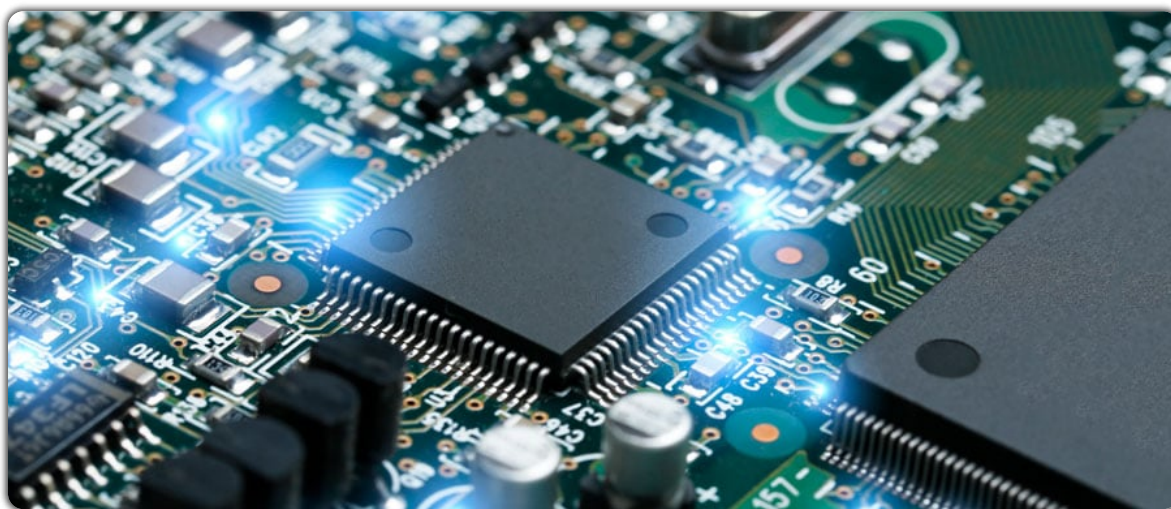




## STM32CubeMx. Быстрый старт с FreeRTOS для STM32.

👤 Aveal | 📅 20 августа, 2020 | 💬 4 комментариев



Давно не было статей с использованием [FreeRTOS](#) на нашем сайте. Что еще более удивительно, если учесть, что в повседневной жизни эта ОС используется регулярно. Так что сегодня без лишних слов и предисловий создадим базовый пример с поддержкой **FreeRTOS** для **STM32**. Прошли те времена, когда для включения ОС в свой проект приходилось перетаскивать кучу файлов, некоторые из которых оказывались несовместимы, некоторых просто не хватало... В STM32CubeMx все намного менее затратно и делается в пару кликов. В общем, приступаем!

При работе с FreeRTOS я чаще всего придерживаюсь следующей схемы. Создаются несколько задач (task'ов), каждая из которых вызывается через равные промежутки времени со своим собственным периодом, например:

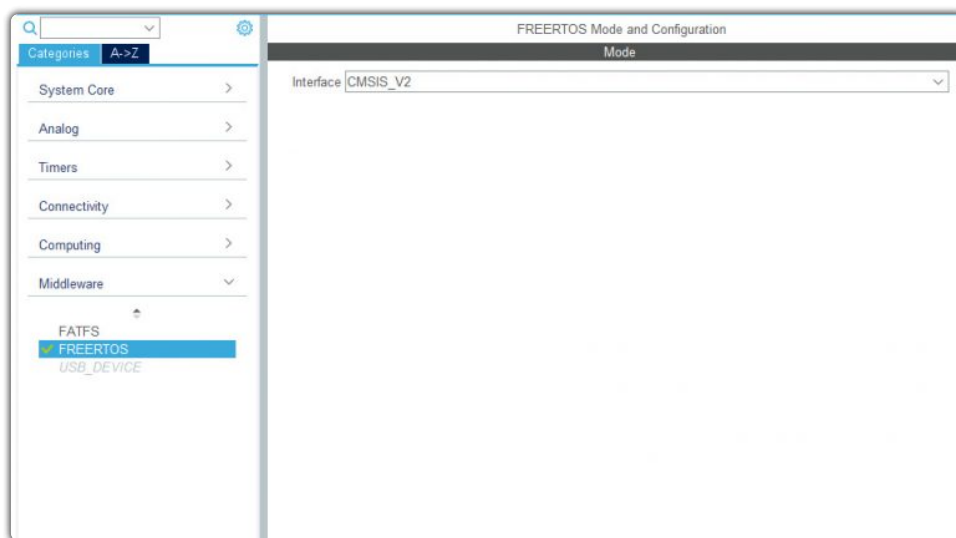
- 1 мс
- 10 мс
- 50 мс
- 100 мс

Конкретные значения могут зависеть уже от конкретных целей конкретного проекта. И далее вся работа распределяется по task'am. Соответственно, те действия, которые необходимо выполнять максимально часто вызываются из задачи, период выполнения которой равен 1 мс. Например, сохранение значений АЦП для последующей обработки. Другие же действия напротив выполняются намного реже, к примеру, обновлять информацию на дисплее. И в итоге вся программа распределяется по этим временным уровням.

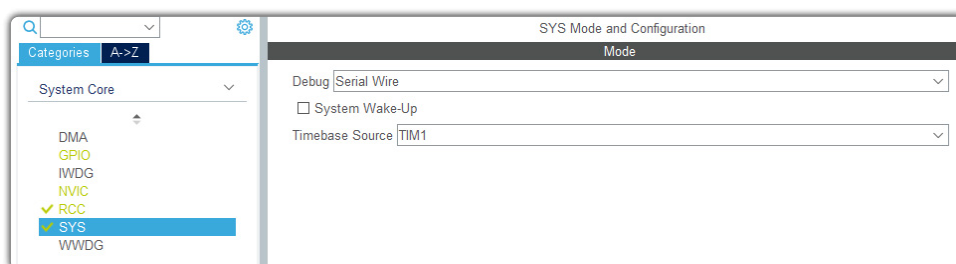
аваться каждые 1 мс, 10 мс и 30 мс. По аналогии можно будет легко и быстро добавить и другие.

Запускаем **STM32CubeMx**. Сразу уточню – не будем подробно погружаться во все нюансы настройки FreeRTOS и сматривать каждую конкретную опцию, иначе получится не статья, а книга ☐ Максимально быстрый старт! Если возникнут те-либо вопросы, смело задавайте их в комментариях или на [форуме](#), я буду рад помочь!

Итак, первым делом активируем FreeRTOS:



Но тут сразу же есть важный нюанс. При использовании FreeRTOS, в качестве базового таймера для HAL рекомендуется рать не *SysTick* (стоит по умолчанию), а другой. Для этого переходим в категорию *SYS* и меняем *SysTick* на один из свободных меров:

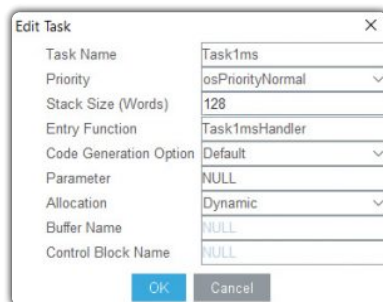


Поскольку мы уже условились делать базовый проект, то оставляем на этом этапе все настройки FreeRTOS без изменений:





Но добавляем наши задачи в разделе *Tasks and Queues* и задаем их приоритет – *Normal*:



Полностью аналогичным образом добавляем остальные задачи:

Task Name	Priority	Stack Size	Entry Function	Code Generation	Parameter	Allocation	Buffer Name	Control Blo...
defaultTask	osPriorityNormal	128	StartDefaultTask	Default	NULL	Dynamic	NULL	NULL
Task1ms	osPriorityNormal	128	Task1msHandler	Default	NULL	Dynamic	NULL	NULL
Task10ms	osPriorityNormal	128	Task10msHandler	Default	NULL	Dynamic	NULL	NULL
Task50ms	osPriorityNormal	128	Task50msHandler	Default	NULL	Dynamic	NULL	NULL

Все готово, генерируем, открываем и собираем проект! Видим, что Cube создал наши task'и:

```

1.  /* creation of Task1ms */
2.  Task1msHandle = osThreadNew(Task1msHandler, NULL, &Task1ms_attributes);
3.
4.  /* creation of Task10ms */
5.  Task10msHandle = osThreadNew(Task10msHandler, NULL, &Task10ms_attributes);
6.
7.  /* creation of Task50ms */
8.  Task50msHandle = osThreadNew(Task50msHandler, NULL, &Task50ms_attributes);

```

Но сейчас единственное, что связывает эти функции с нашим планом вызывать их через равные промежутки времени – это название. Так что необходимо доработать непосредственно код функций. И для того, чтобы обеспечить периодичность оления task'ов мы будем использовать функцию:

```

1.  void vTaskDelayUntil( TickType_t * const pxPreviousWakeTime, const TickType_t xTimeIncrement)

```

- Первый аргумент хранит значение времени, соответствующее моменту, когда задача была разблокирована в предыдущий раз. При первом вызове функции `vTaskDelayUntil()` необходимо инициализировать эту переменную текущим значением времени, а в дальнейшем функция сама будет обновлять это значение.

до момента времени, равного  $(\mu\text{t} \cdot \text{frequency})$  к моменту времени).

Теперь реализуем все на практике. Подключаем:

```
1. #include "task.h"
```

Для task'a, который работает с периодом 10 мс получаем следующее:

```
1. void Task10msHandler(void *argument)
2. {
3.     /* USER CODE BEGIN Task10msHandler */
4.     TickType_t xLastWakeTime;
5.     const TickType_t xFrequency = 10 / portTICK_PERIOD_MS;
6.
7.     xLastWakeTime = xTaskGetTickCount();
8.     /* Infinite loop */
9.     for(;;)
10.    {
11.        // Add code here
12.
13.        vTaskDelayUntil(&xLastWakeTime, xFrequency);
14.    }
15.    /* USER CODE END Task10msHandler */
16. }
```

А непосредственно свой код мы добавляем перед вызовом `vTaskDelayUntil()`, внутри цикла `for(;;)`. Здесь значение 10 мс зется в строке:

```
1. const TickType_t xFrequency = 10 / portTICK_PERIOD_MS;
```

Абсолютно аналогично делаем и для других наших задач, меняя только значение периода. Кроме того, давайте добавим гчики, которые будут инкрементироваться при вызове каждого из task'ов:

```
1. /* USER CODE BEGIN PV */
2. uint32_t task1msCnt = 0;
3. uint32_t task10msCnt = 0;
4. uint32_t task50msCnt = 0;
5.
6. /* USER CODE END PV */
```

И итоговый код, например, для task'a 50 мс:

```
1. void Task50msHandler(void *argument)
2. {
3.     /* USER CODE BEGIN Task50msHandler */
4.     TickType_t xLastWakeTime;
5.     const TickType_t xFrequency = 50 / portTICK_PERIOD_MS;
6.
7.     xLastWakeTime = xTaskGetTickCount();
8.     /* Infinite loop */
9.     for(;;)
10.    {
11.        // Add code here
12.        task50msCnt++;
13.
14.        vTaskDelayUntil(&xLastWakeTime, xFrequency);
15.    }
16.    /* USER CODE END Task50msHandler */
17. }
```

Вот такой механизм для организации периодических задач в FreeRTOS на контроллере STM32. Давайте соберем проект и верим, как все это работает. Индикацией для нас будут служить счетчики вызовов task'ов:

Expression	Value
task1msCnt	19895
task10msCnt	1990
task50msCnt	398
<click to add>	

Все отработывает четко по плану, задачи вызываются с заданной нами периодичностью. И на этом заканчиваем нашу юю, максимально **быстрый старт с FreeRTOS для STM32!** А в будущих статьях уже будем изучать работу с ОС более робно!

Ссылка на полный проект – [MT\\_FreeRTOS\\_Base](#).

Поделиться!


☒ Подписаться ▼Соединить с  

Присоединиться к обсуждению

**B** *I* U        



4 КОММЕНТАРИЕВ

  старее ▼

Srl

 1 месяц назад

Хорошая статья. Спасибо. Ждем новых статей по этой теме..



2

 Ответить**Aveal**

Автор

 Reply to Srl 1 месяц назад

Благодарю!



0

 Ответить

Денис

 2 дней назад

Отличная статья! Расскажи, пожалуйста, побольше про работу FreeRTOS с прерываниями.



1

 Ответить**Aveal**

Автор

 Reply to Денис 14 минут назад

Спасибо за отличный отзыв! Постараюсь по FreeRTOS побольше статей добавить.



0

 Ответить