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)

CMSIS-RTOS
ДЛЯ МИКРОКОНТРОЛЛЕРОВ
С ЯДРОМ CORTEX-M3

Рекомендовано редакционно-издательским советом федерального государственного автономного образовательного учреждения высшего образования «Самарский национальный исследовательский университет имени академика С.П. Королева» в качестве методических указаний для студентов Самарского университета, обучающихся по основным образовательным программам высшего образования по направлениям подготовки 11.04.03 Конструирование и технология электронных средств, 24.04.01 Ракетные комплексы и космонавтика

Составители: *И. А. Кудрявцев,*
Д. В. Корнилин,
О. О. Мякинин

Самара
Издательство Самарского университета
2020

УДК 004.382.7(075)
ББК 32.973.26я7

Рецензент канд. техн. наук М. П. К а л а е в

CMSIS-RTOS для микроконтроллеров с ядром Cortex-M3: методические указания к лабораторной работе / составители: *И.А. Кудрявцев, Д.В. Корнилин, О.О. Мякинин.* – Самара: Издательство Самарского университета, 2020. – 16 с.

В методических указаниях рассмотрены вопросы разработки программ для микроконтроллеров с ядром Cortex-M3 с использованием CMSIS-RTOS. Продемонстрированы разработка и основные приемы отладки программ для данного типа микропроцессоров.

Методические указания предназначены для студентов, обучающихся по направлениям подготовки 11.04.03 Конструирование и технология электронных средств, 24.04.01 Ракетные комплексы и космонавтика, выполняющих лабораторные работы по дисциплинам «Цифровые устройства и микропроцессоры», «Основы микропроцессорных систем и программирование микроконтроллеров».

Подготовлены на кафедре лазерных и биотехнических систем.

УДК 004.382.7(075)
ББК 32.973.26я7

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Операционная система – популярный инструмент, используемый для улучшения функциональности цифровых устройств на базе микроконтроллеров и микропроцессоров. Основные преимущества, предоставляемые операционными системами, включают в себя переносимость между различными платформами, выполнение нескольких задач в параллельном режиме, предоставление системных сервисов ввода-вывода и т.д. Операционные системы реального времени (ОСРВ, англ. Real-Time Operating System – RTOS) пользуются популярностью благодаря своим компактным ядрам и возможности обеспечить непрерывное выполнение определенного потока, если это необходимо. Рекомендуется изучить [1] для лучшего понимания основ ОСРВ.

Это руководство сфокусировано на CMSIS-RTOS v.2, предназначенной для архитектур с ядром CORTEX. Основные функциональные возможности этой ОСРВ будут исследованы с помощью микроконтроллера с ядром CORTEX-M3 (K1986BE92QI) в среде Keil µVision. Рекомендуется предварительно ознакомиться с ядром CORTEX-M3 [2, 3]. Это руководство не претендует на то, чтобы быть руководством или справочником по ОСРВ, поэтому все подробности функциональной реализации следует изучить самостоятельно, например, с помощью [4].

1 Создание проекта

Создайте пустой проект, как описано в [2], затем добавьте ядро ОСРВ, как показано на рис. 1.

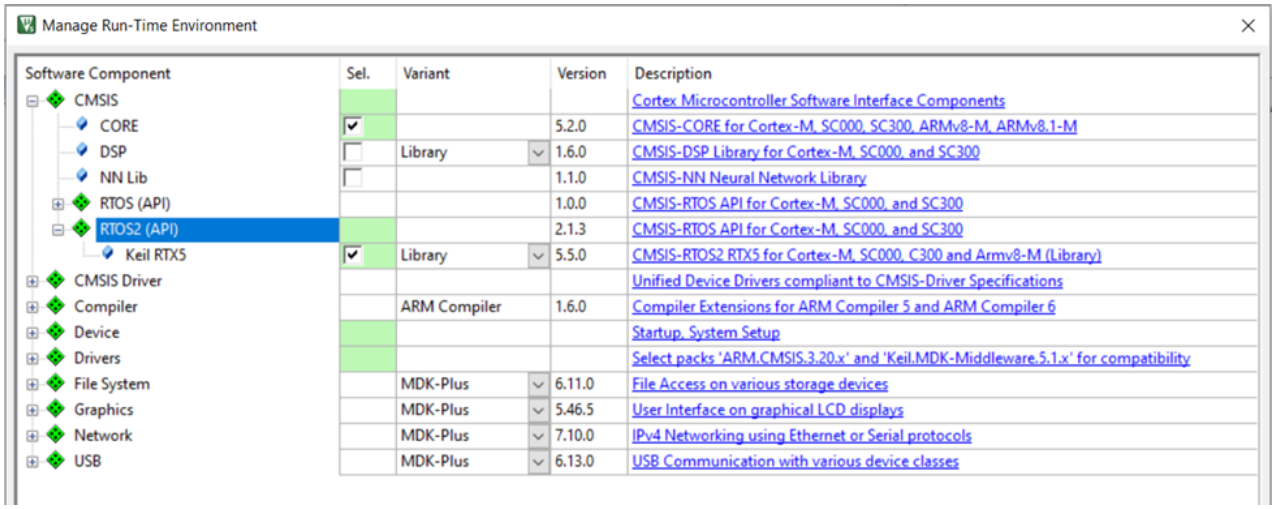

Рис. 1. Добавление поддержки RTOS2 в проект

В проектах ОСРВ все пользовательские функции сосредоточены в потоках, поэтому основная функция обычно выполняет только инициализацию ОСРВ, ядра, периферийных модулей и пользовательских структур. ОСРВ инициализируется функцией `osStatus_t osKernel-Initialize (void)`. Пользовательские потоки создаются путем вызова `osThreadId_t osThreadNew` (функция `osThreadFunc_t`, аргумент `void *`, `const osThreadAttr_t *attr`). Когда все готово, необходимо запустить планировщик, вызвав `osStatus_t osKernelStart (void)`. Эта строка будет последней, выполненной в основной функции. Все необходимые прототипы объявлены в файле `rtx_os.h`, который необходимо добавить строкой `#include <rtx_os.h>`.

Конфигурирование ОСРВ может быть легко выполнено с использованием `RTX_config.h`. Обратите внимание, что вы можете редактировать этот файл в текстовом виде или с помощью утилиты-мастера, как показано на рис. 2. Для нашего первого эксперимента не изменяйте никакие настройки, за исключением снятия флажка **Round-Robin Thread Switching**. Это циклическое переключение потоков обеспечивает поочередное вытеснение, обсуждаемое ниже. В нашем эксперименте мы будем использовать отладочную плату, описанную в [3], с её ЖК-дисплеем, используемым для визуализации. Все необходимые детали использования ЖКИ описаны в [3], нам нужен только вывод текста в строках ЖКИ, выполняемый функцией `void LCD_PutString (const char *string, uint8_t y)`. В нашем эксперименте мы создадим восемь потоков, записывающих информацию в отдельные строки ЖК-дисплея и будем наблюдать за их поведением.

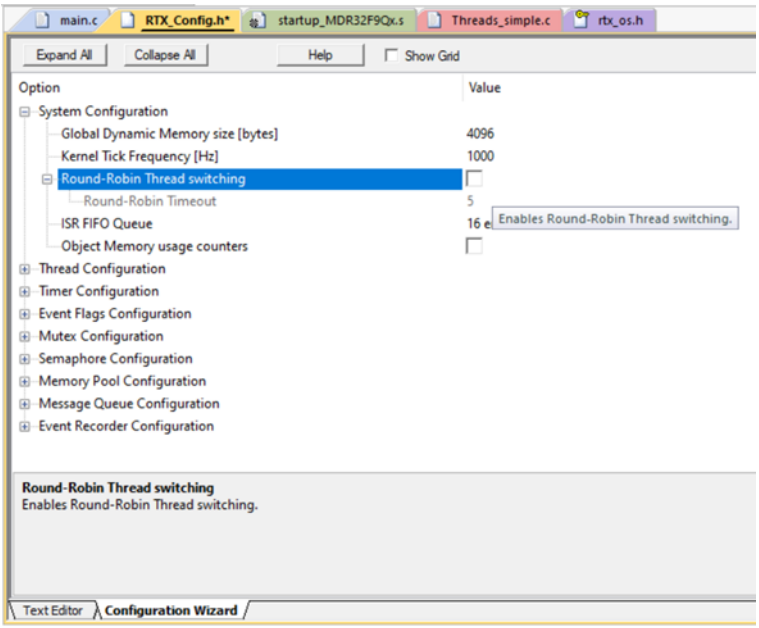

Рис. 2. Конфигурация RTOS

Создайте файл `main.h` и скопируйте туда код из приложения А, затем создайте другой файл `threads.c` и скопируйте туда код из приложения Б. Вы можете видеть, что функции идентичны, за исключением строки, содержащей текст и номер задания. Вызов функции `osDelay (1);` в конце цикла необходим, чтобы передать управление планировщику, однако возможны и другие способы.

Сконфигурируйте проект для размещения в ОЗУ, как показано на рис. 3, соберите его и запустите, как описано в [2]. Вы можете видеть, что все потоки работают синхронно, а числа, отображаемые на ЖК-дисплее меняются почти одновременно.

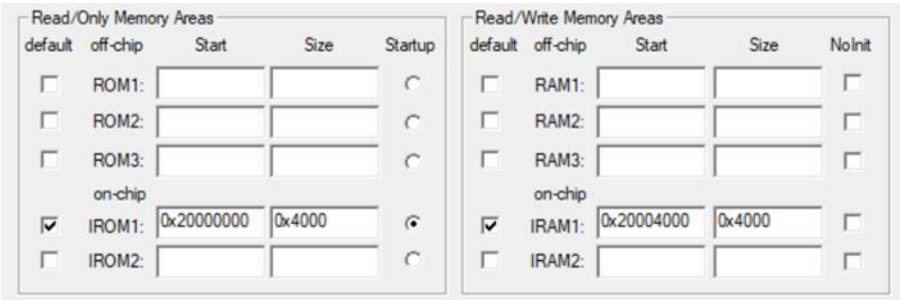

Рис. 3. Конфигурация памяти для эксперимента с размещением кода в ОЗУ

В этом случае функциональность потоков не требует много процессорного времени, и все потоки имеют одинаковый приоритет, что не является универсальным рецептом для всех случаев. В реальных условиях нам часто необходимы потоки, требующие различное количество процессорного времени и, возможно, имеющие различные приоритеты.

2 Вытесняющее поведение

Давайте сымитируем два потока с различными требованиями времени процессора. Добавьте `for (int i = 0; i <1000000; i ++);` в потоки № 1 и № 2 прямо перед строкой, записывающей данные в ЖК-дисплей. Вы можете видеть, что несмотря на замедление только двух потоков, все остальные потоки тоже стали работать заметно медленнее. Причина в том, что все потоки должны ждать медленные потоки. В этом случае вытеснение медленных потоков может улучшить ситуацию. При циклическом переключении (Round-Robin) выполнение потока приостанавливается планировщиком после истечения времени ожидания независимо от завершения операции, и управление передается другому ожидающему потоку. В этом случае все потоки получают

процессорное время в соответствии с их уровнем приоритета, таким образом, в случае равных приоритетов процессорное время распределяется поровну. Активируйте **Round-Robin Thread Switching** в настройках ОСРВ (рис. 2), скомпилируйте проект и запустите программу. Мы видим, что потоки, в которые мы не вставляли никаких задержек, работают быстро. Однако ЖКИ работает с некоторыми неточностями (ошибками). Причина этого – в переключении потоков в момент, когда запись в ЖКИ с помощью функции `LCD_PutString` не завершена.

3 Совместное использование ресурсов

В случаях, подобных вышеописанному, когда потоки должны совместно использовать ресурсы (операция записи в ЖКИ должна быть завершена до того, как планировщик переключит потоки). Для реализации такого случая есть несколько решений: мьютексы, критические секции, семафоры и флаги.

4 Мьютексы

Мьютекс – это двоичный семафор, который может находиться в одном из двух состояний и отображать доступность источника. Поток, пытающийся получить доступ к ресурсу, должен попытаться получить доступ к мьютексу. Если ресурс недоступен, поток приостанавливается до его освобождения ресурса текущим владельцем.

Объявите мьютекс, вставив `osMutexId_t MutexId;` в декларативную часть и создайте его с помощью `MutexId = osMutexNew (NULL);` в `main.c`. Затем вставьте в поток функции `osMutexAcquire (MutexId, osWaitForever);` до вызова `LCD_PutString` и освободите мьютекс с помощью вызова `osMutexRelease (MutexId);` сразу после него. Чтобы сделать MutexId видимым в файле `threads.c`, необходимо объявить его в этом файле с атрибутом `extern`. Запустите проект и наблюдайте за изменениями. Измените настройки **Round-Robin** и выясните, как это влияет на производительность.

5 Критические секции

Критические секции явно не объявляются в CMSIS-RTOS, однако позволяют предотвратить переключение потоков планировщиком с помощью `int32_t osKernelLock (void)`. Вставьте `osKernelLock ()` и

osKernelUnlock () вместо osMutex-Acquire (MutexId, osWaitForever); и osMutexRelease (MutexId); и сравните с использованием мьютекса.

6 Семафоры

Семафор используется, когда необходимо обеспечить ограниченный доступ к ресурсу для нескольких потоков (вместо атомарного доступа единственным потоком, что достигается использованием мьютексов). В нашем эксперименте мы исследуем поведение семафора, ограничивая доступ к LCD только для четырёх потоков. Объявите семафор (osSemaphoreId_t osSemaphoreId;) и создайте его, вызвав osSemaphoreId = osSemaphoreNew (4,4, NULL).

Вставьте вызовы osSemaphoreAcquire (osSemaphoreId, osWaitForever); перед while (1) во все функции потока. Не забудьте объявить свой идентификатор семафора в файле threads.c с атрибутом extern. Затем вставьте в любые четыре функции потоков после вызова osKernelUnlock () следующий блок:

```
if(Count>400)
  {
  osSemaphoreRelease(osSemaphoreId);
  return;
  }
```

Эти четыре потока будут завершены после того, как переменная Count достигнет 400, тогда ожидающие функции могут получить доступ к семафору и ресурсу.

7 Флаги

В CMSIS-RTOS есть два типа флагов: флаги событий и флаги потоков. Вы можете узнать больше об этих объектах в [4]. Мы будем использовать флаги событий, чтобы информировать поток о событии (завершение преобразования АЦП). Детали поведения и настройки АЦП были описаны в [3], поэтому мы не будем обсуждать их здесь. Скопируйте код, конфигурирующий АЦП, из приложения В в функцию потока Thread_One (перед while (1)). Теперь этот поток будет ожидать флаг события, который устанавливается в обработчике прерывания. Скопируйте код обработчика из приложения В в файл main.c.

Объявите событие (`osEventFlagsId_t event;`) и создайте его, вызвав `event = osE-ventFlagsNew (NULL);` в файле `main.c`. Изучите код и проверьте отображение данных, меняя напряжение с помощью TRIM. Код, используемый в этом эксперименте, может потребовать больше памяти, чем раньше. Выберите соответствующие настройки для конфигурации памяти, как показано на рис. 3. Манипуляции с данными, выполняемые потоками, могут потребовать больше места, чем выделено по умолчанию. Вы можете изменить этот параметр, создав поток, как показано ниже, или напрямую изменив настройки в структуре `ThreadAttr`.

```
ThreadId[0] = osThreadNew(Thread_One,NULL,&(osThreadAttr_t)
{.stack_size=400});
```

Процесс выбора не прост, но вы можете посмотреть некоторые оценки, используя меню **View / Watch Windows / RTX RTOS**. Рекомендуется использовать не более 80% стека.

8 Очереди сообщений

Очереди сообщений описаны в [4]. Они предоставляют буфер типа FIFO, который можно использовать для обмена сообщениями. Мы можем изучить очереди сообщений, взяв за основу предыдущий эксперимент. Замените события очередями, используя следующие строки:

1. Объявление - `osMessageQueueId_t MsgQueue;`
2. Создание - `MsgQueue = osMessageQueueNew(4,4,NULL);`
3. Запись -
```
uint16_t digit = MDR_ADC->ADC1_RESULT &
ADC_RESULT_Msk;
osMessageQueuePut(MsgQueue, &digit, osPriorityNormal, 0);
```
4. Чтение - `osMessageQueueGet(MsgQueue, &data, NULL, 0U);`

Данные должны иметь тип (uint32_t). Внесите эти изменения и изучите работу кода.

9 Приоритет потоков

Приоритеты потоков могут быть установлены в диапазоне от `osPriorityLow` до `osPriorityRealtime` с некоторыми дополнительными градациями, как описано в [4]. Изменяя это значение на этапе создания потока, подобно размеру стека, как показано выше, исследуйте поведение RTOS и сделайте выводы.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Гриценко, Ю. Б. Системы реального времени [Электронный ресурс] : учебное пособие / Ю. Б. Гриценко. – Москва : ТУСУР, 2017. – 253 с.

2 Разработка и отладка программ для микроконтроллеров с ядром Cortex-M3 в среде µVision: методические указания к лабораторной работе / составители: И.А. Кудрявцев, Д.В. Корнилин, О.О. Мякинин. – Самара : Издательство Самарский университет, 2020. – 24 с.

2 Работа с периферией микроконтроллеров с ядром Cortex-M3 в среде µVision: методические указания к лабораторной работе / составители: И.А. Кудрявцев, Д.В. Корнилин, О.О. Мякинин. – Самара : Издательство Самарский университет, 2020. – 24 с.

4. CMSIS-RTOS2 Documentation. – URL: https://www.keil.com/pack/doc/CMSIS/RTOS2/html/index.html (accessed: 10/15/2019).

Приложение А

```
#include <rtx_os.h>
#include "lcd.h"

void Thread_One(void *argument);
void Thread_Two(void *argument);
void Thread_Three(void *argument);
void Thread_Four(void *argument);
void Thread_Five(void *argument);
void Thread_Six(void *argument);
void Thread_Seven(void *argument);
void Thread_Eight(void *argument);

static osThreadAttr_t ThreadAttr[8];
osThreadId_t ThreadId[8];

int main()
{
  LCD_Init();
  osKernelInitialize();

ThreadId[0] = osThreadNew(Thread_One,NULL,&ThreadAttr[0]);
ThreadId[1] = osThreadNew(Thread_Two,NULL,&ThreadAttr[1]);
ThreadId[2] = osThreadNew(Thread_Three,NULL,&ThreadAttr[2]);
ThreadId[3] = osThreadNew(Thread_Four,NULL,&ThreadAttr[3]);
ThreadId[4] = osThreadNew(Thread_Five,NULL,&ThreadAttr[4]);
ThreadId[5] = osThreadNew(Thread_Six,NULL,&ThreadAttr[5]);
ThreadId[6] = osThreadNew(Thread_Seven,NULL,&ThreadAttr[6]);
ThreadId[7] = osThreadNew(Thread_Eight,NULL,&ThreadAttr[7]);

osKernelStart();
}
```

Приложение Б

```
#include <rtx_os.h>
#include "lcd.h"

void Thread_One(void *argument)
{
int Count=0;
char Buffer[32];
while (1)
  {
  sprintf(Buffer,"Task 0, Count=%d", Count++);
  LCD_PutString(Buffer,0);
  osDelay(1);
  }
}

void Thread_Two(void *argument)
{
int Count=0;
char Buffer[32];
while (1)
  {
  sprintf(Buffer,"Task 1, Count=%d", Count++);
  LCD_PutString(Buffer,1);
  osDelay(1);
  }
}

void Thread_Three(void *argument)
{
int Count=0;
char Buffer[32];
while (1)
  {
  sprintf(Buffer,"Task 2, Count=%d", Count++);
  LCD_PutString(Buffer,2);
  osDelay(1);
  }
}
```

```
void Thread_Four(void *argument)
{
int Count=0;
char Buffer[32];
while (1)
  {
  sprintf(Buffer,"Task 3, Count=%d", Count++);
  LCD_PutString(Buffer,3);
  osDelay(1);
  }
}



void Thread_Five(void *argument)
{
int Count=0;
char Buffer[32];
while (1)
  {
  sprintf(Buffer,"Task 4, Count=%d", Count++);
  LCD_PutString(Buffer,4);
  osDelay(1);
  }
}

void Thread_Six(void *argument)
{
int Count=0;
char Buffer[32];
while (1)
  {
  sprintf(Buffer,"Task 5, Count=%d", Count++);
  LCD_PutString(Buffer,5);
  osDelay(1);
  }
}

void Thread_Seven(void *argument)
{
int Count=0;
```

```
char Buffer[32];
while (1)
  {
  sprintf(Buffer,"Task 6, Count=%d", Count++);
  LCD_PutString(Buffer,6);
  osDelay(1);
  }
}

void Thread_Eight(void *argument)
{
int Count=0;
char Buffer[32];
while (1)
  {
  sprintf(Buffer,"Task 7, Count=%d", Count++);
  LCD_PutString(Buffer,7);
  osDelay(1);
  }
}
```

Приложение В

```
  void Thread_One(void *argument)
  {
  char Buffer[32];
  ADC_InitTypeDef sADC;
  ADCx_InitTypeDef sADCx;

  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_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;
  MDR_ADC->ADC1_STATUS = (1 << ADC_STATUS_ECOIF_IE_Pos);
  NVIC_SetPriority(ADC_IRQn, 1);
  NVIC_EnableIRQ(ADC_IRQn);

  ADC1_Init (&sADCx);
  ADC1_Cmd (ENABLE);

  while (1)
    {
    ADC1_Start();
     osEventFlagsWait(event, 0x00000001, osFlagsWaitAny,
osWaitForever);
    sprintf(Buffer,"Task 0, Result=0x%0X", MDR_ADC->ADC1_RESULT &
ADC_RESULT_Msk);
    osKernelLock();
    LCD_PutString(Buffer,0);
    osKernelUnlock();
    osDelay(1);
    }
  }

  // Interrupt handler
  void ADC_IRQHandler(void)
  {
  osEventFlagsSet(event, 0x00000001U);
  }
```

Методические материалы

CMSIS-RTOS ДЛЯ МИКРОКОНТРОЛЛЕРОВ С ЯДРОМ CORTEX-M3

Методические указания к лабораторной работе

Составители: ***Кудрявцев Илья Александрович,***
Корнилин Дмитрий Владимирович,
Мякинин Олег Олегович

Редактор А.В. Ярославцева
Компьютерная вёрстка А.В. Ярославцевой

Подписано в печать 30.12.2020. Формат 60×84 1/16.
Бумага офсетная. Печ. л. 1,0.
Тираж 25 экз. Заказ . Арт. – 38(Р3М)/2020.

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.