Описание реализации потока Beep thread.

Аппаратные ресурсы, задействованные в реализации:

TIM2, TIM2\_IRQn (13, 1), GPIOA (1).

Реализация потока содержит аппаратно зависимую часть и часть не зависящею от аппаратуры. Реализация представлена четырьмя файлами: .c и .h для части зависящей от аппаратуры и .c и .h для части не зависящей от аппаратуры.

Файлы реализации:

Beep\_Drv.h и Beep\_Drv.c

Beep\_Task.h и Beep\_Task.c

Аппаратно-независимая часть.

Beep\_Task.h.

Beep\_Task.h содержит объявление функции потока void vBeepTask(void \*pvParameters), которая необходима при создании потока в функции Create\_OS\_Objects() (main.c).

Beep\_Task.c.

Beep\_Task.c включает хедеры Beep\_Task.h (определение функции vBeepTask), Beep\_Drv.h (определение аппаратно-зависимых функций) и CommonDefines.h (для взаимодействия с FreeRTOS).

Beep\_Task.c содержит в себе описание функции потока Beep thread - vBeepTask.

vBeepTask() инициализирует аппаратуру посредством вызова функции InitBeep(), которая будет описана ниже, и становится в бесконечный цикл ожидания сообщений от других потоков, посредством оператора while и вызова функции FreeRTOS xQueuePeek(), которая “усыпляет” поток до тех пор, пока не придет сообщение в очередь qTo\_Beep. Когда какой-либо поток пошлет сообщение в очередь qTo\_Beep, FreeRTOS “пробуждает” поток и передает ему управление. xqueuePeek() не извлекает сообщение из очереди. Затем вызывается функция xSemaphoreTake(), которая пытается захватить семафор, защищающий доступ к аппаратуре (о этом семафоре ниже), если удается захватить семафор, то xQueueReceive() извлекает сообщение из очереди qTo\_Beep и функция Beep() настраивает аппаратуру “beepera”, согласно данным извлеченного сообщения, если нет, то поток “засыпает” и ждет пока семафор не освободится (в обработчике прерывания).

Аппаратно-зависимая часть (“драйвер“).

Beep\_Drv .h.

Beep\_Drv .h включает хедер InterDefines.h (тут определены типы данных для сообщений).

Beep\_Drv .h содержит объявление функций InitBeep() (инициализация оборудования, ответственного за ”бипанье” ), Beep() (перенастройка оборудования, согласно указанным параметрам) и обновление глобальной переменной Num\_T (ответственной за реализацию серии звуковых импульсов). Также включен хедер stm32f10x.h и объявления переменных для настройки и работы с аппаратурой.

Beep\_Drv.c

Beep\_Drv.c ключает хедер Beep\_Drv.h (определение аппаратно-зависимых функций).

Beep\_Drv.c содержит в себе описание функций InitBeep() и Beep().

InitBeep().

InitBeep() разрешает тактирование таймера TIM2, настраивает его, настраивает NVIC на прерывание по TIM2, разрешает тактирование порта GPIOA и настраивает ножку PA1, с которой связана база транзистора, управляющего пьезоэлементом.

Beep().

Beep() до настраивает таймер TIM2, согласно данным полученным из BeepParam (сообщение от других потоков), вычисляет значение глобальной переменной Num\_T, запускает TIM2 и разрешает прерывание.

Часть кода из файла stm32f10x\_it.c (обработчики прерываний):

Механизм защиты аппаратуры от одновременного доступа.

Поскольку пьезоэлемент один, то доступ к нему нужно контролировать. Пусть поток А посылает сообщение нашему потоку. FreeRTOS будит наш поток, передавая управление функции QueuePeek(). Затем вызывается xSemaphoreTake(), которой удается захватить семафор, значит продолжаем выполнение дальше, извлекаем сообщение и передаем его функции Beep(). Beep() настраивает TIM2, и он начинает считать. Пока таймер не досчитал до конца (и не освободил семафор) другой поток Б (а может и этот же самый А) посылает новое сообщение нашему потоку. Если бы не было механизма защиты, то Beep() перенастроила бы TIM2, не дав ему досчитать предыдущие промежутки (насчет помежуткИ, смотри реализацию обработчика прерывания). А в текущей реализации наш поток заснет, пока семафор не будет освобожден в обработчике прерывания, после того как TIM2 обслужит предыдущий запрос.

Описание реализации потока Calc thread. (вспомогательный поток)

Аппаратные ресурсы, задействованные в реализации:

none.

Реализация потока содержит аппаратно зависимую часть и часть не зависящею от аппаратуры. Реализация представлена четырьмя файлами: .c и .h для части зависящей от аппаратуры и .c и .h для части не зависящей от аппаратуры.

Файлы реализации:

Calc\_Drv.h и Calc\_Drv.c

Calc\_Task.h и Calc\_Task.c

Аппаратно-независимая часть.

Calc\_Task.h.

Calc\_Task.h содержит объявление функции потока vCalcTask, которая необходима при создании потока в функции Create\_OS\_Objects() (main.c).

Calc\_Task.c.

Calc\_Task.c включает хедеры Calc\_Task.h (определение функции vCalcTask), Calc\_Drv.h (определение аппаратно-зависимых функций) и CommonDefines.h (для взаимодействия с FreeRTOS).

Beep\_Task.c содержит в себе описание функции потока Calc thread - vCalcTask.

vcalcTask() реализует бесконечное ожидание сообщений от других потоков, посредством оператора while и xQueueReceive(), которая ожидает сообщений на очереди queu\_to\_calc. Как только сообщение было послано, нашему потоку, FreeRTOS нас будит и передает управление xQueueReceive(), которая извлекает сообщение-указатель, указывающий на буфер, данные которого необходимо обработать. После обработки данные передаются функции show\_result() для отображения на экране. Более подробно об обработке данных и буфере смотри описание потока Measuring thread.

Посылка сообщений нашему потоку возможно только лишь если переменная lock\_send\_message\_to\_calc\_thread равна нулю.

Аппаратно-зависимая часть (лишь для терминологии, не содержит непосредственной работы с аппаратурой).

Calc\_Drv .h.

Calc\_Drv .h содержит объявление функций init\_calc() (заглушка) и show\_result() (вывод на экран данных переданных через параметры функций).

Calc\_Drv.c

Calc\_Drv.c ключает хедер Calc\_Drv.h (определение аппаратно-зависимых функций).

Calc\_Drv.c содержит в себе описание функций init\_calc(), show\_result() и обьявление внешней функции lcd\_write() для вывода информации на экран.

show\_result().

Создает форматированные строки, содержащие информацию о токах, напряжениях и мощностях, для вывода на LCD экран. Токи в мА, напряжения в В, мощности в мВт.

Это вспомогательный поток, задача которого является вывод мгновенных, не усредненных значений токов, напряжений и мощностей для визуального восприятия человеком.

Описание реализации потока LCD thread.

Аппаратные ресурсы, задействованные в реализации:

GPIOC (12), GPIOB (5 - 10).

Реализация потока содержит аппаратно зависимую часть и часть не зависящею от аппаратуры. Реализация представлена четырьмя файлами: .c и .h для части зависящей от аппаратуры и .c и .h для части не зависящей от аппаратуры.

Файлы реализации:

LCD\_Drv.h и LCD\_Drv.c

LCD\_Task.h и LCD\_Task.c

LCD\_Delay.s

Аппаратно-независимая часть.

LCD\_Task.h.

LCD\_Task.h содержит объявление функции потока vLCDTask, которая необходима при создании потока в функции Create\_OS\_Objects() (main.c).

LCD\_Task.c.

LCD\_Task.c включает хедеры LCD\_Task.h (определение функции vLCDTask), LCD\_Drv.h (определение аппаратно-зависимых функций), CommonDefines.h (для взаимодействия с FreeRTOS) и InterDefines.h (тут определены типы данных для сообщений).

LCD\_Task.c содержит в себе описание функции потока LCD thread - vLCDTask().

vLCDTask() инициализирует аппаратуру посредством вызова функции InitLCD(), которая будет описана ниже, и становится в бесконечный цикл ожидания сообщений от других потоков, посредством оператора while и вызова функции FreeRTOS xQueueReceive(), которая ожидает сообщений на очереди qM\_to\_LCD. После поступления сообщения, оно извлекается из очереди и дальше, основываясь на ID\_cmd, управлении передается одной из функций: SendString(), ShiftBlink(), ControlBlink().

Аппаратно-зависимая часть (“драйвер“).

LCD\_Drv .h.

LCD\_Drv .h содержит объявление функций InitLCD() (инициализация аппаратуры, связанной с LCD), SendString(), ShiftBlink(), ControlBlink(), ClearLCD() (функции управления экраном и выводом на экран), InitGPIO\_LCD(), SendFirstCommand() и Preinit\_LCD() (ответственны за инициализацию аппаратуры), SendCommand(), SendData() и CreateSigmaPattern() (ответственны за низкоуровневые операции с LCD), Delay\_asm() (функция-задержки, написанная на ассемблере, для реализации маленьких задержек, от 238 нс, при 72 МГц тактовой частоты) и дефайны для функций, работающих с аппаратурой LCD.

LCD\_Drv.c

LCD\_Drv.c включает хедер LCD\_Drv.h (определение аппаратно-зависимых функций).

LCD\_Drv.c реализует низко-уровненные функции для работы с LCD аппаратурой (библиотека LCD), функции инициализации LCD аппаратуры и функции, представляющие интерфейс драйвера ( SendString(), …).

CreateSigmaPattern().

Функция CreateSigmaPattern() создат в памяти LCD контроллера картинку символа, для отображения символа дисперсия.

Delay\_asm().

Данная функция написана на ассемблере, для большой точности измерения временных интервалов (задержек), и для интереса :-).

**Строка, указатель на которую приходит в сообщении, не должна быть расположена в стеке вызова функции, которая вызывается из любой функции-потока. Расположение в стеке возможно только, если поток, который посылает сообщение, имеет приоритет меньше чем приоритет у LCD thread.** **(в таком случае LCD thread получит управление сразу и указатель будет указывать на валидные данные**)

Описание реализации потока Menu thread.

Аппаратные ресурсы, задействованные в реализации:

2kB SRAM. Must be rechecked.

Реализация потока содержит часть, зависящею от FreeRTOS и аппаратуры и не зависящею часть (меню). Реализация представлена четырьмя файлами: .c и .h для части зависящей и .c и .h для части не зависящей.

Упор делается на независимость реализации меню.

Файлы реализации:

Menu\_Drv.h и Menu\_Drv.c

Menu\_Task.h и Menu\_Task.c

Зависящая часть.

Menu\_Task.h.

Menu\_Task.h содержит объявление функции потока vMTask, которая необходима при создании потока в функции Create\_OS\_Objects() (main.c), и объявление аппаратно-зависимых функций.

Menu\_Task.c.

Menu\_Task.c включает хедеры Menu\_Task.h (определение функции vMTask), Menu\_Drv.h (определение аппаратно-зависимых функций), CommonDefines.h (для взаимодействия с FreeRTOS) и InterDefines.h (тут определены типы данных для сообщений).

LCD\_Task.c содержит в себе описание функции потока Menu thread - vMTask() и реализацию аппаратно-зависимых функций.

VMTask().

VMTask() инициализирует, посредством функции InitView(), первоначальное самостояние меню, а значит и всего генератора-измерителя, и становится в бесконечный цикл ожидания сообщений от других потоков, посредством оператора while и вызова функции FreeRTOS xQueueReceive(), которая ожидает сообщений на очереди qPB\_to\_M. После прихода сообщения, оно передается функции PrepareMesg(), которая извлекает из него необходимую информацию (тип клавиатуры, код клавиши и тип события ). Затем эта информация передается state-machine, реализующей меню MenuStateMach().

ChangeChar().

Функция ChangeChar() меняет символ 'ъ' на символ, код которого равен 0xFF, и который будет отображен контроллером LCD, как символ дисперсии. (предварительно он туда загружен функцией CreateSigmaPattern()).

**Строка, указатель на которую передается в функцию ChangeChar(), не может располагаться в Flash-памяти, то есть быть const-антной, поскольку функция меняет значение строки. Располагайте или в стеке или в куче. На счет кучи в данном проекте смотрите дальше.**

Не зависящая часть.

Menu\_Drv .h.

Menu\_Drv .h содержит объявление функций InitView() (инициализация состояния меню), MenuStateMach() (state-machine меню), Switch\_to\_EditMode(), EditParam() и ReDrawParam() , ReDrawZeroRow(), re\_draw\_body() (внутренние функции state-machine меню), объявление структур sParam, sSignalParam и sChannel (ответсвенны за реализацию state-machine меню), определение перемнных этих и других типов.

Menu\_Drv.c

Menu\_Drv.c включает хедер Menu\_Drv.h (определение аппаратно-зависимых функций).

Menu\_Drv.c реализуте функции InitView(), MenuStateMach(), Switch\_to\_EditMode(), EditParam(), ReDrawParam(), ReDrawZeroRow() и re\_draw\_body() и определяет строки-константы для отображения меню на экране.

InitView().

InitView() устанавливает текущий канал 1:

CurChannel = 0;

режим редактирования выключен:

EditMode = 0;

режим измерения включен:

is\_meas\_mode = 1;

Заполнят поля массива структур Channel, для каждого канала задает сигналы, их параметры (доступность, значения) и другую вспомогательную информацию.

Затем перерисовывает верхнюю строку (ReDrawZeroRow()) и остальные строки (re\_draw\_body()) LCD экрана, выключает мигание курсора blink\_control(), посылает сообщения потоку, ответственному за генерацию сигналов (Oscillation thread), о необходимости настроить каналы на генерацию, заданных выше сигналов (preInitDAC()) и включает измерения (посылает сообщение потоку Measuring thread, ответственному за измерения) (meas\_control()).

MenuStateMach().

Каждый раз, когда Poliing button thread посылает сообщение нашему потоку, сообщая таким образом, что произошло какое-то событие с клавиатурой, вызывается функция MenuStateMach(), которая управляет состоянием меню. Первым делом она прекращает свое дальнейшее выполнение, если меню (а значит и весь генератор), находится в режиме измерений и текущее событие не является событием о нажатии кнопок '+' или '-'. Затем идет switch по номеру кнопки, для которой пришло сообщение. Для кнопок с кодами 2-5 сперва выходим из режима редактирования:

EditMode = 0;

, а затем вызываем функцию Switch\_to\_EditMode(), с параметрами соответствующими коду кнопки. Для кнопок с кодом 6 и 7, в режиме редактирования — вызываем функцию EditParam() с параметром, соответствующими коду, иначе — управляем режимом измерения (вкл./выкл. и изменение состояния переменной lock\_send\_message\_to\_calc\_thread) (для кнопки с кодом 6); посылаем сообщение потоку SD writing thread. А он уже следит нужно ли включить запись и разрешить посылку сообщений себе или выключить и запретить посылку сообщений себе. (посредством переменной lock\_send\_message\_to\_sd\_thread) (для кнопки с кодом 7).

Описание реализации потока Measuring thread.

Аппаратные ресурсы, задействованные в реализации:

DMA1 ch 1, GPIOC (0-3), ADC1 (10-13), ADC\_ExternalTrigConv\_T3\_TRGO, ADC\_DMA, DMA1\_Channel1\_IRQn, TIM3, T3\_TRGO, 8kB SRAM.

Реализация потока содержит аппаратно зависимую часть и часть не зависящею от аппаратуры. Реализация представлена четырьмя файлами: .c и .h для части зависящей от аппаратуры и .c и .h для части не зависящей от аппаратуры.

Файлы реализации:

Meas\_Drv.h и Meas\_Drv.c

Meas\_Task.h и Meas\_Task.c

Аппаратно-независимая часть.

Meas\_Task.h.

Meas\_Task.h содержит объявление функции потока vMeasTask(), которая необходима при создании потока в функции Create\_OS\_Objects() (main.c).

Meas\_Task.c.

Meas\_Task.c включает хедеры Meas\_Task.h (определение функции vMeasTask()), Meas\_Drv.h (определение аппаратно-зависимых функций), CommonDefines.h (для взаимодействия с FreeRTOS).

Meas\_Task.c содержит в себе описание функции потока Measuring thread - vMeasTask().

vmeasTask().

Функция vmeasTask() становится в бесконечный цикл ожидания сообщений от других потоков, посредством оператора while и вызова функции FreeRTOS xQueueReceive(), которая ожидает сообщений на очереди queu\_to\_meas. Функция контролирует текущее состояние связки TIM-ADC-DMA и запускает/останавливает эту цепочку, то есть измерение параметров каналов генерации, исходя от сообщения и текущего состояния. Запуск/останов выполянется посредством вызова функций start\_meas()/stop\_meas().

Аппаратно-зависимая часть (“драйвер“).

Meas\_Drv .h.

Meas\_Drv .h содержит объявление функций start\_meas() (запуск аппаратуры ответственной за измерения) и stop\_meas() (останов аппаратуры ответственной за измерения), определение глобального указателя p\_beg\_adc\_buff (используется в обработчике прерывания DMA1\_Channel1\_IRQHandler() ) и некоторые дефайны.

Meas\_Drv.c

Meas\_Drv.c включает хедер Meas\_Drv.h (определение аппаратно-зависимых функций).

Meas\_Drv.c выделяет, статически, память под циклический double-buffer gl\_adc\_buff для хранения измеренных значений и реализует функции start\_meas() и stop\_meas().

start\_meas().

Функция start\_meas() устанавливает глобальный указатель p\_beg\_adc\_buff на циклический double-buffer gl\_adc\_buff, настраивает и запускает цепочку TIM3-ADC1-DMA1.

stop\_meas().

Функция stop\_meas() останавливает цепочку TIM3-ADC1-DMA1. (после останова можно использовать аппаратуру в других целях).

Описание реализации потока Oscillation thread.

Аппаратные ресурсы, задействованные в реализации:

DAC (1, 2), DMA2 ch 3, DMA2\_Channel3\_IRQn, TIM4, TIM4\_TRGO\_event, 4kB SRAM.

Реализация потока содержит аппаратно зависимую часть и часть не зависящею от аппаратуры. Реализация представлена четырьмя файлами: .c и .h для части зависящей от аппаратуры и .c и .h для части не зависящей от аппаратуры.

Файлы реализации:

Osc\_Drv.h и Osc\_Drv.c

Osc\_Task.h и Osc\_Task.c

Аппаратно-независимая часть.

Osc\_Task.h.

Osc\_Task.h содержит объявление функции потока vOscTask(), которая необходима при создании потока в функции Create\_OS\_Objects() (main.c).

Osc\_Task.c.

Osc\_Task.c включает хедеры Osc\_Task.h (определение функции vOscTask()), Osc\_Drv.h (определение аппаратно-зависимых функций), CommonDefines.h (для взаимодействия с FreeRTOS).

Osc\_Task.c содержит реализацию функцию vOscTask().

vOscTask().

Функция vOscTask() инициализирует аппаратуру, ответственную за работу связки DAC-TIM-DMA (InitDAC\_TIM\_DMA()) и становится в бесконечный цикл ожидания сообщений от других потоков, посредством оператора while и вызова функции FreeRTOS xQueueReceive(), которая ожидает сообщений на очереди qTo\_Osc. После прихода сообщения, проверяется откуда оно пришло (по полю Sig\_Type). Если из обработчика прерываний DMA2\_Channel3\_IRQHandler(), то вызывается функция ReFill(), которая сменяет текущую заполняемую область циклического double-buffer DAC\_Buff, измененяет указатель pDAC\_Buff и заполняет текущую область циклического double-buffer DAC\_Buff отсчетами текущих сигналов обоих каналов. Иначе, то есть от потока Menu thread, вызывается функция ReCalc(), которая заполняет следующую область циклического double-buffer DAC\_Buff пересчитанными, согласно полученным данным из сообщения, отсчетами сигнала для ОДНОГО канала.

Аппаратно-зависимая часть (“драйвер“).

Osc\_Drv .h.

Osc\_Drv .h включает хедер InterDefines.h (тут определены типы данных для сообщений).

Osc\_Drv .h содержит объявление функций InitDAC\_TIM\_DMA() (запуск аппаратуры ответственной за генерацию), ReFill(), ReCalc() (реализация double-buffer и обслуживание запросов), и sweep\_control() (управление качением частоты), выделение, статическое, памяти под циклический double-buffer DAC\_Buff, некоторые структуры и переменные, массив указателей на функции, ответственных за вычисления отсчетов конкретных сигналов, переменные для DDS и некоторые дефайны.

Osc\_Drv.c

Osc\_Drv.c включает хедер Osc\_Drv.h (определение аппаратно-зависимых функций).

Osc\_Drv.c реализует InitDAC\_TIM\_DMA(), ReFill(), ReCalc() и sweep\_control().

InitDAC\_TIM\_DMA().

Данная функция заполняет массив указателей на функции, ответственных за вычисления отсчетов конкретных сигналов, адресами соответствующих функций. Заполняет обе части циклического double-bufferа отсчетами zero-сигнала для двух каналов, вызывая 4 раза функцию ReCalc(), устанавливает текущую половину заполняемого буфера (CurFillArea = 1) , для коррктного старта генерации. Затем происходит настройка и запуск аппаратуры, ответственной за генерацию сигналов.

ReFill().

Данная функция (вызывается по событию заполнение половины циклического буфера) меняет указатель на половину циклического буфера, которая должна быть заполнена новыми отсчетами и вызывает функции для заполнения этой половины буфера отсчетами двух каналов.

ReCalc().

Данная функция (вызывается по событию от потока Menu thread) устанавливает текущий канал (CurDAC\_Ch = pOscParam->Ch\_num;), исходя из сообщения от Menu thread, и вызывает функцию для заполнения отсчетами текущей половины циклического буфера для установленного канала. Внутри этой функции происходит пересчет модельки сигнала благодаря first\_entry[CurDAC\_Ch] = 0.

sweep\_control().

Данная функция управляет качением частоты генерируемых сигналов. Следит за направлением изменения частоты, удерживает частоту в заданных рамках :-), и загружает в регистр частоты FREQ\_REG новое, вычисленной, значение.

Описание реализации потока Polling button thread.

Аппаратные ресурсы, задействованные в реализации:

GPIOC(7-9), GPIOA(8-12).

Реализация потока содержит аппаратно зависимую часть и часть не зависящею от аппаратуры. Реализация представлена четырьмя файлами: .c и .h для части зависящей от аппаратуры и .c и .h для части не зависящей от аппаратуры.

Файлы реализации:

PolBut\_Drv.h и PolBut\_Drv.c

PolBut\_Task.h и PolBut\_Task.c

Аппаратно-независимая часть.

PolBut\_Task.h.

PolBut\_Task.h содержит объявление функции потока vPBTask(), которая необходима при создании потока в функции Create\_OS\_Objects() (main.c) и Send\_PolBut\_Message() (отправка сообщения потоку Menu thread).

PolBut\_Task.c.

PolBut\_Task.c включает хедеры PolBut\_Task.h (определение функции vPBTask() и Send\_PolBut\_Message() ), PolBut\_Drv.h (определение аппаратно-зависимых функций), CommonDefines.h (для взаимодействия с FreeRTOS).

PolBut\_Task.c содержит реализацию функцию vPBTask().

vPBTask().

Функция vPBTask() инициализирует аппаратуру, ответственную за работу опроса клавиатуры и становится в бесконечный цикл ожидания получения управления от FreeRTOS, посредством оператора while и вызова функций FreeRTOS xTaskGetTickCount() и vTaskDelayUntil(). После получения управления функция вызывает GetKeyPadState() для проверки, произошло ли какое-либо событие на клавиатуре и, если произошло, посылает сообщение потоку Menu thread, посредством вызова Send\_PolBut\_Message().

Send\_PolBut\_Message().

Формирует сообщение для посылки потоку Menu thread.

Аппаратно-зависимая часть (“драйвер“).

PolBut\_Drv .h.

PolBut\_Drv .h включает определение структуры sKeyMesg и sButtonDescr и определение функций InitPad\_Task(), GetKeyPadState(), initGPIO\_Pads(), GetSst\_Key(), Key\_State\_machine() и выделение памяти под массив sButtonDescr[8].

PolBut\_Drv.c

PolBut\_Drv.c включает хедер PolBut\_Drv.h (определение аппаратно-зависимых функций).

PolBut\_Drv.c реализует InitPad\_Task(), GetKeyPadState(), initGPIO\_Pads(), GetSst\_Key() и Key\_State\_machine().

InitPad\_Task().

Вызывает initGPIO\_Pads() и заполняет поля структур массива ButtonDescr\_K[].

initGPIO\_Pads().

Настраивает аппаратуру, связанную с опросом клавиатуры.

GetKeyPadState().

Вызывает функцию GetSst\_Key() для проверки состояния на линиях ввода/вывода и, если нажата только ОДНА кнопка, вызывает для всех восьми кнопок функцию Key\_State\_machine(), которая следит за их состоянием.

GetSst\_Key().

Проверяет состояние линий ввода/вывода.

Key\_State\_machine().

Реализует state-machine для опроса кнопок. Следит за состоянием кнопки (нажата, отжата и зажата).

Таблица распределения кучи под стеки потоков

|  |  |
| --- | --- |
| Имя потока | Размер стека (в байтах) |
| PBTask | 256\*4 |
| MTask | 800\*4 |
| LCDTask | 256\*4 |
| OscTask | 512\*4 |
| CalcTask | 256\*4 |
| SDTask | 256\*4 |
| MeasTask | 256\*4 |
| Общее потребление памяти: 10368 байт | |

Таблица распределения памяти под глобальные переменные

|  |  |
| --- | --- |
| Тип и имя переменой | Размер (в байтах) |
| sChannel Channel[2] | 2048 |
| unsigned short gl\_adc\_buff[8192] | 16384 |
| uint32\_t DAC\_Buff[1024] | 4096 |
| Общее потребление памяти: 22528 байт | |

Таблица распределения RAM

|  |  |
| --- | --- |
| Тип памяти | Размер (в байтах) |
| Общий стек (под main()) | 800 |
| Куча под стеки потоков | 10368 |
| глобальные переменные | 22528 |
| Куча под очереди, семафоры и служебную информацию FreeRTOS + не учтенные глобальные переменые |  |
| Общее потребление памяти: байт | |

main().

Данная функция получает управление после возникновения прерывания reset, настройки flash-памяти и частоты работы ядра контроллера.

Выводит на LCD строку загрузки "Loading...", производит инициализацию некоторых глобальных переменных, которые должны быть инициализированныдо запуска FreeRTOS, настраивает аппаратуру ответственную за мониторинг ножек контролера (для старта записи измерений на sd- карточку), создает объекты FreeRTOS (потоки, семафоры и мютексы) и запускает планировщик. Ели запуск произошел успешно, то эта функция не получит управление назад. Дальнейший ход программы будет зависеть от FreeRTOS, приоритетов потоков и внешних событий.

Обработчики прерывания.

DMA2\_Channel3\_IRQHandler().

Обслуживает прерывание о заполнение половины буфера от DMA2 Channel 3. Используется для генерации сигналов. Посылает сообщение, с полем Sig\_Type = 7, в голову очереди потоку Oscillation thread. В голову, потому что потоку Oscillation thread может посылать сообщения еще и Menu thread, а реакция на сообщение из обработчика прерывания должна быть максимально быстрой.

DMA1\_Channel1\_IRQHandler().

Обслуживает прерывание о заполнение половины буфера от DMA1 Channel 1. Используется для измерения параметров генерируемых сигналов. Посылает сообщение потоку Calculate thread, если разрешено, и потоку SD write control thread, если разрешено.

TIM5\_IRQHandler().

Обслуживает прерывание о переполнение счетчика таймера TIM5. Используется потоком SD write control thread. Вызывает функцию disk\_timerproc() для нужд FATFs.

EXTI4\_IRQHandler()/EXTI9\_5\_IRQHandler().

Обслуживают прерывание о изменении состоянии ножек контроллера. Используются для запуска записи измеренных данных на sd-карточку, без нажатия кнопки. Блокирует посылку сообщений потоку SD write control thread, из обработчика прерывания DMA1\_Channel1\_IRQHandler, и посылает потоку SD write control thread событие SD\_EVENT.