|  |  |
| --- | --- |
|  | УТВЕРЖДАЮ  Главный инженер  ООО «Компания «Стальэнерго»  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Ю.А. Федоркин  « » 2019 г. |

**ОБЪЕКТНЫЙ КОНТРОЛЛЕР ПРИВОДА СТРЕЛКИ**

**ОКПС-Е-К**

Требования к реализации программного обеспечения

ЕИУС.хххххх.ххх 01 92 01 01

|  |  |
| --- | --- |
| СОГЛАСОВАНО | РАЗРАБОТАЛ |
| Письмом заведующего  испытательным центром  ЖАТ ПГУ ПС  Исх. \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  от \_\_.\_\_.\_\_\_\_ г. | Начальник отдела разработок новых видов продукции  ООО «Компания «Стальэнерго»  \_\_\_\_\_\_\_\_\_\_ И.В. Солодовник  « » 2019 г. |

**История изменений**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| п/п | Дата | Номер версии | Статус | Автор | Причина изменения, № извещения | Комментарии |
| 1 | 14.10.2019 | 1.0 | Проект | Агулов М.А. |  | Создание документа |
| 2 |  | 1.1 |  |  |  |  |

**Содержание**

[1 Введение 6](#_Toc46320737)

[1.1 Назначение документа 6](#_Toc46320738)

[1.2 Термины, определения и сокращения 6](#_Toc46320739)

[1.3 Рекомендации к ознакомлению 6](#_Toc46320740)

[1.4 Ссылки 7](#_Toc46320741)

[2 Описание среды разработки 8](#_Toc46320742)

[3 Функциональное назначение ПО 9](#_Toc46320743)

[4 Реализации компонентов 10](#_Toc46320744)

[4.1 Уровень управления потоками и режимами 10](#_Toc46320745)

[4.1.1 Компонент Main 10](#_Toc46320746)

[4.2 Уровень режимов 25](#_Toc46320747)

[4.2.1 Компонент Fsm 25](#_Toc46320748)

[4.2.2 Компонент FsmStates 29](#_Toc46320749)

[4.3 Уровень компонент 41](#_Toc46320750)

[4.3.1 Компонент ActivityManager 41](#_Toc46320751)

[4.3.2 Компоненты AnalogInput, AnalogMeasurement 59](#_Toc46320752)

[4.3.3 Компонент BinIn 68](#_Toc46320753)

[4.3.4 Компонент BlackBox 75](#_Toc46320754)

[4.3.5 Компонент CheckSupply 76](#_Toc46320755)

[4.3.6 Компонент ConfigMK 78](#_Toc46320756)

[4.3.7 Компонент DebugTools 79](#_Toc46320757)

[4.3.8 Компонент DeviceAddress 80](#_Toc46320758)

[4.3.9 Компонент Eeprom 84](#_Toc46320759)

[4.3.10 Компонент HysteresisFilter 85](#_Toc46320760)

[4.3.11 Компонент Indication 87](#_Toc46320761)

[4.3.12 Компонент Initial 89](#_Toc46320762)

[4.3.13 Компонент IntegrCtrl 93](#_Toc46320763)

[4.3.14 Компонент InterChannel 98](#_Toc46320764)

[4.3.15 Компонент ModeProtection 137](#_Toc46320765)

[4.3.16 Компонент OverloadDet 141](#_Toc46320766)

[4.3.17 Компонент PositionDet 144](#_Toc46320767)

[4.3.18 Компонент RelayCtrl 147](#_Toc46320768)

[4.3.19 Компонент RS422 151](#_Toc46320769)

[4.3.20 Компонент SafetyPowerControl 169](#_Toc46320770)

[4.3.21 Компонент ShuntShift 172](#_Toc46320771)

[4.4 Системные каталоги 176](#_Toc46320772)

[4.4.1 Drv 177](#_Toc46320773)

[4.4.2 Systems 195](#_Toc46320774)

[4.4.3 Testing 213](#_Toc46320775)

[4.4.4 Utility 215](#_Toc46320776)

[5 Список литературы 216](#_Toc46320777)

1. Введение
   1. Назначение документа

Этот документ предоставляет информацию для разработки компонентов.

Документ распространяется на объектный контроллер привода стрелки ОКПС-Е-К (далее по тексту – прибор) для обеспечения безопасного управления стрелочным приводом путем формирования трехфазного напряжения для питания электродвигателей переменного тока в 9-ти проводных схемах управления.

* 1. Термины, определения и сокращения

Для применения данного документа имеют значение термины, определения и сокращения «Объектный контроллер привода стрелки. ОКПС-Е-К. ЕИУС.хххххх.ххх 96.01».

* 1. Рекомендации к ознакомлению

Этот документ:

* определяет детальную архитектуру компонентов ПО прибора;
* описывает основные алгоритмы компонентов;
* соответствует функциональным требованиям к ПО;
* должен корректироваться при изменении аппаратной реализации, влияющей на алгоритмы ПО;
* не предназначен для разработки поддерживающего ПО (ПО стендов регулировки, ПО стендов проверки, ПО средств отладки);
* не предназначен для рассмотрения коммерческих вопросов.

Данный документ предоставляет дополнительную информацию для реализации конкретных компонентов, которая не отражена в функциональных требованиях к ПО «Объектный контроллер привода стрелки. ОКПС-Е-К. ЕИУС.хххххх.ххх 93.01» и архитектуре ПО «Объектный контроллер привода стрелки. ОКПС-Е-К. ЕИУС.хххххх.ххх 94.01».

При реализации ПО, кроме этого документа, необходимо использовать функциональные требования к ПО «Объектный контроллер привода стрелки. ОКПС-Е-К. ЕИУС.хххххх.ххх 93.01» и архитектуру ПО «Объектный контроллер привода стрелки. ОКПС-Е-К. ЕИУС.хххххх.ххх 94.01».

Описание функций, переменных, констант находиться в описании ПО «Объектный контроллер привода стрелки. ОКПС-Е-К. ЕИУС.хххххх.ххх 13.01». В этом документе они приводятся в объеме, необходимом для проектирования работы компонента. Описание интерфейсных функций компонентов приведено в архитектуре ПО «Объектный контроллер привода стрелки. ОКПС-Е-К. ЕИУС.хххххх.ххх 94.01».

* 1. Ссылки

1. технические требования «Объектный контроллер привода стрелки (ОКПС-Е-К)»;
2. функциональные требования к ПО «Объектный контроллер привода стрелки. ОКПС-Е-К. ЕИУС.хххххх.ххх 93.01»;
3. архитектура ПО «Объектный контроллер привода стрелки. ОКПС-Е-К. ЕИУС.хххххх.ххх 94.01»;
4. Unified Modeling Language Specification Version 2.5.
5. Описание среды разработки

ПО написано на языке Си, откомпилировано и скомпановано программными средствами пакета MPLAB X IDE v.5.15 фирмы Microchip. Использован компилятор XC16 версии 1.33.

1. Функциональное назначение ПО

ПО прибора выполняет следующие основные функции:

* обмен информацией с УС по интерфейсу RS-422;
* формирование трёхфазного питающего напряжения для электродвигателя стрелочного привода с чередованием фаз, согласно заданному направлению перевода в приказе от УС;
* контроль тока по каждой из фаз электродвигателя и выдача его значения в УС;
* контроль обрыва и перегрузки по каждой из фаз электродвигателя и выдача информации об этом в УС;
* периодический контроль состояния питающего фидера и обмоток электродвигателя при отсутствии перевода;
* формирование сигнала контрольного генератора для определения положения фактического стрелки;
* приём сигналов от датчика положения стрелки и выдачу в УС текущем положении стрелки;
* обеспечивает индикацию на передней панели прибора;
* автоматическое переключение на резервный прибор;
* контроль работоспособности узлов прибора и переход в 3С в случае обнаружения неисправности.

1. Реализации компонентов

   2. Уровень управления потоками и режимами

В ПО прибора реализовано четыре потока управления (см. раздел 4 (Потоки управления) документа «Архитектура ПО»).

* + 1. Компонент Main
       1. Назначение

Управление главным потоком (ГП), потоком временной синхронизации (ПВС), потоком синхронизации данных (ПСД) и потоком неиспользуемых прерываний.

* + - 1. Состав

В состав компонента входят следующие модули:

* *ControlSystem* – управление работой прибора;
* *InterruptsHandlers* – управление ПВС, обработчики всех прерываний;
* *Main* – управление ГП;
* *UnusedInterrupts* – обработка всех неиспользованных прерываний, управление потоком неиспользуемых прерываний;
  + - 1. Описание

Модуль Main осуществляет:

1. инициализацию всех компонент;
2. управление ГП в фоновом режиме путем вызова диспетчера режимов – функции ***Sheduler\_run()***;
3. управление ПВС путем обработки прерываний главного таймера и по изменению уровня сигнала временной синхронизации;
4. контроль времени выполнения цикла главного потока – 1 мс;
5. счет системного времени – ***cTimeWork()***;
6. сброс сторожевого таймера – ***ClrWdt()***.

Модуль InterruptsHandlers (обработка ПВС) осуществляет:

1. временная синхронизация каналов Мaster и Slave прибора;
2. контроль сигнала временной синхронизации по времени и фазе;
3. вызов следующих функций, требующих жесткой временной привязки и синхронности выполнения в обоих МК:
4. ***SafetyPowerControl\_runInterrupt()*** – контроль безопасного блока питания;
5. ***BinIn\_interruptDataRead()*** – синхронное чтение дискретных входов;
6. ***AnalogMeasurement\_runInterrupt()*** – чтение результатов измерения АЦП;
7. ***BinIn\_interruptAddrSet()*** – синхронная установка адреса для дискретных входов;
8. ***CheckRAM\_run()*** – циклический контроль исправности ОЗУ;
9. ***InterChannel\_runDrv()*** – опрос драйвера готовности межканального обмена;
10. ***Rs422\_interrupt()*** – драйвер обмена по RS422 с УС.
11. ***ShuntShiftGen\_interrupt()*** – драйвер ШИМ-генератора привода электродвигателя;
12. ***PosDetGenerator\_interrupt()*** – драйвер ШИМ-генератора сигнала контрольного генератора для определения положения стрелки;
13. ***BlockExch\_interrupt()*** – драйвер межблочного обмена (между основным и резервным прибором).
14. запуск таймера МКО (CAN\_TIMER\_START).
15. запуск таймера потока синхронизации данных.

Модуль ControlSystem осуществляет:

1. обработку приказов, поступающих от УС и формирования статусов ответа для УС;
2. определение положения стрелки;
3. контроль целостности обмоток двигателя;
4. управление индикацией на передней панели прибора.

Модуль UnusedInterrupts осуществляет обработку всех неиспользуемых прерываний МК. Также к неиспользуемым прерываниям относятся исключения, генерируемые процессором. К ним относятся такие операции, как попытка деления на ноль, попытка выполнить несуществующую операцию или обратиться к недопустимой области памяти и др. При возникновении таких прерываний прибор переходит в 3С.

* + - * 1. Настройка битов конфигурации МК

Настройка битов конфигурации МК в регистрах конфигурации выполняется единоразово в компоненте Main сразу при включении прибора. Во время выполнения ПО не использует операции записи в конфигурационные биты регистров конфигурации. При работе программы состояние регистров конфигурации периодически контролируется.

Настройка системы тактирования

Система тактирования служит для получения частоты синхронизации ядра. Частота синхронизации ядра, обозначаемая далее *FCY*, является частотой выполнения команд процессором.

Режим тактирования процессора задаётся в два этапа. При включении питания процессор использует встроенный RC-генератор для получения частоты синхронизации ядра (первый этап). Это сделано для гарантированной работы процессора на стабильной частоте, пока не будет установлен режим работы PLL и частота стабилизируется. При переключении на работу от PLL и стабилизации частоты PLL, процессор переключается на работу от PLL (второй этап). Настройку PLL и переключение на работу от PLL выполняет функция ***ClockConfig()***.

Первый этап – выбор источника тактирования и алгоритм переключения между ними задаётся конфигурационными регистрами при программировании процессора. Второй этап – конфигурация PLL – задаётся программно в процессе выполнения программы.

Тип используемого источника опорного сигнала и его режим работы определяется двумя конфигурационными регистрами - FOSС и FOSCEL. Установка битовых полей и их назначение показаны в Таблице 1 и Таблице 2. Эти регистры расположены во флэш-памяти программ МК и записываются при программировании процессора. При запуске процессора эти регистры определяют один из возможных источников опорной частоты. Так, поле FNOSC в регистре FOSCEL определено как FRC, что определяет запуск процессора при включении питания от встроенного RC-генератора. Также бит IESO в регистре FOSCSEL определён как Off, что определяет запуск процессора от RC-генератора и ручное переключение на другой тип генератора.

Таблица 1 – Регистр конфигурации FOSС

| **Поле** | **Значение** | **Назначение** | **Установка** |
| --- | --- | --- | --- |
| *POSCMD* | *EC* | Primary Oscillator Mode Select Bit | EC (External Clock mode) |
| *OSCIOFNC* | *Off* | OSC2 Pin Function bit | OSC2 is clock output |
| *IOL1WAY* | *Off* | Peripheral pin select config | Allow multiply configuration |
| *FCKSM* | *CSECMD* | Clock Switching Mode bits | Clock switching is enabled Fail-safe clock monitor is disabled |

Таблица 2 – Регистр конфигурации FOSСEL

|  |  |  |  |
| --- | --- | --- | --- |
| **Поле** | **Значение** | **Тип** | **Установка** |
| *FNOSC* | *FRC* | Oscillator Source Selection | Internal Fast RC (FRC) |
| *PWMLOCK* | *Off* | PWM Lock Enable Bit Enable | PWM registers may be written without key sequence |
| *IESO* | *Off* | Two-Speed Oscillator Start-up Enable bit | Start up with user-selected oscillator source |

При запуске процессор работает от встроенного RC-генератора. Далее следует переключение на частоту, получаемую с PLL. Переключение происходит в функции ***ClockConfig()****.* В качестве источника опорной частоты для PLL используется внешний генератор (POSCMD=EC), сигнал которого подаётся на вход OSC1. После того как режим работы PLL будет настроен, частота PLL стабилизируется, синхронизация переключится на работу от PLL.

Используемый в проекте МК dsPIC33EP512GM710 имеет систему тактирования на основе фазовой автоподстройки частоты – ФАПЧ (PLL), структурная схема которой приведена на рисунке 1. Подробно система тактирования процессора и её настройка рассмотрена в [2].



Рисунок 1 – Cтруктурная схема тактирования МК dsPIC33EP512GM710

Функция ***ClockConfig()*** выполняет настройку системы тактирования процессора. В качестве источника синхронизации (FIN) МК используется внешний кварцевый генератор с частотой 14,7456 МГц. Сигнал с его выхода подаётся на вывод МК CLK1.

Для обеспечения нормальной работы частотно-фазового детектора (PFD) частота FREF на его входе должна быть в пределах 3…5,5 МГц. Для этого выбираем коэффициент деления прескаллера *N1*=4. При этом частота FREF на входе PFD составит 3,6864 МГц.

Частотой FOSC на выходе системы тактирования связана с частотой FIN выражением:

где:

|  |
| --- |
|  |

Частота FVCO вычисляется по формуле:

Для достижения максимальной производительности ядра частота FOSС должна быть 120 МГц.

Для этого выбираем *M=64*. При этом:

|  |
| --- |
|  |

Частота выполнения инструкций процессором вычисляется по формуле:

|  |
| --- |
|  |

Время выполнения одной однотактной ассемблерной команды составит:

|  |
| --- |
|  |

Значения конфигурационных регистров определяется по формулам XXX.

Такими образом, производительность ядра составляет 58,9824 MIPS, что соответствует времени выполнения одной команды 16,954 нс.

Настройка сторожевого таймера

Сторожевой таймер предназначен для сброса процессора при обнаружении неправильной работы программы. Период срабатывания сторожевого таймера определяется двумя делителями – предделителем (WDTPRE) и постделителем (WDTPOST). Значения делителей хранятся в памяти программ процессора, в конфигурационном слове FWDT, задаются на этапе программирования процессора и не могут быть изменены программно.

Режим работы сторожевого таймера (WDT) при запуске процессора определяются регистром конфигурации FWDT (см. Таблицу 3).

Таблица 3 – Регистр конфигурации сторожевого таймера *FWD*

|  |  |  |  |
| --- | --- | --- | --- |
| **Поле** | **Значение** | **Тип** | **Установка** |
| *WDTPRE* | *PR32* | Watchdog timer prescaler bit | 1:32 |
| *WDTPOST* | *PR1024* | Watchdog timer postscaler bit | 1:1024 |
| *WINDIS* | *Off* | Watchdog timer Window enable bit | Watchdog timer in Non-Window mode |
| *FWDTEN* | *Off* | Watchdog timer Enable bit | Watchdog timer enabled/disabled by user software |

Из таблицы 3 видно, что значение предделителя WDTPRE установлено 32, постделителя WDTPOST установлено 1024. Время срабатывания сторожевого таймера определяется выражением:

Режим окна, определяемый битом, WINDIS выключен. Сторожевой таймер по включению процессора выключен (бит FWDTEN установлен в состояние Off). В процессе выполнения программа включает сторожевой таймер, после чего он начинает работать. Включается сторожевой таймер макросом ENABLE\_WATCHDOG. В этом макросе бит SWDEN в регистре RCON устанавливается равным 1. Далее в программе сторожевой таймер нигде не выключается. В каждый миллисекундный цикл происходит сброс сторожевого таймера функцией ***ClrWdt()***.

Настройка сброса по включению и провалу напряжения питания процессора

Процессор имеет встроенные средства контроля снижения напряжения питания ниже нормы. Так, при снижении напряжения питания ниже нормы, процессор будет сброшен. Режим контроля напряжения питания включается установкой бита BOREN в регистре конфигурации FPOR. Бит BOREN установлен в состояние On*,* что разрешает контроль напряжения питания.

* + - * 1. Реализация главного потока управления

Основные принципы организации работы данного потока управления описаны в 4.2.1в)

ГП управления реализован в виде основной функции программы ***main()***, которая получает управление сразу после сброса МК.

В функции ***main()*** выполняются следующие действия:

1. с помощью функции ***checkCauseReset\_run()*** определяются причины сброса МК;
2. с помощью функции ***ModeProtection\_ClearPS()*** выполняется процедура снятия ЗО при установленной перемычке снятия ЗС и наличии записи с кодом ЗО во внешней EEPROM. После снятия ЗО функция не передает управление, а переходит в «пустой» бесконечной цикл и находится в нем до следующего запуска прибора. При отсутствии перемычки снятия ЗС управление переходит к следующей за ней функции ***IdentHex\_run()***;
3. функция ***IdentHex\_run()*** выполняет запись в EEPROM следующих значений:

* версии ПО;
* даты сборки ПО;
* идентификатор типа прибора.

1. Инициализируются все компоненты ПО, за исключением компонента RS422, ***Eeprom\_ctor()***, ***ModeProtection\_ctor()***, ***BinIn\_ctor()***, ***CheckSupply\_ctor()***, ***DeviceAddress\_ctor()***, ***SafetyPowerControl\_ctor()***, ***LedFailure\_ctor()***, ***AnalogMeasurement\_ctor()***, ***Sheduler\_ctor()*** и ***ControlMK\_ctor()***;
2. макрос ENABLE\_WATCHDOG конфигурирует и разрешает работу сторожевого таймера;
3. функция ***TimeBeginSynchronization\_run()*** обеспечивает начальную синхронизацию процессоров;
4. макрос CAN\_TIMER\_INITразрешает работу таймера для работы с модулем CAN, макрос MAIN\_TIMER\_INIT\_AND\_START разрешает работу таймера управления главным потоком и макрос INTERRUPT\_INIT инициализируют прерывания потока временной синхронизации;
5. после всех процедур инициализации и настройки процессора следует бесконечный цикл ***while(true)***. В нем происходят следующие действия:
6. обнуляется счетчик прерываний синхронизации *cInterrMainFor*;
7. сбрасывается WDT;
8. вызывается на выполнение основная рабочая функция диспетчера режимов ***Sheduler\_run()***, обеспечивающая работу ДКА (компонент FsmStates);
9. вызывается функция ***cTimeWork()***, выполняющая подсчет времени работы программы, после чего происходит обнуление счетчика прерываний синхронизации;
10. вызывается функция ***waitAndCheckInterrupt()*** с параметром MAIN\_NUMBER\_OF\_INTERRUPT. Если при входе в функцию счетчик *cInterrMainFor* превышает MAIN\_NUMBER\_OF\_INTERRUPT, происходит переход в ЗС, т.к. время выполнения данной итерации основного цикла длится дольше MAIN\_NUMBER\_OF\_INTERRUPT прерываний, т.е. 1 мс, что свидетельствует о сбое в алгоритме работы программы. Если все действия данного миллисекундного цикла завершились менее чем за 1 мс, происходит ожидание окончания кванта главного потока (пока счетчик *cInterrMainFor* не станет равным MAIN\_NUMBER\_OF\_INTERRUPT).
    * + - 1. Реализация потока временной синхронизации

Основные принципы организации работы данного потока управления описаны в 4.2.2 документа [1].

Для организации работы ПВС задействованы следующие ресурсы МК:

1. один 16-битный таймер T5 – системный таймер, который отсчитывает периоды (кванты) синхронизации, равные 62,5 мкс;
2. порт PORTЕ14, к которому подключена входная линия временной синхронизации – синхросигнал от соседнего МК (вход);
3. порт PORTЕ15, к которому подключена выходная линия временной синхронизации – синхросигнал к соседнему МК (выход).

Инициализация таймера Т5 выполняется макросом MAIN\_TIMER\_INIT\_AND\_START и состоит в следующем:

1. установкой бита TON=0 в регистре T5CON таймер 5 выключается;
2. инициализируются поля регистра T4CON, которые задают режим работы таймера.

В функции ***TimeBeginSynchronization\_run()*** инициализируются порты синхронизации PORTЕ14 и PORTЕ15 при помощи макроса TIME\_SYNCHRO\_INIC\_PORTS.

Инициализация прерывания по изменению уровня состоит в следующем:

1. в регистре CNENE устанавливается бит CNIEE15 = 1 – разрешение прерывания по изменению уровня на входе PORTE15;
2. устанавливается уровень приоритета прерываний по изменению уровня на входах CN, равный 4 (регистр IPC4*,* поле CNIP=4);

Системный таймер Т5 отсчитывает кванты синхронизации, по достижении регистра TMR5 значения регистра PR5 возникает прерывание, которое обрабатывается с помощью макроса MAIN\_TIMER\_FUNCTION\_INTERRUPT, в которой последовательно выполняются следующие действия:

1. макрос CHECK\_INTERRUPT\_CN проверяется наличие с момента предыдущего вызова функции MAIN\_TIMER\_FUNCTION\_INTERRUPT прерывания по изменению уровня синхросигнала (в регистре IEC1 бит CNIE равен «0»), в случае отсутствия происходит переход в ЗС с кодом 0xA001;
2. меняет на противоположный уровень сигнала на выходной линии временной синхронизации с помощью макроса TIME\_SYNCHRO\_TOGGLE\_LEVEL;
3. изменяет состояние бита синхронизации на противоположное TIME\_SYNCHRO\_TOGGLE\_LEVEL;
4. разрешает прерывание по изменению уровня синхросигнала (в регистре IEC1 бит CNIE = 1).

Обработка прерывания синхронизации (по изменению уровня синхросигнала) выполняется с помощью макроса CN\_FUNCTION\_INTERRUPT, которая вызывается при соблюдении двух условий:

1. данное прерывание разрешено (в регистре IEC1 бит CNIE равен 1);
2. есть изменение уровня синхросигнала (PORTE14).

Считается что макросы CN\_FUNCTION\_INTERRUPT выполняются синхронно в обоих МК (с точностью до нескольких машинных циклов).

В макросе CN\_FUNCTION\_INTERRUPT последовательно выполняются следующие действия:

1. макрос MAIN\_TMR\_SYNCHRO – синхронизация системных таймеров T5 обоих МК c коррекцией на время вызова функции прерывания путем записи в регистры TMR5 определенного значения (TMR5 = 90). В процессе работы таймеры T5 работают непрерывно (программно не останавливаются и не запускаются). Если по каким-либо причинам они в разных МК «разбежались», то данное действие заставляет их начать счет с одинакового значения и далее работать синхронно;
2. запрет прерываний по изменению уровня;
3. контроль уровня сигнала временной синхронизации. В прерываниях с четными номерами (0, 2, …) уровень синхросигнала должен быть высоким (1), в прерываниях с нечетными номерами – низким (0). Если уровень синхросигнала не соответствует ожидаемому, происходит переход в ЗС (имеет место рассинхронизация);
4. вызываются функции компонентов, выполняющие действия, требующие жесткой синхронизации по времени работы в обоих МК:
5. прерывание 0: ***BinIn\_interruptAddrSet()*** – установка выходных сигналов для функции ***BinIn()***;
6. прерывание 1: ***AnalogMeasurement\_runInterrupt()*** – чтение из АЦП результатов текущего цикла преобразования и запуск нового цикла. Таким образом, частота дискретизации аналоговых сигналов составляет 1 кГц.
7. прерывание 3: контроль МК;
8. прерывание 15: **BinIn\_interruptAddrRead()** – чтение входных сигналов для функции ***BinIn()***;
9. функции ***ShuntShiftGen\_interrupt()***, ***PosDetGenerator\_interrupt()***, ***BlockExch\_interrupt()*** вызываются при каждом прерывании.
10. проверка на переполнение счетчика прерываний синхронизации *cInterrMainFor*. Если *cInterrMainFor* больше либо равен MAIN\_NUMBER\_OF\_INTERRUPT, происходит переход в ЗС. Счетчик *cInterrMainFor* обнуляется в начале каждой итерации основного рабочего цикла ***main()***, поэтому данная ситуация возможна если время выполнения итерации основного цикла длится дольше MAIN\_NUMBER\_OF\_INTERRUPT прерываний (1 мс) что свидетельствует о сбое в алгоритме работы программы;
11. инкремент счетчика прерываний синхронизации;
12. запуск таймера для при помощи макроса CAN\_TIMER\_START в каждом прерывании кроме последнего в цикле из MAIN\_NUMBER\_OF\_INTERRUPT прерываний.
    * + - 1. Реализация потока синхронизации данных

Основные принципы организации работы данного потока управления описаны в 4.2.3 документа [1]. Подробное описание компонента InterChannel приведено в 4.3.14.

Для реализации потока синхронизации данных используется таймер МК T4.

Инициализация таймера Т4 выполняется в макросе CAN\_TIMER\_INIT и состоит в следующем:

1. Т4 является таймером типа B, в его управляющем регистре T4CON выполняются следующие установки:
2. TON=0 – таймер остановлен.
3. TCS=0 – источником тактового сигнала является внутренняя системная тактовая частота;
4. TGATE = 0 – таймерное накопление времени отключено;
5. TCKPS<1:0> = 00 – значение входного делителя тактовой частоты таймера равно 1:1, т.е. входная тактовая частота не делится, и значение счетчика таймера (регистр TMR4) увеличивается на 1 за 1 командный цикл.
6. обнуляется содержимое регистра счетчика таймера: TMR4 = 0;
7. путем записи в регистр PR4 (регистр периода таймера) значения 128, устанавливается период счета;
8. в макросе INTERRUPT\_INIT устанавливается уровень приоритета прерываний от таймера Т4, равный 2: в регистре IPC6 поле T4IP2=2;
9. с помощью макроса CAN\_TIMER\_ENABLE\_INTERRUPT разрешается прерывание от T4;

В конце каждого прерывания временной синхронизации, кроме последнего из цикла MAIN\_NUMBER\_OF\_INTERRUPT прерываний (см. 4.1.1.3.3), запускается таймер T4. По истечении 7 машинных циклов, соответствующих примерно 237,3 нс, возникает прерывание от Т4 и управление передается макросу его обработки CAN\_TIMER\_FUNCTION\_INTERRUPT.

В макросе CAN\_TIMER\_FUNCTION\_INTERRUPT выполняются следующие действия:

* с помощью макроса CAN\_TIMER\_STOP останавливается счет таймера;
* запускается функция ***InterChannel\_runCommunication()*** компонента МКО InterChannel, реализующая работу с соседним каналом через физическую линию связи, которой в качестве параметра передается указатель на константную структуру *mainCanDrvFunc* из указателей на интерфейсные функции драйвера dsPIC33CanAck**.** Функция ***InterChannel\_runCommunication()*** определяет события готовности данных к чтению или передаче, выполняет чтение данных из драйвера и инициирует передачу;
* с помощью макроса CAN\_TIMER\_INTERRUPT\_CLEAR\_FLAG сбрасывается флаг запроса прерывания от T4.
  + - * 1. Реализация потока обработки неиспользуемых прерываний

Данный поток реализован в виде функций обработки неиспользуемых прерываний, которые определены в модуле UnusedInterrupts*.* При возникновении неиспользуемого прерывания прибор переходит в 3С.

* + - 1. Модуль ControlSystem
         1. Назначение

Модуль ControlSystem предназначен для реализации функций верхнего уровня управления логикой работы прибора.

* + - * 1. Описание

Инициализация модуля производится с помощью функции ***ControlSystem\_ctor()***, вызываемой при переходе ДКА в состояние *StateMain*.

Основная рабочая функция модуля ***ControlSystem\_run()*** вызывается в режиме работы (состоянии) *StateMain* в главном потоке управления (основном цикле *main*) с периодом 1 мс. Вызывает на выполнение следующие функции:

1. ***ControlSystem\_exchangeRun()*** – управление обменом данными с УС по Rs422;
2. ***ControlSystem\_activityManagerRun()*** – управление активностью;
3. ***ControlSystem\_indicationRun()*** – управление индикацией на передней панели прибора;
4. ***ControlSystem\_shuntShiftPosDetRun()*** – определение положения стрелки;
5. ***ControlSystem\_processingOrders()*** – обработка поступающих от УС приказов.
   * + 1. Модуль UnusedInterrupts
          1. Назначение

В модуле UnusedInterrupts определены обработчики неиспользуемых в штатной работе ПО прерываний.

* + - * 1. Интерфейсы

Не имеет собственных интерфейсов.

* + - * 1. Описание

В модуле UnusedInterrupts определены подпрограммы обработки (обработчики) всех неиспользуемых в штатной работе прерываний, включая немаскируемые прерывания – «ловушки» (traps), возникающие вследствие ошибочных программных операций или аппаратных сбоев МК. Возникновение любого такого прерывания считается критическим сбоем, исключающим дальнейшее продолжение работы прибора.

Обработчики прерываний основной таблицы векторов выполняют следующие действия:

* в «черный ящик» с помощью функции ***BlackBox\_saveAddCodeProtectionState()*** записывается уникальный для каждого прерывания код – номер вектора в таблице (параметр, уточняющий причину перехода в ЗС);
* переход в ЗС с помощью макроса ERROR\_ID(eGrPS\_Main, ePS\_MainUnusedInterrupt).

Обработчики прерываний альтернативной таблицы векторов выполняют следующие действия:

* в «черный ящик» с помощью функции ***BlackBox\_saveAddCodeProtectionState()*** записывается уникальный для каждого прерывания код – номер вектора в таблице (параметр, уточняющий причину перехода в ЗС).
  + - 1. Модуль InterruptsHandlers
         1. Назначение
         2. Описание
  1. Уровень режимов
     1. Компонент Fsm
        1. Назначение

Компонент Fsm реализует иерархический ДКА, на базе которого реализован диспетчер режимов FsmStates.

* + - 1. Состав компонента

Компонент состоит из одноимённого модуля.

* + - 1. Описание
         1. Общие сведения

Основой автомата состояний являются структура данных, которая описывает состояние автомата и содержит функции, выполняющиеся в данном состоянии, а также ссылки на другие состояния вниз и вверх по иерархии. Состояния могут быть следующих типов:

* корневое состояние;
* родительское состояние;
* дочернее состояние;
* дочернее состояние по умолчанию.

Каждому состоянию должна соответствовать своя структура *FsmState*, которая при создании проинициализирована соответствующими функциями и ссылками на другие состояния (корневое, родительское и дочернее по умолчанию), связанные с данным состоянием по иерархии. Обработка состояний выполняется сверху вниз от корневого состояния к дочернему. Обрабатываются все состояния до тех пор, пока не встретится состояние, которое не имеет дочерних или пока выполнившаяся функция не сообщит автомату, что выполнять обработку дочерних состояний не нужно. Такая необходимость возникает если принято решение о переходе в другое состояние.

Для каждого уровня иерархии выполняется одно состояние – текущее. Переход от состояния к состоянию может осуществляться на разных уровнях иерархии. При каждом переходе выстраивается новый маршрут прохождения автомата от корневого состояния к последнему дочернему. При построении маршрута всем дочерним, по отношению к новому, состояниям устанавливаются дочерние состояния по умолчанию. Для родительских состояний выстраивается связь с дочерними состояниями относительно нового. При построении связей изменяются ссылки на дочерние состояния. Ссылки на корневые и родительские состояния остаются неизменными.

Для управления структурами данных ДКА используются интерфейсные функции ***Fsm\_start()***, ***Fsm\_transit()***, ***Fsm\_run()*** и локальные функции ***stateExit()***, ***stateEntry()***.

* + - * 1. Структура состояния ДКА

Для каждого состояния могут быть определены следующие функции:

* обработчик текущего состояния автомата ***–*** ***onRun()***;
* функция, которая вызывается при входе в состояние – ***onEntry()***;
* функция, которая вызывается при выходе из состояния ***–*** ***onExit()***.

Состояние может не иметь этих функций, тогда указатели на них должны быть проинициализированы нулем.

Для каждого состояния могут быть определены ссылки на следующие состояния:

* ссылка на корневое состояние автомата – *top*;
* ссылка на родительское состояние *– super*;
* ссылка на текущее дочернее состояние – *fsm*;
* ссылка на дочернее состояние, которое устанавливается по умолчанию при переходе в текущее состояние, – *initial*.

Состояние может иметь не все вышеперечисленные ссылки. Те ссылки, которых состояние не имеет, должны быть проинициализированы нулем.

Обработчик текущего состояния автомата ***onRun()*** вызывается для корневого состояния и всех текущих дочерних состояний вниз по иерархии при вызове функции ***Fsm\_run()***. Родительская функция ***onRun()*** может запретить вызов функций дочерних состояний. Чтобы функции ***onRun()*** дочерних состояний были вызваны, функция ***onRun()*** родительского состояния должна вернуть *true*. Если она вернет *false*, функции дочерних состояний вызваны не будут.

Функция ***onEntry()*** предназначена для выполнения действий по инициализации работы данного состояния. Вызывается один раз при входе в состояние.

Функция ***onExit()*** предназначена для выполнения действий по завершению работы данного состояния. Вызывается один раз при выходе из состояния.

* + - * 1. Функции состояния ДКА

Для создания автомата необходимо для каждого состояния объявить структуру *FsmState*, при необходимости реализовать функции ***onRun()***, ***onEntry()***, ***onExit()*** и проинициализировать структуру, задав указатели на реализованные функции и корневое, родительское и дочернее состояние по умолчанию.

Для инициализации автомата необходимо вызвать функцию ***Fsm\_start()***, которой в качестве параметра передать структуру, описывающую корневое состояние. При этом будет построен маршрут от корневого состояния к дочернему, используя дочерние состояния по умолчанию. Работа автомата выполняется при вызове функции ***Fsm\_run()*** с корневым состоянием в качестве параметра. При этом будут вызваны функции ***onRun()*** всех состояний из установленного маршрута, если какая-то из функций не вернет *false*.

Для смены маршрута работы автомата необходимо вызвать функцию ***Fsm\_transit()***, которой в качестве параметра передать структуру, описывающую новое состояние. Функция перестроит маршрут ДКА от корня до последнего дочернего состояния так чтобы он проходил через новое состояние. При перестройке маршрута для всех состояний, которые больше не будут участвовать в работе автомата, будут вызваны функции ***onExit()***. Для всех состояний, которые вступят в работу ДКА, будут вызваны функции ***onEntry()***.

При построении маршрута ДКА функция ***Fsm\_transit()*** использует функции ***stateExit()*** и ***stateEntry()***. Функция ***stateExit()*** проходит по всем дочерним состояниям текущего состояния, вызывает для них функцию ***onExit()*** и, в качестве текущего дочернего состояния, устанавливает ноль, разрушая прежний маршрут вплоть до родительского состояния нового состояния. Функция ***stateEntry()*** строит новый маршрут от нового родительского состояния, вызывая при этом функции ***onEntry()***. Дальше от нового состояния вниз по иерархии строится маршрут из дочерних состояний по умолчанию при помощи функции ***Fsm\_start()***, которой в качестве параметра передается структура, описывающая новое состояние. При этом для каждого состояния, участвующего в маршруте ДКА, вызывается функция ***onEntry()***.

* + 1. Компонент FsmStates
       1. Назначение

Диспетчер режимов на основе иерархического ДКА обеспечивает работу ПО в определенном режиме. Каждый режим определяет состояние прибора в текущий момент и обеспечивает выполнение функций, принадлежащих только этому режиму.

* + - 1. Состав компонента

Для управления режимами используются следующие компоненты:

* *Sheduler* − планировщик заданий;
* *States* − объявления состояний ДКА;
* *StateActive* – активное состояние;
* *StateInitial* – состояние инициализации;
* *StateMain* – состояние основной работы;
* *StatePassive* – пассивное состояние;
* *StateSafeActive* − безопасное активное состояние;
* *StateSafePassive* − безопасное пассивное состояние;
* *StateTop* − корневое состояние ДКА;
* *StateWorking* − рабочее состояние.
  + - 1. Описание
         1. Общие сведения

Компонент построен на базе иерархического ДКА компонента Fsm. В состав компонента входит главный модуль Sheduler, модули состояний и интерфейсный модуль States (один для всех состояний). Главный модуль Sheduler является интерфейсом всего компонента. Он содержит реализацию функций ***Sheduler\_ctor()*** и ***Sheduler\_run()***, которые используются пользовательским ПО для инициализации и обеспечения работы диспетчера режимов.

Модули состояний реализуют структуру *FsmState* и функции ***onRun()***, ***onEntry()***, ***onExit()*** для каждого состояния, а именно:

* *StateActive* – активное состояние;
* *StateInitial* – состояние инициализации;
* *StateMain* – состояние основной работы;
* *StatePassive* – пассивное состояние;
* *StateSafeActive* − безопасное активное состояние;
* *StateSafePassive* − безопасное пассивное состояние;
* *StateTop* − корневое состояние ДКА.

Диаграмма состояний режимов работы ПО представлена на рисунке 2.

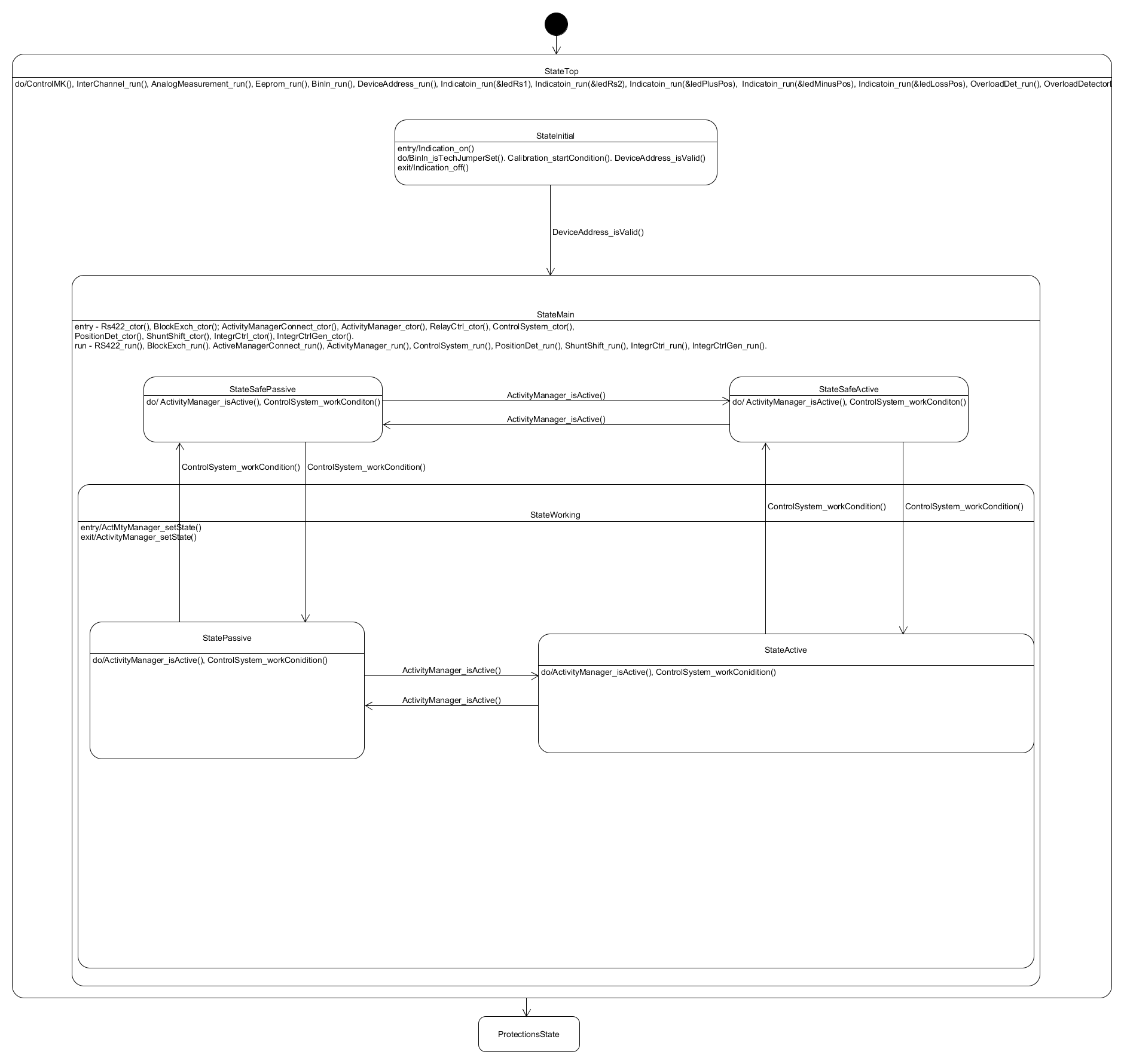


Рисунок 2 – Диаграмма состояний режимов работы ПО

Поскольку каждое состояние должно содержать ссылки на состояния, стоящие выше и ниже по иерархии, структуры всех модулей описаны в одном интерфейсном файле States.h.

Для использования компонента FsmStates необходимо использовать файл Sheduler.h, вызвать функцию ***Sheduler\_ctor()*** и в главном потоке с периодом его работы вызвать функцию ***Sheduler\_run()***.

* + - 1. Модуль Sheduler
         1. Назначение

Реализация диспетчера режимов на основе иерархического ДКА. Главный модуль компонента.

* + - * 1. Описание

Модуль содержит функции ***Sheduler\_ctor()*** и ***Sheduler\_run()***. Функция ***Sheduler\_ctor()*** инициализирует работу ДКА, при помощи функции ***Fsm\_start()*** и корневого состояния *StateTop*, строит маршрут по умолчанию. Функция ***Sheduler\_run()*** обеспечивает работу ДКА. Также обеспечивает одиночный проход по маршруту, построенному автоматом при помощи функции ***Fsm\_run()*** и корневого состояния *StateTop*.

* + - 1. Модуль States
         1. Назначение

Интерфейсный модуль, обеспечивающий доступ ко всем структурам, которые описывают состояние ДКА.

* + - * 1. Интерфейсы

Модуль содержит объявление следующих структур данных:

* *StateTop* – корневое состояние ДКА;
* *StateInitial* – cостояние инициализации;
* *StateMain* – состояние основной работы;
* *StateSafePassive* – безопасное пассивное состояние;
* *StateSafeActive* – безопасное активное состояние;
* *StateWorking* – рабочее состояние;
* *StatePassive* – пассивное состояние;
* *StateActive* – активное состояние.
  + - 1. Модуль StateTop
         1. Назначение

Корневое состояние ДКА.

* + - * 1. Описание

Структура *StateTop* инициализируется следующими значениями:

* обработчик текущего состояния автомата *onRun* – ***StateTop\_onRun()***;
* функция, которая вызывается при входе в состояние *onEntry* – 0;
* функция, которая вызывается при выходе из состояния *onExit* – 0;
* ссылка на корневое состояние автомата *top* – *StateTop*;
* ссылка на родительское состояние *super* – 0;
* ссылка на дочернее состояние *initial* – *StateInitial*.

Функция ***StateTop\_onRun()*** вызывает функции тех компонентов, которые должны вызываться во всех режимах. Метод вызывает следующие функции:

* ***ControlMK\_run()***;
* ***InterChannel\_run()***;
* ***AnalogMeasurement\_run()***;
* ***Eeprom\_run()***;
* ***BinIn\_run()***;
* ***DeviceAddress\_run()***;
* ***Indication\_run(&ledRs1);***
* ***Indication\_run(&ledRs2);***
* ***Indication\_run(&ledPlusPos);***
* ***Indication\_run(&ledMinusPos);***
* ***Indication\_run(&ledLossPos);***
* ***CheckSupply\_run();***
* ***OverloadDet\_run();***

Функция всегда возвращает *true*.

* + - 1. Модуль StateInitial
         1. Назначение

Состояние инициализации.

* + - * 1. Описание

Структура *StateTop* инициализируется следующими значениями:

* обработчик текущего состояния автомата *onRun* – ***StateInitial\_onRun()***;
* функция, которая вызывается при входе в состояние *onEntry* – ***StateInitial\_onEntry()***;
* функция, которая вызывается при выходе из состояния *onExit* – ***StateInitial\_onExit()***;
* ссылка на корневое состояние автомата *top* – *StateTop*;
* ссылка на родительское состояние *super* – *StateTop*;
* ссылка на дочернее состояние *initial* – 0.

Функция ***StateInitial\_onEntry()*** зажигает на лицевой панели светодиодные индикаторы «RS1», «RS2», «ПОЛОЖЕНИЕ+», «ПОЛОЖЕНИЕ-», «Потеря контроля».

Функция ***StateInitial\_onExit()*** гасит на лицевой панели светодиодные индикаторы «RS1», «RS2», «ПОЛОЖЕНИЕ+», «ПОЛОЖЕНИЕ-», «Потеря контроля».

* + - 1. Модуль StateMain
         1. Назначение

Состояние основной работы.

* + - * 1. Описание

Структура *StateMain* инициализируется следующими значениями:

* обработчик текущего состояния автомата *onRun* – ***StateMain\_onRun()***;
* функция, которая вызывается при входе в состояние *onEntry* – ***StateMain\_onEntry()***;
* функция, которая вызывается при выходе из состояния *onExit* – 0;
* ссылка на корневое состояние автомата *top* – *StateTop*;
* ссылка на родительское состояние *super* – *StateTop*;
* ссылка на дочернее состояние *initial* – *StateSafePassive*.

Функция ***StateMain\_onEntry()*** вызывает конструкторы и разрешает работу следующих компонентов:

* ***Rs422\_ctor()***;
* ***BlockExch\_ctor()***;
* ***ActivityManagerConnect\_ctor()***;
* ***ActivityManager\_ctor()***;
* ***RelayCtrl();***
* ***ControlSystem\_ctor()***;
* ***PositionDet\_ ctor ()***;
* ***ShuntShift\_ ctor ();***
* ***IntegrCtrl\_ ctor ();***
* ***IntegrCtrlGen\_ ctor ().***

Функция ***StateMain\_onRun()*** вызывает рабочие функции следующих компонентов:

* ***Rs422\_run()***;
* ***BlockExch\_run()***;
* ***ActivityManagerConnect\_run()***;
* ***ActivityManager\_ run ()***;
* ***RelayCtrl();***
* ***ControlSystem\_ctor()***;
* ***PositionDet\_run()***;
* ***ShuntShift\_run();***
* ***IntegrCtrl\_run();***
* ***IntegrCtrlGen\_run().***

Функция всегда возвращает значение *true*.

* + - 1. Модуль StateSafePassive
         1. Назначение

Безопасное пассивное состояние.

* + - * 1. Описание

Структура *StateSafePassive* инициализируется следующими значениями:

* обработчик текущего состояния автомата *onRun* – ***StateSafePassive\_onRun()***;
* функция, которая вызывается при входе в состояние *onEntry* – 0;
* функция, которая вызывается при выходе из состояния *onExit* – 0;
* ссылка на корневое состояние автомата *top* – *StateTop*;
* ссылка на родительское состояние *super* – *StateMain*;
* ссылка на дочернее состояние *initial* – 0.

Функция ***StateSafePassive\_onRun()*** определяет условия перехода в активное состояние при помощи функции ***ActivityManager\_isActive()*** и, в случае выполнения этих условий, осуществляет переход в безопасное активное состояние *StateSafeActive*. Одновременно при помощи функции ***ControlSystem\_workCondition()*** выполняется проверка условий перехода в рабочий режим и, если эти условия выполняются, происходит переход в пассивное состояние рабочего режима *StatePassive*. Функция возвращает *true*, кроме случаев перехода в другие состояния.

* + - 1. Модуль StateSafeActive
         1. Назначение

Безопасное активное состояние.

* + - * 1. Описание

Структура *StateSafeActive* инициализируется следующими значениями:

* обработчик текущего состояния автомата *onRun* – ***StateSafeActive\_onRun()***;
* функция, которая вызывается при входе в состояние *onEntry* – 0;
* функция, которая вызывается при выходе из состояния *onExit* – 0;
* ссылка на корневое состояние автомата *top* – *StateTop*;
* ссылка на родительское состояние *super* – *StateMain*;
* ссылка на дочернее состояние *initial* – 0.

Функция ***StateSafeActive\_onRun()*** определяет условия перехода в пассивное состояние при помощи функции ***ActivityManager\_isActive()*** и, в случае выполнения этих условий, осуществляет переход в безопасное пассивное состояние *StateSafePassive*. Одновременно при помощи функции ***ControlSystem\_workCondition()*** выполняется проверка условий перехода в рабочий режим и, если эти условия выполняются, происходит переход в активное состояние рабочего режима *StateActive*. Функция возвращает *true*, кроме случаев перехода в другие состояния.

* + - 1. Модуль StateWorking
         1. Назначение

Рабочее состояние.

* + - * 1. Интерфейсы

Структуры данных

*StateWorking* – структура, описывающая рабочее состояние.

* + - * 1. Описание

Структура *StateWorking* инициализируется следующими значениями:

* обработчик текущего состояния автомата *onRun* – ***StateWorking\_onRun()***;
* функция, которая вызывается при входе в состояние *onEntry* – ***StateWorking\_onEntry()***;
* функция, которая вызывается при выходе из состояния *onExit* – ***StateWorking\_onExit()***;
* ссылка на корневое состояние автомата *top* – *StateTop*;
* ссылка на родительское состояние *super* – *StateMain*;
* ссылка на дочернее состояние *initial* – *StatePassive*.

Функция ***StateWorking\_onEntry()*** устанавливает ActivityManager в рабочее состояние при помощи функции ***ActivityManager\_setState()***.

Функция ***StateWorking\_onExit()*** устанавливает ActivityManager в безопасное состояние при помощи функции ***ActivityManager\_setState().***

Функция ***StateWorking\_onRun()*** ничего не делает и всегда возвращает *true*. Она необходима для вызова дочерних состояний. Если она будет отсутствовать, дочерние состояния вызываться не будут.

* + - 1. Модуль StatePassive
         1. Назначение

Пассивное состояние.

* + - * 1. Описание

Структура *StatePassive* инициализируется следующими значениями:

* обработчик текущего состояния автомата *onRun* – ***StatePassive\_onRun()***;
* функция, которая вызывается при входе в состояние *onEntry* – ***StatePassive\_onEntry()***;
* функция, которая вызывается при выходе из состояния *onExit* – ***StatePassive\_onExit()***;
* ссылка на корневое состояние автомата *top* – *StateTop*;
* ссылка на родительское состояние *super* – *StateWorking*;
* ссылка на дочернее состояние *initial* – *StatePassiveNormal*.

Функции ***StatePassive\_onEntry()*** и ***StatePassive\_onExit()*** ничего не делают.

Функция ***StatePassive\_onRun()*** определяет условия перехода в активное состояние при помощи функции ***ActivityManager\_isActive()*** и, в случае выполнения этих условий, осуществляет переход в активное состояние рабочего режима *StateActive*. Одновременно при помощи функции ***ControlSystem\_workCondition()*** выполняется проверка условий перехода в рабочий режим и, если эти условия не выполняются, происходит переход в безопасное пассивное состояние *StateSafePassive*. Функция возвращает *true*, кроме случаев перехода в другие состояния.

* + - 1. Модуль StateActive
         1. Назначение

Активное состояние.

* + - * 1. Описание

Структура *StateActive* инициализируется следующими значениями:

* обработчик текущего состояния автомата *onRun* – ***StateActive\_onRun()***;
* ссылка на корневое состояние автомата *top* – *StateTop*;
* ссылка на родительское состояние *super* – *StateWorking*;
* ссылка на дочернее состояние *initial* – *StateActiveNormal*.

Функция **StateActive\_onRun()** определяет условия перехода в пассивное состояние при помощи функции **ActivityManager\_isActive()**и, в случае выполнения этих условий, осуществляет переход в пассивное состояние рабочего режима *StatePassive*. Одновременно при помощи функции ***ControlSystem\_workCondition()*** выполняется проверка условий перехода в рабочий режим и, если эти условия не выполняются, происходит переход в безопасное активное состояние *StateSafeActive*. Функция возвращает *true*, кроме случаев перехода в другие состояния.

* 1. Уровень компонент
     1. Компонент ActivityManager
        1. Назначение

Компонент ActivityManager предназначен для управления активностью прибора (в зависимости от сложившихся условий осуществляет переход прибора из пассивного в активное состояние и наоборот) и определяет интерфейсы для взаимодействия с другими компонентами ПО.

* + - 1. Состав компонента

В состав компонента входят следующие модули:

* *ActivityManager* − управление активностью прибора. Основной модуль компонента;
* *ActivityManager\_connect* − обеспечение взаимодействия модулей ActivityManager и BlockExch;
* *ActivityManager\_internal* − реализация внутренних (не интерфейсных) функций компонента;
* *BlockExch* − обмен между основным и резервным приборами.
  + - 1. Описание
         1. Общие сведения

Управление активностью осуществляется при работе ОКПС-Е-К в режиме ненагруженного резервирования, при этом один из пары смежных приборов является основным, а второй – резервным. При работе без резервирования каждый прибор считается основным и всегда активным. Управление РПВ осуществляется только основным прибором. Компонент работает в режимах «Работа/РС» и «БС». Активный прибор формирует на выходе сигналы в соответствии с состоянием перемычек и приказами, полученными от УС по интерфейсу RS-422, отвечает полным статусом на приказы от УС. Пассивный прибор передает короткий статус (длина поля данных 1 байт).

* + - * 1. Данные для работы модуля

Входные данные

Входные данные для работы модуля ActivityManager:

1. тип прибора:
2. основной;
3. резервный.
4. состояние прибора:
5. РС;
6. БС.
7. состояние связи прибора по RS-422:
8. есть;
9. нет.
10. состояние контактов реле:
11. «на мне»;
12. «не на мне»
13. состояние активной команды АК1:
14. действует (ВКЛ);
15. не действует (ВЫКЛ).
16. состояние связи с соседом:
17. есть пакеты данных от соседнего прибора, соседний прибор принимает данные от меня (S3=1, см. **Ошибка! Источник ссылки не найден.**);
18. есть пакеты данных от соседнего прибора, соседний прибор не принимает данные от меня (S3=0, см. **Ошибка! Источник ссылки не найден.**);
19. нет пакетов от соседнего прибора.
20. состояние соседнего прибора:
21. РС;
22. БС;
23. состояние активности соседнего прибора:
24. прибор активен;
25. прибор пассивен;
26. состояние связи соседнего прибора с УС по интерфейсу RS-422:
27. есть;
28. нет;
29. состояние выдачи управляющих воздействий:
30. ВКЛ;
31. ВЫКЛ.
32. Состояние поля ACT приказа от управляющей системы:
33. установить активным основной прибор;
34. установить активным резервный прибор.

Выходные данные

Выходные данные работы модуля ActivityManager:

1. состояние активности прибора:
2. прибор активный;
3. прибор пассивный;
4. состояние команды управления РПВ:
5. команда выдана;
6. команда не выдана;
7. состояние запроса на выключение управляющих воздействий:
8. ВКЛ;
9. ВЫКЛ.

Временны́е параметры

Временные параметры работы модуля ActivityManager:

1. Т1 – время после сброса, в течение которого основной прибор остаётся пассивным, составляющее 3с;
2. Т2 – время после сброса, в течение которого резервный прибор остается пассивным (Т2 > Т1), составляющее 5с;
3. Т3 – время после выдачи прибором команды на включение РПВ, после которого можно считывать сигнал(ы) состояния реле, составляющее 120мс;
4. Т4 – время после выдачи прибором команды на выключение РПВ, после которого можно считывать сигнал(ы) состояния реле, составляющее 110мс;
5. Т5 – время после перехода прибора из активного в пассивное состояние, в течение которого он остается пассивным независимо от состояния соседнего прибора, составляющее 100мс;
   * + - 1. Алгоритмы работы

Модуль работает в соответствии с алгоритмами, приведенными на рисунках .



Рисунок – Алгоритмы управления активностью

Рисунок 3(продолжение) – Алгоритмы управления активностью



Рисунок 4 – Алгоритмы управления активностью (вспомогательные алгоритмы (часть 1))



Рисунок 5 – Алгоритмы управления активностью (вспомогательные алгоритмы (часть 2))

* + - * 1. Особенности реализации модуля ActivityManager

При переходе ПО из состояния инициализации *StateInitial* в состояние основной работы *StateMain* модуль инициализируется и включается в работу (вызов функций ***ActivityManager\_ctor()*** и ***ActivityManager\_setEnable()***с параметром *true)*, получая в качестве параметра определенный в процессе инициализации статус прибора (основной/резервный).

Основная рабочая функция модуля ***ActivityManager\_run()*** вызывается в режиме работы (состоянии) *StateMain* в главном потоке управления (основном цикле *main*) с периодом 1 мс.

Функция выполняет следующие действия:

1. обработка таймаутов;
2. реализация КА, работающего в соответствии с алгоритмами (см. рисунки **Ошибка! Источник ссылки не найден.**-6).

КА имеет следующие состояния:

* *eRunStart* – начальное состояние после сброса и инициализации;
* *eRunPassiveState* – пассивное состояние;
* *eRunTransitionToActive* – переход в активное состояние;
* *eRunActiveState* – активное состояние;
* *eRunTransitionToPassive* – переход в пассивное состояние.

В состояние *eRunStart* автомат попадает после сброса МК (после инициализации и включения модуля) и находится в нем до истечения времени *T\_O\_START\_MASTER* (для МК Master) или *T\_O\_START\_STANDBY* (для МК Slave), оставаясь пассивным, после чего переходит в состояние *eRunPassiveState*.

Состояние *eRunPassiveState* обрабатывается с помощью функции ***passiveProcessing()***, в которой, на основе анализа состояния своего и смежного прибора, принимается решение либо о безусловной смене активности (автомат переходит в состояние *eRunActiveState*), либо о смене активности в случае успешного выполнения определенных действий (автомат переходит в состояние *eRunTransitionToActive*).

Состояние *eRunTransitionToActive* обрабатывается с помощью функции ***transitionToActive()***, в которой выполняются определенные действия по переходу прибора в активное состояние, и, в случае их успеха, автомат переходит в состояние *eRunActiveState*.

Состояние *eRunActiveState* обрабатывается с помощью функции ***passiveProcessing()***, в которой, на основе анализа состояния своего и смежного прибора, принимается решение либо о безусловной смене активности (автомат переходит в состояние *eRunPassiveState*), либо о смене активности в случае успешного выполнения определенных действий (автомат переходит в состояние *eRunTransitionToPassive*).

Состояние *eRunTransitionToPassive* обрабатывается с помощью функции ***transitionToPassive()***, в которой выполняются определенные действия по переходу прибора в пассивное состояние, и, в случае их успеха, автомат переходит в состояние *eRunPassiveState*.

* + - 1. Модуль ActivityManager\_connect
         1. Назначение

ActivityManager\_connect является вспомогательным модулем компонента ActivityManager. Он обеспечивает взаимодействие (обмен данными) между модулями ActivityManager и BlockExch.

* + - * 1. Описание

Основная рабочая функция модуля ***ActivityManagerConnect\_run()*** вызывается в режиме работы (состоянии) *StateMain* в главном потоке управления (основном цикле *main*) с периодом 1 мс. Функция вызывается после вызова***BlockExch\_run()*** и перед вызовом ***ActivityManager\_run()****.*

Функция ***ActivityManagerConnect\_run()*** выполняет следующие действия:

1. формирует структуру данных о состояния «своего» прибора для обмена с соседним прибором и передает её в модуль BlockExch;
2. получает от блока BlockExch данные о состоянии соседнего прибора и на их основе формирует (заполняет) структуры данных для работы модуля ActivityManager, используя интерфейсы модуля ActivityManager\_internal (см. **Ошибка! Источник ссылки не найден.**).
   * + 1. Модуль ActivityManager\_internal
          1. Назначение

Вспомогательный модуль компонента ActivityManager. Реализует внутренние (не интерфейсные) функции компонента ActivityManager; предоставляет интерфейсы для модуля ActivityManager\_connect.

* + - * 1. Описание

Модуль использует константы *eRs422\_line\_1* и *eRs422\_line\_2* из компонента RS‑422 (см. **Ошибка! Источник ссылки не найден.**) в качестве номеров линий связи.

Модуль ActivityManager\_internal не требует дополнительного описания, т.к. реализованные в нем функции выполняют элементарные действия.

* + - 1. Модуль BlockExch
         1. Назначение

Модуль реализует обмен данными между основным и резервным приборами.

* + - * 1. Описание

Общее описание межприборного обмена

Физический уровень МПО представляет собой две гальванически развязанные линии связи для передачи и приема данных. Сигналы с приемной линии параллельно поступают на оба МК, сигналы для передачи формируются синхронно обоими МК. Данные по каждой линии передаются только в одном направлении. Обмен данными осуществляется последовательно (бит за битом) и асинхронно. Прием и передача осуществляются программно через порты ввода/вывода общего назначения без использования специальных периферийных модулей МК. Таким образом, программно реализуется интерфейс, подобный стандартному UART. Временные параметры работы приведены в описании модуля (драйвера) BlockExch\_driver (см. **Ошибка! Источник ссылки не найден.**).

Обмен информацией между приборами происходит пакетами/кадрами (далее по тексту – сообщениями).

Описание формата сообщений межприборного обмена «Основной-Резервный»

Передаваемое сообщение состоит из следующих элементов:

* старт-посылки, состоящей из одного логического 0;
* прикладных данных;
* 8-битной контрольной суммы в виде CRC-8 для контроля целостности сообщения;
* стоп-посылки, состоящей из 8 последовательных бит, соответствующей логической 1. Стоп-посылка служит разделителем между последовательно передаваемыми сообщениями, а также для синхронизации начала работы приемной части интерфейса после сброса прибора.

При передаче сообщения первым передается байт с CRC-8, затем – байт прикладных данных. Каждый байт передается от старшего значащего бита к младшему. Циклограмма передаваемого сообщения приведена на рисунке **Ошибка! Источник ссылки не найден.**.

Прикладные данные представляют собой один байт (см. таблицу **Ошибка! Источник ссылки не найден.**), старший бит которого зарезервирован и всегда должен быть «0», младшие 7 бит являются признаками определенных состояний прибора (S1…S7).

Таблица 4 – Формат прикладных данных передаваемого сообщения

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Бит 7 | Бит 6 | Бит 5 | Бит 4 | Бит 3 | Бит 2 | Бит 1 | Бит 0 |
| S8 | S7 | S6 | S5 | S4 | S3 | S2 | S1 |

S1 – признак безопасного состояния прибора:

0x0 – безопасное состояние;

0x1 – отсутствие безопасного состояния.

S2 – состояние активности прибора:

0x0 – пассивное состояние;

0x1 – активное состояние.

S3 – состояние обмена сообщениями между смежными приборами:

0x0 – отсутствие приема сообщений от смежного прибора дольше определенного времени;

0x1 – наличие приема сообщений от смежного прибора.

S4 – состояние связи с УС по линии 1 интерфейса RS-422:

0x0 – отсутствие связи;

0x1 – наличие связи.

S5 – состояние связи с УС по линии 2 интерфейса RS-422:

0x0 – отсутствие связи;

0x1 – наличие связи.

S6 – состояние связи по линии 1 интерфейса RS-422:

0x0 – отсутствие связи;

0x1 – наличие связи.

S7 – состояние связи по линии 2 интерфейса RS-422:

0x0 – отсутствие связи;

0x1 – наличие связи.

S8 - состояние инициализации менеджера активности прибора:

0x0 – инициализация не завершена;

0x 1 – инициализация завершена.

Примечание.

**Отсутствие связи с УС** – отсутствие корректных приказов в течение не менее 1,5 с.

**Отсутствие связи по RS-422** – отсутствие корректных приказов и коротких сообщений в течение не менее 2 с.



Рисунок 6 – Циклограмма передачи сообщения МПО

Особенности реализации модуля BlockExch

По сбросу МК МПО выключен.

После инициализации МК формирует в МКО (см. 4.3.12) запрос на синхронизацию параметра с идентификатором *eICId\_BlockExch* и значением *SYNCHRO\_EXCHANGE* для синхронизации начала работы модуля в обоих МК. Таймаут успешной синхронизации (обмен между МК значением *SYNCHRO\_EXCHANGE*) равен *SYNCHRO\_EXCHANGE\_TO \* Tsync = 100 \* 62,5 мкс = 6,25 мс*. Если за это время синхронизация не произошла, прибор переходит в ЗС, в другом случае модуль BlockExch включается в штатную работу.

Основные рабочие функции модуля – ***BlockExch\_run()*** и ***BlockExch\_interrupt()***.

Функция ***BlockExch\_run()*** вызывается в режиме работы (состоянии) *StateMain* в главном потоке управления (основной цикл *main*) с периодом 1 мс и выполняет следующие действия:

1. до включения модуля в штатную работу:

проверяет установление синхронизации со смежным прибором;

1. после включения модуля в штатную работу:

изменяет на одну дискрету значение таймера отсутствия обмена со смежным прибором;

вызывает функцию ***transmitionRun()***, которая передает драйверу BlockExch\_driver (см. **Ошибка! Источник ссылки не найден.**) данные, предназначенные для передачи в смежный прибор;

вызывает функцию ***receptionRun()***, которая управляет приемом данных.

Процесс приема данных состоит из 2-х фаз (*eRecept\_begin* и *eRecept\_synchro*), последовательно сменяющих друг друга.

В фазе *eRecept\_begin* выполняются следующие действия:

1. считывание принятого по линии связи сообщения с помощью функции ***BlockExch\_get()*** (см. **Ошибка! Источник ссылки не найден.**);
2. отправка на межканальную синхронизацию (см. 4.3.12) контрольной суммы CRC-8 принятого сообщения;
3. установка таймаута синхронизации;
4. переход к фазе *eRecept\_synchro*.

Примечание.

***BlockExch\_get()*** (см. **Ошибка! Источник ссылки не найден.**) записывает в буферный массив сообщение с порядком байт, обратным принятому, т.е. в первый элемент массива записывается байт прикладных данных, а во второй – CRC-8.

В фазе *eRecept\_synchro* выполняются следующие действия:

1. ожидание окончания синхронизации CRC-8: если по истечении времени *T\_O\_DATA\_SYNCHRO* от соседнего МК не принята идентичная CRC-8 по истечении 3 попыток синхронизации – переход в ЗС, если принята – подсчет собственной CRC-8, сравнение с полученной в сообщении и, в случае совпадения, сохранение принятых данных в буфер, в случае несовпадения – переход к фазе *eRecept\_begin*.
2. загрузка/старт счетчика времени отсутствия обмена;
3. переход к фазе *eRecept\_begin*.

Примечание.

Считается, что связи с соседним прибором нет, если от него в течение *BLOCK\_EXCHANGE\_TIMEOUT* = 60 мс не было принято и синхронизировано ни одного сообщения.

Функция ***BlockExch\_interrupt()*** вызывается в прерывании синхронизации (из функции ***\_CNInterrupt()*** (см. **Ошибка! Источник ссылки не найден.**) – обработчика прерывания по изменению уровня синхросигнала) с периодом, равным кванту синхронизации *Tsync = 62,5 мкс*. ***BlockExch\_interrupt()*** выполняет следующие действия:

1. до включения модуля в штатную работу:

* изменяет на одну дискрету значение таймера синхронизации начала работы модуля со смежным прибором;
* устанавливает признак включения модуля в штатную работу при условии, что произошла синхронизация со смежным прибором и данное прерывание является 16-м (последним в текущем 1 мс цикле главного потока);

1. после включения модуля в штатную работу:

вызывает функцию ***BlockExch\_run()*** драйвера BlockExch\_driver (см. **Ошибка! Источник ссылки не найден.**), которая, в свою очередь, вызывает функции***transmition()*** и ***reception()***, непосредственно осуществляющих передачу и прием данных, взаимодействуя с физическими линиями связи.

* + - 1. Модуль ActivityManager\_dataTypes
         1. Назначение

Типы данных для модуля управления активностью прибора.

* + - * 1. Интерфейсы

T\_O\_RELAY\_OPERATE– таймаут на включение реле, мс. Время после выдачи прибором команды на включение реле, по истечении которогокоманда должна быть выполнена (при условии, что реле исправно);

T\_O\_RELAY\_RELEASE– таймаут на выключение реле, мс. Время после выдачи прибором команды на выключение реле, по истечении которого команда должна быть выполнена (при условии, что реле исправно).

* + - * 1. Описание
    1. Компоненты AnalogInput, AnalogMeasurement
       1. Назначение

Компоненты *AnalogMeasurement* и *AnalogInput* предназначены для оцифровки, обработки, приведения к физическим величинам и контролю аналоговых сигналов, поступающих на входы МК. Их работа тесно связана, поэтому будет рассмотрена в одном разделе.

* + - 1. Состав компонента

В состав компонента входят модули:

1. *AnalogInput* − обработка аналоговых сигналов:
   1. *Adc* – модуль работы со встроенным АЦП;
   2. *AnalogInput* − обработка аналоговых сигналов;
   3. *DigitalFilter* − модуль управления цифровых фильтров;
   4. *dsPI33\_FIR\_filter* – модуль реализации КИХ фильтра (модуль написан на языке ассемблер для увеличения быстродействия).
2. *AnalogMeasurement* − измеритель аналоговых сигналов.
   * + 1. Описание
          1. Принцип работы

Модуль выполняет оцифровку аналоговых сигналов, поступающих на входы МК, и их обработку в зависимости от заданного алгоритма. Частота дискретизации – 1 кГц для всех каналов, разрядность АЦП – 12 бит.

Аналоговые сигналы поступают на входы МК, настроенный в аналоговый режим. При помощи встроенного АЦП сигнал преобразуется в цифровое значение. Для работы используется только один АЦП (в контроллере их имеется два), сигнал на который через встроенный мультиплексор последовательно подаётся с аналоговых входов. АЦП работает в режиме автоматического сканирования каналов. При запуске преобразования АЦП автоматически сканирует двенадцать измерительных каналов. В одном цикле преобразования аналоговые входы через мультиплексор последовательно подключаются к АЦП. Таким образом, после завершения цикла преобразования в буфере ADCBUFАЦП появляются двенадцать отсчётов входных сигналов (по одному на каждый канал). ADCBUF− аппаратный буфер, в который АЦП сохраняет результат преобразования при сканировании каналов. Перед началом нового преобразования все данные из буфера ADCBUF должны быть обработаны пользователем с помощью функции ***Adc\_run()***.

Цикл преобразования запускается в начале 0-го прерывания (см. рисунок 7). Этим достигается синхронность начала преобразований в двух процессорах, т.к. сетка прерываний в двух процессорах синхронна. После окончания цикла преобразования данные всех каналов, измеренных АЦП, находятся в буфере ADCBUF. При этом данные с канала 0 помещены в ADCBUF0, канала 1 − ADCBUF1 и т.д. Так образуются каналы обработки данных. Для определения завершения цикла преобразования используется функция ***Adc\_isReady()***, которая возвращает значение *true* когда новые данные помещены в буфер ADCBUFи ожидают обработки.



Рисунок 7 – Синхронизация измерений сигналов АЦП и их обработка.

В каждом миллисекундном цикле функция ***Adc\_run()*** копирует данные из буфера ADCBUF0-ADCBUF11 в буфер *adcData*, освобождая буфер ADCBUF0-ADCBUF11 для следующего преобразования. В буфере *adcData* создаётся очередь из отсчётов для их дальнейшей обработки. Функция ***AnalogInput\_run()***, которая обрабатывает отсчёты входного сигнала, асинхронна (не привязана к номеру прерывания), вызывается один раз за миллисекундный цикл и может быть вызвана планировщиком задач в любое время (см. рисунок 7). Таким образом, в буфере *adcData* могут находиться один или два отсчёта по каждому из каналов.

Структурная схема обработки сигналов (измерительных каналов) показана на рисунке 8.



Рисунок 8 – Структурная схема обработки сигналов (измерительных каналов).

Путь, который проходит отсчёт входного сигнала от буфера выборки АЦП ADCBUF0-ADCBUF11 до выхода AnalogMeasurement, будем называть измерительным каналом.

Обработку отсчётов выполняет функция ***AnalogInput\_run()***. Она обрабатывает данные, накопленные в буфере *adcData*. На этапе проектирования, в зависимости от особенностей входного сигнала для каждого из каналов, задан тип входного сигнала. Возможные типы сигналов перечислены в структуре *AnalogInputScript* (компонент AnalogInput).

Типы сигналов могут быть:

* *eAinScriptNone* − обработка сигнала не выполняется;
* *eAinScriptPeriodic* − периодический сигнал;
* *eAinScriptDirect* − постоянный сигнал;
* *eAinScriptError* – неинициализированное значение (при попытке обработки сигнала с этим значение прибор переходит в ЗС).

На выходе функции ***AnalogInput\_run()*** формируются отсчёты, обработанные по установленному алгоритму согласно типа входного сигнала.

Если тип обработки определён как *eAinScriptNone*, то обработка сигнала не выполняется, и функция выдаёт данные, которые были считаны с АЦП.

Если тип обработки определён как *eAinScriptPeriodic*, то для входного сигнала выполняется следующая обработка:

* устранение смещения нуля сигнала;
* фильтрация сигнала при помощи цифрового фильтра (опционально);
* вычисление СКЗ;
* вычисление среднего значения сигнала.

Если тип обработки определён как *eAinScriptDirect*, то для входного сигнала выполняется следующая обработка:

* вычисление СКЗ;
* вычисление среднего значения сигнала.

Тип обработки сигнала для каждого канала задаётся функцией ***AnalogInput\_setChannelSettings()*** перед работой модуля. Тип обработки передаётся в качестве параметра *AnalogInputScript*.

***AnalogInput\_getData()*** – функция для чтения обработанного значения данных в соответствующем канале измерения.

Инициализация измерительного канала выполняется функцией ***AnalogInput\_setChannelSettings()****.* В качестве параметра передаются следующие параметры:

* *channel* – номер инициализируемого канала;
* *filterId* – идентификатор цифрового фильтра;
* *script* – тип обработки сигнала в инициализируемом канале (*eAinScriptNone, eAinScriptPeriodic* или *eAinScriptDirect*);
* *bias* – смещение нуля, если тип обработки задан как *eAinScriptDirect* (для других типов обработки параметр игнорируется);
* *aws* – размер окна усреднения сигнала.

В приборе используется четыре аналоговых канала измерения (см. таблицу 5).

Таблица 5 – Назначение измерительных каналов прибора ОКПС-Е-К

| **Канал** | **Название** | **Назначение сигнала** |
| --- | --- | --- |
| 0 | *eAinchKREF* | Половина опорного напряжения АЦП смежного МК |
| 1 | *eAinch3V3* | Половина напряжения питания МК |
| 2 | *eAinchREF* | Напряжение для контроля правильности функционирования АЦП |
| 3 | *eAinchGEN* | Напряжение выходного сигнала прибора |
| 4 | *eAinchR1* | Канал сигнала датчика R1 |
| 5 | *eAinchR2* | Канал сигнала датчика R2 |
| 6 | *eAinchIV* | Канал тока фазы V |
| 7 | *eAinchIU* | Канал тока фазы U |
| 8 | *eAinchIW* | Канал тока фазы W |
| 9 | *eAinchUV* | Канал напряжения фазы V |
| 10 | *eAinchUU* | Канал напряжения фазы U |
| 11 | *eAinchUW* | Канал напряжения фазы W |

Канал 0 (*eAinchKREF)* служит для контроля исправности опорного напряжения ИОН для АЦП контроллера. Т.к. измерение напряжения собственного ИОН невозможна, контролируется половина напряжения ИОН смежного контроллера. Таким образом, в идеальном случае, результат измерения АЦП должен быть половиной максимального значения АЦП (в нашем случае, 4096/2 = 2048). В реальной схеме имеются погрешности, обусловленные неточностью ИОН, элементов делителя напряжения и погрешностью АЦП, поэтому происходит проверка на допустимое отклонение.

Канал 1 (*eAinch3V3)* служит для контроля напряжения питания МК. На вход АЦП подаётся половина напряжения питания МК.

Канал 2 (*eAinchREF)* служит для контроля функционирования АЦП. На вход этого канала подаётся опорное напряжение. Результат преобразования должен быть равен максимальному значению АЦП – 4096.

Канал 3 (*eAinchGEN)* служит для измерения напряжения контрольного генератора.

Канал 4 (*eAinchR1)* служит для измерения напряжения датчика R1.

Канал 5 (*eAinchR2)* служит для измерения напряжения датчика R2.

Канал 6 (*eAinchIV)* служит для измерения тока двигателя по фазе V.

Канал 7 (*eAinchIU)* служит для измерения тока двигателя по фазе U.

Канал 8 (*eAinchIW)* служит для измерения тока двигателя по фазе W.

Канал 9 (*eAinchUV)* служит для измерения напряжения по фазе V.

Канал 10 (*eAinchUU)* служит для измерения напряжения двигателя по фазе U.

Канал 11 (*eAinchUW)* служит для измерения напряжения двигателя по фазе W.

Измеренные значение синхронизируется со смежным контроллером с периодом 100 мс (определено константой *ANALOG\_SYNCHRO\_PERIOD*) в функции ***AnalogMeasurement\_run()****.*

Модуль *DigitalFilter* содержит в себе функции, реализующие управление цифровыми фильтрами и вызов функции фильтрации. Входные сигналы от датчиков R1 и R2 содержит помехи. Такие помехи вносят искажения в амплитудное значение сигнала, вызывая погрешности при измерениях. Для выделения полезного сигнала служит полосовой фильтр, настроенный на 62,5 Гц – частоту сигнала контрольного генератора.

Эффективная реализация цифрового фильтра требует использования специфических ассемблерных команд DSP-процессора (МАС-команд), аккумуляторов данных (длиной 40 бит), а организация кольцевого буфера данных использует специальные режимы индексной адресации. Эти особенности являются аппаратно-зависимыми, а реализация таких команд в языке С не предусмотрена. Поэтому функция фильтрации написана на языке ассемблер и оформлена отдельным модулем *dsPIC33\_FIR\_Filter.*

Модуль *dsPIC33\_FIR\_filter* реализует функцию КИХ-фильтра (см. рисунок 9). Несмотря на то, что КИХ-фильтр реализуется при большем объёме используемой памяти, по сравнению с фильтром с бесконечной импульсной характеристикой, он обладает существенным преимуществом – всегда устойчив, т. к. в нём отсутствуют обратные связи.



Рисунок 9 − Структурная схема КИХ-фильтра.

Как видно, выходной сигнал фильтраполучается в результате суммирования задержанных и взвешенных входных сигналов.Так, если *x(n)* – сигнал на входе линии задержки с *k-*отводами, то на выходе будет сигнал:

|  |  |
| --- | --- |
|  | (1) |

где – постоянные весовые коэффициенты фильтра, рассчитанные на этапе проектирования фильтра, – отсчёты входного сигнала фильтра в линии задержки.

В ОКПС-Е-К реализован полосовой фильтр 228-го порядка с центральной частотой 62,5 Гц и шириной полосы пропускания 6 Гц. Фильтр синтезирован методом Кайзера. АЧХ фильтра показана на рисунке 10.

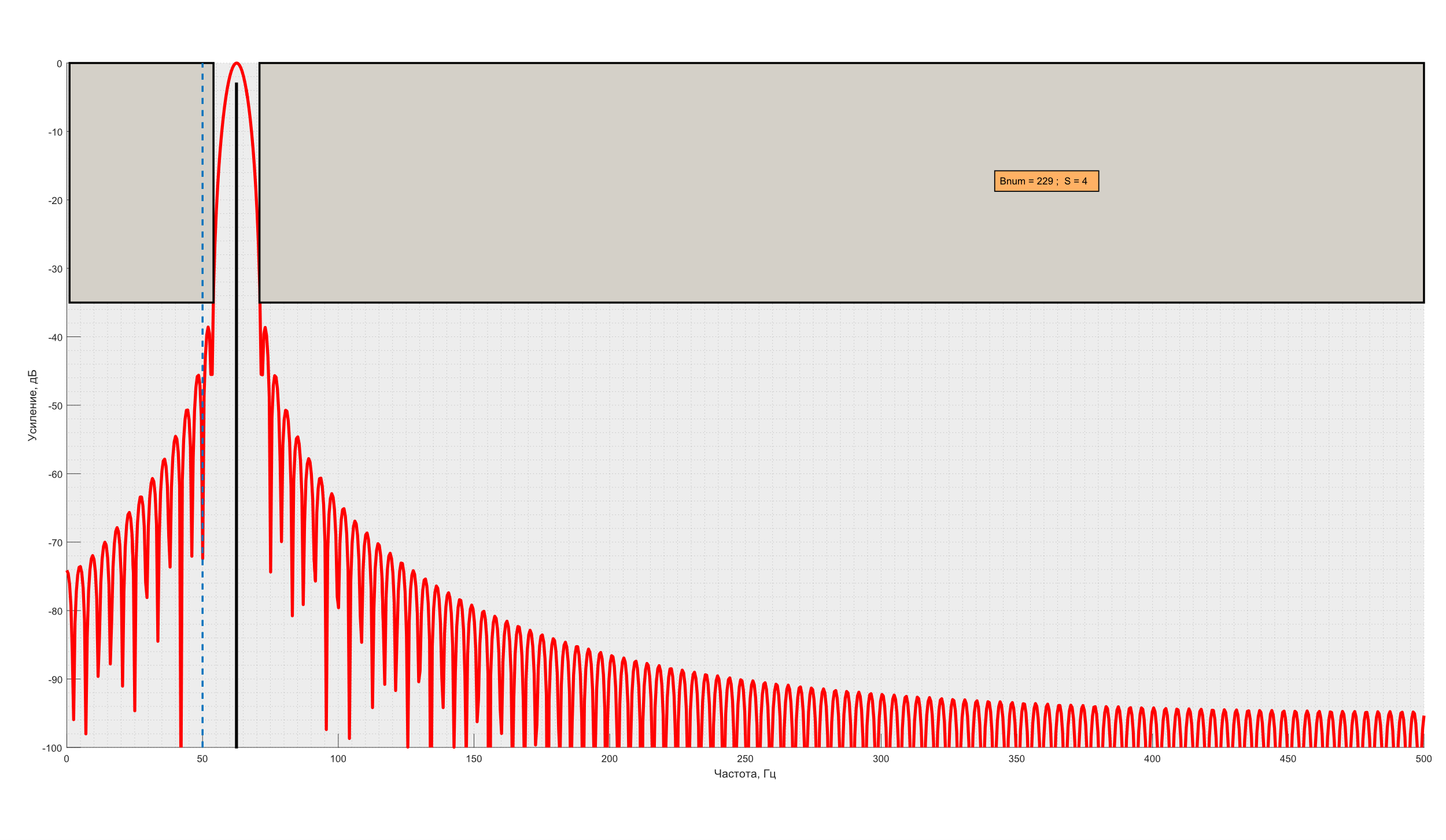


Рисунок 10 – АЧХ цифрового фильтра для сигналов R1 и R2.

Компонент AnalogMeasurementпредназначен для:

* настройки режимов работы измерительных каналов;
* приведения измерений в аналоговых каналах к физическим величинам;
* синхронизации полученных измерений между МК для сигналов, отвечающих за безопасную работу прибора.

Перед началом работы все измерительные каналы должны быть проинициализированы функцией ***AnalogInput\_setChannelSettings()***.

В зависимости от настройки каналов, коэффициентов масштабирования, смещения и типов измеряемых величин, данные обрабатываются и помещаются в буфер *analogMeasData*. Эти данные являются выходом измерительного канала и доступны для использования внешними функциями.

* + - * 1. Статическая модель

Статическая модель компонентов AnalogMeasurement и AnalogInput показана на рисунке 11.

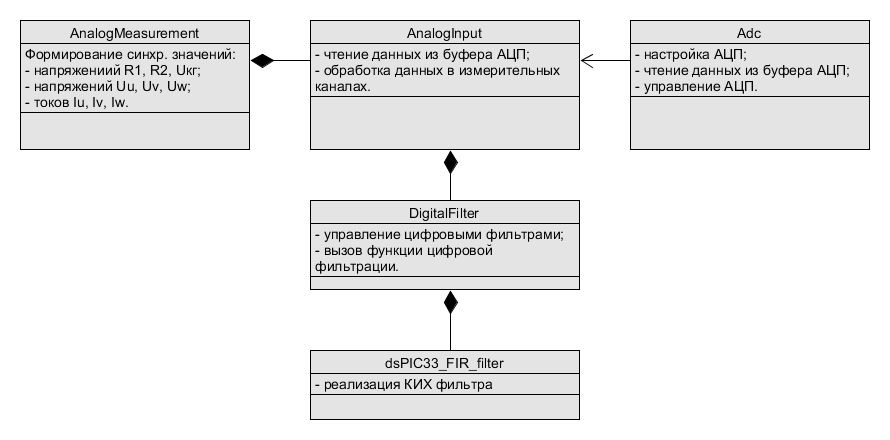


Рисунок 11 – Статическая модель компонентов AnalogMeasurement и AnalogInput.

* + 1. Компонент BinIn
       1. Назначение

Компонент BinIn предназначен для чтения, обработки и формирования синхронизированных значений входных дискретных сигналов, а также для контроля исправности аппаратных средств чтения входных дискретных сигналов.

* + - 1. Состав компонента

В состав компонента входят следующие модули:

* *BinIn* − чтение дискретных входов и синхронизация считанных значений;
* *BinInDecoder* – дешифратор считанных значений (модуль написан на языке ассемблер для увеличения быстродействия).
  + - 1. Описание
         1. Общие сведения

Дискретные входы, считываемые МК, делятся на две части – отвечающие за безопасность и не отвечающие за безопасность. Как отвечающие за безопасность и не отвечающие за безопасность дискретные входы синхронизируются между МК Master и MK Slave. Компоненты и модули ПО считывают состояние дискретных входов не с выводов МК, а через специальные интерфейсные функции компонента BinIn. Это позволяет ПО, выполняемому в разных МК, оперировать одинаковыми данными, т.к. считанные значения дискретных сигналов синхронизируются.

К числу отвечающих за безопасность дискретным входам относятся:

* дискретные входы адресных перемычек (установлены на кросс-плате);
* дискретные входы режимных перемычек (установлены на кросс-плате);
* дискретные входы перемычек контрольной суммы (установлены на кросс-плате);
* дискретный вход сигнала контроля РПВ.

К числу не отвечающих за безопасность дискретным входам относятся:

* дискретный вход сигнала состояния источника питания 24 В (сигнал KPOW14);
* дискретный вход сигнала состояния источника питания 220 В (сигнал KPOW220).

Упрощенная схема, поясняющая принцип работы компонента *BinIn,* изображена на Рисунке 12. Дискретные сигналы, считываемые с кросс-платы объединены в матрицу. Строки матрицы (выходные сигналы) образованы девятью сигналами SA0…SA4, SK5, SS6, SS7 и SR8, а столбцы (входные сигналы) четырьмя сигналами AD0…AD3. Сигналы подачи питания на строки матрицы подключены к МК Master, а сигналы чтения строк – к МК Slave.

Считывание сигналов производится последовательным сканированием матрицы. Для сканирования тетрады сигналов, подключённой к SA0 МК Master выставляет уровень лог. 0 на сигнал А0\_М. Все остальные сигналы строки должны быть установлены в лог. 1. Цепь SA0 запитывается напряжением +5VA через открытый транзистор оптопары V3.1. Тетрада сигналов AD0…AD3 считывается МК Slave (сигналы D0…D3 на входах МК Slave, DD2) через оптопары V10 и V11. Таким образом последовательно сканируя тетрады сигналов, состояние всех перемычек считывается.

Для контроля исправности схем считывания сигналов столбца установлен оптрон V1.1. Так, при отсутствии питания на ряду SA1, сигнал D0 должен быть в состоянии лог. 1 при исправных цепях считывания. Для контроля исправности цепи считывания установлен оптрон V1.1. При уровне лог. 0 сигнала CH-D0 выходной транзистор оптрона V1.1 открывается и ток протекает через V10.1. При этом сигнал D0 примет значение лог. 0. Таким образом обеспечивается контроль исправности цепей чтения дискретных сигналов.

Дискретные сигналы KPOW24 и KPOW220 подаются на входы МК PA8 и PA13 через оптроны V16 и V23 соответственно. Исправность цепей чтения этих сигналов не контролируются.

Модуль *BinIn* осуществляет чтение сигналов, заключённых в матрицу и сигналов, которые непосредственно поданы на выводы процессора в одном цикле (каждую миллисекунду). Период опроса равен 56 мс для всех сигналов.

При формировании выходного значения сигнала используется фильтр «4 из 6». Для формирования лог. 1 должно быть считано четыре состояния лог. 1 из шести. Для формирования лог. 0 должно быть считано четыре состояния лог. 0 из шести. Если ни одно из условий не выполняется, то предыдущее стабильно считанное значение остаётся неизменным. В каждом цикле значение дискретных сигналов синхронизируется модулем МКО.

В каждом цикле чтение проводится диагностика исправности цепей считывания дискретных сигналов, которые являются безопасными. При обнаружении шести последовательных ошибок при контроле схем считывания дискретных входов, прибор переходит в 3С с кодом *ePS\_BinInErrorTestDecoder*.

Рисунок 12 – Упрощенная схема чтения дискретных сигналов.

* + - * 1. Принцип работы модуля BinIn

Модуль инициализируется путем вызова функции *BinIn\_ctor()* до основного рабочего цикла. *BinIn\_ctor()* инициализирует все переменные, структуры и порты МК, необходимые для работы модуля.

Для обеспечения синхронности выдачи управляющих воздействий на линии строк и считывания данных со столбцов двумя МК функции модуля *BinIn\_DataRead\_interrupt()* и *BinInAddrSet\_interrupt ()* вызывается в 15-ом и 0-ом прерывании (полностью работают в прерываниях). Это сделано для фиксированного время установления сигналов. Функция *BinIn\_run()*, которая вызывается в основном цикле, служит для обработки данных, считанных в прерывании функцией *BinIn\_DataRead\_interrupt().*

* + - * 1. Статическая модель

Основная рабочая функция модуля ***BinInD\_run()*** вызывается в режиме работы (состоянии) *StateTop* в главном потоке управления (основном цикле Main) с периодом 1 мс.

Статическая модель компонента BinIn изображена на Рисунке 13.

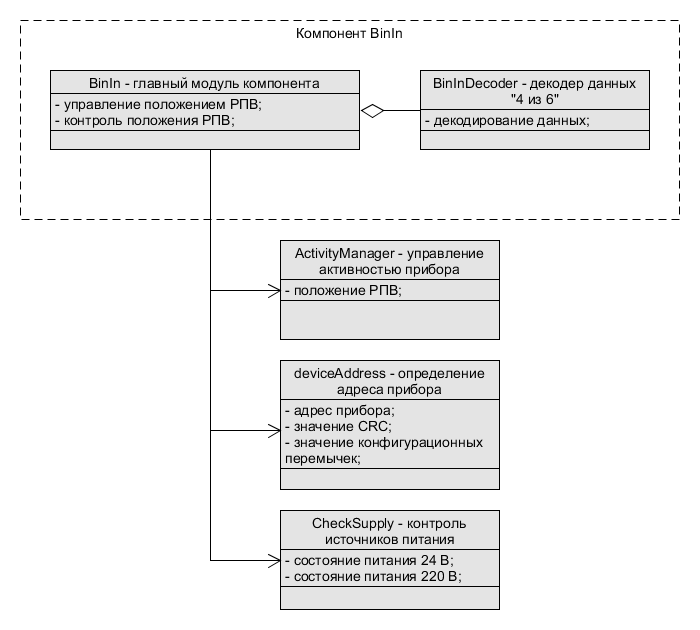


Рисунок 13 – Статическая диаграмма модуля BinIn.

* + - 1. Модуль BinInDecoder
         1. Назначение

Чтение дискретных входов, подключенных через дешифратор.

* + - * 1. Описание

Модуль проводит считывание дискретных входов, подключенных через дешифратор, контроль исправности дешифратора и антидребезговую (антипомеховую) обработку.

**Сценарий работы:**

1. задается состояние дешифратора для чтения группы входов;
2. задержка на установление сигналов входов;
3. считывается состояние входов;
4. задержка на 1 цикл работы (обычно 1 мс) для гарантии корректного считывания входов в обоих каналах;
5. если считаны не все группы входов, то переход на а);
6. задается состояние дешифратора на проверку его исправности (тест);
7. задержка на установление сигналов входов;
8. считывается состояние входов теста;
9. повторение ж) и к)и) несколько раз;
10. антипомеховая обработка состояний входов теста;
11. антипомеховая обработка состояний дискретных входов;
12. фиксация нового состояния дискретных входов;
13. переход на а).

**Исключения:**

1. состояние входов теста не совпадает с эталонным – состояние дискретных входов бракуется;
2. состояние входов теста не совпадает с эталонным несколько раз подряд – переход в ЗС.

**Дополнительные сведения:**

1. параллельно сценарию проводится синхронизация состояний дискретных входов с периодом 100 мс.
   * 1. Компонент BlackBox
        1. Назначение

Компонент предназначен для накопления служебной информации, сохранения её во внешний EEPROM и чтения информации при помощи специальных интерфейсных средств через шину CAN.

* + - 1. Состав компонента

Компонент состоит из одноимйнного модуля.

* + - * 1. Описание

***BlackBox\_save()*** − сохранение в EEPROM данных из буферов ОЗУ. Данные, накопленные в буферах ОЗУ, копируются в EEPROM.

***BlackBox\_read()*** − чтение из EEPROM информации и выдача ее в канал связи. Данные, сохранённые во внешнем EEPROM, могут быть вычитаны при помощи специального оборудования. Эта функция обеспечивает чтение и передачу данных из внешнего EEPROM.

Взаимодействие модулей и компонентов при работе с *BlackBox* показано на рисунке 14.

***BlackBox\_saveAddCodeProtectionState()*** − сохранение параметра, уточняющего причину ЗС. При переходе прибора в ЗС может сохраняться в EEPROM не только код ЗС, но и дополнительные параметры. Эти параметры содержат уточняющую информацию о причине перехода в ЗС. Их содержание задаёт модуль, вызвавший переход в 3С.



Рисунок 14 – Диаграмма взаимодействия при работе с ЧЯ

* + 1. Компонент CheckSupply
       1. Назначение

Компонент CheckSupply предназначен для определения состояния источников питания прибора 24 В и 220 В. Компонент обрабатывает информацию с датчиков, фильтрует, принимает решение о наличии либо отсутствии напряжения питания и синхронизирует значение с соседним МК. Все компоненты ПО, которым необходима информация о состоянии источников питания, должны получать её из компонента CheckSupply, т.к. значение синхронизированное.

* + - 1. Состав компонента

Компонент состоит из одного одноименного модуля.

* + - 1. Описание

Дискретные сигналы с датчиков напряжений питания 24 В и 220 В считываются через функции *BinIn\_is220vOn()* и *BinIn\_is24vOn()*. При наличии питания соответствующий ему сигнал принимает значение логического 0, при отсутствии – логической 1. Сигналы подаются внешней схемой детектора напряжения и являются асинхронными. Для корректного определения состояния источника питания и исключения ложных срабатываний в условиях помех осуществляется фильтрация сигналов датчиков.

Фильтрацию сигнала выполняет функция ***CheckSupply\_run()***. Каждую миллисекунду выполняется опрос датчиков. Фильтрация сигналов с датчиков выполняется функцией ***HystFltr\_run()***. фильтра с временным гистерезисом. Временные параметры гистерезиса (время включения и выключения) задаются функцией ***HystFltr\_ctor()*** при начальной инициализации компонента (функция ***CheckSupply\_ctor()***). Так, время включения для сигнала с датчика источника питания 24 В определено константой SUPPLY\_24V\_EN, равной 500 мс, время выключения определено константой SUPPLY\_24V\_DIS, равной 250 мс. Для сигнала с датчик источника питания 220 время включения определено константой SUPPLY\_220V\_EN, равной 900 мс, время выключения определено константой SUPPLY\_220V\_DIS, равной 10 мс. Алгоритм обработки сигнала питания 24 В (см. рисунок 15). Аналогичным образом обрабатывается сигнал с датчика напряжения питания 220 В.



Рисунок 15 − Алгоритм обработки сигналов KPOW24и KPOW220

Функции ***CheckSupply\_isU220vOn()*** и ***CheckSupply\_isU24vOn()*** возвращают отфильтрованное значение сигналов с датчиков напряжений питаний. Возвращаемое значение *true* означает наличие питающего напряжения, *false* – его отсутствие. Эти значения используют все компоненты ПО для определения состояния источников питания прибора.

* + 1. Компонент ConfigMK
       1. Назначение

Прибор ОКПС-Е-К построен по двухпроцессорной схеме. Имеется два процессора – Master и Slave. ПО в МК Master и Slave одинаковое. Однако в некоторых алгоритмах необходимо иметь информацию, на каком месте находится процессор. Компонент определяет на каком месте установлен процессор – на месте Master или на месте Slave.

* + - 1. Состав компонента

Компонент состоит из одного одноименного модуля.

* + - 1. Описание

Конфигурация процессора (*Master* или *Slave*) определяется состоянием порта RG6. Если на этом порту присутствует логическая 1 – процессор конфигурируется как *Master*, если логический 0 – процессор конфигурируется как *Slave*. Состояние порта определяется один раз при запуске программы.

Функция ***ConfigMK\_ctor()*** выполняет настройку порта RG6 и при старте ПО определяет состояние порта.

Для определения типа процессора все компоненты ПО пользуются функциями ***ConfigMK\_isMaster()*** и ***ConfigMK\_isSlave()***. Если процессор имеет тип *Master*, то функция ***ConfigMK\_isMaster()*** возвращает значение *true*, а функция ***ConfigMK\_isSlave()*** – *false*. Если процессор имеет тип *Slave*, то функция ***ConfigMK\_isMaster()*** возвращает значение *false*, а функция ***ConfigMK\_isSlave()*** – *true*.

* + 1. Компонент DebugTools
       1. Назначение

Компонент содержит функции, позволяющие проводить отладку прибора, выдавать отладочную информацию по специальным интерфейсам, сохранять текущее состояние переменных («чёрный ящик») для анализа причин перехода прибора в ЗС.

* + - 1. Состав компонента

В состав компонента входят следующие модули:

* *DebugTools* − средства отладки прибора;
* *Tracing* − трассировка параметров.
  + - 1. Описание

Модуль DebugTools предназначен для выдачи диагностической информации по интерфейсу SPI и на тестовые пины процессора. Используется при отладке прибора. В рабочей версии ПО все выводы, которые задействованы в выдаче информации, отключены (макрос ENABLE\_DEBUG\_PINS) не определён.

* + - 1. Модуль Tracing
         1. Назначение

Компонент содержит функции, позволяющие проводить отладку прибора, сохранять текущее состояние переменных для анализа при переходе прибора в ЗС.

* + - * 1. Описание

В процессе работы некоторые компоненты программы записывают служебную информацию в массив данных, который хранится в ОЗУ процессора. При переходе прибора в ЗС по этой информации будет возможно установить цепочку событий, которые привели к этому переходу. При переходе в ЗС накопленные в ОЗУ данные копируются в EEPROM, чтобы их сохранить при выключении питания. Всего сохраняется семь (определяется константой TRACING\_BUF\_QTY) буферов памяти, размер каждого буфера равен 256 байт (определяется константой TRACING\_BUF\_SIZE).

* + 1. Компонент DeviceAddress
       1. Назначение

Компонент предназначен для чтения перемычек на кросс-плате и определения по ним адреса прибора и конфигурационных признаков, а также для периодической их проверки в процессе работы.

* + - 1. Состав компонента

Компонент состоит из одного одноименного модуля.

* + - 1. Описание

На кросс-плате АКРЦ-ЕН находятся перемычки, определяющие адрес (для связи по интерфейсу RS-422 с УС согласно Протоколу обмена) и конфигурацию прибора (всего 24 перемычки и соответствующих им сигнала):

1. сигналы 7-0 (*A7…A0*):
2. сигнал *A0* – признак статуса прибора:

* *A0* = «1» – прибор должен работать как резервный;
* *A0* = «0» – прибор должен работать как основной.

1. сигналы *A7…A1* определяют биты 7…1 в поле адреса прибора;
2. сигналы 15-8 (*A15…A8*):
3. сигнал *A8* – тип УС:

* *A8* = «1» – УС типа «Ebilock950»;
* *A8* = «0» – УС, отличная от «Ebilock950» («Стрела-10»).

1. сигналы 23-16 (CRC7…CRC0) – контрольная сумма CRC-8 для сигналов 15-0.

Чтение состояния перемычек выполняется с помощью интерфейсных функций компонента BinIn, которые возвращают прошедшие антидребезговую обработку синхронизированные значения сигналов (см. 4.3.12):

* ***BinIn\_getAddrJumpers()*** для чтения адресных перемычек (сигналы *A15…A0*);
* ***BinIn\_getCrcJumpers()*** для чтения перемычек контрольной суммы CRC-8 (сигналы CRC7…CRC0).

Функция ***DeviceAddress\_run()*** реализует основной алгоритм работы модуля и вызывается в режиме работы (состоянии) *StateTop* в главном потоке управления (основном цикле *main*) с периодом 1 мс. Функция выполняет основной алгоритм если включен модуль deviceAddress (была вызвана функция ***DeviceAddress\_ctor()***).

Работа ***DeviceAddress\_run()*** состоит из двух фаз:

1. *eDeviceAddr\_firstCheck* – фаза первичной проверки, в которую функция переходит после инициализации модуля (вызова ***DeviceAddress\_ctor()***);
2. *eDeviceAddr\_periodCheck* – фаза периодической проверки состояния перемычек, в которую функция переходит после успешного окончания фазы первичной проверки *eDeviceAddr\_firstCheck*.

В работе модуля используются два счетчика (таймера) для отсчета временных интервалов:

1. *failureTimeoutCnt* – счетчик времени таймаута перехода в ЗС;
2. *timeoutCnt*:

* счетчик времени для периодической проверки состояния перемычек в фазе *eDeviceAddr\_firstCheck*;
* счетчик времени для периодической проверки изменения состояния перемычек в фазе *eDeviceAddr\_periodCheck*.

В ***DeviceAddress\_ctor()*** инициализируются следующие счетчики:

* *failureTimeoutCnt* – значением FAILURE\_TRANSITION\_T\_O = 20000 мс;
* *timeoutCnt* – значением FIRST\_CHECK\_T\_O = 300 мс.

Счетчики работают на уменьшение и декрементируются в ***DeviceAddress\_run()*** независимо от фазы работы.

В фазе *eDeviceAddr\_firstCheck* выполняются следующие действия:

1. ожидание окончания таймаута для чтения перемычек и выполнения проверки их корректности (*timeoutCnt*);
2. по истечении времени FIRST\_CHECK\_T\_O – считывание состояния перемычек, вычисление контрольной суммы CRC-8 и ее сравнение со считанной. В случае их несовпадения – перезапуск таймера *timeoutCnt* значением FIRST\_CHECK\_T\_O;
3. проверка счетчика *failureTimeoutCnt*: если он досчитал, т.е. за время FAILURE\_TRANSITION\_T\_O не было считано корректное значение перемычек, происходит переход в ЗС с кодом *ePS\_DeviceAddressFirstCheckError*;
4. если в пункте б) расчетная CRC-8 совпала со считанной:
5. формирование на основе считанных данных признаков статуса прибора, типа УС и адреса прибора для работы по интерфейсу RS-422 (установка бита 0 в «1» и бита 8 в «0»);
6. установка признака корректности считанных перемычек;
7. перезапуск таймеров:

*timeoutCnt* – значением *PERIODIC\_CHECK\_T\_O* = 5000 мс;

*failureTimeoutCnt* – значением *FAILURE\_TRANSITION\_T\_O*;

1. переход к фазе *eDeviceAddr\_periodCheck*.

В фазе *eDeviceAddr\_periodCheck* выполняются следующие действия:

1. считывание состояния адресных перемычек и перемычек CRC-8;
2. переход в ЗС с кодом *ePS\_DeviceAddressCheckError* при одновременном (по И) выполнении следующих условий:
3. прибор основной;
4. текущее считанное значение адресных перемычек или текущее считанное значение перемычек CRC-8 не совпадает со значением, считанным в фазе *eDeviceAddr\_firstCheck*;
5. неисправно РПВ;
6. питание «24 В» в норме;
7. переход в ЗС с кодом *ePS\_DeviceAddressCheckError* при одновременном (по И) выполнении следующих условий:
8. прибор резервный;
9. текущее считанное значение адресных перемычек или текущее считанное значение перемычек CRC-8 не совпадает со значением, считанным в фазе *eDeviceAddr\_firstCheck*;
10. питание «24 В» в норме.

Отказ с кодом *ePS\_DeviceAddressCheckError* является комплексным и вызывается отказом источника питания «7V\_AD», запитывающим схему чтения перемычек и схему чтения состояния реле.

1. по истечении времени *PERIODIC\_CHECK\_T\_O* – считывание состояния перемычек, сравнение их значения со считанным в фазе *eDeviceAddr\_firstCheck*, в случае несовпадения – перезапуск таймера *timeoutCnt* значением *PERIODIC\_CHECK\_T\_O*;
2. проверка счетчика *failureTimeoutCnt*: если он досчитал, т.е. за время *FAILURE\_TRANSITION\_T\_O* текущее считываемое значение перемычек не совпало со значением, считанным в фазе *eDeviceAddr\_firstCheck*, происходит переход в ЗС с кодом *ePS\_DeviceAddressCheckError*;
3. если в пункте г) значения перемычек совпали, происходит перезапуск таймеров:

* timeoutCnt – значением PERIODIC\_CHECK\_T\_O;
* failureTimeoutCnt – значением FAILURE\_TRANSITION\_T\_O.

Если при входе в функцию ***DeviceAddress\_run()*** переменная, содержащая значение фазы работы функции, отлична от *eDeviceAddr\_firstCheck* или *eDeviceAddr\_periodCheck*, происходит переход в ЗС с кодом *ePS\_DeviceAddressStepCntError*.

* + 1. Компонент Eeprom
       - 1. Назначение

Компонент Eeprom предназначен для обеспечения программного интерфейса доступа к внешней EEPROM, который подключен к МК dsPIC33.

* + - * 1. Состав компонента

Компонент состоит из одноимённого модуля.

* + - * 1. Описание

Общие сведения

Модуль выполняет следующие операции:

* чтение данных из EEPROM;
* запись данных в EEPROM.

Модель использования модуля

Перед использованием любой функции модуля необходимо проинициализировать его работу, вызвав функцию ***Eeprom\_ctor()***. Поскольку модуль использует компонент InterChannel, а при инициализации модуля EEPROM настраивается один из параметров синхронизации, данная функция должна вызываться после инициализации компонента InterChannel.

Для обеспечения работы асинхронных операций в главном потоке периодически должна вызываться функция ***Eeprom\_run()***. Данная функция реализует все операции с EEPROM. При каждом вызове функция выполняет один шаг любой операции.

Для обеспечения чтения данных из EEPROM необходимо использовать функцию ***Eeprom\_read()***, которая сразу же возвращает результат чтения. Операция чтения не такая затратная, с точки зрения использования процессорного времени, по сравнению с операцией записи, поэтому аналогичной асинхронной операции не предусмотрено. При работе на частоте 58 МГц операция чтения занимает около 2 мкс, а операция записи – около 4 мс (ограничена быстродействием внешней EEPROM).

Для обеспечения записи данных в EEPROM необходимо использовать функцию ***Eeprom\_write()***. Узнать об окончании операции можно при помощи функции ***Eeprom\_isReady()***.

* + 1. Компонент HysteresisFilter
       1. Назначение

Фильтрация сигналов с цифровых (дискретных) входах с гистерезисом.

* + - 1. Состав компонента

Компонент состоит из модуля HystFlt.

* + - 1. Описание

Под гистерезисом здесь понимается разница (задержка) во времени между реальным изменением сигнала на входе МК и принятием решения о его новом значении для дальнейшего использования в ПО. В общем случае, сигнал имеет разные гистерезисы по включению (при изменении сигнала с «0» на «1») и по выключению (при изменении сигнала с «1» на «0»). Под фильтрацией здесь понимается отсутствие реакции на изменение сигнала, длящееся меньше времени гистерезиса (см. рисунок 16).



Рисунок 16 – Диаграмма обработки дискретного сигнала с помощью модуля HystFltr.

Модуль HystFltr определяет тип – структуру данных объекта для фильтрации с гистерезисом – *digInpHystFilt\_type.*

Для каждого сигнала, который предполагается обрабатывать путем фильтрации с гистерезисом, с помощью функции ***HystFltr\_ctor()*** необходимо создать объект типа *digInpHystFilt\_type* и инициализировать его, передав в качестве параметров указатель на объект и значения (времена) гистерезисов по включению (*enHyst*) и выключению (*disHyst*). В исходном состоянии считается, что сигнал находится в состоянии «0».

Непосредственная обработка сигнала выполняется основной рабочей функцией модуля ***HystFltr\_run()***, которая вызывается для каждого инициализированного объекта с периодом 1 мс в главном потоке управления (в общем случае, для каждого сигнала период вызова может быть свой).

Компонентом HysteresisFilter по обработке дискретных сигналов пользуются компоненты CheckSupply и DiscreteInputsLogic.

* + 1. Компонент Indication
       1. Назначение

Компонент Indication предназначен для управления светодиодными индикаторами на передней панеле прибора.

* + - 1. Состав компонента

В состав компонента входят следующие модули:

* *Indication* − управление светодиодными индикаторами «RS1», «RS2», «ПОЛОЖЕНИЕ +», «ПОЛОЖЕНИЕ –» и «ПОТЕРЯ КОНТРОЛЯ» расположенными на передней панели прибора;
* *LedFailure* − управление светодиодным индикатором «ОТКАЗ», расположенным на передней панели прибора.
  + - 1. Описание
         1. Общие сведения

Модуль Indication обеспечивает управление всеми светодиодными индикаторами, находящимися на передней панели прибора, кроме светодиодного индикатора «ОТКАЗ».

Модуль обеспечивает выполнение следующих функций:

* включение и выключение индикатора;
* включение или выключение индикатора на заданный промежуток времени с возвратом в начальное состояние;
* циклическое включение и выключение индикатора с заданной длиной импульса и паузы (функция должна поддерживать задание логического уровня импульса);
* циклическое включение и выключение индикатора с двойным периодом, отдельно заданной длительностью импульса и паузы для первого и для второго импульса (функция должна поддерживать задание логического уровня импульса).

Для использования модуля Indication для каждого индикатора необходимо объявить переменную типа *indOutput\_type*. Затем проинициализировать ее вызвав одну из функций ***Indication\_ctorLed()*** или ***Indication\_ctorDummy()***. Каждая из этих функций соответственно проинициализирует переданную ей структуру.

Для использования модуля Indication для каждого светодиодного индикатора необходимо при помощи ***Indication\_ctorLed()*** или ***Indication\_ctorDummy()*** проинициализировать переменную типа *indOutput\_type*. Светодиодный индикатор подключен к выводу одного из МК (*Master* или *Slave*) и им управляется. Второй процессор участия в управления индикатором не принимает. Для удобства написания модулей, использующих в своей работе функции компонента Indication, различия в управлении скрыты внутри модуля Indication. ПО, работающее в МК *Master* и *Slave* пользуются функциями Indication не зависимо от того, подключен индикатор к нему или нет. Функция ***Indication\_ctorLed()*** должна работать в МК (*Master* или *Slave*), вывод которого непосредственно подключен к светодиоду. Функция ***Indication\_ctorLed()*** должна работать в МК (*Master* или *Slave*), вывод которого непосредственно подключен к светодиоду. Функция ***Indication\_ctorDummy()*** должна работать в МК (*Master* или *Slave*), который не управляет светодиодом.

Для обеспечения функционирования индикаторов необходимо вызывать функцию ***Indication\_run()*** для каждого созданного объекта с периодичностью один раз в миллисекунду. В качестве параметра этой функции необходимо передать структуру типа *indOutput\_type*, соответствующую конкретному индикатору.

Можно в произвольное время давать указание индикаторам выполнять ту или иную функцию.

Для включения и выключения индикатора необходимо использовать функции ***Indication\_on(***) и ***Indication\_off()***. Для включения или выключения индикатора на заданный промежуток времени с возвратом в начальное состояние необходимо использовать функции ***Indication\_pulseOn()***, ***Indication\_pulseOff()*** задав время на которое индикатор сменит свое состояние. Для запуска циклического включения и выключения индикатора (мигания) с заданной длиной импульса и паузы необходимо использовать функции ***Indication\_blink()***. Для задания двойного периода циклического включения и выключения индикатора (двойного мигания) необходимо или ***Indication\_blink2()***.

* + - 1. Модуль LedFailure
         1. Назначение

Управление светодиодным индикатором «ОТКАЗ», расположенным на передней панели прибора.

* + - * 1. Описание

Модуль LedFailure обеспечивает управление светодиодным индикатором «ОТКАЗ», находящимися на передней панели прибора. Этот светодиодный индикатор имеет схему управления, которая отличается от всех других светодиодных индикаторов. Поэтому управление им вынесено в отдельный модуль. Для включения и выключения индикатора «ОТКАЗ» необходимо использовать функцию ***LedFailure\_set().***

* + 1. Компонент Initial
       1. Назначение

Компонент предназначен для выполнения служебных функций при запуске программы.

* + - 1. Состав компонента

Компонент состоит из трёх модулей:

* *CheckCauseReset* – проверка причин сброса МК;
* *IdentHex* – запись в EEPROM даты сборки и версии прошивки;
* *TimeBeginSynchronization* – начальная синхронизация двух МК при запуске прибора;
* *TimeSynchronizationPorts* – порты для временной синхронизации МК.
  + - 1. Модуль IdentHex
         1. Назначение

Идентификация прибора (имя и идентификатор типа прибора) и ЗМ (версия ПО и дата создания ПО).

* + - * 1. Описание

Работа модуля (функция ***IdentHex\_run()***) обычно происходит по включению прибора. Запись в ЭНП проводится только в том случае, если данные в ЭНП отличаются от данных для записи. Таким образом, ресурс количества записей в ЭНП понапрасну не расходуется.

* + - 1. Модуль CheckCauseReset
         1. Назначение

Модуль *CheckCauseReset* предназначен для определения причин сброса МК. При корректном сбросе МК (корректным считается сброс по включению питания или по сигналу внешнего сброса на входе MCLR) выполнение программы продолжается. При любой другой причине сброса прибор переходит в 3С. Таким образом контролируется корректность запуска МК по включению питания прибора.

* + - * 1. Описание

Основная рабочая функция модуля ***checkCauseReset\_run()*** вызывается первой в главном потоке управления и выполняет следующие действия:

* проверяет содержимое регистра RCON (Reset Control Register), при некорректном сбросе (причины сброса, отличные от сброса по питанию или внешнего сброса) – переводит прибор в 3С с сохранением регистра RCON в «черном ящике» (массиве, содержащем информацию, уточняющую причину ЗС);
* при корректном сбросе обнуляет биты признаков причин сброса в регистре RCON и программа продолжает работать далее.
  + - 1. Модуль TimeBeginSynchronization
         1. Назначение

Модуль TimeSynchronization предназначен для решения задачи начальной синхронизация МК по времени.

* + - * 1. Описание

Основная рабочая функция модуля ***TimeSynchronization\_beginSynchro()*** вызывается в функции ***main()***, которая перед основным цикла главного потока управления.

Выполняет следующие действия:

1. с помощью макроса TIME\_SYNCHRO\_INIC\_PORTS инициализирует порты МК, подключенные к линиям синхронизации:
2. порт PORTE14 (входная линия синхронизации) – сигнал от соседнего МК;
3. порт PORTE15 (выходная линия синхронизации) – сигнал к соседнему МК, устанавливает начальное значение PORTE15 = 0.
4. в течение времени (примерно 20 с) ожидает появление устойчивого начального значения «0» входного синхросигнала (PORTE15), в другом случае – переход в ЗС с кодом *ePS\_MainErrorBeginSynchro*;
5. TIME\_BEGIN\_SYNCHRO\_WAIT = 1000000 раз в цикле анализирует состояние синхросигнала и подсчитывает значения PORTE15 = «0». Если PORTE15 был равен «0» меньше (TIME\_BEGIN\_SYNCHRO\_WAIT - TIME\_BEGIN\_SYNCHRO\_WAIT / 128)) = 7812 раз, происходит переход в ЗС с кодом *ePS\_MainErrorBeginSynchro*, в другом случае – выставляет PORTE14 = «1»;
6. TIME\_BEGIN\_SYNCHRO\_WAIT = 1000000 раз в цикле анализирует состояние синхросигнала и подсчитывает значения PORTE15 = «1». PORTE15 был равен «0» меньше (TIME\_BEGIN\_SYNCHRO\_WAIT - TIME\_BEGIN\_SYNCHRO\_WAIT / 128)) = 7812 раз, происходит переход в ЗС с кодом *ePS\_MainErrorBeginSynchro*, в другом случае – выставляет PORTE14 = «1» и заканчивает работу.

После этого считается, что оба МК достигли начальной синхронизации и готовы к штатной работе.

* + - 1. Модуль IdentHex

Предназначен для записи в EEPROM прибора даты сборки, версии ПО и идентификатора прибора. При первом запуски прибора проводится проверка на наличии корректных данных о версии, дате сборки и идентификаторе в EEPROM прибора. Если данные не корректны либо не инициализированы, происходит их запись в EEPROM.

* + - * 1. Описание

Версия ПО (VERSION\_NUMBER), дата сборки (VERSION\_DATE) и идентификатор прибора (*acNameHex*) определяются константами в момент сборки проекта. В EEPROM прибора *VERSION\_DATE* записывается по адресу ADDRESS\_EEPROM\_IDENT\_DATE, VERSION\_NUMBER по адресу ADDRESS\_EEPROM\_IDENT\_VERSION, а идентификатор прибора по адресу ADDRESS\_EEPROM\_IDENT\_ID.

* + - 1. Модуль TimeSynchronizationPorts
         1. Назначение

Временная синхронизация МК.

* + - * 1. Описание

Файл содержит макросы инициализации и работы с портами ввода/вывода временной синхронизации.

* + 1. Компонент IntegrCtrl
       1. Назначение

Компонент предназначен для контроля целостности обмоток двигателя и контроля контроллера коэффициента мощности (ККМ).

* + - 1. Состав компонента

Компонент состоит из двух модулей:

* *IntegrCtrl* – модуль, осуществляющий управление генератором тестовых импульсов и выполняющий контроль целостности рабочих цепей, а также контроль корректора коэффициента мощности;
* *IntegrCtrlGen –* модуль, генерирующий импульсы специальной формы для контроля целостности обмоток.
  + - 1. Описание

Алгоритм работы реализован как автомат состояний в функции ***IntegrCtrl\_run()***. При поступлении нового актуального приказа на перевод стрелки (то есть того, который требует выполнения), формирование тестовых импульсов модулем прекращается, а выполнение генерации трёхфазного сигнала, используя ШИМ-модуляцию модуля PWM МК в функции ***shuntShiftGen\_run()***будет осуществляться ТОЛЬКО после того, как на выходе генератора контрольных импульсов отсутствует формируемое напряжение. Формирование контрольных импульсов будет возобновлено при отсутствии выполнения приказа на перевод стрелки. По значениям токов фаз и напряжений фаз принимается решение об исправности обмоток двигателя и корректора коэффициента мощности. Модуль может определить как обрыв обмоток двигателя, так и их короткое замыкание. Контроль ККМ выполняется выдачей на все три фазы постоянного напряжения.

Контрольные импульсы а также сигналы тестирования корректора формируются в соответствии с циклограммой(см.рисунок 17)



Рисунок 17 – Циклограмма формирования импульсов тестирования рабочих цепей

Тестирование рабочих цепей можно разбить на 2 условные фазы, отображённые на рисунке 17:

* проверка целостности обмоток двигателя. Здесь формируются контрольные импульсы для каждой фазы, амплитуда которых при нормальном состоянии рабочих цепей соответствует значению номинального тока Iном = 500 мА. Время приращения одной «дискреты» импульса составляет 20 мс, время спада одной «дискреты» импульса составляет 1мс. Период тестирования (время Т2 на рисунке 17) будет составлять около 8 с, паузы между тестированиями двух соседних фаз сигнала составляют 400 мс;
* проверка исправности корректора коэффициента мощности (ККМ). При этом на всех 3 фазах формируется одновременно постоянное напряжение, составляющее 90% от максимального значения и называемое Uккм. Далее производится усреднение полученного значения по 32 отсчётам сигнала датчиков напряжений фаз. Решение о ненорме этого напряжения и соответственно неисправности ККМ принимается при повторении 3 таких неудачных тестов.

Общий (упрощённый) алгоритм работы модуля соответствует следующей диаграмме деятельности:

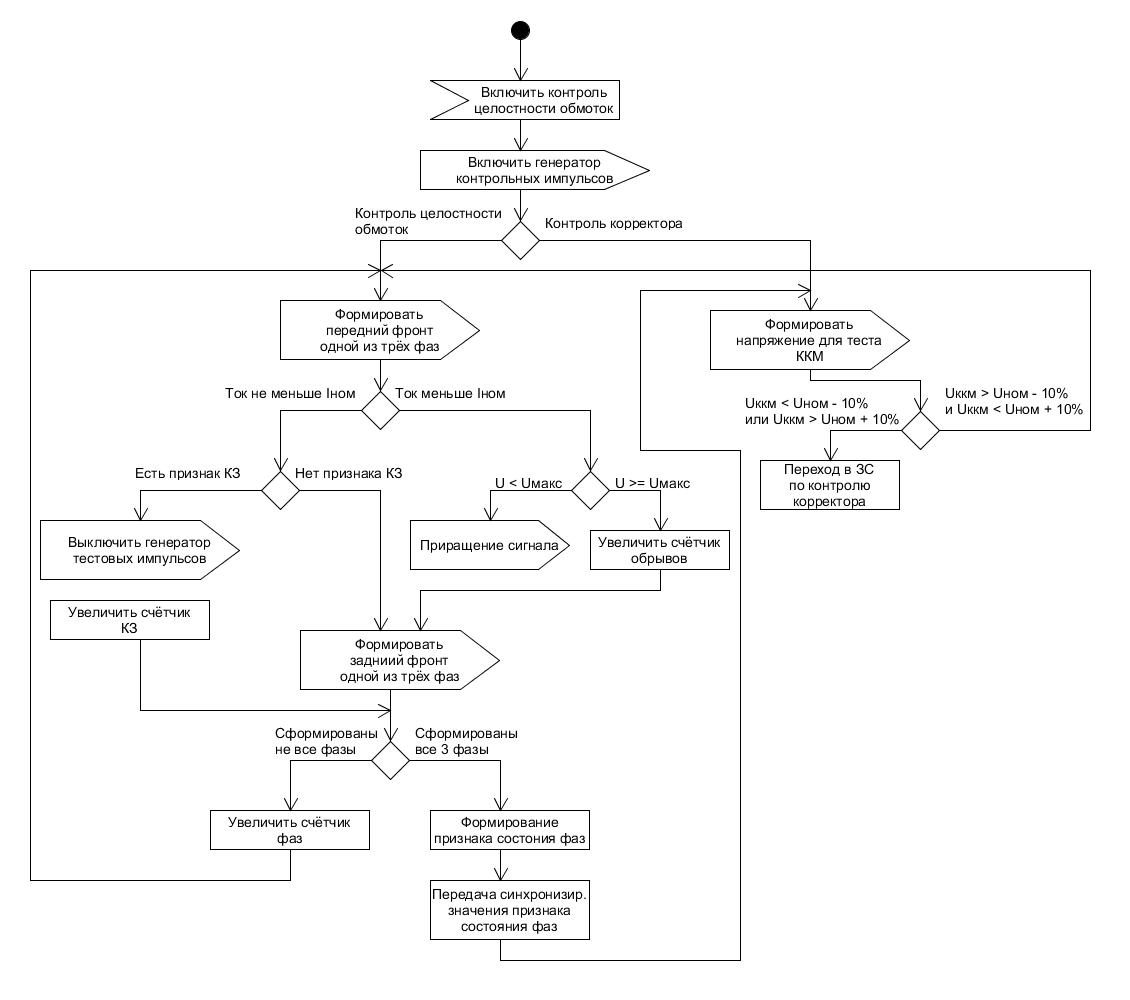


Рисунок 18 – Диаграмма деятельности IntegrCtrl

* + - 1. Модуль IntegrCtrlGen
         1. Назначение

Модуль, генерирующий импульсы специальной формы для контроля целостности обмоток.

* + - * 1. Описание
      1. Порядок применения

1. включить в проект компонент IntegrCtrl;
2. включить в проект драйвер ШИМ трёхфазного генератора (драйвер shuntShiftGenDrv);
3. организовать вызовы конструкторов модулей:
4. ***IntegrCtrlGen\_ctor()*** из ***IntegrCtrl\_ctor()***;
5. ***IntegrCtrl\_ctor()*** при входе в StateMain после успешной инициализации в StateInitial.
6. организовать в главном потоке вызов функций:
7. ***IntegrCtrl\_run()***;
8. ***IntegrCtrlGen\_run()***.
   * 1. Компонент InterChannel
        1. Назначение

МКО предназначен для выполнения синхронизации данных двух МК. Данные, которые синхронизируются при помощи МКО, будем называть параметрами.

В двух МК функционирует ПО, которое выполняет одинаковые алгоритмы. При этом на входах МК, из-за разбросов параметров элементной базы прибора, те или иные события могут наступать в разные моменты времени. Несмотря на это, оба МК должны выполнять одинаковую программу по одинаковым алгоритмам. Модуль МКО и предназначен для синхронизации событий и значений входных данных, результатов обработки и состояния ПО в двух вычислителях.

Программные компоненты, обрабатывающие входные данные, должны обращаться к МКО с запросом на синхронизацию данных и дожидаться от МКО ответа о готовности их синхронизации. Компонент, получив ответ, продолжит выполнение алгоритма, используя синхронизированное значение. Результатом синхронизации (данными) могут пользоваться и другие компоненты ПО, не отправлявшие этот запрос.

Программные компоненты, отправившие запрос на синхронизацию, условно называются инициаторами. Программные компоненты, ожидающие от МКО результатов синхронизации, условно называются клиентами.

* + - 1. Состав компонента

В состав компонента входят следующие модули:

* *InterChannel* − МКО;
* *log2* − определение таблицы Log2\_log2Lkup[];
* *pwr2* − определение таблиц Pwr2\_pwr2Lkup[], Pwr2\_invPwr2Lkup[], and Pwr2\_div8Lkup[];
* *set* − очередь с приоритетом, состоящая из 8 или 64 элементов.
  + - 1. Интерфейс
         1. Константы

В модуле определены следующие константы: *InterChannelScript*, *InterChannelProcSync* и *InterChannelProcCheck*.

Константы типа *InterChannelScript* определяют сценарий синхронизации, используются при настройке параметра и имеют следующие значения:

* *eScriptNil* − сценарий не определен (такое значение приведет к отказу при синхронизации);
* *eScriptSync* − синхронизация параметра;
* *eScriptTransmit* − передача параметра;
* *eScriptLocal* – сформировать синхронизированный параметр без отправки в соседний канал;
* *eScriptDebug* − передача параметра без контроля и синхронизации;
* *eScriptChVal* − синхронизация параметра по изменению значения;
* *eScriptCount* − количество сценариев синхронизации.

Константы типа *InterChannelProcSync* определяют процедуру синхронизации, используются при настройке параметра и имеют следующие значения:

* *eProcSyncNil* − не выполняется никаких действий с параметром;
* *eProcSyncOff* − нет синхронизации (в качестве синхронизируемых данных возвращается значение своего канала);
* *eProcSyncHi* − выбор большего значения;
* *eProcSyncLo* − выбор меньшего значения;
* *eProcSyncAverage* − выбор арифметически среднего значения;
* *eProcSyncAND* − выбор по «AND»;
* *eProcSyncOR* − выбор по «OR»;
* *eProcSyncEqual* – обновление значений параметров, если они совпадают.

Константы типа *InterChannelProcCheck* определяют процедуру контроля данных, используются при настройке параметра и имеют следующие значения:

* *eProcCheckNil* − не выполняется никаких действий с параметром;
* *eProcCheckOff* − нет контроля (применяется при регулировке аппаратуры или передаче параметра);
* *eProcCheckEqual* − абсолютное совпадение данных;
* *eProcCheckMask* − совпадение данных по маске;
* *eProcCheckDeltaX* − абсолютная разность находится в пределах «deltaХ» (значение);
* *eProcCheckDeltaP* − абсолютная разность находится в пределах «deltaP» (процент).

Константы типа *InterChannelCommPos* определяют позиции параметра в физическом протоколе сообщения, передаваемого между каналами, используются при формировании пакета для передачи и при распаковке принятого пакета. Имеют следующие значения:

* *eInterChannelCommPos0* − первый параметр передается вместе с идентификатором, идентификатор может быть любым;
* *eInterChannelCommPos1* − второй параметр передается вместе с идентификатором, идентификатор может быть любым.
  + - 1. Описание
         1. Общие сведения

Для синхронизации значений обоих каналов, МКО должен выполнять следующие функции:

* обработку запросов на синхронизацию;
* обмен данными между МК;
* формирование синхронизированного значения;
* контроль расхождения значений передаваемых данных;
* формирование признака готовности синхронизации;
* контроль временных характеристик выполнения процесса синхронизации.
  + - * 1. Обработка запросов на синхронизацию

Обработка запросов на синхронизацию заключается в том, чтобы, получив запрос на синхронизацию данных, поместить эти данные в приоритетную очередь запросов. Каждый синхронизируемый параметр должен иметь свой уникальный идентификатор, который также должен являться уровнем приоритета в очереди. В дальнейшем эти данные будут использоваться для осуществления обмена данными между МК. Запрос считается обработанным, когда был обмен данными между МК, выполнен контроль расхождения значений передаваемых данных, сформировано синхронизированное значение и сформирован признак готовности синхронизации.

Каждый синхронизируемый параметр должен иметь свой идентификатор. Перечень идентификаторов должен быть определен в интерфейсном модуле InterChannelId.

Каждый синхронизируемый параметр предварительно должен быть настроен. Для этого МКО должны быть заданы (см. 4.3.14.4.8):

* сценарий синхронизации *InterChannelScript*;
* процедура синхронизации *InterChannelProcSync*;
* процедура контроля *InterChannelProcCheck*;
* временные параметры Т1, Т2, Т3.

Запросы на синхронизацию параметров, настройка которых не производилась, должны восприниматься как ошибки ПО.

* + - * 1. Обмен данными между МК

Обмен данными между МК выполняет следующие подфункции:

* инициализация передачи данных в соседний канал;
* прием подтверждения приема данных соседним каналом;
* прием данных от соседнего канала;
* формирование подтверждения приема данных для соседнего канала;
* инициализация повторной передачи данных при отсутствии подтверждения;
* управление модулем CAN МК.

МКО для обмена данными между МК может использовать любой физический интерфейс работа с которым реализована при помощи драйвера, поддерживающего интерфейс ArrayIoDriver (см. **Ошибка! Источник ссылки не найден.**(iodrv)). В этом проекте для передачи данных используется шина CAN и два порта дискретного ввода/вывода. Для управления этими модулями в проекте должен быть разработан модуль dsPIC33CanAck\_driver (см. **Ошибка! Источник ссылки не найден.Ошибка! Источник ссылки не найден.**).

Работа с драйвером осуществляется при помощи функций ***InterChannel\_runCommunication()*** и ***InterChannel\_runDrv()***.

Инициализация передачи данных в соседний канал

Инициализация передачи данных в соседний канал заключается в извлечении данных из очереди в порядке их приоритетов и передаче их в соседний канал при готовности передатчика.

Прием подтверждения приема данных соседним каналом

Прием подтверждения приема данных соседним каналом заключается в опросе передатчика модуля CAN и порта дискретного ввода. После приема данных соседним МК уровень на этом входе должен измениться на противоположный. Прием данных соседним МК считается выполненным, когда передатчик модуля CAN сообщил об успешной передаче и дискретный порт изменил уровень на противоположный.

Прием данных от соседнего канала

Прием данных от соседнего канала заключается в опросе приемника модуля CAN и чтении принятых данных при формировании модулем соответствующего признака.

Формирование подтверждения приема данных для соседнего канала

Формирование подтверждения приема данных для соседнего канала заключается в том, чтобы изменить состояние дискретного выхода на противоположное, когда ПО зафиксировало признак того, что модуль CAN успешно осуществил прием данных.

Инициализация повторной передачи данных при отсутствии подтверждения

В случае отсутствия факта подтверждения приема соседним каналом или факта завершения передачи передатчиком на протяжении 8 циклов ПВС, должна выполниться повторная передача данных. Если после того, как было произведено 6 попыток передать данные, а подтверждение приема получено не было – прибор должен быть переведен в ЗС.

Функция управления модулем CAN МК

Управление модулем CAN должно осуществляться при помощи модуля dsPIC33CanAck\_driver (см. **Ошибка! Источник ссылки не найден.Ошибка! Источник ссылки не найден.**).

Функция управления модулем CAN МК фиксирует состояние шины CAN, факт наличия новых данных в приемнике модуля CAN, факт завершения передачи передатчиком модуля CAN и факт наличия подтверждения приема соседним каналом по линиям дискретного ввода/вывода. В случае если шина CAN переходит в пассивное состояние, данная функция выполняет повторную инициализацию модуля CAN для вывода шины из этого состояния. В случае фиксации на шине ошибок приема, модуль приводится в состояние, готовое к приему новых данных. Зафиксированные факты наличия новых данных в приемнике модуля CAN, завершение передачи передатчиком модуля CAN и наличие подтверждения приема соседним каналом по линиям дискретного ввода/вывода используются для выполнения функций приема подтверждения приема данных соседним каналом, инициализации повторной передачи и приема данных от соседнего канала.

* + - * 1. Формирование синхронизированного значения

Получив значения параметра своего и соседнего МК, МКО должен сформировать синхронизированное значение в соответствии со следующими процедурами синхронизации:

* нет синхронизации (*eProcSyncOff*) – в качестве синхронизируемых данных возвращается значение своего канала;
* выбор большего значения (*eProcSyncHi*) – за синхронизируемое значение принимается большее значение;
* выбор меньшего значения (*eProcSyncLo*) – за синхронизируемое значение принимается меньшее значение;
* выбор арифметического среднего (*eProcSyncAverage*) – за синхронизируемое значение принимается арифметическое среднее значений своего и соседнего МК;
* выбор по «AND» (*eProcSyncAND*) – за синхронизируемое значение принимается значение, полученное в результате выполнения операции «И» над значениями своего и соседнего МК;
* выбор по «OR» (*eProcSyncOR*) – за синхронизируемое значение принимается значение, полученное в результате выполнения операции «ИЛИ» над значениями своего и соседнего МК;
* совпадение значений (*eProcSyncEqual*) – формировать синхронизируемое значение только при совпадении значений в обоих каналах.
  + - * 1. Контроль расхождения значений передаваемых данных

При выполнении синхронизации параметров МКО должен осуществлять контроль значений параметров при помощи следующих процедур контроля:

* нет контроля (*eProcCheckOff*) – применяется при регулировке аппаратуры или передаче параметра;
* совпадение данных абсолютное (*eProcCheckEqual*) – данные в обоих каналах должны совпадать полностью;
* совпадение данных по маске (*eProcCheckMask*) – при булевом сравнении данных должны совпадать биты, которые заданы в маске;
* абсолютная разность находится в пределах deltaХ (значение) (*eProcCheckDeltaX*) – абсолютная разность между значениями своего и соседнего канала не должна превышать заданное значение;
* абсолютная разность находится в пределах deltaP (процент) (*eProcCheckDeltaP*) – абсолютная разность между значениями своего и соседнего канала не должна превышать заданное в процентах значение.

Примечание

Абсолютная разность в процентах определяется от измеренного значения. Выбирается большее значение, которое принимается за 100%, и от него считается расхождение значений. Если измеренное значение меньше значения указанного при настройке параметра на синхронизацию функцией ***InterChannel\_setParamSettings()*** в параметре *middleRangeCheck*, то за 100% принимается это значение (см. 4.4.5.6 документа «Архитектура ПО»).

* + - * 1. Формирование признака готовности синхронизации

Признак готовности синхронизации должен формироваться в обоих МК одновременно в начале нового цикла ГП после того, как были выполнены сценарий синхронизации, процедуры синхронизации и контроля.

* + - * 1. Контроль временных характеристик выполнения процесса синхронизации

Модуль МКО в процессе синхронизации должен выполнять контроль следующих временных параметров:

* время получения параметра от обоих каналов (Т1), мс;
* время формирования синхронизированного параметра (Т2), мс;
* период поступления запросов на синхронизацию (Т3), мс.

Контроль каждого из времен может быть выключен если установить значение контролируемого времени равным нулю.

Единица контроля времени соответствует периоду вызова функции ***InterChannel\_run()***. При вызове функции раз в миллисекунду размерность единицы контроля соответствует миллисекунде.

Время получения параметра от обоих каналов (Т1)

Время получения параметра от обоих каналов предназначено для контроля выполнения сценария синхронизации «Синхронизация параметра» (*eScriptSync*) для контроля расхождения во времени событий, сформировавших запросы синхронизации параметра в разных МК. Отсчет времени начинается с того момента как один из МК обратился с запросом синхронизации параметра к МКО. Останавливается отсчет времени при поступлении запроса на синхронизацию того же параметра от второго МК что является критерием для завершения сценария. Если время достигнет значения заданного пользовательским ПО, прибор должен быть переведен в ЗС с кодом *eGrPS\_InterChannel* группы *ePS\_IntChTime1Error*.

Время формирования синхронизированного параметра (Т2)

Время формирования синхронизированного параметра определяет время на протяжении которого оба МК могут фиксировать значения параметра, которые не проходят процедуру контроля и МКО не формирует синхронизированное значение. Отсчет времени начинается с того момента, как процедура контроля выдала отрицательный результат, а останавливается, когда процедура контроля выдала положительный результат. Если время достигнет значения, заданного пользовательским ПО, прибор должен быть переведен в ЗС с кодом *eGrPS\_InterChannel* группы *ePS\_IntChTime2Error*.

Время Т2 должно задаваться исходя из тех соображений, что для того, чтобы процедура контроля была вызвана повторно необходимо чтобы МК послали очередной запрос синхронизации того параметра, который не прошел эту процедуру в первый раз. То есть, если параметр синхронизируется с фиксированным периодом и, исходя из прикладной задачи, допускается рассогласование значений на протяжении двух периодов, то это время должно быть равным или более двух периодов.

Период поступления запросов на синхронизацию (Т3)

Для параметров, которые должны синхронизировать свои значения с заданным периодом в МКО должна быть предусмотрена возможность контроля периода поступления запросов на синхронизацию значений параметров. Отсчет времени начинается с того момента как в МКО поступит запрос на синхронизацию параметров как от своего так и от соседнего канала. При поступлении очередного запроса счетчик времени сбрасывается. Если время достигнет значения, заданного пользовательским ПО, прибор должен быть переведен в ЗС с кодом *eGrPS\_InterChannel* группы *ePS\_IntChTime3Error*.

* + - * 1. Распределение функций по потокам

В ГП выполняются следующие функции:

* обработка запросов на синхронизацию;
* формирование синхронизированного значения;
* контроль расхождения значений передаваемых данных;
* формирование признака готовности синхронизации;
* контроль временных характеристик выполнения процесса синхронизации.

Подфункции функции обмена данными между МК распределены между ПВС и ПСД.

В ПВС выполняются следующие подфункции:

* управление модулем CAN МК;
* формирование подтверждения приема данных для соседнего канала;
* инициализация повторной передачи данных при отсутствии подтверждения.

В ПСД выполняются следующие подфункции:

* инициализация передачи данных в соседний канал;
* прием подтверждения приема данных соседним каналом;
* прием данных от соседнего канала.

Принцип распределения подфункций по разным потокам обусловлен тем, что ПВС получает управление максимально синхронно в обоих МК, поэтому фиксация событий на линии связи и управление линией передачи данных осуществляется в нем. Однако, поскольку данный поток не может быть заблокирован ГП, доступ к общим данным из ГП и ПВС был бы не безопасным. Поэтому, для безопасной работы с данными между потоками, часть функций выполняется в ПСД, который получает управления строго после того, как закончил работу ПВС и может быть заблокирован ГП при обращении к общим данным.

Поскольку МКО должен работать в трех потоках управления, для каждого из них должны быть реализованы отдельные функции, которые будут вызываться на каждом цикле каждого потока и обеспечивать функционирование модуля. Для работы в ГП должна быть реализована функция ***InterChannel\_run()***, для работы в ПВД – ***InterChannel\_runDrv()***, для работы в ПСД – ***InterChannel\_runCommunication()***.

* + - * 1. Взаимодействие потоков управления

МКО получает управление в начале каждого нового цикла ГП. Все остальные компоненты получают управление после него. Модуль обрабатывает все события, зафиксированные на протяжении предыдущего цикла ГП в ПСД, и на их основании формирует признак завершения синхронизации и синхронизированные значения параметров.

Синхронизация параметра осуществляется в три этапа:

* получение запроса и передача данных в соседний МК;
* прием подтверждения приема от соседнего МК;
* прием данных от соседнего МК.

Примечание

В зависимости от разновидностей сценариев синхронизации некоторые этапы могут исключаться.

Получение запроса и передача данных в соседний МК

На этом этапе выполняется функция «Обработка запросов на синхронизацию» и подфункция «Инициализация передачи данных в соседний канал». Получение запроса на синхронизацию осуществляется в ГП, после чего значение синхронизируемого параметра и его идентификатор помещаются в приоритетную очередь. С приоритетной очередью работа ведется из двух потоков: ГП и ПСД. Безопасный доступ к очереди осуществляется при помощи механизма блокировки ПСД. В ПСД данные извлекаются из очереди и передаются драйверу модуля CAN, который уже в ПВС инициализирует передачу данных в соседний канал. Схема взаимодействия между потоками приведена на рисунке 19.

Передача значения нового параметра возможна только тогда, когда передатчик модуля CAN завершил предыдущую передачу.



Рисунок 19 – Схема взаимодействия между потоками при получении запроса и передача данных в соседний МК

Прием подтверждения приема от соседнего МК

На этом этапе выполняются подфункции «Прием подтверждения приема данных соседним каналом», «Инициализация повторной передачи данных при отсутствии подтверждения», «Управление модулем CAN МК».

В ПВС непрерывно выполняется подфункция «Управление модулем CAN МК», которая фиксирует факт завершения передачи передатчиком модуля CAN и факт наличия подтверждения приема соседним каналом по линиям дискретного ввода/вывода. При фиксации этих двух фактов формируется обобщенный признак окончания передачи данных в соседний канал. При отсутствии одного из событий на протяжении заданного времени выполняется подфункция «Инициализация повторной передачи данных при отсутствии подтверждения».

Признак окончания передачи данных в соседний канал в ПСД помещается в буфер для обработки в ГП. Сбрасывается данный признак при инициализации передачи новых данных. Буфер между ГП и ПСД предназначен для безопасного обмена данными между этими потоками при помощи механизма блокировки ПСД.

В ГП признак окончания передачи данных в соседний канал извлекается из буфера и используется для инициализации выполнения функций «Контроль расхождения значений передаваемых данных», «Формирование синхронизированного значения», «Формирование признака готовности синхронизации» в зависимости от сценария синхронизации. Схема взаимодействия функций между потоками приведена на рисунке 20.



Рисунок 20 – Схема взаимодействия между потоками при приеме подтверждения приема от соседнего МК

Прием данных от соседнего МК

На этом этапе выполняются подфункции «Управление модулем CAN МК» и «Прием данных от соседнего канала».

В ПВС непрерывно выполняется подфункция «Управление модулем CAN МК», которая фиксирует факт наличия новых данных в приемнике модуля CAN и формирует соответствующий признак. В ПСД при наличии данного признака выполняется чтение принятых данных, которые помещаются в буфер для обработки в ГП. После чтения данных признак сбрасывается. Буфер между ГП и ПСД предназначен для безопасного обмена данными между этими потоками при помощи механизма блокировки ПСД.

В ГП принятые данные извлекаются из буфера и используются при выполнении функций «Контроль расхождения значений передаваемых данных», «Формирование синхронизированного значения», «Формирование признака готовности синхронизации» в зависимости от сценария синхронизации. Схема взаимодействия функций между потоками приведена на рисунке 21.



Рисунок 21 – Схема взаимодействия между потоками при приеме данных от соседнего МК

* + - * 1. Обеспечение ограничения на передачу данных

Для обеспечения ограничения на передачу данных перед началом нового цикла пользовательское ПО должно сообщить может ли МКО выдавать данные на шину или нет. Для этого при вызове функции ***InterChannel\_runCommunication()*** ей в качестве параметра необходимо передать разрешение на передачу данных. Схема работы ограничения на передачу данных приведена на рисунке 22.



Рисунок 22 – Схема работы ограничения на передачу данных

Задача ограничения передачи данных заключается в том, чтобы исключить формирование событий приема данных и подтверждения передачи во время работы МКО в ГП, которое осуществляется в начале каждого цикла ГП. Исходя из того, что передача данных по шине CAN в обе стороны занимает четыре прерывания, пользовательское или системное ПО должно запретить передачу если до завершения ГП осталось 5 прерываний. Таким образом, в начале цикла ГП не будут формироваться события приема данных и подтверждения передачи аж до четвертого прерывания следующего цикла. За это время МКО должен завершить обработку данных. На рисунке 22 приведена диаграмма при плотной загрузке канала, когда новые данные передаются сразу же как только закончился обмен предыдущими. При слабой загрузке, когда запросы на МКО поступают редко и данные передаются по мере их поступления, во всех случаях, последний обмен данными заканчивается до начала нового цикла, с запасом в одно прерывание.

* + - * 1. Сценарии синхронизации

Каждому параметру должен соответствовать свой уникальный идентификатор, который определен в интерфейсном системном модуле InterChannelId (см. **Ошибка! Источник ссылки не найден.**) как часть перечисляемого типа *InterChannelId* и должен быть расположен между значениями *eInterChannelIdBegin* и *eInterChannelIdCount*. Значение *eInterChannelIdCount* определяет общее количество синхронизируемых параметров.

Примечание

Не допускается существование двух идентификаторов с одинаковыми значениями.

Сценарий «Сценарий не определен»

Данный сценарий должен быть задан при создании компонента МКО. При попытке синхронизации параметра с таким сценарием прибор должен быть переведен в ЗС.

Сценарий «Синхронизация параметра»

Предусловия:

1. проведена настройка сценариев МКО, процедур синхронизации и контроля данных.

Постусловия:

1. клиенты обработали новое синхронизированное значение.

Сценарий:

1. инициатор отправляет запрос на проведение синхронизации параметра в МКО;
2. формируется признак выполнения сценария синхронизации;
3. параметр передается в соседний канал;
4. осуществляется прием параметра из соседнего канала;
5. в соответствии с выбранной процедурой контроля производится контроль значений параметров, полученных от своего и соседнего каналов;
6. в соответствии с выбранной процедурой синхронизации формируется синхронизированное значение;
7. формируется признак завершения синхронизации МКО;
8. снимается признак выполнения сценария синхронизации;
9. клиенты обрабатывают синхронизированное значение.

Диаграмма последовательности работы МКО, описывающая основной и альтернативный сценарий, приведена на рисунке 23.



Рисунок 23 ‒ Диаграмма последовательности работы МКО

Альтернативный сценарий:

1. МКО принимает от соседнего канала параметр, требующий синхронизации;
2. МКО ожидает запрос на синхронизацию от инициатора;
3. инициатор отправляет запрос на проведение синхронизации параметра в МКО;
4. в МКО формируется признак выполнения сценария синхронизации;
5. МКО передает параметр в соседний канал;
6. в соответствии с выбранной процедурой контроля МКО производит контроль значений параметров, полученных от своего и соседнего каналов;
7. в соответствии с выбранной процедурой синхронизации определяется значение синхронизированного параметра;
8. формируется признак завершения синхронизации МКО;
9. снимается признак выполнения сценария синхронизации;
10. клиенты обрабатывают синхронизированное значение.

Исключения:

1. истекло время Т1 – переход в ЗС;
2. процедура контроля выполнилась с ошибкой – сценарий обработки ошибки процедуры контроля;
3. истекло время Т3 – переход в ЗС;
4. не была проведена настройка параметра запрос на синхронизацию или передачу которого поступил в МКО – переход в ЗС;
5. поступил запрос на синхронизацию уже обрабатываемого параметра – переход в ЗС.

Сценарий обработки ошибки процедуры контроля:

1. инициатор отправляет запрос на проведение синхронизации параметра в МКО;
2. формируется признак выполнения сценария синхронизации;
3. параметр передается в соседний канал;
4. осуществляется прием параметра из соседнего канала;
5. в соответствии с выбранной процедурой контроля, производится контроль значений параметров, полученных от своего и соседнего каналов;
6. процедура контроля определяет ошибку;
7. снимается признак выполнения сценария синхронизации;
8. запускается отсчет Т2;
9. инициатор отправляет новый запрос на проведение синхронизации параметра в МКО;
10. формируется признак выполнения сценария синхронизации;
11. параметр передается в соседний канал;
12. осуществляется прием параметра из соседнего канала;
13. в соответствии с выбранной процедурой контроля, производится контроль значений параметров, полученных от своего и соседнего каналов;
14. в соответствии с выбранной процедурой синхронизации, формируется синхронизированное значение;
15. формируется признак завершения синхронизации МКО;
16. снимается признак выполнения сценария синхронизации;
17. клиенты обрабатывают синхронизированное значение.

Исключения:

1. истекло время Т1 – переход в ЗС;
2. истекло время Т2 – переход в ЗС;
3. истекло время Т3 – переход в ЗС;
4. не была проведена настройка параметра запрос на синхронизацию или передачу которого поступил в МКО – переход в ЗС;
5. поступил запрос на синхронизацию уже обрабатываемого параметра – переход в ЗС.

Диаграмма последовательности работы МКО, описывающая сценарий ошибки процедуры контроля, приведена на рисунке 24.



Рисунок 24 ‒ Диаграмма последовательности работы МКО

Сценарий «Синхронизация параметра по изменению значения»

Сценарий синхронизации «Синхронизация по изменению значения» предназначен для синхронизации непериодических сигналов. Данный сценарий не выполняет синхронизацию событий, а только ликвидирует различия между значениями параметра в двух МК. Отличие от обычной синхронизации заключается в том, что МКО не ожидает от соседнего канала синхронизируемого значения, а выполняет процедуры контроля и синхронизации сразу после успешной передачи, используя значение соседнего канала, полученное при предыдущих синхронизациях.

При инициализации МКО все значения проинициализированы нулями. При поступлении новых значений, значения сравниваются с ними.

Основной сценарий

Предусловия:

1. проведена настройка сценариев МКО, процедур синхронизации и контроля данных.

Постусловия:

1. клиенты обработали новое синхронизированное значение.

Сценарий:

1. инициатор своего канала запрашивает синхронизацию параметра;
2. МКО своего канала формируется признак выполнения сценария синхронизации;
3. передает параметр в соседний канал;
4. получает подтверждение приема;
5. выполняет процедуру контроля, используя свое новое значение и значение соседнего канала, полученное ранее;
6. выполняет процедуру синхронизации;
7. устанавливает признак наличия синхронизированных данных;
8. сбрасывает признак выполнения сценария синхронизации;
9. клиенты обрабатывают синхронизированное значение.

Альтернативный сценарий:

1. МКО принимает от соседнего канала параметр, требующий синхронизации;
2. выполняет процедуру контроля, используя новое значение соседнего канала и своего, полученное при последнем запросе на синхронизацию;
3. выполняет процедуры синхронизации;
4. устанавливает признак наличия синхронизированных данных;
5. клиенты обрабатывают синхронизированное значение.

Исключения:

1. процедура контроля выполнилась с ошибкой – основной или альтернативный сценарий обработки ошибки процедуры контроля;
2. истекло время Т3 – переход в ЗС;
3. не была проведена настройка параметра запрос на синхронизацию или передачу которого поступил в МКО – переход в ЗС;
4. поступил запрос на синхронизацию уже обрабатываемого параметра – переход в ЗС.

Основной сценарий обработки ошибки процедуры контроля:

1. инициатор своего канала запрашивает синхронизацию параметра;
2. формируется признак выполнения сценария синхронизации;
3. получает подтверждение приема;
4. выполняет процедуру контроля, используя свое новое значение и значение соседнего канала, полученное ранее;
5. процедура контроля определяет ошибку;
6. сбрасывает признак выполнения сценария синхронизации;
7. параметр передается в соседний канал;
8. осуществляется прием параметра из соседнего канала;
9. в соответствии с выбранной процедурой контроля, производится контроль значений параметров, полученных от своего и соседнего каналов;
10. процедура контроля определяет ошибку;
11. снимается признак выполнения сценария синхронизации;
12. запускается отсчет Т2;
13. инициатор отправляет новый запрос синхронизации;
14. формируется признак выполнения сценария синхронизации;
15. параметр передается в соседний канал;
16. осуществляется прием параметра из соседнего канала;
17. в соответствии с выбранной процедурой контроля, производится контроль значений параметров, полученных от своего и соседнего каналов;
18. формируется признак завершения синхронизации МКО;
19. снимается признак выполнения сценария синхронизации;
20. клиенты обрабатывают синхронизированное значение.

Альтернативный сценарий обработки ошибки процедуры контроля:

1. МКО принимает от соседнего канала параметр, требующий синхронизации;
2. МКО выполняет процедуру контроля, используя новое значение соседнего канала и своего, полученное при последнем запросе на синхронизацию;
3. МКО выполняет процедуру синхронизации;
4. устанавливает признак наличия синхронизированных данных;
5. клиенты обрабатывают синхронизированное значение.

Исключения:

1. истекло время Т2 – переход в ЗС;
2. истекло время Т3 – переход в ЗС;
3. не была проведена настройка параметра запрос на синхронизацию или передачу которого поступил в МКО – переход в ЗС;
4. поступил запрос на синхронизацию уже обрабатываемого параметра – переход в ЗС.

Дополнительные сведения

После того как процедура контроля выдала ошибку и запустился подсчет Т2, дальнейшая ситуация может развиваться как по основному, так и по альтернативному сценарию. На рисунке 25 изображена диаграмма последовательности на которой в первом канале отработал основной сценарий, а во втором – альтернативный, при этом процедура контроля обнаружила ошибку и был запущен счет времени Т2. После этого основной сценарий отработал во втором канале, а альтернативный в первом. При этом процедура контроля выполнилась без ошибки и были сформированы синхронизированные данные, а счет Т2 остановлен. Аналогичная ситуация была бы если бы после запуска Т2 в первом канале отработал основной сценарий, а во втором альтернативный и при этом процедура контроля завершилась бы без ошибки.



Рисунок 25 ‒ Диаграмма последовательности работы МКО

Сценарий «Передача параметра»

Предусловие:

1. проведена настройка сценариев МКО, процедур синхронизации и контроля данных.

Постусловие:

1. клиенты обработали новое синхронизированное значение.

Основной сценарий:

1. инициатор отправляет запрос на проведение передачи параметра в МКО;
2. формируется признак выполнения сценария синхронизации;
3. параметр передается в соседний канал;
4. осуществляется прием подтверждения из соседнего канала;
5. значение синхронизированного параметра принимается равным передаваемому значению;
6. устанавливает признак наличия синхронизированных данных
7. снимается признак выполнения сценария синхронизации;
8. клиенты обрабатывают синхронизированное значение.

Альтернативный сценарий:

1. МКО принимает параметр от соседнего канала;
2. значение синхронизированного параметра принимается равным принятому значению;
3. устанавливает признак наличия синхронизированных данных;
4. клиенты обрабатывают синхронизированное значение.

Исключения:

1. истекло время Т3 – переход в ЗС;
2. не была проведена настройка параметра запрос на синхронизацию или передачу которого поступил в МКО – переход в ЗС;
3. поступил запрос на передачу уже обрабатываемого параметра – переход в ЗС.

Схема взаимодействия при передаче параметра соседнему каналу отображена на рисунке 26.



Рисунок 26 ‒ Схема взаимодействия при передаче параметра соседнему каналу

Сценарий «Сформировать синхронизированный параметр без отправки в соседний канал»

Сценарий:

1. инициатор отправляет запрос на синхронизацию;
2. МКО, без обращения к соседнему каналу, формирует синхронизированное значение на основании значения, принятого в запросе;
3. устанавливает признак наличия синхронизированных данных;
4. снимается признак выполнения сценария синхронизации;
5. клиенты обрабатывают синхронизированное значение.

Сценарий «Передача параметра без контроля и синхронизации»

Сценарий:

1. инициатор отправляет запрос на синхронизацию;
2. МКО передает его в соседний канал;
3. признак наличия синхронизированных данных не устанавливается никогда.
   * + - 1. Организация работы модуля

Все данные, синхронизируемые или передаваемые при помощи этого модуля, должны иметь тип *param\_t* и каждому из них должен быть определен идентификатор типа *id\_t*.

Для обмена данными между МК (см. 4.3.14) должен быть использован драйвер ArrayIoDriver(см. **Ошибка! Источник ссылки не найден.**(iodrv)).

Основная работа МКО осуществляется в ГП. Однако, работа с периферией МК при помощи которой осуществляется обмен между МК должна осуществляться в ПВС для синхронной фиксации состояний обмена данными и в ПСД – для безопасного обмена данными с ГП. Поскольку МКО должен работать в трех потоках управления, для каждого из них должны быть реализованы отдельные функции, которые будут вызываться на каждом цикле каждого потока обеспечивать функционирование модуля. Для работы в ГП должна быть реализована функция ***InterChannel\_run()***, для работы в ПВД – ***InterChannel\_runDrv()***, для работы в ПСД – ***InterChannel\_runCommunication()***. Пользовательское ПО, в свою очередь, должно обеспечить вызов этих функций на каждом цикле соответствующего потока.

Для того чтобы обеспечить ограничение на передачу данных перед началом нового цикла, пользовательское ПО должно сообщить может ли МКО выдавать данные на шину или нет. Для этого при вызове функции ***InterChannel\_runCommunication()*** ей в качестве параметра необходимо передать разрешение на передачу данных.

Поскольку функции МКО работают с общими данными из разных потоков, доступ к этим данным должен быть защищен. Для обеспечения приостановки ПСД на время обращения к ним из ГП в модуле MainRegisters должны быть определены следующие макроопределения:

* CAN\_TIMER\_DISABLE\_INTERRUPT;
* CAN\_TIMER\_IS\_ENABLE\_INTERRUPT;
* CAN\_TIMER\_SET\_ENABLE\_INTERRUPT( en ).
  + - * 1. Настройка модуля

Настройка идентификаторов

Все идентификаторы, которые будут синхронизироваться при помощи модуля МКО, должны быть описаны в модуле InterChannelId. Этот модуль предназначен для настройки МКО для работы с заданным количеством идентификаторов для каждого конкретного проекта, поэтому расположен в системном каталоге systems. Для определения идентификатора необходимо определить его значение в перечисляемом типе *InterChannelId*. Все идентификаторы должны быть добавлены до значения *eInterChannelIdCount*, которое используется модулем для определения количества параметров. Максимально возможное количество параметров ограничено разрядностью множества *Set64\_t* и составляет 64 параметра.

Идентификаторы также определяют приоритет параметра в очереди на синхронизацию: чем больше значение идентификатора, тем выше его приоритет в очереди. Сначала будут передаваться параметры с наивысшим приоритетом, параметры с низким приоритетом будут переданы по мере опустошения очереди.

Настройка блокировки доступа к данным

В модуле MainRegisters должны быть определены макроопределения так как описано в **Ошибка! Источник ссылки не найден.**:

* CAN\_TIMER\_DISABLE\_INTERRUPT;
* CAN\_TIMER\_IS\_ENABLE\_INTERRUPT;
* CAN\_TIMER\_SET\_ENABLE\_INTERRUPT( en ).

Настройка драйвера

Перед использованием модуля МКО, должен быть проинициализирован драйвер, используемый для обмена данными между МК. В данном проекте это dsPIC33CanAck\_driver, а функция ***dsPIC33CanAck\_ctor()***. Функция инициализации этого драйвера должна вернуть указатель на структуру типа *ArrayIoDriver*.

После чего должна быть вызвана функция инициализации ***InterChannel\_ctor()***, которой в качестве параметра должна быть передана структура, которую вернул драйвер. После инициализации МКО должен быть готов работе.

Инициализация параметров

Каждый параметр, который будет синхронизироваться при помощи модуля МКО, должен быть предварительно инициализирован при помощи функции ***InterChannel\_setParamSettings()***. Если к модулю обратиться с запросом на синхронизацию параметра, который был не инициализирован, произойдет переход в ЗС с кодом *eInterChannelScript.*

* + - * 1. Использование модуля

Работа функций ***InterChannel\_runDrv(), InterChannel\_runCommunication()*** и ***InterChannel\_run()*** должна быть организована как описано в 4.3.14, а также выполнены все действия, описанные в 4.3.14.4.

Для того, чтобы отправить параметр на синхронизацию, необходимо вызвать функцию ***InterChannel\_synchronize()***, которой в качестве параметра необходимо передать идентификатор и значение синхронизируемого параметра.

Модуль МКО одновременно может обрабатывать только один запрос для одного параметра. Если вызвать ***InterChannel\_synchronize()*** для параметра, для которого предыдущий запрос не был обработан, произойдет переход в ЗС с кодом *ePS\_IntChRepeat* группы *eGrPS\_InterChannel*.

Узнать о том, что обработка запроса на синхронизацию завершилась можно опросив функцию ***InterChannel\_isHandling()***. Функция при возвращении истины сигнализирует о том, что параметр с указанным идентификатором находится в обработке.

Узнать о том, что синхронизация параметра завершилась можно опросив функцию ***InterChannel\_isSynchronized()***. Функция при возвращении истины сигнализирует о том, что параметр был успешно синхронизирован.

Обработка параметра может быть завершена, но при этом синхронизация параметра может быть не выполнена. Такая ситуация возможна, когда запрос на синхронизацию от соседнего МК получен, но значения своего и соседнего параметра не прошли верификацию. В этом случае можно еще раз послать параметр на синхронизацию.

Прочитать синхронизированные значения можно при помощи функции ***InterChannel\_getData()***. Функция возвращает последнее синхронизированное значение кроме случая, когда не было выполнено ни одной синхронизации. В этом случае функция возвращает ноль.

* + - 1. Модуль log2
         1. Назначение

Модуль содержит таблицу *Log2\_log2Lkup[]*, которая используется модулем set для расчета табличным способом логарифма по основанию 2.

* + - * 1. Интерфейсы

Интерфейс доступа к таблице *Log2\_log2Lkup* описан в модуле set как *extern uint8\_t const Log2\_log2Lkup[256]*.

* + - 1. Модуль pwr2
         1. Назначение

Модуль содержит определение таблиц *Pwr2\_pwr2Lkup[], Pwr2\_invPwr2Lkup[]*, и *Pwr2\_div8Lkup[]*, которые используется модулем set.

*Pwr2\_pwr2Lkup* – таблица поиска для (1 << ((n-1)% 8)), где n – индекс в таблице. Диапазон индексов n = 0..64. Первый индекс (n == 0) никогда не должен использоваться.

*Pwr2\_invPwr2Lkup* – таблица поиска для ~(1 << ((n-1) % 8)), где n – индекс в таблице. Диапазон индексов n = 0..64. Первый индекс (n == 0) никогда не должен использоваться.

*Pwr2\_div8Lkup* –таблица поиска для (n-1)/8. Диапазон индексов n = 0..64. Первый индекс (n == 0) никогда не должен использоваться.

* + - * 1. Интерфейсы

Интерфейс доступа к таблицам *Pwr2\_pwr2Lku*p, *Pwr2\_invPwr2Lkup* и *Pwr2\_div8Lkup* описан в модуле set как:

* extern uint8\_t const Pwr2\_pwr2Lkup[65];
* extern uint8\_t const Pwr2\_invPwr2Lkup[65];
* extern uint8\_t const Pwr2\_div8Lkup[65].
  + - 1. Модуль set
         1. Назначение

Очередь с приоритетом, состоящая из 8 или 64 элементов. Позволяет добавлять в нее элементы в произвольном порядке и извлекать их в порядке их приоритетов. Добавление, извлечение и поиск элементов в очереди осуществляется табличным способом.

* + - * 1. Интерфейс

Типы данных

*Set8\_t* – очередь с приоритетом из 8 элементов.

*Set64\_t* – очередь с приоритетом из 64 элементов.

Макроопределения для работы с объектами Set8\_t

* *Set8\_isEmpty(me\_)* – макрос определяется как *true* если очередь *me\_* пуста.

Параметры:

*me\_* – объект очередь типа Set8\_t.

* *Set8\_notEmpty(me\_)* – макрос определяется как *true*, если очередь *me\_* не пуста.

Параметры:

*me\_* – объект очередь типа Set8\_t.

* *Set8\_hasElement(me\_, n\_)* – макрос определяется как *true*, если приоритетная очередь *me\_* содержит элемент указанный в параметре *n\_*.

Параметры:

*me\_* – объект очередь типа Set8\_t;

*n\_* – идентификатор элемента очереди.

* *Set8\_insert(me\_, n\_)* – макрос добавляет элемент, указанный в параметре *n\_* в очередь *me\_*.

Параметры:

*me\_* – объект очередь типа Set8\_t;

*n\_* – идентификатор элемента очереди.

* *Set8\_remove(me\_, n\_)* – макрос удаляет элемент, указанный в параметре *n\_* из очереди *me\_*.

Параметры:

*me\_* – объект очередь типа Set8\_t;

*n\_* – идентификатор элемента очереди.

* *Set8\_findMax(me\_, n\_)* – макрос находит в очереди *me\_* элемент с максимальным приоритетом и сохраняет его идентификатор в параметре *n\_*. Если очередь пуста в параметр *n\_* будет записан идентификатор ноль.

Параметры:

*me\_* – объект очередь типа Set8\_t;

*n\_* – идентификатор найденного элемента очереди.

Макроопределения для работы с объектами Set64\_t

* *Set64\_isEmpty(me\_)* – макрос определяется как *true*, если очередь *me\_* пуста.

Параметры:

*me\_* – объект очередь типа Set64\_t.

* *Set64\_notEmpty(me\_)* – макрос определяется как *true*, если очередь *me\_* не пуста.

Параметры:

*me\_* – объект очередь типа Set64\_t.

* *Set64\_hasElement(me\_, n\_)* – макрос определяется как *true*, если приоритетная очередь *me\_* содержит элемент указанный в параметре *n\_*.

Параметры:

*me\_* – объект очередь типа Set64\_t;

*n\_* – идентификатор элемента очереди.

* *Set64\_insert(me\_, n\_)* – макрос добавляет элемент, указанный в параметре *n\_*, в очередь *me\_*.

Параметры:

*me\_* – объект очередь типа Set64\_t;

*n\_* – идентификатор элемента очереди.

* *Set64\_ remove(me\_, n\_)* – макрос удаляет элемент, указанный в параметре *n\_*, из очередь *me\_*.

Параметры:

*me\_* – объект очередь типа Set64\_t;

*n\_* – идентификатор элемента очереди.

* *Set64\_findMax(me\_, n\_)* – макрос находит в очереди *me\_* элемент с максимальным приоритетом и сохраняет его идентификатор в параметре *n\_*. Если очередь пуста в параметр *n\_* будет записан идентификатор ноль.

Параметры:

*me\_* – объект очередь типа Set64\_t;

*n\_* – идентификатор найденного элемента очереди.

* *Set64\_clear(me\_)* – макрос удаляет все элементы из очереди *me\_*.

Параметры:

*me\_* – объект очередь типа Set64\_t;

* *Set64\_copy(dest\_, source\_)* – макрос копирует все элементы из очереди *source\_* в очередь *dest\_*.

Параметры:

*dest\_*– объект очередь типа Set64\_t в который копируются элементы;

*source\_* – объект очередь типа Set64\_t из которого копируются элементы.

* + - * 1. Описание

Для использования очереди из 8 или 64 элементов необходимо объявить переменную типа *Set8\_t* или *Set64\_t* соответственно. Работа с элементами очереди ведется при помощи их идентификаторов. Каждый элемент должен иметь свой уникальный идентификатор. Идентификатор ноль использовать не допускается. Каждому идентификатору соответствует свой уровень приоритета: чем больше значение идентификатора, тем выше его приоритет в очереди.

Чтобы из очереди извлекать элементы в порядке их приоритета необходимо при помощи макросов *Set8\_findMax* или *Set64\_findMax* найти элемент в очереди с наибольшим значением идентификатора, затем удалить его при помощи макроопределений *Set8\_remove* или *Set64\_remove*. Если найденный элемент имеет значение ноль, значит очередь пуста.

* + 1. Компонент ModeProtection
       1. Назначение

Компонент ModeProtection служит для перевода прибора в 3С и снятия 3С.

* + - 1. Состав компонента

Компонент состоит из одноимённого модуля.

* + - * 1. Описание
        2. Общие сведения

Компонент *ModeProtection* обеспечивает управление переходом прибора в 3C и снятием 3С.

Перевод прибора в 3С

При необходимости перевести прибор в 3С вызывается функция ***ModeProtection\_run()***. Функция выполняет следующие действия:

1. отключает все периферийные устройства МК и переводит выводы МК в Z-состояние.
2. записывает код защитного отказа в EEPROM.
3. сохраняет содержимое буферов памяти «чёрного ящика» из ОЗУ МК в EEPROM.
4. переходит в бесконечный цикл, выход из которого невозможен без выключения питания прибора.

Диаграмма взаимодействия ModeProtection с другими компонентами ПО приведена на рисунке 27.



Рисунок 27 – Диаграмма взаимодействия ModeProtection с другими компонентами ПО при переводе прибора в 3С.

Снятие 3С с прибора

Снятие 3С с прибора является важной функцией обеспечения функциональной безопасности прибора. Для обеспечения достоверности определения должных условий для снятия 3С предусмотрены специальные программно-аппаратные меры, которые будут описаны далее.

Следующие условия должны быть выполнены для снятия 3С:

1. Снятие 3С возможно только в момент его включения (при подаче напряжения питания 24 В). В процессе работы прибора снятие 3С невозможно;
2. На оба МК (Master и Slave) должна быть установлена специальная перемычка (джампер).

При запуске прибора однократно вызывается ***ModeProtection\_ClearPS()***. В ней функция ***checkJumperPS()*** проверяет наличие установленной перемычки. Если функция возвращает значение *true*, то состояние 3C снимается. При этом в ячейки EEPROM, определённые макросами ADDRESS\_EEPROM\_PROTECTION\_CODE\_1…ADDRESS\_EEPROM\_PROTECTION\_CODE\_3 записывается значение PROTECTION\_CODE\_ABSENCE. После этого прибор остаётся в 3С, три раза мигнув индикаторами «RS1» и «RS2». При следующем включении прибора он начнёт нормально функционировать. Если функция возвращает значение *false*, то прибор остаётся в 3С.

Как видно, ключевую роль в снятии 3С играет функция ***checkJumperPS()***, т.к. именно она определяет наличие установленной перемычки снятия 3С. Для обеспечения высокой достоверности определения наличия установленной перемычки приняты аппаратные и программные меры защиты от ложного обнаружения:

1. обнаружение перемычки только один раз при запуске прибора;
2. определение исправности порта МК, который определяет наличие перемычки;
3. многократное определение исправности порта и обнаружения установленной перемычки. Только при всех положительных результатах обнаружения принимается решение о наличии перемычки;
4. перемычки должны быть установлены одновременно как в Master, так и в Slave МК. Только при обнаружении перемычки обоими МК происходит снятие 3С с прибора.

Фрагмент принципиальной схемы, поясняющий алгоритм определения установки перемычки снятия 3С, приведён на Рисунке 28.



Рисунок 28 – Фрагмент принципиальной схемы, поясняющий алгоритм определения установки перемычки снятия 3С.

Эта функция проверяет наличие специальной перемычке на плате (джампера). При наличии перемычки снимается защитное состояние с прибора. Наличие перемычки проверяется по алгоритму, изображённому на Рисунке 29. Исправность порта RB0, работающего как вход, проверяется периодической сменой логического уровня на порту RB1. При наличии перемычки и исправном порте RB0 на входе должна наблюдаться последовательность смены логических уровней. Т.е. состояние на выходе порта RB1 должно отображаться на порту RB1. При отсутствии перемычки или неисправном порту – лог. 1 или лог. 0 будут присутствовать статично. Реализованный алгоритм опроса позволяет диагностировать неисправность порта и определить наличие перемычки снятия 3С.

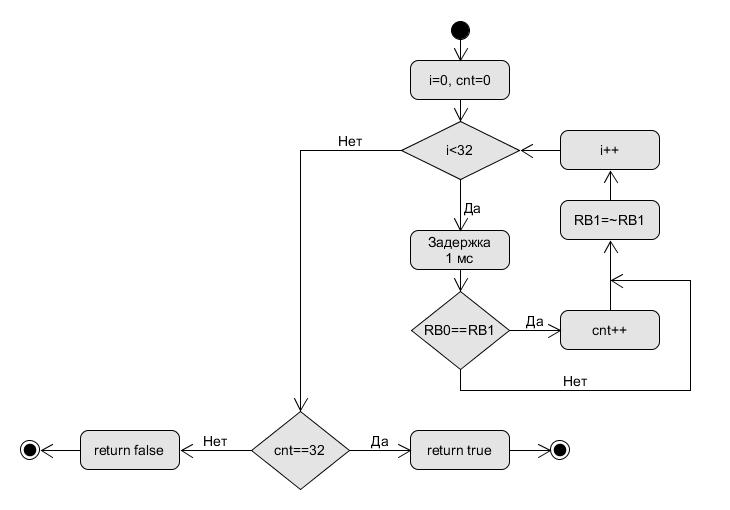


Рисунок – Алгоритм опроса перемычки снятия 3С

* + 1. Компонент OverloadDet
       1. Назначение

Компонент OverloadDet предназначен для обнаружения состояния токовой перегрузки на выходе прибора. Компонент обрабатывает данные с датчика тока в линейных фазах, цепи контрольного генератора и выносит решение о состоянии перегрузки.

* + - 1. Состав компонента

Компонент состоит из одноимённого модуля.

* + - 1. Описание
         1. Общие сведения

Компонент служит для обнаружения состояния токовой перегрузки в линейных цепях управления двигателем, а также в цепи контрольного генератора.

* + - * 1. Принцип работы определения перегрузки рабочих цепей

В соответствии с назначением блока трёхфазного формирователя, изображённого на структурной схеме прибора в «Функциональных требованиях ПО ОКПС-Е-К» для определения перегрузки по току используется сигнал KZ, а сигнал KZ\_DIS используется для сброса триггера-защёлки и последующей работы прибора. Алгоритм определения перегрузки по току рабочих цепей реализовани в виде конечного автомата и имеет следующий вид:

1. при наличии питания 24 В производится чтение сигнала KZ в главном потоке с периодом 1 мс;
2. если этот сигнал равен «1» в течение 4 подряд повторяющихся отсчётов тогда, когда отсутствует выполнение приказа на перевод стредки и отключен контроль целостности рабочих цепей в комноненте IntegrCtrl (например, на пассивном приборе) – прибор переходит в ЗС, так как неисправна схема определения перегрузки по току. Если читается такой же сигнал («1»), но при этом дибо выполняется перевод стрелки, либо работает контроль целостности рабочих цепей, то устанавливаются признак короткого замыкания (флаг *isScThreePhaseGen*) и триггерный флаг *ScThreePhaseGenTrig* (который необходим для корректной выдачи в управляющую систему данных о состоянии рабочих цепей без «промигивания» в моменты перезапуска защёлки триггера схемы защиты) и защёлкивается триггер схемы защиты от короткого замыкания («1» в сигнале KZ\_DIS);
3. по истечении таймаута 8 секунд флаг *isScThreePhaseGen* сбрасывается, также сбрасывается триггер-защёлка схемы защиты от КЗ;
4. повторяются пункты а)-в). Если отсутствуют перечисленные для определения перегрузки по току в пункте б) условия, то флаг *ScThreePhaseGenTrig* сбрасывается и переход к пункту в) осуществляться не будет;
5. флаг перегрузки по току отправляется в синхронизацию с периодом 100 мс.
   * + - 1. Принцип работы определения перегрузки цепей контрольного генератора

В соответствии с назначением блока генератора, изображённого на структурной схеме прибора в «Функциональных требованиях ПО ОКПС-Е-К» для определения перегрузки по току используется сигнал KZ\_GEN. Алгоритм определения перегрузки по току цепей контрольного генератора реализован в виде конечного автомата и имеет следующий вид:

1. при наличии питания 24 В производится чтение сигнала KZ\_GEN в главном потоке с периодом 1 мс;
2. если этот сигнал равен «1» в течение 4 подряд повторяющихся отсчётов тогда, когда контрольный генератор отключен – прибор переходит в ЗС, так как неисправна схема определения перегрузки по току в цепях контрольного генератора. Если читается такой же сигнал («1»), но при этом контрольный генератор включён, то устанавливаются признак короткого замыкания(флаг *isScContrGen*), триггерный флаг *ScContrGenTrig* (который необходим для корректной выдачи в управляющую систему данных о состоянии контрольных цепей без «промигивания» в моменты перезапуска контрольного генератора;
3. по истечении таймаута 5 с флаг *isScThreePhaseGen* сбрасывается;
4. повторяются пункты а)-в). Если отсутствуют перечисленные для определения перегрузки по току в пункте б) условия, то флаг *ScContrGenTrig* сбрасывается и переход к пункту в) осуществляться не будет;
5. флаг перегрузки по току контрольных цепей отправляется в синхронизацию с периодом 100 мс.
   * + 1. Порядок применения
6. включить в проект компонент OverloadDet;
7. организовать вызов конструктора модуля:

* ***OverloadDet\_ctor()*** при входе в StateMain после успешной инициализации в StateInitial.

1. организовать в ГП вызов функции ***OverloadDet\_run()*** в StateTop, включающую в себя статичские функции определения перегрузки в рабочих цепях и определение перегрузки в цепях контрольного генератора (***procScThreePhaseGen\_run()*** и ***procScContrGen\_run()*** соответственно).
   * 1. Компонент PositionDet
        1. Назначение

Компонент предназначен для определения текущего положения стрелочного привода. Он обеспечивает генерацию сигналов для работы контрольного генератора (КГ) и определение положения стрелочного привода по сигналам R1 и R2.

* + - 1. Состав компонента

Компонент состоит из модулей:

1. *positionDet* – модуль определения положения стрелочного привода;
2. *posDetGenerator* – модуль контрольного генератора;
3. *positionDetFiveEC* – модуль определения положения стрелочного привода при пятипроводном подключении (в данной версии ПО не используется);
4. *positionDetNineWire* – модуль определения положения стрелочного привода при девятипроводном подключении;
5. *positionDetTypes –* модуль содержит прототипы функций определения положения.
   * + 1. Описание
          1. Общие сведения

Компонент служит для определения текущего положения стрелочного привода. В данной версии ПО реализовано определение положения стрелочного привода в девятипроводной схеме включения. Функции генератора контрольного сигнала реализует модуль *posDetGenerator*. Функции определения текущего положения реализованы в модуле *positionDetNineWire*.

* + - 1. Модуль posDetGenerator
         1. Назначение
         2. Описание

Модуль *posDetGenerator* генерирует сигналы, необходимые для работы контрольного генератора (КГ). Контрольный генератор представляет собой генератор синусоидального сигнала на частоте 62,5 Гц. Его сигнал необходим для питанвия датчиков положения стрелки, которые находятся в стрелочном приводе.

Для компенсации падения напряжения на фидерной линии, соединяющей прибор со стрелочным приводом, предусмотрена регулировка выходного уровня контрольного генератора в пределах 21…28 В. Уровень выходного сигнала задаётся в приказе от УС.

Амплитуда сигнала КГ на выходе прибора контролируется в модуле *posDetGenerator*. При отклонении амплитуды сигнала более чем на 10% в сторону увеличения или уменьшения передаётся код аларма 0x6B – напряжение контрольных цепей не в норме.

Логический сигнал с датчика короткого замыкания в цепи КГ поступает на DD2, PORTF12 (сигнал KZGEN), см. [3]. При определении КЗ в цепи КГ формирование сигнала КГ прекращается на 5 секунд, а в статусах в управляющую систему передаётся код аларма 0x0A – контрольное замыкание контрольных цепей.

Сигнал КГ формируется методом ШИМ-модуляции. Модуль *posDetGenerator* обеспечивает формирование импульсов ШИМ-модуляции для управления силовыми транзисторами VT3…VT6 (см. схему [4]). Сигналы GEN1 и GEN2 формируемые МК на выводах PORTA7 и PORTB15 подаются на модуль генератора, схема которого приведена в [4]. На модуле генератора одноимённые сигналы через оптроны опторазвязки VT3 и VT4 подаются на вход управления драйверов DA6 и DA7 (см. схему [4]). С выхода драйвера сигналы подаются на управление транзисторами VT3…VT6 (см. схему [4]).

* + - 1. Модуль positionDetNineWire
         1. Назначение
         2. Описание

Модуль positionDetNineWire служит для определения положения стрелки по сигналам R1 и R2. Отношение амплитуд сигналов R1 и R2 соответствует положению стрелки. При этом абсолютное значение каждого сигнала должно находиться в диапазоне от 12 до 40 Вольт включительно. Выход за пределы этого диапазона любого из сигнала интерпретируется как некорректный сигнал, при этом положение стрелки определяется как «Потеря контроля». Положение стрелки определяется в функции ***posDetermin()***, которая работает согласно алгоритму, показанному на Рисунке 30.

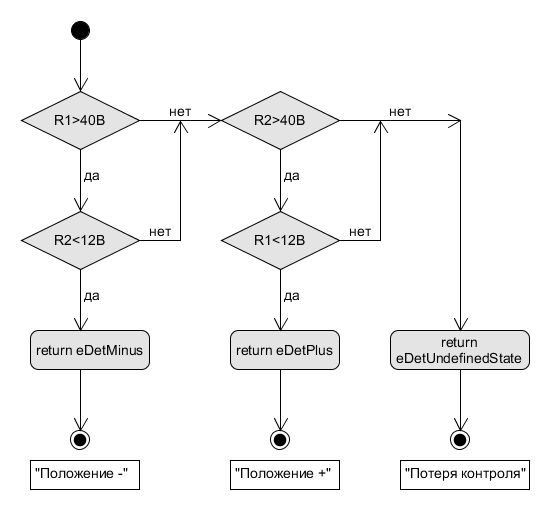


Рисунок – Алгоритм определения положения стрелочного привода по сигналам R1 и R2

Модуль posDetGenerator генерирует сигналы, необходимые для работы контрольного генератора (КГ). Контрольный генератор представляет собой генератор синусоидального сигнала на частоте 62,5 Гц. Его сигнал необходим для питания датчиков положения R1 и R2, которые находятся в стрелочном приводе.

* + - 1. Модуль positionDetFiveEC
         1. Назначение
         2. Описание
      2. Модуль positionDetTypes
         1. Назначение
         2. Описание
    1. Компонент RelayCtrl
       1. Назначение

Компонент предназначен для управления РПВ и контроля исправности схем управления реле.

* + - 1. Состав компонента

Компонент состоит из модулей:

* *relayCtrl* – контроль РПВ, главный модуль;
* *relayCtrlFiveEC* – управление реле в пятипроводном режиме подключения (в данной версии ПО не используется);
* *relayCtrlNineEC* – управление реле в девятипроводном режиме подключения;
* *relayCtrlTypes –* прототипы функций управления и контроля модуля реле.
  + - 1. Описание
         1. Общие сведения

ОКПС-Е-К предназначен для работы в составе АУС-М. В кассете АУС-М расположены шесть приборов – три основных и три резервных. Тремя стрелочными приводами может быть осуществлено управление. Для управления стрелочным приводом задействовано два прибора – основной и резервный. Причём только один прибор может быть подключен к приводу и осуществляет управление приводом. Для физического подключения выходов одного из приборов к приводу служит РПВ. РПВ подключают выход основного либо резервного прибора к приводу.

Управление РПВ (включение и выключение) и его контроль осуществляется только основным прибором.

Управление реле осуществляется подачей на обмотку напряжения питания «24 В» с помощью схемы, состоящей из двух гальванически развязанных ключей. Упрощённая схема, иллюстрирующая процесс управления и контроля состояния РПВ изображена на Рисунке 31.



Рисунок 31 – Упрощенная схема управления РПВ.

Управление РПВ осуществляется транзисторными ключами VT8 и VT9 через оптроны VT15, выполняющие функции гальванической развязки. РПВ включится при одновременном открытии транзисторов VT8 и VT9, которые управляются оптронами VT15.1 и VT15.2 соответственно. Только при одновременном появлении лог. 0 на выходах RC7 DD1 (МК Master) и RC7 DD2 (МК Slave) транзисторы VT8 и VT9 откроются и РПВ включатся.

В модуле реле АУКС-9 РПВ состоит из реле трёх реле – K1, K2 и K3, обмотки которых включены параллельно (см. Рисунок 31). Эти реле установлены на Модуле реле АУКС-9. В свою очередь Модуль реле АУКС-9 установлен на Кросс-плате, к которой подключен также прибор. Для простоты понимания, на Рисунке 31 не обозначены соединительные разъёмы между модулями.

Контроль состояния РПВ проводится по состоянию контактных групп K1.1, K2.1 и К3.1, которые будем называть контрольные контактные группы. Эти контактные группы используются только для контроля состояния реле. Они включены последовательно. Поэтому, только при срабатывании всех трёх реле цепь между сигналами SR и GND\_A будет замкнута через контрольные контактные группы.

Замыкание контрольных контактных групп для прибора аналогично установке внешней перемычки на кросс-плате прибора. Контроль состояния перемычек и безопасное чтение подробно описаны в компоненте BinIn.

* + - * 1. Статическая модель

Статическая модель показана на Рисунке 32. Для объектных типов используется графическое изображение класса. В качестве атрибутов указаны сущности, с которыми работают объектные типы. Для графического изображения структур, макросов, перечислений и др. используется прямоугольник. Прерывистая стрелка указывает на отношение «использует».

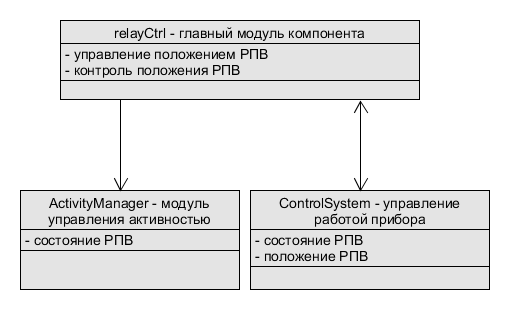


Рисунок 32 – Статическая модель компонента RelayCtrl

* + - 1. Модуль relayCtrlFiveEC
         1. Назначение
         2. Описание
      2. Модуль relayCtrlNineWire
         1. Назначение
         2. Описание
      3. Модуль relayCtrlNineWire
         1. Назначение
         2. Описание
    1. Компонент RS422
       1. Назначение

Компонент предназначен для реализации обмена информацией с управляющей системой по протоколу RS-422.

* + - 1. Состав компонента

В состав компонента входят следующие модули:

* *Rs422* − обеспечение связи по интерфейсу RS-422;
* *Rs422\_crc8* − вычисление контрольной суммы;
* *Rs422\_dataTyp*es − модуль RS-422. Внутренние типы данных;
* *Rs422\_lineExch* − организация обмена по линии RS-422;
* *Rs422\_transceiver* − внутренние функции обмена по интерфейсу RS-422.
  + - 1. Описание
         1. Общие сведения

Обмен с УС осуществляется по двум независимым линиям связи с интерфейсом физического уровня RS-422. Канальный уровень организован на базе бит-ориентированного протокола канального уровня HDLC.

Компонент RS422 обеспечивает прием телеграмм приказов или коротких сообщений «Запрос статуса» и передачу телеграмм статусов.

При приеме телеграмм приказов компонент обеспечивает выполнение следующих функций:

* прием кадров с физического уровня;
* HDLC-декодирование (распаковку телеграмм);
* проверку корректности;
* синхронизацию приказов по двум линиям связи;
* предоставление блока данных из телеграммы приказа прикладному ПО.

Примечание.

Обработку полей блока данных телеграмм приказов выполняет прикладное ПО.

При передаче телеграмм статусов компонент обеспечивает выполнение следующих функций:

* формирование телеграмм статусов из блока данных и алармов, принятых от прикладного ПО;
* HDLC-кодирование (упаковку телеграмм);
* передачу кадров в физический уровень.

Примечание.

Формирование блока данных телеграмм статусов и алармов выполняет прикладное ПО.

Кроме того, компонент предоставляет информацию прикладному ПО о состоянии связи с УС и об ошибках, возникающих в процессе информационного обмена.

Работа компонента осуществляется в двух потоках (главном и потоке синхронизации по времени). В главном потоке выполняется обработка принятых с линий связи данных и взаимодействие с прикладным ПО. В этом потоке работа организована при помощи периодического вызова функции ***Rs422\_run()***. В потоке временной синхронизации выполняется прием данных с физической линии для дальнейшей их обработки в главном потоке и выдача сформированных в главном потоке данных в физическую линию. В этом потоке работа организована при помощи периодического вызова функции ***Rs422\_interrupt()***.

* + - * 1. Структура компонента Rs422

Компонент RS422 состоит из двух объектов *Rs422\_lineExch* и одного объекта *StateRS422*. Каждый объект *Rs422\_lineExch* состоит из объектов *InRS422* и *OutRS422*, которые используют драйвер dsPIC33\_rs422. Статическая модель компонента Rs422 приведена на рисунке 33.



Рисунок 33 – Статическая модель компонента RS422

* + - * 1. Двухканальная работа

Безопасный обмен с УС осуществляется при помощи обработки входных данных и формирования ответов в двух МК. Компонент Rs422 работает в двух МК, при этом оба объекта этого компонента синхронизируют свою работу друг с другом при помощи компонента МКО InterChannel (см. 4.3.12). Синхронизируется состояние независимых автоматов для двух физических линий RS-422 объектов *Rs422\_lineExch*.

Для обеспечения синхронизации двух автоматов в модуле InterChannelId.h объявлены два идентификатора параметров синхронизации, которые имеют наивысший приоритет:

* *eICId\_Rs422Sync\_Rs1* – RS-422 синхронизация 1-й канал;
* *eICId\_Rs422Sync\_Rs2* – RS-422 синхронизация 2-й канал.

При помощи функции ***InterChannel\_setParamSettings()*** для каждого параметра настраиваются следующие параметры синхронизации:

* *eScriptSync* – сценарий синхронизации;
* *eProcSyncEqual* – процедура синхронизации;
* *eProcCheckEqual* – процедура контроля.

Значения остальных настроек задаются равными нулю. Процедура контроля *eProcCheckEqual* не использует значения параметров *paramCheck* и *middleRangeCheck*. Контроль времени Т1, Т2, Т3 отключен, поскольку, в случае сбоев, происходит не переход в ЗС, а переинициализация. Поэтому контроль времени синхронизации параметров выполняется компонентом RS422.

* + - * 1. Прием телеграмм приказов и коротких сообщений «Запрос статуса»

Прием кадров физического уровня

Прием кадров физического уровня осуществляется объектом *InRS422* при помощи драйвера dsPIC33\_rs422 (см. 4.4.1.10). Объект *InRS422* осуществляет побайтное считывание данных из приемника драйвера своей линии. В принятых данных объект осуществляет поиск специальных символов HDLC-пакета, обнаруживает начало и конец телеграммы и восстанавливает исходные данные телеграммы в соответствии с алгоритмом, описанным в разделе 21 Протокола. Наряду с декодированием осуществляется анализ адреса при приеме приказа или короткого сообщения. Если принятый адрес совпадает с установленным, принятая телеграмма фиксируется в циклическом буфере для дальнейшей обработки, а прием кадров продолжается в свободную ячейку буфера. Зафиксированная телеграмма становится доступной для дальнейшей ее обработки. Место в буфере освобождается после того, как ее прочитают компоненты более верхнего уровня. Необходимость наличия циклического буфера обусловлена тем, что прием данных с физической линии осуществляется в потоке временной синхронизации, а обработка принятой телеграммы – в главном потоке.

Проверка корректности принятой телеграммы

Проверка корректности принятой телеграммы приказа осуществляется объектом *Rs422\_lineExch* в соответствии с разделом 5 Протокола. Проверка корректности принятого короткого сообщения «Запрос статуса» выполняется в соответствии с разделом 20 Протокола.

При приеме первой телеграммы выполняются следующие проверки:

* проверяется соответствие фактической длины принятой телеграммы и длины, указанной в принятом поле ML/CO;
* проверяется тип телеграммы;
* сравниваются подсчитанное и принятое значения CRC-8.

Телеграмма приказа считается принятой при выполнении всех условий.

Для контроля целостности коротких сообщений используется инверсный код. Значение CTRL формируется следующим образом: если в комбинации  
бит 0-3 четное число единиц, то CTRL повторяет биты 0-3, а если нечетное, то CTRL содержит инверсные значения бит 0-3. Короткое сообщение считается принятым при условии выполнения контроля целостности.

Если телеграмма приказа или короткое сообщение прошли проверку, выполняется проверка того, что в соседнем МК была принята та же телеграмма или сообщение. Для проверки приема телеграммы соседним МК при помощи компонента InterChannel синхронизируется принятая CRC-8. Если на протяжении 3 мс от соседнего МК не будет получено то же значение CRC-8, телеграмма дальше игнорируется. В случае с коротким сообщением для синхронизации используется значение самого сообщения.

После успешной синхронизации:

* при приеме короткого сообщения формируется телеграмма статуса и инициализируется ее передача в ответ;
* при приеме телеграммы приказа ожидается прием второй телеграммы.

Если вторая телеграмма на протяжении 3 мс не будет принята, первая телеграмма будет проигнорирована.

При приеме второй телеграммы выполняется проверка ее корректности аналогично первой. После выполнения успешной проверки выполняется проверка согласованности телеграмм:

* проверяется согласованность счетчиков циклов;
* проверятся тип телеграмм (одна должна быть телеграммой А, вторая – B);
* проверяется совпадение значений поля длины телеграмм;
* проверяется совпадение значений блока данных телеграммы А с инвертированными значениями блока данных телеграммы B.

Если проверка согласованности телеграмм не проходит, обе телеграммы игнорируются. В случае выполнения всех условий, выполняются следующие действия:

* сравнение значения счетчика циклов в принятой телеграмме со значением счетчика, хранящимся в компоненте. Если принятое значение больше на один или два, для формирования телеграммы статуса используется принятое значение. В другом случае, в статусе передается значение счетчика, хранящееся в компоненте;
* формирование блока данных телеграммы статуса;
* синхронизация CRC-8 второй телеграммы с соседним МК. Если синхронизация не состоялась в течение 3 мс, все принятые телеграммы игнорируются.

В случае успешной синхронизации, компонент обновляет поле блока данных приказа, хранящееся в структуре *StateRS422*, запоминает значение принятого счетчика, перегружает счетчик таймаута отсутствия связи с УС (счетчик используется функцией ***Rs422\_getNoLineExchangeCtrlSystem()*** для формирования признака отсутствия связи с УС).

Обновление блока данных приказа

Обновление блока данных приказа происходит только при выполнении условия согласованности счетчика циклов внутри компонента со счетчиками, принятыми в телеграммах. Критерием согласованности является отличие принятых значений счетчиков на единицу или два в большую сторону от значения счетчика, хранящегося в структуре *StateRS422* компонента Rs422.

Счетчик циклов, хранящийся в структуре *StateRS422*, является общим счетчиком для обеих линий связи. Таким образом, при помощи условия согласованности разрешается вопрос обработки одного и того же приказа при приеме его копии по второй линии связи. Если приказ был принят и обновлен по одной линии, по второй линии происходит просто ответ телеграммой статуса, без увеличения внутреннего счетчика и обновления блока данных приказа.

* + - * 1. Передача телеграмм статуса

Формирование телеграммы статуса

Формирование телеграммы статуса происходит после приема объектом *Rs422\_lineExch* первой корректной телеграммы приказа или короткого сообщения «Запрос статуса» обоими МК. Для формирования телеграмм статусов *Rs422\_lineExch* использует данные, хранящиеся в структуре *StateRS422* компонента Rs422, которые устанавливаются пользовательским ПО.

Блок данных статуса формируется из значений, установленных при помощи функции ***Rs422\_setStatus()***.

Алармы формируются из циклической очереди. Очередь формируется прикладным ПО при помощи функций ***Rs422\_addAlarm()*** и ***Rs422\_removeAlarm()***, которые, соответственно, добавляют алармы в очередь и удаляют их из очереди. При передаче телеграммы статуса передается только один аларм. При приеме нового приказа в телеграмме статуса будет передан очередной аларм. Когда будут переданы все алармы, при приеме очередного приказа будет передаваться первый аларм в очереди.

Счетчик циклов в телеграммах статуса зависит от выполнения условий согласованности счетчиков. Если условие выполняется, счетчик будет соответствовать счетчику цикла, принятому в приказе, если нет – будет передано значение внутреннего счетчика.

Передача телеграмм статуса

Сформированная телеграмма статуса передается объекту *OutRS422*. Также этому объекту задается время, по истечении которого должна быть начата передача. Для МК Master это время составляет 3 мс от приема первого байта телеграммы приказа или короткого сообщения «Запрос синхронизации», для МК Slave – 4,2 мс.

Объект *OutRS422*, обнаружив что у него для передачи есть готовые данные, выжидает заданное ему время и, при помощи драйвера dsPIC33\_rs422, переключает направление передачи на передачу и выдает данные в линию. После того как все байты будут переданы он переключает направление передачи на прием и ожидает готовности новой порции данных.

Если принята вторая телеграмма приказа, которая не прошла проверку на корректность, не прошла проверку на согласованность двух телеграмм или получена только одним МК, начатая передача данных отменяется при помощи переинициализации объекта *OutRS422* с помощи функции ***outRS422\_ctor()***. В этом случае УС, не приняв двух телеграмм статусов, проигнорирует уже переданные данные.

После окончания передачи телеграммы статуса, оба МК синхронизируют свое состояние при помощи МКО после чего формируется событие, которое сигнализирует об успешном завершении процедуры обмена данными. Данное событие может быть прочитано прикладным ПО при помощи функции ***Rs422\_getLineEvent()***. После прочтения данное событие будет сброшено. Также перегружается счетчик таймаута отсутствия обмена с УС (счетчик используется функцией ***Rs422\_getNoLineExchange()*** для формирования признака отсутствия обмена с УС).

* + - 1. Модуль Rs422\_dataTypes
         1. Назначение

Модуль Rs422\_dataTypes содержит определение типов и констант, используемых модулями компонента RS422. Через переменные этих типов осуществляется обмен данными между модулями.

* + - * 1. Интерфейсы

Модуль содержит объявления следующих структур: *StateRS422*, *InRS422*, *OutRS422, CommunicationRS422*.

Структура *StateRS422* структура состояния модуля, предназначена для хранения данных компонента Rs422, которые совместно используются модулем Rs422 и объектами типа *Rs422\_lineExch*. Структура содержит следующие поля:

* *ctrl* – включение/отключение модуля. Если данное поле содержит значение *false*, то происходит выход из функций ***Rs422\_run()*** и ***Rs422\_interrupt()*** без выполнения каких-либо действий;
* *numberCycleOK* – количество циклов обмена;
* *alarm[MAX\_ALARM\_COUNT]* – массив кодов алармов для телеграмм статуса. Предназначен для организации циклической очереди для передачи алармов в телеграммах статусов;
* *numAlarm* – количество алармов в очереди;
* *order[MAX\_ORDER\_DATA]* – принятый блок данных приказа;
* *orderLength* – длина блока данных приказа;
* *status[MAX\_STATUS\_DATA]* – блок данных статуса;
* *statusLength* – длина блока данных статуса;
* *address* – адрес прибора на шине RS-422;
* *noExchangeTime* – время по истичению которого принимается решение об отсутствии событий обмена на линии связи.;
* *noExchangeTimeCtrlSystem* – время по истичению которого принимается решение об отсутствии связи с УС.

Структура *InRS422* Структура данных для приема по линии связи RS-422. Предназначена для хранения данных объектов данного типа. Работа с этими объектами осуществляется группой функций, имена которых начинаются с «*inRS422\_*» и которые реализованы в модуле Rs422\_transceiver. Структура содержит следующие поля:

* *aInRS422[V\_ARRAY\_IN\_RS422][MAX\_ORDER\_TELE]* – массив для хранения принятых телеграмм. Предназначен для организации циклической очереди, которая обеспечивает буферизацию данных при их обмене между функциями главного потока и потока временной синхронизации;
* *aLength[V\_ARRAY\_IN\_RS422]* – длина принятой телеграммы.
* *acTimeIn[V\_ARRAY\_IN\_RS422]* – массив счетчиков, которые считают время с момента приема первого байта телеграммы приказа или короткого сообщения «Запрос статуса». Разрешение счетчика равно периоду потока временной синхронизации. Счетчик используется для формировании времени начала передачи телеграммы статуса;
* *indWriteArray* – индекс ячейки массива принятых телеграмм, в которую осуществляется запись данных, принятых драйвером модуля UART;
* *indReadArray* – индекс ячейки массива принятых телеграмм, из которой будет произведено чтение принятой телеграммы пользовательским ПО;
* *indDate* – текущий индекс в ячейки массива принятых телеграмм;
* *EnumInByte state* – состояние автомата приема и HDLC-декодирования телеграммы;
* *inByte* – указатель на функцию драйвера, принимающую байты с линии связи.

Структура *OutRS422* структура данных для передачи по линии связи RS-422. Предназначена для хранения данных объектов данного типа. Работа с этими объектами осуществляется группой функций, имена которых начинаются с «*outRS422\_*» и которые реализованы в модуле Rs422\_transceiver. Структура содержит следующие поля:

* *aOutRS422[N\_ARRAY\_OUT\_RS422]* – массив для хранения телеграммы статуса, которую необходимо передать;
* *length* – длина телеграммы статуса;
* *indDate* – текущий индекс в массиве указывает на байт, который следующим будет передан в линию связи;
* *timeOut* – счетчик времени до запуска передачи телеграммы. Разрешение счетчика равно периоду потока временной синхронизации;
* *EnumOutByte state* – состояние автомата передачи телеграммы;
* *outByte* – указатель на функцию драйвера, передающую байты в линию связи;
* *manageInOut* – указатель на функцию драйвера, которая переключает направление передачи;
* *transmCompl* – указатель на функцию драйвера, которая информирует о завершении передачи текущего байта и готовности передатчика к передаче следующего байта.

Структура *CommunicationRS422* cтруктура данных линии RS-422, содержит настройки и состояния объектов *Rs422\_lineExch*. Содержит следующие поля:

* *numLine* – номер линии;
* *state* – состояние автомата обработки принятых телеграмм;
* *aInTele[2][MAX\_ORDER\_TELE]* – массивы для принятых телеграмм;
* *lengthInTele* – длина последней принятой телеграммы;
* *eICId\_Rs422Sync* – номер синхронизированной переменной;
* *exchangeEvent* – событие обмена;
* *exchangeError* – код ошибки;
* *inRS422* – объект типа *InRS422*, предназначенный для приема телеграмм;
* *outRS422* – объект типа *OutRS422*, предназначенный для передачи телеграмм;
* *noExchangeCnt* – счетчик таймаута отсутствия обмена;
* *noExchangeCtrlSystemCnt* – счетчик таймаута отсутствия обмена c УС;
* *iniPortCtrl* – указатель на функцию инициализации портов управления Rx/Tx для переключения направления передачи;
* *iniUart* – указатель на функцию инициализации регистров UART;
* *timeOut* – переменная таймаута.
  + - 1. Модуль Rs422\_lineExch
         1. Назначение

Модуль должен обеспечивать обмен данными с УС для конкретной физической линии RS-422 в соответствии с Протоколом. Модуль реализует прием, контроль и обработку телеграмм приказов и коротких сообщений «Запрос статуса», формирование и передачу телеграмм статуса по конкретной линии, следит за корректным выполнением приема и передачи двумя МК.

* + - * 1. Описание

Модуль выполнен в виде автомата состояний состояние которого определяются константами типа *eRs422\_execution*. Константы имеют следующие значения:

* *eRs422exec\_resetSynchro* – сброс синхронизации;
* *eRs422exec\_1stTeleReception* – ожидание приема 1-ой телеграммы;
* *eRs422exec\_1stTeleSynchro* – ожидание синхронизации каналов после приема 1-ой телеграммы;
* *eRs422exec\_2stTeleSynchro* – ожидание синхронизации каналов после приема 2-ой телеграммы;
* *eRs422exec\_transmEnd* – ожидание завершения передачи;
* *eRs422exec\_synchroTransmEnd* – ожидание синхронизации окончания передачи телеграмм.

Физическая линия RS-422 подключена к модулям UART одного и второго контроллера. Таким образом, все входные данные в обоих МК должны быть приняты в один момент времени и быть идентичными. Для обеспечения безопасного приема приказов, кроме контроля корректности и согласованности телеграмм А и B, выполняется проверка идентичности принятых данных двумя МК. Для обеспечения безопасной передачи статусов статусы формируются в обоих МК и передаются в разных телеграммах. Телеграмма А от одного контроллера, телеграмма B от другого. Порядок передачи телеграмм (сначала А или B) определяется порядком их приема. Во избежание коллизий на линии, МК Master передает свою телеграмму статуса через 3 мс от начала приема телеграммы приказа, МК Slave – через 4,2 мс.

Прием по второй линии осуществляется аналогично, с использованием второго модуля UART.

Синхронно в обоих МК модуль должен осуществлять переход из одного состояния к другому. В случае расхождения автоматов все передачи должны быть отменены, а автоматы приведены к общему состоянию через состояние «сброс синхронизации».

Состояние «сброс синхронизации» является начальным, которое подготавливает автоматы в двух МК к приему первой телеграммы. В это состояние компонент попадает после начальной инициализации или вследствие возникновения ошибки в других состояниях. При входе в это состояние происходит переинициализация объектов *InRS422* и *OutRS422*, что приводит к отмене всех действий, выполняемых из потока временной синхронизации. Сбрасываются все принятые данные и отменяются все данные, которые передаются или ждут своей очереди для передачи. В этом состоянии компонент ждет второй МК и когда оба МК перешли в это состояние, они переходят в состояние «Ожидание приема 1-ой телеграммы».

В состоянии «Ожидание приема 1-ой телеграммы» ПО, при помощи объекта *InRS422*, ожидает прием телеграммы приказа или короткого сообщения «Запрос статуса». При приеме проверяет корректность телеграммы, а, в случае успешной проверки, отправляет на синхронизацию значение CRC-8 или значение короткого сообщения. Фиксируется время начала передачи телеграммы статуса. Для МК Master это время составляет 3 мс от приема первого байта телеграммы приказа или короткого сообщения «Запрос синхронизации». Для МК Slave это время составляет 4,2 мс от приема первого байта телеграммы приказа или короткого сообщения «Запрос синхронизации». После чего МК переходит в состояние «Ожидание синхронизации каналов после приема 1-ой телеграммы».

В состоянии «Ожидание синхронизации каналов после приема 1-ой телеграммы» ПО ожидает приема значения CRC-8 или короткого сообщения на протяжении 3 мс. После приема, в случае совпадения значений у обоих МК, компонент:

* при приеме короткого сообщения формирует блок данных телеграммы статуса, инициализирует его передачу и переходит в состояние «Ожидание завершения передачи»;
* при приеме телеграммы приказа ожидает приема второй телеграммы. Проверяет ее корректность, а также согласованность первой и второй телеграммы. После чего формирует блок данных телеграммы статуса и инициализирует его передачу, синхронизирует значение CRC-8 этой телеграммы и переходит в состояние «Ожидание синхронизации каналов после приема 2-ой телеграммы».

В состоянии «Ожидание синхронизации каналов после приема 2-ой телеграммы» ожидается прием значения CRC-8 второй телеграммы от соседнего МК. Происходит сравнение этих значений и, в случае их совпадения, происходит обновление блока данных приказа, который будет передан прикладному ПО при помощи функции ***Rs422\_getOrder()***. Затем обновляется внутренний счетчик приказов. Перегружает счетчик времени, который отсчитывает время отсутствия связи с УС и используется функцией ***Rs422\_getNoLineExchangeCtrlSystem()*** для формирования признака отсутствия связи с УС. Затем происходит переход в состояние «Ожидание завершения передачи».

В состоянии «Ожидание завершения передачи» ожидается завершение передачи телеграммы статуса, а затем происходит переход в «Ожидание синхронизации окончания передачи телеграмм».

В состоянии «Ожидание синхронизации окончания передачи телеграмм» ожидается сообщение от соседнего МК о том, что он также перешел в это состояние. После чего формируется событие *Rs422\_getLineEvent*, которое сигнализирует об успешном завершении процедуры обмена. Данное событие может быть прочитано прикладным ПО при помощи одноименной функции. При чтении событие сбрасывается, также перегружает счетчик времени, который отсчитывает время отсутствия обмена с УС и используется функцией ***Rs422\_getNoLineExchange()*** для формирования признака отсутствия обмена с УС. Затем происходит переход в состояние «Ожидание приема 1-ой телеграммы».

* + - * 1. Использование модуля

Для использования модуля необходимо объявить объект типа *CommunicationRS422*, задать в этой переменной значения полей *numLine*, *eICId\_Rs422Sync*, а также назначить функции, работающие с драйвером dsPIC33\_rs422: ***iniPortCtrl()***, ***iniUart()***, ***inRS422.inByte()***, ***outRS422.outByte()***, ***outRS422.manageInOut()***, ***outRS422.transmCompl()***. Вызвать функцию ***Rs422\_lineCtor()***, которой в качестве параметра передать созданную структуру.

Объявить объект типа *StateRS422* и задать следующие поля:

* *ctrl* – отключить работу модуля;
* *numberCycleOK* – обнулить количество циклов обмена;
* *numAlarm* – обнулить количество алармов в очереди;
* *orderLength* – задать длину блока данных приказа для конкретного прибора;
* *statusLength* – задать длину блока данных статуса;
* *address* – задать адрес прибора на шине RS-422;
* *noExchangeTime* – установить время отсутствия обмена с УС, по истечении которого принимается решение о его отсутствии. В данном проекте 1500 мс;
* *noExchangeTimeCtrlSystem* – установить время отсутствия связи с УС, по истечении которого принимается решение о его отсутствии. В данном проекте 1500 мс.

В главном потоке вызывать функцию ***Rs422\_lineRun()***, передавая ей в качестве параметров указатели на переменную типа *CommunicationRS422*, которая передавалась при вызове конструктора, и переменную типа *StateRS422*.

В потоке временной синхронизации вызывать функцию ***Rs422\_lineInterrupt()***, передавая ей в качестве параметров указатели на переменную типа *CommunicationRS422*, которая передавалась при вызове конструктора, и переменную типа *StateRS422*.

* + - 1. Модуль Rs422\_transceiver
         1. Назначение

Модуль содержит набор функций для работы с объектами типа *InRS422* и *OutRS422*, а также функции, которые проверяют корректность и согласованность принятых телеграмм, формируют телеграммы статуса.

* + - * 1. Описание

Для работы с объектом *InRS422* модуль содержит функции ***inRS422\_ctor()***, ***inRS422\_run()*** и ***InRS422\_getTele()***. Для работы с объектом необходимо создать структуру *InRS422*, записать в поле *inByte* адрес функции драйвера, которая принимает байты с линии связи, после чего вызвать функцию ***inRS422\_ctor()***. Для приема телеграмм необходимо с периодом потока временной синхронизации вызывать функцию ***inRS422\_run()***, которая осуществляет обнаружение  
HDLC-пакетов, декодирование их, проверку адреса и копирование в очередь принятой телеграммы. Прочитать принятую телеграмму можно вызвав функцию ***InRS422\_getTele()***, которая скопирует ее по адресу, указанному в параметре *aTele*, и вернет длину принятой телеграммы если в очереди есть принятая телеграмма. Если же принятой телеграммы в очереди нет, функция ничего не будет копировать и вернет ноль. Вместе с копированием телеграммы функция копирует в параметр *ptrTime* время, которое прошло с момента приема первого байта телеграммы до момента вызова функции.

Для работы с объектом *OutRS422* модуль содержит функции ***outRS422\_ctor()***, ***outRS422\_run()*** и ***InRS422\_getTele()***. Для работы с объектом необходимо создать структуру *OutRS422* и проинициализировать следующие поля:

* *outByte* – записать адрес функции драйвера, которая принимает байты с линии связи;
* *manageInOut* – записать адрес функции драйвера, которая управляет переключением направления приема;
* *transmCompl* – записать адрес функции драйвера, которая информирует о готовности передатчика к передаче следующего байта.

После чего вызвать функцию ***OutRS422\_ctor()***. Для работы объекта необходимо с периодом потока временной синхронизации вызвать функцию ***outRS422\_run()***. Для формирования телеграммы статуса необходимо вызвать функцию ***formTeleStatus()***, которой в качестве первого параметра указать поле *aOutRS422* структуры *OutRS422*, указать тип телеграммы, адрес, номер цикла, код аларма, блок данных статуса и его длину. Результат, который вернет функция, необходимо записать в поле *length* структуры *OutRS422*. После того как поле станет ненулевым, функция ***outRS422\_run()*** начнет передачу телеграммы на линию связи.

Для того чтобы из очереди кодов алармов извлечь очередной, необходимо воспользоваться функцией ***formAlarmRS422()***, которая в качестве параметров принимает номер цикла и массив с кодами алармов. Функция переходит к следующему коду только если отличаются сохраненный внутри функции номер цикла и переданный в параметре номер цикла. Таким образом, при каждой смене номера цикла из очереди извлекается очередной элемент. Элементы передаются по кругу. Если в очереди алармы отсутствуют, функция возвращает ноль – код отсутствия алармов. Коды алармов помещаются в очередь и удаляются из нее при помощи функций компонента Rs422 пользовательским ПО.

После завершения передачи телеграммы статуса поле *length* структуры *OutRS422* примет значение ноль.

При проверке целостности данных в телеграмме используется значение поля CRC-8, которое подсчитывается при помощи функции ***crc8\_update()*** модуля Rs422\_crc8.

* + - 1. Модуль Rs422\_crc8
         1. Назначение

Модуль реализует алгоритм подсчета CRC-8 в соответствии с Приложением А Протокола.

* + - * 1. Описание

Модель табличным методом вычисляет значение CRC-8 массива данных, переданного в параметре *data*, длиной *len*, используя начальное значение *crc*.

* + 1. Компонент SafetyPowerControl
       1. Назначение

Управления безопасным блоком питания (ББП) драйверов силовых транзисторов и контроль его состояния.

* + - 1. Состав компонента

Компонент состоит из одного одноименного модуля.

* + - 1. Описание
         1. Общие сведения

Компонент PowerSupplyControl предназначен для:

* управления транзисторами ББП;
* контроля исправности транзисторов ББП;
* перевода прибора в БС при обнаружении неисправности транзисторов ББП.

Упрощенная схема ББП изображена на Рисунке 34. Для упрощения понимания принципов работы и контроля ББП, часть элементов принципиальной схемы прибора не изображена на Рисунке 34.

Рисунок 34 – Упрощенная схема ББП.

Для управления транзисторами ББП МК Master и МК Slave формируют сигнал управления на выходе порта RE13. Этот сигнал представляет собой меандр на частоте 32 кГц. Сигналы управления (PS1-M и PS1-S), формируемые МК Master и МК Slave, – противофазные (см. Рисунок 35). При нормальной эти сигналы управляют работой транзисторов VT12 и VT13, к выводам стока которых подключена обмотка трансформатора T1, с обмотки III которого питающее напряжение поступает в нагрузку. С обмотки II снимается напряжение, из которого транзистором VT11 формируется напряжение контрольного сигнала KPS1, которое подаётся на МК Master и MK Slave. В нормальной работе напряжение KPS1 представляет собой меандр, имеющий небольшой фазовый сдвиг относительно управляющих сигналов (см. Рисунок 35).

При обнаружении опасной ситуации любой из МК должен обеспечить снятие напряжение питания с нагрузки. Для этого МК отключает формирование управляющего сигнала на порту RE13. Однако при пробое одного из транзисторов VT12 или VT13 напряжение в нагрузку подаваться будет, но уменьшенное. Для исключения таких случаев проводится диагностика исправности транзисторов VT12 и VT13. При формировании управляющих сигналов в каждом цикле формирования меандра проверяется, что сигнал KPS1 равен лог. 0. Это означает, что транзистор открывается. Если это условие не выполняется, то значение счётчика *spc.PeriodicCheckupFaultCnt* увеличивается на три. При достижении значения порога *PeriodicCheckupFaultThreshold* прибор переходит в 3С. Это сделано для фильтрации помех, которые возникают при работе прибора.

Для диагностики возможности закрывания транзистора (отсутствия пробоя сток-исток) в управляющей последовательности происходит пропуск одного управляющего импульса через каждые 100 мс (задаётся константой *SPC\_TESTING\_PERIOD*). Это не влияет на качество выходного напряжения, однако позволяет достоверно диагностировать отсутствие пробоя в транзисторе. Вид сигналов при тестировании верхнего ключа VT12 и нижнего ключа VT13 показаны на Рисунке 35. Для фильтрации помех, которые возникают при работе прибора, переход в 3С выполняется при превышении числа неудачных тестирований, определённых константой *TestingFaultThreshold*. Т.к. при каждом неудачном тестировании значение счётчика ошибочных тестирований увеличивается на три, то переход в 3С прибора произведёт не более чем через 400 мс с момента возникновения неисправности пробоя одного из транзисторов.

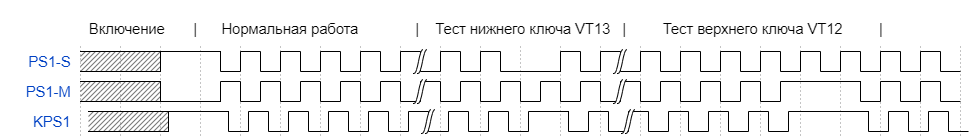


Рисунок 35 – Сигналы управления и контроля ББП

* + - * 1. Статическая модель

Компонент не имеет интерфейсных функций. При запуске МК компонент инициализируется функцией *SafetyPowerControl\_ctor()*. Далее компонент работает автономно, обеспечивая упраление и диагностику ББП.

* + 1. Компонент ShuntShift
       1. Назначение

Предназначен для управления переводом стрелки.

* + - 1. Состав компонента

Компонент состоит из пяти модулей:

* *shuntShift* – общий модуль, включающий в себя общую структуру интерфейсов;
* *shuntShiftFiveEC* – управление переводом стрелки в пятипроводной схеме стрелочного привода;
* *shuntShiftGen* – формирование трёхфазного синусоидального напряжения с заданной частотой и амплитудой;
* *shuntShiftMotor* – управление асинхронным двигателем при переводе стрелки;
* *shuntShiftNineWire* – управление переводом стрелки в девятипроводной схеме стрелочного привода.
  + - 1. Описание

Алгоритм работы реализован как автомат состояний в функции ***ShuntShift\_run()***. При поступлении нового актуального приказа на перевод стрелки (то есть того, который требует выполнения), включается выполнение генерации трёхфазного сигнала, используя ШИМ-модуляцию модуля PWM МК в функции *shuntShiftGen\_run()* (исходя из входных для работы модуля значений уровня сигнала, частоты, последовательности формирования фаз (для управления направлением вращением двигателя). В функции *shuntShiftGen\_interrupt*() выполняется обновление значений длительности цикла ШИМс частотой 14 кГц, формируемых табличным методом. В модуле *shuntShiftMotor* в функции *ShuntShiftMotor\_run()*, реализованной в виде автомата состоянийпроизводится:

* разгон асинхронного двигателя, используя изменение частоты сигнала вместе с амплитудой сигнала в течение 500мс, согласно соотношения U/f = 4.4, где U – действующее значение формируемого напряжения, f – значение частоты сигнала;
* управление вращением двигателя до получения команды на отключение двигателя;
* анализ состояния рабочих цепей во время перевода стрелки (определение возможного КЗ, обрыва, перегрузки по току);
* отключение двигателя путём плавного снижения напряжения в течение 100 мс.

Общий алгоритм работы модуля соотвествует следующей диаграмме деятельности:

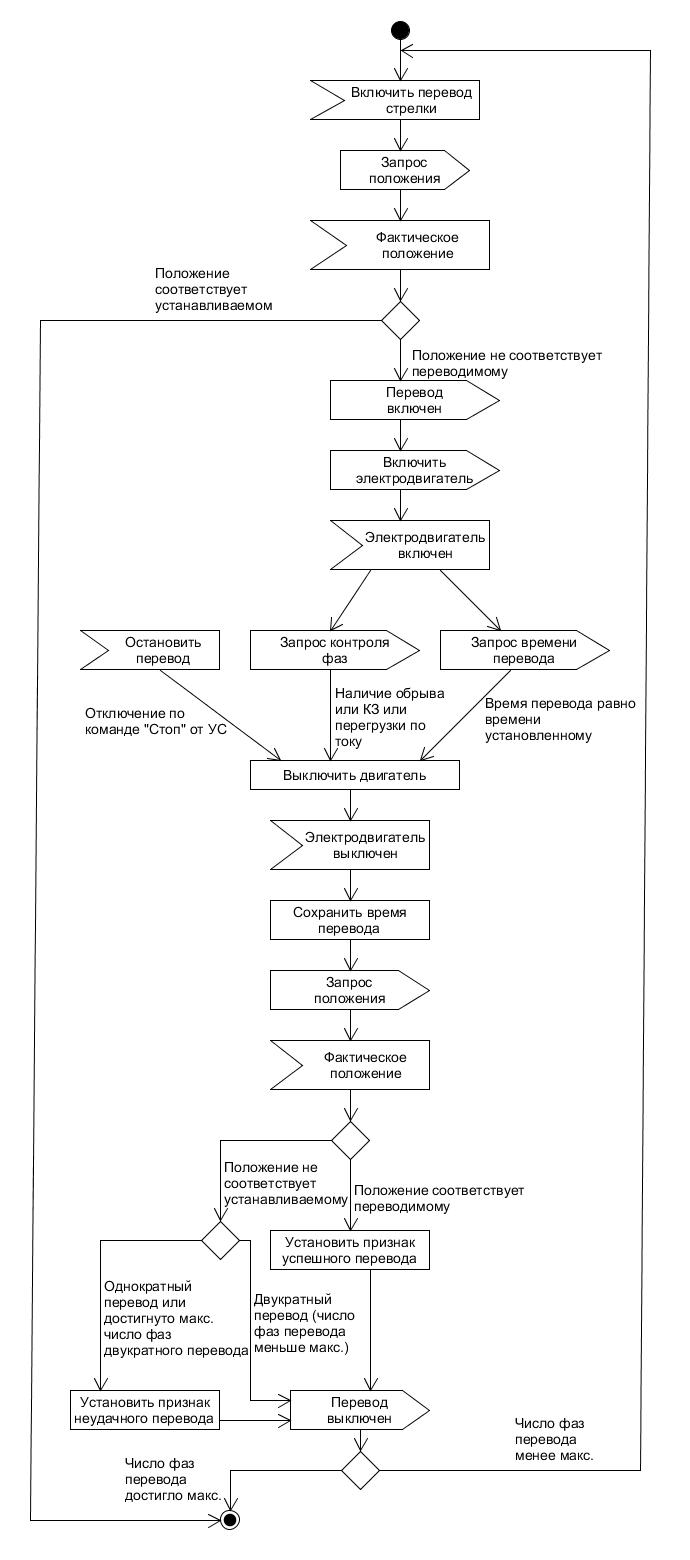


Рисунок – Диаграмма деятельности компонента ShuntShift

* + - 1. Модуль shuntShiftFiveEC
         1. Назначение
         2. Описание
      2. Модуль shuntShiftGen
         1. Назначение
         2. Описание
      3. Модуль shuntShiftMotor
         1. Назначение
         2. Описание
      4. Модуль shuntShiftNineWire
         1. Назначение
         2. Описание
      5. Модуль shuntShiftTypes
         1. Назначение
         2. Описание
      6. Порядок применения

1. включить в проект компонент ShuntShift;
2. включить в проект драйвер ШИМ трёхфазного генератора shuntShiftGenDrv;
3. организовать вызовы конструкторов модулей:
4. ***ShuntShiftGen\_ctor()*** из ***ShuntShiftMotor\_ctor()***;
5. ***ShuntShiftMotor\_ctor()*** из ***ShuntShift\_ctor()***;
6. ***ShuntShift\_ctor()*** при входе в StateMain после успешной инициализации в StateInitial.
7. организовать в ГП вызов функций:
8. ***ShuntShiftGen\_gen()*** из ***ShuntShiftMotor\_run()***;
9. ***ShuntShiftMotor\_run()*** из ***ShuntShift\_run()***;
10. ***ShuntShift\_run()*** в StateMain.
11. организовать в потоке временной синхронизации (обработчик прерывания по изменению уровня входного сигнала) вызов функции ***ShuntShiftGen\_interrupt()***.
    1. Системные каталоги

Системные каталоги содержат программные модули, которые не являются компонентами в том виде, в котором этот термин определен. Системные каталоги имеют разное назначение:

* *drv* – обеспечивают доступ компонентов ПО к аппаратному обеспечению через программные интерфейсы (драйвера);
* *systems* – настраивают компоненты ПО для использования в конкретном проекте;
* *utility* –
* *Testing* –
  + 1. Drv
       1. Назначение

Каталог Drv содержит программные модули, которые обеспечивают доступ компонентов ПО к аппаратному обеспечению через программные интерфейсы (драйвера). Драйвера могут обеспечивать как доступ к периферии МК, так и доступ к блокам схемы прибора, используя периферию МК.

* + - 1. Состав каталога

В состав каталога входят следующие модули:

* *AdcDriver* – интерфейс с АЦП;
* *BloслExch\_driver* – драйвер межблочного обмена;
* *checkRegistersDrv* – драйвер проверки регистров МК;
* *dsPIC33\_rs422 -* драйвер модуля RS-422 для МК dsPIC33;
* *Ecan\_driverPIC33* − драйвер для модуля CAN;
* *EepromDrv* − драйвер работы с EEPROM;
* *I2Cdrv\_PIC33* − драйвер интерфайса I2C;
* *posDetGen\_drv* − драйвер определения положения стрелки;
* *relayCtrlGen\_drv* − драйвер управления реле;
* *shuntShiftGenDrv* − драйвер управления приводом;
* turnOffdsPIC33 – драйвер отключения периферии МК.
  + - 1. Описание

С целью разделения логических функций ПО от аппаратуры используются следующие уровни:

* уровень драйверов − обеспечивает программный интерфейс доступа к аппаратуре;
* логический уровень − реализует логику принятия решений и при помощи уровня драйверов формирует команды (сигналы) для выдачи в аппаратуру.

Данное разделение позволяет логическому уровню быть пригодным к автономному тестированию и проще переносить этот уровень на другие аппаратные платформы.

Драйвера с логическим уровнем могут связываться статически и динамически. Статическое связывание осуществляется на этапе сборки проекта. Компонент логического уровня использует универсальные имена функций. Модуль драйвера содержит реализацию этих функций. Компоновщик находит нужный объектный файл и связывает компонент логического уровня с драйвером. Таким образом, подмена драйверов выполняется при помощи удаления или добавления нужной реализации драйвера в проект. Динамическое связывание осуществляется через структуру, которая содержит поля, хранящие адреса вызовов функций. Компонент ПО, получив указатель на эту структуру, обращается к функциям драйвера через поля этой структуры. Таким образом, драйвер может быть подменен в процессе выполнения кода. Использование динамического связывания драйверов позволяет предопределять драйвера без изменения компонента логического уровня, не вызывая при этом конфликта имен. А также проводить автономное тестирование компонента логического уровня без пересборки тестового проекта, подменяя драйвер различными объектами, имитирующими его поведение с разными тестовыми наборами данных.

* + - 1. Модуль AdcDriver\_PIC33
         1. Назначение

Модуль AdcDriver\_PIC33 предназначен для инициализации встроенного в процессор АЦП и работы с ним.

* + - * 1. Состав компонента

Компонент состоит из модулей:

* AdcDriver\_PIC33 – драйвер модуля АЦП;
* Adc\_Inic1.h – макроопределения для инициализации модуля;
* AdcRegsPIC33.h – регистры модуля Adc.
  + - 1. Описание
         1. Общие сведения

Драйвер АЦП представляет из себя набор измерительных каналов (*AdcChannel*), которые последовательно измеряются по каналу АЦП CH0. Управляет работой драйвер АЦП AdcDriver. Работа драйвера АЦП происходит следующим образом.

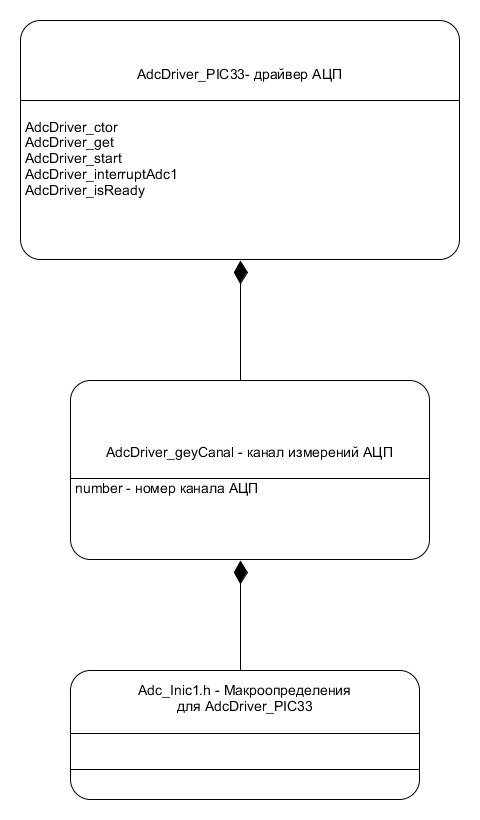
Перед началом работы происходит настройка драйвера АЦП.

По вызову функции ***AdcDriver\_ctor()*** проводится инициализация периферийного модуля МК АЦП. При инициализации дискретных входов АЦП используется макрос INIC\_ADC1\_PIN. Инициализация дискретных входов МК − настройка их, как входов АЦП.

По вызову ***AdcDriver\_start(AdcNumber number)*** запускается измерения канала с индексом *number*. По завершению измерения вызывается функция ***AdcDriver\_runInterruptAdc1()***, которая сохраняет результат измерения в очереди. Переполнение очереди приводит к переходу прибора в ЗС. Затем запускается измерение следующего канала.

Драйвер имеет следующее ограничения:

* частота дискретизации измерительного канала не должна превышать частоту вызова функции ***AdcDriver\_start()*** и должна быть ей кратна;
* необходимо постоянно считывать результаты измерений функцией ***AdcDriver\_getDataChannel()***, что бы не допустить переполнение очереди;
* необходимо обеспечить своевременное считывание измерений с регистров АЦП функцией ***AdcDriver\_runInterrupt()***, обычно для этого организовывается обработка прерывания по завершению конвертации АЦП.
  + - * 1. Статическая модель



* + - 1. Модуль BlockExch\_driver
         1. Назначение

Модуль BlockExch\_driver обеспечивает управление обменом со смежным прибором.

* + - 1. Описание

Компонент состоит из одноимённого модуля.

* + - * 1. Общие сведения

Сигналы для передачи данных в смежный прибор синхронно формируют оба МК: *REZ-TX-M* – сигнал МК Master и *REZ-TX-S* – сигнал МК Slave (порт RB7). Передача логического 0 осуществляется если *REZ-TX-M* = 1 и *REZ-TX-S* = 0. В других случаях происходит передача логической 1.

Прием данных от смежного прибора выполняется обоими МК путем чтения и обработки сигнала *REZ-RX* (порт RB7).

Основная структура *driverStr* объявлена в файле BlockExch\_driver.c и состоит из двух структур – *txData* и *rxData*, которые содержат данные для управления передачей и приемом соответственно. Модуль инициализируется с помощью функции ***BlockExch\_ctor()***. В ней вызываются функции ***transmitIni()*** и ***receptionIni()***, инициализирующие структуры *txData* и *rxData*, в т.ч. указатели на функции для настройки и работы с конкретными портами МК для передачи и приема данных соответственно.

Основная рабочая функция модуля ***BlockExch\_run()*** вызывается из функции ***InterrruptHandler.c***в обработчике прерывания синхронизации с периодом, равным кванту синхронизации *Tsync = 62,5 мкс*.

Функция ***BlockExch\_run()*** выполняет следующие действия:

* определяет условия, позволяющие начать передачу, и устанавливает соответствующий признак;
* вызывает функцию ***transmition()***, непосредственно осуществляющую передачу данных в физическую линию;
* вызывает функцию ***reception()***, непосредственно осуществляющую прием данных с физической линии.

Функция ***transmition()*** вызывается каждое восьмое прерывание синхронизации, т.е. каждые *Tsync \* 8 = 500 мкс*.

Функция ***transmition()***реализована в виде КА последовательная смена состояний которого соответствует определенной фазе выдачи сообщения (за исключением начального состояния, в котором автомат ожидает разрешения начала передачи).

Процесс передачи сообщения начинается при одновременном соблюдении следующих условий:

* выполняется 16-е прерывание синхронизации (последнее в текущем 1 мс цикле главного потока);
* КА процесса передачи находится в начальном состоянии;
* есть данные для передачи (длина передаваемых данных не равна 0).

Функция ***reception()*** вызывается каждое второе прерывание синхронизации, т.е. каждые *Tsync \* 2 = 125 мкс*.

Функция ***reception()*** реализована в виде конечного автомата состояния которого отражают прием отдельных элементов сообщения (см. **Ошибка! Источник ссылки не найден.** и рисунок **Ошибка! Источник ссылки не найден.**). В начальном состоянии происходит ожидание логической 1 на приемной линии, затем – ожидание СТОП-посылки, затем – СТАРТ-бита и т.д. Конец принимаемого сообщения определяется при обнаружении бита 1 вместо СТАРТ-бита после приема очередного байта данных. При обнаружении ошибок формата принимаемого сообщения происходит переход в начальное состояние. Для приема одного бита используется *BIT\_QUANTUM\_NUMBER = 4* выборок (чтений) состояния линии связи (сигнал *IND\_RX*). Обработку выборок выполняет функция ***bitSelector()****.* Решение о значении бита (0 или 1) принимается по критерию «3 из 4». Если в анализируемом наборе выборок число 0 = числу 1, это считается ошибкой и происходит переход в начальное состояние автомата.

Процесс передачи/приема одного бита данных условно показан на рисунке 37.



Рисунок 37 – Временная диаграмма передачи одного бита данных сообщения МПО

* + - 1. Модуль checkRegistersDrv
         1. Назначение

Модуль checkRegistersDrv обеспечивает проверку регистров специального назначения, используемых в работе МК.

* + - 1. Описание
         1. Общие сведения

Модуль содержит функцию контроля исправности регистров модулей периферии. Контроль исправности заключается в проверке, что в битах регистров находятся определенные значения. Проверяются обычно регистры таких модулей периферии, как главный таймер, система прерываний, АЦП, CAN, DMA.

* + - 1. Модуль dsPIC33\_rs422
         1. Назначение

Драйвер модуля RS-422 для МК dsPIC33. Обеспечивает программные интерфейсы доступа к модулям UART1 и UART2, а также к портам дискретного ввода/вывода, которые управляют переключением направления передачи драйверов интерфейса RS-422.

* + - * 1. Описание

Драйвер обеспечивает независимую работу по двум линия связи RS-422 при помощи двух модулей UART1 и UART2 и дискретных портов ввода/вывода 14 и 15 порта D.

Функции ***inicPortContr\_L1()*** и ***inicPortContr\_L2()*** настраивают дискретные порты для работы на выход и устанавливают направление передачи на прием. Функции ***inic\_L1()*** и ***inic\_L2()*** настраивают модули UART1 и UART2 на работу в режиме 8 бит данных, без паритета с 1 стоп-битом на скорости 263314 бит/c. Вызов этих функций должен обязательно осуществиться перед использованием модуля.

Прием данных необходимо выполнять при помощи функций ***inByte\_L1()*** и ***inByte\_L2()***. Если эти функции вернули *true* значит соответствующий модуль UART принял байт, который был прочитан и помещен в параметр этой функции.

Для передачи байта данных необходимо при помощи функций ***isTransmCompl\_L1()*** и ***isTransmCompl\_L2()*** убедиться, что предыдущий байт передан и передать новый при помощи функций ***outByte\_L1()*** и ***outByte\_L2()***.

* + - 1. Модуль EcanAck\_driverPIC33
         1. Назначение

Модуль Ecan\_driverPIC33 предназначен для передачи и приема сообщений по шине CAN с использованием дополнительного подтверждения приема при помощи дискретных портов ввода/вывода МК. нтерфейсы

Состав модуля

Модуль состоит из одноимённого файла.

* + - * 1. Описание

Модель использования модуля

Перед использованием драйвера должна быть вызвана функция ***EcanAck\_ctor()***, которая проинициализирует драйвер на работу с указанным модулем CAN.

Работа драйвера построена на опросе SFR-регистров МК. Опрос регистров осуществляется функцией ***EcanAck\_run()***.

Для отправки сообщения при помощи функции ***EcanAck\_isOutReady()*** необходимо убедиться что передатчик готов к передачи новых данных и инициировать передачу при помощи функции ***EcanAck\_set()***. После начала передачи функция ***EcanAck\_isOutReady()*** будет возвращать *false* до тех пор, пока передача не завершится успешно или с ошибкой, после функция вернет *true*.

Для приема сообщения при помощи функции ***EcanAck\_isInReady()*** необходимо убедиться что приемник принял сообщение и прочитать его при помощи функции ***EcanAck\_\_get()***. После прочтения принятого сообщения признак его наличия будет сброшен и функция ***EcanAck\_\_isInReady()*** будет возвращать *false* до тех пор, пока не примет новое сообщение.

Вызов функций драйвера должен осуществляться из того же потока, что вызов функции ***EcanAck\_\_run()*** поскольку драйвер не имеет буферов данных и если данные не будут вычитаны до приема следующего сообщения они будут утеряны.

Алгоритм работы обмена данными

Кроме модуля CAN дополнительно используются два дискретных порта ввода/вывода, которые соединены между узлами. Выход одного узла является входом второго и наоборот.

Обмен данными с дополнительным подтверждением приема между двумя узлами шины CAN должен осуществляться по следующему сценарию:

* передающий узел перед началом передачи читает состояние дискретного порта, запоминает его и начинает передачу сообщения по шине CAN;
* принимающий узел после приема корректного сообщения инвертирует состояние дискретного порта;
* передающий узел, после окончания передачи и получения подтверждения приема по шине CAN, ожидает изменение состояния дискретного порта на протяжении трех тактов вызова ***EcanAck\_\_run()***;
* если уровень изменился, передача считается успешной и передающий узел становится готовым к передаче следующего сообщения.

Исключения:

* если передающий узел обнаруживает, что при передаче сообщения модуль CAN МК сообщил о том, что произошла ошибка, он повторяет передачу данного сообщения;
* если передающий узел не обнаруживает изменение состояния дискретного порта на протяжении трех тактов вызова функции ***EcanAck\_\_run()*** после того, как модуль CAN сообщил об успешном окончании передачи, он повторяет передачу данного сообщения.

Описание реализации модуля

*EcanAck\_*\_ctor

Функция ***EcanAck\_\_ctor()*** инициализирует внутренние данные драйвера и настраивает указанные периферийные устройства МК.

Для настройки CAN модуля функция принимает параметры: *numPort*, *txSid*, *rxSid* и *mode*. Исходя из значения *numPort* выбирается переменная *dsPIC33\_Regs* или *dsPIC33\_Regs*, которая будет использоваться для доступа к SFR-регистрам соответствующего модуля CAN. Значения параметров *txSid* и *rxSid* записываются в соответствующие регистры TXF0SID и RXF0SID выбранного модуля.

Модуль CAN настраивается на тактирование от тактовой частоты МК Clk = Fcy\*1.(Fcy=7.3728\*4 Mhz). Работа модуля прекращается когда МК находится в Idle mode. Ширина перехода синхронизации настраивается равной 1Tq (0+1). Tq (квант времени) настраивается равным 2\*2\*Tosc = 135/2 = 67,5 нс, а *SEG1PH*, *SEG2PH* и *PRSEG* настраиваются равными 3Tq, 2Tq и 2Tq соответственно. Номинальное время передачи бита (Nominal Bit Time – NBT) составляет 8Tq и равно 1080 нс (921600 бит/с).

Для настройки дискретного порта вывода функция принимает параметры: *outTrisPort*, *outPinPort*, *outPinNum*. Для настройки дискретного порта ввода функция принимает параметры: *inTrisPort*, *inPinPort*, *inPinNum*. Данные параметры запоминаются во внутренней структуре данных и в дальнейшем используются для управления этими портами. Соответствующие порты настраиваются на ввод и вывод, и на них устанавливается значение ноль.

Функция возвращает указатель на структуру, в которой хранятся внутренние данные драйвера. В дальнейшем этот указатель должен передаться в качестве первого параметра для всех функций этого драйвера.

EcanAck\_get

Функция читает принятое сообщение из регистров *RX0B1*, *RX0B2*, *RX0B3*, *RX0B4* в буфер *array*, контролируя при этом его переполнение. Сбрасывает флаги *RX0IF* и *RXFUL* в регистрах *INTF* и *RX0CON*, а также признак готовности принятых данных. Функция возвращает длину принятых данных, которые были помещены в массив *array*.

EcanAck\_\_isInReady

Функция возвращает значение признака готовности принятых данных, который устанавливается функцией ***EcanAck\_run()***, когда модуль CAN принял сообщение с идентификатором, равным заданному при вызове конструктора в параметре *rxSid*. Когда сообщение принято, функция возвращает *true*. Если после инициализации модуля или последнего чтения новое сообщение принято не было, функция возвращает *false*.

EcanAck\_set

Функция копирует данные, принятые в параметре *array*, в регистры *TX0B1*, *TX0B2*, *TX0B3*, *TX0B4* соответствующего модуля CAN. Запоминает состояние дискретного входа, формирующего подтверждение приема, инициализирует счетчик ожидания подтверждения и инициирует передачу, установив бит *TXREQ* регистра *TX0CON* соответствующего модуля.

EcanAck\_isOutReady

Функция возвращает значения признака готовности модуля CAN к передаче нового сообщения. Функция возвращает *true* когда модуль готов к передаче или *false* когда модуль находится в процессе передачи, ожидании подтверждения приема или повторяет передачу сообщения о приеме которого вовремя не было получено подтверждение.

EcanAck\_run

Функция реализует прием и передачу сообщений, повтор передачи сообщений, при передаче которых возникли ошибки или не было получено подтверждение о приеме. Работа функции организована по принципу периодического опроса. Функция вызывается каждые 62,5 мкс, опрашивает состояние регистров модуля CAN и портов дискретного ввода/вывода на основании чего выполняет необходимые действия для реализации алгоритма обмена данными.

При каждом вызове функция проверяет бит *EWARN* (передатчик или приемник в состоянии ошибки) регистра *INTF* и, если он установлен, выполняется сброс модуля CAN. Сброс модуля осуществляется при помощи входа в режим конфигурации и возврата из него в рабочий режим.

Далее функция работает с приемником модуля CAN. Сначала проверяется наличие ошибок приема, проверяются биты *RX0OVR*, *RX1OVR*, *RXEP*, *RXWAR*, *IVRIF* регистра *INTF*. Если хоть один бит установлен, считается что произошла ошибка. Читаются регистры *RX0B1*, *RX0B2*, *RX0B3*, *RX0B4* и обнуляются биты *RX0IF* и *RXFUL*. Если ни один из битов ошибок не установлен, но установлены биты *RX0IF* или *RXFUL*, устанавливается признак готовности принятых данных и инвертируется состояние порта вывода для формирования подтверждения приема. Состояние признака готовности принятых данных можно прочитать при помощи функции ***EcanAck\_isInReady()***, а принятые данные можно прочитать из буфера приемника при помощи функции ***EcanAck\_get()***. Если данные не будут прочитаны до приема следующего сообщения, произойдет ошибка переполнения буфера 0 и данные будут утеряны, поэтому после вызова функции ***EcanAck\_run()*** обязательно должна следовать обработка принятых данных.

Далее следует обработка передачи данных, которая инициируется вызовом функции ***EcanAck\_set()***. Функция опрашивает бит *TXREQ* регистра *TX0CON* для того, чтобы дождаться когда передатчик модуля CAN окончит передачу данных, после чего проверяются биты ошибок передатчика *TXBO*, *TXEP* и *TXWAR*. Если при передаче данных возникла ошибка, устанавливается бит *ABAT* регистра *CTRL*, а при последующем вызове ***EcanAck\_run()*** передача инициируется заново, запоминается состояние дискретного входа, формирующего подтверждение приема, инициализируется счетчик ожидания подтверждения и инициируется передача, затем будет установлен бит *TXREQ* регистра *TX0CON*. Если ошибок не было, функция ожидает изменение состояния дискретного порта ввода на протяжении трех вызовов. Если состояние порта изменилось, устанавливается признак готовности передатчика к последующей передаче данных. Если не изменилось, осуществляется повтор передачи также как это выполняется при возникновении ошибки.

При выходе функция сбрасывает флаги прерываний в регистре *INTF* и флаг прерывания модуля CAN *C1IF* в регистре *IFS1* или *C2IF* в регистре *IFS2*, в зависимости от используемого модуля.

* + - 1. Модуль EepromDrv
         1. Назначение

Модуль EepromDrv предназначен для обеспечения программного интерфейса доступа к внешней памяти EEPROM.

Во EEPROM хранится:

* коды отказа;
* информация о версии ПО и дате выпуска прибора,
* служебная информация о работе ПО прибора.
  + - * 1. Состав модуля

Модуль состоит из одноимённого файла.

* + - * 1. Описание

Поддерживаемые функции модуля

Модуль предоставляет ряд программных интерфейсов для обеспечения следующих операций:

* чтение информации из внешней EEPROM;
* запись информации во внешнюю EEPROM.

Модель использования модуля

Перед использованием любой функции модуля необходимо проинициализировать его работу, вызвав функцию ***Eeprom\_ctor()***. Поскольку модуль использует компонент InterChannel, а при инициализации модуля EEPROM настраивается один из параметров синхронизации, данная функция должна вызываться после инициализации компонента InterChannel.

Для обеспечения работы в главном потоке периодически должна вызываться функция ***Eeprom\_run()***. В этой функции реализован автомат управления операциями чтения и записи в EEPROM. При каждом вызове функция выполняет один шаг любой операции.

Для обеспечения чтения данных из EEPROM необходимо использовать функцию ***Eeprom\_read()***, которая сразу же возвращает результат чтения. Операция чтения занимает около 2 мкс, а операция записи – около 4 мс.

Для обеспечения записи данных в EEPROM необходимо использовать функцию ***Eeprom\_write()***. Функция инициирует процессы стирания, потом записи. Узнать об окончании асинхронной операции можно при помощи функции ***Eeprom\_isReady()***.

* + - 1. Модуль I2Cdrv\_PIC33
         1. Назначение

Модуль I2Cdrv\_PIC33 реализует интерфейс шины I2C для работы с внешними устройствами, подключенные на эту шину.

* + - * 1. Описание

Драйвер обеспечивает операции чтения и записи в устройство на шине I2C. По вызову функции ***I2Cdrv\_ctor()*** проводится настройка модуля I2C МК. Настраивается на 7-битный адрес, скорость обмена задается значением для регистра BRG. Работа драйвера построена на опросе SFR-регистров МК. Опрос регистров осуществляется функцией ***I2Cdrv\_run(),*** реализующей автомат состояния работы с шиной.

* + - 1. Модуль posDetGen\_drv
         1. Назначение

Управление широтно-импульсной модуляцией сигнала контрольного генератора.

* + - * 1. Состав

posDetGen\_drv – интерфейсы и реализация драйвера

* + - * 1. Описание

Модуль осуществляет инициализацию ШИМ (модуль PWM1) в функции *PosDetGenDrv\_init*, включение и выключение ШИМ в функциях *PosDetGenDrv\_setEnable* и *PosDetGenDrv\_setDisable*, а также установка значений в регистре длительности импульса PDC1 в функции *PosDetGenDrv\_setValue.* Используется комплементарное управление транзисторными ключами. Частота PWM составляет 28 кГц

* + - 1. Модуль relayCtrlDrv
         1. Назначение.

Управление с ножек портов ввода-вывода реле, расположенных на модуле реле.

* + - * 1. Состав

relayCtrlDrv – интерфейсы и реализация драйвера.

* + - * 1. Описание

Модуль осуществляет:

* инициализацию пинов портов ввода-выода МК, с которых осуществляется управление реле, расположенных на модуле реле в функции RelayCtrlDrv\_ctor;
* переключение реле РПВ в функции RelayCtrlDrv\_switchRpv;
* переключение реле, использующихся при работе ПО в пятипроводной схеме контроля положения стрелки в функциях RelayCtrlDrv\_switchReceiverL1L2, RelayCtrlDrv\_switchReceiverL3L4, RelayCtrlDrv\_switchPhasesAB.
  + - 1. Модуль shuntShiftGen
         1. Назначение

Управление широтно-импульсной модуляцией сигнала трёхфазного генератора напряжения рабочих цепей.

* + - * 1. Описание

Модуль осуществляет инициализацию ШИМ (модули PWM4, PWM5, PWM6) в функции *ShuntShiftGenDrv\_init*, включение и выключение ШИМ, установку начальных значений в функциях *ShuntShiftGenDrv\_on* и *ShuntShiftGenDrv\_off*, а также установка значений в регистрах длительности импульсов PDC4, PDC5, PDC6 в функции *ShuntShiftGenDrv\_setValues.* Используется комплементарное управление транзисторными ключами. Частота PWM составляет 28 кГц.

* + - 1. Модуль relayCtrlDrv
         1. Назначение

Модуль relayCtrlDrv реализует драйвер управления реле РПВ.

* + - * 1. Описание

Модуль осуществляет инициализацию модуля реле, управление реле.

* + - 1. Модуль turnOffdsPIC33
         1. Назначение

Модуль turnOffsdPIC33 обеспечивает отключении периферии МК.

* + - * 1. Описание

При переходе в 3С все периферийные устройства МК отключаются, выводы переводятся в высокоимпедансное состояние.

* + 1. Systems
       1. Назначение

Каталог Systems содержит программные модули, которые используются компонентами ПО для их настройки в конкретном проекте. Модули содержат: описания типов данных, адреса в области памяти для хранения конкретных данных, количество тех или иных объектов, которое необходимо знать на этапе компиляции, чтобы не использовать динамическое выделение памяти и т.д.

* + - 1. Состав каталога

В состав каталога входят следующие модули:

* *AnalogInputId* − модуль идентификаторов входных аналоговых сигналов;
* *asserts* − управляемые утверждения (assert);
* *asserts\_ex* − расширенные утверждения (assert);
* *defCompil* − определение компилятора;
* *InterChannelId* − модуль идентификаторов параметров МКО;
* *nameHex* – определение имени прибора;
* *PrtotectionState\_codes* − определение кодов защитных отказов;
* *version* – дата и версия сборки.
  + - 1. Описание

При разработке ПО применен модульный подход, ориентированный на использование повторно применяемого кода. Однако разные проекты могут требовать разного количества объектов того или иного типа, разной длины очередей и т.д. Для создания в ПО разного количества объектов, задания различной длины для очередей, применяя язык С, можно использовать:

* динамическое выделение памяти;
* создание объектов пользовательским ПО и передачу их целевому компоненту;
* использование интерфейсных файлов с определенными необходимыми значениями, которые используются компонентами для конфигурирования и применяются на этапе компиляции проекта.

В данном ПО применен третий подход: в структуру проекта введен каталог *systems*, в котором располагаются файлы конфигурации компонент ПО. Кроме конфигурационных файлов каталог содержит базовые модули, на которые опираются при разработке прикладных компонентов ПО (далее по тексту – системные компоненты). Таким образом модули, расположенные в этом каталоге, можно разделить на две категории:

* конфигурационные файлы компонент;
* системные модули.
  + - 1. Модуль AnalogInputId
         1. Назначение

Конфигурационный модуль аналогового измерителя. Используется драйвером АЦП (см. **Ошибка! Источник ссылки не найден.**), компонентами AnalogInput и AnalogMeasurement (см. 4.3.2).

* + - * 1. Интерфейсы

Макроопределения

Модуль содержит макроопределение *AVERAGE\_WINDOW\_SIZE*, которое определяет количество отсчетов за период на протяжении которого производится расчет среднего и среднеквадратического значений.

Перечисляемые константные типы

Модуль содержит константу *AnalogInputDrvQueueSize*, определяющую размер очереди драйвера, в которой хранятся результаты отсчетов до тех пор, пока они не будут обработаны в главном потоке программы.

Модуль содержит перечисляемый тип *AnalogInputChannels*, который определяет список аналоговых параметров, измеряемы модулем *AnalogInput*.

* + - 1. Модуль asserts
         1. Назначение

Утверждение (assert) – это определение, которое может быть использовано в программах, написанных на языках C и C++, и позволяет проверить предположения о программе и ее поведении. Каждое утверждение содержит логическое выражение, которое, по мнению разработчика, будет верным в момент выполнения. В противном случае, будет вызвана функция обработки исключения. Проверяя, что логическое выражение на самом деле верно, утверждение (assert) подтверждает предположения (ожидания) о поведении программы, увеличивая уверенность в том, что программа не содержит ошибок.

Данный модуль реализует часть требований раздела 5 документа «Архитектура ПО».

* + - * 1. Интерфейсы

Макроопределения

* *ASSERT\_DEF\_CODE* – код отказа в случае вызова *ASSERT* или *ERROR* без идентификатора.
* *CASSERT\_INC* – значение, на которое увеличивается счетчик при проверке *CASSERT* и *CASSERT\_ID* в случае, когда проверяемое условие *FALSE*.
* *CASSERT\_DEC* – значение, на которое уменьшается счетчик при проверке *CASSERT* и *CASSERT\_ID* в случае, когда проверяемое условие *TRUE*.

Макрофункции

* *SET\_CODE\_ID(group\_, id\_)* − получение кода отказа, используя тип отказа и идентификатор.

Параметры:

*group\_* − тип отказа;

*id\_* − идентификатор отказа.

Возвращает:

Код отказа, который в старшем байте содержит тип, а в младшем – идентификатор.

* *GET\_GROUP\_FROM\_CODE(code\_)* − получение типа отказа, используя код отказа.

Параметры:

*code\_* − код отказа.

Возвращает:

Тип отказа.

* *GET\_ID\_FROM\_CODE(code\_)* − получение идентификатора отказа, используя код отказа.

Параметры:

*code\_* − код отказа.

Возвращает:

Идентификатор отказа.

* *ERROR\_ID(group\_, id\_)* − ошибка, с использованием типа и идентификатора.

Параметры:

*group\_* − тип отказа;

*id\_* − идентификатор отказа.

* *ERROR( )* − ошибка без идентификатора. В этом случае, для параметров *group\_* и *id\_* используется значение *ASSERT\_DEF\_CODE*.
* *ASSERT\_ID(group\_, id\_, test\_)* − проверка утверждения с использованием типа и идентификатора исключения.

Параметры:

*group\_* − тип отказа;

*id\_* − идентификатор отказа;

*test\_* − проверяемое условие.

* *ASSERT(test\_)* − проверка утверждения без идентификатора. В этом случае, для параметров *group\_* и *id\_* используется значение *ASSERT\_DEF\_CODE*.

Параметры:

*test\_* − проверяемое условие.

* *CASSERT\_ID(group\_, id\_, counter\_, maxcount\_, test\_)* − проверка утверждения со счетчиком, с использованием типа и идентификатора исключения.

Параметры:

*group\_* − тип отказа;

*id\_* − идентификатор отказа;

*counter\_* − счетчик;

*maxcount\_* − максимально допустимое значение счетчика;

*test\_* − проверяемое условие.

* *CASSERT(counter\_, maxcount\_, test\_)* − проверка утверждения со счетчиком без идентификатора. В этом случае, для параметров *group\_* и *id\_* используется значение *ASSERT\_DEF\_CODE*.

Параметры:

*counter\_* − счетчик;

*maxcount\_* − максимально допустимое значение счетчика;

*test\_* − проверяемое условие.

* *DERROR\_ID(group\_, id\_)* − дополнительное макроопределение: ошибка с использованием типа и идентификатора исключения.

Параметры:

*group\_* − тип отказа;

*id\_* − идентификатор отказа.

* *DERROR( )* − дополнительное макроопределение: ошибка без идентификатора. В качетве параметров *group\_* и *id\_* используется значение *ASSERT\_DEF\_CODE*.
* *DASSERT\_ID(group\_, id\_, test\_)* − дополнительное макроопределение, проверяющее условие с использованием типа и идентификатора исключения.

Параметры:

*group\_* − тип отказа;

*id\_* − идентификатор отказа;

*test\_* − проверяемое условие.

* *DASSERT(test\_)* − дополнительное макроопределение, проверяющее условие без идентификатора. В качетве параметров *group\_* и *id\_* используется значение *ASSERT\_DEF\_CODE*.

Параметры:

*test\_* − проверяемое условие.

* *DCASSERT\_ID(group\_, id\_, counter\_, maxcount\_, test\_)* − дополнительное макроопределение, проверяющее условие со счетчиком, с использованием типа и идентификатора исключения.

Параметры:

*group\_* − тип отказа;

*id\_* − идентификатор отказа;

*counter\_* − счетчик;

*maxcount\_* − максимально допустимое значение счетчика;

*test\_* − проверяемое условие.

* *DCASSERT(counter\_, maxcount\_, test\_)* − дополнительное макроопределение, проверяющее условие со счетчиком, без идентификатора. В качетве параметров *group\_* и *id\_* используется значение *ASSERT\_DEF\_CODE*.

Параметры:

*counter\_* − счетчик;

*maxcount\_* − максимально допустимое значение счетчика;

*test\_* − проверяемое условие.

Функции

* *void sysAssertException (char const \*const file, int16\_t line, int16\_t group, int16\_t id)* − обработка исключений.

Параметры:

*file* – имя файла, в котором было обнаружено некорректное поведение системы;

*line* – номер строки, в которой было обнаружено некорректное поведение системы, или тип и идентификатор исключения;

*group* – код ошибки;

*id* – идентификатор ошибки.

* + - * 1. Описание

Общие сведения

Модуль asserts содержит макроопределения для проверки утверждений, приводящих к вызову функции обработки исключений ***sysAssertException()*** или для безусловного вызова этой функции. Также этот модуль содержит реализацию функции ***sysAssertException()***.

Модуль содержит три типа макроопределений *ASSERT*:

* *ERROR* – безусловный вызов функции обработки исключения;
* *ASSERT* – проверка утверждения и вызов функции обработки исключения в случае его несоответствия;
* *CASSERT* – проверка утверждения со счетчиком и вызов функции обработки исключения когда счетчик достигнет максимального значения.

Каждый тип имеет две версии: с идентификатором и без идентификатора. Версии идентификаторов имеют те же имена c суффиксом «\_ID». Всего образуется шесть макроопределений. Каждому такому макроопределению соответствует еще одна отладочная версия. Отладочные макроопределения имеют соответствующее имя с приставкой «D». Отладочные макроопределения отличаются тем, что могут быть отключены или включены отдельной директивой *ENABLED\_DEBUG\_ASSERT*.

Сборка модуля может осуществляться в нескольких конфигурациях. Управление сборкой выполняется при помощи специальных директив.

Функция обработки исключений

Все типы макроопределений ASSERT, в случае несоответствия ожиданиям, приводят к вызову функции обработки исключений***sysAssertException()***.

Данная функция имеет две реализации: отладочную и штатную. Выбор версии реализации осуществляется на этапе компиляции при помощи директивы *DEBUG\_EXCEPTION*, которая включает отладочную версию. Штатная версия вызывает функцию ***ProtectionState\_run()*** компонента ProtectionState (см. **Ошибка! Источник ссылки не найден.**), которой передает значение кода ЗО, полученного из значений параметров *group* и *id* при помощи макроопределения *SET\_CODE\_ID*.

Отладочная версия функции не вызывает функцию ***ProtectionState\_run()***, а при помощи макроопределений из состава компонента UNITY, вызывает ошибку теста с выводом информации о месте, где было вызвано исключение (номер строки и имя файла). В отладочном режиме доступны две глобальные переменные *exceptionExpectedErrorCode* и *exceptionErrorCodeIsExpected*. Эти переменные используются при построении тестов, успех которых заключается в вызове конкретного исключения. Для того, чтобы тест при вызове исключения не был забракован, необходимо переменной *exceptionExpectedErrorCode* присвоить значение ожидаемого исключения. Значение ожидаемого исключения формируется из номера группы исключений и идентификатора исключения при помощи макроопределения *SET\_CODE\_ID(group, id)*. Если произойдет вызов указанного исключения, функция ***sysAssertException()*** не забракует тест, а установит глобальную переменную *exceptionErrorCodeIsExpected* в значение, отличное от нуля.

Примечание.

В данном проекте отладочная версия этой функции не используется, поэтому компонент UNITY не включен в данный проект.

Типы макроопределений

Тип *ERROR* (безусловный вызов функции обработки исключения) используется тогда, когда программа в принципе не должна попадать в ветку, в которой используется данное макроопределение.

Тип *ASSERT* (проверка утверждения) в качестве параметра принимает результат булевого выражения, которое может принимать значения *true* или *false*. Если результат имеет значение *false*, макроопределение выполняет вызов функции обработки исключения.

Тип *CASSERT* (проверка утверждения со счетчиком) кроме результата булевого выражения, принимает счетчик неверных утверждений и максимальное значение при достижении которого будет вызвана функция обработки исключения. Если результат булевого выражения имеет значение *false*, при каждой проверке счетчик увеличивается, если *true* – уменьшается. Счетчик считает от нуля до максимального значения. Значение, на которое счетчик увеличивается, определено макроопределением *CASSERT\_INC* и в данном проекте составляет 2. Значение, на которое счетчик уменьшается, определено макроопределением *CASSERT\_DEC* и в данном проекте составляет 1.

Версии макроопределений

В проекте присутсвует 2 версии макроопределений: с идентификатором и без идентификатора. Версия макроопределений с идентификатором от версии без идентификатора отличаются тем, что позволяют пользователю указать специфический идентификатор, по которому будет сформирован код отказа при переходе прибора в ЗС. Этот идентификатор впоследствии может позволить установить причину перехода прибора в ЗС. Версия без идентификатора также формирует код отказа, но для его формирования использует значение по умолчанию, определенное в макроопределении *ASSERT\_DEF\_CODE*.

Отладочная версия макроопределений полностью повторяет обычную с той разницей, что позволяет включать и отключать эти макроопределения из кода отдельно от других. Такие макроопределения могут быть использованы для проведения более тщательных проверок во время тестирования или испытаний и отключены в рабочей версии проекта для экономии процессорного времени.

Управление сборкой

По умолчанию выполняется сборка с включенными макроопределениями *ASSERT*, выключенными их отладочными версиями и передачей номера строки, на которой расположено макроопределение, вызывающее исключение функции обработки исключений.

Сборка проекта может производиться с отключением всех макроопределений *ASSERT*. Такая версия сборки может понадобиться при регулировке аппаратуры или поиске неисправностей. Чтобы отключить все макроопределения, вызывающие исключения, необходимо в проекте при сборке определить директиву *DISABLED\_ASSERT*. В случае использования директивы *DISABLED\_ASSERT*, сборка проекта будет сопровождаться выводом предупредительных сообщений «The project is compiled with the directive DISABLED\_ASSERT».

Для включения отладочных исключений необходимо в проекте определить директиву *ENABLED\_DEBUG\_ASSERT*.

Функция обработки исключений в качестве одного из параметров принимает номер строки, на которой вызвана обработка исключения. Для формирования номера строки используется стандартное предопределенное макроопределение языка С *\_\_LINE\_\_*. При использовании номера строки после форматирования кода без его функционального изменения в результате переноса макроопределения на другую строку объектный файл может отличаться. В случаях, когда это имеет значение, может быть осуществлена сборка проекта без включения номера строки. Для этого необходимо определить в проекте директиву *DONT\_SAVE\_LINE\_NUMBER*. После определения этой директивы при сборке проекта будет выдаваться предупредительное сообщение «The project is compiled with the directive DONT\_SAVE\_LINE\_NUMBER». Вывод этого сообщения можно отключить, определив в проекте директиву *IGNORE\_DONT\_SAVE\_LINE\_NUMBER\_WARNING*.

При проведении модульного тестирования с использованием компонента UNITY можно включить в сборку отладочную версию функции обработки исключений. Для этого в проекте необходимо определить директиву *DEBUG\_EXCEPTION*.

* + - 1. Модуль asserts\_ex
         1. Назначение

Расширенная версия модуля asserts в случае возникновения ошибки позволяет пользователю указать четыре дополнительных параметра, которые будут сохранены в EERPOM и позволят получить больше информации о возникшей ошибке.

* + - * 1. Интерфейсы
* *ERROR\_EX\_ID(group\_, id\_ , p1\_, p2\_, p3\_, p4\_)* − ошибка с использованием типа и идентификатора.

Параметры:

*group\_* − тип отказа;

*id\_* − идентификатор отказа;

*p1\_* − параметр отказа 1;

*p2\_* − параметр отказа 2;

*p3\_* − параметр отказа 3;

*p4\_* − параметр отказа 4.

* *ERROR\_EX(p1\_, p2\_, p3\_, p4\_)* − ошибка без идентификатора. В этом случае, для параметров *group\_* и *id\_* используется значение *ASSERT\_DEF\_CODE*.

Параметры:

*p1\_* − параметр отказа 1;

*p2\_* − параметр отказа 2;

*p3\_* − параметр отказа 3;

*p4\_* − параметр отказа 4.

* *ASSERT\_EX\_ID(group\_, id\_, test\_, p1\_, p2\_, p3\_, p4\_)* − проверка утверждения с использованием типа и идентификатора исключения.

Параметры:

*group\_* − тип отказа;

*id\_* − идентификатор отказа;

*test\_* − проверяемое условие;

*p1\_* − параметр отказа 1;

*p2\_* − параметр отказа 2;

*p3\_* − параметр отказа 3;

*p4\_* − параметр отказа 4.

* *ASSERT\_EX(test\_, p1\_, p2\_, p3\_, p4\_)* − проверка утверждения без идентификатора. В этом случае, для параметров *group\_* и *id\_* используется значение *ASSERT\_DEF\_CODE*.

Параметры:

*test\_* − проверяемое условие;

*p1\_* − параметр отказа 1;

*p2\_* − параметр отказа 2;

*p3\_* − параметр отказа 3;

*p4\_* − параметр отказа 4.

* *CASSERT\_EX\_ID(group\_, id\_, counter\_, maxcount\_, test\_, p1\_, p2\_, p3\_, p4\_)* − проверка утверждения со счетчиком, с использованием типа и идентификатора исключения.

Параметры:

*group\_* − тип отказа;

*id\_* − идентификатор отказа;

*counter\_* − счетчик;

*maxcount\_* − максимально допустимое значение счетчика;

*test\_* − проверяемое условие;

*p1\_* − параметр отказа 1;

*p2\_* − параметр отказа 2;

*p3\_* − параметр отказа 3;

*p4\_* − параметр отказа 4.

* *CASSERT\_EX(counter\_, maxcount\_, test\_, p1\_, p2\_, p3\_, p4\_)* − проверка утверждения со счетчиком без идентификатора. В этом случае, для параметров *group\_* и *id\_* используется значение *ASSERT\_DEF\_CODE*.

Параметры:

*counter\_* − счетчик;

*maxcount\_* − максимально допустимое значение счетчика;

*test\_* − проверяемое условие;

*p1\_* − параметр отказа 1;

*p2\_* − параметр отказа 2;

*p3\_* − параметр отказа 3;

*p4\_* − параметр отказа 4.

* *DERROR\_EX\_ID(group\_, id\_, p1\_, p2\_, p3\_, p4\_)* − дополнительное макроопределение: расширенная ошибка с использованием типа и идентификатора.

Параметры:

*group\_* − тип отказа;

*id\_* − идентификатор отказа;

*p1\_* − параметр отказа 1;

*p2\_* − параметр отказа 2;

*p3\_* − параметр отказа 3;

*p4\_* − параметр отказа 4.

* *DERROR\_EX(p1\_, p2\_, p3\_, p4\_)* − дополнительное макроопределение: расширенная ошибкаошибка без идентификатора. В этом случае, для параметров *group\_* и *id\_* используется значение *ASSERT\_DEF\_CODE*.

Параметры:

*p1\_* − параметр отказа 1;

*p2\_* − параметр отказа 2;

*p3\_* − параметр отказа 3;

*p4\_* − параметр отказа 4.

* *DASSERT\_EX\_ID(group\_, id\_, test\_, p1\_, p2\_, p3\_, p4\_)* − дополнительное макроопределение, проверяющее условие с использованием типа и идентификатора исключения.

Параметры:

*group\_* − тип отказа;

*id\_* − идентификатор отказа;

*test\_* − проверяемое условие;

*p1\_* − параметр отказа 1;

*p2\_* − параметр отказа 2;

*p3\_* − параметр отказа 3;

*p4\_* − параметр отказа 4.

* *DASSERT\_EX(test\_, p1\_, p2\_, p3\_, p4\_)* − дополнительное макроопределение, проверяющее условие без идентификатора. В этом случае, для параметров *group\_* и *id\_* используется значение *ASSERT\_DEF\_CODE*.

Параметры:

*test\_* − проверяемое условие;

*p1\_* − параметр отказа 1;

*p2\_* − параметр отказа 2;

*p3\_* − параметр отказа 3;

*p4\_* − параметр отказа 4.

* *DCASSERT\_EX\_ID(group\_, id\_, counter\_, maxcount\_, test\_, p1\_, p2\_, p3\_, p4\_)* − дополнительное макроопределение, проверяющее условие со счетчиком, с использованием типа и идентификатора исключения.

Параметры:

*group\_* − тип отказа;

*id\_* − идентификатор отказа;

*counter\_* − счетчик;

*maxcount\_* − максимально допустимое значение счетчика;

*test\_* − проверяемое условие;

*p1\_* − параметр отказа 1;

*p2\_* − параметр отказа 2;

*p3\_* − параметр отказа 3;

*p4\_* − параметр отказа 4.

* *DCASSERT\_EX(counter\_, maxcount\_, test\_, p1\_, p2\_, p3\_, p4\_)* − дополнительное макроопределение, проверяющее условие со счетчиком без идентификатора. В этом случае, для параметров *group\_* и *id\_* используется значение *ASSERT\_DEF\_CODE*.

Параметры:

*counter\_* − счетчик;

*maxcount\_* − максимально допустимое значение счетчика;

*test\_* − проверяемое условие;

*p1\_* − параметр отказа 1;

*p2\_* − параметр отказа 2;

*p3\_* − параметр отказа 3;

*p4\_* − параметр отказа 4.

* + - * 1. Описание

Модуль содержит новую версию с четырьмя дополнительными параметрами каждого макроопределения модуля *ASSERT*. Эти параметры, в случае возникновения ошибки, при помощи функции ***BlackBox\_saveAddCodeProtectionState()*** сохраняются модулем BlackBox (см. **Ошибка! Источник ссылки не найден.**) перед вызовом функции обработки исключения.

При определении в проекте директивы *DEBUG\_EXCEPTION* данные параметры не используются.

* + - 1. Модуль defCompil
         1. Назначение

Модуль определяет директивы *COMPIL\_C30* и *FLASH*, которые используются для определения типов, неподдерживаемых компиляторам XC16.

* + - * 1. Назначение

Конфигурационный модуль модуля InterChannel. Содержит список идентификаторов синхронизируемых параметров и их количество. Идентификаторы используются для идентификации синхронизируемого параметра, количество используется для выделения памяти на этапе компиляции для нужного количества объектов, а также для проверки диапазона при обращении к компоненту InterChannel.

* + - * 1. Интерфейсы

Модуль содержит константы типа *InterChannelId*, которые определяют список идентификаторов синхронизируемых параметров и их количество.

* + - 1. Модуль nameHex
         1. Назначение

Модуль содержит определение имени прибора для формирования имени загрузочного модуля. Имя определено в константе acNameHex как "ОКПС-Е-К".

* + - 1. Модуль ProtectionState\_codes
      2. Назначение

Модуль содержит список констант, определяющих коды отказа.

* + - 1. Модуль version
      2. Назначение

Модуль содержит список констант, определяющих версию ПО и дату сборки.

* + 1. Testing

Каталог содержит модули, обеспечивающие тестирование МК. В процессе работы подвергаются непрерывному тестированию ОЗУ, FLASH-память программ и регистры общего назначения. Также выполняется контроль числа вызванных функций за миллисекундный цикл работы.

* + - 1. Модуль CheckCallFunctions

Предназначен для контроля числа вызванных функций за миллисекундный цикл. Если число вызванных функция в МК Master и Slave различается, прибор переходит в 3C с кодом *ePS\_ControlMkFaultCallFunctions*.

* + - 1. Модуль CheckCPU

Предназначен для контроля служебных регистров ALU процессора и регистров общего назначения W0…W15.

* + - 1. Модуль CheckFlashChecksum

Предназначен для вычисления контрольной суммы FLASH памяти программ процессора. Вычисленная контрольная сумма сравнивается с образцовой, посчитанной на этапе компиляции и хранящейся в специальных ячейках User ID memory. Так же контрольная сумма синхронизируется со смежным процессором. При расхождении в контрольных суммах происходит переход в 3С с кодом *eICId\_ControlMK\_Flash*.

* + - * 1. Описание

Контрольная сумма вычисляется путём арифметического суммирования всех байт, находящихся в памяти программ процессора. Начальный адрес памяти программ определён константой *FLASH\_ADDRESS\_START*, размер памяти программ константой *FLASH\_PROG\_SIZE*. По завершению суммирования всех ячеек памяти, к контрольной сумме содержимое шести добавляется ячеек конфигурации: *FGS*, *FOSCSEL*, *FOSC*, *FWDT*, *FPOR*, *FICD, BASE*. Контрольная сумма (старший и младший байты), рассчитанная на этапе компиляции, хранится в специальных ячейках FLASH-памяти – *FUID0* и *FUID1*. Контрольная сумма, полученная при проверке FLASH-памяти должна совпадать с содержимым ячеек *FUID0* и *FUID1*. Если содержимые разнятся, происходит переход в 3С с кодом *ePS\_ControlMkFaultFlash*.

* + - 1. Модуль CheckRAM

Предназначен для контроля исправности RAM процессора.

* + - 1. Модуль CheckRegister

Предназначен для контроля исправности регистров периферийных устройств процессора.

* + - 1. Модуль ControlMK

Модуль ControlMK является главным модулем проверок МК. В его задачи входят вызов функций контроля, проверок и контроль совпадения обобщенных результатов контроля по двум МК.

* + 1. Utility

Каталог Utility содержит программные модули и макроопределения, которые используются компонентами ПО.

* + - 1. Модуль CopySwap

Предназначен для копирования одного массива в другой с заменой информации. В результате работы массив, находящийся по адресу *pDest* заменяется на массив по адресу *pSrc*. А массив, находящийся по адресу *pSrc* заменяется на массив по адресу *pDest*. Длина массива передаётся в качестве параметра *size*.

* + - 1. Модуль SqrtTopt

Предназначен для быстрого вычисления квадратного корня.

* + - 1. Модуль synchronizeChange

Предназначен для синхронизации параметра по изменению.

* + - 1. Модуль wait

Модуль представляет собой набор функций, в которых реализуют задержки на 1 мкс, 62 мкс, 125 мкс и 1 мс.

1. Список литературы
2. ЕИУС.хххххх.ххх 01 91 01 01 Архитектура ПО ОКПС-Е-К.
3. DS70580B\_*Section 07 Oscillator.pdf*.
4. Схема электрическая принципиальная «ТЭЗ ОКПС Э2017-06-037.100 Э45.sch»
5. Схема электрическая принципиальная «Модуль генератора Э2017-06-037.120 Э42.sch»