1. Общие принципы работы PDM.
   1. Силовые выхода.

В модуле PDM есть 8 каналов на номинальный ток до 20А и 12 каналов на номинальный ток до 8А. Номинальный ток для каждого канала может быть задан программно. В связи с отсутствием в устройстве термокомпенсации измерений, в процессе работы устройства под высокой нагрузкой, реальная гарантированная точность измерения тока составляет ….

1. **Написание пользовательских программ.**
   1. **Исполняемые скрипты. Основные тезисы**

Пользовательские алгоритмы в системе PDM-Sider пишутся на высокоуровневом языке программирования LUA (версия 5.4.4, описание языка и всю дополнительно информацию по нему, в том числе на русском языке, можно найти на http://www.lua.org/).). Для пользователя доступен весь функционал и синтаксис языка, а сложность программы ограничена только размерами оперативной и FLASH памяти конкретной модели PDM. Программа для системы PDM-Sider может быть исполнена в любой другой среде LUA, например на персональном компьютере (на автономной LUA машине, файлы для разных операционных систем доступны на сайте http://www.lua.org/). Эту особенность можно использовать для отладки сложных алгоритмов, поскольку можно написать на LUA не только саму программу для PDM, но и скрипт для тестирования этой PDM. Подробнее об этом в разделе тестирования.

Структура программы на LUA для системы PDM-Sider должна соответствовать правилам, описанным в разделе 2.2. Весь алгоритм работы PDM описываются в виде кода в блоке рабочего цикла. Рабочий цикл, как следует из названия, постоянно исполняется ядром системы. После каждого рабочего цикла ядро производит обработку данных. В

* 1. **Правила написания скрипта.**

Структурно скрипт LUA для системы PDM-Sider должен состоять из определённых блоков, расположенных в следующем порядке:

**[блок конфигурации]**

**[блок пользовательских функций]**

**[блок рабочего цикла]**

* + - 1. **Блок рабочего цикла программы**

В общем случае шаблон пользовательского кода должен выглядеть следeющим образом:

**main = function ()** --функция main. Ее вызывает ядро PDM-Сайдер после старта.

**function stop() In1 = coroutine.yield(Out20,Out19…0ut1) end –**

**while true do** -- бесконечный цикл

**stop()** --обмен данными с ядром

**[Пользовательский код]**

**end** -- конец бесконечного цикла

**end** – конец функции main

Пользовательский код должен быть размещена внутри бесконечного цикла в функции main. Для обмена данными с ядром необходимо вызывать функцию stop внутри бесконечного цикла, перед началом пользовательского кода.

*Функция main – это предопределенная точка входа в пользовательскую программу, т.е. после старта системы и загрузки скрипта, ядро PDM-Сайдер пытается запустить из скрипта функцию c именем main. Можно писать любое количество сервисных функций, однако имя main зарезервировано для осинового рабочего цикла.*

***NB! LUA позволяет объявить функцию и таким образом function main(), это тоже правильный синтаксис, однако при таком написании из-за особенностей языка LUA, PDM не сможет ее увидеть в скрипте и запустить.***

*Непосредственно алгоритм работы устройства должен быть помещен в любой бесконечный цикл внутри функции main. В шаблоне это цикл “while true do”. Это необходимо именно потому, что пользовательская программа полноправный процесс, и если код не помещён в бесконечный цикл, то он будет выполнен только один раз.*

***NB! Внутри функции main можно вставлять любый циклы в любое место, однако они не должны быть бесконечными. В противном случае программ никогда не сможет из них выйти, что приведет к перезагрузке системы по watchdog.***

*Внутри системы PDM-Sider пользовательская программа реализована виде потока LUA (thead). Поэтому, что бы процессор PDM выполнял не только пользовательские задачи, но и запускал внутренние системные процессы ( например контроля тока через выходные ключи, обмена по CAN) пользовательская программа должна приостанавливать периодический свое исполнение. Это делается через специальную функцию LUA coroutine.yield. Тут происходит обмен данными с PDM. Когда LUA программа доходит до этого вызова, она передает в PDM управляющие данные для выходов, а при возобновлении работы получает из PDM значения токов и другие данные.*

*Для удобства работы, в шаблоне вызов coroutine.yield оформлен в виде локальной функции stop(). Резюмирая, пользовательский алгоритм должен быть размещен внутри бесконечного цикла в функции main.*

* + - 1. **Блок конфигурации.**

**В общем случае блок конциграции должен выглядить следующим образом**

**Out1**

После старта PDM, ядро настраивает все параметры аппаратуры по умолчанию. Если параметры по умолчанию соответствуют необходимым, то в конфигурации конкретно этих параметров необходимости нет. К тому же, явная конфигурация из LUA скрипта увеличивает его размер. Однако, для повышения надежности всей системы, явная конфигурация в LUA скрипте предпочтительна. Это позволит избежать проблем, при последующих изменениях LUA кода.

Конфигурирование модуля PDM происходит через вызов специальных функций ядра PDM. Ядро регистрирует эти функции в среде LUA перед загрузкой скрипта. После этого, эти функции доступны для вызова из LUA программы. Для пользователя это означает, что функции конфигурации можно вызывать в любом месте программы любое кол-во раз. При этом, смена конфигурации происходит в реальном времени (т.е. в момент вызова соответствующей функции).

**NB! При запуске скрипта на автономной LUA машине необходимо использовать функции “заглушки”. Подробнее о запуске программы в автономной LUA машине см. соответствующий раздел.**

**Конфигурация мощности и режима плавного пуска.**

OutConfig(пар1,пар2,пар3,пар4)

пра1 – номер канала от 1 до 20

пар2 – номинальный ток канала. 0.1-20.0 для каналов 1-8 и 0.1-8.0 для каналов 9-20.

Значения по умолчания 20.0 для квналов 1-8 и 8.0 для каналов 9-20.

пар3 – время плавного пуска в милисекундах, от 0 до 65535. Значения по умолчания 1000 для каналов 1-8 и 0 для каналов 9-20.

пар4