Использование FreeRTOS в управляющей системе мобильного робота

СКОРИКОВ АНДРЕЙ, ВОЛГГТУ

Что такое ОС для МК?

Микроконтроллер работает в режиме реального времени, то есть время реакции микроконтроллерного устройства на внешнее событие должно быть строго меньше заданной величины и должно быть сопоставимо со скоростью протекания внешних процессов.

Таким образом, ОС для МК — это операционная система реального времени (ОСРВ).

Что такое ОС для МК?

Микроконтроллер - это однокристальный компьютер с сильно ограниченными аппаратными ресурсами.

Основные особенности МК:

- Низкая производительность.
- Малый объем ОЗУ и ПЗУ.
- Отсутствие блока управления памятью (Memory management unit, MMU), используемого большинством современных ОС, например, Windows и UNIX-подобными.
- Отсутствие аппаратных средств поддержки многозадачности (например, средств быстрого переключения контекста).

Что такое ОС для МК?

Микроконтроллер сам по себе предназначен для выполнения низкоуровневых задач, будь то опрос состояния кнопок, передача команды по I2C-интерфейсу или включение обмотки электромотора.

Программа для МК, как правило, обращается к периферии напрямую, программист имеет полный контроль над аппаратной частью, нет необходимости в посредниках между аппаратурой и прикладной программой.

Преимущества ОСРВ для МК

1. Многозадачность.
2. Временная база.
3. Обмен данными между задачами.
4. Синхронизация.
5. Переносимость.

Накладные расходы ОСРВ

1. Дополнительный расход памяти программ для хранения ядра ОСРВ.
2. Дополнительный расход памяти данных для хранения стека каждой задачи, семафоров, очередей, мьютексов и других объектов ядра операционной системы.
3. Дополнительные затраты времени процессора на переключение между задачами.

Обзор FreeRTOS

FreeRTOS — это многозадачная, мультиплатформенная, бесплатная операционная система жесткого реального времени с открытым исходным кодом. FreeRTOS была разработана компанией Real Time Engineers Ltd. специально для встраиваемых систем.

Бóльшая часть кода FreeRTOS написана на языке Си, ассемблерные вставки минимального объема применяются лишь там, где невозможно применить Си из-за специфики конкретной аппаратной платформы.

Основные характеристики FreeRTOS

1. Планировщик FreeRTOS поддерживает три типа многозадачности:
   – вытесняющую;
   – кооперативную;
   – гибридную.
2. Размер ядра FreeRTOS составляет всего 4–9 кбайт, в зависимости от типа платформы и настроек ядра.
3. FreeRTOS написана на языке Си (исходный код ядра представлен в виде всего лишь четырех Си-файлов).

Основные характеристики FreeRTOS

4. Поддерживает задачи (tasks) и сопрограммы (co-routines). Сопрограммы специально созданы для МК с малым объемом ОЗУ.
5. Богатые возможности трассировки.
6. Возможность отслеживать факт переполнения стека.
7. Нет программных ограничений на количество одновременно выполняемых задач.
8. Нет программных ограничений на количество приоритетов задач.

Основные характеристики FreeRTOS

9. Нет ограничений в использовании приоритетов: нескольким задачам может быть назначен одинаковый приоритет.
10. Развитые средства синхронизации «задача – задача» и «задача – прерывание»:

    – очереди;

    – двоичные семафоры;

    – счетные семафоры;

    – рекурсивные семафоры;

    – мьютексы.
11. Мьютексы с наследованием приоритета.

Основные характеристики FreeRTOS

12. Поддержка модуля защиты памяти (Memory protection unit, MPU) в процессорах Cortex-M3.
13. Поставляется с отлаженными примерами проектов для каждого порта и для каждой среды разработки.
14. FreeRTOS полностью бесплатна, модифицированная лицензия GPL позволяет использовать FreeRTOS в проектах без раскрытия исходных кодов.
15. Документация в виде отдельного документа платная, но на официальном сайте в режиме online доступно исчерпывающее техническое описание на английском языке.

С чего начать?

Начать разработку микроконтроллерного устройства, работающего под управлением FreeRTOS, можно с загрузки ее последней версии по адресу http://www.freertos.org

ВАЖНО!!!

Для ARM Cortex-M3, ARM Cortex-M4 необходимо установить обработчики на SysTick, PendSV и SVCCall векторы прерываний.

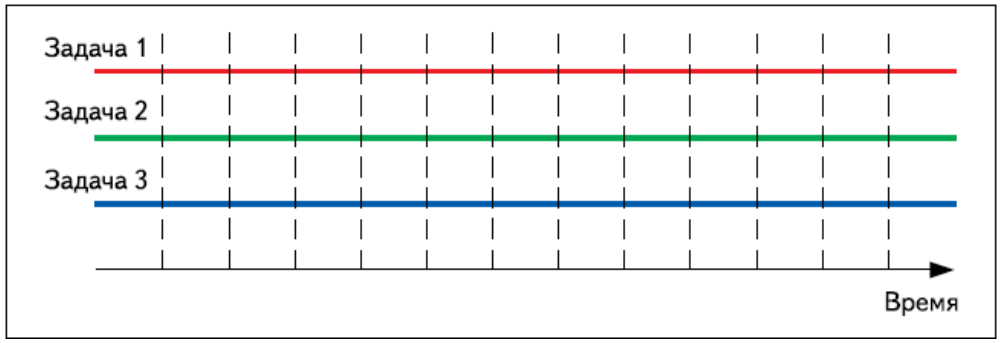
Для этого в файл FreeRTOSConfig.h необходимо добавить следующие строчки:

```
// Setup Interrupt Vectors
#define xPortSysTickHandler SysTick_Handler
#define xPortPendSVHandler PendSV_Handler
#define vPortSVCHandler SVC_Handler
```

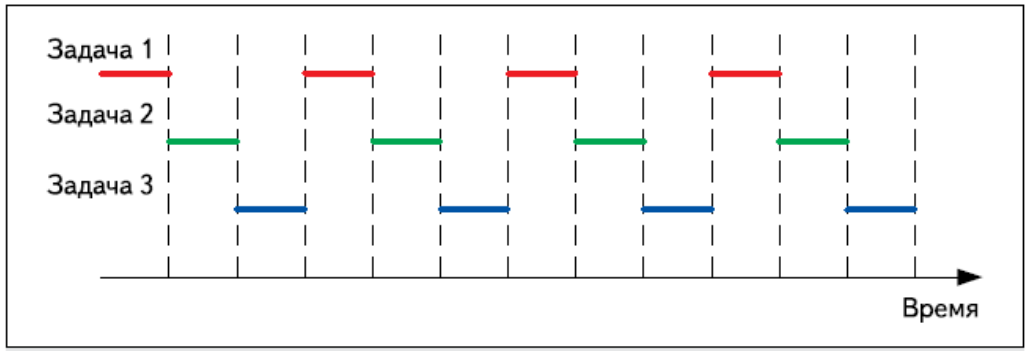
Основы работы ОСРВ

Основой ОСРВ является *ядро (Kernel)* операционной системы. Ядро реализует основополагающие функции любой ОС.

Каждая выполняющаяся программа представляет собой *задачу (Task).* Если ОС позволяет одновременно выполнять множество задач, она является *мультизадачной (Multitasking)*.



Истинно параллельное выполнение задач



Распределение процессорного времени между несколькими задачами в ОСРВ

Основы работы ОСРВ

Планировщик (Scheduler) — это часть ядра ОСРВ, которая определяет, какая из задач, готовых к выполнению, выполняется в данный конкретный момент времени. Планировщик может приостанавливать, а затем снова возобновлять выполнение задачи в течение всего ее жизненного цикла (то есть с момента создания задачи до момента ее уничтожения).

Алгоритм работы планировщика (Scheduling policy) — это алгоритм, по которому функционирует планировщик для принятия решения, какую задачу выполнять в данный момент времени.

Основы работы ОСРВ

Среди всех задач в системе в один момент времени может выполняться только одна задача. Говорят, что она находится в состоянии выполнения. Остальные задачи в этот момент не выполняются, ожидая, когда планировщик выделит каждой из них процессорное время. Таким образом, задача может находиться в двух основных состояниях: выполняться и не выполняться.

Основы работы ОСРВ

Кроме того, что выполнение задачи может быть приостановлено планировщиком принудительно, задача может сама приостановить свое выполнение. Это происходит в двух случаях. Первый — это когда задача «хочет» задержать свое выполнение на определенный промежуток времени (в таком случае она переходит в состояние сна (sleep)). Второй — когда задача ожидает освобождения какого-либо аппаратного ресурса (например, последовательного порта) или наступления какого-то события (event), в этом случае говорят, что задача блокирована (block). Блокированная или «спящая» задача не нуждается в процессорном времени до наступления соответствующего события или истечения определенного интервала времени. Функции измерения интервалов времени и обслуживания событий берет на себя ядро ОСРВ.

Основы работы ОСРВ

Когда задача выполняется, она, как и любая программа, использует регистры процессора, память программ и память данных. Вместе эти ресурсы (регистры, стек и др.) образуют *контекст задачи (task execution context)*.

Одна из основных функций ядра ОСРВ — это обеспечение идентичности контекста задачи до ее приостановки и после ее восстановления. Когда ядро приостанавливает задачу, оно должно сохранить контекст задачи, а при ее восстановлении — восстановить. Процесс сохранения и восстановления контекста задачи называется *переключением контекста (context switching).*

Основы работы ОСРВ

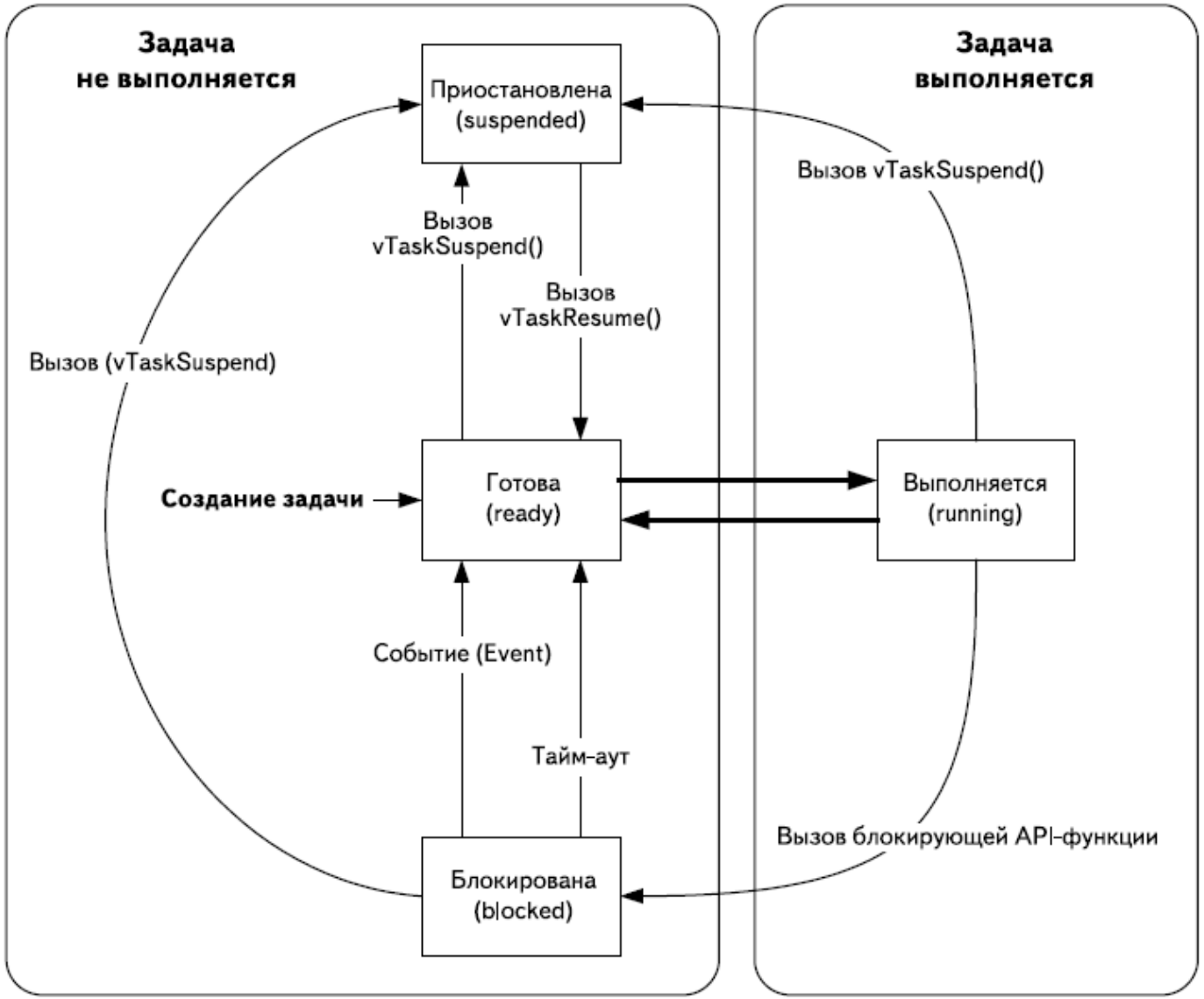
Немаловажным понятием является квант времени работы планировщика — *системный квант (tick)* — это жестко фиксированный отрезок времени, в течение которого планировщик не вмешивается в выполнение задачи. По истечении кванта времени планировщик получает возможность приостановить текущую задачу и возобновить следующую, готовую к выполнению. Для отсчета системных квантов в МК обычно используется прерывание от таймера/счетчика. Системный квант используется как единица измерения интервалов времени средствами ОСРВ.

Задачи

Любая программа, которая выполняется под управлением FreeRTOS, представляет собой множество отдельных независимых задач.

Каждая задача имеет свой собственный стек. При смене задачи ее контекст сохраняется в ее собственном стеке, что позволяет восстановить контекст при возобновлении задачи.

Как было сказано выше, при грубом приближении задача может находиться в двух состояниях: выполняться и не выполняться. При подробном рассмотрении состояние «задача не выполняется» подразделяется на несколько различных состояний в зависимости от того, как она была остановлена.



Состояния задачи в FreeRTOS

Задачи

Для обеспечения преимущества на выполнение более ответственных задач во FreeRTOS применяется механизм приоритетов задач (Task priorities).

Среди всех задач, находящихся в состоянии готовности, планировщик отдаст управление той задаче, которая имеет наивысший приоритет. Задача будет выполняться до тех пор, пока она не будет блокирована или приостановлена или пока не появится готовая к выполнению задача с более высоким приоритетом.

Задачи

```
void ATaskFunction( void *pvParameters )
{

      /* Переменные могут быть объявлены здесь, как и в обычной функции.
     Каждый экземпляр этой задачи будет иметь свою собственную копию
     переменной iVariableExample. Если объявить переменную со
     спецификатором static, то будет создана только одна переменная
     iVariableExample, доступная из всех экземпляров задачи */

      int iVariableExample = 0;

      /* Тело задачи реализовано как бесконечный цикл */
      for( ;; )
      {
          /* Код, реализующий функциональность задачи */
      }

      /* Если все-таки произойдет выход из бесконечного цикла, то задача
     должна быть уничтожена ДО конца функции. Параметр NULL обозначает,
     что       уничтожается задача, вызывающая API-функцию vTaskDelete()
     */

     vTaskDelete( NULL );

}
```

Задачи

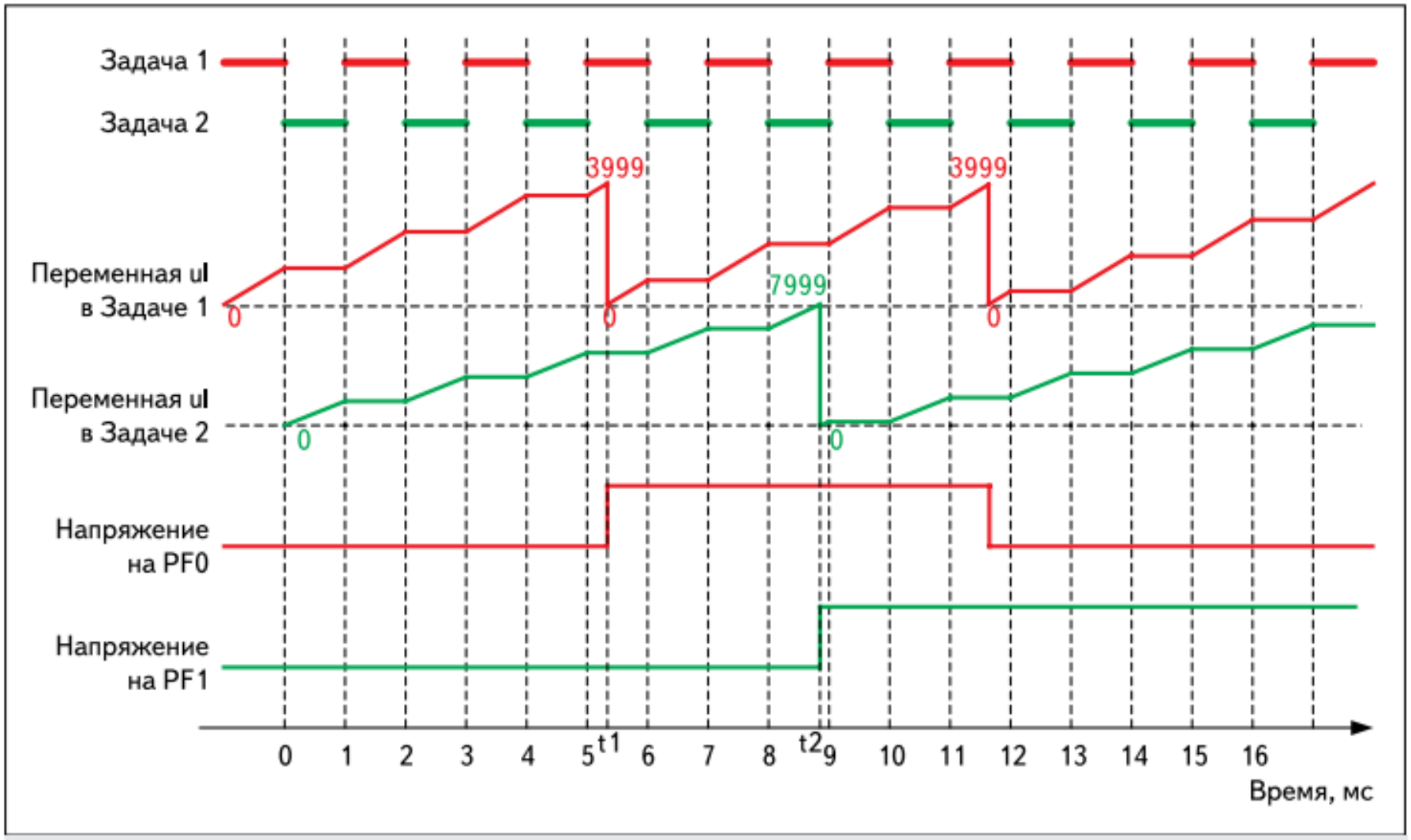
```
portBASE_TYPE xTaskCreate(pdTASK_CODE pvTaskCode,
                          const signed portCHAR * const pcName,
                          unsigned portSHORT usStackDepth,
                          void *pvParameters,
                          unsigned portBASE_TYPE uxPriority,
                          xTaskHandle *pxCreatedTask
);


void vTaskStartScheduler()
```

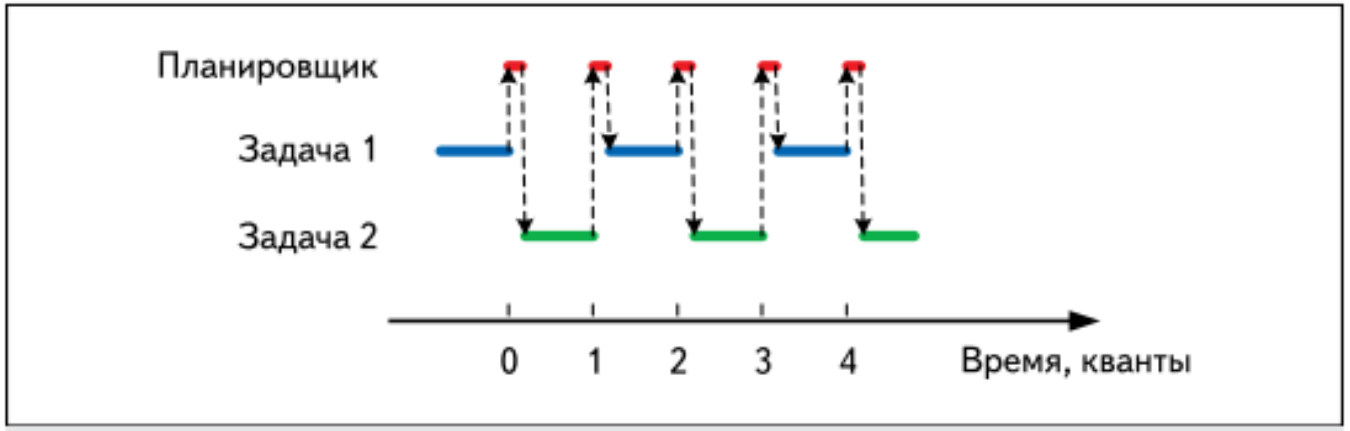
Первая программа

Программа будет содержать две задачи.

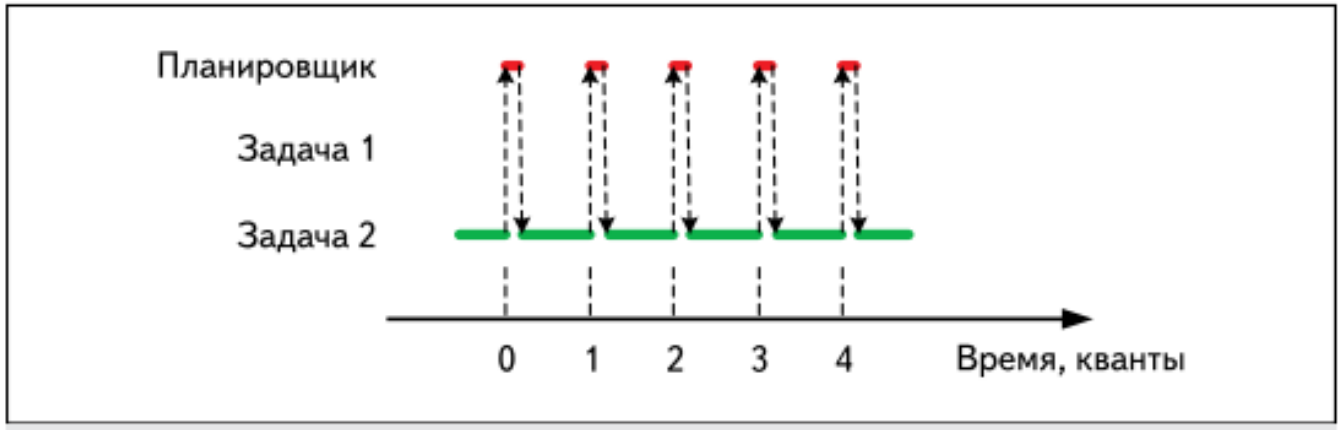
Задача 1 будет переключать логический уровень на одном выводе МК, задача 2 — на другом. Частота переключения для разных выводов будет разной.



Примерная работа программы во времени



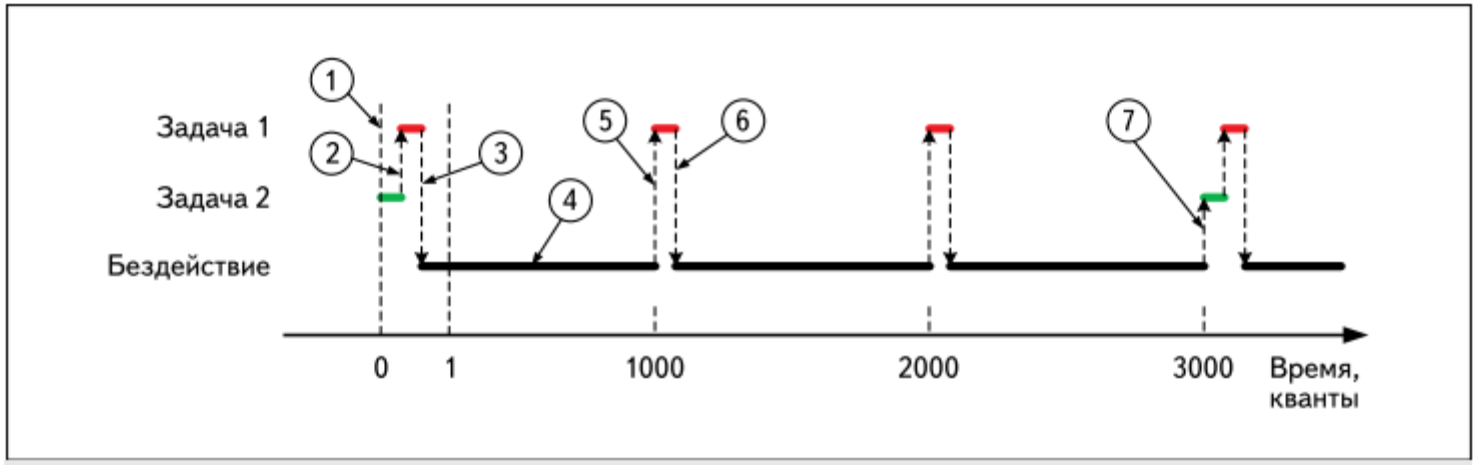
Разделение процессорного времени между задачами



Разделение времени между задачами, когда Задача 2 имеет более высокий приоритет, чем Задача 1

Реализация задержек

```
void vTaskDelay( portTickType xTicksToDelay );
```

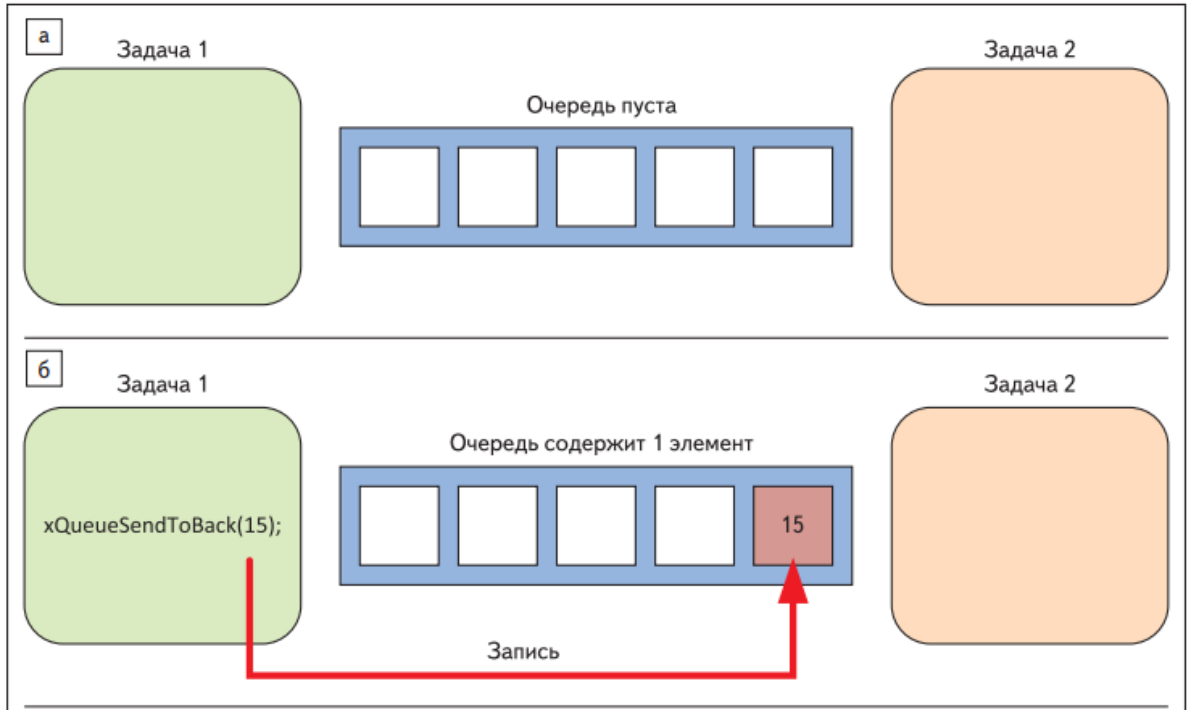


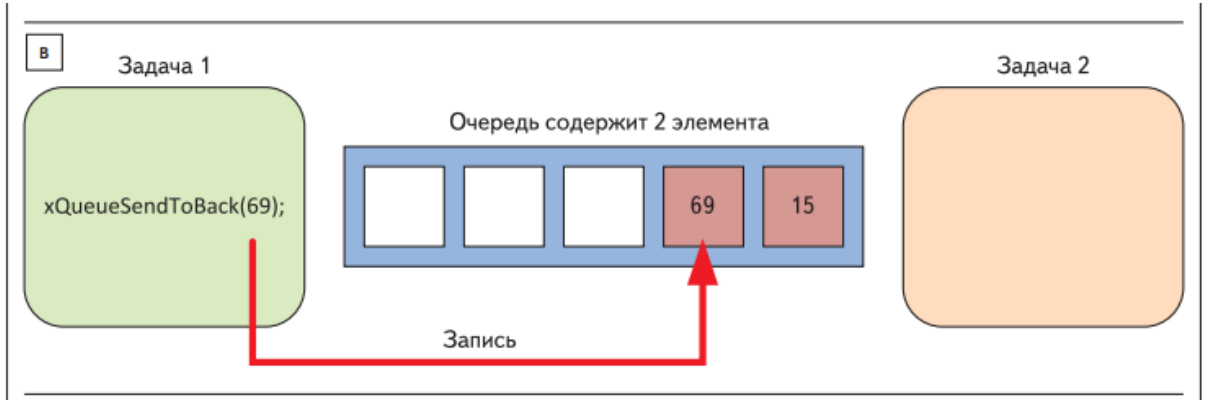

```
void vTaskDelayUntil( portTickType
*pxPreviousWakeTime, portTickType xTimeIncrement );
```

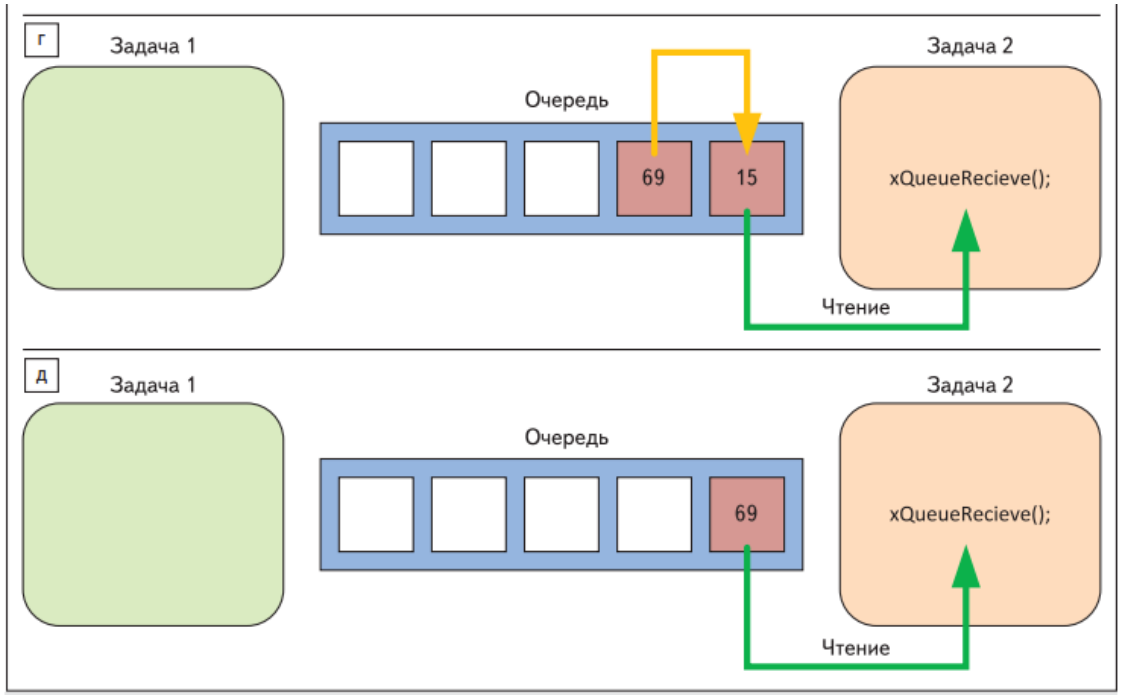
Очереди

Во FreeRTOS очереди представляют собой фундаментальный механизм взаимодействия задач друг с другом. Они могут быть использованы для передачи информации как между задачами, так и между прерываниями и задачами.

Основное преимущество использования очередей — это то, что их использование является безопасным в многозадачной среде (thread safe). То есть при использовании очередей автоматически решается проблема совместного доступа нескольких задач к одному аппаратному ресурсу, роль которого в данном случае играет память.







Запись и чтение элементов из очереди по принципу FIFO

Очереди

Очередь — это самостоятельный объект ядра, она не принадлежит ни одной конкретной задаче. Напротив, любое количество задач могут как читать, так и записывать данные в одну и ту же очередь.

Когда задача пытается прочитать данные из очереди, которая не содержит ни одного элемента, то задача переходит в блокированное состояние. Такая задача вернется в состояние готовности к выполнению, если другая задача (или прерывание) поместит данные в очередь или по истечению тайм-аута.

Задача может находиться в блокированном состоянии, ожидая возможность записи в очередь. Это происходит, когда очередь полностью заполнена и в ней нет свободного места для записи нового элемента данных.

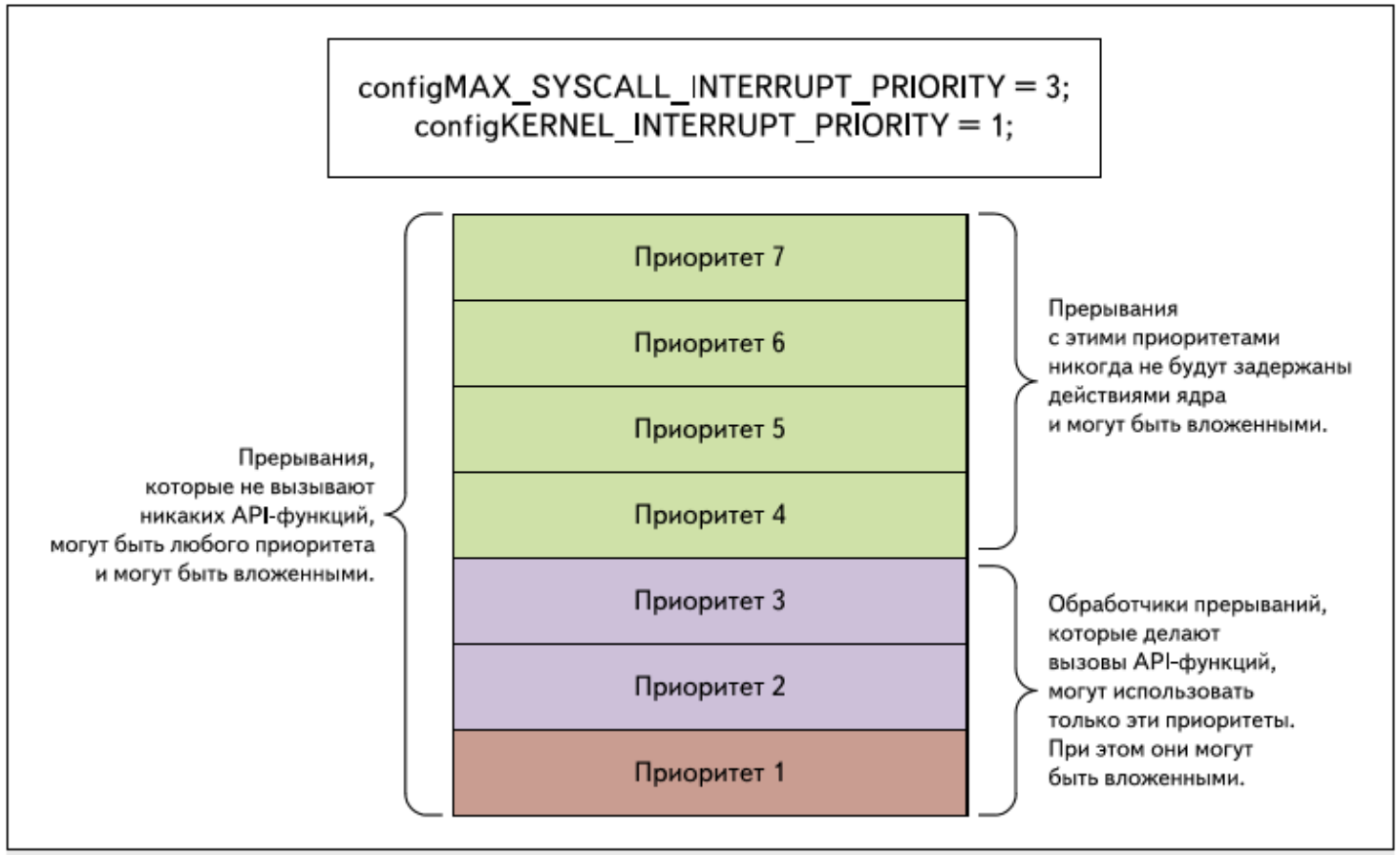
Очереди

```
xQueueHandle xQueueCreate(
                          unsigned portBASE_TYPE uxQueueLength,
                            unsigned portBASE_TYPE xItemSize );

portBASE_TYPE xQueueSend (xQueueHandle xQueue,
                                      const void * pvItemToQueue,
                                     portTickType xTicksToWait );

portBASE_TYPE xQueueReceive(xQueueHandle xQueue,
                                    const void * pvBuffer,
                                    portTickType xTicksToWait);
```

Прерывания


configMAX_SYSCALL_INTERRUPT_PRIORITY = 3;
configKERNEL_INTERRUPT_PRIORITY = 1;
Приоритет 7
Приоритет 6
Приоритет 5
Приоритет 4
Приоритет 3
Приоритет 2
Приоритет 1
Прерывания, которые не вызывают никаких API-функций, могут быть любого приоритета и могут быть вложенными.
Прерывания с этими приоритетами никогда не будут задержаны действиями ядра и могут быть вложенными.
Обработчики прерываний, которые делают вызовы API-функций, могут использовать только эти приоритеты. При этом они могут быть вложенными.

Прерывания

Приоритет в STM32 имеет 4 бита (СТАРШИЕ 4 БИТА), часть из них отводится на preemption ("нормальный приоритет") и часть на подгруппу. Приоритет группы определяет, может ли прерывание одной группы прерывать прерывание другой. Приоритет подгруппы определяет, порядок обработки одновременных прерываний, но прерывать друг друга внутри группы они не могут.

group0 0 bits for preemption, 4 bits for sub priority

group1 1 bits for preemption, 3 bits for sub priority

group2 2 bits for preemption, 2 bits for sub priority

group3 3 bits for preemption, 1 bits for sub priority

group4 4 bits for preemption, 0 bits for sub priority

Необходимо использовать группу 4 для FreeRTOS

http://www.freertos.org/RTOS-Cortex-M3-M4.html

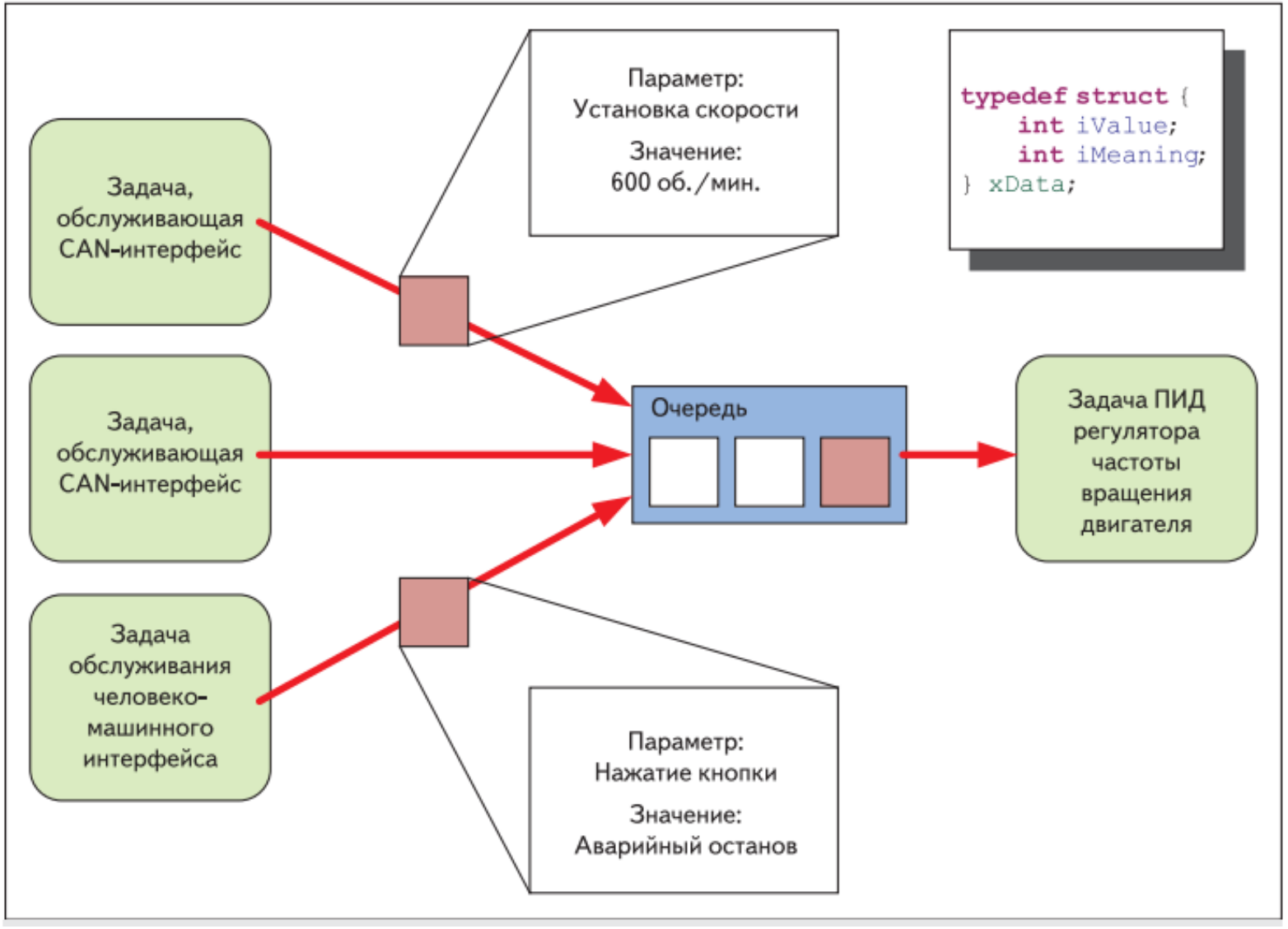
NVIC_PriorityGroupConfig(NVIC_PriorityGroup_4);

Вторая программа

Необходимо с высокоскоростного интерфейса получать данные и передавать на низкоскоростной интерфейс (например, CAN – USART, Bluetooth-USART, USART-SPI).

Решение: получить с высокоскоростного интерфейса порцию данных, поместить в очередь; когда низкоскоростной интерфейс будет готов, он возьмет данные из очереди и выполнит необходимые действия.

Для примера будем реализовывать передачу с USART2, работающего на частоте 115 200 бод/с, на USART1, работающий на частоте 9600 бод/с.



Пример организации обмена информацией между задачами

Полезные источники

1. Курниц А. FreeRTOS — операционная система для микроконтроллеров //Компоненты и технологии. 2011. № 2–10.

2. www.freertos.org