**Базовые вещи:**

* Код должен быть читаемым, понятным, не нужно использовать сущности и инструменты избыточные. Знание какие инструменты избыточные появляется с опытом.
* Код требуется писать предсказуемый, чтобы любой другой человек мог понять намерение разработчика. Файлы называть понятно, функционал реализовывать в разных файлах в соответстии с логикой.
* Не используемый функционал отключать.
* Мы не пользуемся самыми последними версиями сред, библиотек, прошивок и пр.никаких ночных сборок, только LTS, только стабильные версии всего. В самых свежих версиях альфа, бета и др. куча багов и вещей, которые тормозят разработку. Главная цель — стабильное качество, предсказуемость поведения кода.

**Список правил MIRSA C обязателен к ознакомлению и использованию:**

https://easyelectronics.ru/files/Book/misra\_c\_rus.pdf

**Статический анализатор, который мы используем - регулярно в конце дня проверять код:**

http://cppcheck.net/

**Замечанию по твоему коду:**

Зачем здесь функция бинарного поиска? Давай попробуем делать не усложняя, эта функция не нужна, твою задачу можно сделать через конечный автомат switch, проверку условия и все, никаких массивов на “будущее” делать не нужно. Нужно поступать в твоем случае прямолинейно искать путь решить задачу. При нажатии кнопки начинается анализ значения температуры и установления нового состояния реле.

1. использовать статическое размещение задач, очередей и др. обьектов ОС, можно динамическое но никогда не использовать удаление объектов — фрагментация не нужна.

2. переменные по умолчанию объявлять как static

3. никаких переопределения задач, никаких дефолтных задач, дефолтная задача либо пустая вообще, либо ты ее в дебаге только используешь, чтобы реализовать оценку остатка стека памяти, либо в ней оцениваешь загрузку проца (по тому какой процент времени проц исполняет дефолтную задачу с наименьшим приоритетом)

4. Оценка остатка ресурсов - никаких плагинов в продакшене, только в дебаге, в продакшене все должно быть отключено

5. задачи инициализируются из одного места, не надо инициализировать задачи из задач:

if ( initHardware() == 1){

if(CreateDynamicObject() != 0){

if ( xTaskCreate(vRun,"vRun", 80, NULL, 2, NULL ) != pdTRUE )

task\_done = 0;

if ( xTaskCreate(vButtons,"vButtons", 50, NULL, 1, NULL ) != pdTRUE )

task\_done = 0;

if ( xTaskCreate(vDashboard,"vDashboard", 100, NULL, 2, NULL ) != pdTRUE )

task\_done = 0;

if ( xTaskCreate(vKeyboard,"vKeyboard", 130, NULL, 1, NULL ) != pdTRUE )

task\_done = 0;

if ( xTaskCreate(vShnek,"vShnek", 50, NULL, 2, NULL ) != pdTRUE )

task\_done = 0;

if ( xTaskCreate(vShnekCycle,"vShnekCycle", 100, NULL, 2, NULL ) != pdTRUE )

task\_done = 0;

if ( xTaskCreate(vCO\_GWS,"vCO\_GWS", 50, NULL, 1, NULL ) != pdTRUE )

task\_done = 0;

if ( xTaskCreate(vVentFSM,"vVentFSM", 70, NULL, 3, NULL ) != pdTRUE )

task\_done = 0;

if ( xTaskCreate(vSysMonitoring,"vSysMonitoring", 60, NULL, 2, NULL ) != pdTRUE )

task\_done = 0;

if ( xTaskCreate(vShnekOverHeat,"vShnekOverHeat", 70, NULL, 3, NULL ) != pdTRUE )

task\_done = 0;

if ( xTaskCreate(vShnekRevers,"vShnekRevers", 70, NULL, 3, NULL ) != pdTRUE )

task\_done = 0;

// if ( xTaskCreate(vGSMreceive,"vGSMreceive", 60, NULL, 3, NULL ) != pdTRUE )

// task\_done = 0;

// if ( xTaskCreate(vGSSend,"vGSSend", 100, NULL, 1, NULL ) != pdTRUE )

// task\_done = 0;

// if ( xTaskCreate(vSheduler,"vSheduler", 70, NULL, 1, NULL ) != pdTRUE )

// task\_done = 0;

if ( xTaskCreate(vFullpid,"vFullpid", 50, NULL, 3, NULL ) != pdTRUE )

task\_done = 0;

if ( xTaskCreate(vWDT,"vWDT", 35, NULL, 1, NULL ) != pdTRUE )

task\_done = 0;

if ( task\_done == 1 )

vTaskStartScheduler();//запуск диспетчера задач

}

}

Такая схема позволяет в одном месте выделить под все место и если что-то крашится, можно прям тут и разбираться.

6. Очереди инициализируются из одного места, так же проверяется возвращаемое значение — выделилось успешно или нет. Когда все объекты ОС созданы, приоритеты расставлены — все можно запускать планировщик. Во время работы планировщик объекты мы не удаляем, вообще, размер очередей не меняем, динамически ничего не создаем, если нет особой нужны. Инициализацию всех объектов в ОС лучше также вынести отдельно, чтобы это было не раскидано по задачам.

7. Все задачи должны иметь корректный приоритет и чем реже выполняется задача тем выше приоритет должен быть, каждая задача должна иметь реалистичную периодичность, кроме случаем когда задачу управляет аппаратным ресурсом и находится большую часть времени в suspended и только тогда когда в очередь или семафор приходит что-то она просыпается и делает работу, потому опять проваливается в suspended.

Под реалистично периодичностью имеется в виду, что опрос кнопок не нужно проводить с перерывом 1мс, не нужно тратить процессорное время на это, нужно знать физику переходных процессов чем управляет задача. Например для кнопок 100мс адекватное время, это означает, что задача будет справляться с работой, но будет проц грузить в 100 раз меньше. Если это задачи регулирования, измерение медленно меняющейся температуры — достаточно 1 сек задержки. Процессорное время не резиновое, надо его экономить всегда, это эмбедед.

void vButtons( void \*pvParameters )

{

static int8\_t dir = 0;

for( ;; ){

dir = getMenuButton();

if(dir>0)

xQueueSend(xEncoder, ( void \* )&dir, 0); // отправляем переменную в очередь

dir = getBackButton();

if(dir>0)

xQueueSend(xEncoder, ( void \* )&dir, 0); // отправляем переменную в очередь

vTaskDelay(100);

stack\_size = uxTaskGetStackHighWaterMark(NULL);

taskYIELD();

}

vTaskDelete(NULL);

}

8. когда создаешь очереди и массивы используй дефайны чтобы определять размер

xSIM900Queue = xQueueCreate(SIM900\_QUEUE,sizeof(u8)\*(SIM900\_QUEUE\_BUFFER\_SIZE+ 1));

Если объекты работаю друг с другом и должны иметь один размер, то используй один дефайн на несколько объектов, например на буфер на прием и отправку байт по уарту или обработку этой же полученной из уарт строки.

9. Дефайны если они локальные и не используется выше располагаются в h файле с определенной функциональностью, например SIM900\_QUEUE\_BUFFER\_SIZE используется в файле GSM.h, больше нигде. Если есть дефайн — количество попыток достучатся на сервер, он глобальный, то глобальные вещи описаны SystemDefines.h и больше нигде.

#define DEBUG\_MODE 0 // 1 включен, 0 выключен РЕЖИМ ОТЛАДКИ

#define FLASH\_PROTECTION !DEBUG\_MODE // включена ли защита памяти

#define DEVICE\_INFO "2022 г. v3.0.2 "

#define MODEL "3"

#define FIRMWARE\_VERSION "3"

#define MAIN\_SERVERNAME "http://dm-iot.info/api/v1/"

#define TIMER\_GSM\_PERIOD\_S 90 // периодичность отправки данных на серер

#define LCD\_BACKLIGHT\_TIME 90 // време работы подсветки после последнего действия

10. Про управление реле — это критичная задача, ее нужно реализовать так, чтобы при любой программной ошибке не было возможно одновременно включить реле. Нужно сделать отдельно функцию через которую будет реализовать включение выключение реле, функция будет реализован через семафор ожидающий завершение предыдущей операции с реле + задержка 1 секунда. С обмоток ток долго стекает, чтобы не было скачков во всей технике коммерческой реализованы задержки на переходные процессы : холодильники, машинки стиральные и др.

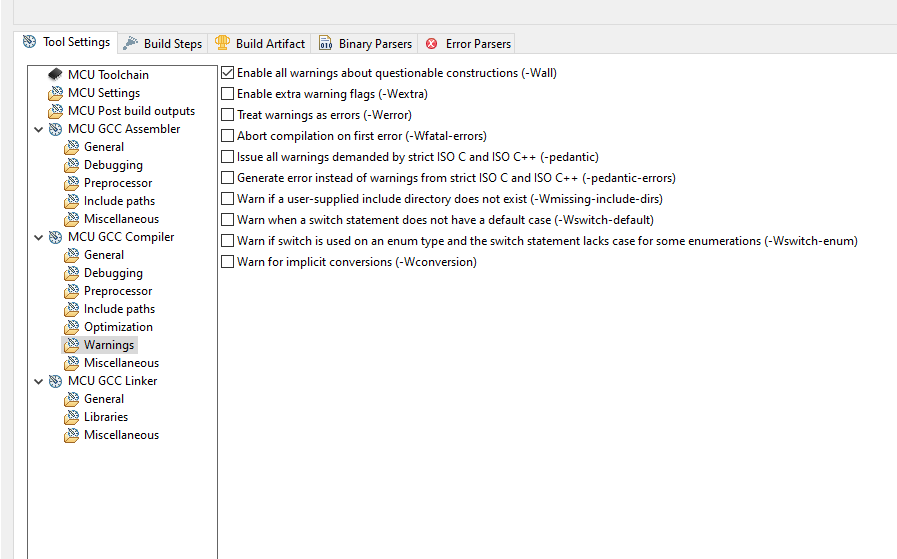
11. В конфига фриртос надо отключить то, что не используется, чтобы уменьшить

<https://en.freertos.org/2020/09/decrease-ram-footprint-and-accelerate-execution-with-freertos-notifications.html>

<https://kalapiinfotech.in/stm32-cube-ide-freertos-memory-management/>

http://www.openrtos.net/Documentation/161204\_Mastering\_the\_FreeRTOS\_Real\_Time\_Kernel-A\_Hands-On\_Tutorial\_Guide.pdf

12. Надо включать все варнинги и другие флаги, которые позволяют рано обнаружить проблемы, компиляторы очень умные сейчас, и по факту они начинают как статические анализаторы работать, я думаю скоро компиляторы вберут в себя функционал статических анализаторов и их отдельно не будет продаваться даже, все будет встроено в компилятор.



Нужно переработать код с учетом замечаний, получить размер билда 10-15 кб