Программная реализация МПС

Тема 3

FreeRTOS

Философия разработки

FreeRTOS разработана как:

- Простая
- Портируемая
-Маленькая

Система FreeRTOS находится в стадии активной разработки, которая была начата Ричардом Барри (Richard Barry) в 2002 году.

Некоторые возможности FreeRTOS

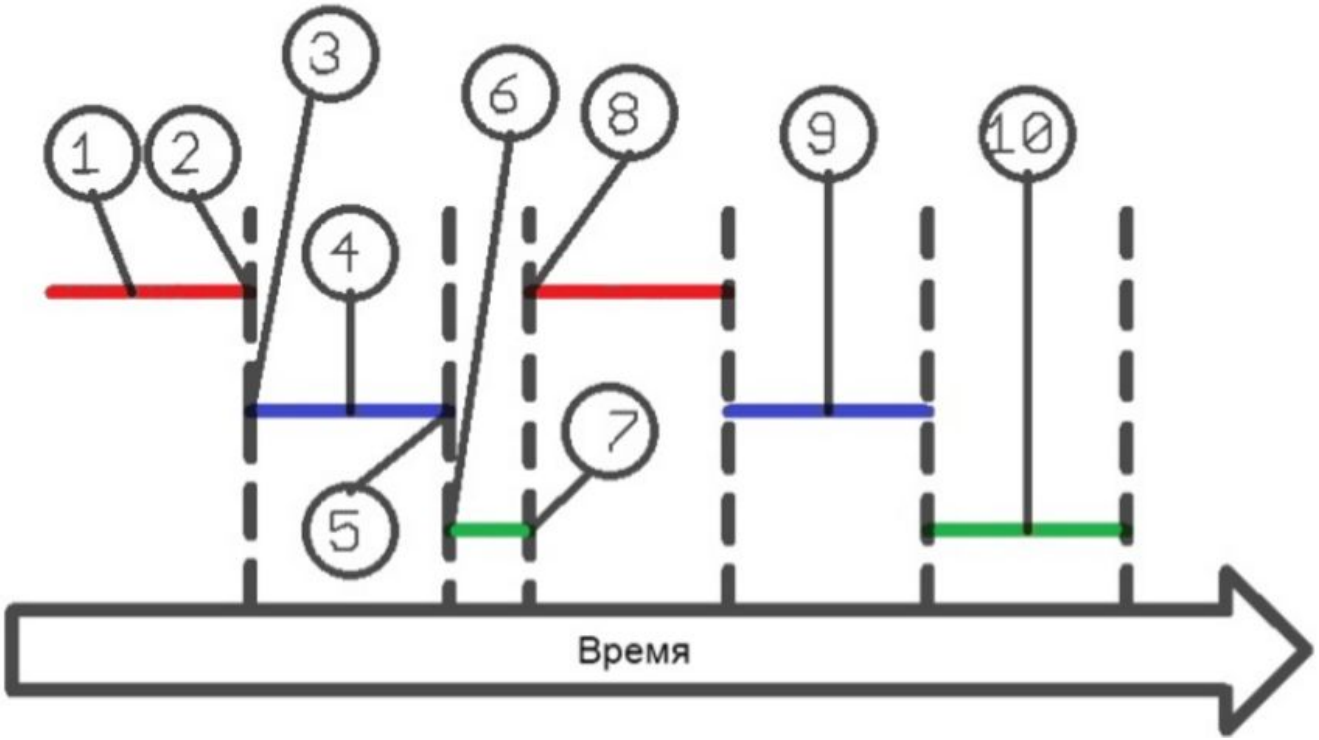
- Выбор политики планирования
- Всегда работает задача с наивысшим приоритетом из доступных. Задачи с одинаковым приоритетом делят процессорное время.
- Coroutines(сопрограммы) - маленькие задачи, использующие очень мало RAM.
- БОльшая часть кода одинакова для всех средств разработки.
- Много портов и примеров.
- Дополнительные возможности могут быть легко и быстро добавлены.

- **Многозадачность и параллельность**

Обычный процессор может выполнять только одну задачу одновременно - но за счёт быстрого переключения между задачами делается вид, будто задачи выполняются одновременно.

- **Планирование**

Планировщик - это часть ядра, отвечающий за то, какая задача должна выполняться в какой момент времени. Ядро может приостановить и потом продолжить задачу несколько раз за время работы задачи.



Задачи

Задачей является определяемая пользователем функция на языке C с заданным приоритетом.
В tasks.c и task.h делается вся тяжелая работа по созданию, планированию и обслуживанию задач.

☺	Простая
☺	Нет ограничений использования
☺	Поддерживает полное вытеснение
☺	Приоритет задаётся всей задаче
☹	Каждая задача требует отдельного стека - это приводит к большому потреблению оперативной памяти
☹	Повторная входимость должна быть тщательно продумана, если используется вытеснение

Сопрограммы

Сопрограммы концептуально схожи с задачами, но имеют некоторые различия.

☺	Требуется меньше оперативной памяти за счёт общего стека
☺	Совместная работа упрощает проблему повторного входа.
☺	Хорошо переносимо между разными архитектурами
😐	Полностью приоритезована по отношению к другим сопрограммам, но всегда может быть вытеснена задачей, имеющей больший приоритет, чем задача, запускающая сопрограммы
☹	Отсутствие индивидуального стека требует особого рассмотрения
☹	Ограниченный API
☹	Кооперативная многозадачность только между сопрограммами.

Очереди

Система FreeRTOS позволяет задачам с помощью очередей общаться и синхронизироваться друг с другом. Процедуры сервиса прерываний (ISR) также используют очереди для взаимодействий и синхронизации.

Семафоры

Механизм семафоров основан на механизме очередей. По большому счету API-функции для работы с семафорами представляют собой макросы — «обертки» других API-функций для работы с очередями.

Семафор должен быть явно создан перед первым его использованием.

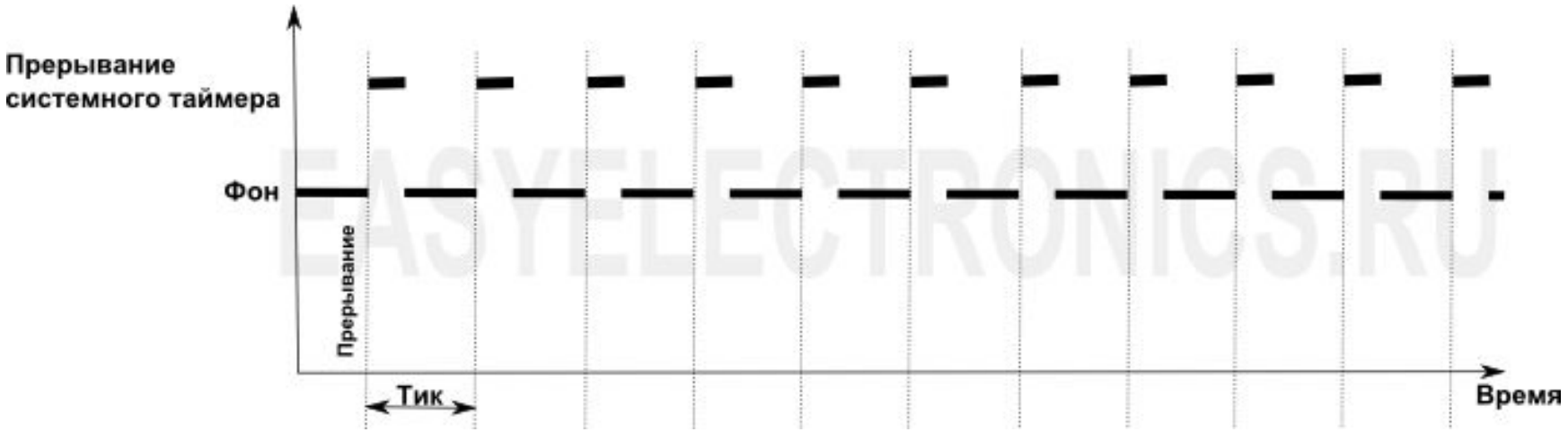
Мьютекс

2. Мьютекс используется для защиты общего ресурса. Задача включает мьютекс, использует общий ресурс, а затем отключает мьютекс. Никакая задача не может включить мьютекс, пока он включен другой задачей.

2. Мьютекс во FreeRTOS представляет собой специальный тип двоичного семафора, который используется для реализации совместного доступа к ресурсу двух или большего числа задач.

Системный тик

Один из таймеров микроконтроллера, самый ненужный, настраивают на генерацию системных тиков. Один тик делается, обычно, раз в 1мс, но можно и чаще или реже. В зависимости от того какая реакция и дискретность системы нам нужна.



Каждый тик это вызов прерывания таймера, в котором вызывается диспетчер, чьими усилиями проворачиваются шестеренки ОС. Это время еще называют квантом, говоря, что задаче выделяется квант времени.

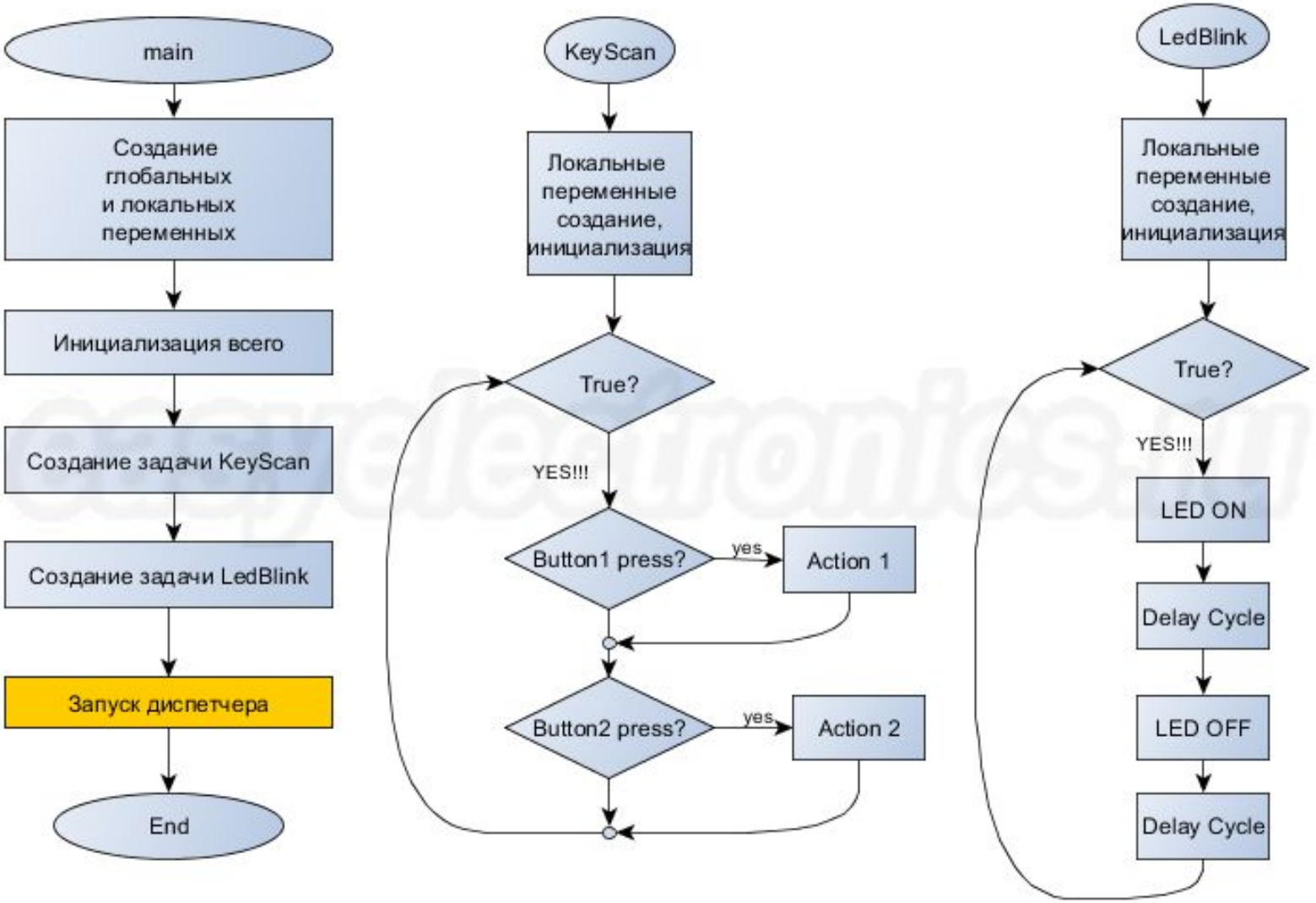
Задача

Краеугольным камнем любой RTOS является задача. Задача выглядит как функция которая крутит бесконечный цикл делающий какую-либо относительно простую процедуру.

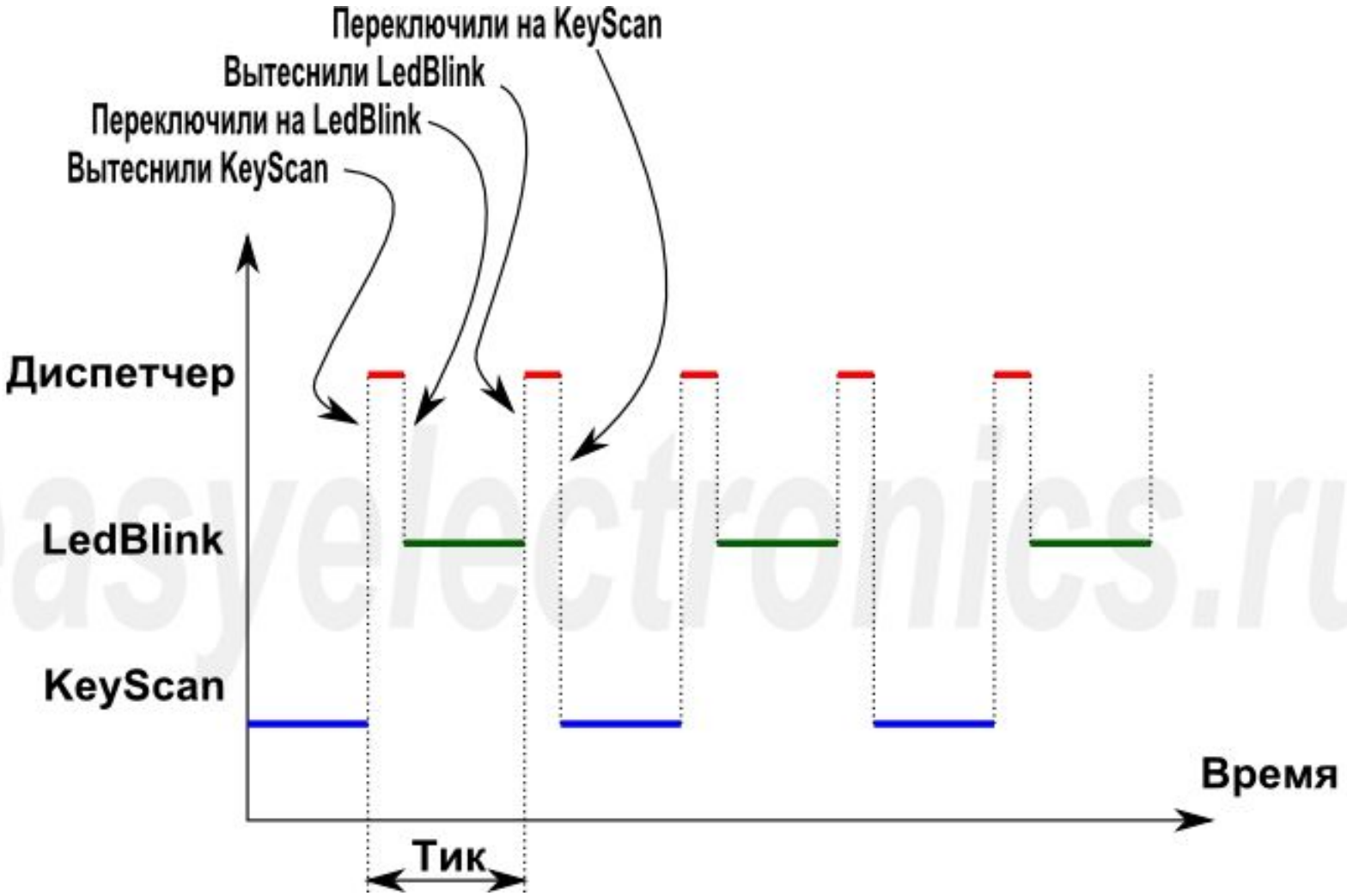
```
void KeyScan(void)
{
        while(1)
        {
         if (Button1 == Pressed)
                {
                do_action_1();
                }

        if (Button2 == Pressed)
                {
                do_action_2();
                }
        }
}
```

```
void LedBlink(void)
{
        while(1)
        {
         LED_ON();                  // Зажечь диод

        for (i=0;i<1000000;i++)     // Выдержка в 1000000 тактов.
                {
                _NOP();
                }

         LED_OFF();                 // Погасить диод

        for (i=0;i<1000000;i++)     // Выдержка в 1000000 тактов.
                {
                _NOP();
                }
        }
}
```

Алгоритм этой программы при реализации ее средствами ОС.
Связей между задачами нет, они работают каждая сама по себе.



Алгоритм работы диспетчера задач.



Объявление задач

- Задача должна иметь следующую структуру:

```
void vATaskFunction( void *pvParameters )
{
        for ( ; ; )
        {  /* Код задачи пишется тут */ }
}
```

Функции-задачи никогда не прерываются, поэтому обычно реализуются при помощи непрерывного цикла. Опять-же можно посмотреть примеры.

Задачи создаются с помощью xTaskCreate() и удаляются с помощью vTaskDelete()

Допустимые состояния «задачи»

READY Задача запущена и готова принять на себя управление. Ждет только момента когда на нее обратит внимание диспетчер. Как только дойдет ее очередь так сразу же задача перейдет в режим RUN.
RUN Т.е. диспетчер переключил управление на нее, процессор прогоняет непосредственно ее код через себя в данный момент. В этот момент задача живет, потребляет процессорное время и делает полезную работу ради которой она была записана.
WAIT Задача в спячке. Т.к. ждет некого события, например, значения таймера, или пока что-нибудь в системе не случится, на что эта задача должна среагировать. При этом диспетчер не переключается на нее, процессорное время не тратится. Как только ожидаемое событие произойдет, то RTOS назначит этой задаче состояние READY.
SUSPEND Выключено. Т.е. задача не выгружена из памяти, данные ее все сохранены, но она неактивна. Ни на какие события не реагирует и сама из этого состояния не выйдет. Вывести ее из этого состояния можно только API командой ОС, вручную.

Допустимые состояния «задачи»

Приоритеты «задач»

У задачи есть такой важный параметр как приоритет. Он задается при создании и его можно на лету вручную менять через API функции RTOS .

Приоритет определяет в каком порядке будут работать задачи.

Т.е. если есть две задачи в статусе Ready, но у одной приоритет выше другой. Задача с низким приоритетом в таком случае не получит управление до тех пор, пока высокоприоритетная задача не свалится в WAIT. Диспетчер всегда будет выбирать ту READY задачу у которой приоритет выше.

А если READY задач нет, то будет вращать IDLE цикл. В котором происходит обслуживание памяти, зачистка неиспользованной оперативки, удаление ошметков от удаленных задач и прочей служебной фигней. Ну и туда же (на IDLE) можно повесить свою callback функцию, в которой, например, контроллер будет отправляться в режим энергосбережения.

API функции управления задачами

xTaskCreate — создает новую задачу, выделяя под нее память и натравливая на нее диспетчер.

vTaskDelete — удаляет задачу. Память потом освобождает IDLE задача.

vTaskDelay(N) — эта функция вызывает диспетчер, который переводит задачу в WAIT на N системных тиков. Можно на ней лепить всякие простые задержки, вроде опроса кнопок.

vTaskDelayUntil(N) — функция аналогичная предыдущей, но считает время N не от момента ее срабатывания, а от момента прошлого пробуждения задачи.

uxTaskPriorityGet — возвращает приоритет задачи. Т.е. можно посмотреть приоритет текущей или любой другой задачи заголовок (handle) которой мы знаем.

API функции управления задачами

uxTaskPrioritySet — устанавливает приоритет задачи. Т.е. можно приоритет менять.

vTaskSuspend — глушит задачу, что она перестает отвечать на события. Перестает работать, но не выгружается из памяти, а зависает в текущем состоянии.

vTaskResume — возврат задачи из SUSPEND состояния. Эту функцию нельзя выполнять из обработчика прерывания.

vTaskResumeFromISR — аналогичная команда, но ее как раз можно выполнять из обработчика прерывания, но нельзя запускать вне него. Там еще есть ряд особенностей, о которых я расскажу ниже отдельно, когда буду описывать все *FromISR функции оптом.

Утилиты задач.
Не используются для управления задачами, но позволяют получить из диспетчера некоторые сведения.

xTaskGetCurrentTaskHandle — узнать Handle текущей задачи. Зная заголовок можно можно менять ее приоритет, запускать, удалять и так далее.

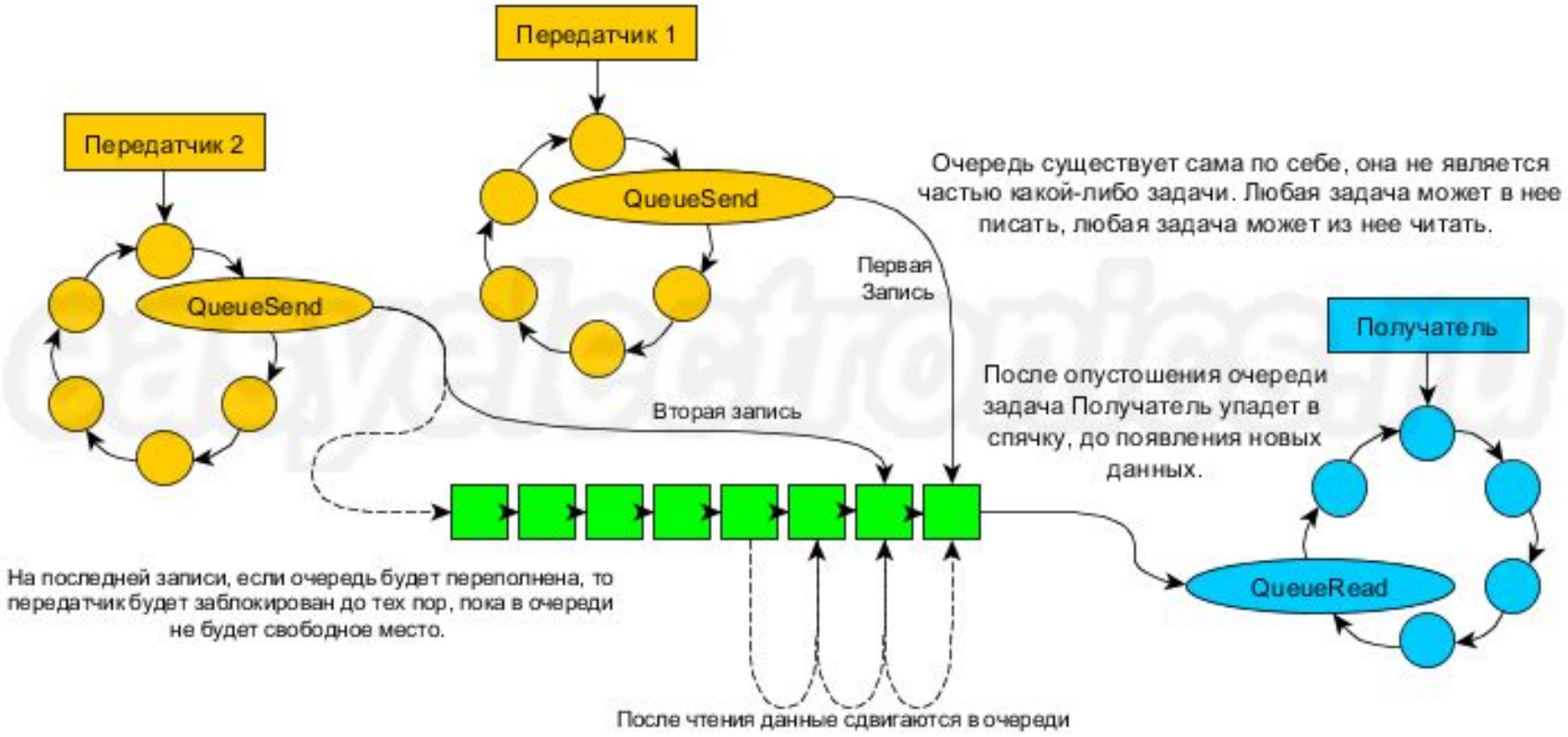
xTaskGetTickCount — выдает количество тиков с момента запуска планировщика. Это этакий глобальный таймер, отсчитывающий время с начала времен.

xTaskGetSchedulerState — выдает состояние диспетчера. Запущен, работает, выключен и так далее.

uxTaskGetNumberOfTasks — показывает количество загруженных задач. Например если надо определить хватит ли памяти. Или для отладки.

Очереди. Обмен данными между задачами

Одни задачи помещают данные в очередь, а другие оттуда читают



В случае если очередь пуста/переполнена, то та задача которая хочет считать/записать в очередь сваливается в WAIT и диспетчер ее разбудит когда очередь будет готова отдать/принять данные.