



Василий @qdx

Пользователь

25 сентября 2011 в 01:08

FreeRTOS: введение

Программирование микроконтроллеров*

Здравствуйтесь. В короткой серии постов я постараюсь описать возможности, и подходы работы с одной из наиболее популярной и развивающейся РТОС для микроконтроллеров – FreeRTOS. Я предполагаю базовое знакомство читателя с теорией многозадачности, о котором можно почитать в одном из соседних постов на Хабре или ещё где-то.

Ссылки на остальные части:

FreeRTOS: межпроцессное взаимодействие.

FreeRTOS: мьютексы и критические секции.

Зачем все это? Или введение в многозадачные системы, от создателей FreeRTOS.

Традиционно существует 2 версии многозадачности:

- «Мягкого» реального времени(soft real time)
- «Жесткого» реального времени(hard real time)

К ОСПВ мягкого типа можно отнести наши с Вами компьютеры т.е. пользователь должен видеть, что, например, нажав кнопку с символом, отведенный символ, а если же он нажал кнопку, и спустя время не увидел реакции, то ОС будет считать задачу «не отвечающей»(по аналогии Windows — «Программа не отвечает»), но ОС остается пригодной для использования. Таким образом, ОСПВ мягкого времени просто определяет предполагаемое время ответа, и если оно истекло, то ОС относит task к не отвечающим.

К ОСПВ жесткого типа, как раз относят ОСПВ во встраиваемых устройствах. В чем-то они похожи на ОСПВ на дестопках(многопоточное выполнение на одном процессоре), но и имеют главное отличие — каждая задача **должна** выполняться за отведенный квант времени, не выполнение данного условия ведет к краху всей системы.

А все таки зачем?

Если у Вас есть устройство с нетривиальной логикой синхронизации обмена данными между набором сенсоров, если Вам действительно нужно гарантировать время отклика, и наконец-то если Вы думаете, что система может разрастись, но не знаете насколько, то РТОС Ваш выбор.

Не стоит применять РТОС, для применения РТОС т.е. не нужно применять РТОС в слишком тривиальных задачах(получить данные с 1 сенсора и отправить дальше, обработать нажатие 1 кнопки и т.д) т.к. это приведет к ненужной избыточности, как полученного кода, так и решения самой задачи.

Работа с задачами(или задачами, процессами).

Для начала приведу несколько определений, для того чтобы внести ясность в дальнейшие рассуждения:

"Операционные системы реального времени (ОСПВ(RTOS)) предназначены для обеспечения интерфейса к ресурсам критических по времени систем реального времени. Основной задачей в таких системах является своевременность (timeliness) выполнения обработки данных".

"FreeRTOS — многозадачная операционная система реального времени (ОСПВ) для встраиваемых систем. Портинг на несколько микропроцессорных архитектур.

От хабраюзера @andrewsh, по поводу лицензии: разрешено не публиковать текст приложения, которое использует FreeRTOS, несмотря на то что OS линкуется с ним. Исходники самой же RTOS должны всегда прикладываться, изменения, внесенные в неё — тоже."

FreeRTOS написана на Си с небольшим количеством ассемблерного кода(логика переключения контекста) и ее ядро представлено всего 3-мя файлами. Более подробно о поддерживаемых платформах можно почитать на [официальном сайте](#).

Перейдем к делу.

Любой task представляет собой Си функцию со следующим прототипом:

```
void vTask( void *pvParameters );
```

Каждый task — это по сути мини подпрограмма, которая имеет свою точку входа, и исполняется внутри бесконечного цикла и обычно не должна выходить из него, а также имеет собственный стек. Одно определение taska может использоваться для создания нескольких taskов, которые выполняются независимо и также имеют собственный стек.

Тело taska не должно содержать явных **return;** конструкций, и в случае если task больше не нужен, его можно удалить с помощью вызова **A** функции. Следующий листинг, демонстрирует типичный скелет taska:

```

void vTask( void *pvParameters) {
    /* Данный фрагмент кода будет вызван один раз, перед запуском таска.
       Каждый созданный таск будет иметь свою копию someVar.
       Кроме объявления переменных, сюда можно поместить некоторый инициализационный код.
    */
    int someVar;

    // Так как каждый таск - это по сути бесконечный цикл, то именно здесь начинается тело таска.
    for( ;; ) {
        // Тело таска
    }

    // Так как при нормальном поведении мы не должны выходить из тела таска, то в случае если это все таки произошло, мы удаляем таск.
    // Функция vTaskDelete принимает в качестве аргумента хэндл таска, который стоит удалить.
    // Вызов внутри тела таска с параметром NULL, удаляет текущий таск

    vTaskDelete( NULL );
}

```

Для создания таска, и добавления ее в планировщик используется специальная API функция со следующим прототипом:

```

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

```

pvTaskCode – так как таск – это просто Си функция, то первым параметром идет ее значение.

pcName – имя таска. По сути это нигде не используется, и полезно только при отладке с соответствующими плагинами для IDE.

usStackDepth – так как каждый таск – это мини подпрограмма, со своим стеком, то данный параметр отвечает за его глубину. При скачивании RTOS и разворачивания системы для своей платформы, вы получаете файл FreeRTOSConfig.h настройкой которого можно конфигурировать поведение самой ОС. В данном файле также объявлена константная величина **configMINIMAL_STACK_SIZE**, которую и стоит передавать в качестве usStackDepth с соответствующим множителем, если это необходимо.

pvParameters – при создании, каждый таск может принимать некоторые параметры, значения, или ещё что-то что может понадобится внутри тела самого таска. С точки зрения инкапсуляции, этот подход наиболее безопасный, и в качестве pvParameters стоит передавать, например, некоторую структуру, или NULL, если ничего передавать не нужно.

uxPriority – каждый таск имеет свой собственный приоритет, от 0(min) до (**configMAX_PRIORITIES** – 1). Так как, по сути, нет верхнего предела для данного значения, то рекомендуется использовать как можно меньше значений, чтобы не было дополнительно расхода RAM на данную информацию.

pxCreatedTask — хэндл созданного таска. При создании таска, опционально можно передать указатель на хэндл будущего таска, для последующего управления работой самого таска. Например, для удаления определенного таска.

Данная функция возвращает **pdTRUE**, в случае успешного создания таска, или **errCOULD_NOT_ALLOCATE_REQUIRED_MEMORY**, в случае если размер стека был указан слишком большим, т.е. недостаточно размера хипа для хранения стека таска, и самого таска.

На следующем листинге я привел, короткий пример законченной программы, которая создает 2 таска, каждый из которых мигает светодиодом.

```

void vGreenBlinkTask( void *pvParameters ) {
    for( ;; ) {
        P8OUT ^= BIT7;

        // Выполнить задержку в 700 FreeRTOS тиков. Величина одного тика задана в FreeRTOSConfig.h и как правило составляет 1мс.
        vTaskDelay( 700 );
    }
}

void vRedBlinkTask( void *pvParameters ) {
    for( ;; ) {
        P8OUT ^= BIT6;

        // Выполнить задержку в 1000 FreeRTOS тиков. Величина одного тика задана в FreeRTOSConfig.h и как правило составляет 1мс.
        vTaskDelay( 1000 );
    }
}

```

```
}

void main(void) {
    // Инициализация микроконтроллера. Данный код будет у каждого свой.
    vInitSystem();

    // Создание задач. Я не включил код проверки ошибок, но не стоит забывать об этом!
    xTaskCreate( &vGreenBlinkTask,
                (signed char *) "GreenBlink",
                configMINIMAL_STACK_SIZE,
                NULL,
                1,
                NULL );
    xTaskCreate( &vRedBlinkTask,
                (signed char *) "RedBlink",
                configMINIMAL_STACK_SIZE,
                NULL,
                1,
                NULL );

    // Запуск планировщика т.е начало работы задач.
    vTaskStartScheduler();

    // Сюда стоит поместить код обработки ошибок, в случае если планировщик не заработал.
    // Для примера я использую просто бесконечный цикл.
    for( ;; ) { }
```

В следующем посте я планирую написать о взаимодействии между задачами, и работе с прерываниями.

freertos, осрв, микроконтроллеры

↑ +52 ↓ 70,3k ★ 189



Василий @qdx

карма
рейтинг
28,2 0,0

ПОХОЖИЕ ПУБЛИКАЦИИ

12 сентября 2016 в 09:35

Как штатными настройками убить микроконтроллер. Часть 2

↑ +22 11,5k ★ 39 2

29 августа 2016 в 09:37

Как штатными настройками убить микроконтроллер. Часть 1

↑ +30 35k ★ 50 40

6 июля 2016 в 10:37

Нестандартный подход к программированию микроконтроллера

↑ +9 11,2k ★ 74 20

САМОЕ ЧИТАЕМОЕ

Разра

Сутки

Неделя

Месяц