MINISTERIO DE CIENCIA Y EDUCACIÓN SUPERIOR FEDERACIÓN RUSA

Institución educativa autónoma del estado federal de educación superior
"UNIVERSIDAD NACIONAL DE INVESTIGACIÓN
DE SAMARA nombre académico S. P. Korolev"

CMSIS-RTOS PARA MICROCONTROLADORES 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 обучающихся Самарского университета по основным образовательным программам высшего образования 24.04.01 Ракетные комплексы и космонавтика, 11.04.01 Радиотехника, 03.04.01 Прикладные математика и физика

Составители: *I.A. Kudryavtsev, D.V. Kornilin,*
O.O. Myakinin

САМАРА
Издательство Самарского университета
2021

УДК 004.9(075)
ББК 32.81я7

Составители: ***I.A. Kudryavtsev, D.V. Kornilin, O.O. Myakinin***

Рецензент: д-р физ.-мат. наук, доц. В. С. Павельев

Traducción al Español **Victor Niels Romero Alva**
Revisión de estilo y formato **José Pedro Laverde Estrada**

CMSIS-RTOS para microcontroladores CORTEX-M3: методические указания / сост.: *I.A. Kudryavtsev, D.V. Kornilin, O.O. Myakinin.* – Самара: Издательство Самарского университета, 2021. – 24 с.

La siguiente guía se encuentra enfocada en el desarrollo de software basado en RTOS para microcontroladores con núcleo Cortex-M3, demostrando las principales técnicas de desarrollo y depuración con CMSIS-RTOS. Las pautas establecidas están dirigidas hacia estudiantes de las carreras de “Radio ingeniería” 11.04.01, “Matemáticas y Física Aplicada” 03.04.01, y “Sistemas de Vehículos Espaciales y Cosmonáutica” 24.03.01/24.04.01. El laboratorio de entrenamiento con CMSIS-RTOS puede ser llevado a cabo dentro de los cursos “Dispositivos digitales y microprocesadores”, “Sistemas fundamentales de un microprocesador” y “Programación de microcontroladores”. Esta guía fue elaborada por el “Departamento de Láser y Sistemas Biotécnicos”.

© Самарский университет, 2021

ÍNDICE

Introducción 4
1. Creación de un proyecto basado en RTOS 5
2. Comportamiento desplazador 8
3. Intercambio de recursos 8
4. Mutexes.. 9
5. Secciones críticas 9
6. Semáforos 10
7. Banderas... 10
8. Colas de mensajes 12
9. Propiedades de subprocesos... 12
Referencias.. 13
Apéndice A 14
Apéndice B 16
Apéndice C 19

INTRODUCCIÓN

Un sistema operativo es un instrumento popular utilizado para mejorar la funcionalidad de los dispositivos digitales basados en MCU. Las ventajas principales proporcionadas por los sistemas operativos incluyen la fácil portabilidad entre varias plataformas, ejecución de varias tareas en paralelo, implementación más fácil de funciones triviales, entre otras. Los Sistemas Operativos en Tiempo Real (RTOS) son comúnmente utilizados debido a sus núcleos compactos y a las oportunidades que ofrecen de asegurar una ejecución sin interrupción de un subproceso determinado en caso de ser necesario. Se recomienda estudiar [1] para comprender mejor los fundamentos de un RTOS.

Esta guía se enfoca en CMSIS-RTOS v.2, el cual está destinado para arquitecturas CORTEX. La funcionalidad principal de estos RTOS será investigada usando CORTEX-M3 MCU (K1986WE92QI) en el entorno Keil µVision. Se recomienda tener un conocimiento previo del núcleo CORTEX-M3 antes de este entrenamiento de laboratorio, utilizando [2, 3]. Esta guía no pretende ser un manual o una guía de referencia para RTOS, así que todos los detalles de la implementación funcional deberán de ser investigadas, por ejemplo, en [4].

1. CREACIÓN DE UN PROYECTO BASADO EN RTOS

El CMSIS-RTOS requiere de algunas operaciones para la creación de un proyecto y hacer uso de su funcionalidad. Primeramente, se debe crear un proyecto vacío, como se describe en [2], y luego añadir el núcleo RTOS como se muestra en la fig. 1.

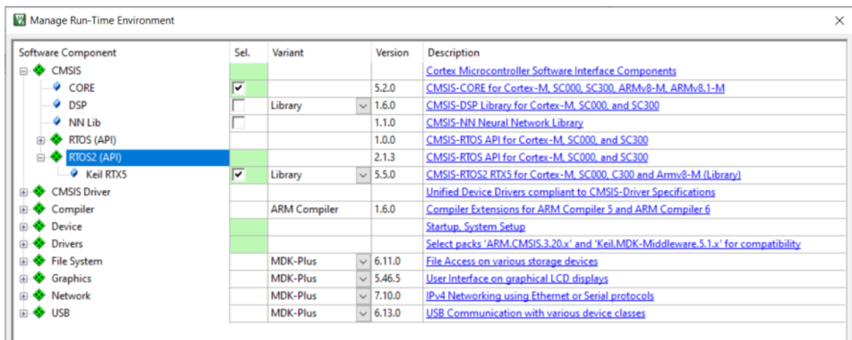

Figura 1 – Añadiendo soporte RTOS2 a un proyecto

En los proyectos de RTOS, toda la funcionalidad de usuario se concentra en subprocesos, por lo que la función principal generalmente realiza solo la inicialización de RTOS, núcleos, módulos periféricos y estructuras de usuario. El RTOS se inicializa mediante la función **osStatus_t osKernelInitialize (void).** Posteriormente, los subprocesos de usuario son creados llamando a **osThreadId_t osThreadNew (osThreadFunc_t func, void * argument, const osThreadAttr_t * attr)**. Una vez todo esté listo, es necesario iniciar el programador llamando a **osStatus_t osKernelStart (void).** Esta línea será la última a

ejecutar en la función **main.** Todos los prototipos necesarios son declarados en el archivo rtx_os.h, el cual es necesario agregar por la línea **#include <rtx_os.h>**.

La configuración del RTOS puede ser realizada fácilmente, utilizando **RTX_config.h.** Tenga en cuenta que puede editar este archivo en formato de texto o como asistente (Wizard), como se muestra en la figura 2. Para nuestro primer experimento, no se debe modificar ninguna configuración, excepto desmarcando **Round-Robin Thread Switching.** Este cambio cíclico de subprocesos proporciona un desplazamiento alternativo, el cual se discute a continuación. En nuestro experimento usaremos una placa de evaluación, descrita en [3], con una pantalla LCD utilizada para la visualización. Todos los detalles

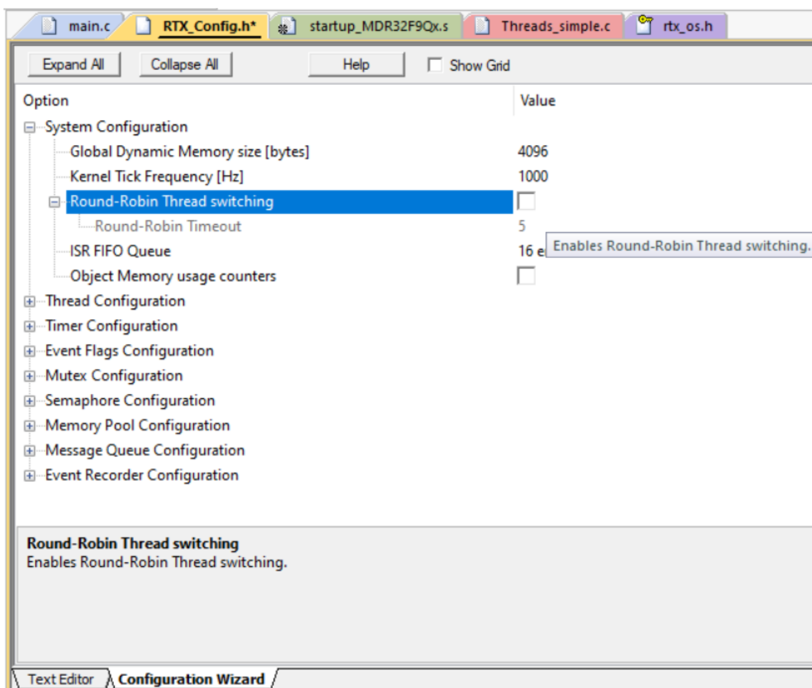

Figura 2 – Configuración RTOS

necesarios del uso del LCD se describen en [3], únicamente se requiere de la salida del texto en las filas de la pantalla, realizada por la función **void LCD_PutString(const char* string, uint8_t y).** En nuestro experimento, creamos ocho subprocesos, escribiendo la información por líneas separadas de la pantalla LCD y observando su comportamiento.

Cree el archivo **main.c** y copie allí el código del Anexo A, posteriormente, cree otro archivo threads.c y copie allí el código del Anexo B. Puede observar que las funciones son idénticas excepto la fila de la línea, donde está escrito el texto y el número de tareas. Ahora, llame la función **osDelay(1);** al final del bucle es necesario ceder el control al programador, sin embargo, también es posible a través de otras formas.

Configure el proyecto para que se ubique en la memoria RAM, como se muestra en la figura 3, posteriormente, constrúyalo y ejecútelo como se describe en [2]. Puede observar que todos los subprocesos son ejecutados sincrónicamente y los números, escritos en la pantalla LCD, cambian casi simultáneamente.

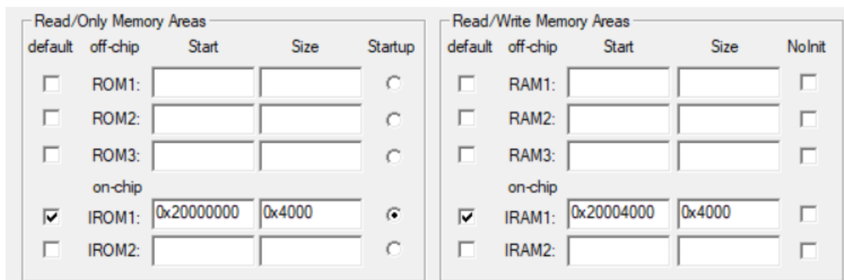

Figura 3 – Configuración de memoria para experimentos basados en RAM

En este caso, la funcionalidad de los subprocesos no requiere mucho tiempo de CPU y todos los subprocesos tienen la misma prioridad, lo cual no es el caso para todas las tareas. En condiciones reales, a menudo

tenemos subprocesos que toman distintas cantidades de tiempo de CPU y tal vez tienen varias prioridades.

2. COMPORTAMIENTO DESPLAZADOR

Simularemos varios requerimientos de tiempo de CPU en dos subprocesos. Añada **for (int i=0; i<1000000; i++);** en los subprocesos 1 y 2 justo antes de la línea que escribe los datos en la pantalla LCD. Se puede observar que, a pesar de que solo se ralentizan dos subprocesos, todos los demás subprocesos también se volvieron notablemente más lentos. La razón es que todos los subprocesos tienen que esperar a que dichos procesos lentos cedan el control. Pero para este caso, el comportamiento desplazador puede mejorar la situación. En el Round-Robin Thread Switching, el programador puede suspender la ejecución del subproceso después de que expire el tiempo de espera, independientemente de la finalización de la operación, el control es transferido a otro subproceso en espera. De esta forma, todos los subprocesos obtienen tiempo de CPU de acuerdo con su nivel de prioridad, por lo que, en caso de prioridades iguales, el tiempo de CPU se comparte equitativamente. Active Round-Robin Thread Switching en la configuración de RTOS en estado marcado (fig. 2), compile el proyecto y ejecute el programa. Podemos ver que los subprocesos donde no insertamos ningún retraso trabajan rápidamente, sin embargo, el LCD trabaja con algunas fallas. La razón está en el cambio de subproceso en el momento en que la escritura en la pantalla LCD, utilizando la función **LCD_PutString**, no se completa.

3. INTERCAMBIO DE RECURSOS

Para un caso como el que tenemos ahora, cuando los subprocesos deben compartir los recursos (la operación de escritura en la pantalla

LCD debe completarse antes de que el programador cambie los subprocesos), existen varias soluciones: mutexes, secciones críticas, semáforos, y banderas.

4. MUTEXES

Mutex es un semáforo binario, el cual puede estar en uno de dos estados, mostrando la disponibilidad de un recurso. Todo subproceso que intenta acceder a un recurso debe intentar adquirir un mutex. Si el recurso no está disponible, el subproceso se suspende hasta que el propietario actual lo libere.

Declare un mutex insertando **osMutexId_t MutexId;** en la parte declarativa y creelo con **MutexId = osMutexNew(NULL);** en **main.c**. Luego, inserte en las funciones de subprocesos **osMutexAcquire(MutexId,osWaitForever);** antes de llamar a **LCD_PutString** y liberar el mutex con una llamada a **osMutexRelease(MutexId);** justo después de eso. Para hacer que **MutexId** sea visible en el archivo **threads.c**, es necesario declararlo en este archivo con el atributo **extern.** Se deben de corregir todas las funciones de subprocesos. Ejecute el proyecto y observe los cambios. Cambie la configuración de Round-Robin y estudie cómo esta afecta el rendimiento.

5. SECCIONES CRÍTICAS

Las secciones críticas no se declaran explícitamente en CMSIS-RTOS, sin embargo, permite evitar que el programador cambie el subproceso con **int32_t osKernelLock (void).** Inserte **osKernelLock()** y **osKernelUnlock()** en lugar de **osMutexAcquire(MutexId, osWaitForever);** y **osMutexRelease(MutexId);** y compare lo observado con el uso de mutex.

6. SEMÁFOROS

El semáforo es utilizado cuando se necesita proporcionar acceso limitado a un recurso para varios subprocesos (en lugar del acceso atómico de un único subproceso, que se logra mediante el uso de mutex). En nuestro experimento, estudiamos el comportamiento del semáforo, limitando el acceso a la pantalla LCD solo para cuatro subprocesos. Declare el semáforo (**osSemaphoreId_t osSemaphoreId;**) y creelo llamando a **osSemaphoreId = osSemaphoreNew(4,4,NULL).** Inserte llamadas a **osSemaphoreAcquire(osSemaphoreId,osWaitForever);** antes de **while(1)** en todas las funciones de subprocesos. No olvide declarar su ID de semáforo en el archivo **threads.c** con el atributo **extern.** Luego, inserte el siguiente bloque en cualquiera de las cuatro funciones de subprocesos después de llamar **osKernelUnlock().**

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

Estos cuatro subprocesos deben de ser completados después de que la variable Count llegue a 400 posteriormente, las funciones pendientes podrán acceder al semáforo. Investigue este comportamiento.

7 BANDERAS

Existen dos tipos de banderas en CMSIS-RTOS: banderas de eventos y banderas de subprocesos. Puede ver más acerca de estos objetos en [4]. Se utilizan banderas de eventos para informar al subproceso sobre un evento (finalización de la conversión ADC). Los

detalles del comportamiento y la configuración del ADC se han descrito en [3], por lo tanto, no volveremos a discutirlo aquí. Copie el código de configuración de ADC, del anexo C a la función de subproceso **Thread_One** (antes de **while(1)**). Ahora este subproceso esperará una bandera de evento, la cual se establece en el controlador de interrupciones. Copie el código del controlador del Anexo C en el archivo main.c.

Declare el evento **(osEventFlagsId_t event;)** y creelo llamando a **event = osEventFlagsNew(NULL);** en el archivo **main.c**. Investigue el código y verifique la visualización de los datos cambiando cuidadosamente el voltaje con la ayuda del TRIM. El código utilizado en el experimento puede requerir más memoria que la utilizada anteriormente, para esto seleccione los ajustes apropiados para la configuración de la memoria, como se muestra en la figura 3. Las manipulaciones de datos, ejecutadas por subprocesos, pueden requerir más espacio que el asignado por defecto. Puede cambiar esta configuración, creando un subproceso, como se muestra a continuación o modificando directamente la configuración en la estructura **ThreadAttr**.

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

El proceso de selección no es sencillo, pero puede observar algunas evaluaciones usando el menú View/Watch Windows/RTX RTOS. Se recomienda tener no más del 80% del uso de la batería.

8 COLAS DE MENSAJES

Las colas de mensajes se describen en [4]. Estas proporcionan un búfer de tipo FIFO, que se puede utilizar para el intercambio de mensajes. Se pueden investigar las colas de mensajes basándonos en el experimento anterior. Sustituya eventos por colas utilizando las siguientes líneas:

1. Declaración – osMessageQueueId_t MsgQueue;
2. Creación – MsgQueue = osMessageQueueNew(4,4,NULL);
3. Anotación –

uint16_t digit = MDR_ADC->ADC1_RESULT & ADC_RESULT_Msk;

osMessageQueuePut(MsgQueue, &digit, osPriorityNormal, 0);

4. Lectura – osMessageQueueGet(MsgQueue, &data, NULL, 0U);

Los **datos** deben de ser un marcador de posición de tipo **(uint32_t)**. Realice los cambios e investigue el comportamiento del código.

9 PROPIEDADES DE SUBPROCESOS

Las prioridades de los subprocesos se pueden establecer dentro de un rango de osPriorityLow a osPriorityRealtime con algunos grados adicionales, como se describe en [4]. Al cambiar este valor durante la fase de creación de subprocesos, similar al tamaño de la pila, como se muestra arriba, investigue el comportamiento de RTOS y saque conclusiones.

REFERENCIAS

1. Cooling, J. Real-time Operating Systems: Book 1 – The Theory (The engineering of real-time embedded systems) / J. Cooling. – Lindentree Associates, 2013.
2. Kudryavtsev, I.A. Software development for Cortex-M3 in Keil µVision: Guide / I.A. Kudryavtsev, D.V. Kornilin, O.O. Myakinin. – Samara: Samara National Research University, 2020. – 23 p.
3. Kudryavtsev, I.A. Studing of peripherals of Cortex-M3 in Keil µVision: Guide / I.A. Kudryavtsev, D.V. Kornilin, O.O. Myakinin. – Samara: Samara National Research University, 2020. – 25 p.
4. CMSIS-RTOS2 Documentation. URL: https://www.keil.com/ pack/doc/
5. CMSIS/RTOS2/html/index.html (accessed: 10/15/2019).

APÉNDICE A

```
#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();
    }
```

APÉNDICE B

```
#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);
}}
```

```
    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 PARA MICROCONTROLADORES CORTEX-M3

Методические указания

Составители:
Kudryavtsev Ilya Aleksandrovich,
Kornilin Dmitry Vladimirovich,
Myakinin Oleg Olegovich

Редактор Л.Р. Д м и т р и е н к о
Компьютерная верстка Л.Р. Д м и т р и е н к о

Подписано в печать 15.12.2021. Формат 60x84 1/16.
Бумага офсетная. Печ. л. 1,5.
Тираж 25 экз. Заказ . Арт. – 14(Р4М)/2021.

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.