### Что хочет реализовать программист в данной программе

1. Периодическое измерение аналоговой величини с выбранной частотой (без временнЫх промежутков).   
   Каждое измерение должно выполняться в строго заданный момент, т.е. не раньше и не позже.
2. Усреднение по серии измерений (предварительная обработка – НЧ фильтрация и сжатие).  
   Должно формировать усредненный результат не позже окончания следующей серии.
3. Передачу усредненных значений в виде текстовых строк в другой компьютер.

Последовательность шагов (1-2-3) должна производиться неограниченно долго (в реальном времени).

### Функции uCOS-II, использованные в программе

1. OSInit()
2. OSTaskCreate(....)
3. OSStart()
4. BSP\_Init()
5. OS\_CPU\_SysTickInit()
6. OSTaskDel(.)
7. OSTaskResume(.)
8. OSTimeDly(.)
9. OSTaskSuspend()   
   Зеленым отмечены функции стандартной библиотеки uCOS-II v.2.86,  
   бирюзовым – функции, не предназначенные для (многократного) использования прикладным программистом.

### Что должен сделать прикладной программист чтобы реализовать программу под uCOS

1. Разбить алгоритм на части
   1. В первом нашем примере мы сделаем это разбиение поэтапно, начав с самого простого варианта: весь прикладной код – в одной части.
2. Каждую часть следует оформить как *Задачу*
   1. Минимальная конфигурация нашей программы под uCOS-II будет содержать не менее трех *Задач*:
      1. Стартовая *Задача*
      2. Одна *Задача*, содержащая прикладной код
      3. *Задача* TaskIdle (Бездействие Системы), создается автоматически, ее код в файле os\_core.c:1763.
   2. Что такое *Задача*?   
      это совокупность:   
      а) кода, его надо оформить по правилам (см. справочник по функциям uCOS-II, функция OSTaskCreate);   
      б) стека, индивидуального для каждой *Задачи*, для стека в uCOS-II надо явно выделить память (определив массив ?? какого размера); в стеке локальные переменные.  
      в) заполненной системной структуры Task Control Block TCB, эта структура будет заполнена автоматически при создании *Задачи*.
   3. Перед дальнейшими действиями следует выполнить настройку (конфигурирование) uCOS-II.  
      Это делается путем редактирования конфигурационных заголовочных файлов.   
      Для этих файлов в ОС имеются заготовки, содержащие начальные значения настроек и подсказки/комментарии, помогающие выполнять настройки.   
      Таких файлов в uCOS-II v.2.86 два:
      1. Файл app\_cfg.h, в нем рекомендуется (но не обязательно) помещать определения констант, используемых многократно в прикладном коде.   
         В нашем первом примере в файле app\_cfg.h определены константы для уровней приоритета создаваемых *Задач* и константы для размеров стеков каждой *Задачи.*  
         Важно !!! В uC/OS-II каждая *Задача* должна иметь уникальный приорите. Две *Задачи* НЕ МОГУТ иметь одинаковый уровень приоритета. Это упрощает код *Диспетчера Задач*.
      2. Файл os\_cfg.h – содержит настроечные константы ОСРВ.  
         Вот перечень настроек, которые приходится делать в первую очередь (с указанием номеров строк) в файле os\_cfg.h.   
         (:39) OS\_LOWEST\_PRIO – задает приоритет *Задачи* TaskIdle (самый низкий).  
         (:47) OS\_MAX\_TASKS – максимальное число *Задач* (у нас будет 5). При инициализации ОС будет создано (OS\_MAX\_TASKS+1) экземпляров системной структуры TCB.  
         (:52) OS\_TICKS\_PER\_SEC – частота запросов от Системного Таймера SysTick. Период вызова обработчика SysTick – единица измерения задания времени тайм-аутов.  
         (:62...74) разрешение трансляции функций группы Task Management  
           
         Диапазон возможных значений приоритета в uCOS-II:   
          от 0 (самый высокий) до OS\_LOWEST\_PRIO (самый низкий),   
          для OS\_LOWEST\_PRIO в данной прог. установлено значение 20, т.е. всего уровней 21.  
           
         Число уровней приоритета больше числа *Задач.*   
         Это позволит свободно на этапе исполнения изменять уровень приоритета *Задач*, если этого захочется.
   4. Каждую *Задачу* надо «создать», это делается вызовом функции OSTaskCreate().Детально разбираем особенности этой функции.
   5. *Задача* в многозадачной программе может находиться в нескольких *Состояниях*.   
      Смотрим презентацию, стр.48, обсуждаем *Состояния* *Задач*.   
      О терминологии, используемой разными Авторами (увы) – разные названия *Состояний*  
      Сразу после создания *Задача* находится в состоянии готовности к исполнению (Ready)
      1. В первой (простейшей) версии нашей программы надо создать две *Задачи*.  
         Для двух функций, содержащих код *Задач*, выбраны имена  
         App\_TaskStart Стартовая *Задача* (определение в строках 96...207)   
          Стартовая *Задача* обязательно должна быть создана до запуска многозадачности.  
         App\_TaskMeas1 *Задача* – Измеритель (определение в строках 225...)
      2. Перед созданием *Задач* следует принять решение об их приоритетах.   
         В uCOS-II приоритет кодируется целым числом типа uint8\_t,   
         чем МЕНЬШЕ число ‑ тем ВЫШЕ приоритет.   
         В FreeRTOS наоборот: чем БОЛЬШЕ число, тем ВЫШЕ приоритет.  
           
         Смотрим в файл app\_cfg.h, там заданы значения уровней приоритета. Обсуждаем:  
         Для Стартовой Задачи приоритет выбран = 3. Это значение пока непринципиально.  
         Для Измерителя приоритет =5. Это ниже, чем для Стартовой. Вспоминаем о рекомендации в Стартовой *Задаче* создавать и остальные пользовательские Задачи. Каким должно быть соотношение приоритетов Стартовой *Задачи* и прочих *Задач*, создаваемых в Стартовой ?
   6. При создании *Задачи* ‑ в коде функции OSTaskCreate() из набора заранее (статически) созданных системных структур TCB (Task Control Block, Блок Управления *Задачей*) берется свободная, и в ней заполняются поля в соответствии с настройками, заданными прикладным программистом. Сколько штук TCB было создано ??
   7. Смотрим презентацию, стр.63. Обсуждаем основные поля Блока Управления *Задачей*.

==================================================================================

1. Выполнение простейшей версии программы.
   1. После перехода в отладку смотрим начальное состояние МК.
      1. Тактирование: ядро и все прочие тактовые частоты используют 8 МГц.  
         В этом состоянии МК работоспособен.
      2. От запуска прошло чуть больше 1000 тактов.
   2. Входим внутрь функции OSInit().
      1. Часть строк не оттранслировалась. Это следствие значений настроек конфигурации (в файле os\_cfg.h и в других местах), которые используются в директивах условной трансляции.
      2. Смотрим строку [os\_core.c:578]. Внутри функции OS\_InitTaskIdle() создается «Пустая *Задача*».  
         Входим внутрь, в строке [os\_core.c:1399] видим вызов OSTaskCreate() – создающую «Пустую *Задачу*. Первый параметр это точка входа в код OS\_TaskIdle, можем взгянуть на этот код.
      3. В строке74, в функции main() создается Стартовая *Задача*. Область памяти для ее стек выделена в строке [26] в результате определения массива элементов типа OS\_STK. Этот тип данных процессорно-зависимый. Он переопределяется, когда ОСРВ портируют на новый тип процессора.
      4. Размер стека для создаваемой *Задачи* должен выбрать прикладной программист так, чтобы исключить переполнение этого стека при работе отлаженной программы.   
         Как можно контролировать состояние стеков – подумайте. (для каждой *Задачи* стек свой !).
      5. Выполняем строку 74, создаем Стартовую *Задачу*. В OSTaskCreate() заполняются поля блока TCB для Стартовой *Задачи*, это можно посмотреть.
      6. Следующий шаг – вызов функции OSStart().   
         Он весьма странный:   
         поставим остановы в строках 81 и 99, и затем пуск.   
         Обсуждаем, что произошло.  
         Внутри OSStart() управление получил Диспетчер. В программе две *Задачи*:   
         TaskIdle и Стартовая. Будет запущена та, у которой более высокий приоритет.
      7. Теперь можно открыть ЛА. В нем настроен вывод 6-ти графиков.  
         Системная переменная OSPrioCur содержит уровень приоритета исполняемой *Задачи*.  
         Прикладной программист (мы!!) определил несколько отладочных переменных, чтобы можно было наблюдать события, происходящие при выполнении программы. Но пока еще почти ничего не происходит.
      8. Строка 101 Вызов BSP\_Init() - Настраивается подсистема тактирования.
   3. Строка 164 – создается *Задача-*Измеритель. Что может помешать созданию *Задачи* ‑  
      неверная настройка. Надо проверять значение, возвращаемое функцией ОС, перед этим следует выяснить, какие ошибки проверяются в используемой функции.
   4. Каково соотношение между приоритетами двух созданных *Задач*?  
      Что произойдет, если приоритет создаваемой *Задачи*-Измерителя выше приоритета стартовой.
   5. Стартовая *Задача* выполнила все, что от нее хотел прикладной программист.  
      Что можно сделать, чтобы начала исполняться *Задача*-Измеритель ?  
      Как поступить со Стартовой *Задачей* ?  
      Что произойдет, если удалить Стартовую *Задачу* ? Сначала поставьте останов в строке227.  
      Запустим на продолжение, посмотрим графики в ЛА
   6. Смотрим код в Задаче-Измерителе.
      1. Строки 214...220 ‑ параметры измерений и рабочие переменные (глобальные)
      2. Строка 233 – настройка канала АЦП  
          236 – и далее – смотрите комментарии в коде
   7. Точку останова в строку 263. Запускаем программу. Смотрим в диаграммы.  
      Что там можно увидеть. Как изменить программу, чтобы измерения выполнялись и в периоды, когда идет передача строк ? Как в этом может помочь многозадачность ?
   8. Переносим код, выполняющий передачу, в другую *Задачу*, для этого следует раскомментировать строку 23. Какой приоритет должен иметь Передатчик по отношению к Измерителю ?
   9. Запускаем программу. Правильно ли она работает ?
   10. Разбираем, как Измеритель взаимодействует с Передатчиком.
       1. Пока Результат не готов, Передатчик должен быть неактивен (находиться в Waiting)/  
          Есть ли свободное процессорное время, в которое Передатчик может работать? Да, времени много. Как это увидеть ?  
          Почему это происходит (Передатчик не работает).