Захотел я написать программу демонстрашку для своей платы. Чтобы красиво что нибудь на экран выводила, да отвечала на внешние воздействия. А также своим телом демонстрировала принципы и основы работы кооперативной RTOS, а то вижу много сомневающихся и непонимающих всей прелести :) **Итак, вначале ТЗ:**

- Программа постоянно выводит на экран статичную картинку, точнее надпись: Pinboard v1.0 by DI HALT 2009"
- При этом она постоянно шлет в USART на скорости 9600 текущее значение из АЦП.
- При резком изменении показаний АЦП на экране возникает надпись "АЦП, канал 0 xxx - значение", где вместо XXX текущее показание АЦП. Через две секунды показа такой инфы она вновь сменяется на начальную надпись "PinBoard бла бла бла".
- При нажатии на кнопку на экране должно высвечиваться "Нажата кнопка X - номер кнопки", где вместо X - имя кнопки которую нажали. Также инфа эта висит 2 секунды, потом сменяет на заставку.
- Ну и, наконец, при получении байта по USART выводит на экран надпись "Терминальный ввод", где во второй строке идет строка введенная с терминала.

Вот так вот. Простенько и со вкусом.

Теперь начинаем дробить наше ТЗ на подзадачи.

Что у нас будет:

- Задача опроса АЦП
- Задача отправки данных по UART
- Вывод данных на экран
- Задача Приема данных по UART

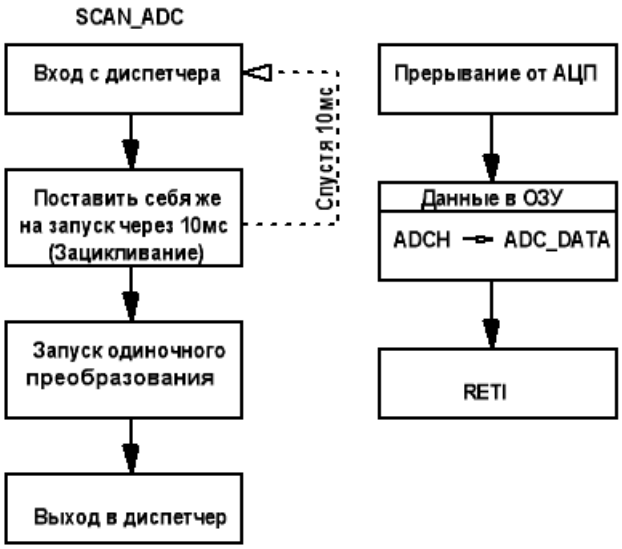
Это первое приближение. Потом задчи будут дробиться на подзадачи, но об этом позже.

Итак, задача сканирования.

Пути решения, собственно, два.

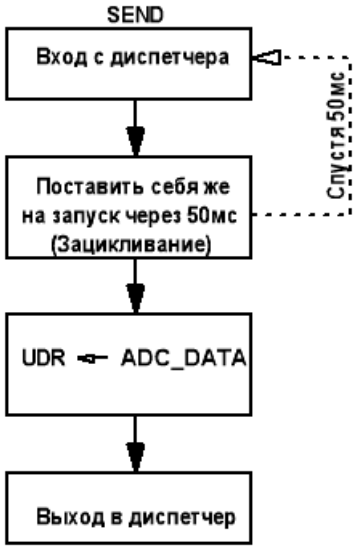
- 1. Непрерывное преобразование. Заводим АЦП и оно нам гонит данные, выбрасывая их через прерывание. Можно и так, но не хочу. Задача не критичная, поэтому нагружать процессор бессмысленным потоком прерываний я не хочу.
- 2. Циклический опрос. Раз в 10мс мы запускаем преобразование и оно нам складывает в прерывании данные в ОЗУ в специальную ячейку. Да будет так.

Получили первую задачу, задача сама себя зацикливает через диспетчер задач, первым делом добавляя себя в очередь на исполнение с задержкой. Когда будет выполнено прерывание, то в прерывании мы ничего особенного не делаем, а просто сохраняем данные в ОЗУ:



Задача отправки данных по USART.

Данные АЦП у нас уже есть, они лежат в переменной ADC_DATA и потоянно обновляются там задачей ScanADC. Так что алгоритм ее будет тоже до смешного прост:



Вывод данных на экран

Тут тоже два варианта действий -- первый при каждом чихе писать напрямую в дисплей. Этот метод плох тем, что события могут быть частыми, а дисплей штука все же медленная и требует времени на запись. А человеческий глаз еще медленней и быстро менющаяся картинка ему не нужна. Вполне хватит частоты обновления 5-10 раз в секунду. Мы же не видео гоним, а текст. Так что выберем второй вариант -- создаем в ОЗУ область из 32 байт (2 строки по 16 символов) -- видеопамять, и все изменения делаем в ОЗУ, а 5 раз в секунду эту видеопамять копируем непосредственно в дисплей. Можно и реже, но будет заметно глазу. Итак, задача вывода на дисплей распалась на две:

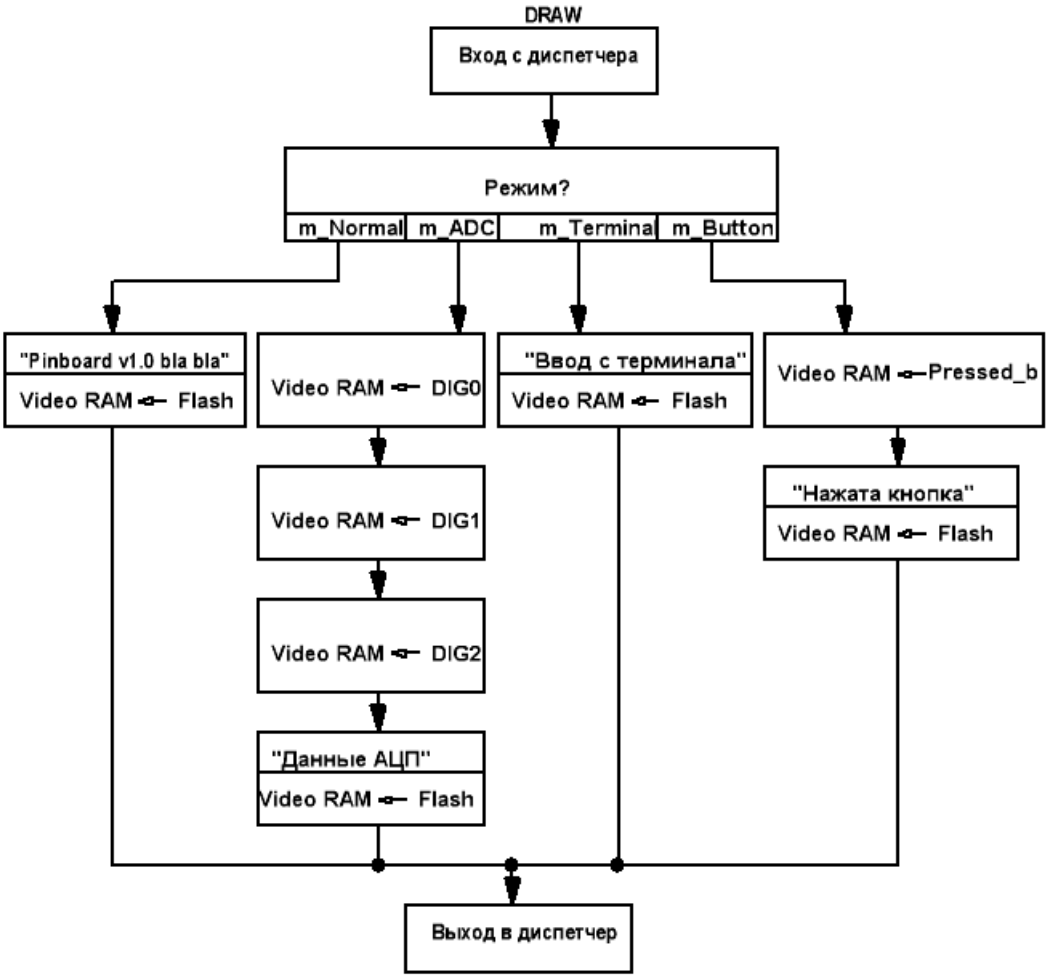
- Обновление дисплея
- Рисование в ОЗУ

Обновление дисплея

Опять же задача примитивна донельзя - взять данные и тупо скопировать. И зациклиться через диспетчер задач с выдержкой.

Рисование в ОЗУ

Тут тоже все могло быть просто -- берем и записываем туда нужные данные. Вот только данные у нас состоят из статичных строк всякие там "АЦП канал 0 бла бла бла" и динамических, которые меняются в зависимости от условий. А на экран еще картинка выводится исходя из текущего режима. Так что алгоритм усложняется:



Как видим, процедура рисования оперирует уже кучей разных переменных и констант. С константами понятно все, они во флеш записаны и оттуда просто копируются. А откуда брать переменные? А их нам подготовят другие задачи!

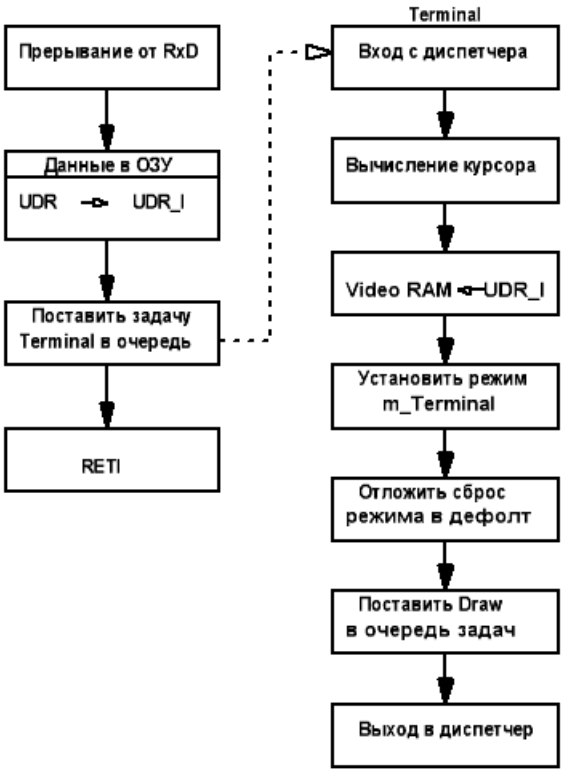
Начнем с режима терминала.

Тут первым делом сработает прерывание -- при приходе байта мы попадаем в обработчик прерывания, где:

- Сохраняем принятый байт в переменную ОЗУ
- Забрасываем в очередь задачу обработки терминала
- Выходим

Кратко и быстро - 11 команд на весь экшн. Нефига в обработчиках прерываний сопли жевать. Не для того они сделаны.

Задача Terminal этот сохраненный байт потом запишет в нижную строку видеопамяти, выставит режим "Терминал", отложит сброс режима в дефолт на 5 секунд и поставит в очередь задачу рисования статичного заголовка - Draw, который перисует содержимое видеопамяти:



Задача Button

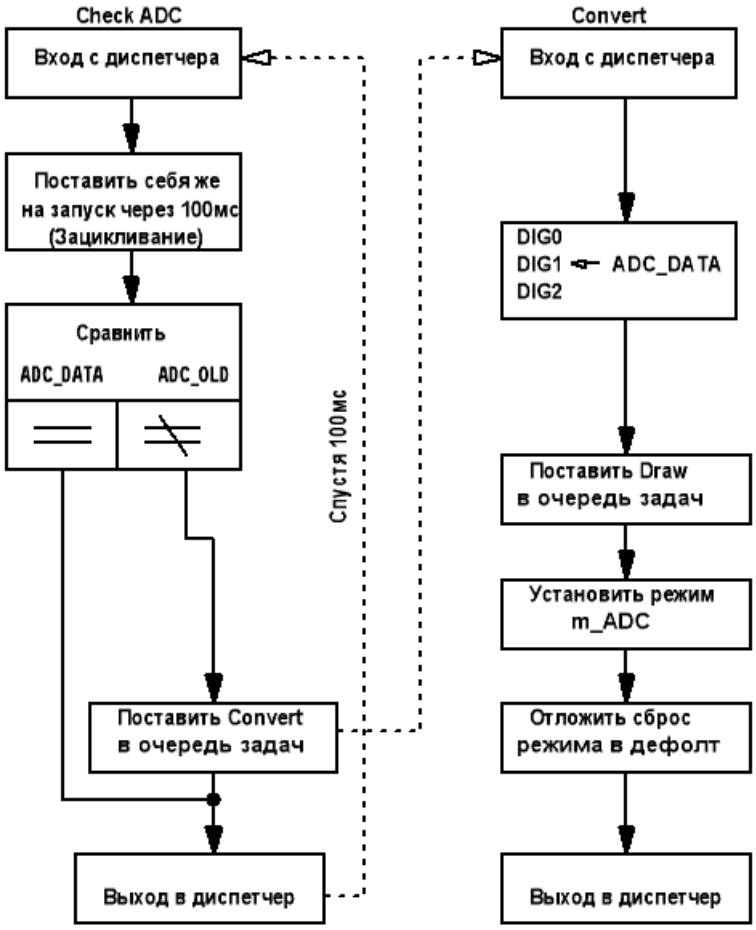
Состоит из зацикленной через диспетчер процедуры опроса клавиатуры и в зависимости от нажатой кнопки выводит сохраняет в ячейку памяти имя кнопки и установив режим "Button" вызывает задачу рисования.

Определение изменения состояния АЦП.

Еще одна фоновая задача, зацикленная диспетчером. Тут тоже все линейно и просто -- мы берем из памяти старое значение и новое значение. Отрезаем нестабильные младшие биты, чтобы отловить только значительные изменения. Сравниваем -- если данные не изменились то выходим, если изменились то кидаем в конвеер задачу конвертации

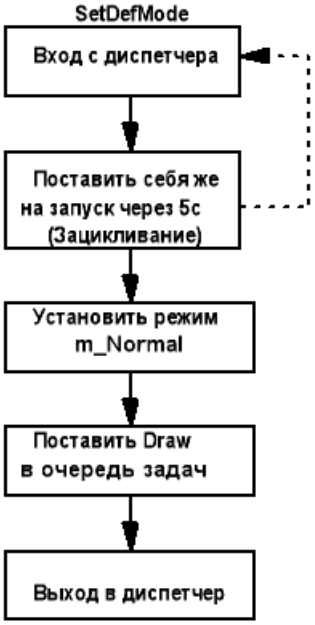
Обработка данных АЦП

Задача Convert берет из переменной АЦП данные и сконвертит их в ASCII код пригодны для вывода на экран. Далее откладывает задержку сброса режима в дефолтное состояние и вызывает задачу отрисовки данных в видеопамять.

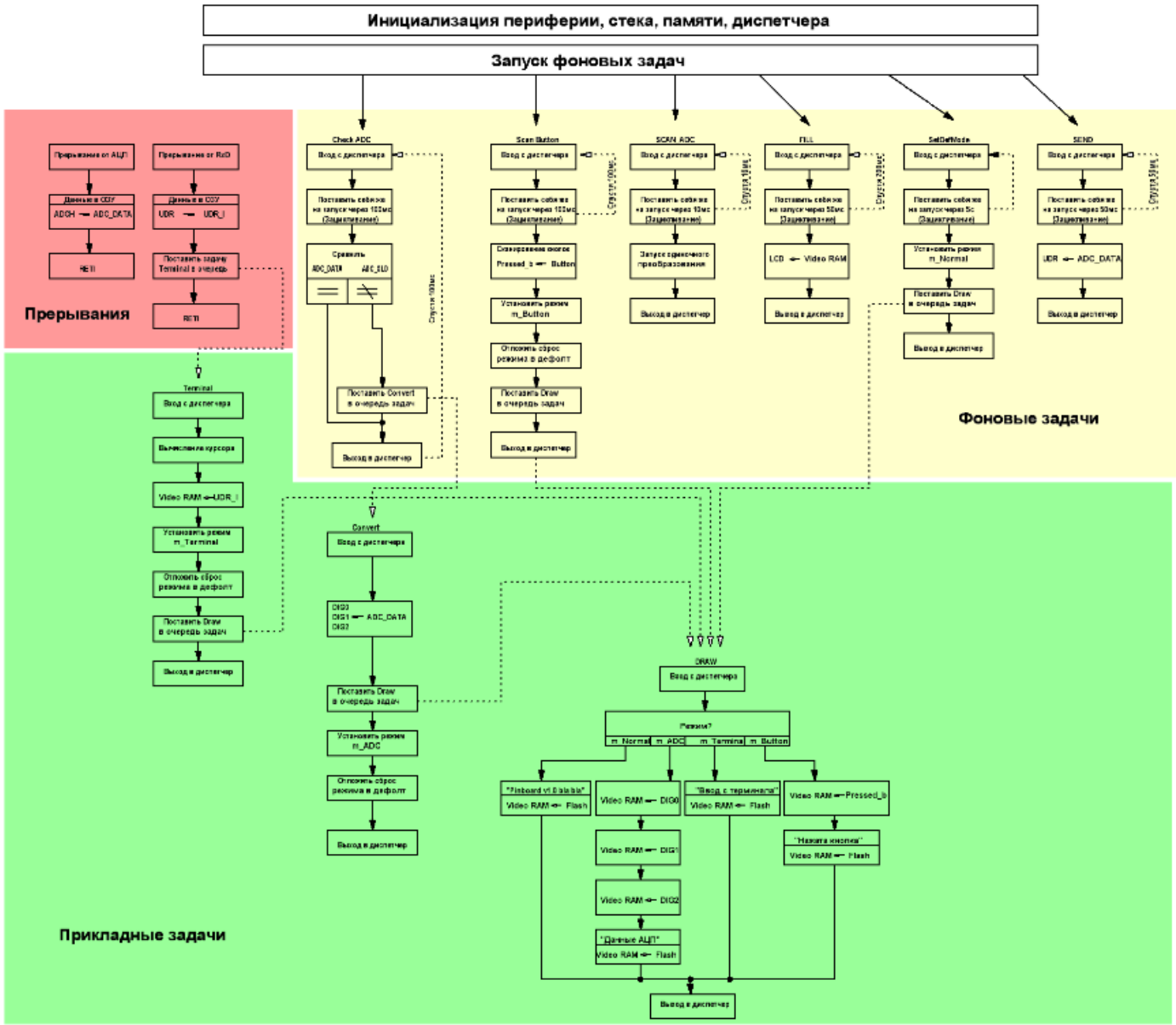


Сброс в дефолтное состояние

Тупейшая циклическая задача. Зациклена диспетчером. Раз в 5 секунд (если ее не сбросить) сбрасывает текущий режим работы в дефолтный. В дефолтном режиме на экран вводится заставка



Вот так все это выглядит в комплексе:



Увеличить

Черные стрелки это непосредственное выполнение, а пунктирные стрелки это вызов через диспетчер. То есть они выполнятся, но не сейчас, а чуть позже, в порядке живой очереди. И вот такая петрушка крутится в мозгах контроллера -- одни задачи набрасывают на конвееры другие задачи, каждая из них максимально простая и линейная, а все промежуточные данные хранятся в ячейках памяти ОЗУ. А вместе все это образует довольно причудливую систему. Но при этом, за счет простоты конечных кусочков написание и отладка такой системы это сплошное удовольствие. У меня все заработало с первого же раза без отладки, я этому уже и удивляться перестал.

При этом усложнить и нарастить программу элементарно, не придется даже менять код уже написанный - достаточно добавить новых фоновых и/или прикладных задач, прерываний, да чего угодно. Главное чтобы они вовремя отдавали управление диспетчеру и их число не переполняло очереди задач и (особенно!) таймеров. Для перестраховки число таймеров можно сделать равным числу задач вообще, либо числу задач вызываемых с задержкой. Всего задач может быть 253 штуки, так что есть где разгуляться. Если где то критично время выполнения, то можно не разбивать одну фичу на цепь задач, а написать в классическом ключе с процедурами и функциями, главное помнить, что длинная процедура тормознет весь конвеер на время своего выполнения.

Еще стоит помнить про общее обращение к портам и регистрам. Чтобы не получилось, что две задачи шлют в один usart данные и данные перемешиваются побайтно (байт из одной посылки, байт из другой), тут придется вводить флаги занятости и/или буффера обмена. В общем, голову включать тоже не помешает :))))

Ну и исходник проги:

Тыц!

как всегда подробно комментирован. Если будут вопросы по работе RTOS, то обращайся по адресу dihalt@dihalt.ru - отвечу на любые вопросы.