Лабораторная работа

2023-12-25

Данная лабораторная работа подготовлена Кириленко Яковом Александровичем по заказу [Альянса RISC‑V](https://riscv-alliance.ru/), допускается к использованию под лицензией [CC BY 4.0](https://creativecommons.org/licenses/by/4.0/deed.ru).

Приведённые ниже команды рассчитаны на выполнение в рабочей среде *Syntacore Kit*, распространяемой [Альянсом RISC‑V](https://riscv-alliance.ru/) в образовательных целях. Обратите внимание, что в другом окружении приведённые примеры команд могут не работать или работать иначе.

# Установка исходного кода FreeRTOS

В данной работе используется релиз FreeRTOSv202212.01. С целью воспроизводимости лабораторной рекомендуется скачать его следующим образом:

**Скачивание**

wget https://github.com/FreeRTOS/FreeRTOS/releases/download/202212.01/FreeRTOSv202212.01.zip

**Распаковка**

unzip FreeRTOSv202212.01.zip && mv FreeRTOSv202212.01 FreeRTOS

# Сборка и запуск демонстрационной версии приложения

Перед выполнением команд проверьте, указан ли в переменной PATH путь /opt/syntacore/sc-dt/2023.08/riscv-gcc/bin (2023.08 — версия Syntacore Kit):

echo $PATH | grep riscv-gcc/bin

Если нет, добавьте в файл ~/.bashrc следующую строку (*путь зависит от версии используемой рабочей среды*):

export PATH=$PATH:/opt/syntacore/sc-dt/2023.08/riscv-gcc/bin

Перейдите в каталог с демонстрационным приложением RISC-V-Qemu-virt\_GCC.

cd FreeRTOS/FreeRTOS/Demo/RISC-V-Qemu-virt\_GCC/

Попробуйте собрать демо.

make

При попытке сборки будет получено следующее сообщение об ошибке:

main.c: Assembler messages:  
main.c:53: Error: unrecognized opcode `csrw mtvec,a5'  
main.c:84: Error: unrecognized opcode `csrc mstatus,8'  
main.c:111: Error: unrecognized opcode `csrc mstatus,8'  
main.c:125: Error: unrecognized opcode `csrc mstatus,8'  
make: \*\*\* [Makefile:62: build/main.o] Error 1

Обратите внимание, что в Makefile указаны флаги компилятора и компоновщика -march=rv32imac, задающие архитектуру набора команд rv32imac со стандартными расширениями. Однако перечисленные в листинге инструкции являются частью расширения Zicsr, вынесенного из базовой ISA версии выше 2.2. И для gcc версии 11.1.0 и выше необходимо отдельно указывать расширение Zicsr.

Итак, чтобы собрать демо, необходимо в Makefile в строках 17 и 24 заменить -march=rv32imac на -march=rv32ima\_zicsr.

**Makefile**

CROSS = riscv64-unknown-elf-  
CC = $(CROSS)gcc  
OBJCOPY = $(CROSS)objcopy  
ARCH = $(CROSS)ar  
  
BUILD\_DIR = build  
RTOS\_SOURCE\_DIR = $(abspath ../../Source)  
DEMO\_SOURCE\_DIR = $(abspath ../Common/Minimal)  
  
CPPFLAGS = \  
 -D\_\_riscv\_float\_abi\_soft \  
 -DportasmHANDLE\_INTERRUPT=handle\_trap \  
 -I . -I ../Common/include \  
 -I $(RTOS\_SOURCE\_DIR)/include \  
 -I $(RTOS\_SOURCE\_DIR)/portable/GCC/RISC-V \  
 -I $(RTOS\_SOURCE\_DIR)/portable/GCC/RISC-V/chip\_specific\_extensions/RV32I\_CLINT\_no\_extensions  
CFLAGS = -march=rv32imac\_zicsr -mabi=ilp32 -mcmodel=medany \  
 -Wall \  
 -fmessage-length=0 \  
 -ffunction-sections \  
 -fdata-sections \  
 -fno-builtin-printf  
LDFLAGS = -nostartfiles -Tfake\_rom.lds \  
 -march=rv32imac\_zicsr -mabi=ilp32 -mcmodel=medany \  
 -Xlinker --gc-sections \  
 -Xlinker --defsym=\_\_stack\_size=300 \  
 -Xlinker -Map=RTOSDemo.map  
  
ifeq ($(DEBUG), 1)  
 CFLAGS += -Og -ggdb3  
else  
 CFLAGS += -O2  
endif  
  
SRCS = main.c main\_blinky.c riscv-virt.c ns16550.c \  
 $(DEMO\_SOURCE\_DIR)/EventGroupsDemo.c \  
 $(DEMO\_SOURCE\_DIR)/TaskNotify.c \  
 $(DEMO\_SOURCE\_DIR)/TimerDemo.c \  
 $(DEMO\_SOURCE\_DIR)/blocktim.c \  
 $(DEMO\_SOURCE\_DIR)/dynamic.c \  
 $(DEMO\_SOURCE\_DIR)/recmutex.c \  
 $(RTOS\_SOURCE\_DIR)/event\_groups.c \  
 $(RTOS\_SOURCE\_DIR)/list.c \  
 $(RTOS\_SOURCE\_DIR)/queue.c \  
 $(RTOS\_SOURCE\_DIR)/stream\_buffer.c \  
 $(RTOS\_SOURCE\_DIR)/tasks.c \  
 $(RTOS\_SOURCE\_DIR)/timers.c \  
 $(RTOS\_SOURCE\_DIR)/portable/MemMang/heap\_4.c \  
 $(RTOS\_SOURCE\_DIR)/portable/GCC/RISC-V/port.c  
  
ASMS = start.S vector.S\  
 $(RTOS\_SOURCE\_DIR)/portable/GCC/RISC-V/portASM.S  
  
OBJS = $(SRCS:%.c=$(BUILD\_DIR)/%.o) $(ASMS:%.S=$(BUILD\_DIR)/%.o)  
DEPS = $(SRCS:%.c=$(BUILD\_DIR)/%.d) $(ASMS:%.S=$(BUILD\_DIR)/%.d)  
  
$(BUILD\_DIR)/RTOSDemo.axf: $(OBJS) fake\_rom.lds Makefile  
 $(CC) $(LDFLAGS) $(OBJS) -o $@  
  
$(BUILD\_DIR)/%.o: %.c Makefile  
 @mkdir -p $(@D)  
 $(CC) $(CPPFLAGS) $(CFLAGS) -MMD -MP -c $< -o $@  
  
$(BUILD\_DIR)/%.o: %.S Makefile  
 @mkdir -p $(@D)  
 $(CC) $(CPPFLAGS) $(CFLAGS) -MMD -MP -c $< -o $@  
  
clean:  
 rm -rf $(BUILD\_DIR)  
  
-include $(DEPS)

Снова соберите демо.

make

Теперь можно запустить демонстрационную версию приложения, имитирующего мигание светодиода.

**Запуск приложения в qemu**

qemu-system-riscv32 -nographic -machine virt -net none \  
 -chardev stdio,id=con,mux=on -serial chardev:con \  
 -mon chardev=con,mode=readline -bios none \  
 -smp 4 -kernel ./build/RTOSDemo.axf

**Вывод**

Hello FreeRTOS!  
0: Tx: Transfer1  
0: Rx: Blink1  
0: Tx: Transfer2  
0: Rx: Blink2  
0: Tx: Transfer1  
0: Rx: Blink1  
0: Tx: Transfer2  
0: Rx: Blink2  
0: Tx: Transfer1  
0: Rx: Blink1  
0: Tx: Transfer2  
0: Rx: Blink2  
0: Tx: Transfer1  
0: Rx: Blink1  
0: Tx: Transfer2  
0: Rx: Blink2  
0: Tx: Transfer1  
0: Rx: Blink1  
0: Tx: Transfer2  
0: Rx: Blink2  
0: Tx: Transfer1  
0: Rx: Blink1  
0: Tx: Transfer2  
0: Rx: Blink2  
0: Tx: Transfer1  
0: Rx: Blink1  
0: Tx: Transfer2  
0: Rx: Blink2  
0: Tx: Transfer1  
0: Rx: Blink1  
0: Tx: Transfer2  
0: Rx: Blink2  
0: Tx: Transfer1  
0: Rx: Blink1  
0: Tx: Transfer2  
0: Rx: Blink2

# Создание своего приложения

## Создание задач

В этом же каталоге создадим файл example.c, создающий и запускающий две задачи vTask1 и vTask2 с одинаковым приоритетом.

**example.c**

#include <FreeRTOS.h>  
#include <task.h>  
#include <queue.h>  
  
#include <stdio.h>  
  
#include "riscv-virt.h"  
#include "ns16550.h"  
  
#define mainDELAY\_LOOP\_COUNT ( 10000000 )  
#define exampleTASK\_PRIORITY ( tskIDLE\_PRIORITY + 1 )  
  
  
void vTask1( void \*pvParameters )  
{  
 const char \*pcMessage = "Task 1 is running";  
 volatile unsigned long ul;  
  
 for( ;; )  
 {  
 vSendString( pcMessage );  
 for( ul = 0; ul < mainDELAY\_LOOP\_COUNT; ul++ ) { }  
 }  
}  
  
void vTask2( void \*pvParameters )  
{  
 const char \*pcMessage = "Task 2 is running";  
 volatile unsigned long ul;  
  
 for( ;; )  
 {  
 vSendString( pcMessage );  
 for( ul = 0; ul < mainDELAY\_LOOP\_COUNT; ul++ ) { }  
 }  
}  
  
  
int main\_example( void )  
{  
 xTaskCreate( vTask1, "Task 1", configMINIMAL\_STACK\_SIZE \* 2U, NULL,  
 exampleTASK\_PRIORITY, NULL );  
 xTaskCreate( vTask2, "Task 2", configMINIMAL\_STACK\_SIZE \* 2U, NULL,  
 exampleTASK\_PRIORITY + 1, NULL );  
  
 vTaskStartScheduler();  
  
 return 0;  
}

Для запуска примера, измените следующие строки в файле main.c:

* 33 — #define DEMO\_EXAMPLE 1
* добавьте объявление int main\_example( void );
* в функции main замените
* #if defined(DEMO\_BLINKY)  
   ret = main\_blinky();  
  #else  
  #error "Please add or select demo."  
  #endif
* на
* #if defined(DEMO\_BLINKY)  
   ret = main\_blinky();  
  #elif defined(DEMO\_EXAMPLE)  
   ret = main\_example();  
  #else  
  #error "Please add or select demo."  
  #endif

В файле Makefile в 35 строку добавьте example.c:

SRCS = main.c main\_blinky.c example.c riscv-virt.c ns16550.c \

Теперь приложение можно собрать и запустить аналогично предыдущему.

**Вывод**

Task 1 is running  
Task 2 is running  
Task 2 is running  
Task 1 is running  
Task 1 is running  
Task 2 is running  
Task 2 is running  
Task 1 is running  
Task 2 is running  
Task 1 is running  
Task 2 is running  
Task 1 is running  
Task 2 is running  
Task 1 is running  
Task 2 is running  
Task 1 is running  
Task 2 is running  
Task 1 is running  
Task 1 is running  
Task 2 is running

## Задание приоритетов

Посмотрите, что получится, если задачам задать разные приоритеты. Это можно сделать, поменяв в 47 строке exampleTASK\_PRIORITY на exampleTASK\_PRIORITY + 1.

При изменении приоритетов можно получить следующий вывод программы:

Task 2 is running  
Task 2 is running  
Task 2 is running  
Task 2 is running  
Task 2 is running  
Task 2 is running  
Task 2 is running  
Task 2 is running  
Task 2 is running  
Task 2 is running  
Task 2 is running  
Task 2 is running  
Task 2 is running  
Task 2 is running  
Task 2 is running  
Task 2 is running  
Task 2 is running  
Task 2 is running  
Task 2 is running  
Task 2 is running

*Всё время выполняется задача с большим приоритетом.*

Изменим реализацию задержки в задачах:

**Изменённый example.c**

#include <FreeRTOS.h>  
#include <task.h>  
#include <queue.h>  
  
#include <stdio.h>  
  
#include "riscv-virt.h"  
#include "ns16550.h"  
  
#define exampleDELAY pdMS\_TO\_TICKS( 1000 )  
#define exampleTASK\_PRIORITY ( tskIDLE\_PRIORITY + 1 )  
  
  
void vTask1( void \*pvParameters )  
{  
 const char \*pcMessage = "Task 1 is running";  
 TickType\_t xNextWakeTime;  
 xNextWakeTime = xTaskGetTickCount();  
  
 for( ;; )  
 {  
 vSendString( pcMessage );  
 vTaskDelayUntil( &xNextWakeTime, exampleDELAY );  
 }  
}  
  
void vTask2( void \*pvParameters )  
{  
 const char \*pcMessage = "Task 2 is running";  
 volatile unsigned long ul;  
  
 TickType\_t xNextWakeTime;  
 xNextWakeTime = xTaskGetTickCount();  
  
 for( ;; )  
 {  
 vSendString( pcMessage );  
 vTaskDelayUntil( &xNextWakeTime, exampleDELAY );  
 }  
}  
  
  
int main\_example( void )  
{  
 xTaskCreate( vTask1, "Task 1", configMINIMAL\_STACK\_SIZE \* 2U, NULL,  
 exampleTASK\_PRIORITY, NULL );  
 xTaskCreate( vTask2, "Task 2", configMINIMAL\_STACK\_SIZE \* 2U, NULL,  
 exampleTASK\_PRIORITY + 1, NULL );  
  
 vTaskStartScheduler();  
  
 return 0;  
}

Запустив изменённую версию, получим следующий вывод:

Task 2 is running  
Task 1 is running  
Task 2 is running  
Task 1 is running  
Task 2 is running  
Task 1 is running  
Task 2 is running  
Task 1 is running  
Task 2 is running  
Task 1 is running  
Task 2 is running  
Task 1 is running  
Task 2 is running  
Task 1 is running  
Task 2 is running  
Task 1 is running  
Task 2 is running  
Task 1 is running  
Task 2 is running  
Task 1 is running  
Task 2 is running  
Task 1 is running

Теперь задачи выполняются поочерёдно. Что изменилось?

Дело в том, что в первом случае задержка активная, то есть задача остаётся в состоянии running. При завершении цикла Task2 переходит в состояние ready, как и задача с меньшим приоритетом, и планировщик снова выбирает для выполнения задачу с большим приоритетом.

Во втором случае для задержки вызывается функция vTaskDelayUntil, переводящая задачу в состояние blocked, позволяя выполнится задаче с меньшим приоритетом.