

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ
ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«САМАРСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ
УНИВЕРСИТЕТ ИМЕНИ АКАДЕМИКА С.П. КОРОЛЕВА»
(САМАРСКИЙ УНИВЕРСИТЕТ)

РАБОТА С ПЕРИФЕРИЕЙ МИКРОКОНТРОЛЛЕРОВ С ЯДРОМ
CORTEX-M3 В СРЕДЕ μ Vision

Методические указания для выполнения лабораторной работы

Учебно-методическое пособие

Самара
Издательство Самарского университета
2020

Рецензент канд-т техн. наук М.П. Калаев

Кудрявцев И.А. Работа с периферией микроконтроллеров с ядром Cortex-M3 в среде μ Vision. Методические указания для выполнения лабораторной работы: Учебно-методическое пособие / И.А. Кудрявцев, Д.В. Корнилин, О.О. Мякинин. – Самара: Издательство Самарского университета, 2020. 25 с.

В методических указаниях рассмотрены вопросы разработки программ для микроконтроллеров с ядром Cortex-M3 с использованием периферии. Продемонстрированы разработка и основные приемы отладки программ для данного типа микропроцессоров в среде программирования Keil μ Vision 5. Приведены порядок выполнения лабораторных работ и требования к отчету.

Методические указания предназначены для студентов, обучающихся по направлениям 12.03.04 «Биотехнические системы и технологии», 12.03.05 «Лазерная техника и лазерные технологии», 03.03.01 «Прикладная математика и физика», выполняющих лабораторные работы по дисциплинам «Цифровые устройства и микропроцессоры», «Микропроцессорные средства и системы» и др. Подготовлены на кафедре лазерных и биотехнических систем.

УДК 004.384

© Самарский университет, 2020

ОГЛАВЛЕНИЕ

Введение	4
1 Использование АЦП	4
2 Использование ЖК-дисплея.....	10
3 Использование таймера	11
4 Использование прямого доступа к памяти	12
5 Задания для самостоятельного выполнения.....	16
Список использованных источников	17
Приложение А Отладочный модуль	18
Приложение Б Исходный код программы, демонстрирующий возможности DMA	19
Приложение В Массив выборок синусоидального сигнала	24

Введение

В настоящее время микроконтроллеры с ядром CORTEX-M3 являются одними из наиболее распространенных в данном сегменте. Большинство ведущих производителей микроконтроллеров имеют в своей продуктовой линейке микросхемы на базе ядра CORTEX-M3. В методических указаниях рассматривается микроконтроллер K1986BE92QI производства российской компании Миландр.

Микроконтроллеры K1986BE92QI [1], помимо ядра CORTEX-M3 [2], способного работать на частоте до 80МГц, обладают значительным объемом FLASH-памяти на кристалле, и большой набор периферийных устройств, включая АЦП, ЦАП, порты ввода-вывода, таймеры и модули различных интерфейсов. Микроконтроллеры снабжены системой внутрисхемного программирования на базе интерфейса JTAG и специальным загрузчиком, позволяющим загружать код программы через асинхронный приемопередатчик. Для разработки ПО используется широко известная среда Keil μ Vision, использование которой рассматривается в данных методических указаниях.

Большая часть работы посвящена изучению особенностей периферийных устройств микроконтроллера посредством отладочной платы, предоставленной компанией Миландр. Особое внимание уделено особенностям отладки программного обеспечения.

Методические указания позволяют студентам изучить основы разработки и отладки программного обеспечения микроконтроллеров с ядром CORTEX-M3. Указания не претендуют на полноту описания особенностей ядра, равно как и микроконтроллеров K1986BE92QI или среды разработки, приводятся лишь краткие пояснения, необходимые для понимания приведенных фрагментов кода. Методические указания содержат также список дополнительных заданий для самостоятельных экспериментов с микроконтроллером и вопросы для самоконтроля. Перед выполнением данного лабораторного практикума рекомендуется ознакомление с техникой работы со средой программирования, описанной в [3].

1 Использование АЦП

Микроконтроллер K1986BE92QI имеет два 12-разрядных АЦП, которые могут обеспечивать оцифровку сигналов с частотой дискретизации до

500kSps. Подробную информацию об АЦП и его конфигурации вы можете найти в [4, 5]. В первом эксперименте изучается режим с обработкой напряжения от внешнего источника. (Потенциометр встроен в отладочную плату). Схема данной отладочной платы приведена в приложении А.

Создайте файл со следующим кодом:

```
#include <MDR32F9Qx_port.h>
#include <MDR32F9Qx_rst_clk.h>
#include <MDR32F9Qx_adc.h>

uint16_t DU;

int main()
{
    RST_CLK_PCLKcmd (RST_CLK_PCLK_ADC | RST_CLK_PCLK_PORTD, ENABLE);

    PORT_InitTypeDef Nastroyka;
    Nastroyka.PORT_Pin = PORT_Pin_7;
    Nastroyka.PORT_OE = PORT_OE_IN;
    Nastroyka.PORT_MODE = PORT_MODE_ANALOG;
    PORT_Init (MDR_PORTD, &Nastroyka);
    ADC_InitTypeDef sADC;
    ADCx_InitTypeDef sADCx;

    ADC_DeInit();
    ADC_StructInit(&sADC);
    sADC.ADC_SynchronousMode= ADC_SyncMode_Independent;
    ADCx_StructInit (&sADCx);
    sADCx.ADC_ClockSource= ADC_CLOCK_SOURCE_CPU;
    sADCx.ADC_SamplingMode= ADC_SAMPLING_MODE_SINGLE_CONV;
    sADCx.ADC_ChannelNumber= ADC_CH_ADC7;
    sADCx.ADC_Channels= 0;
    sADCx.ADC_VRefSource= ADC_VREF_SOURCE_INTERNAL;
    sADCx.ADC_IntVRefSource= ADC_INT_VREF_SOURCE_INEXACT;
    sADCx.ADC_Prescaler= ADC_CLK_div_None;
    ADC1_Init (&sADCx);
    ADC1_Cmd (ENABLE);
    while (1)
    {
        ADC1_Start();
```

```

while ((ADC1_GetStatus() & ADC_STATUS_FLG_REG_EOCIF) == 0);
DU = ADC1_GetResult() & 0x0FFF;
printf("U=0x%03x\r\n",DU);      // for simulation
}
}

```

Функция `printf` позволяет отображать необходимые переменные в удобочитаемом формате в соответствии со стандартами языка Си. Соответствующие кодовые блоки, поддерживающие операции ввода/вывода, должны быть включены в код (обратите внимание, что они занимают дополнительное пространство памяти). Чтобы добавить их, нажмите и отметьте блок `STDOUT`, как показано на рисунке 1.

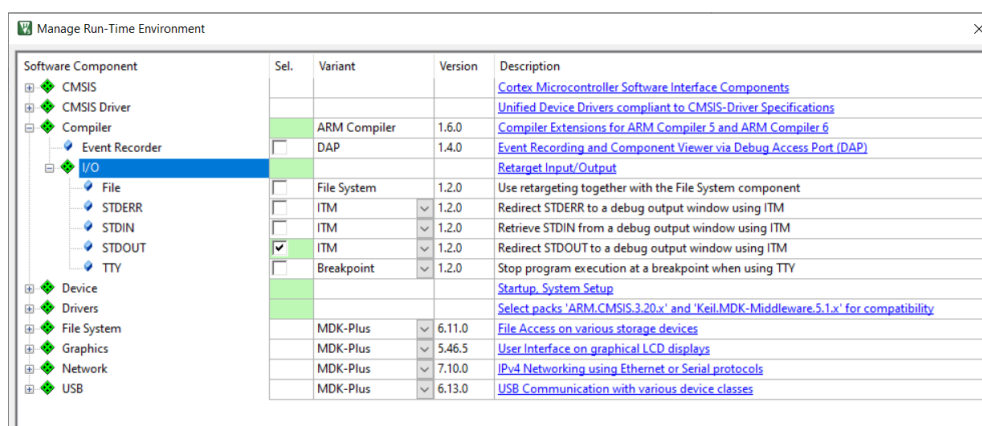


Рис. 1. Конфигурация подсистемы вывода текста

Скомпилируйте и запустите проект в режиме симуляции. Так как возможности моделирования периферийных устройств ограничены, будут необходимы некоторые дополнительные действия. Как видно из приведенного выше исходного кода, программа ожидает завершения преобразования. Для имитации завершения конвертации необходимо установить флаг `Flg_REG_EOCIF` вручную. Используйте меню `Peripherals / System Viewer / MDR_ADC` и выберите этот блок, как показано на рисунке 2. Введите некоторое значение в окно `RESULT` (например, 6, как показано на рисунке 2).

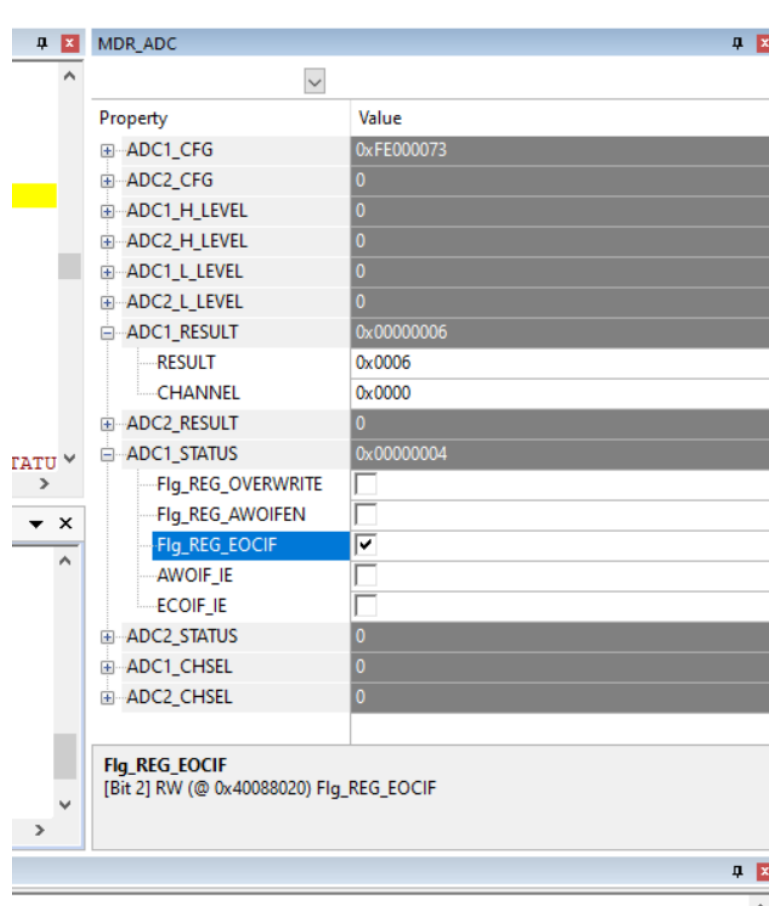


Рис. 2. Имитация завершения преобразования сигнала

Откройте вкладку Debug (printf) в правой нижней части экрана и посмотрите результат преобразования. Если эта вкладка не появилась, откройте ее, используя меню View / Serial Windows / Debug (printf) Viewer. Убедитесь, что наша программа показывает код, который мы вручную вставили в регистр результата АЦП.

Для моделирования сигналов произвольной формы можно использовать механизм файлов сигналов. Создайте с помощью текстового редактора (и сохраните в папке проекта) файл Sin.ini со следующим содержимым:

```
SIGNAL void Sine (void) {
    float volts;           // peak-to-peak voltage
    float frequency;       // output frequency in Hz
    float offset;          // voltage offset
    float val;

    volts      = 1.4;
    offset     = 1.6;
    frequency  = 100;
```

```

while(1)
{
    val = __sin(frequency*(((float) STATES) /CLOCK) *2*3.1415926);
    _WDWORD(0x40088018, (int) (((val*volts)+offset)*4095/3.3));
    twatch (1000);
}
}

```

Обратите внимание, что функция twatch () всегда должна присутствовать в файлах сигналов. Эта функция генерирует паузу, в которой в качестве аргумента используется количество машинных циклов. Значение паузы не влияет на частоту сигнала, потому что вычисление синуса основано на текущем количестве тактов процессора.

Используя меню Debug / Function Editor (Open Ini File), выберите созданный файл. При необходимости вы можете редактировать этот файл в открывшемся окне. Для активации файла нажмите Compile. В случае ошибок информация о них будет отображаться в поле Compile Errors. Если ошибок нет, вы увидите сообщение «Compilation complete. No errors». Содержимое файла отображается в окне Command. Можете закрыть окно.

Чтобы запустить созданный сценарий, введите Sine () в командной строке и нажмите Enter, как показано на рисунке 3. Чтобы остановить сценарий, введите KILL FUNC Sine в командной строке.

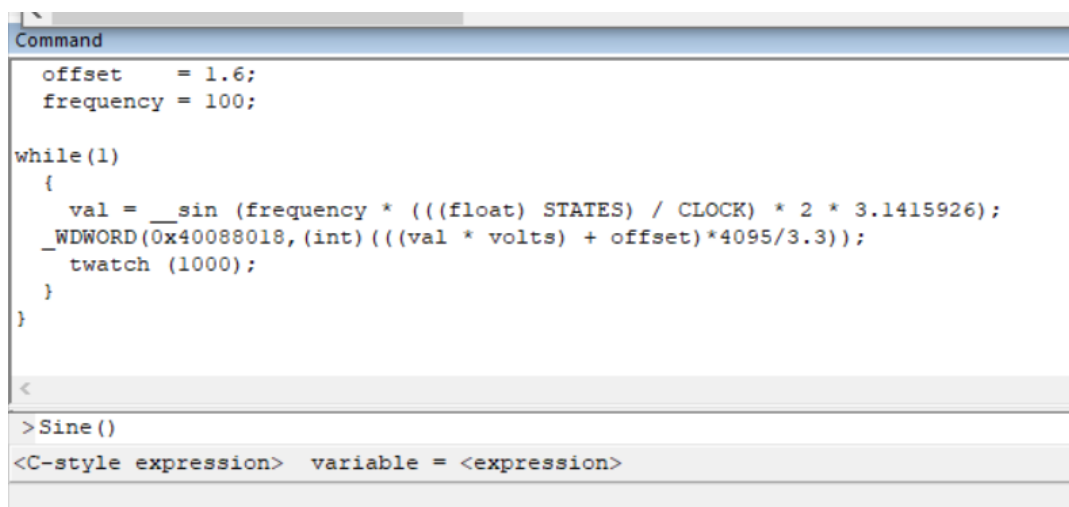


Рис. 3. Запуск скрипта в командной строке

Наблюдать за процессом измерения можно с помощью инструмента Logic Analyzer, который можно активировать, выбрав View / Analysis Windows /

Logic Analyzer. В открывшемся окне нажмите Setup и введите имя желаемой переменной, как показано на рисунке 4.

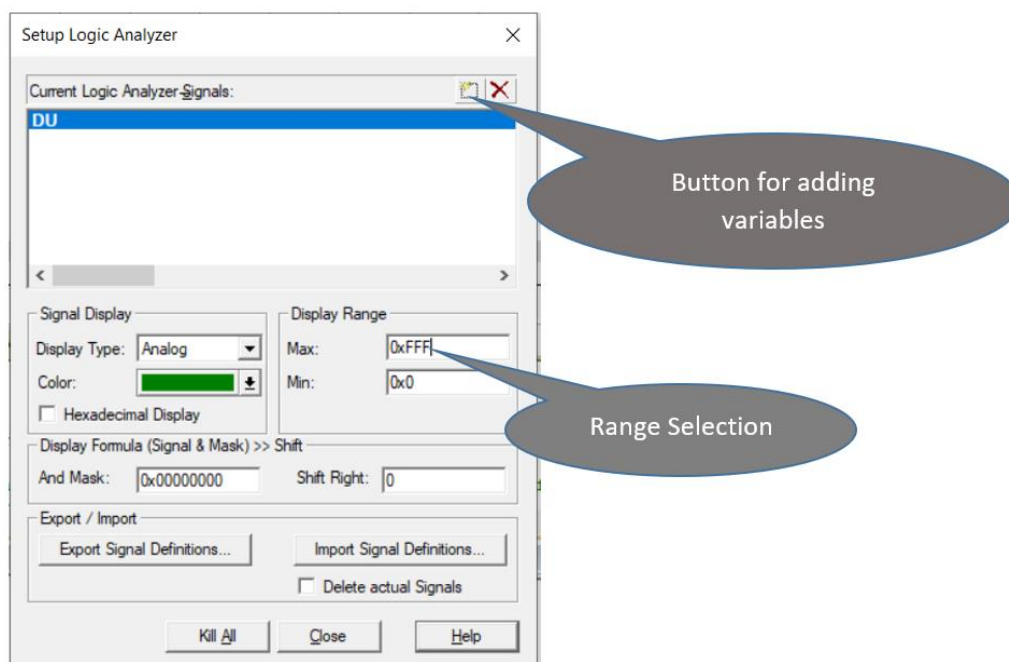


Рис. 4. Окно визуализации

После запуска программы можно наблюдать синусоидальный сигнал, как показано на рисунке 5. При необходимости можно настроить параметры дисплея по своему усмотрению.

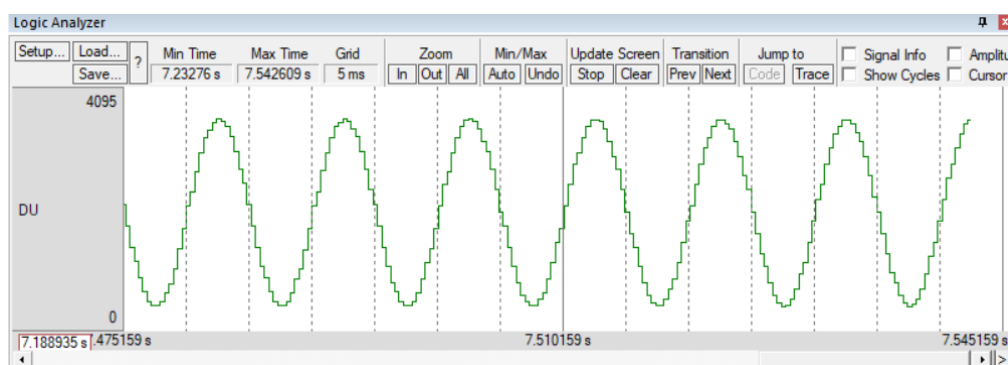


Рис. 5. Визуализация с использованием инструмента Logic Analyzer

Этот подход имеет очевидный недостаток – частота дискретизации зависит от задач, выполняемых ядром процессора, и выбранной тактовой частоты, что может быть неприемлемо для задач анализа сигналов.

Стабильная частота дискретизации, независимая от ядра процессора, может быть получена с помощью таймера. Альтернативным вариантом является

установка АЦП с бесконечным преобразованием и применение прямого доступа к памяти (DMA) для сохранения данных в памяти или вывода результатов через некоторый интерфейс.

К сожалению, в режиме аппаратной отладки использование визуализации данных адаптера в логическом анализаторе и окне текстового вывода не поддерживается. Поэтому для дальнейших экспериментов необходимо изучить ЖК-дисплей (LCD), размещенный на плате оценки.

2 Использование ЖК-дисплея

Краткие теоретические сведения о ЖК-дисплее вы можете найти в [6]. Также полезно ознакомиться с документацией для ЖК-дисплея MT-12864J v.1 российской компании «МЭЛТ» [7].

Основные функции, поддерживающие вывод информации на ЖК-дисплей, вы можете найти в папке Generic. Спросите преподавателя о местонахождении этой папки. В окне конфигурации компилятора необходимо указать ссылку на эту папку, как показано на рисунке 6. Добавьте файл lcd.c в проект; он находится в той же папке Generic.

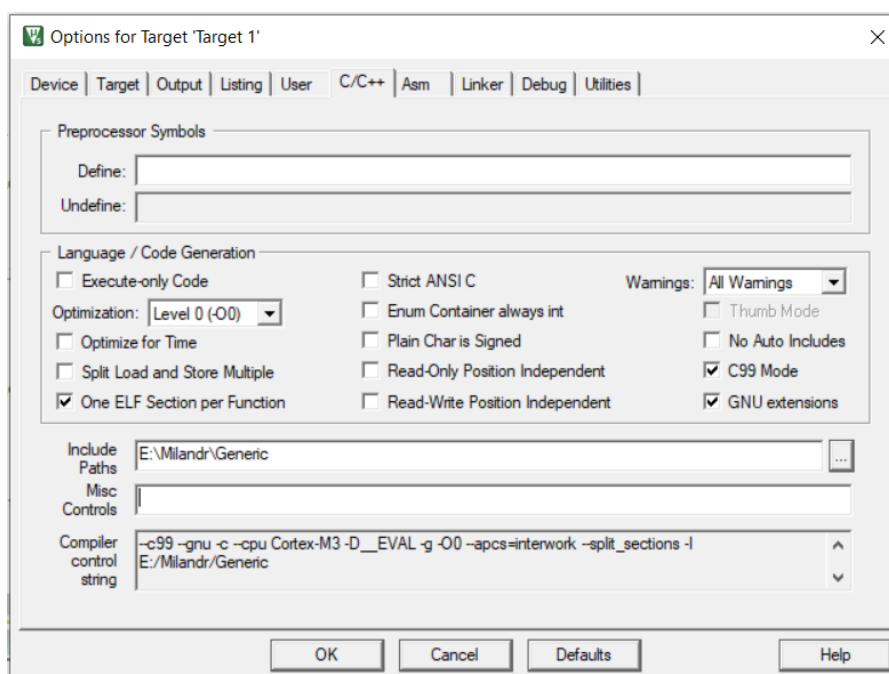


Рис. 6. Ссылка на дополнительные файлы

Инициализация модуля LCD выполняется путем вызова функции LCD_Init(), она должна быть вызвана один раз при запуске программы. Вы-

вод текстовых данных осуществляется с помощью функции LCD_PutString (Buffer, n), где n – номер соответствующей ЖК-страницы.

3 Использование таймера

В этом эксперименте мы будем использовать таймер 1. Для его настройки добавим функцию TimerInit (); в блок инициализации программы и вставим в нашу программу следующий код:

```
void TimerInit()
{
    // Indication the type of structure and structure name
    TIMER_CntInitTypeDef TIM1Init;
    // Enabling of clocking
    RST_CLK_PCLKcmd(RST_CLK_PCLK_TIMER1, ENABLE);
    // Filling the structure with default values
    TIMER_CntStructInit(&TIM1Init);
    // Setup of clock frequency divider
    TIMER_BRGInit (MDR_TIMER1, TIMER_HCLKdiv1);
    // Setup of clock frequency prescaler
    TIM1Init.TIMER_Prescaler = 8000;
    // Setup the timer period
    TIM1Init.TIMER_Period = 200;
    // Initialization of a timer port by a declared structure
    TIMER_CntInit (MDR_TIMER1, &TIM1Init);
    // Run interrupts
    NVIC_EnableIRQ (Timer1_IRQn);
    // Setup Interrupt Priority
    NVIC_SetPriority (Timer1_IRQn, 0);
    // Enabling interrupt when value TIMER1 is zero
    TIMER_ITConfig(MDR_TIMER1, TIMER_STATUS_CNT_ZERO, ENABLE);
    // Run timer
    TIMER_Cmd(MDR_TIMER1, ENABLE);
}
```

Эта функция должна быть объявлена в декларативной части программы путем добавления строки «void TimerInit (void);»

Также необходимо добавить обработчик прерывания по таймеру:

```
// Timer Interrupt handler
```

```

void Timer1_IRQHandler()
{
    if (TIMER_GetITStatus(MDR_TIMER1, TIMER_STATUS_CNT_ZERO))
        TIMER_ClearITPendingBit(MDR_TIMER1, TIMER_STATUS_CNT_ZERO);
    ADC1_Start();
    while((ADC1_GetStatus() & ADC_STATUS_FLG_REG_EOCIF) == 0);
    DU = ADC1_GetResult() & 0xFFFF;
}

```

Этот обработчик использует АЦП в одноканальном режиме, но теперь преобразование начинается в момент запуска выходного сигнала таймера. Чтобы отобразить результаты на ЖК-дисплее, добавьте следующий код в основную программу:

```

while (1)
{
    sprintf(Buffer, "U=0x%03X", DU);
    LCD_PutString(Buffer, 4);
}

```

Скомпилируйте проект. Перед загрузкой кода на отладочную плату убедитесь, что перемычка XP2 находится в положении TRIM. Осторожно вращая регулировочный винт подстроечного резистора, проверьте измерения напряжения.

Задание: измените формат выходных данных измерения для отображения напряжения в вольтах.

4 Использование прямого доступа к памяти

Использование прямого доступа к памяти (DMA) требует настройки контроллера DMA, перемещения данных без использования ядра процессора. Для этого эксперимента мы будем использовать передачу данных из массива памяти в ЦАП. Подробную информацию о работе DMA и ЦАП вы можете найти в [4, 5].

Чтобы продемонстрировать возможности DMA, мы создадим синусоидальную волну, используя ЦАП. Форма сигнала будет иметь параметры, как на рисунке 4. Соедините выход ЦАП с входом АЦП для проверки результа-

тов. АЦП преобразует сигнал и записывает данные в файл памяти для дальнейшего анализа.

Исходный код программы размещен в приложении Б. Скопируйте его в проект и скомпилируйте. Код состоит из большой части инициализации, где настраиваются таймер, ЦАП, АЦП и обработчик прерываний. После этого основная программа работает в бесконечном цикле (`while (1)`). Можно было бы использовать режим низкого энергопотребления, но в этом режиме функции аппаратной отладки не работают, и теряется возможность контролировать работу программы. В автономном режиме этот оператор бесконечного цикла может быть заменен функцией `_wfi ()`, которая позволяет снизить энергопотребление.

К сожалению, доступный аппаратный отладчик и среда Keil μ Vision имеют некоторые функциональные ограничения. В режиме трассировки использование инструментов визуализации и вывода данных в окне программы, описанном выше, недоступно, некоторые команды отладчика не работают, более того, адаптер не позволяет провести модификацию Flash-памяти, поэтому необходимо использовать специальные приемы. В первой части нашего эксперимента мы будем отлаживать загрузку данных в ОЗУ в режиме симуляции, как описано в [3]. Мы создадим образцы синусоидального сигнала, сгенерированные по специальному сценарию.

Создайте файл-скрипт со следующим кодом:

```
SIGNAL void Dac (void)
{
    float val,volts,offset;
    int Sig;
    volts=1.4;
    offset=1.6;
    Sig=0;
    while (Sig<256)
    {
        val = __sin ((float) (Sig * 3.1415926/128.0))*volts +
offset;
        _WWORD(&Sin_Arr[Sig++],(int16_t)(val*4095/3.3));
    }
    twatch (10);
}
```

Этот скрипт заполняет массив Sin_Arr в памяти 256 выборками синусоидального сигнала с параметрами, как на рисунке 2. Запустите скрипт и убедитесь, что массив памяти заполнен соответствующими выборками. После этого, используя команду SAVE, сохраните дамп памяти в файл. Определите начальный адрес массива Sin_Arr (можно ввести имя массива в командной строке в качестве команды) и выполните следующую команду:

```
SAVE < file name with extension> start address of array, end  
address of array
```

Исследуйте работу программы в режиме имитации прерываний и АЦП.

Теперь давайте перейдем ко второй части нашего эксперимента с аппаратной отладкой. Важно, чтобы расположение данных было таким же, как и в первой части эксперимента, то есть конфигурация свойств проекта не должна изменяться (окно с настройкой тактовой частоты и объема памяти).

Перед загрузкой программы на оценочную плату необходимо установить переключки XP2 и XP4 в положение EXT_CON. Используя провод, соедините (спросите преподавателя) центральные части разъемов XS1 (ADC) и XS8 (DAC_OUT), как показано на рисунке 7.

Внимание! Все соединения и коммутация переключек должны выполняться перед подключением платы к адаптеру питания.



Рис. 7. Подключение ЦАП и АЦП

Включите питание и загрузите программу. Поставьте точку останова на функцию TimerInit(). Когда программа остановится, введите команду (в командной строке):

```
LOAD <name of earlier saved file with extension >
```

Эта команда должна принудительно загрузить подготовленные образцы в массив данных в ОЗУ. Проверьте наличие данных и запустите программу. Вы можете увидеть динамические изменения массива Rez в окне памяти. Следует отметить, что в режиме отладки работа АЦП иногда останавливается (когда сигнал готовности не выдается или пропускается). В результате программа блокируется в цикле `while ((ADC1_GetStatus () & ADC_STATUS_FLG_REG_EOCIF) == 0);` в обработчике прерывания по таймеру (`Tim-er2_IRQHandler ()`). Это не происходит в автономном режиме.

Остановите микроконтроллер (меню Debug/Stop) и проанализируйте данные в массиве Rez. Определите амплитуду и смещение сигнала и сравните с данными в скрипте.

Использованный прием с загрузкой данных из файла в оперативную память возможен только в режиме отладки, однако он позволяет использовать мощный механизм скриптов. Для выполнения описанной программы в автономном режиме необходимо сохранить массив данных в энергонезависимой памяти или скопировать данные в массив, помещенный в ОЗУ во время инициализации. В этом случае можно достичь лучшей производительности. Можно записывать данные в энергонезависимую память с использованием языка Си, объявив массив с ключевым словом `const` и инициализировав его.

```
Const Array[]={N1,N2,N3...};
```

Компилятор должен автоматически поместить этот массив в энергонезависимую память. В приложении В представлен массив данных, идентичных сгенерированным скриптом. Иногда имеет смысл поместить некоторые данные по фиксированному адресу. Это может быть сделано с помощью следующего приема:

```
int16_t Arr[256] __attribute__((section(".ARM.__at_0x08001000")));
```

5 Задания для самостоятельного выполнения

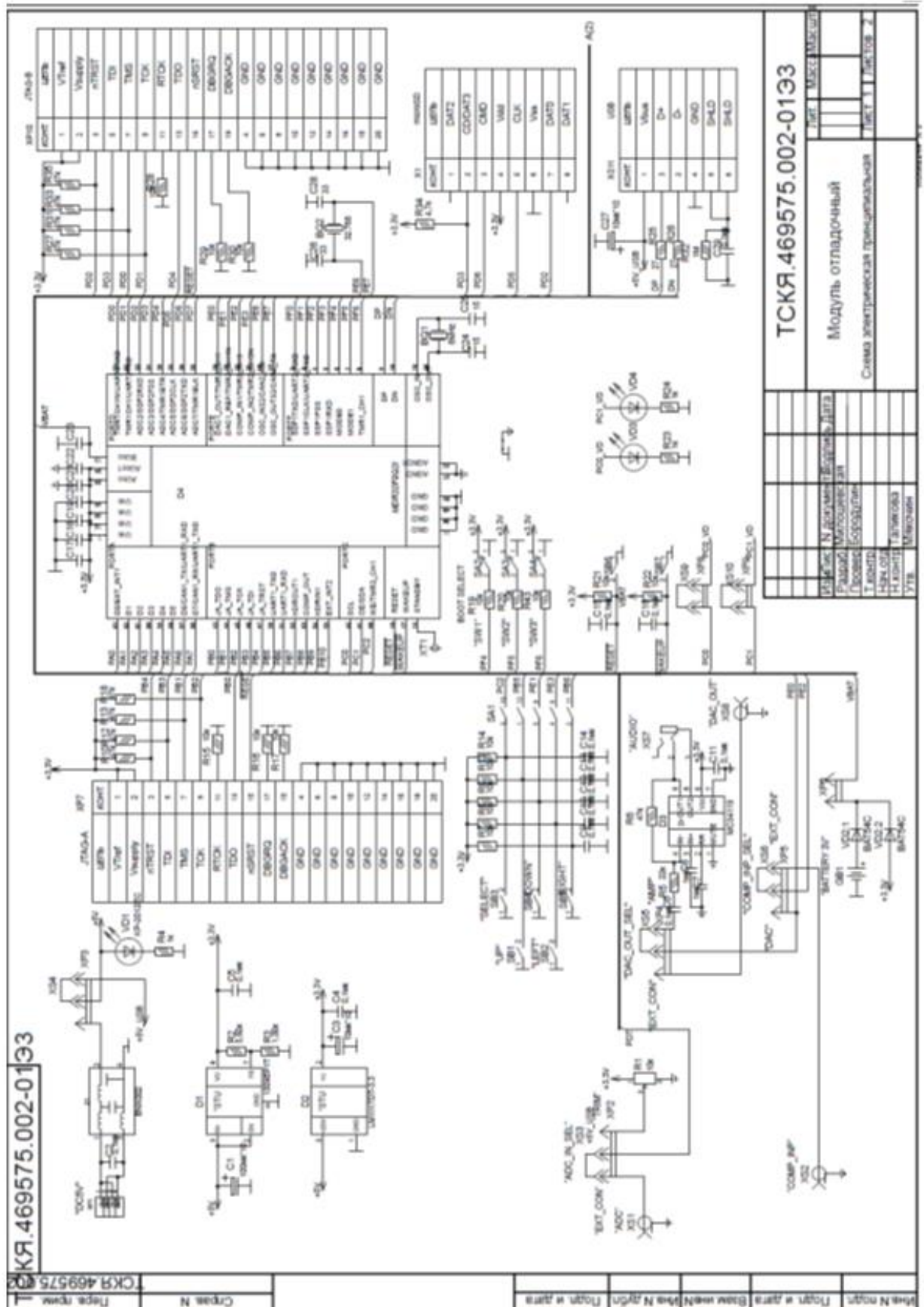
1. Чтобы сэкономить память, измените программу генерации синусоидальной волны таким образом, чтобы она использовала только четверть массива данных, указанного в приложении В.
2. Перепишите программу таким образом, чтобы использовать DMA для сохранения результата преобразования АЦП в ОЗУ (вместо прямого запуска АЦП в обработчике прерывания по таймеру).
3. Напишите программу, отображающую результаты аналого-цифрового преобразования в графическом режиме на ЖК-дисплее отладочной платы.
4. Напишите программу с использованием часов реального времени с отображением времени на ЖК-дисплее.
5. Напишите программу, с помощью которой можно нарисовать линию (цепочку точек) на ЖК-дисплее, используя кнопки на отладочной плате.
6. Напишите программу, передающую информацию (количество нажатий кнопок) через интерфейс CAN.
7. Напишите программу контроля передачи данных по интерфейсу CAN (CAN-линии) и отображения полученных данных (содержимого пакета) на ЖК-дисплее.
8. Напишите программу, считывающую идентификационную информацию микроконтроллера и отображающую ее на дисплее при нажатии кнопки SB4.
9. Напишите программу для отображения на ЖК-дисплее времени (с разрешением до 10мс), прошедшего с момента нажатия клавиши «Старт» при нажатии кнопки «Стоп».
10. Напишите программу измерения нестабильности встроенного тактового генератора с использованием реальных часов и отображением на ЖК-дисплее.

Список использованных источников

1. K1986BE92QI // Официальный сайт компании Миландр. URL: https://ic.milandr.ru/products/mikrokontrollery_i_protssory/k1986ve92qi/ (дата обращения: 15.10.2019).
2. Cortex-M3 Devices Generic User Guide // Официальный сайт ARM. URL: <http://infocenter.arm.com/help/index.jsp?topic=/com.arm.doc.dui0552a/index.html> (дата обращения: 15.10.2019).
3. Кудрявцев И.А. Разработка и отладка программ для микроконтроллеров с ядром Cortex-M3 в среде μ Vision. Методические указания для выполнения лабораторной работы: Учебно-методическое пособие / И.А. Кудрявцев, Д.В. Корнилин, О.О. Мякинин. – Самара: Издательство Самарского университета, 2020. 23 с.
4. Спецификация микросхем серии 1986BE9ху, K1986BE9ху, K1986BE9хуK, K1986BE92QI, K1986BE92QC, 1986BE91H4, K1986BE91H4, 1986BE94H4, K1986BE94H4. М.: Миландр, 2015. 521 с.
5. Благодаров А.В., Владимиров Л.Л. Программирование микроконтроллеров на основе отечественных микросхем семейства 1986BE9х разработки и производства компании «Миландр». М.: Миландр, 2016. 242 с.
6. Евдокимов А.П., Владимиров Л.Л. Программирование микроконтроллера K1986BE92QI компании «Миландр». Волгоград: ФГБОУ ВО Волгоградский ГАУ, 2018. 76 с.
7. LCD display module MT-12864J. URL: http://www.melt.com.ru/docs/MT-12864J_en.pdf (дата обращения: 15.10.2019).

Приложение А

Отладочный модуль



Приложение Б

Исходный код программы, демонстрирующий возможности DMA

```
#include <MDR32F9Qx_port.h>
#include <MDR32F9Qx_rst_clk.h>
#include <MDR32F9Qx_adc.h>
#include <MDR32F9Qx_dac.h>
#include <MDR32F9Qx_dma.h>
#include <MDR32F9Qx_timer.h>

void TimerInit(void);
void Timer2Init(void);
void DMA_Config(void);
void ADC_Config(void);

DMA_CtrlDataInitTypeDef DMA_InitStructure;
DMA_ChannelInitTypeDef DMA_Channel_InitStructure;
ADC_InitTypeDef sADC;
ADCx_InitTypeDef sADCx;

int16_t Sin_Arr[256], Rez[256];
int Index;

int main()
{
    RST_CLK_PCLKcmd (RST_CLK_PCLK_ADC | RST_CLK_PCLK_DAC |
RST_CLK_PCLK_PORTE | RST_CLK_PCLK_PORTD | RST_CLK_PCLK_DMA |
RST_CLK_PCLK_SSP1 | RST_CLK_PCLK_SSP2, ENABLE);
    PORT_InitTypeDef Nastroyka;
    // Setup of DAC output
    Nastroyka.PORT_Pin = PORT_Pin_0;
    Nastroyka.PORT_OE = PORT_OE_OUT;
    Nastroyka.PORT_MODE = PORT_MODE_ANALOG;
    PORT_Init (MDR_PORTE, &Nastroyka);
    // Setup of ADC output
    Nastroyka.PORT_Pin = PORT_Pin_7;
    Nastroyka.PORT_OE = PORT_OE_IN;
    Nastroyka.PORT_MODE = PORT_MODE_ANALOG;
    PORT_Init (MDR_PORTD, &Nastroyka);
    DMA_Config(); //DMA setup
    ADC_Config(); //ADC setup
```

```

//DAC setup
DAC_DeInit();
DAC2_Init(DAC2_AVCC);
DAC2_Cmd (ENABLE);
ADC1_Cmd (ENABLE);
Index=0;
TimerInit(); //Setup timer for work with DMA
Timer2Init(); // Setup timer 2 for work with ADC

    while(1);
}
void DMA_IRQHandler (void)
{
    // Prepare a new cycle of digital-to-analog conversion
    DMA_InitStructure.DMA_CycleSize = 256;
    DMA_Init (DMA_Channel_TIM1, &DMA_Channel_InitStructure);
}

void TimerInit()
{
    // Indication the type of structure and structure name
    TIMER_CntInitTypeDef TIM1Init;
    // Enabling of clocking
    RST_CLK_PCLKcmd(RST_CLK_PCLK_TIMER1, ENABLE);
    // Filling the structure with default values
    TIMER_CntStructInit(&TIM1Init);
    // Setup of clock divider
    TIMER_BRGInit (MDR_TIMER1, TIMER_HCLKdiv1);
    // Setup of clock frequency prescaler
    TIM1Init.TIMER_Prescaler = 0;
    // Setup the timer period
    TIM1Init.TIMER_Period = 100;
    // Initialization of a timer port by a declared structure
    TIMER_CntInit (MDR_TIMER1, &TIM1Init);
    TIMER_DMACmd (MDR_TIMER1, TIMER_STATUS_CNT_ARR, ENABLE);
    // Run timer
    TIMER_Cmd(MDR_TIMER1, ENABLE);
}

void Timer2Init()

```

```

{
// Indication type of structure and structure name
TIMER_CntInitTypeDef TIM2Init;
// Enabling of clock
RST_CLK_PCLKcmd(RST_CLK_PCLK_TIMER2, ENABLE);
// Filling the structure with default values
TIMER_CntStructInit(&TIM2Init);
// Set of clock divider
TIMER_BRGInit (MDR_TIMER2, TIMER_HCLKdiv1);
// Setup of clock frequency prescaler
TIM2Init.TIMER_Prescaler = 0;
// Setup the timer period
TIM2Init.TIMER_Period = 200;
// Initialization of a timer port by a declared structure
TIMER_CntInit (MDR_TIMER2, &TIM2Init);
// Enabling interrupt
NVIC_EnableIRQ (Timer2_IRQn);
// Setup Interrupt Priority
NVIC_SetPriority (Timer2_IRQn, 1);
// Enabling interrupt when value TIMER1 is zero
TIMER_ITConfig(MDR_TIMER2, TIMER_STATUS_CNT_ZERO, ENABLE);
// Run timer
TIMER_Cmd(MDR_TIMER2, ENABLE);
}

// Timer Interrupt Handling Procedure
void Timer2_IRQHandler()
{
if (TIMER_GetITStatus(MDR_TIMER2, TIMER_STATUS_CNT_ZERO)) TIM-
ER_ClearITPendingBit(MDR_TIMER2, TIMER_STATUS_CNT_ZERO);
ADC1_Start();
while((ADC1_GetStatus() & ADC_STATUS_FLG_REG_EOCIF) == 0);
Rez[Index++] = ADC1_GetResult() & 0xFFFF;
if(Index>=256) Index=0;
}

void DMA_Config()
{
NVIC->ICPR[0] = 0xFFFFFFFF;
NVIC->ICER[0] = 0xFFFFFFFF;
DMA_DeInit();
}

```

```

DMA_StructInit (&DMA_Channel_InitStructure);
DMA_InitStructure.DMA_DestBaseAddr    =    (uint32_t) (&(MDR_DAC->DAC2_DATA));
DMA_InitStructure.DMA_SourceBaseAddr = (uint32_t) Sin_Arr;
DMA_InitStructure.DMA_CycleSize = 256;
DMA_InitStructure.DMA_DestIncSize = DMA_DestIncNo;
DMA_InitStructure.DMA_SourceIncSize = DMA_SourceIncHalfword;
DMA_InitStructure.DMA_MemoryDataSize    =
DMA_MemoryDataSize_HalfWord;
DMA_InitStructure.DMA_NumContinuous = DMA_Transfers_1024;
DMA_InitStructure.DMA_SourceProtCtrl = DMA_SourcePrivileged;
DMA_InitStructure.DMA_DestProtCtrl = DMA_DestPrivileged;
DMA_InitStructure.DMA_Mode = DMA_Mode_Basic;
DMA_Channel_InitStructure.DMA_PriCtrlData    =
&DMA_InitStructure;
DMA_Channel_InitStructure.DMA_SelectDataStructure    =
DMA_CTRL_DATA_PRIMARY;
DMA_Channel_InitStructure.DMA_Priority = DMA_Priority_Default;
DMA_Channel_InitStructure.DMA_UseBurst = DMA_BurstClear;
DMA_Init (DMA_Channel_TIM1, &DMA_Channel_InitStructure);
MDR_DMA->CHNL_REQ_MASK_CLR = 1 << DMA_Channel_TIM1;
MDR_DMA->CHNL_USEBURST_CLR = 1 << DMA_Channel_TIM1;
DMA_Cmd (DMA_Channel_TIM1, ENABLE);
NVIC_SetPriority (DMA_IRQn, 0);
NVIC_EnableIRQ (DMA_IRQn);
}

void ADC_Config()
{
ADC_DeInit();
ADC_StructInit(&sADC);
sADC.ADC_SynchronousMode= ADC_SyncMode_Independent;
sADC.ADC_StartDelay = 15;
ADC_Init (&sADC);
/* ADC1 Configuration */
ADCx_StructInit (&sADCx);
sADCx.ADC_ClockSource= ADC_CLOCK_SOURCE_CPU;
sADCx.ADC_SamplingMode= ADC_SAMPLING_MODE_SINGLE_CONV;
sADCx.ADC_ChannelNumber= ADC_CH_ADC7;
sADCx.ADC_Channels= 0;
sADCx.ADC_VRefSource= ADC_VREF_SOURCE_INTERNAL;

```

```
sADCx.ADC_IntVRefSource= ADC_INT_VREF_SOURCE_INEXACT;  
sADCx.ADC_Prescaler= ADC_CLK_div_None;  
sADCx.ADC_DelayGo = 7;  
ADC1_Init (&sADCx);  
}
```

Приложение В

Массив выборок синусоидального сигнала

```
const int16_t Array[] =
    {0x7C1,0x7EC,0x816,0x841,0x86B,0x896,0x8C0,0x8EA,0x914,0x93E,0
    x967,0x990,0x9B9,0x9E2,0xA0A,0xA32,0xA5A,0xA81,0xAA8,0xACE,0xAF4
    ,0xB19,0xB3E,0xB62,0xB86,0xBA9,0xBCC,0xBEE,0xC0F,0xC30,0xC50,0xC
    6F,0xC8D,0xCAB,0xCC8,0xCE4,0xD00,0xD1B,0xD34,0xD4D,0xD65,0xD7D,0
    xD93,0xDA9,0xDBD,0xDD1,0xDE3,0xDF5,0xE06,0xE16,0xE25,0xE33,0xE3F
    ,0xE4B,0xE56,0xE60,0xE69,0xE71,0xE77,0xE7D,0xE82,0xE86,0xE88,0xE
    8A,0xE8A,0xE8A,0xE88,0xE86,0xE82,0xE7D,0xE77,0xE71,0xE69,0xE60,0
    xE56,0xE4B,0xE3F,0xE33,0xE25,0xE16,0xE06,0xDF5,0xDE3,0xDD1,0xDBD
    ,0xDA9,0xD93,0xD7D,0xD65,0xD4D,0xD34,0xD1B,0xD00,0xCE4,0xCC8,0xC
    AB,0xC8D,0xC6F,0xC50,0xC30,0xC0F,0xBEE,0xBCC,0xBA9,0xB86,0xB62,0
    xB3E,0xB19,0xAF4,0xACE,0xAA8,0xA81,0xA5A,0xA32,0xA0A,0x9E2,0x9B9
    ,0x990,0x967,0x93E,0x914,0x8EA,0x8C0,0x896,0x86B,0x841,0x816,0x7
    EC,0x7C1,0x796,0x76C,0x741,0x717,0x6EC,0x6C2,0x698,0x66E,0x644,0
    x61B,0x5F2,0x5C9,0x5A0,0x578,0x550,0x528,0x501,0x4DA,0x4B4,0x48E
    ,0x469,0x444,0x420,0x3FC,0x3D9,0x3B6,0x394,0x373,0x352,0x332,0x3
    13,0x2F5,0x2D7,0x2BA,0x29D,0x282,0x267,0x24E,0x235,0x21C,0x205,0
    x1EF,0x1D9,0x1C5,0x1B1,0x19E,0x18D,0x17C,0x16C,0x15D,0x14F,0x142
    ,0x137,0x12C,0x122,0x119,0x111,0x10A,0x105,0x100,0x0FC,0x0FA,0x0
    F8,0x0F8,0x0F8,0x0FA,0x0FC,0x100,0x105,0x10A,0x111,0x119,0x122,0
    x12C,0x137,0x142,0x14F,0x15D,0x16C,0x17C,0x18D,0x19E,0x1B1,0x1C5
    ,0x1D9,0x1EF,0x205,0x21C,0x235,0x24E,0x267,0x282,0x29D,0x2BA,0x2
    D7,0x2F5,0x313,0x332,0x352,0x373,0x394,0x3B6,0x3D9,0x3FC,0x420,0
    x444,0x469,0x48E,0x4B4,0x4DA,0x501,0x528,0x550,0x578,0x5A0,0x5C9
    ,0x5F2,0x61B,0x644,0x66E,0x698,0x6C2,0x6EC,0x717,0x741,0x76C,0x7
    96};
```


Редактор
Компьютерная верстка

Подписано для тиражирования
Объем издания
Количество носителей экз.

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ
ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«САМАРСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ
УНИВЕРСИТЕТ ИМЕНИ АКАДЕМИКА С.П. КОРОЛЕВА»
(САМАРСКИЙ УНИВЕРСИТЕТ)
443086, Самара, Московское шоссе, 34

Издательство Самарского университета
443086, Самара, Московское шоссе, 34