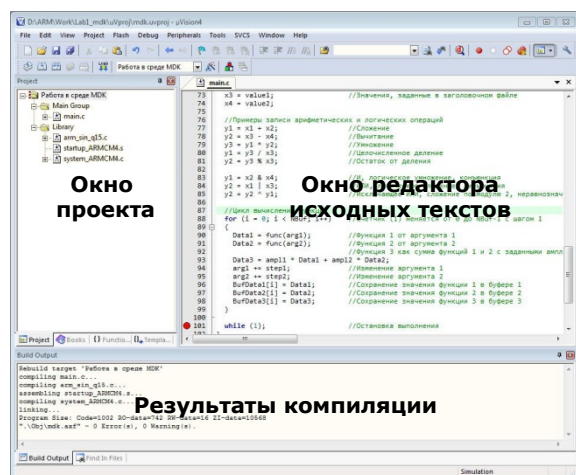


Рис. 1.2

Загрузка и выполнение рабочей программы

После сброса процессор выполняет находящуюся во Flash-памяти специальную программу-загрузчик, позволяющую по USB интерфейсу загрузить из компьютера в оперативную память процессора рабочую программу пользователя. После компиляции проекта в среде MDK-ARM и отсутствия ошибок автоматически запускается утилита ARM Loader, при нажатии в ее окне кнопки "Load & Run" машинный код передается в отладочную плату и запускается на выполнение.

Внимание! Для перезапуска рабочей программы (после модификации и компиляции) необходимо осуществить сброс кнопкой **WAKEUP** или **RESET** и затем нажать "Load & Run". При возникновении различных сбоев нажать RESET.

Порядок выполнения

Перед началом работы выполнить процедуру очистки рабочего каталога, используя значок на рабочем столе **"Инициализация ARM"**. Запустить среду разработки значком **"Keil uVision"**.

1. Ознакомление со средой разработки, операторами языка Си

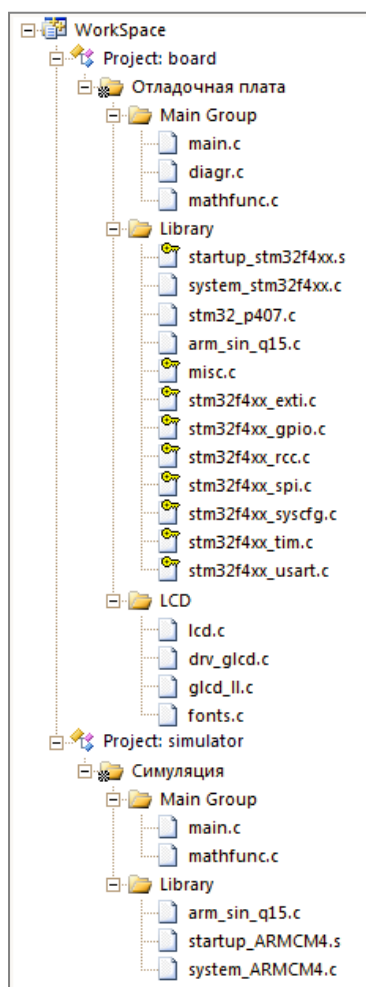
Файл одиночного проекта имеет расширение ***.uvproj** или ***.uvprojx**, мульти-проекта — ***.uvmpw**, открывается выбором пункта основного меню **Project/Open project**. Как правило, в проект входит большое число модулей, которые для удобства разбивают на группы. Проект может содержать различные типы файлов (табл. 1.2).

Таблица 1.2. Типы файлов в проектах среды MDK-ARM

*.c, *.cpp	Исходные тексты программы на языке C/C++, в лабораторном практикуме достаточным является использование языка C.
*.h, *.inc	Заголовочные (подключаемые) файлы, содержащие объявления общих типов данных, констант, функций. Эти файлы автоматически подключаются к проекту при наличии ссылок на них в исходных файлах (директива #include).
*.s, *.a, *.asm	Исходные тексты на языке ассемблера.
*.obj, *.o	Ранее откомпилированные модули (объектные модули).
*.lib	Файлы стандартных или пользовательских библиотек.
*.txt	Текстовые файлы для различных пояснений (компилятором игнорируются).

С помощью пункта меню **Project/Open Project...** открыть проект

D:\ARM\Work\Lab1_diagr\MDK-ARM\multiproject.uvmpw



Просмотреть структуру проекта и входящие в него модули. Проект представляет собой так называемый мульти-проект: в одной рабочей области (Workspace) содержится в данном случае два проекта. Один – для реальной отладочной платы (board), другой – для симуляции (simulator).

В режиме симуляции работа процессора (фактически только его ядра) моделируется компьютерной программой. Для этой цели используется некий виртуальный процессор, который в настройках проекта обозначен как «Cortex-M4 FPU» (FPU – блок вычислений с плавающей точкой).

Проект кроме основного пользовательского файла **main.c** содержит несколько стандартных (библиотечных), из которых, как правило, всегда присутствуют следующие два:

startup... .s — файл с таблицей векторов прерываний, наиболее важным в этой таблице с меткой **__Vectors** являются первые две записи, которые содержат адрес для регистра-указателя стека (SP) и

начальный адрес программы, загружаемый в программный счетчик (РС).

system... .c — файл, содержащий инициализационные действия, производимые перед началом выполнения пользовательской программы, например, инициализацию тактового генератора процессора (после сброса он работает на минимальной частоте).

В состав проекта для отладочной платы входят библиотечные файлы для программирования периферийных устройств, например:

stm32_p407.c — подпрограммы для работы с компонентами отладочной платы (кнопками, индикаторами и др.);

misc.c — обслуживание векторной системы прерываний ядра;

stm32f4xx_exti.c — конфигурирование прерываний от линий ввода-вывода;

stm32f4xx_gpio.c — программирование портов ввода-вывода;

stm32f4xx_rcc.c — конфигурирование системы синхронизации;

lcd.c — обслуживание графического дисплея и др.

В файле **main.c** имеется функция **main()**, с которой всегда начинается выполнение пользовательской программы. Так как файл является общим для обоих проектов, то различия в работе учитываются использованием макроопределения **SIMUL** (макроопределение задано только в настройках проекта для симуляции).

Для переключения между проектами необходимо в окне проекта правой кнопкой мыши выделить строку **Project: board** или **Project: simulator**, выбрать пункт **Set as Active Project**.

Активизировать проект **simulator**, откомпилировать, перейти в режим отладки, **выполнить по шагам** арифметические и логические операции, несколько циклов вычисления функций, наблюдая результат в:

- окне Call Stack – Locals;
- наводя курсор на имена переменных в исходном тексте.

Сопоставить исполняемые операции на языке высокого уровня и соответствующие им команды ассемблера (дизассемблированный текст отображается в окне “**Disassembly**”). Заполнить и **привести в отчете** табл. 1.3, указав команды только самих арифметических и логических операций. Учесть, что один оператор языка Си обычно транслируется в несколько команд, среди которых, например — **LDR** (загрузка из памяти), **STR** (запись в память), **MOV** (пересылки), **SXT** (изменение разрядности), также учесть, что мнемоники команд могут иметь дополнительные суффиксы (например, **S**).

Таблица 1.3. Обозначение операций на языках Си и ассемблера

Операция	Язык Си	Команда ассемблера
Сложение	+	ADDS
Вычитание	-	
Умножение	*	
Целочисленное деление	/	
Остаток от деления	%	
Инверсия	~	
И (логическое умножение)	&	
ИЛИ (логическое сложение)		
Исключающее ИЛИ	^	

Выполнить построение графиков функций в следующем порядке (изложен также в виде комментария в файле main.c):

- Откомпилировать проект (F7).
- Инициировать режим отладки (Ctrl+F5).
- Поставить точку остановки на последний оператор – `while(1)` – должна присутствовать красная метка слева.
- Открыть логический анализатор: меню View / Analysis Windows / Logic Analyzer.
- Если левое поле анализатора пусто, в тексте программы последовательно выделить имена `Data1`, `Data2`, `Data3` и перетащить их на это поле анализатора.
- Запустить программу на выполнение (F5).
- После остановки отрегулировать масштаб диаграмм полем Zoom (In/Out/All).
- Возвратиться из отладочного режима (Ctrl+F5).

Пример вывода диаграммы с исходными значениями амплитуд сигналов показан на рис. 1.3, при работе с отладочной платой осуществляется построение только сигнала `Data3` на мини-дисплее устройства с возможностью прокрутки и изменения масштаба посредством органа управления — джойстика.

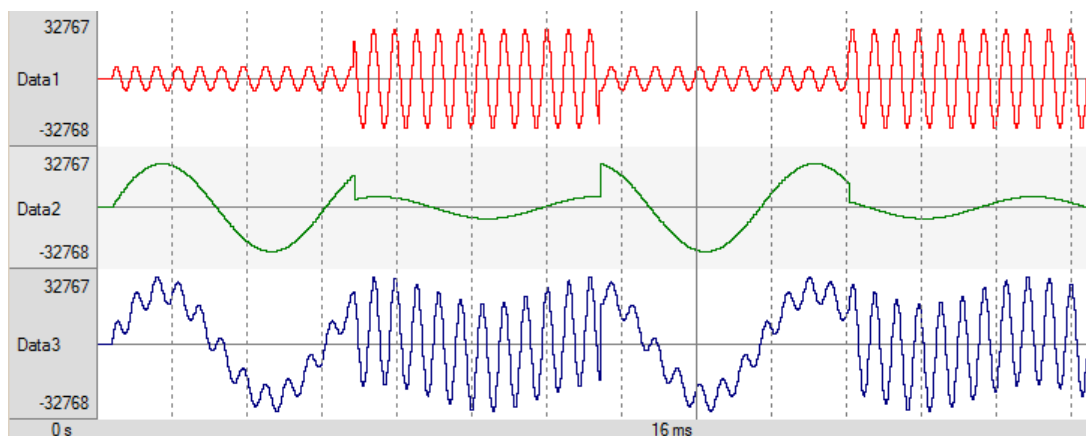


Рис. 1.3

Переключиться на проект **Project: board**. Проверить работу программы на реальном устройстве (откомпилировать проект, в открывшемся окне ARM Loader нажать кнопку "Load & Run"). Рекомендуется дальнейшие действия производить также на макете.

Изменить амплитуды суммируемых функций, чтобы максимальное значение амплитуды было более единицы: $\text{amp11} + \text{amp12} = 1.05 \dots 1.5$.

Привести в отчете полученные диаграммы, объяснить возникшие «артефакты».

Не изменяя значений скорректированных амплитуд, реализовать режим ограничения (насыщения), используя средства, заложенные в ядре Cortex-M4 — ассемблерную инструкцию SSAT или Си-функцию `__ssat`. Определение данной функции (в имени функции используется двойное подчеркивание):

```
int __ssat(int val, unsigned int sat);
```

где `val` — значение, к которому применяется операция насыщения; `sat` — число бит кода (от 1 до 32); функция возвращает `val`, ограниченное ном $-2^{\text{sat}-1} \dots 2^{\text{sat}-1} - 1$.

Привести в отчете диаграмму в режиме насыщения, скорректированный текст программы.

2. Построение графиков функций стандартной библиотеки Си

Скорректировать проект предыдущего пункта для отладочной платы, осуществить построение графика функции по одному из заданий, приведенных в табл. 1.4 (номер варианта соответствует номеру рабочего места или согласуется с преподавателем). Использовать то же самое число точек (NBuF), обеспечить масштабирование кривой на всё поле диаграммы, вывод названия функции. Не использовать насыщение. Если функция в каких-то точках стремится к беско-

нечности, ограничить (или обнулить) ее значения в некоторой окрестности значений аргумента. **Привести в отчете** скорректированные фрагменты программы, скриншоты дисплея.

Таблица 1.4. Задания для построения графиков функций

Вариант	
2.1	erf – функция ошибок. Диапазон изменения аргумента: $-2 \dots 2$.
2.2	Секанс (выразить через \cos). Диапазон изменения аргумента: $-\pi \dots +\pi$, особые точки: $-\pi/2, +\pi/2$.
2.3	expm1 – смещенная экспонента. Диапазон изменения аргумента: $-3 \dots 3$.
2.4	Функция $\sin(x)/x$. Диапазон изменения аргумента: $-10 \dots 10$. Реализовать корректное поведение вблизи 0.
2.5	\tan – тангенс. Диапазон изменения аргумента $-\pi \dots +\pi$, особые точки: $-\pi/2, +\pi/2$.
2.6	\cosh – гиперболический косинус. Диапазон изменения аргумента: $-5 \dots 5$.
2.7	Котангенс (выразить через \sin, \cos). Диапазон изменения аргумента: $-3\pi/4 \dots +3\pi/4$, особая точка: 0.
2.8	$\text{fmod}(x, 2)$ – дробная часть. Диапазон изменения аргумента: $-10 \dots 10$.
2.9	\log_{10} – десятичный логарифм. Диапазон изменения аргумента $0 \dots 10$, особая точка: 0.
2.10	Косеканс (выразить через \sin). Диапазон изменения аргумента: $-3\pi/4 \dots +3\pi/4$, особая точка: 0.
2.11	acosh – ареакосинус. Диапазон изменения аргумента $0 \dots 10$. Функция определена для аргумента > 1 .
2.12	\lgamma – натуральный логарифм от абсолютного значения гамма-функции. Диапазон изменения аргумента: $-10 \dots 10$.
2.13	\sin^3 – кубический синус. Построить 2-3 периода.
2.14	cbrrt – кубический корень. Диапазон изменения аргумента: $-100 \dots 1000$.
2.15	\log_b – показатель аргумента. Диапазон изменения аргумента: $-10 \dots 10$, особая точка: 0.
2.16	tgamma – гамма-функция. Диапазон изменения аргумента: $-0.01 \dots 0.01$.

3. Индивидуальное программирование вычислительных операций

Выбрать вариант по номеру рабочего места или получить от преподавателя (при необходимости согласовать также исходные данные).

Для всех параметров (кроме констант) определить переменные, числовые значения задать в программе компактным блоком. Учесть, что при вычислениях необходимо использовать систему единиц СИ. В заданиях с вариацией исходных значений не дублировать расчетную формулу, а включить ее в цикл или в функцию. Осуществить вывод на дисплей исходных данных и конечных результатов.

В качестве шаблона использовать следующий проект (убедиться в работе демонстрационного примера, удалить/заменить строки, помеченные как Тест, добавить свои операторы):

D:\ARM\Work\Lab1_indiv\MDK-ARM\individ.uvmpw

Продемонстрировать результаты работы преподавателю. **Привести в отчете** формулы, исходные значения, полученные результаты (в виде снимка дисплея), текст основной программы.

3.1. Рассчитать 3 точки оконной функции Барлетта-Ханна:

$$w(n) = a_0 - a_1 \cdot \left| \frac{n}{N-1} - 0.5 \right| - a_2 \cdot \cos\left(\frac{2\pi \cdot n}{N-1}\right)$$

для $a_0 = 0.62$; $a_1 = 0.48$; $a_2 = 0.38$; $N = 100$; $n = 10, 50$ и 90 .

3.2. Рассчитать резонансную частоту параллельного колебательного контура с потерями:

$$f = \frac{1}{2\pi\sqrt{LC}} \cdot \sqrt{\frac{(L/C) - R^2}{(L/C)}},$$

при $L = 15$ мкГн, $C = 22$ пФ, $R = 0.1$ кОм и 10 Ом. Вывести значение частоты с единицей измерения без использования степенного множителя в двух форматах: с максимальной точностью и с округлением до 3-4 знаков.

3.3. Рассчитать полный импеданс последовательной RLC-цепи:

$$Z = \sqrt{R^2 + \left(\omega L - \frac{1}{\omega C}\right)^2},$$

где $\omega = 2\pi f$; $f = 10$ и 50 кГц; $L = 2$ мГн; $C = 330$ нФ; $R = 200$ Ом.

3.4. Определить корни квадратного уравнения

$$0.1234 \cdot x^2 + 5.67 \cdot 10^{-3} \cdot x + 8.9 \cdot 10^{-5}$$

$$x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}.$$

Проверить (программными средствами): если уравнение не имеет вещественных корней, изменить знак одного из коэффициентов, на дисплей вывести также результаты проверки.

3.5. Найти коэффициент передачи активного фильтра верхних частот на двух частотах ω , отстоящих на $\pm 25\%$ от частоты среза $\omega_0 = 1/RC$:

$$k_{\omega} = \frac{K\omega^2 R^2 C^2}{\sqrt{(1 - (\omega CR)^2)^2 + (3 - K)^2 (\omega CR)^2}},$$

где $K = 2$ (коэффициент усиления); $C = 22$ нФ; $R = 1.5$ кОм.

3.6. Вычислить значение функции \sin по приведенной ниже приближенной формуле и с использованием библиотечной функции для значения аргумента, соответствующего 30° (перевести в радианы).

$$\sin x = x - \frac{x^3}{3!} + \frac{x^5}{5!} - \frac{x^7}{7!}.$$

3.7. Найти значение автокорреляционной функции для $\tau = 0.5$ мс :

$$\Psi(\tau) = \frac{1}{N} \sum_{i=0}^{N-1} f\left(T \frac{i}{N}\right) \cdot f\left(T \frac{i}{N} - \tau\right), \text{ где } f(t) = \sin\left(\frac{2\pi}{T} t\right),$$

$T = 1$ мс, $N = 50$.

3.8. Рассчитать емкость двухпроводной линии длиной $l = 1.5$ км, расстояние между проводниками 5 мм, радиус проводников 0.2 и 0.6 мм, диэлектрическая проницаемость вакуума $\varepsilon_0 = 8.85 \cdot 10^{-12}$ Ф/м, относительная диэлектрическая проницаемость $\varepsilon = 2.5$:

$$C = \frac{2\pi \cdot \varepsilon_0 \cdot \varepsilon \cdot l}{\ln(d/r)}.$$

3.9. Рассчитать энергию излучения (формула Планка):

$$u_{\lambda T} = \frac{2\pi \cdot h \cdot c^2}{\lambda^5} \cdot \frac{1}{\exp(hc/(k\lambda T)) - 1},$$

где $c = 2.998 \cdot 10^8$ м/с — скорость света; $k = 1.38 \cdot 10^{-23}$ Дж/К — постоянная Больцмана; $h = 6.626 \cdot 10^{-34}$ Дж·с — постоянная Планка; $\lambda = 700$ нм — длина волны (для красного света); $T = 273$ и 373 К — абсолютная температура.

3.10. Рассчитать три точки вольтамперной характеристики р-п перехода для напряжений $u = -0.5, 0$ и $+0.5$ В:

$$i = i_s \cdot \left[\exp \frac{u}{\varphi_T} - 1 \right], \quad \text{где } \varphi_T = \frac{kT}{q} \quad \text{— температурный потенциал;}$$

$i_s = 1$ мкА — ток насыщения; $T = 280$ К — абсолютная температура; $k = 1.38 \cdot 10^{-23}$ Дж/К — постоянная Больцмана; $q = 1.6 \cdot 10^{-19}$ Кл — элементарный заряд.

3.11. Найти три точки Гауссовой функции для $\mu = -0.5$, $\sigma^2 = 2$ и вариации $x = -1.5, -0.5, 0.5$:

$$f = \frac{1}{\sigma \sqrt{2\pi}} \exp \left(-\frac{(x - \mu)^2}{2\sigma^2} \right),$$

3.12. Рассчитать три точки Гауссова радиоимпульса:

$$y = \exp(-\alpha t^2) \cdot \cos(2\pi f_c t),$$

где $\alpha = 4.5$ — коэффициент длительности; $f_c = 20$ МГц — частота несущей; $t = -50$ нс, 50 нс, 200 нс — временные отсчеты.

3.13. Вычислить значение экспоненциальной функции, аппроксимированной рядом Маклорена, и той же функции с использованием стандартной библиотеки для аргумента $x = 3.3$:

$$e^x = \sum_{n=0}^3 \frac{x^n}{n!}.$$

3.14. Найти коэффициенты взаимной индуктивности двух антенных катушек с параметрами: числа витков $N_{1,2} = 5$ и 2 ; радиусы витков $R_{1,2} = 55$ и 33 мм; расстояния между катушками $x = 1, 5, 20$ мм; $\mu_0 = 4\pi \cdot 10^{-6}$ В·с / А·м — магнитная постоянная:

$$M = \frac{\mu_0 \cdot N_1 \cdot R_1^2 \cdot N_2 \cdot R_2^2 \cdot \pi}{2\sqrt{(R_2^2 + x^2)^3}}.$$

3.15. Определить коэффициент ослабления электромагнитной волны при прохождении через слой земли толщиной $d = 5$ и 15 м:

$$k = e^{-pd}, \quad \text{где} \quad p = \left[\frac{X \cdot B}{2} \cdot \left(\sqrt{1 + \left(\frac{\sigma \cdot 10^9}{B} \right)^2} - 1 \right) \right]^{1/2};$$

$X = 0.8 \cdot 10^{-16} \pi^2 \cdot f$; $B = 0.556 \varepsilon \cdot f$; $\sigma = 3 \cdot 10^{-2}$ См/м — проводимость земли; $\varepsilon = 5$ — диэлектрическая проницаемость земли; $f = 10$ МГц — частота волны.

3.16. Найти значения тока транзистора, соответствующие фазам 0° , 20° , 90° , 180° , 320° входного колебания, при аппроксимации переходной вольтамперной характеристики кусочно-линейной функцией:

$$i = \begin{cases} 0, & \text{при } U_0 + U_m \cdot \cos(\omega t) \leq U_H \\ S \cdot U_m \left(\cos(\omega t) - \frac{U_H - U_0}{U_m} \right), & \text{при } U_0 + U_m \cdot \cos(\omega t) > U_H \end{cases},$$

где $U_0 = 2.5$ В — постоянная составляющая входного колебания; $U_m = 4$ В — амплитуда входного колебания; $U_H = 1.5$ В — напряжение отсечки; $S = 90$ мА/В — крутизна характеристики.

Содержание отчета

Таблицы, диаграммы/скриншоты с заголовками, обозначениями осей и объяснениями возникающих особенностей (указаны в соответствующих пунктах выполнения работы).

Полный исходный текст программы с внесенными коррекциями (п. 1, 2), коррекции рекомендуется выделить, например, другим шрифтом или заливкой.

Текст основного модуля и результаты работы программы по индивидуальному заданию (п. 3).

Внимание! При оформлении текстов программ применять только моноширинные шрифты (Courier New, Consolas, Lucida Console и т.п.). Исключить автопереносы строк. При необходимости использовать альбомную ориентацию страницы. Образец оформления — в отдельных файлах с текстами программ и в приложении. Рекомендуется осуществлять коррекции и печать непосредственно из указанных файлов, а при необходимости копирования — предварительно выставить такие же поля документа.

Контрольные вопросы

1. Перечислите возможности среды разработки программного обеспечения.
2. Как осуществляется загрузка программы в реальный процессор?

3. Какие файлы и с какой целью входят в состав проекта?
4. Как задается начало рабочей программы? Какие дополнительные действия осуществляются перед ее стартом?
5. Что такое режим симуляции, какие действия возможны в этом режиме?
6. Чем отличается режим пошагового выполнения с заходом и без захода в функции?
7. Расскажите о порядке графической визуализации данных и сигналов.
8. Какие форматы данных обрабатываются процессором с максимальной производительностью? Как осуществляется работа с другими типами данных?
9. Каковы причины нарушения «плавности» изменения сигналов при увеличении их амплитуды?
10. Каковы причины возникновения некорректных результатов при арифметических операциях? Дайте рекомендации по их исправлению.
11. Как проверить флаги переполнения, переноса с использованием ассемблерных команд и операторов языка Си?
12. Когда используют программирование на языках высокого уровня и когда на языке ассемблера?
13. Охарактеризуйте основные арифметические, логические операции на языке Си, их обозначения, возможные форматы данных.
14. Расскажите о форматах вывода данных и правилах их задания в операторе `printf`.
15. Расскажите о правилах объявления, реализации, вызова функций, передачи им аргументов в языке Си.
16. Поясните принцип построения точек графика на дисплее отладочной платы. Каким образом графическая функция получает доступ к массиву данных значительного объема?
17. Дайте определение основным характеристикам процессора.
18. Какие виды памяти и какого объема встроены в процессорный кристалл, какую память можно подключить дополнительно, каков может быть ее объем?
19. Как распределяется адресное пространство процессора?
20. Охарактеризуйте внутренние регистры процессора (количество, разрядность, назначение).
21. Какие основные группы команд используются в процессорах с ядром ARM?
22. Дайте характеристику основных компонентов отладочной платы.

Лабораторная работа № 2

Порты ввода-вывода общего назначения. Таймеры.

Целью лабораторной работы является освоение работы с параллельными портами ввода-вывода общего назначения, таймерами, ознакомление с системой прерываний. Все задания выполняются в среде MDK-ARM (μVision 5) с практической проверкой на отладочной плате STM32-P407. Функциональная схема макета представлена на рис. 2.1.

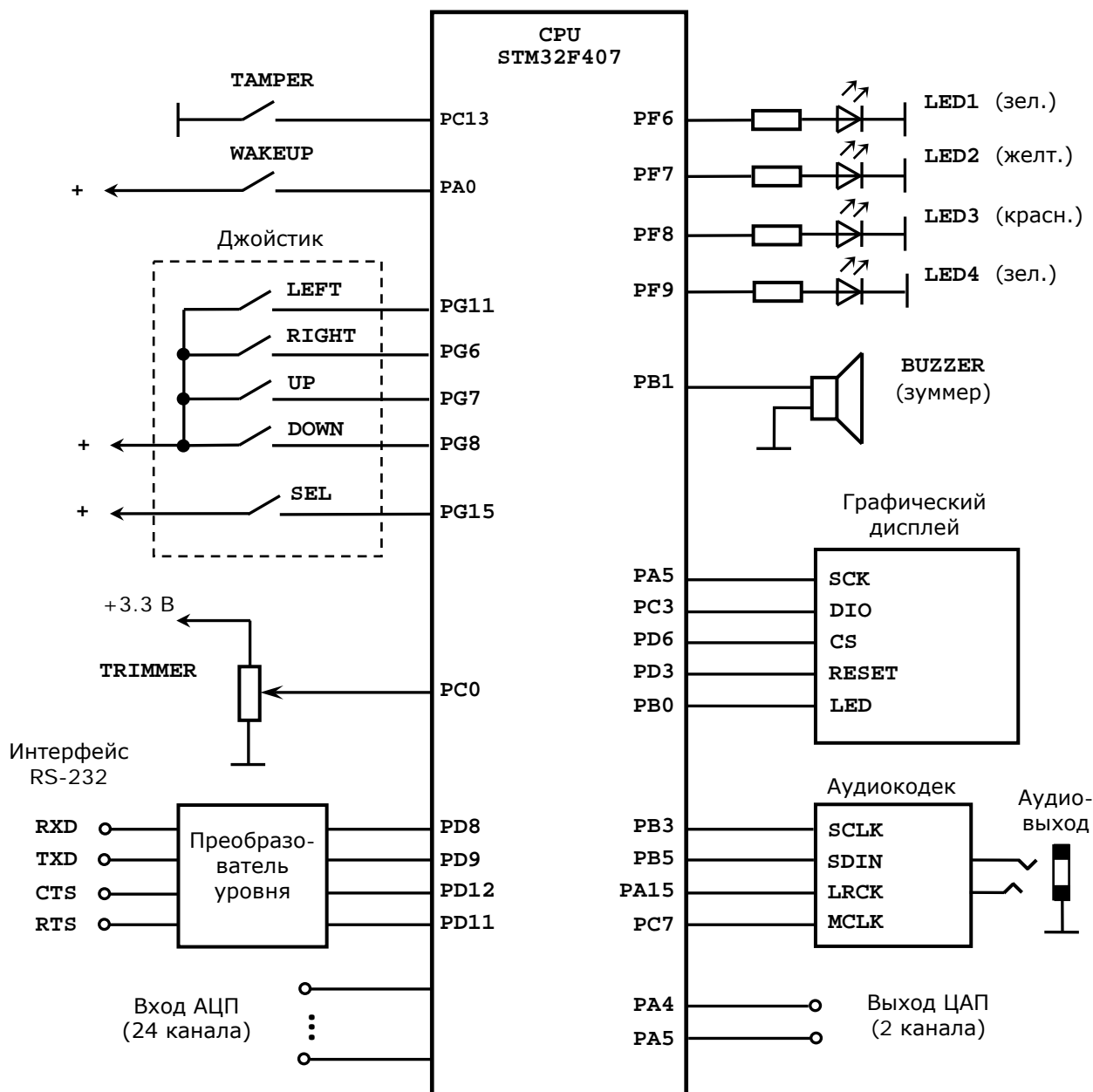


Рис. 2.1

Порядок выполнения

Перед началом работы выполнить процедуру очистки рабочего каталога, используя значок на рабочем столе **"Инициализация ARM"**. Запустить среду разработки значком **"Keil uVision5"**.

Напоминание. Для загрузки рабочей программы в отладочную плату после успешной компиляции проекта используется кнопка "Load & Run" в отдельном окне ARM Loader. Перед новой загрузкой необходимо завершить рабочую программу, как правило, кнопкой WAKEUP отладочной платы, или, в крайнем случае, кнопкой RESET.

1. Ввод-вывод логических сигналов через параллельные порты

Изучить функциональную схему лабораторного макета (рис. 2.1).

Открыть проект

D:\ARM\Work\Lab2_gpio\MDK-ARM\gpio.uvproj

В проекте реализованы: ввод состояний джойстика и кнопок, вывод этих состояний на светодиодные индикаторы и звукоизлучатель. Используются библиотечные подпрограммы временных задержек, построенных на основе подсчета пустых циклов.

Программирование портов, операции ввода-вывода осуществляются путем прямого обращения к регистрам периферийных устройств. Это способствует получению компактного программного кода, но требует изучения технических руководств по форматам данных в используемых регистрах, а кроме того, делает программу зависимой от конкретной аппаратной конфигурации. Другие подробности проекта описаны в виде комментария в основном файле `main.c`.

Откомпилировать, загрузить программу в отладочную плату, проверить работу, **привести в отчете** табл. 2.1, описав реакцию выходных устройств. При динамической реакции (мигание, переключение, звук) указать точные временные или частотные параметры (период/частота мигания/переключения, скважность и др.), определив их из текста программы и сопоставив с визуальным восприятием.

Для лучшего понимания действий программы рекомендуется внести некоторые коррекции: например, изменить соответствие позиций джойстика определенным индикаторам, изменить временные задержки (периоды, коэффициенты длительности). Оформление результатов этих действий не требуется. **В отчете привести** исходный текст программы.

Таблица 2.1. Реакция устройства при воздействии на органы управления

Воздействие на основной орган управления	Воздействие на доп. орган управления	Поведение выходных устройств с указанием временных или частотных параметров
Позиционирование джойстика	нет	
	Кнопка TAMPER	
Кнопка выбора джойстика	нет	<div>Яркость 1 Яркость 2</div> <div>Время включ. состояния: мс мс</div> <div>Время выключ. состояния:..... мс мс</div> <div>Период попарного переключения: мс</div> <div>Период смены яркости: мс</div>
	Кнопка TAMPER	

Примечание. Периодом переключения/смены следует считать полное время повторения состояния индикатора или излучателя, т.е. в него входят как включенное, так и выключенное состояния.

2. Использование таймеров для формирования временных интервалов и сигналов с широтно-импульсной модуляцией

Открыть проект

D:\ARM\Work\Lab2_tim_pwm\MDK-ARM\timer.uvproj

В проекте продемонстрированы некоторые режимы работы таймеров. Два таймера с номерами TIM10, TIM11 сконфигурированы на генерацию сигналов ШИМ с независимой настройкой периода и длительности импульсов, выходы каналов сравнения таймеров непосредственно подключены к двум светодиодным индикаторам (LED1, LED2). Таймер TIM4 работает в базовом режиме, генерируя прерывания с заданным временным интервалом, соответствующим полупериоду выходного сигнала, в подпрограмме же обслуживания этого прерывания осуществляется программное управление третьим светодиодным индикатором (LED3).

Запустить программу на выполнение. Наблюдая работу индикаторов, изобразить примерные временные диаграммы их переключения (рис. 2.2).



Рис 2.2

Из текста программы определить и **привести в отчете** точные параметры генерируемых сигналов (табл. 2.2).

Таблица 2.2. Параметры исходных сигналов

	Период	Коэффициент длительности
Таймер 10 (LED1)		
Таймер 11 (LED2)		
Таймер 4 (LED3)		

Не трогая управление индикатором LED3, добавить независимый канал управления индикатором LED4. Для этого использовать систему прерываний и свободный таймер с номером 2, 3, 5 или 7 (все указанные таймеры тактируются половинной системной частотой).

Скорректировать программу в соответствии с одним из вариантов, приведенных далее в табл. 2.3. Обеспечить по возможности наиболее точную реализацию заданных периодов/частот, используя возможности не только основного счетчика таймера, но и предварительного делителя. При наличии измерительного оборудования осуществить точное измерение параметров полученных выходных сигналов.

Продемонстрировать работу преподавателю.

На тех же или других диаграммах отобразить изменения. **Привести в отчете** диаграммы, данные об исходных и измененных параметрах, исходные и модифицированные фрагменты программы.

Таблица 2.3. Исходные данные для управления индикаторами

Вариант	LED1		LED2		LED3	LED4
	Период	Коэф.длит.	Период	Коэф.длит.	Период	Частота
2.1	100 мс	3 %	3.2 с	41 %	543 мс	2.1 Гц
2.2	200 мс	5 %	3.0 с	51 %	654 мс	3.2 Гц
2.3	300 мс	15 %	2.8 с	61 %	765 мс	4.3 Гц
2.4	400 мс	20 %	2.6 с	71 %	865 мс	5.4 Гц
2.5	500 мс	25 %	2.4 с	81 %	987 мс	6.5 Гц
2.6	600 мс	30 %	2.2 с	54 %	432 мс	7.6 Гц
2.7	700 мс	35 %	1.95 с	64 %	321 мс	8.7 Гц
2.8	800 мс	40 %	1.85 с	84 %	210 мс	9.8 Гц
2.9	900 мс	45 %	1.65 с	94 %	109 мс	10.9 Гц
2.10	999 мс	50 %	1.45 с	26 %	98 мс	40 Гц
2.11	1.1 с	55 %	1.25 с	21 %	87 мс	34.5 Гц
2.12	1.2 с	60 %	1.15 с	16 %	76 мс	32.1 Гц
2.13	1.3 с	65 %	777 мс	12 %	65 мс	23.4 Гц
2.14	1.4 с	70 %	555 мс	8 %	54 мс	20.2 Гц
2.15	1.5 с	75 %	333 мс	6 %	43 мс	16.6 Гц
2.16	1.6 с	80 %	222 мс	4 %	32 мс	13.3 Гц

3. Использование таймеров для генерации тональных сигналов

Открыть проект

D:\ARM\Work\Lab2_tim_ton\MDK-ARM\timer.uvproj

В проекте задействован таймер TIM3, который настроен на генерацию импульсов с коэффициентом длительности 50 % (или скважностью 2) и частотой, изменяемой при воздействии на органы управления (джойстик, кнопки). Сигнал напрямую передается звуковому излучателю. Генерируемые частоты соответствуют музыкальным тонам. **Привести в отчете** описание результата воздействия на каждый орган управления — длительность и частоту генерируемого сигнала.

В качестве простейших упражнений предлагается поменять тона, установленные по умолчанию, сгенерировать тона из более низких или высоких октав. На следующем шаге выполнить одно из следующих заданий:

3.1. При каждом нажатии на одну и ту же кнопку (джойстик) последовательно выдавать различные тона, составляющие заданную в виде массива кодов

музыкальную фразу.

3.2. При однократном воздействии на орган управления сформировать заданную массивом кодов последовательность звуковых колебаний (режим музыкального звонка).

3.3. Сформировать плавно увеличивающийся и уменьшающийся по частоте сигнал (режим сирены). Учесть, что «плавность» достигается дискретностью изменения частоты не более 1%.

Привести в отчете текст модифицированной программы.

4. Использование таймеров для измерения временных интервалов

Открыть проект

D:\ARM\Work\Lab2_tim_sys\MDK-ARM\timer.uvproj

Для отсчета времени выполнения фрагментов программы используется 24-разрядный системный таймер ядра ARM Cortex, работающий на той же частоте (168 МГц). Метки времени фиксируются функциями TimeMarkerX(). По нажатию кнопки TAMPER запомненные значения счетного регистра таймера выводятся на дисплей. Для повышения точности измеряемый фрагмент включен в цикл. **Привести в отчете** скриншот вывода на дисплей, табл. 2.4 с расчетными временами выполнения математических функций (привести все расчеты). При защите иметь текст исходной программы.

Таблица 2.4. Оценка времени выполнения математических функций

Функция	Число тактов таймера для цикла вычислений	Время всего цикла вычисления функции, мкс	Время однократного вычисления функции, мкс
1. ...			
2. ...			
...			

Содержание отчета

Таблицы, диаграммы, пояснения, исходные тексты программ со всеми произведенными коррекциями.

Внимание! При оформлении текстов программ применять только моноширинные шрифты (Courier New, Consolas, Lucida Console и т.п.). Исключить ав-

топереносы строк. При необходимости использовать альбомную ориентацию. Образец оформления — в файлах с текстами программ и приложениях.

Контрольные вопросы

1. Изобразите на функциональной схеме лабораторного макета подтягивающие резисторы. Каково их назначение?
2. Для чего нужны резисторы, включенные последовательно со светодиодными индикаторами?
3. Какие параметры задаются при конфигурировании порта общего назначения? Возможна ли одновременная работа частей порта на ввод и вывод?
4. Опишите регистры, используемые при работе с портами и их отдельными разрядами.
5. Насколько точными и почему являются временные задержки на основе циклических операций?
6. Перечислите все возможные режимы работы таймера.
7. Какие параметры задаются при конфигурировании таймера для работы в базовом режиме?
8. Какие параметры задаются при конфигурировании таймера для генерации ШИМ сигналов?
9. Как выбрать соотношение между коэффициентами деления основного счетчика и предделителя для повышения точности временных параметров генерируемого сигнала?
10. Каков предельный диапазон генерируемых частот и временных задержек в 16-разрядном и 32-разрядном таймерах?
11. Опишите механизм измерения временных интервалов посредством таймеров. Каковы минимальный и максимальный интервал для системного таймера ядра? Какова погрешность таких измерений и что ее обуславливает?
12. Как измерить длительность и период внешних сигналов?
13. Как осуществить амплитудную и частотную манипуляцию сигнала посредством таймеров?
14. В программе п. 1 заменить библиотечные подпрограммы временных задержек на основе циклических операций задержками, реализованными с помощью таймеров.
15. Дайте определения понятиям: запрос прерывания, вектор прерывания, обработчик прерывания.
16. Как осуществляется конфигурирование прерываний?

Лабораторная работа № 3

Аналого-цифровые периферийные устройства

Целью лабораторной работы является освоение работы с аналого-цифровой периферией: АЦП, ЦАП, аудиокодеком, а также изучение принципов передачи данных по системе прерываний и прямого доступа к памяти. Все задания выполняются в среде MDK-ARM (μ Vision 5) с практической проверкой на отладочной плате STM32-P407.

Порядок выполнения

Перед началом работы выполнить процедуру очистки рабочего каталога, используя значок на рабочем столе **"Инициализация ARM"**. Запустить среду разработки значком **"Keil uVision5"**.

Напоминание. Для загрузки рабочей программы в отладочную плату после успешной компиляции проекта используется кнопка «Load & Run» в отдельном окне ARM Loader. Перед новой загрузкой необходимо завершить рабочую программу, как правило, кнопкой WAKEUP отладочной платы, или, в крайнем случае, кнопкой RESET.

1. Измерения с помощью аналого-цифрового преобразователя

Открыть проект

D:\ARM\Work\Lab3_adc\MDK-ARM\adc.uvproj

В проекте демонстрируются: конфигурирование АЦП, диапазоны цифровых данных в смещенном и дополнительном кодах, особенности оцифровки при наличии помех, режим "оконного" аналогового компаратора (или аналоговой «сторожевой схемы»).

Исследование работы АЦП осуществляется в «статическом» режиме — для медленно меняющейся (практически постоянной) амплитуды сигнала, формируемой установленным на плате потенциометром (триммером). Регулятор выдает положительное напряжение в пределах 0...3.3 В (в реальном устройстве при прохождении цепей, например, с разделительными конденсаторами или трансформаторами такой сигнал будет знакопеременным). Результаты измерений выводятся на графический дисплей отладочной платы в десятичном формате. Переключение режима усреднений производится джойстиком.

Наличие внутренних помех, нестабильностей при аналого-цифровом преобразовании, плохого качества опорного напряжения, внешних флуктуаций во

входном сигнале и других факторов обуславливают нестабильность цифрового кода — так называемый шум младших разрядов. Оценить шум можно по величине изменения кода в течение некоторого времени (например, нескольких секунд). Произвести такую оценку для 3-4-х различных входных уровней, поместив в табл. 3.1 некоторое усредненное значение. Рекомендуется найти диапазон изменения кода в режиме выравнивания вправо, разность кодов перевести в двоичную систему, число значащих двоичных разрядов и будет соответствовать числу "шумовых". **Привести** таблицу в отчете.

Таблица 3.1. Оценка шумов при преобразовании

	Диапазон изменения кода	Число "шумовых" двоичных разрядов
Без усреднения		
С усреднением		

Проверить работу АЦП во всем диапазоне уровней входного напряжения с включенным или выключенным режимом усреднений (для чтения стабильных показаний можно использовать одну из позиций джойстика, временно останавливающую измерения). Обратит внимание на поведение красного индикатора. В табл. 3.2 и 3.3 занести только коды в 10-чной системе — соответствующие ячейки обведены жирной рамкой, остальные параметры рассчитать при домашней подготовке. Три строки с номерами 1, 5, 9 (первую, среднюю, последнюю), если не удастся ручным регулятором установить заданный уровень, заполнить теоретически. Уровень в строке 2 взять близким к уровню в строке 1, в строках 4 и 6 — близким к уровню в строке 5, в строке 8 — близким к уровню в строке 9. Одноименные строки в таблицах должны соответствовать одному и тому же уровню, т.е. должны заполняться одновременно!

Различие напряжений $U_{\text{АЦП}}$ и $U_{\text{ВХ}}$ (для АЦП), а также $U_{\text{ЦАП}}$ и $U_{\text{ВЫХ}}$ (для ЦАП) иллюстрируется схемой рис. 3.1.

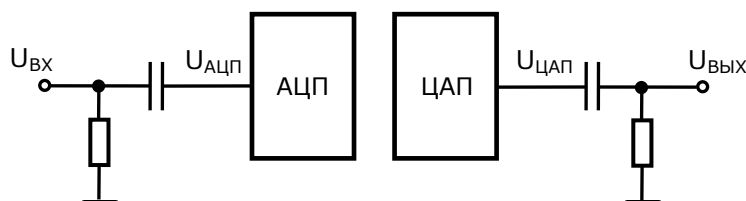


Рис. 3.1

Таблица 3.2. Напряжения и коды АЦП, выравненные влево

№	$U_{ВХ}, В$	$U_{АЦП}, В$	Смещенный код		Дополнительный код	
			10- чный	двоичный (16 бит)	10- чный	двоичный (16 бит)
1		3.3	(макс.)			
2						
3						
4						
5					0	
6						
7						
8						
9		0	(мин.)			

Таблица 3.3. Напряжения и коды АЦП, выравненные вправо

№	$U_{ВХ}, В$	$U_{АЦП}, В$	Смещенный код		Дополнительный код	
			10- чный	двоичный (16 бит)	10- чный	двоичный (16 бит)
1		3.3	4095			
2						
3						
4						
5					0	
6						
7						
8						
9		0	0			

Встроенные АЦП и ЦАП имеют соответственно входной и выходной диапазоны напряжений от 0 до опорного напряжения V_{ref} . Для внешних устройств, подключаемых через разделительные конденсаторы, напряжения из-за отсутствия постоянной составляющей будут знакопеременными. Рабочий режим АЦП при наличии разделительного конденсатора необходимо обеспечить дополнительно подачей на вход постоянного смещения.

Привести в отчете результаты измерений, формулы для расчета напряжений по одному из кодов.

Модифицировать программу для отображения на дисплее реальных уровней напряжения в вольтах, снимаемых с потенциометра и одного из внутренних источников (опорного или батарейного) или температурного датчика. Добиться, чтобы красный индикатор сигнализировал об отклонении напряжения более,

чем, например, $2.5 \text{ В} \pm 10\%$, при этом он может включаться, выключаться или мигать (характер поведения и значения порогов согласовать с преподавателем). Продемонстрировать работу, **привести в отчете** скорректированный текст программы.

2. Вывод сигналов с использованием ЦАП и аудиокодека

Открыть проект

D:\ARM\Work\Lab3_dac\MDK-ARM\dac.uvproj

В проекте показан пример вывода периодических сигналов, заданных табличным способом, через одноканальный ЦАП и двухканальный аудиокодек. Особенность основной программы состоит в том, что после проведения инициализации в ней не производится никаких операций, процессорное ядро переводится в состояние «сна». Все действия — передача отсчетов сигнала в ЦАП, передача отсчетов в интерфейс аудиокодека, реакция на нажатие кнопок — осуществляются по системе прерываний. При возникновении соответствующих запросов процессор пробуждается, выполняет подпрограмму обработки прерывания и возвращается в спящий режим. Предусмотрена возможность еще большей разгрузки процессора — передача отсчетов в режиме прямого доступа к памяти (DMA).

В предыдущей лабораторной работе был продемонстрирован способ измерения времени выполнения фрагментов программы посредством таймера. Здесь представлен другой способ — регистрация с помощью осциллографа специально сформированных сигналов-меток. В частности, при передаче отсчета в ЦАП осуществляется вывод сигнала лог. "1" в начале обработчика прерывания и вывод лог. "0" по завершении обслуживания. Длительность импульса позволяет оценить время активной работы процессора, пауза между импульсами — время бездействия. По этим данным легко определяется коэффициент загрузки процессора (обычно в %). Также подобные сигналы-метки очень полезны для целей отладки, позволяя отследить прохождение определенных участков программы.

Проверить работу устройства, измерить с помощью осциллографа периоды сигналов на выходе аудиокодека. Осциллограммы **привести в отчете**, указав на них значения периодов.

Если имеется возможность регистрации сигналов с выхода ЦАП и отладочного сигнала-метки, произвести определение их периодов (а для сигнала-метки — и длительности). При отсутствии такой возможности воспользоваться

иллюстрациями в файле с исходными текстами программ. **Привести в отчете** диаграммы сигналов с ЦАП и отладочного сигнала с необходимыми временными параметрами. При защите уметь объяснить различие сигналов ЦАП и кодека: по форме кривой, амплитуде, периоду. Определить коэффициент загрузки процессора, **привести в отчете** соответствующие расчеты.

Если есть возможность наблюдения сигналов на выходе ЦАП, убрать символы комментария с определения "#define DMA_ENA", перекомпилировать программу, проверить работу. **Указать в отчете** на изменения в сигналах, при невозможности экспериментальной оценки сделать это теоретически.

Модифицировать программу для получения одновременно на выходе ЦАП и двух выходах аудиокодека трех разных сигналов (их форма должна соответствовать исходным сигналам). Продемонстрировать работу, **привести в отчете** скорректированный текст программы.

3. Дискретизация сигналов с помощью АЦП

Открыть проект

D:\ARM\Work\Lab3_discret\MDK-ARM\discret.uvproj

В проекте реализована система ввода-вывода аналогового сигнала с потенциальной его цифровой обработкой (рис. 3.2). Исходный сигнал частотой $F_{\text{сигн}}$ генерируется самим же процессором с помощью ЦАП, выход которого соединен с входом АЦП. Выходной сигнал $F_{\text{вых}}$ формируется посредством аудиокодека. Изначально частота дискретизации $F_{\text{дискр}}$ АЦП выбрана значительно большей допустимой частоты дискретизации по теореме Котельникова, при этом значение $F_{\text{вых}}$ должно соответствовать $F_{\text{сигн}}$.

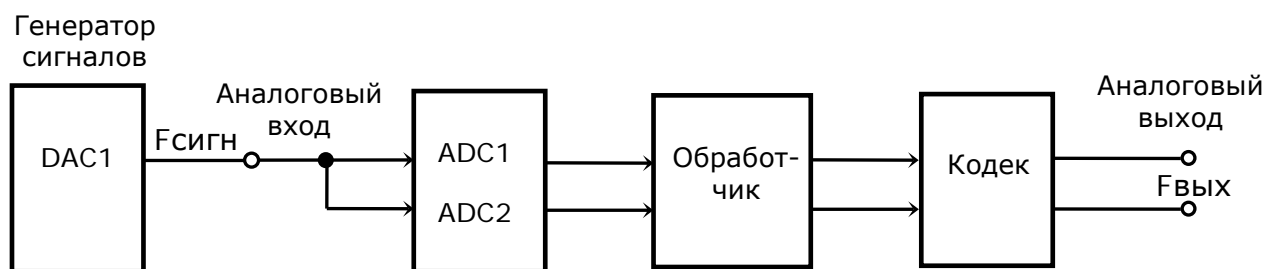


Рис. 3.2

По осциллографу на выходе кодека определить частоту сигнала $F_{\text{сигн}}$, из программы или посредством тестовых выходных сигналов-меток — частоту дискретизации АЦП ($F_{\text{дискр}}$). **Привести в отчете** осциллограммы сигналов на выходе при следующих частотах дискретизации, указав при этом реальную ча-

стоту наблюдаемого выходного сигнала $F_{\text{вых}}$:

- $F_{\text{дискр}} \gg F_{\text{сигн}}$ (исходный вариант);
- $F_{\text{дискр}} = (2.1 - 2.3) F_{\text{сигн}}$;
- $F_{\text{дискр}} = (1.2 - 1.5) F_{\text{сигн}}$.

Осуществить простейшую цифровую обработку сигнала только в одном из каналов, регистрируя осциллографом одновременно оба выходных сигнала.

Обязательные варианты:

- а) произвести однополупериодное и двухполупериодное выпрямление (детектирование);
- б) осуществить двухстороннее ограничение сигнала (сверху и снизу) на уровне 40...80% от максимальной амплитуды;

Варианты, согласуемые с преподавателем:

- в) реализовать систему шумоподавления (обнулить сигнал уровня менее 5-20%), одним из органов управления изменять пороговый уровень;
- г) осуществить амплитудную манипуляцию сигнала с некоторым программно заданным периодом (в 5...10 раз большим периода сигнала), кнопкой/джойстиком изменять величину периода;
- д) осуществить умножение частоты входного сигнала в 2 и 4 раза (рекомендуется использовать тригонометрическую формулу $\sin x = (1 - \cos 2x)/2$);
- е) измерить амплитуду (пиковое значение) входного сигнала, вывести результат на дисплей или включить индикатор при превышении некоторого уровня;
- ж) реализовать систему автоматической регулировки — обеспечить одинаковую амплитуду выходного сигнала при различных амплитудах входного;
- з) осуществить амплитудную модуляцию гармоническим сигналом частотой, в несколько раз меньшей частоты входного сигнала, нажатием одной из кнопок изменять коэффициент модуляции;
- и) осуществить временную задержку на $1/4...1/2$ периода, одним из органов управления изменять величину задержки;
- к) осуществить амплитудную модуляцию треугольным сигналом, кнопкой изменять период модулирующего сигнала;
- л) осуществить амплитудную модуляцию пилообразным сигналом, кнопкой управлять типом фронта пилообразного сигнала — возрастающим или убывающим;
- м) преобразовать входной синусоидальный сигнал в треугольный с той же

частотой и фазой;

н) осуществить деление частоты в 2 раза, например, пропуская на выход без изменений нечетные периоды и инвертируя четные;

о) осуществить амплитудную модуляцию ступенчатым сигналом (3-4 ступени); нажатием кнопки изменять амплитуду каждого четного периода;

п) осуществлять попеременный вывод двух типов сигналов: исходного и сдвинутого на четверть периода (\sin , \cos), обеспечить вывод целого числа периодов;

р) осуществить фазо-импульсную модуляцию, органом управления изменять величину модуляции;

с) формировать на выходе ШИМ сигнал, модулированный входным сигналом;

т) двумя позициями джойстика увеличивать/уменьшать амплитуду выходного сигнала с шагом 5-10%.

Продемонстрировать работу, **привести в отчете** диаграммы выходных сигналов до и после обработки, скорректированный текст программы.

Содержание отчета

Тексты заданий, таблицы, диаграммы, тексты программ (указаны в соответствующих пунктах выполнения работы)

Внимание! При оформлении текстов программ применять только моноширинные шрифты (Courier New, Consolas, Lucida Console и т.п.). Исключить автопереносы строк. При необходимости использовать альбомную ориентацию страницы. Образец оформления — в отдельных файлах с текстами программ и в приложении. Рекомендуется осуществлять коррекции и печать непосредственно из указанных файлов, а при необходимости копирования — предварительно выставить такие же поля документа.

Контрольные вопросы

1. Перечислите основные характеристики АЦП. Какие параметры задаются при конфигурировании АЦП?
2. Опишите функционирование и приведите передаточную характеристику стандартного аналогового компаратора. Какими дополнительными свойствами обладает «оконный» аналоговый компаратор («аналоговая сторожевая схема»)?

3. Как осуществляется работа с несколькими входными аналоговыми сигналами, в том числе при использовании только одного преобразователя?
4. Перечислите основные характеристики ЦАП. Какие параметры задаются при конфигурировании ЦАП?
5. Перечислите основные характеристики аудиокодека. Какие параметры задаются при его конфигурировании?
6. Чем обусловлено различие форм кривых одного и того же сигнала на выходе ЦАП и выходе аудиокодека?
7. Дайте рекомендации по уменьшению искажений и снижению уровня шумов как при оцифровке, так и генерации аналоговых сигналов.
8. Какими рекомендациями следует руководствоваться при выборе частоты дискретизации и разрядности АЦП-ЦАП?
9. Какую максимальную частоту дискретизации возможно получить при использовании лабораторного макета?
10. Покажите все необходимые компоненты на входе АЦП для работы со знакопеременными сигналами, сигналами с большим диапазоном значений напряжений.
11. Расскажите о правилах подключения, алгоритме работы при наличии многоканальных сигналов.
12. Предложите техническое решение на основе лабораторного макета для формирования, например, 8-канального звукового сигнала.
13. Дайте определения понятиям: запрос прерывания, вектор прерывания, обработчик прерывания. Как осуществляется конфигурирование прерываний? Что такое флаги запросов прерываний?
14. Опишите подробно процесс перехода на обслуживание прерывания и возврат из него. Как происходит обслуживание одновременно нескольких запросов? Оцените временной интервал между запросом и началом обслуживания, насколько он является стабильным.
15. Что такое прямой доступ к памяти? Какие модули могут работать в этом режиме? Как происходит работа одновременно нескольких каналов прямого доступа по общим шинам?
16. Сравните коэффициент загрузки процессора, например, в задачах генерации сигналов: а) без использования системы прерываний и DMA; б) при использовании системы прерываний; в) при использовании системы DMA.
17. Как обеспечить непрерывное аналого-цифровое и цифро-аналоговое преобразование, а также цифровую обработку нескольких сигналов одним процессорным ядром?

Список литературы

1. Новожилов О.П. Основы микропроцессорной техники. В 2 т.— М.: ИП Радиософт, 2007. Т. 1. — 432 с.
2. Микушин А.В., Сажнев А.М., Сединин В.И. Цифровые устройства и микропроцессоры. — СПб.: БВХ-Петербург, 2010. — 832 с.
3. Джозеф Ю. Ядро Cortex-M3 компании ARM. Полное руководство / Джозеф Ю; пер. с англ. А.В.Евстифеева. — М.: Додэка-XXI, 2012. — 552 с.
4. Богаченков А.Н. Процессоры с ядром ARM в лабораторном практикуме и курсовом проектировании [Электронный ресурс] : Учебное пособие по выполнению лабораторных и курсовых работ — М.: МИРЭА, 2016.
5. STM32-P407 Development board. User Manual (на англ. языке) [Электронный ресурс], 2012. — Режим доступа:
<https://www.olimex.com/Products/ARM/ST/STM32-P407/>.
6. Библиотека цифровой обработки сигналов и стандартной периферии:
http://www.st.com/content/st_com/en/products/embedded-software/mcus-embedded-software/stm32-embedded-software/stm32-standard-peripheral-libraries/stsw-stm32065.html
7. Trevor Martin. The Insider's Guide to the STM32 ARM-based Microcontroller // Hitex (UK) Ltd (www.hitex.com), 2009. Перевод: Ознакомительное руководство по ARM-микроконтроллерам Cortex-M3 [Электронный ресурс]. — Режим доступа:
http://www.gaw.ru/html.cgi/txt/doc/micros/arm/cortex_arh/index.htm.
8. Магда Ю.С. Современные микроконтроллеры. Архитектура, программирование, разработка устройств. — М.: ДМК, 2010. — 228 с.
9. Иванов Р. Лидер по производительности среди ядер Cortex-M4 – STM32F4xx. — Новости электроники, 2012, № 2, с. 17-22.
10. Бугаев В., Мусиенко М., Крайнык Я. Визуализация возможностей: графический генератор кода STM32CubeMX — Новости электроники, 2014, № 11, с. 19-25.
11. Мошкин В.В. Микропроцессоры и микроконтроллеры в устройствах и системах [Электронный ресурс]: метод. указания по выполнению курсовых работ / В. В. Мошкин. — М.: РТУ МИРЭА, 2018. — Электрон. опт. диск (ISO)
<https://library.mirea.ru/share/2965>
12. Керниган Б., Ритчи Д. Язык программирования С. : Пер. с англ. — М.: Издательский дом "Вильямс", 2009. — 304 с.

ПРИЛОЖЕНИЕ. ИСХОДНЫЕ ТЕКСТЫ ПРОГРАММ

Тексты к лабораторной работе № 1

Проект ..\Lab1_diagr\MDK-ARM\multiproject.uvprw

1. Ознакомление со средой разработки, структурой проекта, операторами языка Си
2. Построение графиков функций стандартной библиотеки Си

Исходный модуль ..\Lab1_diagr\main.c

```
//-----  
//  
// УЧЕБНЫЙ ПРОЕКТ  
// АРИФМЕТИЧЕСКИЕ, ЛОГИЧЕСКИЕ ОПЕРАЦИИ. ПОСТРОЕНИЕ ГРАФИКОВ ФУНКЦИЙ.  
// Графические построения: на дисплее отладочной платы,  
// в режиме симуляции - средствами среды.  
// Copyright (C) 2016 МИРЭА  
//  
//-----  
/*  
    ОСНОВНЫЕ ОПЕРАЦИИ  
  
    Компиляция проекта: клавиша F7 (кнопка Build или меню Project/Build)  
    Вход в режим отладки: клавиша Ctrl+F5 (кнопка или меню Debug/Start/Stop Debug Session)  
    Вставка точки останова: клавиша F9 (кнопка или меню Debug/Insert/Remove Breakpoint)  
    Шаговое выполнение с заходом  
    в вызываемые функции: клавиша F11 (кнопка Step или пункт меню Debug/Step)  
    Шаговое выполнение без захода  
    в вызываемые функции: клавиша F10 (кнопка Step Over или пункт меню Debug/Step Over)  
    Непрерывное выполнение: клавиша F5 (кнопка Run или пункт меню Debug/Run)  
    Возврат из режима отладки: клавиша Ctrl+F5 (кнопка или меню Debug/Start/Stop Debug Session)  
  
    ПОРЯДОК ВИЗУАЛИЗАЦИИ ГРАФИКА ФУНКЦИИ  
  
    Откомпилировать проект (F7).  
    Инициировать режим отладки (Ctrl+F5).  
    Поставить точку останова на последний оператор - while(1) - должна присутствовать красная метка слева.  
    Открыть логический анализатор: меню View / Analysis Windows / Logic Analyzer.  
    Если левое поле анализатора пусто, в тексте программы последовательно выделить имена  
    Data1, Data2, Data3 и перетащить их на это поле анализатора.  
    Запустить программу на выполнение (F5).  
    После остановки отрегулировать масштаб диаграмм полем Zoom (In/Out/All).  
    Возвратиться из отладочного режима (Ctrl+F5).  
*/  
//-----  
// Заголовочные файлы  
#include <stdint.h> //Объявления типов данных  
#include <math.h> //Функции стандартной библиотеки C/C++  
#include "arm_math.h" //Функции стандартной библиотеки CMSIS  
#include "lcd.h" //Функции работы с дисплеем  
#include "values.h" //Числовые значения для операндов  
#ifndef SIMUL  
#include "stm32_p407.h" //Файл конфигурации отладочной платы STM32-P407  
#endif // (обходится при компиляции в режиме симуляции)  
  
// Прототипы вызываемых функций  
int16_t func(float arg); //В этом файле  
void LCD_DiagramDraw(short* Samples, int begindex, int length); //В файле diagr.c  
  
// Глобальные параметры (доступны всем подпрограммам и другим модулям программы)  
#define NBuf 1000 //Длина буфера данных (константа)  
int16_t BufData[NBuf]; //Буфер данных (массив размером Nbuf 16-разрядных элементов)  
int16_t Data1, Data2, Data3; //Текущие значения функций  
int8_t y1, y2; //Промежуточные значения  
int16_t y3;  
int dIndex, dLength, dMin; //Параметры для построения диаграммы
```



```

//-----
// ГЛАВНАЯ ФУНКЦИЯ
// Имеет стандартное имя: main.
// При запуске программы управление передается на первый оператор этой функции.
// Главная функция не должна иметь выхода.
int main()
{
    //Объявление локальных переменных
    int i; //Счетчик цикла
    int8_t x1, x2, x3, x4; //8-битные операнды
    float ampl1 = 0.2; //Амплитуда для функции 1
    float ampl2 = 0.8; //Амплитуда для функции 2
    float arg1 = 0, arg2 = 0; //Начальные значения аргументов функций
    // 1 вариант: задание шага изменения аргумента (используется для периодических функций)
    float step1 = PI/11, //Величины приращений аргументов, где PI = 3.141592...
        step2 = PI/110; // (операцию деления здесь осуществляет компилятор)
    // 2 вариант: задание минимального и максимального значений аргумента
    //float arg1_min = 0, arg1_max = 90*PI,
    // arg2_min = 0, arg2_max = 9*PI;

    //Присвоение начальных значений 8-битным операндам
    x1 = 0x12; //Константа в 16-ричной системе
    x2 = -34; //Константа в 10-ричной системе
    x3 = value1; //Значения, заданные в заголовочном файле
    x4 = value2;

    //Примеры записи арифметических и логических операций
    y1 = x1 + x2; //Сложение
    y2 = x3 - x4; //Вычитание
    y3 = y1 * y2; //Умножение
    y1 = y3 / x3; //Целочисленное деление
    y2 = y3 % x3; //Остаток от деления

    x2 = ~x2; //НЕ, инверсия
    y1 = x2 & x4; //И, логическое умножение, конъюнкция
    y2 = x1 | x3; //ИЛИ, логическое сложение, дизъюнкция
    y2 = y2 ^ y1; //Исключающее ИЛИ, сложение по модулю 2, неравнозначность

    //Цикл вычисления функций
    for (i = 0; i < NBuf; i++) //Счетчик (i) меняется от 0 до NBuf-1 с шагом 1
    {
        //::::: Начало одного цикла вычислений ::::::::::::::::::::

        if (i < NBuf/4 || (i >= NBuf/2 && i < NBuf*3/4)) //Ветвление по 4 интервалам изменения аргумента
        { Data1 = ampl1 * func(arg1); //Функция 1 от аргумента 1
          Data2 = ampl2 * func(arg2); //Функция 2 от аргумента 2
        }
        else
        { Data1 = ampl2 * func(arg1); //Функция 1 от аргумента 1
          Data2 = ampl1 * func(arg2); //Функция 2 от аргумента 2
        }
        Data3 = Data1 + Data2; //Функция 3 как сумма функций 1 и 2

        // 1 вариант: приращение аргумента на заданный шаг
        arg1 += step1; //Изменение аргумента 1
        arg2 += step2; //Изменение аргумента 2
        // 2 вариант: вычисление аргумента по текущему номеру отсчета
        //arg1 = arg1_min + (arg1_max - arg1_min) * i / NBuf;
        //arg2 = arg2_min + (arg2_max - arg2_min) * i / NBuf;

        //::::: Конец одного цикла вычислений ::::::::::::::::::::

        BufData[i] = Data3; //Сохранение значения функции 3 в буфере
    }

    //Следующий фрагмент, ограниченный директивами условной компиляции (#if.. #else),
    // компилируется только для режима симуляции (Project: simulator)
#ifdef SIMUL //=====
    while (1); //Остановка выполнения: оператор представляет собой
              // бесконечный цикл, не позволяющий выйти из функции main
#else //=====
    //Следующий фрагмент, ограниченный директивами условной компиляции (#else..#endif),
    // компилируется только для отладочной платы (Project: board)

```

```

//Инициализация кнопок и джойстика
STM_PBInit(BUTTON_TAMPER, BUTTON_MODE_GPIO); //Кнопка TAMPER (здесь не используется)
STM_PBInit(BUTTON_WAKEUP, BUTTON_MODE_GPIO); //Кнопка WAKEUP (для завершения программы)
STM_PBInit(BUTTON_RIGHT, BUTTON_MODE_GPIO); //Позиции джойстика
STM_PBInit(BUTTON_LEFT, BUTTON_MODE_GPIO); // - " -
STM_PBInit(BUTTON_UP, BUTTON_MODE_GPIO); // - " -
STM_PBInit(BUTTON_DOWN, BUTTON_MODE_GPIO); // - " -
STM_PBInit(BUTTON_SEL, BUTTON_MODE_GPIO); // - " -

//Инициализация дисплея
LCD_Init();

//Значения по умолчанию для вывода диаграммы
dIndex = 0; //Начальный номер отсчета
dMin = 15; //Минимальное число отсчетов
dLength = dMin*8; //Текущее число отсчетов

//Начальное построение диаграммы на дисплее
LCD_DiagramDraw(BufData, dIndex, dLength);

//Бесконечный цикл в основной программе
while (true)
{
    //Проверка положения джойстика "Влево" и возможности смещения диаграммы к началу
    if (STM_PBGetState(BUTTON_LEFT) && dIndex > 0)
    { if (dIndex - dLength > 0) dIndex -= dLength;
      else dIndex = 0;
      LCD_DiagramDraw(BufData, dIndex, dLength);
      while (STM_PBGetState(BUTTON_LEFT)); //Ожидание отпускания кнопки
    }
    //Проверка положения джойстика "Вправо" и возможности смещения диаграммы к концу
    if (STM_PBGetState(BUTTON_RIGHT) && dIndex + dLength < NBuf)
    { if (dIndex + dLength*2 < NBuf) dIndex += dLength;
      else dIndex = NBuf - dLength;
      LCD_DiagramDraw(BufData, dIndex, dLength);
      while (STM_PBGetState(BUTTON_RIGHT));
    }
    //Проверка положения джойстика "Вверх" и возможности раздвижки диаграммы
    if (STM_PBGetState(BUTTON_UP) && dLength > dMin)
    { dLength /= 2;
      LCD_DiagramDraw(BufData, dIndex, dLength);
      while (STM_PBGetState(BUTTON_UP));
    }
    //Проверка положения джойстика "Вниз" и возможности сдвижки диаграммы
    if (STM_PBGetState(BUTTON_DOWN) && dLength < NBuf/2)
    { dLength *= 2;
      if (dIndex + dLength > NBuf) dIndex = NBuf - dLength;
      LCD_DiagramDraw(BufData, dIndex, dLength);
      while (STM_PBGetState(BUTTON_DOWN));
    }

    //Проверка воздействия на кнопку WAKEUP, при нажатии - сброс, эквивалентный
    // нажатию кнопки RESET (здесь - завершение данной программы)
    if (STM_PBGetState(BUTTON_WAKEUP)) NVIC_SystemReset();
}

#endif //Конец компиляции операторов работы с отладочной платой
}

//-----
// ФУНКЦИЯ
// Обратите внимание, что аргумент arg - вещественное число (с плавающей точкой),
// значение же функции приводится к целочисленному 16-разрядному значению
int16_t func(float arg)
{
    return sinf(arg) * 32767;
}
//-----

Исходный модуль ..\Lab1_diagr\diagr.c
//-----
//
// УЧЕБНЫЙ ПРОЕКТ
// Модуль построения диаграмм

```

```
//
// Copyright (C) 2016 МИРЭА
//
//-----

#include "lcd.h"                //Функции для работы с дисплеем

#define MAX_SAMPLE 0x7FFF       //Максимальная амплитуда отсчетов
unsigned int ColorText = 0x0F0; //Цвет символов текста
unsigned int ColorAxis = 0xF00; //Цвет осей диаграммы
unsigned int ColorCurve = 0xFFF; //Цвет основной кривой
//Границы диаграммы в экранных пикселах:
int r_left = 10;               // слева
int r_top = 30;                // сверху
int r_right = 120;              // справа
int r_bottom = 120;            // снизу

//-----
// ПОДПРОГРАММА ПОСТРОЕНИЯ ГРАФИКА ФУНКЦИИ
// Входные параметры:
// Samples - указатель на массив значений функции
// beginindex - индекс начальной точки
// length - число точек
// Для экранных координат начало (0,0) находится в левом верхнем углу
void LCD_DiagramDraw(short* Samples, int beginindex, int length)
{
    short sample;
    int i, x, y;
    int Y0 = r_top + (r_bottom - r_top) / 2; //Y-координата горизонтальной оси
    LCD_Clear(1); //Очистка экрана
    LCD_FontColor(ColorText); LCD_FontSize(9); //Задание цвета и размера шрифта заголовка
    LCD_print(" Диапазон отсчетов\r\n" //Вывод заголовка
              " %d...%d", beginindex, beginindex + length);
    LCD_PenColor(ColorAxis); //Задание цвета осей
    LCD_Rectangle(r_left-1, r_left-1, r_top-1, r_bottom+1); //Построение вертикальной оси
    LCD_Rectangle(r_left-1, r_right+1, Y0, Y0); //Построение горизонтальной оси
    LCD_PenColor(ColorCurve); //Задание цвета точек основной кривой

    Samples += beginindex; //Начальный адрес значений функции
    for (i = 0; i < length; i++) //Цикл по всем выводимым точкам
    {
        sample = *Samples++; //Чтение значения из массива с инкрементом
        //указателя
        x = r_left + (r_right - r_left) * i / length; //X-координата точки
        y = Y0 - (r_bottom - Y0) * sample / MAX_SAMPLE; //Y-координата точки
        LCD_Rectangle(x, x+1, y, y+1); //Построение точки
    }
}

//-----
```

Заголовочный файл ..\Lab1_diagr\values.h

```
//-----
//
// ИНИЦИАЛИЗАЦИОННЫЕ ЗНАЧЕНИЯ
//
//-----

#define value1 0x78
#define value2 0xCD
```

Проект ..\Lab1_indiv\MDK-ARM\individ.uvmpw

3. Индивидуальное программирование вычислительных операций

Исходный модуль ..\Lab1_indiv\main.c

```
//-----
//
// УЧЕБНЫЙ ПРОЕКТ
```

```

// Программирование вычислительных операций, форматированный вывод
//
// В демонстрационном примере осуществляется нахождение среднего арифметического
// элементов массива, предварительно заполненного случайными числами.
// Исходные элементы - целые 32-разрядные. Два варианта: элементы знаковые и беззнаковые.
//
// При выполнении собственного задания закомментировать или удалить строки с пометкой Тест
//
// В режиме симуляции активно макроопределение SIMUL (задано в свойствах проекта).
// Возможен вывод в окно, которое открывается/закрывается в режиме отладки
// выбором пункта меню "View / Serial Windows / Debug (printf) Viewer"
//
//-----

#ifndef SIMUL
#include "stm32_p407.h"           //Файл конфигурации отладочной платы STM32-P407
#include "lcd.h"                 //Функции для работы с дисплеем
#endif
#include <math.h>                //Математические функции стандартной библиотеки C/C++
#include "arm_math.h"            //Определения и функции библиотеки CMSIS DSP Library
#include <stdio.h>                //Функции ввода-вывода (в данном проекте - printf)
#include <stdlib.h>               //Функции общего назначения (в данном проекте - rand)
#include "debug_printf.h"        //Определения для вывода в отладочное окно

int TestData[256];               //Тест (массив исходных данных)

//----- Главная функция -----

int main()
{
    int* address = TestData;      //Тест (указатель на начало массива)
    int number = sizeof(TestData)/sizeof(int); //Тест (число элементов массива)
    float sum;                   //Тест (сумма элементов)
    float mean;                  //Тест (среднее арифметическое)
    int i;                       //Тест (счетчик-индекс элементов)
    for (i = 0; i < number; i++) //Тест
        TestData[i] = rand() << 1; //Тест

    #ifndef SIMUL
        //Инициализация кнопки WAKEUP выхода из программы
        STM_PBInit(BUTTON_WAKEUP, BUTTON_MODE_GPIO);

        //Инициализация дисплея
        LCD_Init(); LCD_Clear(1);
        LCD_FontSize(9);
    #endif

    //Решаемая задача
    //=====
    for (i = 0, sum = 0; i < number; i++) //Тест
    {                                     //Тест
        sum += (float)*(address+i);      //Тест
    }                                     //Тест
    mean = sum / number;                 //Тест
    LCD_printf(" *** РЕЗУЛЬТАТ ***\r\n"); //Тест
    LCD_printf("Вариант 1 (знак)\r\n N=%d\r\n sum =%g\r\n mean=%g\r\n", //Тест
        number, sum, mean);             //Тест
    for (i = 0, sum = 0; i < number; i++) //Тест
    {                                     //Тест
        sum += (float)(unsigned int)*(address+i); //Тест
    }                                     //Тест
    mean = sum / number;                 //Тест
    LCD_printf("Вариант 2 (6/зн)\r\n N=%d\r\n sum =%g\r\n mean=%g\r\n", //Тест
        number, sum, mean);             //Тест
    //=====

    //Окончание
    while (1)
    {
        #ifndef SIMUL
            //Проверка нажатия кнопки WAKEUP завершения программы
            if (STM_PBGetState(BUTTON_WAKEUP)) NVIC_SystemReset();
        #endif
    }
}

```

Подключаемый модуль ..\Lab1_indiv\debug_print.h

```
#ifndef _DEBUG_PRINTF_H
#define _DEBUG_PRINTF_H

////////////////////////////////////
// Следующий блок операторов используется при симуляции для перенаправления //
// вывода функции printf вместо последовательного порта в отладочное окно //
// "Debug (printf) Viewer". Окно открывается/закрывается в режиме отладки //
// выбором пункта меню "View / Serial Windows / Debug (printf) Viewer" //
// //
#ifdef SIMUL //
#define ITM_Port8(n) (*((volatile unsigned char *) (0xE0000000+4*n))) //
#define ITM_Port16(n) (*((volatile unsigned short *) (0xE0000000+4*n))) //
#define ITM_Port32(n) (*((volatile unsigned long *) (0xE0000000+4*n))) //
#define DEMCR (*((volatile unsigned long *) (0xE000EDFC))) //
#define TRCENA 0x01000000 //
//
struct __FILE { int handle; }; //
FILE __stdout, __stdin; //
//
int fputc(int ch, FILE *f) //
{ //
    if (DEMCR & TRCENA) //
    { while (ITM_Port32(0) == 0); //
        ITM_Port8(0) = ch; //
    } //
    return (ch); //
} //
//
#define LCD_printf(...) printf(__VA_ARGS__) //
#else //
#define LCD_printf(...) LCD_print(__VA_ARGS__) //
#endif //
////////////////////////////////////

#endif /* _DEBUG_PRINTF_H */
```

Фрагменты библиотечного модуля ..\Libraries\Board\lcd.c

```
/*
 * Функция: LCD_Rectangle
 * Построение закрашенного прямоугольника
 * Параметры: x1, x2, y1, y2 - координаты в пикселах (0...), включительно
 * Возврат: Нет
 */
void LCD_Rectangle(unsigned int x1, unsigned int x2, unsigned int y1, unsigned int y2)
{
    unsigned int i, j;
    LCD_SET_WINDOW(x1,x2,y1,y2);
    j = (y2 - y1 + 1) * (x2 - x1 + 1);
    for (i = 0; i < j; i++) LCD_WRITE_PIXEL(_Pen_Color);
    LCD_FLUSH_PIXELS();
}

/*
 * Функция: LCD_Picture
 * Построение растрового изображения
 * Параметры: pData - указатель на пиксельные данные в формате 0x0BGR, порядок
 * следования данных: слева направо, сверху вниз.
 * Возврат: Нет
 */
void LCD_Picture(unsigned short* pData)
{
    Int32U Data;
    Data = 0 | ((GLCD_HORIZONTAL_SIZE - 1) << 8);
    GLCD_SendCmd(CASET, (pInt8U)&Data, 0);\
    Data = 0 | ((GLCD_VERTICAL_SIZE - 1) << 8);
    GLCD_SendCmd(RASET, (pInt8U)&Data, 0);
    GLCD_SendCmd(RAMWR, (pInt8U)pData, GLCD_HORIZONTAL_SIZE * GLCD_VERTICAL_SIZE * 2);
}
```

Тексты к лабораторной работе № 2

Проект ..\Lab2_gpio\MDK-ARM\gpio.uvproj

1. Ввод-вывод логических сигналов через параллельные порты

Исходный модуль ..\Lab2_gpio\main.c

[illegible]

```

//
// Для работы с портом в заголовочном файле объявлена следующая структура
// typedef struct
// {
//     __IO uint32_t MODER;    /* Регистр режимов работы порта (вход/выход) */
//     __IO uint32_t OTYPER;   /* Регистр задания типа выходного каскада */
//     __IO uint32_t OSPEEDR;  /* Регистр задания максимальной скорости вывода */
//     __IO uint32_t PUPDR;    /* Регистр конфигурирования подтягивающего резистора */
//     __IO uint32_t IDR;      /* Регистр входных данных */
//     __IO uint32_t ODR;      /* Регистр выходных данных */
//     __IO uint16_t BSRRL;    /* Регистр установки/сброса выходных бит, младшая часть (установка) */
//     __IO uint16_t BSRRH;    /* Регистр установки/сброса выходных бит, старшая часть (сброс) */
//     __IO uint32_t LCKR;     /* Регистр блокировки изменения конфигурации (до сброса процессора) */
//     __IO uint32_t AFR[2];   /* Регистры выбора альтернативной функции (по 4 бита на разряд) */
// } GPIO_TypeDef;
//
//
// Ввод из порта (чтение входных данных) осуществляется 16-разрядным словом. Вывод в порт
// (запись выходных данных) через регистр ODR производится одновременно во все 16-разрядов,
// а через регистры BSRRL, BSRRH возможна выборочная установка или сброс отдельных разрядов.
//
//-----

// Заголовочные файлы
#include "stm32_p407.h"           //Файл конфигурации отладочной платы STM32-P407

// Библиотечные функции временных задержек
// void Delay_ms(int ms);         //Входной аргумент: значение задержки в миллисекундах
// void Delay_us(int us);        //Входной аргумент: значение задержки в микросекундах

//Прототипы функций, определенных в этом файле
void PB1_Init(void);

//-----
// ГЛАВНАЯ ФУНКЦИЯ
int main()
{
    //Текущие переменные
    uint16_t joy, indic;          //Код от джойстика, код для вывода на индикаторы
    int i = 0;                    //Счетчик

    // Разрешение тактирования используемых портов (A, C, F, G),
    // производится установкой бит в системном регистре ANHB1ENR -
    // 0000 0000 0000 0000 0000 0000 0110 0101
    //                ||  |  |
    //                KJINGFE DCBA - обозначения портов
    RCC->AHB1ENR |= 0x65;

    // Конфигурирование разрядов портов A, C, G на ввод, разрядов 6, 7, 8, 9 порта F - на вывод.
    // В регистрах MODER каждые два бита определяют состояние соответствующего разряда порта:
    // 00 - ввод (по умолчанию), 01 - вывод, 10 - альтернативная функция, 11 - аналоговый вход
    //GPIOA->MODER = 0;    // \ Эти операторы можно опустить (значения заданы по умолчанию после сброса),
    // при отладке с внутрисхемным эмулятором, который использует PA13, PA14,
    // такой способ конфигурирования для порта A является неправильным.
    GPIOG->MODER = 0;
    GPIOF->MODER = 1 << 6*2 | 1 << 7*2 | 1 << 8*2 | 1 << 9*2;

    // Конфигурирование режимов работы разрядов портов:
    // в регистре OSPEEDR: быстроедействие на вывод - минимальное,
    // в регистре OTYPER: тип выходного каскада - двухтактный,
    // в регистре PUPDR: подтягивающий резистор - отсутствует (на плате установлены внешние).
    // Эти режимы заданы по умолчанию, поэтому никаких действий не предпринимаем.

    //Конфигурирование разряда PB1 на вывод посредством библиотечных функций
    PB1_Init();

    //ГЛАВНЫЙ ЦИКЛ ПРОГРАММЫ
    while (1)
    {
        //Ввод 16-разрядного кода из порта G
        joy = GPIOG->IDR;

        //Перепаковка 4-х бит (смотри также схему выше):
        indic = (joy >> 5 & 0x40) |    // - сдвиг на 5 бит вправо и выделение 6-го разряда
                (joy & 0x180) |         // - без сдвига, выделение 7-го и 8-го разрядов
                (joy << 3 & 0x200);    // - сдвиг на 3 бита влево и выделение 9-го разряда
    }
}

```

```

// ~ ~ ~ ~ ~
//Проверка 13-го разряда в порту C (т.е. кнопки TAMPER)
// при отсутствии нажатия перепакованный код с кнопок постоянно передается в порт с индикаторами,
// при нажатии - формируется последовательность: вывод кода, временная задержка, снятие кода,
// временная задержка.
if (GPIOC->IDR & 0x2000)
    GPIOF->ODR = indic; //Если не нажата, вывод 16-разрядного кода в порт F (плохой пример!)
else //Здесь кнопка TAMPER находится в нажатом состоянии:
{ GPIOF->BSRR = indic; // - вывод 4-разрядного кода в порт F (только разряды с лог. '1')
  Delay_ms(30); // - временная задержка
  GPIOF->BSRRH = 0x3C0; // - вывод логических '1' в разряды 9..6 регистра сброса бит,
  // что соответствует обнулению индикаторных разрядов в порту F
  Delay_ms(30); // - временная задержка
}

// ~ ~ ~ ~ ~
//Цикл, постоянно выполняемый при определенном сочетании бит в портах C и G:
// когда бит PC13 равен 0 и бит PG15 не равен 0 (т.е. нажаты TAMPER и SEL).
// При каждом проходе осуществляется формирование полного периода сигнала на звуковой излучатель -
// выдача лог."1", временная задержка на пол-периода, выдача лог."0", временная задержка на пол-периода.
while (!(GPIOC->IDR & 0x2000) && GPIOG->IDR & 0x8000)
{ GPIOB->BSRR = 0x02; // - зап. в 1-й разряд регистра установки выходов порта PB (уст. PB1)
  Delay_ms(1); // - временная задержка
  GPIOB->BSRRH = 0x02; // - запись в 1-й разряд регистра сброса выходов порта PB (сброс PB1)
  Delay_ms(1); // - временная задержка
}

// ~ ~ ~ ~ ~
//Проверка 15-го разряда кода с джойстика (кнопки SEL)
// При нажатой кнопке при каждом прохождении фрагмента (т.е. циклически) выполняется последовательность:
// включение каких-то двух индикаторов, временная задержка (задает длительность включенного состояния),
// гашение индикаторов, временная задержка (задает длительность выключенного состояния).
// Счетчик на переменной i увеличивается с каждым проходом. Оператор i & 0x8 проверяет состояние
// 3-го бита счетчика, этот бит изменяется через каждые 2 в степени 3 циклов (т.е. 8 циклов),
// соответственно через 8 циклов происходит переключение пар индикаторов.
// Оператор i & 0x80 проверяет состояние 7-го бита счетчика, бит изменяется через каждые 2 в степени 7
// циклов (т.е. 128 циклов), соответственно через 128 циклов изменяется временная задержка, задающая
// время включенного состояния индикатора. Отношение длительности импульса к периоду определяет
// среднюю мощность управления индикаторами (яркость).
if (joy & 0x8000) //Если кнопка нажата, выполняются операции:
{ GPIOF->BSRR = i & 0x8 ? 0x300 : 0xC0; // - в зависимости от состояния счетчика: установка в порту PF
  // разрядов 9,8 (код 0x300) или 7,6 (код 0xC0)
  Delay_us(i & 0x80 ? 1000 : 50); // - в зависимости от состояния счетчика: задание одной из двух
  // задержек
  GPIOF->BSRRH = 0x3C0; // - запись '1' в разр. 9..6 регистра сброса бит (гашение индикации)
  Delay_ms(10); // - фиксированная временная задержка
  i++; // - инкремент счетчика (с периодом, примерно равным сумме задержек)
}

//Проверка нажатия кнопки WAKEUP, если нажата, сброс процессора (завершение программы)
if (GPIOA->IDR & 1) NVIC_SystemReset();
}

}

//-----
//
// ПОДПРОГРАММА КОНФИГУРИРОВАНИЯ РАЗРЯДА ПОРТА PB1
void PB1_Init(void)
{
    GPIO_InitTypeDef GPIO_InitStructure; //Структура конфигурации портов общего назначения

    RCC_AHB1PeriphClockCmd(RCC_AHB1Periph_GPIOB, ENABLE); //Разрешение тактирования порта

    GPIO_InitStructure.GPIO_Pin = GPIO_Pin_1; //Маска на выходной разряд
    GPIO_InitStructure.GPIO_Mode = GPIO_Mode_OUT; //Режим: вывод
    GPIO_InitStructure.GPIO_Speed = GPIO_Speed_2MHz; //Задание быстродействия (2MHz, 25MHz, 50MHz, 100MHz)
    GPIO_InitStructure.GPIO_OType = GPIO_OType_PP; //Установка типа выходного каскада: двухтактный
    GPIO_InitStructure.GPIO_PuPd = GPIO_PuPd_NOPULL; //Подтягивающий резистор: нет
    GPIO_Init(GPIOB, &GPIO_InitStructure); //Функция конфигурирования
}

//-----

```


Проект ..\Lab2_tim_pwm\MDK-ARM\timer.uvproj

2. Использование таймеров для генерации временных интервалов и сигналов с широтно-импульсной модуляцией

Исходный модуль ..\Lab2_tim_pwm\main.c

```
//-----  
//  
//  УЧЕБНЫЙ ПРОЕКТ  
//  Использование таймеров для генерации временных интервалов и ШИМ-сигналов  
//  Copyright (C) 2016 МИРЭА  
//  
//-----  
  
#include "stm32_p407.h"           //Файл конфигурации отладочной платы STM32-P407  
  
//Прототипы вызываемых функций  
void TimerConfig(void);  
  
//-----  
//  ГЛАВНАЯ ФУНКЦИЯ  
int main()  
{  
    //Инициализация органов управления  
    STM_PBInit(BUTTON_TAMPER, BUTTON_MODE_GPIO); //Кнопка TAMPER  
    STM_PBInit(BUTTON_WAKEUP, BUTTON_MODE_GPIO); //Кнопка WAKEUP  
    STM_PBInit(BUTTON_RIGHT, BUTTON_MODE_GPIO); //Позиции джойстика  
    STM_PBInit(BUTTON_LEFT, BUTTON_MODE_GPIO);  
    STM_PBInit(BUTTON_UP, BUTTON_MODE_GPIO);  
    STM_PBInit(BUTTON_DOWN, BUTTON_MODE_GPIO);  
    STM_PBInit(BUTTON_SEL, BUTTON_MODE_GPIO);  
  
    //Инициализация индикаторов LED3, LED4  
    // (LED1, LED2 управляются напрямую от таймеров)  
    STM_LEDInit(LED3); STM_LEDOff(LED3);  
    STM_LEDInit(LED4); STM_LEDOff(LED4);  
  
    //Конфигурирование таймеров (см. файл tim_pwm.c)  
    TimerConfig();  
  
    //Цикл в основной программе  
    while (1)  
    {  
        //Проверка воздействия на кнопку WAKEUP, при нажатии - выход из цикла  
        if (STM_PBGetState(BUTTON_WAKEUP)) break;  
    }  
  
    //Сброс процессора - завершение выполнения данной программы, запуск начального загрузчика  
    NVIC_SystemReset();  
}  
  
//-----  
//  ОБРАБОТЧИК ПРЕРЫВАНИЯ ОТ ТАЙМЕРА 4  
void TIM4_IRQHandler(void)  
{  
    //Проверка установленного флага (источника прерывания)  
    if (TIM_GetITStatus(TIM4, TIM_IT_Update) != RESET)  
    {  
        TIM_ClearITPendingBit(TIM4, TIM_IT_Update); //Очистка флага  
        STM_LEDToggle(LED3);                          //Управление индикатором LED3  
    }  
}  
//-----
```

Исходный модуль ..\Lab2_tim_pwm\tim_pwm.c

```
//-----  
//  
//  УЧЕБНЫЙ ПРОЕКТ
```

```

// Работа с таймерами. Режим генерации ШИМ-сигнала
// Copyright (C) 2016 МИРЭА
//
//-----

#include "stm32f4xx.h"          //Константы и структуры данных для процессоров семейства STM32F4xx

//-----
// ИНИЦИАЛИЗАЦИЯ ТАЙМЕРОВ
/* Таймеры 10,11 подключены к шине APB2, тактируются частотой 168 МГц.
Предделение: 168 000 кГц / 16800 = 10 кГц.
Основное деление для TIM10: 10000 Гц / 10000 = 1 Гц   (TIM_Period = 10000-1).
Основное деление для TIM11: 10000 Гц / 20000 = 0.5 Гц (TIM_Period = 20000-1).
Выход TIM10_CH1 подключен к индикатору LED1 (разряд порта PF6).
Выход TIM11_CH1 подключен к индикатору LED2 (разряд порта PF7).
ШИМ TIM10 имеет коэффициент заполнения длительности импульса
(duty cycle) = 10%, загружаемое число для этого
TIM_CCR = TIM_Period * duty cycle[%] / 100[%] = 10000 * 10 / 100 = 1000.
Для ШИМ TIM11 (duty cycle) = 75%, TIM_CCR = 20000 * 75 / 100 = 15000.
Прерывания от таймеров 10, 11 не используются.
*/

void TimerConfig(void)
{
    GPIO_InitTypeDef GPIO_InitStructure;          //Структура конфигурации портов общего назначения
    TIM_TimeBaseInitTypeDef TIM_TimeBaseStructure; //Структура конфигурации базового таймера
    TIM_OCInitTypeDef TIM_OCInitStructure;        //Структура конфигурации таймера с ШИМ
    NVIC_InitTypeDef NVIC_InitStructure;          //Структура конфигурации прерываний

    //Разрешение тактирования TIM10, TIM11
    RCC_APB2PeriphClockCmd(RCC_APB2Periph_TIM10 | RCC_APB2Periph_TIM11, ENABLE);

    //Разрешение тактирования порта PF
    RCC_AHB1PeriphClockCmd(RCC_AHB1Periph_GPIOF, ENABLE);

    //Конфигурирование разрядов порта PF6, PF7 как выходных с альтернативной функцией
    GPIO_InitStructure.GPIO_Pin = GPIO_Pin_6 | GPIO_Pin_7; //Номер разряда (маска)
    GPIO_InitStructure.GPIO_Mode = GPIO_Mode_AF;           //Альтернативная функция
    GPIO_InitStructure.GPIO_Speed = GPIO_Speed_100MHz;     //Задание быстродействия
    GPIO_InitStructure.GPIO_OType = GPIO_OType_PP;         //Установка типа выходного каскада
    GPIO_InitStructure.GPIO_PuPd = GPIO_PuPd_UP;           //Подтягивающий резистор: к "+" питания
    GPIO_Init(GPIOF, &GPIO_InitStructure);               //Функция конфигурирования

    //Подключение выхода канала CH1 таймера TIM10 к разряду порта PF6 (как альтернативная функция AF2)
    GPIO_PinAFConfig(GPIOF, GPIO_PinSource6, GPIO_AF_TIM10);

    //Подключение выхода канала CH1 таймера TIM11 к разряду порта PF7 (как альтернативная функция AF2)
    GPIO_PinAFConfig(GPIOF, GPIO_PinSource7, GPIO_AF_TIM11);

    //Базовая конфигурация таймера
    TIM_TimeBaseStructure.TIM_Period = 10000-1;             //Основной коэф.деления для TIM10
    TIM_TimeBaseStructure.TIM_Prescaler = 16800-1;          //Предделение
    TIM_TimeBaseStructure.TIM_ClockDivision = 0;            //Делитель для входного фильтра
    TIM_TimeBaseStructure.TIM_CounterMode = TIM_CounterMode_Up; //Счетчик вверх: от 0 до TIM_Period
    TIM_TimeBaseInit(TIM10, &TIM_TimeBaseStructure);       //Функция конфигурирования

    TIM_TimeBaseStructure.TIM_Period = 20000-1;             //Основной коэф.деления для TIM11
    TIM_TimeBaseInit(TIM11, &TIM_TimeBaseStructure);       //Функция конфигурирования

    //Конфигурирование ШИМ
    TIM_OCInitStructure.TIM_OCMode = TIM_OCMode_PWM1;       //Режим выхода: ШИМ
    TIM_OCInitStructure.TIM_OutputState = TIM_OutputState_Enable; //Разрешение аппаратного выхода
    TIM_OCInitStructure.TIM_Pulse = 1000;                  //Загружаемое значение
    TIM_OCInitStructure.TIM_OCPolarity = TIM_OCPolarity_High; //Полярность импульса
    TIM_OC1Init(TIM10, &TIM_OCInitStructure);              //Функция конфигурирования

    TIM_OCInitStructure.TIM_Pulse = 15000;
    TIM_OC1Init(TIM11, &TIM_OCInitStructure);

    //Общее разрешение работы таймеров
    TIM_Cmd(TIM10, ENABLE);
    TIM_Cmd(TIM11, ENABLE);

    // КОНФИГУРИРОВАНИЕ ТАЙМЕРА 4
    /* Подключен к шине APB1, тактируется частотой 84 МГц.

```

```

    Предделение: 84000 кГц / 42000 = 2 кГц.
    Основное деление: 2000 Гц / 200 = 10 Гц (период 100 мс).
    Аппаратный выход не задействован.
    Таймер используется для генерации прерываний при автоперезагрузке.
*/
//Разрешение тактирования TIM4
RCC_APB1PeriphClockCmd(RCC_APB1Periph_TIM4, ENABLE);

//Базовая конфигурация таймера
TIM_TimeBaseStructure.TIM_Period = 200-1;           //Основной коэф.деления
TIM_TimeBaseStructure.TIM_Prescaler = 42000-1;      //Предделение
TIM_TimeBaseStructure.TIM_ClockDivision = 0;
TIM_TimeBaseStructure.TIM_CounterMode = TIM_CounterMode_Up; //Счетчик вверх: от 0 до TIM_Period
TIM_TimeBaseInit(TIM4, &TIM_TimeBaseStructure);     //Функция конфигурирования

//Разрешение прерываний при перезагрузке таймера
TIM_ITConfig(TIM4, TIM_IT_Update, ENABLE);

//Общее разрешение работы таймера TIM4
TIM_Cmd(TIM4, ENABLE);

//Конфигурирование прерывания от таймера
NVIC_InitStructure.NVIC_IRQChannel = TIM4_IRQn;     //Номер (линия) прерывания
NVIC_InitStructure.NVIC_IRQChannelPreemptionPriority = 0; //Приоритет группы
NVIC_InitStructure.NVIC_IRQChannelSubPriority = 1;   //Приоритет внутри группы
NVIC_InitStructure.NVIC_IRQChannelCmd = ENABLE;     //Разрешение прерывания
NVIC_Init(&NVIC_InitStructure);                    //Функция конфигурирования
}

//-----

```

Проект ..\Lab2_tim_ton\MDK-ARM\timer.uvproj

3. Использование таймеров для генерации тональных сигналов

Исходный модуль ..\Lab2_tim_ton\main.c

```

//-----
//
// УЧЕБНЫЙ ПРОЕКТ
// Использование таймеров для генерации тональных сигналов
// Copyright (C) 2016 МИРЭА
//
//-----

#include "stm32_p407.h"           //Файл конфигурации отладочной платы STM32-P407
#include <stdlib.h>               //Заголовочный файл с определением функции rand()
/*
    Справка: Частоты музыкальных тонов

    Индекс  Нота  Частота  |  Индекс  Нота  Частота  |  Индекс  Нота  Частота  |  Индекс  Нота  Частота
           Гц      |      Гц      |      Гц      |      Гц
    -----|-----|-----|-----
    0      до   261.63  |  12      до   523.25  |  24      до   1046.50  |  36      до   2093.01
    1      ре   277.18  |  13      ре   554.37  |  25      ре   1108.73  |  37      ре   2217.46
    2      ми   293.66  |  14      ми   587.33  |  26      ми   1174.66  |  38      ми   2349.32
    3      фа   311.13  |  15      фа   622.25  |  27      фа   1244.51  |  39      фа   2489.02
    4      соль 329.63  |  16      соль 659.26  |  28      соль 1318.51  |  40      соль 2637.02
    5      ля   349.23  |  17      ля   698.46  |  29      ля   1396.91  |  41      ля   2793.83
    6      си   369.99  |  18      си   739.99  |  30      си   1479.98  |  42      си   2960.00
    7      до   392.00  |  19      до   783.99  |  31      до   1567.98  |  43      до   3135.96
    8      ре   415.30  |  20      ре   830.61  |  32      ре   1661.22  |  44      ре   3322.44
    9      ми   440     |  21      ми   880.00  |  33      ми   1760.00  |  45      ми   3520
    10     фа   466.16  |  22      фа   932.33  |  34      фа   1864.66  |  46      фа   3729.31
    11     соль 493.88  |  23      соль 987.77  |  35      соль 1975.54  |  47      соль 3951.07
                                |                |  36      соль 2093.01  |  48      соль 4186.02
                                |                |  37      соль 2217.46  |
                                |                |  38      соль 2349.32  |
                                |                |  39      соль 2489.02  |
                                |                |  40      соль 2637.02  |
                                |                |  41      соль 2793.83  |
                                |                |  42      соль 2960.00  |
                                |                |  43      соль 3135.96  |
                                |                |  44      соль 3322.44  |
                                |                |  45      соль 3520
                                |                |  46      соль 3729.31
                                |                |  47      соль 3951.07
                                |                |  48      соль 4186.02

*/

//Таблица частот музыкальных тонов (одна строка составляет октаву)
float ToneTable[] =
{ 261.63, 277.18, 293.66, 311.13, 329.63, 349.23, 369.99, 392.00, 415.30, 440, 466.16, 493.88,
  523.25, 554.37, 587.33, 622.25, 659.26, 698.46, 739.99, 783.99, 830.61, 880, 932.33, 987.77,

```

```
1046.50, 1108.73, 1174.66, 1244.51, 1318.51, 1396.91, 1479.98, 1567.98, 1661.22, 1760, 1864.66, 1975.54,
2093.01, 2217.46, 2349.32, 2489.02, 2637.02, 2793.83, 2960.00, 3135.96, 3322.44, 3520, 3729.31, 3951.07,
4186.02 };
```

```
//Прототипы вызываемых функций
```

```
void TimerConfig(void);
```

```
void ToneGeneration(Button_TypeDef Button, uint32_t ToneIndex);
```

```
//-----
```

```
// ГЛАВНАЯ ФУНКЦИЯ
```

```
int main()
```

```
{
```

```
    uint32_t Octave;
```

```
    //Инициализация органов управления
```

```
    STM_PBInit(BUTTON_TAMPER, BUTTON_MODE_GPIO); //Кнопка TAMPER
```

```
    STM_PBInit(BUTTON_WAKEUP, BUTTON_MODE_GPIO); //Кнопка WAKEUP
```

```
    STM_PBInit(BUTTON_RIGHT, BUTTON_MODE_GPIO); //Позиции джойстика
```

```
    STM_PBInit(BUTTON_LEFT, BUTTON_MODE_GPIO);
```

```
    STM_PBInit(BUTTON_UP, BUTTON_MODE_GPIO);
```

```
    STM_PBInit(BUTTON_DOWN, BUTTON_MODE_GPIO);
```

```
    STM_PBInit(BUTTON_SEL, BUTTON_MODE_GPIO);
```

```
    //Инициализация индикаторов
```

```
    STM_LEDInit(LED4); STM_LEDOff(LED4);
```

```
    //Конфигурирование таймера (см. файл tim_ton.c)
```

```
    TimerConfig();
```

```
    //Цикл в основной программе
```

```
    while (1)
```

```
    {
```

```
        //Проверка состояния кнопки TAMPER (октавный сдвиг)
```

```
        Octave = STM_PBGetState(BUTTON_TAMPER) ? 0 : 12;
```

```
        //Проверка положения джойстика "Влево"
```

```
        if (STM_PBGetState(BUTTON_LEFT)) ToneGeneration(BUTTON_LEFT, Octave + 0);
```

```
        //Проверка положения джойстика "Вверх"
```

```
        if (STM_PBGetState(BUTTON_UP)) ToneGeneration(BUTTON_UP, Octave + 3);
```

```
        //Проверка положения джойстика "Вправо"
```

```
        if (STM_PBGetState(BUTTON_RIGHT)) ToneGeneration(BUTTON_RIGHT, Octave + 7);
```

```
        //Проверка положения джойстика "Выбор"
```

```
        if (STM_PBGetState(BUTTON_SEL)) ToneGeneration(BUTTON_SEL, Octave + 24);
```

```
        //Проверка положения джойстика "Вниз"
```

```
        if (STM_PBGetState(BUTTON_DOWN)) ToneGeneration(BUTTON_DOWN, rand() % 49);
```

```
        rand();
```

```
        //Проверка воздействия на кнопку WAKEUP, при нажатии - выход из цикла
```

```
        if (STM_PBGetState(BUTTON_WAKEUP)) break;
```

```
    }
```

```
    //Сброс процессора - завершение выполнения данной программы, запуск начального загрузчика
```

```
    NVIC_SystemReset();
```

```
}
```

```
//-----
```

```
// ПОДПРОГРАММА ВЫДАЧИ ТОНАЛЬНОГО СИГНАЛА
```

```
// Входные параметры: Button - номер (индекс) нажатой кнопки
```

```
// ToneIndex - индекс генерируемого тона по таблице ToneTable[]
```

```
void ToneGeneration(Button_TypeDef Button, uint32_t ToneIndex)
```

```
{
```

```
    STM_LEDOn(LED4);
```

```
    //Контрольная индикация
```

```
    //Расчет и загрузка в таймер коэффициента деления
```

```
    TIM_SetAutoreload(TIM3, 1e6f / ToneTable[ToneIndex] - 1);
```

```
    TIM_Cmd(TIM3, ENABLE);
```

```
    //Разрешение работы таймера
```

```
    if (Button == BUTTON_DOWN) Delay_ms(300);
```

```
    //Для одной из кнопок - временная задержка,
```

```
    else while (STM_PBGetState(Button));
```

```
    // для других - ожидание отпускания кнопки
```

```
    TIM_Cmd(TIM3, DISABLE);
```

```
    //Запрет работы таймера
```

```
    STM_LEDOff(LED4);
```

```
}
```

```
//-----
```

Исходный модуль ..\Lab2_tim_ton\tim_ton.c

```
//-----  
//  
//  УЧЕБНЫЙ ПРОЕКТ  
//  Работа с таймерами. Режим деления частоты  
//  Copyright (C) 2016 МИРЭА  
//  
//-----  
  
#include "stm32f4xx.h"           //Константы и структуры данных для процессоров семейства STM32F4xx  
  
//-----  
// ИНИЦИАЛИЗАЦИЯ ТАЙМЕРА  
/* Таймер 3 подключен к шине APB1, тактируется частотой 84 МГц.  
   Предделитель с коэффициентом деления 42 снижает частоту до 2 МГц.  
   Основной коэффициент деления определяется частотой музыкального тона,  
   при этом следует учесть, выходная частота получается еще в 2 раза меньше,  
   так как при каждом переполнении счетного регистра формируется полпериода сигнала.  
   Выход таймера - канал CH4 - выведен на разряд порта PB1.  
*/  
void TimerConfig(void)  
{  
  
    GPIO_InitTypeDef GPIO_InitStructure;           //Структура конфигурации портов общего назначения  
    TIM_TimeBaseInitTypeDef TIM_TimeBaseStructure; //Структура конфигурации базового таймера  
    TIM_OCInitTypeDef TIM_OCInitStructure;         //Структура конфигурации таймера с ШИМ  
  
    //Разрешение тактирования таймера  
    RCC_APB1PeriphClockCmd(RCC_APB1Periph_TIM3, ENABLE);  
  
    //Разрешение тактирования порта PB  
    RCC_AHB1PeriphClockCmd(RCC_AHB1Periph_GPIOB, ENABLE);  
  
    //Конфигурирование разряда порта PB1 как выходного с альтернативной функцией  
    GPIO_InitStructure.GPIO_Pin = GPIO_Pin_1;      //Номер разряда (маска)  
    GPIO_InitStructure.GPIO_Mode = GPIO_Mode_AF;   //Альтернативная функция  
    GPIO_InitStructure.GPIO_Speed = GPIO_Speed_100MHz; //Задание быстродействия  
    GPIO_InitStructure.GPIO_OType = GPIO_OType_PP;  //Установка типа выходного каскада: двухтактный  
    GPIO_InitStructure.GPIO_PuPd = GPIO_PuPd_UP;    //Подтягивающий резистор: к "+" питания  
    GPIO_Init(GPIOB, &GPIO_InitStructure);         //Функция конфигурирования  
  
    //Подключение выхода канала CH4 таймера TIM3 к разряду порта PB1  
    GPIO_PinAFConfig(GPIOB, GPIO_PinSource1, GPIO_AF_TIM3);  
  
    //Базовая конфигурация таймера  
    TIM_TimeBaseStructure.TIM_Period = 1000-1;      //Основной коэф. деления по умолчанию  
    TIM_TimeBaseStructure.TIM_Prescaler = 42-1;     //Предделение до 2 МГц  
    TIM_TimeBaseStructure.TIM_ClockDivision = 0;  
    TIM_TimeBaseStructure.TIM_CounterMode = TIM_CounterMode_Up; //Счетчик вверх: от 0 до TIM_Period  
    TIM_TimeBaseInit(TIM3, &TIM_TimeBaseStructure); //Функция конфигурирования  
  
    //Конфигурирование ШИМ  
    TIM_OCInitStructure.TIM_OCMode = TIM_OCMode_Toggle; //Режим выхода: переключение уровня  
    TIM_OCInitStructure.TIM_OutputState = TIM_OutputState_Enable; //Разрешение аппаратного выхода  
    TIM_OCInitStructure.TIM_Pulse = 0;               //Пороговое значение для переключения уровня  
    TIM_OCInitStructure.TIM_OCPolarity = TIM_OCPolarity_High; //Полярность импульса - положительная  
    TIM_OC4Init(TIM3, &TIM_OCInitStructure);         //Функция конфигурирования  
  
    //Работу таймера пока не разрешаем  
    TIM_Cmd(TIM3, DISABLE);  
}  
  
//-----
```

Проект ..\Lab2_tim_sys\MDK-ARM\timer.uvproj

4. Использование таймеров для измерения временных интервалов

Исходный модуль ..\Lab2_tim_sys\main.c

```

//-----
//
//  УЧЕБНЫЙ ПРОЕКТ
//  Использование таймеров для измерений
//  Copyright (C) 2016 МИРЭА
//
//-----

#include "math.h"                //Математические функции стандартной библиотеки C/C++
#include "stm32_p407.h"          //Файл конфигурации отладочной платы STM32-P407
#include "lcd.h"                 //Функции работы с дисплеем

uint32_t Screen = 0;            //Номер экрана (0 - исходный, 1 - метки времени)
uint32_t SysTickPeriod = 1000000; //Период счета системного таймера, тактов

//Прототипы функций, реализованных в этом же файле, но после операторов их вызова
void BaseScreen(void);
void TimeMarkerOut(void);
void TimeMarker1(void);
void TimeMarker2(void);
void TimeMarker3(void);
void TimeMarker4(void);
void TimeMarker5(void);
void TimeMarker6(void);

//-----
// ГЛАВНАЯ ФУНКЦИЯ
int main()
{
    volatile int i;                //Счетчик циклов
    float farg = 2;                //Аргументы функции
    double darg = 1;

    //Инициализация кнопок
    STM_PBInit(BUTTON_TAMPER, BUTTON_MODE_GPIO);
    STM_PBInit(BUTTON_WAKEUP, BUTTON_MODE_GPIO);

    //Инициализация дисплея, вывод начального сообщения
    LCD_Init();
    BaseScreen();

    //Инициализация измерительного таймера
    SysTick->LOAD = SysTickPeriod - 1;
    SysTick->VAL = 0;
    SysTick->CTRL = SysTick_CTRL_CLKSOURCE_Msk | SysTick_CTRL_ENABLE_Msk;

    //Начало бесконечного цикла в программе
    while (1)
    {
        //ФИКСАЦИЯ ВРЕМЕН ВЫЧИСЛЕНИЙ ФРАГМЕНТОВ ПРОГРАММЫ
        TimeMarker1();                //Метка времени 1
        for (i = 200; i; i--) sin(darg);

        TimeMarker2();                //Метка времени 2
        for (i = 200; i; i--) sinf(farg);

        TimeMarker3();                //Метка времени 3
        for (i = 200; i; i--) sinh(farg);

        TimeMarker4();                //Метка времени 4
        for (i = 200; i; i--) fabsf(farg);

        TimeMarker5();                //Метка времени 5
        for (i = 200; i; i--) powf(farg, farg);

        TimeMarker6();                //Метка времени 6

        /*
        //Варианты с другими функциями
        TimeMarker1();
        for (i = 100; i; i--) tanf(farg);

        TimeMarker2();
        for (i = 100; i; i--) hypotf(farg, farg);

        TimeMarker3();
        for (i = 100; i; i--) asinf(farg);
        */
    }
}

```

```

    TimeMarker4();
    for (i = 100; i; i--) log10f(farg);
    //for (i = 100; i; i--) expf(farg);

    TimeMarker5();
    for (i = 100; i; i--) sqrtf(farg);
    //for (i = 100; i; i--) sqrt(farg);

    TimeMarker6();
*/

//Проверка нажатия кнопки TAMPER
if (!STM_PBGetState(BUTTON_TAMPER))
{
    if (Screen) BaseScreen();           //Переключение между исходным экраном
    else TimeMarkerOut();               // и экраном вывода меток времени
    while (!STM_PBGetState(BUTTON_TAMPER)); //Ожидание отпускания кнопки
}

//Проверка нажатия кнопки WAKEUP
if (STM_PBGetState(BUTTON_WAKEUP)) NVIC_SystemReset();
}
}

//-----
// ФОРМИРОВАНИЕ ИСХОДНОГО СООБЩЕНИЯ
void BaseScreen(void)
{
    Screen = 0;
    LCD_Clear(1);
    LCD_FontSize(11);
    LCD_FontColor(0xF0F);
    LCD_print(" Вывод меток \r\n"
              " времени \r\n"
              " выполнения: \r\n"
              " кнопка \r\n"
              " \"TAMPER\" \r\n\r\n"
              "Выход - \"WAKEUP\"");
}

//-----

uint32_t TimeCur1, TimeCur2, TimeCur3, //Текущие значения меток времени
          TimeCur4, TimeCur5, TimeCur6;
uint32_t TimeFix1, TimeFix2, TimeFix3, //Зафиксированные значения меток времени
          TimeFix4, TimeFix5, TimeFix6;

// ВЫВОД ВРЕМЕННЫХ МЕТОК
void TimeMarkerOut(void)
{
    Screen = 1;
    LCD_Clear(1);
    LCD_FontSize(11);
    LCD_FontColor(0xFFF);
    LCD_TextPos(0,0);
    LCD_print("N Такты\r\n");
    LCD_FontColor(0x4F4);
    LCD_print("1: %07d \r\n", TimeFix1);
    LCD_print("2: %07d \r\n", TimeFix2);
    LCD_print("3: %07d \r\n", TimeFix3);
    LCD_print("4: %07d \r\n", TimeFix4);
    LCD_print("5: %07d \r\n", TimeFix5);
    LCD_print("6: %07d ", TimeFix6);
}

//-----
// ФИКСАЦИЯ МЕТОК ВРЕМЕНИ ВЫПОЛНЕНИЯ
// Для последнего измерения - сохранение всех значений для вывода
void TimeMarker1(void)
{
    TimeCur1 = SysTick->VAL;
}
void TimeMarker2(void)
{
    TimeCur2 = SysTick->VAL;
}

```

```

void TimeMarker3(void)
{
    TimeCur3 = SysTick->VAL;
}
void TimeMarker4(void)
{
    TimeCur4 = SysTick->VAL;
}
void TimeMarker5(void)
{
    TimeCur5 = SysTick->VAL;
}
void TimeMarker6(void)
{
    TimeCur6 = SysTick->VAL;
    TimeFix1 = TimeCur1, TimeFix2 = TimeCur2, TimeFix3 = TimeCur3,
    TimeFix4 = TimeCur4, TimeFix5 = TimeCur5, TimeFix6 = TimeCur6;
}

//-----

```

Заголовочный модуль ..\Libraries\Board\lcd.h

```

/*****
 * Файл      lcd.h
 * Автор     МИРЭА
 * Версия    V1.1
 * Дата      01.09.2015
 * Описание  Заголовочный файл с функциями верхнего уровня для работы
 *           с LCD дисплеем на плате STM32-P407
 *****/

/* Инициализация дисплея */
void LCD_Init(void);

/* Отображение стандартной заставки (логотипа) */
void LCD_Logo(void);

/* Очистка дисплея */
void LCD_Clear(int back);

/* Установка размера шрифта */
void LCD_FontSize(unsigned int Size);

/* Установка цвета символов */
void LCD_FontColor(unsigned int Color);

/* Установка цвета фона при выводе символов */
void LCD_BackColor(unsigned int Color);

/* Установка окна для вывода */
void LCD_SetWindow(int Left, int Top, int Right, int Bottom);

/* Установка позиции для последующего вывода символов */
void LCD_TextPos(unsigned int X, unsigned int Y);

/* Форматированный вывод строки */
void LCD_print(const char * string, ...);

/* Включение / гашение дисплея */
void LCD_OnOff(unsigned int On);

/* Плавное включение / гашение дисплея */
void LCD_OnOffSmooth(unsigned int On);

/* Задание цвета линий */
void LCD_PenColor(unsigned int Color);

/* Построение закрашенного прямоугольника */
void LCD_Rectangle(unsigned int x1, unsigned int x2, unsigned int y1, unsigned int y2);

/* Построение растрового изображения */
void LCD_Picture(unsigned short* pData);

```


Тексты к лабораторной работе № 3

Общие модули для работы с АЦП, ЦАП, кодеком.

Используются во всех проектах данной лабораторной работы.

Исходный модуль adc.c

```
/**
*****
* Файл      adc.c
* Copyright (C) 2016 МИРЭА
* Версия    2.1
* Описание:  Функции инициализации и обслуживания АЦП
*
* Формат цифровых данных от АЦП - 12-разрядный смещенный код:
* код 1111 1111 1111 соответствует Uвх = максимум (практически Vref);
* код 1000 0000 0000 соответствует Uвх = Vref / 2;
* код 0000 0000 0000 соответствует Uвх = минимум (практически 0);
* где Vref - опорное напряжение (для отладочной платы равно напряжению питания)
*
* Сигналы запуска АЦП: выходы таймеров TIM1, TIM2, TIM3, TIM4, TIM5, TIM8;
* внешний сигнал.
*
* Распределение аналоговых входов по разрядам портов и источникам:
* Порт      ADC      Channel
* PA0       1,2,3     0
* PA1       1,2,3     1
* PA2       1,2,3     2
* PA3       1,2,3     3
* PA4       1,2       4
* PA5       1,2       5
* PA6       1,2       6
* PA7       1,2       7
* PB0       1,2       8
* PB1       1,2       9
* PC0       1,2,3     10
* PC1       1,2,3     11
* PC2       1,2,3     12
* PC3       1,2,3     13
* PC4       1,2       14
* PC5       1,2       15
* PF3       3         9
* PF4       3         14
* PF5       3         15
* PF6       3         4
* PF7       3         5
* PF8       3         6
* PF9       3         7
* PF10      3         8
* -         1         16 (внутренний температурный датчик)
* -         1         17 (опорное напряжение Vref внутреннего источника)
* -         1         18 (напряжение батареи Vbat)
*
* Примечания.
* 1. Значение температуры с температурного датчика вычисляется по формуле:
*    Температура в °C = {(VSENSE - V25) / Avg_Slope} + 25,
*    где VSENSE - измеренное напряжение с датчика;
*    V25 - напряжение с датчика при 25°C, для STM32F407 равно 0.76 В;
*    Avg_Slope - температурный коэффициент, для STM32F407 равен 2.5 мВ/°C.
*    Время преобразования АЦП должно быть не менее 10 мкс.
* 2. На входе канала измерения напряжения батареи включен резистивный делитель на 2.
* 3. Для разрешения измерений по внутренним каналам необходимо дополнительно вызывать
*    функции ADC_TempSensorVrefintCmd(ENABLE) или ADC_VBATCmd(ENABLE).
*
*****
*/

#include "stm32_p407.h"
#include "adc.h"

//Файл конфигурации отладочной платы STM32-P407
//Функции для работы с АЦП
```

```

//-----
// ИНИЦИАЛИЗАЦИЯ АЦП ДЛЯ РАБОТЫ С ОДИНОЧНЫМ КАНАЛОМ
// Входные аргументы функции:
//   ADCx - указатель на объект: ADC1, ADC2, ADC3
//   ADC_Channel - номер канала: ADC_Channel_0...ADC_Channel_18
//   ADC_ExternalTrigConv - источник запуска, одна из следующих констант:
//     ADC_ExternalTrigConv_T1_CC1
//     ADC_ExternalTrigConv_T1_CC2
//     ADC_ExternalTrigConv_T1_CC3
//     ADC_ExternalTrigConv_T2_CC2
//     ADC_ExternalTrigConv_T2_CC3
//     ADC_ExternalTrigConv_T2_CC4
//     ADC_ExternalTrigConv_T2_TRGO
//     ADC_ExternalTrigConv_T3_CC1
//     ADC_ExternalTrigConv_T3_TRGO
//     ADC_ExternalTrigConv_T5_CC1
//     ADC_ExternalTrigConv_T5_CC2
//     ADC_ExternalTrigConv_T5_CC3
//     ADC_ExternalTrigConv_T8_CC1
//     ADC_ExternalTrigConv_T8_TRGO
//     ADC_ExternalTrigConv_Ext_IT11
//   Если соответствующий источник не сконфигурирован для запуска, аргумент может быть любой, например, 0,
//   в этом случае запуск осуществляется программно - вызовом функции ADC_SoftwareStartConv.
//
void ADC_Initialize(ADC_TypeDef* ADCx, uint8_t ADC_Channel, uint32_t ADC_ExternalTrigConv)
{
    ADC_CommonInitTypeDef ADC_CommonInitStruct; //Структура общей конфигурации АЦП
    ADC_InitTypeDef ADC_InitStructure;          //Структура конфигурации АЦП
    GPIO_InitTypeDef GPIO_InitStructure;        //Структура конфигурации портов общего назначения
    GPIO_TypeDef* GPIO_Port;                    //Указатель на базовый адрес порта
    uint32_t RCC_AHB1Periph;                    //Параметр для разрешения тактирования порта
    uint16_t GPIO_Pin;                          //Маска на разряд порта
    uint8_t flag_port = 1;                      //Признак использования разряда порта как входного

    //Сопоставление номеров АЦП и каналов разрядам портов
    switch (ADC_Channel)
    {
        case 0: RCC_AHB1Periph = RCC_AHB1Periph_GPIOA, GPIO_Port = GPIOA, GPIO_Pin = GPIO_Pin_0; break;
        case 1: RCC_AHB1Periph = RCC_AHB1Periph_GPIOA, GPIO_Port = GPIOA, GPIO_Pin = GPIO_Pin_1; break;
        case 2: RCC_AHB1Periph = RCC_AHB1Periph_GPIOA, GPIO_Port = GPIOA, GPIO_Pin = GPIO_Pin_2; break;
        case 3: RCC_AHB1Periph = RCC_AHB1Periph_GPIOA, GPIO_Port = GPIOA, GPIO_Pin = GPIO_Pin_3; break;
        case 4: if (ADCx == ADC3)
                    RCC_AHB1Periph = RCC_AHB1Periph_GPIOF, GPIO_Port = GPIOF, GPIO_Pin = GPIO_Pin_6;
                else RCC_AHB1Periph = RCC_AHB1Periph_GPIOA, GPIO_Port = GPIOA, GPIO_Pin = GPIO_Pin_4;
                break;
        case 5: if (ADCx == ADC3)
                    RCC_AHB1Periph = RCC_AHB1Periph_GPIOF, GPIO_Port = GPIOF, GPIO_Pin = GPIO_Pin_7;
                else RCC_AHB1Periph = RCC_AHB1Periph_GPIOA, GPIO_Port = GPIOA, GPIO_Pin = GPIO_Pin_5;
                break;
        case 6: if (ADCx == ADC3)
                    RCC_AHB1Periph = RCC_AHB1Periph_GPIOF, GPIO_Port = GPIOF, GPIO_Pin = GPIO_Pin_8;
                else RCC_AHB1Periph = RCC_AHB1Periph_GPIOA, GPIO_Port = GPIOA, GPIO_Pin = GPIO_Pin_6;
                break;
        case 7: if (ADCx == ADC3)
                    RCC_AHB1Periph = RCC_AHB1Periph_GPIOF, GPIO_Port = GPIOF, GPIO_Pin = GPIO_Pin_9;
                else RCC_AHB1Periph = RCC_AHB1Periph_GPIOA, GPIO_Port = GPIOA, GPIO_Pin = GPIO_Pin_7;
                break;
        case 8: if (ADCx == ADC3)
                    RCC_AHB1Periph = RCC_AHB1Periph_GPIOF, GPIO_Port = GPIOF, GPIO_Pin = GPIO_Pin_10;
                else RCC_AHB1Periph = RCC_AHB1Periph_GPIOB, GPIO_Port = GPIOB, GPIO_Pin = GPIO_Pin_0;
                break;
        case 9: RCC_AHB1Periph = RCC_AHB1Periph_GPIOB, GPIO_Port = GPIOB, GPIO_Pin = GPIO_Pin_1; break;
        case 10: RCC_AHB1Periph = RCC_AHB1Periph_GPIOC, GPIO_Port = GPIOC, GPIO_Pin = GPIO_Pin_0; break;
        case 11: RCC_AHB1Periph = RCC_AHB1Periph_GPIOC, GPIO_Port = GPIOC, GPIO_Pin = GPIO_Pin_1; break;
        case 12: RCC_AHB1Periph = RCC_AHB1Periph_GPIOC, GPIO_Port = GPIOC, GPIO_Pin = GPIO_Pin_2; break;
        case 13: RCC_AHB1Periph = RCC_AHB1Periph_GPIOC, GPIO_Port = GPIOC, GPIO_Pin = GPIO_Pin_3; break;
        case 14: if (ADCx == ADC3)
                    RCC_AHB1Periph = RCC_AHB1Periph_GPIOF, GPIO_Port = GPIOF, GPIO_Pin = GPIO_Pin_4;
                else RCC_AHB1Periph = RCC_AHB1Periph_GPIOC, GPIO_Port = GPIOC, GPIO_Pin = GPIO_Pin_4;
                break;
        case 15: if (ADCx == ADC3)
                    RCC_AHB1Periph = RCC_AHB1Periph_GPIOF, GPIO_Port = GPIOF, GPIO_Pin = GPIO_Pin_5;
                else RCC_AHB1Periph = RCC_AHB1Periph_GPIOC, GPIO_Port = GPIOC, GPIO_Pin = GPIO_Pin_5;
                break;
        default: flag_port = 0;
    }
}

```

```

//Конфигурирование разряда порта как аналогового входа
if (flag_port)
{
    RCC_AHB1PeriphClockCmd(RCC_AHB1Periph, ENABLE); //Разрешение тактирования порта
    GPIO_InitStructure.GPIO_Pin = GPIO_Pin; //Номер разряда (маска)
    GPIO_InitStructure.GPIO_Mode = GPIO_Mode_AIN; //Альтернативная функция: аналоговый вход
    GPIO_Init(GPIO_Port, &GPIO_InitStructure); //Функция конфигурирования
}

//Разрешение тактирования АЦП
if (ADCx == ADC1) RCC_APB2PeriphClockCmd(RCC_APB2Periph_ADC1, ENABLE);
if (ADCx == ADC2) RCC_APB2PeriphClockCmd(RCC_APB2Periph_ADC2, ENABLE);
if (ADCx == ADC3) RCC_APB2PeriphClockCmd(RCC_APB2Periph_ADC3, ENABLE);

//Задание общей конфигурации АЦП по умолчанию:
// - режим одиночного АЦП;
// - предварительный делитель тактовой частоты на 2;
// - запрет режима прямого доступа к памяти при совместной работе АЦП;
// - сдвиг между преобразованиями при совместной работе - 5 тактов.
ADC_CommonStructInit(&ADC_CommonInitStruct);
ADC_CommonInit(&ADC_CommonInitStruct);

//Конфигурирование АЦП
ADC_InitStructure.ADC_Resolution = ADC_Resolution_12b; //Разрядность преобразования: 12 (10, 8, 6)
ADC_InitStructure.ADC_ScanConvMode = DISABLE; //Многоканальное сканирование: не разрешено
ADC_InitStructure.ADC_ContinuousConvMode = DISABLE; //Непрерывное преобразование: запрещено
ADC_InitStructure.ADC_ExternalTrigConvEdge = ADC_ExternalTrigConvEdge_Rising; //Фронт сигнала запуска
ADC_InitStructure.ADC_ExternalTrigConv = ADC_ExternalTrigConv; //Источник запуска
ADC_InitStructure.ADC_DataAlign = ADC_DataAlign_Left; //Выравнивание кода в 16-разрядном поле: влево
ADC_InitStructure.ADC_NbrOfConversion = 1; //Число каналов для преобразования: 1
ADC_Init(ADCx, &ADC_InitStructure); //Функция конфигурирования

//Дополнительные параметры для АЦП:
// - номер входного канала,
// - номер группы преобразования - 1 (возможно до 16 групп),
// - время преобразования - 56 тактов (возможные значения: 3, 15, 28, 56, 84, 112, 144, 480)
ADC-RegularChannelConfig(ADCx, ADC_Channel, 1, ADC_SampleTime_56Cycles);

//Разрешение работы АЦП
ADC_Cmd(ADCx, ENABLE);
}

//-----
// ИНИЦИАЛИЗАЦИЯ АЦП ДЛЯ РАБОТЫ С НЕСКОЛЬКИМИ КАНАЛАМИ
// Входные аргументы функции:
// ADCx - указатель на объект: ADC1, ADC2, ADC3
// ADC_NbrOfConversion - число каналов (от 1 до 16)
// ADC_ExternalTrigConv - источник запуска, одна из констант (см. функцию выше)
//
void ADC_Init_MultiChannels(ADC_TypeDef* ADCx, uint8_t ADC_NbrOfConversion, uint32_t ADC_ExternalTrigConv)
{
    ADC_CommonInitTypeDef ADC_CommonInitStruct; //Структура общей конфигурации АЦП
    ADC_InitTypeDef ADC_InitStructure; //Структура конфигурации АЦП

    //Разрешение тактирования АЦП
    if (ADCx == ADC1) RCC_APB2PeriphClockCmd(RCC_APB2Periph_ADC1, ENABLE);
    if (ADCx == ADC2) RCC_APB2PeriphClockCmd(RCC_APB2Periph_ADC2, ENABLE);
    if (ADCx == ADC3) RCC_APB2PeriphClockCmd(RCC_APB2Periph_ADC3, ENABLE);

    //Задание общей конфигурации АЦП по умолчанию:
    // - режим одиночного АЦП;
    // - предварительный делитель тактовой частоты на 2;
    // - запрет режима прямого доступа к памяти при совместной работе АЦП;
    // - сдвиг между преобразованиями при совместной работе - 5 тактов.
    ADC_CommonStructInit(&ADC_CommonInitStruct);
    ADC_CommonInit(&ADC_CommonInitStruct);

    //Конфигурирование АЦП
    ADC_InitStructure.ADC_Resolution = ADC_Resolution_12b; //Разрядность преобразования: 12 (10, 8, 6)
    ADC_InitStructure.ADC_ScanConvMode = ENABLE; //Многоканальное сканирование: разрешено
    ADC_InitStructure.ADC_ContinuousConvMode = DISABLE; //Непрерывное преобразование: запрещено
    ADC_InitStructure.ADC_ExternalTrigConvEdge = ADC_ExternalTrigConvEdge_Rising; //Фронт сигнала запуска
    ADC_InitStructure.ADC_ExternalTrigConv = ADC_ExternalTrigConv; //Источник запуска
    ADC_InitStructure.ADC_DataAlign = ADC_DataAlign_Left; //Выравнивание кода в 16-разрядном поле: влево
    ADC_InitStructure.ADC_NbrOfConversion = ADC_NbrOfConversion; //Число каналов
    ADC_Init(ADCx, &ADC_InitStructure); //Функция конфигурирования
}

```

```
//-----
// КОНФИГУРИРОВАНИЕ КАНАЛА ПРИ МНОГОКАНАЛЬНОМ РЕЖИМЕ РАБОТЫ АЦП
// Входные аргументы функции:
// ADCx - указатель на объект: ADC1, ADC2, ADC3
// ADC_Channel - номер канала: ADC_Channel_0...ADC_Channel_18
// Rank - номер канала в последовательности опроса
// ADC_SampleTime - время преобразования в тактах, одна из констант:
//   ADC_SampleTime_3Cycles, ADC_SampleTime_15Cycles, ADC_SampleTime_28Cycles,
//   ADC_SampleTime_56Cycles, ADC_SampleTime_84Cycles, ADC_SampleTime_112Cycles,
//   ADC_SampleTime_144Cycles, ADC_SampleTime_480Cycles
// GPIO_Port - базовый адрес порта: GPIOA, GPIOB, GPIOC, GPIOD, GPIOE,
// GPIO_F - номер разряда порта в форме маски: GPIO_Pin_0...GPIO_Pin_15
void ADC_ChannelConfig(ADC_TypeDef* ADCx, uint8_t ADC_Channel, uint8_t Rank, uint8_t ADC_SampleTime,
                      GPIO_TypeDef* GPIO_Port, uint32_t GPIO_Pin)
{
    GPIO_InitTypeDef GPIO_InitStructure;           //Структура конфигурации портов общего назначения
    uint32_t RCC_AHB1Periph;                       //Параметр для разрешения тактирования порта

    //Конфигурирование разряда порта
    if (GPIO_Port == GPIOA) RCC_AHB1Periph = RCC_AHB1Periph_GPIOA;
    else if (GPIO_Port == GPIOB) RCC_AHB1Periph = RCC_AHB1Periph_GPIOB;
    else if (GPIO_Port == GPIOC) RCC_AHB1Periph = RCC_AHB1Periph_GPIOC;
    else if (GPIO_Port == GPIOD) RCC_AHB1Periph = RCC_AHB1Periph_GPIOD;
    else if (GPIO_Port == GPIOE) RCC_AHB1Periph = RCC_AHB1Periph_GPIOE;
    else RCC_AHB1Periph = 0;
    RCC_AHB1PeriphClockCmd(RCC_AHB1Periph, ENABLE); //Разрешение тактирования порта
    GPIO_InitStructure.GPIO_Pin = GPIO_Pin;        //Номер разряда (маска)
    GPIO_InitStructure.GPIO_Mode = GPIO_Mode_AIN;  //Альтернативная функция: аналоговый вход
    GPIO_Init(GPIO_Port, &GPIO_InitStructure);    //Функция конфигурирования

    ADC_RegularChannelConfig(ADCx, ADC_Channel, Rank, ADC_SampleTime);
}

```

Заголовочный модуль adc.h

```
/**
 * Файл      adc.h
 * Copyright (C) 2016 МИРЭА
 * Версия    2.1
 * Описание: Функции инициализации и обслуживания АЦП
 */

#ifndef _ADC_H
#define _ADC_H

/* Объявления функций для их использования в других программных модулях */
void ADC_Initialize(ADC_TypeDef* ADCx, uint8_t ADC_Channel, uint32_t ADC_ExternalTrigConv);
void ADC_Init_MultiChannels(ADC_TypeDef* ADCx, uint8_t ADC_NbrOfConversion, uint32_t ADC_ExternalTrigConv);
void ADC_ChannelConfig(ADC_TypeDef* ADCx, uint8_t ADC_Channel, uint8_t Rank, uint8_t ADC_SampleTime,
                      GPIO_TypeDef* GPIO_Port, uint32_t GPIO_Pin);

//-----
#endif /* _ADC_H */

```

Исходный модуль dac.c

```
/**
 * Файл      dac.c
 * Copyright (C) 2015 МИРЭА
 * Версия    2.2
 * Описание: Функции инициализации и обслуживания ЦАП
 *
 * ЦАП может быть сконфигурирован для запуска от таймеров
 * TIM2, TIM4, TIM5, TIM6, TIM7 соответствующей коррекцией в файле dac.h
 * (другие источники - EXTI Line9, TIM8 - требуют более сложной модификации)
 *
 * Формат данных для ЦАП - 12-разрядный смещенный код:
 * код 1111 1111 1111 соответствует Uвых = максимум (практически Vref);
 * код 1000 0000 0000 соответствует Uвых = Vref / 2;
 * код 0000 0000 0000 соответствует Uвых = минимум (практически 0);
 * где Vref - опорное напряжение (для отладочной платы равно напряжению питания)
 */

```

```

*
*****
*/

#include "stm32_p407.h"           //Файл конфигурации отладочной платы STM32-P407
#include "dac.h"                 //Константы и функции для работы с ЦАП

//define USE_DAC2                //Определение для использования второго канала ЦАП,
                                //режим возможен, если не используется графический дисплей

void DACTimerConfig(uint32_t SamplingFreq);
void DACConfig(uint32_t DAC_Trigger);

//-----
// ИНИЦИАЛИЗАЦИЯ ЦАП И СИСТЕМЫ ТАКТИРОВАНИЯ
// Входной параметр - частота дискретизации в Гц; если задано нулевое значение,
// то синхронизация не используется, запуск ЦАП производится программно -
// вызовом функции DAC_SetChannel1Data или DAC_SetChannel2Data
void DACInit(uint32_t SamplingFreq)
{
    if (SamplingFreq)
    {
        //Инициализация таймера - источника тактирования ЦАП
        DACTimerConfig(SamplingFreq);

        //Инициализация ЦАП с запуском от таймера
        DACConfig(DAC_Trigger_TD_TRGO);

        //Разрешение прерываний от таймера
        //DACTimerInterruptConfig(ENABLE);
    }
    //Если не задана частота дискретизации, инициализация ЦАП
    // с режимом запуска по загрузке данных
    else DACConfig(DAC_Trigger_None);
}

//-----
// ИНИЦИАЛИЗАЦИЯ ТАЙМЕРА ДЛЯ ТАКТИРОВАНИЯ ЦАП
/* Подпрограмма используется для таймеров TIM2, TIM4, TIM5, TIM6, TIM7,
подключенных к шине APB1. По умолчанию частота работы шины 42 МГц,
частота тактирования таймеров этой шины - 84 МГц.
При отсутствии предделения максимальный коэффициент деления для 16-разрядного
таймера равен 65536, т.е. минимальная частота дискретизации для этого случая
84000000 Гц / 65536 = 1282 Гц.
Для меньших значений частот необходимо предделение или применение 32-разрядного таймера.
В настоящей подпрограмме для частоты дискретизации SamplingFreq, заданной в Гц:
- если предделитель не используется (его коэф. деления = 1),
число, загружаемое в счетчик: TIM_Period = 84000000[Гц] / SamplingFreq[Гц] - 1;
- если коэффициент предделителя равен 1000,
число, загружаемое в счетчик: TIM_Period = 84000[Гц] / SamplingFreq[Гц] - 1;
Для еще более низких частот (< 2 Гц), а также повышения точности необходимо
выбирать 32-разрядный таймер (TIM2, TIM5).
Прерывания от таймера конфигурируются по событию его автоперезагрузки.
*/
void DACTimerConfig(uint32_t SamplingFreq)
{
    //Структура конфигурации базового таймера
    TIM_TimeBaseInitTypeDef TIM_TimeBaseStructure;

    //Разрешение тактирования таймера
    RCC_APB1PeriphClockCmd(RCC_APB1Periph_TIMD, ENABLE);

    //Базовая конфигурация таймера
    if (SamplingFreq > 1300)
    {
        TIM_TimeBaseStructure.TIM_Period = 84000000u/SamplingFreq-1; //Основной коэф. деления
        TIM_TimeBaseStructure.TIM_Prescaler = 0;                     //Предделение не используется
    }
    else
    {
        TIM_TimeBaseStructure.TIM_Period = 84000u/SamplingFreq-1; //Основной коэф. деления
        TIM_TimeBaseStructure.TIM_Prescaler = 1000-1;             //Предделитель в 1000 раз
    }
    TIM_TimeBaseStructure.TIM_ClockDivision = 0;                  //Делитель для входного фильтра
    TIM_TimeBaseStructure.TIM_CounterMode = TIM_CounterMode_Up; //Счетчик вверх: от 0 до TIM_Period
    TIM_TimeBaseInit(TIMD, &TIM_TimeBaseStructure);              //Функция конфигурирования

    //Активизация триггерного выхода TRGO таймера для запуска ЦАП
    TIM_SelectOutputTrigger(TIMD, TIM_TRGOSource_Update);
}

```

```

//Разрешение прерываний при перезагрузке таймера
TIM_ITConfig(TIMD, TIM_IT_Update, ENABLE);

//Общее разрешение работы таймера
TIM_Cmd(TIMD, ENABLE);
}

//-----
// ИНИЦИАЛИЗАЦИЯ ЦАП
// Используются DAC1 с выходом на разряд порта PA4
//           и DAC2 с выходом на разряд порта PA5
// Входной параметр: источник запуска ЦАП - одна из predetermined констант:
// DAC_Trigger_None, DAC_Trigger_T2_TRGO, DAC_Trigger_T4_TRGO, DAC_Trigger_T5_TRGO,
// DAC_Trigger_T6_TRGO, DAC_Trigger_T7_TRGO, DAC_Trigger_T8_TRGO, DAC_Trigger_Ext_IT9,
// DAC_Trigger_Software
void DACConfig(uint32_t DAC_Trigger)
{
    GPIO_InitTypeDef GPIO_InitStructure;           //Структура конфигурации портов общего назначения
    DAC_InitTypeDef DAC_InitStructure;             //Структура конфигурации ЦАП

    //Разрешение тактирования порта PA, разряды которого используются как выходы ЦАП
    RCC_AHB1PeriphClockCmd(RCC_AHB1Periph_GPIOA, ENABLE);

    //Разрешение тактирования ЦАП
    RCC_APB1PeriphClockCmd(RCC_APB1Periph_DAC, ENABLE);

    //Конфигурирование разрядов PA.4 (PA.5) как аналоговых
#ifdef USE_DAC2
    GPIO_InitStructure.GPIO_Pin = GPIO_Pin_4 | GPIO_Pin_5;           //Номера разрядов (маски) для DAC1, DAC2
#else
    GPIO_InitStructure.GPIO_Pin = GPIO_Pin_4;                         //Номер разряда (маска) для DAC1
#endif
    GPIO_InitStructure.GPIO_Mode = GPIO_Mode_AN;                     //Альтернат.функция: аналоговый вход-выход
    GPIO_InitStructure.GPIO_PuPd = GPIO_PuPd_NOPULL;                 //Без подтягивающих резисторов
    GPIO_Init(GPIOA, &GPIO_InitStructure);                           //Функция конфигурирования

    //Конфигурирование режимов работы ЦАП
    DAC_InitStructure.DAC_Trigger = DAC_Trigger;                     //Источник запуска
    DAC_InitStructure.DAC_WaveGeneration = DAC_WaveGeneration_None; //Встроенный генератор: не используется
    DAC_InitStructure.DAC_LFSRUnmask_TriangleAmplitude = 0xA00;      //Маска/амплитуда для встр. генератора
    DAC_InitStructure.DAC_OutputBuffer = DAC_OutputBuffer_Enable;    //Разреш. выходного буфера повыш. мощности
    DAC_Init(DAC_Channel_1, &DAC_InitStructure);                     //Функция конфигурирования для DAC1
#ifdef USE_DAC2
    DAC_Init(DAC_Channel_2, &DAC_InitStructure);                     //Функция конфигурирования для DAC2
#endif

    //Общее разрешение работы ЦАП, загрузка начального значения данных ЦАП (код - смещенный)
    DAC_Cmd(DAC_Channel_1, ENABLE);
    DAC_SetChannel1Data(DAC_Align_12b_L, 0x0000);
#ifdef USE_DAC2
    DAC_Cmd(DAC_Channel_2, ENABLE);
    DAC_SetChannel2Data(DAC_Align_12b_L, 0x0000);
#endif
}

//-----
// РАЗРЕШЕНИЕ/ЗАПРЕТ ГЛОБАЛЬНЫХ ПЕРЕРЫВАНИЙ ОТ ТАЙМЕРА
void DACTimerInterruptConfig(FunctionalState NewState)
{
    NVIC_InitTypeDef NVIC_InitStructure;           //Структура конфигурации прерываний

    NVIC_InitStructure.NVIC_IRQChannel = TIMD_IRQn; //Номер (линия) прерывания
    NVIC_InitStructure.NVIC_IRQChannelPreemptionPriority = 1; //Приоритет группы
    NVIC_InitStructure.NVIC_IRQChannelSubPriority = 0; //Приоритет внутри группы
    NVIC_InitStructure.NVIC_IRQChannelCmd = NewState; //Разрешение/запрет прерывания
    NVIC_Init(&NVIC_InitStructure); //Функция конфигурирования
}

//-----
/* Ш А Б Л О Н
// ОБРАБОТЧИК ПЕРЕРЫВАНИЯ ОТ ТАЙМЕРА
// Вызывается с заданной частотой дискретизации ЦАП.
// Код для передачи в ЦАП - смещенный, выравненный влево.
uint16_t DACData1, DACData2; //Данные для ЦАП в смещенном коде
void TIMD_IRQHandler(void)

```

```

{
    if (TIM_GetITStatus(TIMD, TIM_IT_Update) != RESET)    //Проверка флага типа прерывания
    {                                                       //(необязательна, если других типов нет)
        TIM_ClearITPendingBit(TIMD, TIM_IT_Update);       //Очистка флага
        DAC_SetChannel1Data(DAC_Align_12b_L, DACData1);   //Загрузка данных в ЦАП 1
        DAC_SetChannel2Data(DAC_Align_12b_L, DACData2);   //Загрузка данных в ЦАП 2
    }
}
*/
//-----

```

Заголовочный модуль dac.h

```

/**
*****
* Файл      dac.h
* Copyright (C) 2015 МИРЭА
* Версия    2.2
* Описание:  Функции инициализации и обслуживания ЦАП
*****
*/

#ifndef _DAC_H
#define _DAC_H

/* В следующих определениях вместо TIM7 могут быть использованы TIM2, TIM4, TIM5, TIM7 */
#define TIMD      TIM7
#define DAC_Trigger_TD_TRGO DAC_Trigger_T7_TRGO
#define RCC_APB1Periph_TIMD RCC_APB1Periph_TIM7
#define TIMD_IRQn TIM7_IRQn
#define TIMD_IRQHandler TIM7_IRQHandler

/* Определения при использовании TIM6: */
#define TIMD      TIM6
#define DAC_Trigger_TD_TRGO DAC_Trigger_T6_TRGO
#define RCC_APB1Periph_TIMD RCC_APB1Periph_TIM6
#define TIMD_IRQn TIM6_DAC_IRQn
#define TIMD_IRQHandler TIM6_DAC_IRQHandler
*/

/* Объявления функций для их использования в других программных модулях */
void DACInit(uint32_t SamplingFreq);
void DACTimerInterruptConfig(FunctionalState NewState);

//-----
#endif /* _DAC_H */

```

Исходный модуль codec.c

```

/**
*****
* Файл      codec.c
* Copyright (C) 2016 МИРЭА
* Версия    2.2
* Описание:  Функции инициализации и обслуживания аудиокодека
*****
*/

#include "stm32_p407.h"    //Файл конфигурации отладочной платы STM32-P407
#include "codec.h"         //Функции для работы с кодеком

//-----
// ИНИЦИАЛИЗАЦИЯ ЗВУКОВОГО КОДЕКА
/*
Формат данных для аудиокодека - 16-разрядный дополнительный код,
с учетом включенного после аудиокодека усилителя:
код 0111 1111 1111 1111 соответствует Uвых = +Um;
код 0000 0000 0000 0000 соответствует Uвых = 0;
код 1000 0000 0000 0000 соответствует Uвых = -Um;
где Um примерно равно 1 В

Рекомендуемые настройки PLL I2S для I2S_Mode_Master
Частота  PLLI2SN PLLI2SR I2SDIV
8000 Гц   256      5      12

```


16000 Гц	213	2	13
22050 Гц	429	4	9
32000 Гц	213	2	6
44100 Гц	271	2	6
48000 Гц	258	3	3

Реальная частота дискретизации: $1 \text{ МГц} * PLLI2SN / PLLI2SR / I2SDIV / 512$
 PLLI2SN PLLI2SR - задаются в RCC_PLLI2SConfig(PLLI2SN, PLLI2SR)
 I2SDIV вычисляется в библиотечной I2S_Init()

Для задания частоты дискретизации в stm32f4xx_spi.h объявлены следующие константы

```
#define I2S_AudioFreq_192k ((uint32_t)192000)
#define I2S_AudioFreq_96k ((uint32_t)96000)
#define I2S_AudioFreq_48k ((uint32_t)48000)
#define I2S_AudioFreq_44k ((uint32_t)44100)
#define I2S_AudioFreq_32k ((uint32_t)32000)
#define I2S_AudioFreq_22k ((uint32_t)22050)
#define I2S_AudioFreq_16k ((uint32_t)16000)
#define I2S_AudioFreq_11k ((uint32_t)11025)
#define I2S_AudioFreq_8k ((uint32_t)8000)
```

*/

```
void SoundCodecConfig(uint32_t AudioFreq)
{
```

```
//Структуры конфигурации
GPIO_InitTypeDef GPIO_InitStructure;
I2S_InitTypeDef I2S_InitStructure;
```

```
//Конфигурирование синтезатора ФАПЧ для интерфейса I2S (PLL I2S)
```

```
RCC_PLLI2SConfig(213,2);
//Разрешение работы PLL I2S
RCC_PLLI2SCmd(ENABLE);
//Ожидание готовности PLL I2S
while(RESET == RCC_GetFlagStatus(RCC_FLAG_PLLI2SRDY));
//Задание PLL I2S как источника тактирования I2S
RCC_I2SCLKConfig(RCC_I2SCLKSource_PLLI2S);
//Разрешение тактирования модуля SPI3/I2S3 по шине APB1
RCC_APB1PeriphClockCmd(RCC_APB1Periph_SPI3, ENABLE);
//Деинициализация (сброс) интерфейса SPI3/I2S3
SPI_I2S_DeInit(SPI3);
```

```
//Конфигурирование интерфейса I2S3
```

```
I2S_InitStructure.I2S_Mode = I2S_Mode_MasterTx; //Режим: _SlaveTx _SlaveRx _MasterTx _MasterRx
I2S_InitStructure.I2S_Standard = I2S_Standard_Phillips; // _Phillips _MSB _LSB _PCMSHORT _PCMLong
I2S_InitStructure.I2S_DataFormat = I2S_DataFormat_16b; //Формат: _16b _16bextended _24b _32b
I2S_InitStructure.I2S_MCLKOutput = I2S_MCLKOutput_Enable; //Разрешение выдачи на выход тактового сигнала
I2S_InitStructure.I2S_AudioFreq = AudioFreq; //Частота дискретизации: _8k ... _192k
I2S_InitStructure.I2S_CPOL = I2S_CPOL_Low; //Исходный уровень тактового сигнала: _Low _High
I2S_Init(SPI3, &I2S_InitStructure); //Функция конфигурирования
```

```
//Запрет прерывания I2S3 TXE
```

```
SPI_I2S_ITConfig(SPI3, SPI_I2S_IT_TXE, DISABLE);
```

```
//Конфигурирование выводов процессора
```

```
//Разрешение тактирования портов A, B, C
```

```
RCC_AHB1PeriphClockCmd(RCC_AHB1Periph_GPIOA | RCC_AHB1Periph_GPIOB | RCC_AHB1Periph_GPIOC, ENABLE);
```

```
//Конфигурирование PA15 (внешний сигнал I2S3_WS)
```

```
GPIO_InitStructure.GPIO_Pin = GPIO_Pin_15;
GPIO_InitStructure.GPIO_OType = GPIO_OType_PP;
GPIO_InitStructure.GPIO_Mode = GPIO_Mode_AF;
GPIO_InitStructure.GPIO_PuPd = GPIO_PuPd_NOPULL;
GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;
GPIO_Init(GPIOA, &GPIO_InitStructure);
GPIO_PinAFConfig(GPIOA, GPIO_PinSource15, GPIO_AF_SPI3); //Подключение вывода к I2S
```

```
//Конфигурирование PB3 (внешний сигнал I2S3_CK) и PB5 (внешний сигнал I2S3_SD)
```

```
GPIO_InitStructure.GPIO_Pin = GPIO_Pin_3 | GPIO_Pin_5;
GPIO_InitStructure.GPIO_OType = GPIO_OType_PP;
GPIO_InitStructure.GPIO_Mode = GPIO_Mode_AF;
GPIO_InitStructure.GPIO_PuPd = GPIO_PuPd_NOPULL;
GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;
GPIO_Init(GPIOB, &GPIO_InitStructure);
GPIO_PinAFConfig(GPIOB, GPIO_PinSource3, GPIO_AF_SPI3);
GPIO_PinAFConfig(GPIOB, GPIO_PinSource5, GPIO_AF_SPI3);
```

```
//Конфигурирование PC7 (внешний сигнал I2S3_MCK)
```

```
GPIO_InitStructure.GPIO_Pin = GPIO_Pin_7;
GPIO_InitStructure.GPIO_OType = GPIO_OType_PP;
```



```

GPIO_InitStructure.GPIO_Mode = GPIO_Mode_AF;
GPIO_InitStructure.GPIO_PuPd = GPIO_PuPd_NOPULL;
GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;
GPIO_Init(GPIOC, &GPIO_InitStructure);
GPIO_PinAFConfig(GPIOC, GPIO_PinSource7, GPIO_AF_SPI3);

//Разрешение определенного типа прерываний: при пустом передающем буфере
SPI_I2S_ITConfig(SPI3, SPI_I2S_IT_TXE, ENABLE);

//Разрешение глобальных прерываний
SoundCodecInterruptConfig(ENABLE);

//Разрешение работы интерфейса SPI3/I2S3
I2S_Cmd(SPI3, ENABLE);
}

//-----
// РАЗРЕШЕНИЕ/ЗАПРЕТ ГЛОБАЛЬНЫХ ПРЕРЫВАНИЙ ОТ SPI3/I2S3 (КОДЕКА)
void SoundCodecInterruptConfig(FunctionalState NewState)
{
    NVIC_InitTypeDef NVIC_InitStructure;           //Структура конфигурации прерываний

    NVIC_InitStructure.NVIC_IRQChannel = SPI3_IRQn; //Номер (линия) прерывания
    NVIC_InitStructure.NVIC_IRQChannelPreemptionPriority = 2; //Приоритет группы
    NVIC_InitStructure.NVIC_IRQChannelSubPriority = 0; //Приоритет внутри группы
    NVIC_InitStructure.NVIC_IRQChannelCmd = NewState; //Разрешение/запрет прерывания
    NVIC_Init(&NVIC_InitStructure); //Функция конфигурирования
}

//-----
// ОБРАБОТЧИК ПРЕРЫВАНИЯ ОТ I2S3 (КОДЕКА)
// Вызывается с двойной частотой дискретизации.
// За одно обслуживание передается 16-битный отсчет одного канала.
// Вызов вторичного обработчика Sample_Handler(), формирующего отсчеты
// сразу двух каналов, происходит в 2 раза реже.
// Формат данных для кодека - дополнительный.

int16_t DataChannel1, DataChannel2; //Выходные отсчеты

void SPI3_IRQHandler(void)
{
    static uint16_t select_chan; //Флаг переключателя каналов
    if (SPI_I2S_GetFlagStatus(SPI3, SPI_I2S_FLAG_TXE) != RESET) //Проверка флага запроса
    {
        SPI_I2S_ClearFlag(SPI3, SPI_I2S_FLAG_TXE); //Сброс флага запроса прерывания
        if (!select_chan)
        {
            Sample_Handler(); //Вторичный обработчик
            SPI_I2S_SendData(SPI3, DataChannel1); //Передача кодеку отсчета первого канала
        }
        else SPI_I2S_SendData(SPI3, DataChannel2); //Передача кодеку отсчета второго канала
        select_chan ^= 1; //Переключение каналов
    }
}

//-----
// ВТОРИЧНЫЙ ОБРАБОТЧИК ОТ КОДЕКА ПО УМОЛЧАНИЮ
// Вызывается с частотой дискретизации
__weak void Sample_Handler(void)
{
}
//-----

```

Заголовочный модуль codec.h

```

/**
 * *****
 * Файл      codec.h
 * Copyright (C) 2016 МИРЭА
 * Версия    2.2
 * Описание: Функции инициализации и обслуживания аудиокодека
 * *****
 */

#ifndef _CODEC_H
#define _CODEC_H

```

```

/* Объявления функций для их использования в других программных модулях */
void SoundCodecConfig(uint32_t AudioFreq);
void SoundCodecInterruptConfig(FunctionalState NewState);
void Sample_Handler(void);           //Эта функция - для реализации в другом модуле

#endif /* _CODEC_H */

```

Фрагменты библиотечного модуля ..\Libraries\Board\lcd.c

```

/* Заголовочные файлы */
#include "drv_glcd_cnfg.h"
#include "drv_glcd.h"
#include "glcd_ll.h"
#include "lcd.h"

pFontType_t _Font_Type;           //Текущий тип (размер) шрифта
unsigned int _Font_Color;         //Текущий цвет символов шрифта
unsigned int _Back_Color;        //Текущий цвет фона дисплея и шрифта
unsigned int _Pen_Color;         //Текущий цвет линий
extern FontType_t Terminal_9,    //Параметры имеющихся шрифтов
                Terminal_11, Terminal_14, Terminal_18;

/*****
* Функция: LCD_Init
* Инициализация дисплея
*/
void LCD_Init(void)
{
    GLCD_PowerUpInit(NULL);
    GLCD_Backlight(BACKLIGHT_ON);
    _Font_Type = &Terminal_14;
    _Font_Color = DEF_FONT_COLOUR;
    _Back_Color = DEF_BACKGND_COLOUR;
    GLCD_SetFont(_Font_Type, _Font_Color, _Back_Color);
    GLCD_TextSetPos(0,0);
}

/*****
* Функция: LCD_Clear
* Очистка дисплея
* Параметры: back - 0 (по умолчанию) светлый фон, 1 - темный фон
* Возврат: Нет
*/
void LCD_Clear(int back)
{
    GLCD_SetWindow(3, 0, GLCD_HORIZONTAL_SIZE - 1, GLCD_VERTICAL_SIZE - 1);
    GLCD_DisplayClear(back);
    if (back)
    {
        _Back_Color = 0x000;
        if (_Font_Color == 0x000) _Font_Color = 0xFFF;
    }
    else
    {
        _Back_Color = DEF_BACKGND_COLOUR;
        if (_Font_Color == 0xFFF) _Font_Color = 0x000;
    }
    GLCD_SetFont(_Font_Type, _Font_Color, _Back_Color);
    GLCD_TextSetPos(0,0);
}

/*****
* Функция: LCD_FontSize
* Установка размера шрифта
* Параметры: Size - размер символов, точек
* Возврат: Нет
*/
void LCD_FontSize(unsigned int Size)
{
    if (Size <= 10) _Font_Type = &Terminal_9;
    else if (Size <= 12) _Font_Type = &Terminal_11;
    else if (Size <= 16) _Font_Type = &Terminal_14;
    else _Font_Type = &Terminal_18;
    GLCD_SetFont(_Font_Type, _Font_Color, _Back_Color);
}

```

Проект ..\Lab3_adc\MDK-ARM\adc.uvproj

1. Измерения с помощью аналого-цифрового преобразователя

Исходный модуль ..\Lab3_adc\main.c

```
//-----  
//  
//  УЧЕБНЫЙ ПРОЕКТ  
//  Измерения с помощью аналого-цифрового преобразователя  
//  Copyright (C) 2016 МИРЭА  
//  
//  Для тестирования работы АЦП используется установленный на плате потенциометр  
//  (триммер), посредством которого на разряде порта PC0 можно изменять напряжение  
//  в пределах 0...3.3 В. Разряд PC0 имеет альтернативные функции: аналоговые входы  
//  ADC1_IN10, ADC2_IN10, ADC3_IN10.  
//  Проект иллюстрирует измерение медленно меняющихся уровней сигнала, слежение  
//  за выходом значений за заданные границы.  
//  
//-----  
  
#include "stm32_p407.h"           //Файл конфигурации отладочной платы STM32-P407  
#include "adc.h"                 //Функции для работы с АЦП  
#include "lcd.h"                 //Функции для работы с дисплеем  
  
#define AVERAGE 200             //Число усреднений данных с АЦП  
  
//-----  
// ГЛАВНАЯ ФУНКЦИЯ  
int main()  
{  
    //Суммированные данные с АЦП  
    unsigned short UData16;      //Приведенный смещенный (беззнаковый) код  
    short Data16;                //Приведенный дополнительный (знаковый) код  
    int Average = 1;             //Текущее число усреднений  
    int i;                       //Текущая переменная цикла  
    NVIC_InitTypeDef NVIC_InitStructure; //Структура конфигурации прерываний  
  
    //Инициализация кнопок  
    STM_PBInit(BUTTON_WAKEUP, BUTTON_MODE_GPIO); //Кнопка WAKEUP выхода из программы  
    STM_PBInit(BUTTON_RIGHT, BUTTON_MODE_GPIO);  //Позиция джойстика для включения режима усреднения  
    STM_PBInit(BUTTON_LEFT, BUTTON_MODE_GPIO);   //Позиция джойстика для выключения режима усреднения  
    STM_PBInit(BUTTON_DOWN, BUTTON_MODE_GPIO);   //Позиция джойстика для временной остановки измерений  
  
    //Инициализация индикаторов  
    STM_LEDInit(LED3); STM_LEDOff(LED3);  
  
    //Инициализация дисплея  
    LCD_Init(); LCD_Clear(1);  
    LCD_FontSize(11);  
    LCD_FontColor(0x0F0);  
    LCD_TextPos(1,1); LCD_print("ВЫРАВНИВ.ВЛЕВО");  
    LCD_TextPos(1,4); LCD_print("ВЫРАВНИВ.ВПРАВО");  
    LCD_FontColor(0x08F);  
    LCD_TextPos(1,2); LCD_print("Смещ. Дополн.");  
    LCD_TextPos(1,5); LCD_print("Смещ. Дополн.");  
    LCD_FontColor(0xFF);  
  
    //Инициализация и первичный запуск АЦП  
    ADC_Initialize(ADC2, ADC_Channel_10, 0);  
    ADC_SoftwareStartConv(ADC2);  
  
    //Конфигурирование прерываний от АЦП  
    NVIC_InitStructure.NVIC_IRQChannel = ADC_IRQn; //Номер (линия) прерывания  
    NVIC_InitStructure.NVIC_IRQChannelPreemptionPriority = 0; //Приоритет группы  
    NVIC_InitStructure.NVIC_IRQChannelSubPriority = 0; //Приоритет внутри группы  
    NVIC_InitStructure.NVIC_IRQChannelCmd = ENABLE; //Разрешение/запрет прерывания  
    NVIC_Init(&NVIC_InitStructure); //Функция конфигурирования  
  
    //Программирование АЦП в режиме "аналогового сторожа"  
    ADC_AnalogWatchdogSingleChannelConfig(ADC2, ADC_Channel_10); //Выбор канала
```

```

ADC_AnalogWatchdogThresholdsConfig(ADC2, 3500, 500); //Верхний и нижний пороги
ADC_ITConfig(ADC2, ADC_IT_AWD, ENABLE); //Разрешение прерываний от "сторожа"
ADC_AnalogWatchdogCmd(ADC2, ADC_AnalogWatchdog_SingleRegEnable); //Разрешение "аналогового сторожа"

//Цикл в основной программе
while (1)
{
    //Включение/отключение режима усреднения
    if (STM_PBGetState(BUTTON_RIGHT))
    { Average = AVERAGE;
      LCD_TextPos(1,0); LCD_print("- Усреднение -");
    }
    if (STM_PBGetState(BUTTON_LEFT))
    { Average = 1;
      LCD_TextPos(1,0); LCD_print(" ");
    }

    //Пользовательская остановка измерений для чтения показаний
    while (STM_PBGetState(BUTTON_DOWN));

    //Цикл по измерениям
    for (i = Average, AdcData = 0; i > 0; i--)
    {
        while (!ADC_GetFlagStatus(ADC2, ADC_FLAG_EOC)); //Ожидание окончания преобразования
        AdcData += ADC_GetConversionValue(ADC2); //Ввод и суммирование показаний
        ADC_SoftwareStartConv(ADC2); //Новый запуск преобразования
        Delay_ms(1);
    }

    //Преобразование кода с АЦП
    UData16 = AdcData / Average; //Вычисление среднего
    Data16 = UData16 ^ 0x8000; //Преобразование смещенного кода в дополнительный

    //Вывод результатов на дисплей
    LCD_TextPos(1,3); LCD_print("%5d", UData16); //Смещенный с выравниванием влево
    LCD_TextPos(8,3); LCD_print("%6d", Data16); //Дополнительный с выравниванием влево
    Data16 >>= 4; UData16 >>= 4; //Имитация выравнивания вправо
    LCD_TextPos(1,6); LCD_print("%5d", UData16); //Смещенный с выравниванием вправо
    LCD_TextPos(8,6); LCD_print("%6d", Data16); //Дополнительный с выравниванием вправо

    if (Average < AVERAGE) Delay_ms(300); //Период обновления индикации
    STM_LEDOff(LED3); //Гашение индикатора "аналогового сторожа"

    //Проверка нажатия кнопки WAKEUP завершения работы (сброса процессора)
    if (STM_PBGetState(BUTTON_WAKEUP)) NVIC_SystemReset();
}
}

//-----
// ОБРАБОТЧИК ПРЕРЫВАНИЯ ОТ АЦП
void ADC_IRQHandler(void)
{
    if (ADC_GetFlagStatus(ADC2, ADC_FLAG_AWD) != RESET) //Проверка флага "аналогового сторожа"
    {
        ADC_ClearFlag(ADC2, ADC_FLAG_AWD); //Сброс флага
        STM_LEDOn(LED3); //Сигнализация о выходе уровня за пределы
    }
}

//-----

```

Проект ..\Lab3_dac\MDK-ARM\dac.uvproj

2. Вывод сигналов с использованием ЦАП и аудиокодека

Исходный модуль ..\Lab3_dac\main.c

```

//-----
//
// УЧЕБНЫЙ ПРОЕКТ
// Вывод сигналов с использованием ЦАП и аудиокодека
// Copyright (C) 2016 МИРЭА

```

```

//
// Формат данных для ЦАП - 12-разрядный смещенный код:
// код 1111 1111 1111 соответствует Uвых = максимум (практически Vref);
// код 1000 0000 0000 соответствует Uвых = Vref / 2;
// код 0000 0000 0000 соответствует Uвых = минимум (практически 0);
// где Vref - опорное напряжение (для отладочной платы равно напряжению питания)
//
// Формат данных для аудиокодека - 16-разрядный дополнительный код,
// с учетом дополнительного усилителя после аудиокодека:
// код 0111 1111 1111 1111 соответствует Uвых = +Um;
// код 0000 0000 0000 0000 соответствует Uвых = 0;
// код 1000 0000 0000 0000 соответствует Uвых = -Um;
// где Um примерно равно 1 В
//
//-----

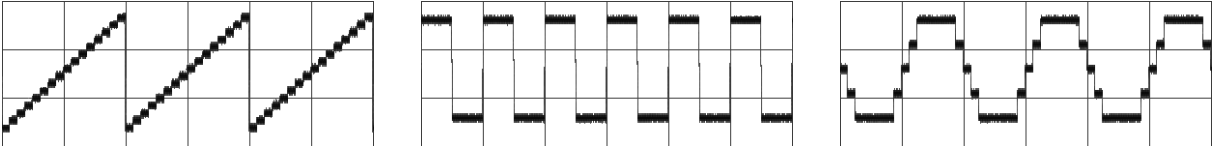
#include "stm32_p407.h" //Файл конфигурации отладочной платы STM32-P407
#include "dac.h" //Константы и функции для работы с ЦАП
#include "codec.h" //Функции для работы с кодеком

#define DMA_ENA //Определение для варианта использования DMA

uint16_t DACData1, DACData2; //Данные для ЦАП в смещенном коде
extern int16_t DataChannel1, //Данные для кодека в дополнительном коде
DataChannel2; // (определены в модуле codec.c)

//Таблицы отсчетов сигналов (в смещенном формате)
uint16_t Signal0[] = { 8000,11000,14000,17000,20000,23000,26000,29000,
32000,35000,38000,41000,44000,47000,50000,53000 };
uint16_t Signal1[] = { 12000,12000,12000,12000,52000,52000,52000,52000,
12000,12000,12000,12000,52000,52000,52000,52000 };
uint16_t Signal2[] = { 22000,32000,42000,52000,52000,52000,52000,52000,
42000,32000,22000,12000,12000,12000,12000,12000 };

// ОСЦИЛЛОГРАММЫ ТЕСТОВЫХ СИГНАЛОВ НА ВЫХОДЕ ЦАП



uint16_t* Signal = Signal0; //Указатель на одну из таблиц отсчетов

uint16_t index_dac; //Индекс отсчета сигнала, выводимого через ЦАП
uint16_t index_codec; //Индекс отсчета сигнала, выводимого через кодек
int signal_type; //Тип генерируемого сигнала (номер: 0, 1 или 2)

void DMA_DAC_Initialize(void);
void DMA_CODEC_Initialize(void);

//-----
// ГЛАВНАЯ ФУНКЦИЯ
int main()
{
//Инициализация органов управления в режиме генерации запросов прерывания
STM_PBInit(BUTTON_WAKEUP, BUTTON_MODE_EXTI); //Кнопка WAKEUP
STM_PBInit(BUTTON_RIGHT, BUTTON_MODE_EXTI); //Позиции джойстика
STM_PBInit(BUTTON_LEFT, BUTTON_MODE_EXTI);
STM_PBInit(BUTTON_UP, BUTTON_MODE_EXTI);

//Инициализация индикаторов
STM_LEDInit(LED1); STM_LEDInit(LED2); STM_LEDInit(LED3); STM_LEDInit(LED4);

//Инициализация ЦАП
DACInit(16000); //Аргумент - частота дискретизации в Гц
#ifdef DMA_ENA
DACTimerInterruptConfig(ENABLE); //Разрешение прерываний от таймера
#else
DMA_DAC_Initialize(); // (в режиме DMA - запрещено)
//Если задан режим DMA, его инициализация
#endif

//Инициализация кодека
SoundCodecConfig(I2S_AudioFreq_11k); //Аргумент - частота дискретизации в Гц
DMA_CODEC_Initialize(); //Функция задания режима DMA для кодека

```

```

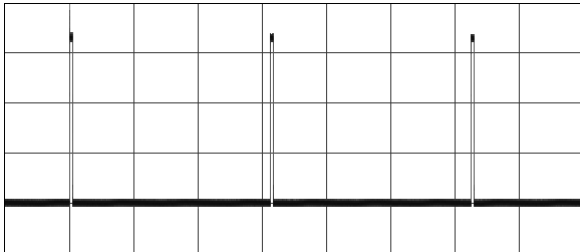
//Программная генерация запроса прерывания для выбора начального типа сигнала
EXTI_GenerateSWInterrupt(EXTI_Line11);

//Цикл в основной программе
while (1)
{
    __WFI(); //Режим пониженного энергопотребления ("сон")
}

//-----
// ОБРАБОТЧИК ПРЕРЫВАНИЯ ОТ ТАЙМЕРА ТАКТИРОВАНИЯ ЦАП (в режиме DMA не используется)
// Код для передачи в ЦАП - смещенный, выравненный влево.
#ifndef DMA_ENA
void TIMD_IRQHandler(void)
{
    uint16_t DACData;
    STM_LEDOOn(LED4); //Отладочная метка (вывод "1")
    if (TIM_GetITStatus(TIMD, TIM_IT_Update) != RESET) //Проверка источника прерывания
    {
        TIM_ClearITPendingBit(TIMD, TIM_IT_Update); //Сброс флага запроса прерывания
        switch (signal_type) //Ветвление по типу сигнала
        {
            case 0: DACData = Signal0[index_dac++]; break; //Загрузка отсчета сигнала из таблицы
            case 1: DACData = Signal1[index_dac++]; break; // с автоинкрементом индекса отсчета
            case 2: DACData = Signal2[index_dac++]; break;
        }
        index_dac &= 0xF; //Приведение индекса к диапазону 0000...1111
        DAC_SetChannel1Data(DAC_Align_12b_L, DACData); //Передача данных в ЦАП
    }
    STM_LEDOff(LED4); //Отладочная метка (вывод "0")
}
#endif

// ОСЦИЛЛОГРАММА ОТЛАДОЧНОГО СИГНАЛА-МЕТКИ (длительность импульса 0.9 мкс, период 62.5 мкс)

```



```

//-----
// ОБРАБОТЧИК ПРЕРЫВАНИЯ ОТ КОДЕКА
// Выборка отсчета из текущей таблицы, преобразование в дополнительный код,
// загрузка в оба канала, инкремент индекса отсчета с переходом на начало.
void Sample_Handler(void)
{
    DataChannel1 = DataChannel2 = Signal[index_codec++] ^ 0x8000;
    index_codec &= 0xF;
}

//-----
// ОБРАБОТЧИК ПРЕРЫВАНИЙ ОТ ВНЕШНИХ ЛИНИЙ EXTI_Line5...EXTI_Line9
// В нашем случае это линия 6 - разряд порта PG6 (джойстик "вправо"),
// линия 7 - разряд порта PG7 (джойстик "вверх").
// Константа EXTI_LineX является маской для соответствующего разряда X.
void EXTI9_5_IRQHandler(void)
{
    if (EXTI_GetITStatus(EXTI_Line6) != RESET) //Проверка источника прерывания (номера линии)
    {
        EXTI_ClearITPendingBit(EXTI_Line6); //Сброс флага запроса прерывания
        signal_type = 2; //Установка номера нового типа сигнала
        Signal = Signal2; //Установка указателя на отсчеты нового сигнала
        STM_LEDOOn(LED3); //Включение индикатора выбранного типа сигнала
        STM_LEDOff(LED1); STM_LEDOff(LED2); //Отключение других индикаторов
    }

    if (EXTI_GetITStatus(EXTI_Line7) != RESET) //Проверка источника прерывания (номера линии)
    {
        EXTI_ClearITPendingBit(EXTI_Line7); //Сброс флага запроса прерывания
        signal_type = 1; //Установка номера нового типа сигнала
    }
}

```

```

    Signal = Signal1;                                //Установка указателя на отсчеты нового сигнала
    STM_LEDOOn(LED2);                                //Включение индикатора выбранного типа сигнала
    STM_LEDOff(LED1); STM_LEDOff(LED3);              //Отключение других индикаторов
}
}

//-----
// ОБРАБОТЧИК ПЕРЕРЫВАНИЙ ОТ ВНЕШНИХ ЛИНИЙ EXTI_Line10...EXTI_Line15
// В нашем случае это линия 11 - разряд порта PG11 (джойстик "влево")
void EXTI15_10_IRQHandler(void)
{
    if (EXTI_GetITStatus(EXTI_Line11) != RESET)      //Проверка источника прерывания (номера линии)
    {
        EXTI_ClearITPendingBit(EXTI_Line11);        //Сброс флага запроса прерывания
        signal_type = 0;                            //Установка номера нового типа сигнала
        Signal = Signal0;                            //Установка указателя на отсчеты нового сигнала
        STM_LEDOOn(LED1);                            //Включение индикатора выбранного типа сигнал
        STM_LEDOff(LED2); STM_LEDOff(LED3);          //Отключение других индикаторов
    }
}

//-----
// ОБРАБОТЧИК ПЕРЕРЫВАНИЯ ОТ ВНЕШНЕЙ ЛИНИИ EXTI_Line0
// К данной линии - разряду порта PA0 - подключена кнопка WAKEUP
void EXTI0_IRQHandler(void)
{
    NVIC_SystemReset();                             //Сброс процессора - завершение программы
}

//-----
// ИНИЦИАЛИЗАЦИЯ РЕЖИМА ПРЯМОГО ДОСТУПА К ПАМЯТИ (DMA) ДЛЯ ЦАП
// Обслуживание ЦАП осуществляет контроллер DMA1 с каналами-потоками:
//   DMA1_Stream5 (для DAC1),
//   DMA1_Stream6 (для DAC2).
// Для старта передачи данных необходим запрос по каналу-запросу с номером DMA_Channel_7.
// Для прямой загрузки данных в DAC1 можно использовать регистры (см. stm32f4xx.h):
//   DHR12R1 - 12-битный код с выравниванием вправо
//   DHR12L1 - 12-битный код с выравниванием влево
//   DHR8R1  - 8-битный код с выравниванием вправо
//
void DMA_DAC_Initialize(void)
{
    DMA_InitTypeDef DMA_InitStructure;              //Структура конфигурации канала DMA

    RCC_AHB1PeriphClockCmd(RCC_AHB1Periph_DMA1, ENABLE); //Разреш. тактирования контроллера DMA
    DMA_DeInit(DMA1_Stream5);                       //Начальная установка (сброс) канала
    DMA_InitStructure.DMA_Channel = DMA_Channel_7;  //Номер источника запроса для канала
    DMA_InitStructure.DMA_PeripheralBaseAddr = (uint32_t)&DAC->DHR12L1; //Адрес периф.устройства (регистра ЦАП)
    DMA_InitStructure.DMA_Memory0BaseAddr = (uint32_t)&Signal2; //Адрес буф. памяти (отсчетов сигнала)
    DMA_InitStructure.DMA_DIR = DMA_DIR_MemoryToPeripheral; //Направл. перед.: память->периферия
    DMA_InitStructure.DMA_BufferSize = sizeof(Signal2) / 2; //Число транзакций для передачи буфера
    DMA_InitStructure.DMA_PeripheralInc = DMA_PeripheralInc_Disable; //Автоувеличение адреса периферии: нет
    DMA_InitStructure.DMA_MemoryInc = DMA_MemoryInc_Enable; //Автоувеличение адреса памяти: да
    DMA_InitStructure.DMA_PeripheralDataSize = DMA_PeripheralDataSize_HalfWord; //Размер транзакции для памяти:
    // полуслово
    DMA_InitStructure.DMA_MemoryDataSize = DMA_PeripheralDataSize_HalfWord; //Размер транзакции для перифе-
    // рии: полуслово
    DMA_InitStructure.DMA_Mode = DMA_Mode_Circular; //Режим передачи буфера: циклический
    DMA_InitStructure.DMA_Priority = DMA_Priority_High; //Уровень приоритета: высокий
    DMA_InitStructure.DMA_FIFOMode = DMA_FIFOMode_Disable; //Использование промежут. FIFO: нет
    DMA_InitStructure.DMA_FIFOThreshold = DMA_FIFOThreshold_HalfFull; //Порог для FIFO (здесь не использ.)
    DMA_InitStructure.DMA_MemoryBurst = DMA_MemoryBurst_Single; //Размер пакета для памяти: одиночный
    DMA_InitStructure.DMA_PeripheralBurst = DMA_PeripheralBurst_Single; //Размер пакета для периферии: одиноч.
    DMA_Init(DMA1_Stream5, &DMA_InitStructure); //Функция конфигурирования

    DMA_Cmd(DMA1_Stream5, ENABLE);                  //Разрешение работы канала DMA

    DAC_DMACmd(DAC_Channel_1, ENABLE);               //Разрешение запроса на DMA от ЦАП 1
}

//-----
// ИНИЦИАЛИЗАЦИЯ РЕЖИМА ПРЯМОГО ДОСТУПА К ПАМЯТИ ДЛЯ КОДЕКА
// Обслуживание интерфейса SPI3/I2S3, используемого для связи с кодеком,
// осуществляет контроллер DMA1 с каналами-потоками DMA1_Stream5, DMA1_Stream7
// и каналом-запросом DMA_Channel_0 (запрос формируется при пустом буфере передачи).
// Регистр для загрузки данных - SPI3->DR.

```

```
// Функция разрешения запроса: SPI_I2S_DMAcmd(SPI3, SPI_I2S_DMAREq_Tx, ENABLE);
// Необходимо также обеспечить:
// - запрет прерываний с частотой дискретизации;
// - отдельные коды для каждого канала кодека;
// - коды в дополнительном формате (со знаком).
void DMA_CODEC_Initialize(void)
{
}

//-----
```

Проект ..\Lab3_discret\MDK-ARM\discret.uvproj

3. Дискретизация сигналов с помощью АЦП

Исходный модуль ..\Lab3_discret\main.c

```
//-----
//
// УЧЕБНЫЙ ПРОЕКТ
// Дискретизация сигналов
// Copyright (C) 2015 МИРЭА
//
// В проекте демонстрируются:
// - генерация тестового гармонического сигнала;
// - оцифровка этого сигнала посредством АЦП;
// - вывод сигнала через ЦАП аудиокодека.
// Физическая связь выхода генератора и входа АЦП осуществлена
// благодаря двойной альтернативной функции разряда порта PA4.
//
//-----

// Заголовочные файлы
#include "stm32_p407.h" //Файл конфигурации отладочной платы STM32-P407
#include "arm_math.h" //Определения и функции библиотеки CMSIS DSP Library
#include "adc.h" //Функции для работы с АЦП
#include "dac.h" //Константы и функции для работы с ЦАП
#include "codec.h" //Функции для работы с кодеком

//-----
// ОБЪЯВЛЕНИЯ ДАННЫХ
int16_t DataADC1, DataADC2; //Текущие отсчеты сигнала с АЦП
extern int16_t DataChannel1, //Данные для кодека в дополнительном коде
DataChannel2; // (определены в модуле codec.c)
TIM_TimeBaseInitTypeDef TimeBase; //Структура конфигурации базового таймера
NVIC_InitTypeDef NVICstruct; //Структура конфигурации прерываний

//-----
// ГЛАВНАЯ ФУНКЦИЯ
int main()
{
//Счетчик для реализации периода изменения тестовой индикации
volatile uint32_t i = 0;

//Задание 4-х уровней групповых приоритетов и 4-х уровней приоритетов в каждой группе
NVIC_PriorityGroupConfig(NVIC_PriorityGroup_2);

//Инициализация тестовых индикаторов
STM_LEDInit(LED1); STM_LEDInit(LED2); STM_LEDInit(LED3); STM_LEDInit(LED4);

//Инициализация кнопок
STM_PBInit(BUTTON_TAMPER, BUTTON_MODE_GPIO);
STM_PBInit(BUTTON_WAKEUP, BUTTON_MODE_GPIO);

//Инициализация ЦАП - как генератора входного сигнала для АЦП
DACInit(192000);
DACTimerInterruptConfig(ENABLE);

//Инициализация генератора частоты дискретизации АЦП; для этой цели
```



```

// задействован таймер 2 с собственной частотой тактирования 84 МГц
RCC_APB1PeriphClockCmd(RCC_APB1Periph_TIM2, ENABLE);
TimeBase.TIM_Period = 840000u/44100-1;
TimeBase.TIM_Prescaler = 100-1;
TimeBase.TIM_ClockDivision = 0;
TimeBase.TIM_CounterMode = TIM_CounterMode_Up;
TIM_TimeBaseInit(TIM2, &TimeBase);
TIM_SelectOutputTrigger(TIM2, TIM_TRGOSource_Update);
TIM_Cmd(TIM2, ENABLE);

//Инициализация двухканального АЦП с запуском от события перезагрузки таймера 2
ADC_Initialize(ADC1, ADC_Channel_4, ADC_ExternalTrigConv_T2_TRGO);
ADC_Initialize(ADC2, ADC_Channel_4, ADC_ExternalTrigConv_T2_TRGO);

//Конфигурирование прерываний от АЦП (по событию окончания преобразования)
ADC_ITConfig(ADC1, ADC_IT_EOC, ENABLE);
NVICstruct.NVIC_IRQChannel = ADC_IRQn;
NVICstruct.NVIC_IRQChannelPreemptionPriority = 2;
NVICstruct.NVIC_IRQChannelSubPriority = 0;
NVICstruct.NVIC_IRQChannelCmd = ENABLE;
NVIC_Init(&NVICstruct);

//Инициализация кодека как устройства вывода сигнала
SoundCodecConfig(I2S_AudioFreq_192k);

//Основной цикл
while (1)
{
    if (i++ == 0x800) STM_LEDOff(LED1);           //Тестовое управление индикатором
    if (i == 0xB0000) STM_LEDOn(LED1), i = 0;

    if (STM_PBGetState(BUTTON_WAKEUP)) NVIC_SystemReset(); //Проверка на завершение
}

//-----
// ОБРАБОТЧИК ПЕРЕРЫВАНИЯ ОТ ТАЙМЕРА ТАКТИРОВАНИЯ ЦАП
void TIMD_IRQHandler(void)
{
    static int16_t arg_sin;
    STM_LEDOn(LED2);
    TIM_ClearITPendingBit(TIMD, TIM_IT_Update);
    DAC_SetChannel1Data(DAC_Align_12b_L, arm_sin_q15(arg_sin)>>1^0x8000);
    arg_sin = (arg_sin + 0x200) & 0x7FFF;
    STM_LEDOff(LED2);
}

//-----
// ОБРАБОТЧИК ПЕРЕРЫВАНИЯ ОТ АЦП
void ADC_IRQHandler(void)
{
    STM_LEDOn(LED4);
    ADC_ClearFlag(ADC1, ADC_FLAG_EOC);
    DataADC1 = ADC_GetConversionValue(ADC1);
    DataChannel1 = DataADC1 ^ 0x8000;
    DataADC2 = ADC_GetConversionValue(ADC2);
    DataChannel2 = DataADC2 ^ 0x8000;
    STM_LEDOff(LED4);
}

//-----
// ОБСЛУЖИВАНИЕ ПЕРЕРЫВАНИЯ ОТ КОДЕКА
// Вызывается из обработчика прерывания ..._IRQHandler(), реализованного в codec.c
void Sample_Handler(void)
{
    //Выводимые значения - в DataChannel1, DataChannel2
}

//-----

```

Сведения об авторах

Богаченков Алексей Николаевич, к.т.н., доцент кафедры радиоэлектронных систем и комплексов Института радиотехнических и телекоммуникационных систем РТУ МИРЭА.