

Прошивку для внутренностей современной встраиваемой электроники написать с нуля практически нереально. На это просто не дают времени. Поэтому ПО для встраиваемых систем создается на базе

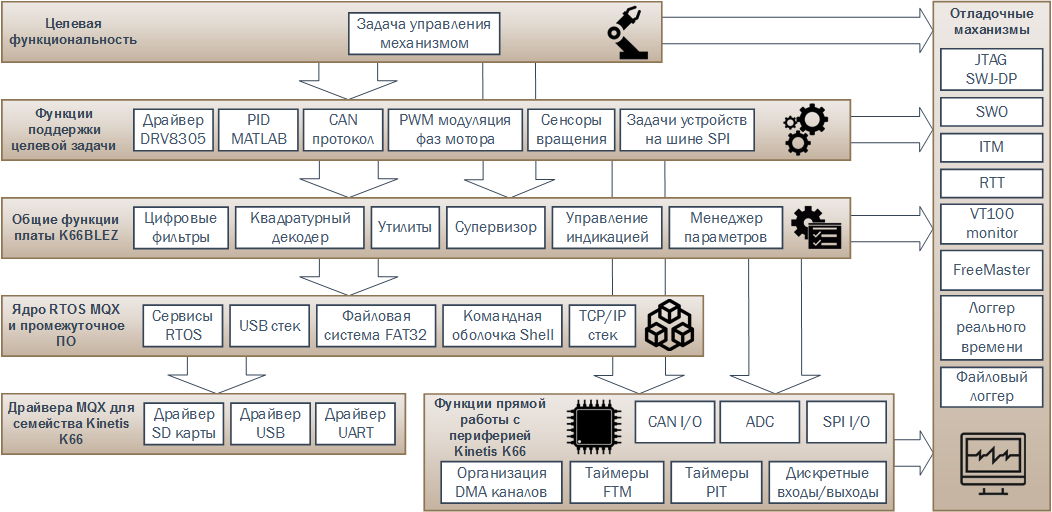
готовых программных платформ - фреймворков. Чем более развит фреймворк, тем быстрее идет разработка. Здесь пойдет речь о фреймворке созданном мной специально для модулей управления моторами и успешно применяемом уже в течении некоторого времени.

<cut />

## Архитектура фреймворка.

Фреймворк предназначен для работы на открытой платформе универсального модуля управления мотором [DMC v2.0](https://geektimes.ru/post/286324/)

Логическую архитектуру фреймворка можно изобразить в виде блок-схемы, показанной ниже. В блок-схеме опущено полное перечисление всех составляющих фреймворка, поскольку это повредило бы наглядности, но для общего представления, я думаю, детализации достаточно.



У фреймворка нет версий, он находится в постоянном развитии, и тут я только попытался зафиксировать его текущее состояние, которое выложено на GitHub.

С точки зрения программных модулей фреймворк содержит следующие ключевые части:

- Набор модулей поддержки прикладного уровня

- Операционная система реального времени **MQX**.

- ПО промежуточного уровня: **файловая система, командная оболочка, стеки коммуникационных протоколов** и т.д.

- Пакет низкоуровневой поддержки платы (board support package или BSP), включающий функции **доступа к периферии** плат модуля BLEZ66V1 и силового модуля.

- Пакет ПО для **мониторинга, отладки и диагностики** - [FreeMaster](http://www.nxp.com/products/power-management/wireless-charging-ics/freemaster-run-time-debugging-tool:FREEMASTER).

- Отладочные средства включающие [**RTT**](https://habrahabr.ru/post/259205/)**,** инструментальную трассировку[**ITM**](https://www.arm.com/files/pdf/AT_-_Advanced_Debug_of_Cortex-M_Systems.pdf)**, логгеры, VT100** терминал.

- Модуль **генерации файлов** параметров.

- Модули сгенерированных исходных текстов на основе алгоритмов в **Matlab**.

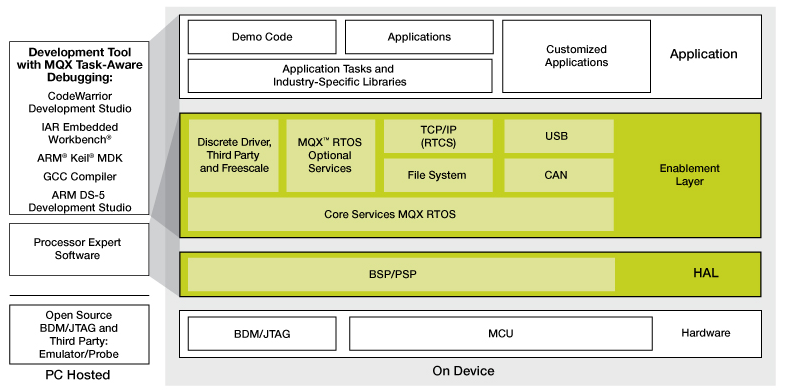
## Почему выбрана MQX?

Операционная система реального времени (real time operating system, RTOS) MQX известна давно, но

появилась в свободном доступе пару лет назад. Эту операционную систему выложила фирма Freescаle до того, как их купила NXP. RTOS поначалу имела лицензию только для использования в микроконтроллерах Freescale, теперь лицензия распространяется и на продукцию NXP. RTOS пережила взрывную популярность, прошла несколько апгрейдов до версии 4.2 после чего Freescale решила ее последующие версии снова сделать коммерческими. Так получилось две версии, одна открытая и замороженная в развитии под названием MQX Classic (она же MQX v4.2) и закрытая коммерческая MQX 5.0.

В описываемом фреймворке используется ветка MQX Classic v4.2. Это стабильная, хорошо протестированная версия. Лицензия позволяет разработчику модификацию исходников MQX Classic и использование ее в коммерческих продуктах, однако не позволяет публиковать MQX Classic в виде исходных текстов. Но это не должно быть проблемой, поскольку MQX Classic доступна для свободного скачивания.

Структура RTOS в общем плане выглядит так:



## Зачем вообще нужна RTOS?

Комплексное приложение где требуется в частности управления мотором достаточно сложное и состоит из множества асинхронных задач, каждая со своим циклом повторения и событиями ее активизирующими и останавливающими. Если мы все эти задачи выполняем в одном суперцикле, то неизбежно сталкиваемся с проблемой задержки выполнения одних задач другими задачами.

С помощью RTOS, такой как MQX удается избавится от взаимозависимости отдельных задач на оси времени, не переписывая их и даже не глядя в их исходные коды.

Например, наша задача логгера может как угодно долго пытаться записать сообщение на SD карту ожидая ее отклика, задача USB может вся уйти в передачу большого массива данных на компьютер, но при этом задача PID алгоритма мотора будет выполняться строго с заданным интервалом, а задача измерения скорости вращения не пропустит ни одного события изменения сигналов энкодера.

Хотя надо признать, есть другой все более популярный способ избавится от комплексности на одном чипе - переход на мультипроцессорность, но и в этом деле RTOS окажет хорошую услугу.

## Главные достоинства RTOS MQX.

- Ядро системы поставляется с широким набором промежуточного программного обеспечения включающего файловую систему, TCP/IP стек, USB стек, командную оболочку и т.д. Все в исходных текстах.

- Готовые наборы BSP для разных плат, избавляющие от написания собственных библиотек работы с периферией.

- Подробная документация в pdf фалах с удобной навигацией.

- Наличие плагина к IDE IAR embedded workbench c весьма детализированной информацией о внутренних структурах RTOS, гораздо детальнее чем для других известных RTOS - uCOS и FreeRTOS.

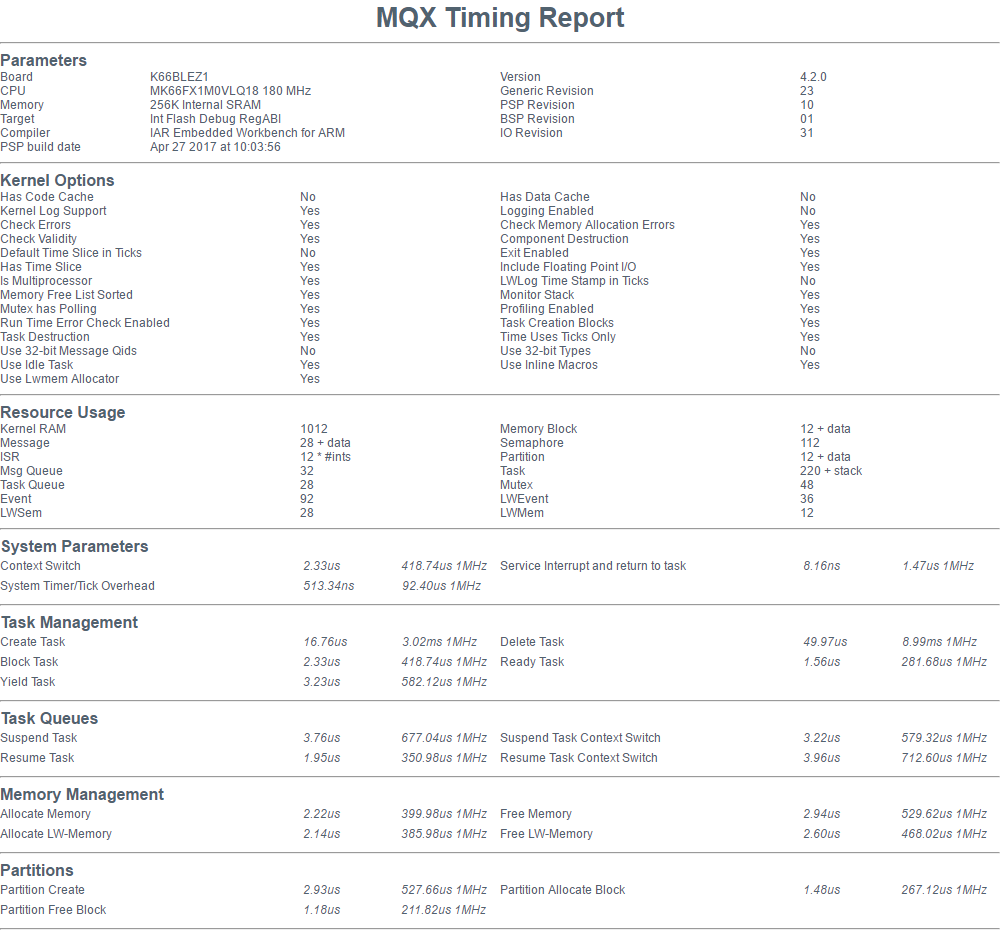
- Множество примеров применения RTOS и тестовых примеров.

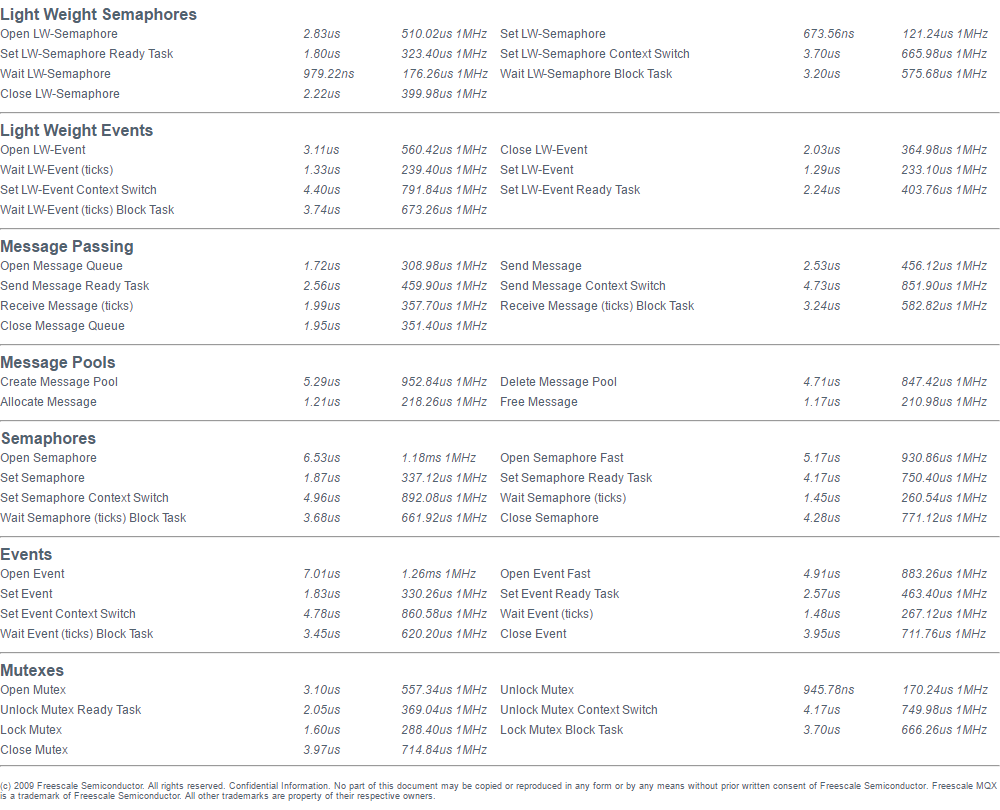
Когда говорят о RTOS, то всегда подчеркивают их способность выполнять задачи вовремя, но количественные оценки чаще всего не приводятся или приводятся для каких-то отдельно выбранных сторонних платформ. Этого явно недостаточно чтобы реализовать управление в жестком реальном времени используя RTOS. А для управления двигателями нужно именно жесткое реальное время.

У MQX есть на этот счет замечательный тестовый пример, который позволяет получить подробнейшую таблицу времени исполнения всех сервисов на той платформе, на которой вы запустили тест.

Ниже дана таблица времени исполнения сервисов на микроконтроллере нашего модуля управления мотором, при включенной в компиляторе максимальной оптимизации по скорости выполнения кода.

<spoiler title="Тайминги исполнения сервисов RTOS MQX Classic">



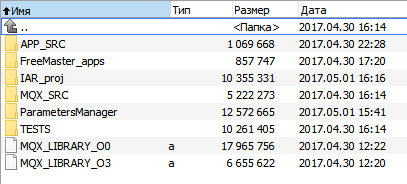


</spoiler>

Таблица также дает представление о том какие сервисы поддерживаются RTOS и какие опции ядра доступны. Проект теста в среде IDE IAR включен в публикуемый фреймворк.

## Состав директорий проекта

Корневой каталог фреймворка выглядит так:



**APP\_SRC** - директория, содержащая все исходники за исключением тех, что относятся к дистрибутиву MQX.

**FreeMaster\_apps** - файлы проектов для выполнения в среде [FreeMaster](http://www.nxp.com/products/power-management/wireless-charging-ics/freemaster-run-time-debugging-tool:FREEMASTER) .

**IAR\_proj** - файлы рабочего пространства и проекта для среды IAR embedded workbench for ARM v7.70.2. В данной среде компилируется и отлаживается конечное приложение.

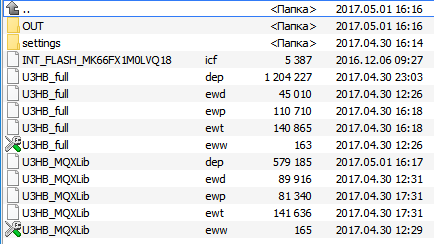
**MQX\_SRC** - директория, содержащая все исходники MQX и промежуточного ПО поставляемого с MQX. Поскольку лицензия не разрешает открытую публикацию исходников из дистрибутива MQX, то в данной директории отсутствуют '**.с'** и '**.s'** файлы. Но тем, кто согласился с лицензионными условиями NXP, могут быть высланы недостающие файлы.

**ParametersManager** - директория программы менеджера параметров. С помощью этой программы создаются списки параметров приложения и генерируются '**.с'** и '**.h'** файлы с объявлениями параметров для встраивания в приложение.

**TESTS** - директория с проектами тестов фреймворка. Здесь находится проект **MQX\_benchmark** для генерации отчета с таймингами MQX.

Файлы **MQX\_LIBRARY\_O0.a** и **MQX\_LIBRARY\_O3.a** - это скомпилированное в библиотеки содержимое директории **MQX\_SRC** с минимальной оптимизацией и максимальной оптимизацией соответственно.

## Содержимое директории IAR\_proj



Файлы **U3HB\_full.eww** и **U3HB\_MQXLib.eww** - это файла рабочих пространств среды IAR.

Покуда в директории MQX нет исходников, то работать будет только файл **U3HB\_MQXLib.eww**. В этом рабочем пространстве используются скомпилированные библиотеки MQX.

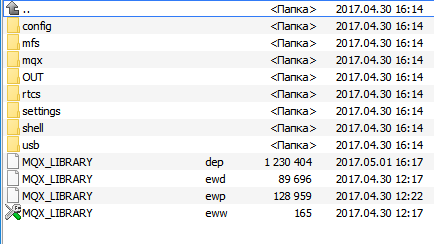
В рабочем пространстве **U3HB\_full.eww** компилируются полные исходники MQX.

Директория **OUT** служит местом, куда среда IAR помещает продукты своей работы, в частности map и hex файлы.

Директория **settings** создается средой IAR автоматически. В ней в частности хранятся настройки отладчика. Если что-то не удается настроить при отладке в IAR, то иногда стоит стереть эту директорию.

Файл **INT\_FLASH\_MK66FX1M0LVQ18.icf** является настроечным файлом линкера IAR. Он определяет адреса регионов памяти куда помещаются линкером области кода, данных, векторов прерываний, стеки и т.д.

## Содержимое директории MQX\_SRC

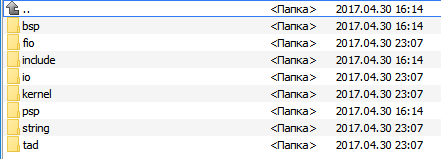


Файлы рабочего пространства **MQX\_LIBRARY.eww** служит для создания библиотек MQX. Пока не размещены '**.с'** и '**.s'** файлы в директориях этот проект компилироваться не будет.

**config** - директория с файлами конфигурации MQX. Состав сервисов и драйверов MQX задается в файле конфигурации **user\_config.h**.

**mfs** - файловая система MQX, включает FAT32 и RAM FS

**mqx** - ядро MQX включает такие поддиректории:



**rtcs** - TCP/IP стек. Включает следующие поддиректории:



**shell** - директория с файлами командной оболочки.

**usb** - директория с файлами стека USB

Функциональность каждого программного модуля MQX хорошо документирована производителем.

В качестве примера приведу ссылки на два документа:

[Инструкция по использованию MQX](https://github.com/Indemsys/Universal3PHalfBridgeController/blob/master/Docs/MQX_Reference_Manual.pdf).

[Справочное руководство по MQX.](https://github.com/Indemsys/Universal3PHalfBridgeController/blob/master/Docs/MQX_Reference_Manual.pdf)

Остальное надо искать в дистрибутиве, [который доступен на сайте NXP](http://www.nxp.com/products/software-and-tools/run-time-software/mqx-software-solutions/mqx-real-time-operating-system-rtos/mqx-classic-software-solutions:MQX)

Сам фреймворк находится в репозитории здесь - <https://github.com/Indemsys/Universal3PHalfBridgeController/tree/master/K66_Firmware>

Дальнейшее описание работы с ПО и примеры применения - в следующих статьях.