

УТВЕРЖДЕН
RU.СГМА.00045 13 01-ЛУ

БЕЗОПАСНЫЙ ЛОКОМОТИВНЫЙ ОБЪЕДИНЕННЫЙ
КОМПЛЕКС – МАСШТАБИРУЕМЫЙ БЛОК-М
ПРОГРАММНЫЙ МОДУЛЬ БЛОКА СОГЛАСОВАНИЯ С ДАТЧИКАМИ
УГЛА ПОВОРОТА
ПРОГРАММНОЕ ОБЕСПЕЧЕНИЕ

Описание программы

RU.СГМА.00045 13 01

Листов 40

Изм. № подл	Подп. и дата
Взам. инв. №	Изм. № подл
Подп. и дата	

АННОТАЦИЯ

В настоящем документе представлены условия применения, назначение и логическая структура программного обеспечения модуля блока согласования с датчиками угла поворота БС-ДПС.

СОДЕРЖАНИЕ

Аннотация	2
1 Общие сведения.....	4
2 Функциональное назначение	5
3 Описание логической структуры.....	6
4 Используемые технические средства.....	29
5 Вызов и загрузка.....	30
6 Входные и выходные данные	31
Приложение 1 (Справочное) Перечень принятых сокращений	37
Приложение 2 (Обязательное) Сборка программного модуля БС-ДПС	38
Лист регистрации изменений.....	40

1 ОБЩИЕ СВЕДЕНИЯ

Наименование программы – Программное обеспечение модуля блока согласования с датчиками угла поворота БС-ДПС безопасного локомотивного объединенного комплекса – масштабируемого БЛОК-М (далее по тексту – ПО БС-ДПС).

Обозначение программы - RU.СГМА.00045.

Язык программирования – AVR Assembler.

ПО БС-ДПС предназначено для работы в составе комплекса БЛОК-М и иных локомотивных систем. Программа записана во внутреннюю память микроконтроллера типа AT90CAN128. Программное окружение для работы ПО БС-ДПС не требуется.

Перечень принятых сокращений приведен в приложении 1.

Сборка программного модуля БС-ДПС приведена в приложении 2.

2 ФУНКЦИОНАЛЬНОЕ НАЗНАЧЕНИЕ

Основной функциональностью блока ПО БС-ДПС является измерение параметров движения: пройденного пути, скорости и ускорения. Эти величины блок определяет, основываясь на анализе сигналов, приходящих от датчика ДПС.

Дополнительной функциональностью блока является определение местоположения изостыка, основанное на расшифровке кодов КПТ по огибающей АЛСН. Изостыком считается место, в котором произошла смена типа КПТ.

3 ОПИСАНИЕ ЛОГИЧЕСКОЙ СТРУКТУРЫ

Проект ПО БС-ДПС DPS_CAN.FLA V.8 включает в себя 157 файлов:

Имя файла	Назначение файла
1 avrBS-DPS.cpp	Основной файл, подключающий и запускающий всё
2 ButtonPressPolling.h	Опрос, нажата ли кнопка
3 CanDescriptors.h	Список используемых дескрипторов CAN
4 debug.h	Функция для отправки диагностики в CAN
5 BusOffDelay.cpp	Пытается "переждать" BusOff, прежде чем перезагружаться
6 BusOffDelay.h	Пытается "переждать" BusOff, прежде чем перезагружаться
7 ExceptionCollector.cpp	Обработчик исключительных ситуаций
8 ExceptionCollector.h	Обработчик исключительных ситуаций
9 IDiagnostic.h	Список кодов ошибок для диагностических сообщений
10 MyDiagnostic.cpp	Отправляет диагностические сообщения в CAN
11 MyDiagnostic.h	Отправляет диагностические сообщения в CAN
12 RestartReasonStore.cpp	Сохраняет код ситуации, если не может отправить в CAN
13 RestartReasonStore.h	Сохраняет код ситуации, если не может отправить в CAN
14 DiameterCorrector.cpp	Базовый класс коррекции диаметра банджа
15 DiameterCorrector.h	Базовый класс коррекции диаметра банджа
16 IDiameterCorrectionCalculator.h	Интерфейс алгоритма вычисления диаметра
17 PerMillimeterDiameterCorrectionCalculator.cpp	Реализация алгоритма вычисления диаметра по одному мм
18 PerMillimeterDiameterCorrectionCalculator.h	Реализация алгоритма вычисления диаметра по одному мм
19 dps.h	Основной класс скоростемера
20 both-break-detector.h	Контролирует "обрыв двух датчиков"
21 chooser.h	Выбирает из двух датчиков один
22 dimetior.h	Обработка сигналов ДПС
23 ec-adjust.h	Подстройка под электронную карту

Имя файла	Назначение файла
24 ExchangeStateMachine.cpp	Синхронизация полукомплектов при старте
25 ExchangeStateMachine.h	Синхронизация полукомплектов при старте
26 hw_defines.h	Логические имена для физических портов
27 id.h	Хранит ID во флеше
28 kpt.h	Дешифрация сигналов КПП
29 MphEepromMemory.cpp	Реализация IMemory для МПХ на EEPROM
30 MphEepromMemory.h	Реализация IMemory для МПХ на EEPROM
31 MphExchangeStateMachine.cpp	Синхронизация МПХ при старте
32 MphExchangeStateMachine.h	Синхронизация МПХ при старте
33 CanPeriodicalTransmitter.cpp	Отправляет в CAN периодические сообщения МПХ
34 CanPeriodicalTransmitter.h	Отправляет в CAN периодические сообщения МПХ
35 CanRequestTransponder.cpp	Отвечает на запросы МПХ
36 CanRequestTransponder.h	Отвечает на запросы МПХ
37 CellsToCanWriter.cpp	Отправляет значения всех ячеек МПХ в CAN
38 CellsToCanWriter.h	Отправляет значения всех ячеек МПХ в CAN
39 CellIdentifier.h	Список использованных ячеек МПХ
40 Cell.h	Ячейка МПХ
41 CellManagerInitialRequesterDecorator.cpp	Опрашивает все ячейки при старте и уведомляет подписчиков ICellManager
42 CellManagerInitialRequesterDecorator.h	Опрашивает все ячейки при старте и уведомляет подписчиков ICellManager
43 CellManagerNiiasDecorator.cpp	Подменяет значения ячеек в соответствии с алгоритмом НИИАС
44 CellManagerNiiasDecorator.h	Подменяет значения ячеек в соответствии с алгоритмом НИИАС
45 CellManagerReservingDecorator.cpp	Резервирует каждую ячейку МПХ
46 CellManagerReservingDecorator.h	Резервирует каждую ячейку МПХ
47 CellRequester.cpp	Опрашивает список ячеек
48 CellRequester.h	Опрашивает список ячеек

Имя файла	Назначение файла
49 DeferredWriteCellManagerDecorator.cpp	Откладывает запись в ячейку МПХ, чтобы сократить количество записей
50 DeferredWriteCellManagerDecorator.h	Откладывает запись в ячейку МПХ, чтобы сократить количество записей
51 DifferentialDeferredWriteCell.h	Ячейка с условием записи по разности значений
52 ExternalMemoryCellStore.cpp	Определяет формат хранения ячейки в памяти
53 ExternalMemoryCellStore.h	Определяет формат хранения ячейки в памяти
54 ICellManager.h	Интерфейс работы с МПХ
55 ICellStore.h	Интерфейс хранилища для МПХ
56 IDDeferredWriteCell.h	Интерфейс ячейки с записью, отложенной по выполнению условия
57 OverallCellRequester.cpp	Опрашивает все ячейки МПХ
58 OverallCellRequester.h	Опрашивает все ячейки МПХ
59 QueuedCellManager.cpp	Обеспечивает очередь при работе с МПХ
60 QueuedCellManager.h	Обеспечивает очередь при работе с МПХ
61 QueuedCellManagerWithPostWriteOperations.cpp	Даёт возможность сделать ещё что-то после записи и до уведомления о завершении
62 QueuedCellManagerWithPostWriteOperations.h	Даёт возможность сделать ещё что-то после записи и до уведомления о завершении
63 Task.h	Задача в очереди МПХ
64 TaskSimplifyQueue.cpp	Очередь, которая удаляет дублирующиеся задачи
65 TaskSimplifyQueue.h	Очередь, которая удаляет дублирующиеся задачи
66 CanFrame.h	CAN-сообщение
67 ICanSender.h	Интерфейс для отправки в CAN
68 IMemory.h	Интерфейс для работы с памятью для МПХ
69 IScheduler.h	Планировщик
70 MemoryWaiter.cpp	Реализует асинхронную запись и чтение из памяти, дожидаясь её освобождения
71 MemoryWaiter.h	Реализует асинхронную запись и чтение из памяти, дожидаясь её освобождения
72 MphSautSynchronizer.cpp	Базовый класс для синхронизации МПХ и Карточки локомотива

Имя файла	Назначение файла
73 MphSautSynchronizer.h	Базовый класс для синхронизации МПХ и Карточки локомотива
74 NeighbourChecksumChecker.cpp	Сравнивает контрольные суммы данных МПХ между полукомплектами
75 NeighbourChecksumChecker.h	Сравнивает контрольные суммы данных МПХ между полукомплектами
76 OverallChecksum.cpp	Считает общую контрольную сумму по всем ячейкам МПХ
77 OverallChecksum.h	Считает общую контрольную сумму по всем ячейкам МПХ
78 WriteCounterProperty.cpp	Счётчик записей в МПХ, хранящийся на "размазанной" памяти
79 WriteCounterProperty.h	Счётчик записей в МПХ, хранящийся на "размазанной" памяти
80 SautCard.h	Карточка локомотива
81 SautCardMaker.cpp	Делает актуальную Карточку локомотива из памяти и МПХ
82 SautCardMaker.h	Делает актуальную Карточку локомотива из памяти и МПХ
83 SautConverter.cpp	Делает Карточку из МПХ
84 SautConverter.h	Делает Карточку из МПХ
85 SautLayout.h	Расположение данных Карточки локомотива в памяти
86 SautLayoutCrc Calculator.cpp	Считает контрольную сумму по Карточке
87 SautLayoutCrc Calculator.h	Считает контрольную сумму по Карточке
88 SautLoader.cpp	Считывает Карточку из памяти
89 SautLoader.h	Считывает Карточку из памяти
90 SautRevertConverter.cpp	Конвертация из Карточки в МПХ
91 SautRevertConverter.h	Конвертация из Карточки в МПХ
92 SpreadMemory Counter.cpp	Счётчик, размазанный по памяти для увеличения ресурса
93 SpreadMemory Counter.h	Счётчик, размазанный по памяти для увеличения ресурса
94 neutral-insertion.h	Определение положения нейтральной вставки
95 programming.h	Прошивка соседнего полукомплекта по SPI через RS-485

Имя файла	Назначение файла
96 SautCom.h	Связь по протоколу САУТ-485
97 SautDat.h	Хранение данных из линии связи САУТ
98 SilentModeCanSenderDecorator.h	Отключение передачи в CAN в "режиме тишины"
99 can-async.h	Драйвер CAN
100 can-dat.h	Хранилище данных, принятых по CAN
101 CanAsyncSender.h	Очередь отправки в CAN
102 CanFrame.h	CAN-сообщение
103 CanImmediatelySender.h	Немедленная блокирующая отправка в CAN
104 atomic.h	Позволяет красиво делать атомарные операции
105 Delegate.h	Делегаты без аллокации на куче
106 delegate.hpp	Делегаты без аллокации на куче
107 delegate_list.hpp	Вспомогательный класс для делегатов
108 dispatcher.h	Выполняет задачи из очереди
109 EepromMemoryAsyncWriter.cpp	Драйвер неблокирующей записи в EEPROM
110 EepromMemoryAsyncWriter.h	Драйвер неблокирующей записи в EEPROM
111 heap.h	Хранилище временных данных
112 ICanImmediatelySender.h	Интерфейс для немедленной блокирующей отправки в CAN
113 ICanSender.h	Интерфейс устройства, отправляющего в CAN
114 IClock.h	Интерфейс часов
115 int-typelist.h	Макросы для генерации списков Loki::Int2Type
116 interrupt-dynamic.cpp	Возможность установить обработчик прерывания во время исполнения
117 interrupt-dynamic.h	Возможность установить обработчик прерывания во время исполнения
118 io.h	Работа с регистрами и портами с помощью битовых полей
119 can.h	Описание регистров CAN на AVR
120 eeprom.h	Описание регистров EEPROM на AVR
121 hole-field.h	Вспомогательный класс для описания регистров с пропусками бит

Имя файла	Назначение файла
122 spi.h	Описание регистров SPI на AVR
123 timers.h	Описание регистров Timer/Counter на AVR
124 usart.h	Описание регистров UART на AVR
125 watchdog.h	Описание регистров watchdog на AVR
126 NullType.h	NullType для шаблонного программирования
127 Sequence.h	Последовательности для шаблонного программирования
128 static_check.h	Проверка условий на этапе компиляции
129 Typelist.h	Список типов для шаблонного программирования
130 TypelistMacros.h	Макросы для работы со списками типов
131 TypeManip.h	Манипуляции с типами для шаблонного программирования
132 TypeTraits.h	Информация о типах для шаблонного программирования
133 map-static.h	Ассоциативный массив с быстрым доступом
134 prog-spi.h	Прошивка AVR по SPI
135 scheduler.h	Планировщик, запускающий задачи по времени
136 smartdog.h	Уведомляет непосредственно перед срабатыванием watchdog
137 thread-safe.h	Потоко-безопасные переменные
138 timers.h	Таймеры/счётчики на AVR
139 types.h	Преобразования типов
140 universal.cpp	Разные универсальные удобные классы и функции
141 universal.h	Разные универсальные удобные классы и функции
142 alloca-wrapper.h	Подменяет alloca в зависимости от платформы
143 ArrayEnumerator.h	Enumerator, обертывающий Array
144 ByteManipulation.h	Побайтовые операции над переменными
145 CrcCalculator.h	Считает CRC
146 delegate.h	Делегаты без аллокации на куче
147 EventHandler.h	Частный вид делегата для "событий"

Имя файла	Назначение файла
148 IEnumerale.h	Интерфейс перечисляемого объекта
149 IEnumerator.h	Интерфейс перечислителя
150 IQueue.h	Интерфейс очереди
151 List.h	Реализация односвязного списка без аллокации
152 ListBase.cpp	Типонезависимая часть реализации списка
153 ListBase.h	Типонезависимая часть реализации списка
154 ListItem.h	Элемент списка
155 ListItemBase.h	Типонезависимая часть элемента списка
157 TypelessMemoryQueue.cpp	Типонезависимая часть очереди
158 TypelessMemoryQueue.h	Типонезависимая часть очереди
159 new.h	Оператор new для C++
160 QueuePoper.h	Кормит асинхронную операция элементами из очереди
161 UnreadyProperty.h	Свойство, которое может быть не готово на данный момент
157 TypelessMemoryQueue.cpp	Типонезависимая часть очереди

Проект ПО БС-ДПС DPS_CAN.FLA V.14 включает в себя 195 файлов:

Имя файла	Назначение файла
1 AuxResourceHandshaking.cpp	Тройное рукопожатие через AuxResource
2 AuxResourceHandshaking.h	Тройное рукопожатие через AuxResource
3 avrBS-DPS.cpp	Основной файл, подключающий и запускающий всё
4 ButtonPressPolling.h	Опрос, нажата ли кнопка
5 CanDescriptors.h	Список используемых дескрипторов CAN
6 debug.h	Функция для отправки диагностики в CAN
7 BusOffDelay.cpp	Пытается переждать BusOff, прежде чем перезагружаться
8 BusOffDelay.h	Пытается переждать BusOff, прежде чем перезагружаться

Имя файла	Назначение файла
9 ExceptionCollector.cpp	Обработчик исключительных ситуаций
10 ExceptionCollector.h	Обработчик исключительных ситуаций
11 IDiagnostic.h	Список кодов ошибок для диагностических сообщений
12 MyDiagnostic.cpp	Отправляет диагностические сообщения в CAN
13 MyDiagnostic.h	Отправляет диагностические сообщения в CAN
14 RestartReasonStore.cpp	Сохраняет код ситуации, если не может отправить в CAN
15 RestartReasonStore.h	Сохраняет код ситуации, если не может отправить в CAN
16 BothUnpluggedDetector.h	Контролирует обрыв двух ДПС
17 DiameterCorrector.cpp	Корректировка диаметра бандажа
18 DiameterCorrector.h	Корректировка диаметра бандажа
19 Distance.h	Конвертация расстояния во всевозможные единицы измерения
20 DistanceCounter.h	Счётчик пройденного пути и перемещения
21 Dps.h	Опрос ДПС
22 DpsChooser.h	Выбирает активный ДПС
23 DpsDirection.h	Вычисляет направление движения из направления вращения, позиции и прочего
24 DpsParameters.h	Рассчитывает длину импульса
25 DpsRotation.h	Направление вращения ДПС
26 EcAdjust.h	Подстройка под электронную карту
27 Hysteresis.h	Округление с гистерезисом
28 MalfunctionSaver.h	Сохраняет неисправность ДПС в энергонезависимую память
29 MileageSaver.h	Сохраняет пробег в энергонезависимую память
30 Ranged.h	Число с заданным диапазоном
31 SpeedAndDistanceModule.h	Модуль измерения пути и скорости
32 SpeedMeasurement.h	Результат измерения скорости
33 SpeedMeasurer.h	Рассчитывает скорость
34 Emulation.h	Имитация движения
35 CanLoaderFudp.h	Реализует протокол FUDP для прошивки по

Имя файла	Назначение файла
	CAN
36 CanLoaderIdProvider.h	Загружает идентификационные данные от загрузчика по CAN
37 IdAreaInFlash.h	Структура идентификационных данных, хранящихся вместе с прошивкой
38 Identification.h	Структура идентификационных данных
39 IdentificationProvider.h	Интерфейс для классов, предоставляющих Identification
40 ProgramControlSumChecker.h	Проверяет контрольную сумму прошивки
41 Saut100hIdProvider.h	Загружает идентификационные данные из области CAUT по 100-ому адресу
42 SysDiagnostics.h	Отвечает AUX_RESOURCE на SYS_DIAGNOSTICS
43 Handshaking.cpp	Алгоритм тройного рукопожатия
44 Handshaking.h	Алгоритм тройного рукопожатия
45 hw_defines.h	Логические имена для физических портов
46 interrupt-dynamic.h	Возможность установить обработчик прерывания во время исполнения
47 kpt.h	Дешифрация сигналов КПП
48 MphEepromMemory.cpp	Реализация IMemory для МПХ на EEPROM
49 MphEepromMemory.h	Реализация IMemory для МПХ на EEPROM
50 MphExchangeStateMachine.cpp	Синхронизация МПХ при старте
51 MphExchangeStateMachine.h	Синхронизация МПХ при старте
52 CanPeriodicalTransmitter.cpp	Отправляет в CAN периодические сообщения МПХ
53 CanPeriodicalTransmitter.h	Отправляет в CAN периодические сообщения МПХ
54 CanRequestTransponder.cpp	Отвечает на запросы МПХ
55 CanRequestTransponder.h	Отвечает на запросы МПХ
56 CellsToCanWriter.cpp	Отправляет значения всех ячеек МПХ в CAN
57 CellsToCanWriter.h	Выдаёт значения всех записанных ячеек МПХ в CAN
58 CellIdentifier.h	Список использованных ячеек МПХ

Имя файла	Назначение файла
59 Cell.h	Ячейка МПХ
60 CellManagerInitialRequesterDecorator.cpp	Опрашивает все ячейки при старте и уведомляет подписчиков ICellManager
61 CellManagerInitialRequesterDecorator.h	Опрашивает все ячейки при старте и уведомляет подписчиков ICellManager
62 CellManagerNiiasDecorator.cpp	Подменяет значения ячеек в соответствии с алгоритмом НИИАС
63 CellManagerNiiasDecorator.h	Подменяет значения ячеек в соответствии с алгоритмом НИИАС
64 CellRequester.cpp	Опрашивает список ячеек
65 CellRequester.h	Опрашивает список ячеек
66 DeferredWriteCellManagerDecorator.cpp	Откладывает запись в ячейку МПХ, чтобы сократить количество записей
67 DeferredWriteCellManagerDecorator.h	Откладывает запись в ячейку МПХ, чтобы сократить количество записей
68 DifferentialDeferredWriteCell.h	Ячейка с условием записи по разности значений
69 ExternalMemoryCellStore.cpp	Определяет формат хранения ячейки в памяти
70 ExternalMemoryCellStore.h	Определяет формат хранения ячейки в памяти
71 ICellManager.h	Интерфейс работы с МПХ
72 ICellStore.h	Интерфейс хранилища для МПХ
73 IDDeferredWriteCell.h	Интерфейс ячейки с записью, отложенной по выполнению условия
74 LightweightSpreadMemoryCounter.cpp	Облегчённая по занимаемой памяти версия счётчика, размазанного по памяти для увеличения ресурса
75 LightweightSpreadMemoryCounter.h	Облегчённая по занимаемой памяти версия счётчика, размазанного по памяти для увеличения ресурса
76 OverallCellRequester.cpp	Опрашивает все ячейки МПХ
77 OverallCellRequester.h	Опрашивает все ячейки МПХ
78 QueuedCellManager.cpp	Обеспечивает очередь при работе с МПХ
79 QueuedCellManager.h	Обеспечивает очередь при работе с МПХ
80 QueuedCellManagerWithPostWriteOperations.cpp	Даёт возможность сделать ещё что-то после записи и до уведомления о завершении
81 QueuedCellManagerWithPostWriteOperations.h	Даёт возможность сделать ещё что-то после записи и до уведомления о завершении

Имя файла	Назначение файла
tWriteOperations.h	записи и до уведомления о завершении
82 ReservingCellStore.cpp	Обёртка над ICellStore, резервирующая данные
83 ReservingCellStore.h	Обёртка над ICellStore, резервирующая данные
84 SpreadingCellStore.cpp	Обёртка над ICellStore, размазывающая избранные ячейки по нескольким, чтобы увеличить ресурс памяти
85 SpreadingCellStore.h	Обёртка над ICellStore, размазывающая избранные ячейки по нескольким, чтобы увеличить ресурс памяти
86 Task.h	Задача в очереди МПХ
87 TaskSimplifyQueue.cpp	Очередь, которая удаляет дублирующиеся задачи
88 TaskSimplifyQueue.h	Очередь, которая удаляет дублирующиеся задачи
89 CanFrame.h	CAN-сообщение
90 ICanSender.h	Интерфейс для отправки в CAN
91 IMemory.h	Интерфейс для работы с памятью для МПХ
92 MemoryWaiter.cpp	Реализует асинхронную запись и чтение из памяти, дожидаясь её освобождения
93 MemoryWaiter.h	Реализует асинхронную запись и чтение из памяти, дожидаясь её освобождения
94 MphFacade.cpp	Базовый класс МПХ
95 MphFacade.h	Базовый класс МПХ
96 MphSautSynchronizer.cpp	Базовый класс для синхронизации МПХ и Карточки локомотива
97 MphSautSynchronizer.h	Базовый класс для синхронизации МПХ и Карточки локомотива
98 NeighbourChecksumChecker.cpp	Сравнивает контрольные суммы данных МПХ между полукompлектами
99 NeighbourChecksumChecker.h	Сравнивает контрольные суммы данных МПХ между полукompлектами
100 OverallChecksum.cpp	Считает общую контрольную сумму по всем ячейкам МПХ
101 OverallChecksum.h	Считает общую контрольную сумму по всем ячейкам МПХ
102 SautCard.h	Карточка локомотива
103 SautCardMaker.cpp	Делает актуальную Карточку локомотива из

Имя файла	Назначение файла
	памяти и МПХ
104 SautCardMaker.h	Делает актуальную Карточку локомотива из памяти и МПХ
105 SautConverter.cpp	Делает Карточку из МПХ
106 SautConverter.h	Делает Карточку из МПХ
107 SautLayout.h	Расположение данных Карточки локомотива в памяти
108 SautLayoutCrcCalculator.cpp	Считает контрольную сумму по Карточке
109 SautLayoutCrcCalculator.h	Считает контрольную сумму по Карточке
110 SautLoader.cpp	Считывает Карточку из памяти
111 SautLoader.h	Считывает Карточку из памяти
112 SautRevertConverter.cpp	Конвертация из Карточки в МПХ
113 SautRevertConverter.h	Конвертация из Карточки в МПХ
114 UnwrittenCellInitializer.cpp	Записывает значение по умолчанию в ячейки МПХ, в которых они всегда должны быть
115 UnwrittenCellInitializer.h	Записывает значение по умолчанию в ячейки МПХ, в которых они всегда должны быть
116 NeutralInsertion.h	Работа с нейтральной вставкой
117 programming.h	Прошивка соседнего полукомплекта по SPI через RS-485
118 SautCom.h	Связь по протоколу САУТ-485
119 SautDat.h	Хранение данных из линии связи САУТ
120 SilentModeCanSenderDecorator.h	Отключение передачи в CAN в 'режиме тишины'
121 SystemTime.cpp	Часы, которые подстраиваются под общее время
122 SystemTime.h	Часы, которые подстраиваются под общее время
123 atomic.h	Позволяет красиво делать атомарные операции
124 AvrCanLoaderIdProvider.h	Получает параметры CAN-загрузчика на AVR
125 AvrFlashMemory.h	Реализация IFlashMemory на AVR
126 AvrIdAreaInFlash.h	Размещает IdAreaInFlash вместе с программой на AVR
127 can-async.h	Драйвер CAN

Имя файла	Назначение файла
128 can-dat.h	Хранилище данных, принятых по CAN
129 CanAsyncSender.h	Очередь отправки в CAN
130 CanImmediatelySender.h	Немедленная блокирующая отправка в CAN
131 EepromMemoryAsyncWriter.cpp	Драйвер неблокирующей записи в EEPROM
132 EepromMemoryAsyncWriter.h	Драйвер неблокирующей записи в EEPROM
133 heap.h	Хранилище временных данных
134 interrupt-dynamic.cpp	Возможность установить обработчик прерывания во время исполнения
135 interrupt-dynamic.h	Возможность установить обработчик прерывания во время исполнения
136 io.h	Работа с регистрами и портами с помощью битовых полей
137 can.h	Описание регистров CAN на AVR
138 eeprom.h	Описание регистров EEPROM на AVR
139 hole-field.h	Вспомогательный класс для описания регистров с пропусками бит
140 spi.h	Описание регистров SPI на AVR
141 timers.h	Описание регистров Timer/Counter на AVR
142 usart.h	Описание регистров UART на AVR
143 watchdog.h	Описание регистров watchdog на AVR
144 prog-spi.h	Прошивка AVR по SPI
145 smartdog.h	Уведомляет непосредственно перед срабатыванием watchdog
146 thread-safe.h	Потоко-безопасные переменные
147 timers.h	Таймеры/счётчики на AVR
148 universal.cpp	Разные универсальные удобные классы и функции
149 universal.h	Разные универсальные удобные классы и функции
150 alloca-wrapper.h	Подменяет аллока в зависимости от платформы
151 ByteManipulation.h	Побайтовые операции над переменными
152 CanFrame.h	CAN-сообщение

Имя файла	Назначение файла
153 CanMessagePeriodController.cpp	Для увеличения срока службы памяти размывает значение по памяти, чтобы записывать в разные области
154 CanMessagePeriodController.h	Для увеличения срока службы памяти размывает значение по памяти, чтобы записывать в разные области
155 CrcCalculator.h	Считает CRC
156 delegate.h	Делегаты без аллокации на куче
157 Dispatcher.h	Очередь задач
158 EventHandler.h	Частный вид делегата для 'событий'
159 ICanImmediatelySender.h	Интерфейс для немедленной блокирующей отправки в CAN
160 ICanSender.h	Интерфейс устройства, отправляющего в CAN
161 IClock.h	Часы, показывающие время
162 IEnumerable.h	Интерфейс перечисляемого объекта
163 IEnumerator.h	Интерфейс перечислителя
164 int-typelist.h	Макросы для генерации списков Loki::Int2Type
165 IQueue.h	Интерфейс очереди
166 List.h	Реализация односвязного списка без аллокации
167 ListBase.cpp	Типонезависимая часть реализации списка
168 ListBase.h	Типонезависимая часть реализации списка
169 ListItem.h	Элемент списка
170 ListItemBase.h	Типонезависимая часть элемента списка
171 NullType.h	NullType для шаблонного программирования
172 Sequence.h	Последовательности для шаблонного программирования
173 static_check.h	Проверка условий на этапе компиляции
174 Typelist.h	Список типов для шаблонного программирования
175 TypelistMacros.h	Макросы для работы со списками типов
176 TypeManip.h	Манипуляции с типами для шаблонного программирования
177 TypeTraits.h	Информация о типах для шаблонного программирования

Имя файла	Назначение файла
178 map-static.h	Ассоциативный массив с быстрым доступом
179 MemoryQueue.h	Реализует очередь на памяти, выделенной во вне
180 TypelessMemoryQueue.cpp	Типонезависимая часть очереди
181 TypelessMemoryQueue.h	Типонезависимая часть очереди
182 new.cpp	Оператор new для C++
183 new.h	Оператор new для C++
184 PeriodicLauncher.cpp	Запускает задачу с определённым периодом через Scheduler
185 PeriodicLauncher.h	Запускает задачу с определённым периодом через Scheduler
186 QueuePoper.h	Кормит асинхронную операция элементами из очереди
187 Scheduler.cpp	Позволяет выполнить функцию через какое-то время
188 Scheduler.h	Позволяет выполнить функцию через какое-то время
189 TimeSpan.h	Время, хранящая значение в системных тиках и конвертирующее его в понятные единицы
190 types.h	Преобразования типов
191 UnreadyProperty.h	Свойство, которое может быть не готово на данный момент
192 ICanReceiver.h	Интерфейс для подписывания на сообщения из CAN
193 ICanSenderDevice.h	Низкоуровневый интерфейс для отправки в CAN
194 IFlashMemory.h	Интерфейс доступа к памяти, на которой хранится прошивка
195 ITimerDevice.h	Интерфейс таймера

3.1 Принцип работы датчика ДПС

Датчик ДПС устанавливается на ось колёсной пары и даёт возможность определить скорость и направление вращения этой оси.

С осью датчика соединяется зубчатое колесо, имеющее N одинаковых зубьев, и, соответственно, N прорезей между зубьями. Размеры прорезей

считаются равными размеру зубьев. На корпусе датчика устанавливается две оптопары на одинаковом расстоянии от оси, таким образом, чтобы угловое расстояние между ними составляло от периода зубчатого колеса.

При вращении колеса последовательно перекрывает оптопары. Таким образом, каждая оптопара генерирует импульсы по своему каналу. По частоте импульсов с любого канала можно определить скорость вращения. По тому, по какому каналу импульс приходит первым, можно определить направление вращения.

3.2 Подсчет пройденного пути

Пройденный путь может быть определён по любому каналу датчика ДПС.

Состоянию с закрытой оптопарой пусть соответствует логический «0», состоянию с открытой – логическая «1». Если последовательно произошло 2 изменения состояния, то можно говорить о том, что с момента первого изменения до момента второго изменения ось повернулась на π/N радиан. Полагая, что колесо не теряет сцепления с дорогой и имеет постоянный диаметр I , получаем, что поезд проехал расстояние .

В программе реализован алгоритм, наблюдающий за состоянием канала с периодом 100 мкс и реагирующий на фронты импульсов. С приходом очередного фронта считается, что поезд проехал ещё 21 метр. Таким образом, пройденный путь рассчитывается по формуле

$$S = \sum_i \frac{\pi D}{N}, \quad (1)$$

где i – количество фронтов полученных импульсов.

Это выражение может быть сведено к

$$S = i \frac{\pi D}{N}, \quad (2)$$

в рамках компьютерной арифметики, если не использовать плавающую точку. Однако используется именно первоначальная формула, т.е. с каждым фронтом к существующему значению пройденного пути добавляется прибавка.

Прибавка и пройденный путь S хранятся в целочисленных переменных в единицах дм/65536. Таким образом, погрешность в l , возникающая из-за целочисленного округления, равна $\Delta l = 1/65536$ дм. Для сравнения погрешность, вносимая в l неточностью задания диаметра D (дискрет), равна $\Delta_D l \sim 1/1000$, т.е. на 2 порядка меньше.

Относительная погрешность $\delta S = \delta(\pi D/N)$, но абсолютное её значение $\Delta S = i \cdot \Delta(\pi D/N)$ растёт с ростом пройденного пути. Этого можно избежать, если суммировать i и каждый раз при надобности рассчитывать S . Однако это не имеет практического смысла, потому что, во-первых, ошибка в 1 метр набегит после 655 км, во-вторых, ошибка задания диаметра на 2 порядка больше.

Полным импульсом будем называть «расстояние» между двумя последовательными фронтами. По причине того, что наблюдение за портом происходит раз в 100 мкс, время импульса, который будет достоверно распознан, должно быть больше 200 мкс. Это накладывает ограничение на максимальную скорость, при которой обеспечивается корректный подсчёт пройденного пути: $V_{max} = \pi D / N T_{imp}$. Однако при реальных параметрах $N = 42$, $D = 810$ мм максимальная скорость оказывается равной $V_{max} \approx 1000$ км/ч, что не является существенным ограничением.

3.3 Определение скорости

При определении скорости анализ происходит по фронтам импульсов одного из каналов ДПС, как и в случае с определением пройденного пути. Состояние порта наблюдается с периодом 100 мкс.

В широком диапазоне скоростей невозможно с достаточной точностью определить скорость по одному импульсу. Так как с ростом скорости время одного импульса уменьшается, и становится сравнимым с погрешностью измерения времени, равной ± 200 мкс.

Используется следующий алгоритм. После прихода 1-го (условно) фронта начинается отсчёт времени. Выдерживается время τ , после чего ожидается приход очередного фронта. По приходу происходит остановка счётчика времени и

счётчика импульсов, рассчитывается скорость. С этого фронта начинается следующее измерение. Такая схема обеспечивает измерение времени с погрешностью не более ± 200 мкс и измерение количества импульсов без погрешности.

Таким образом, скорость вычисляется по формуле

$$V = \frac{\pi D}{NT_d} \cdot \frac{n}{k}, \quad (3)$$

где n – счётчик импульсов (фронтов),

k – счётчик временных дискретов,

T_d – дискрет времени,

D – диаметр колеса,

N – количество зубьев датчика ДПС.

А погрешность определения скорости

$$\Delta V = V \cdot \frac{\Delta k}{k}, \quad \Delta k = 2, \quad (4)$$

Из этого выражения находится $\tau = T_d k$, такое, чтобы удовлетворялись требования к точности определения скорости. Однако чрезмерное увеличение k с целью повысить точность определения скорости повлечёт увеличение времени одного измерения, что может оказаться недопустимым. В программе используется $k = 4096$ при $T_d = 100$ мкс, такие параметры обеспечивают $\delta V = 0.05\%$ и дают удовлетворительные данные для дальнейшего расчёта ускорения.

Однако в соответствии с алгоритмом $k = \tau / T_d$ является минимально возможным k . Выбором k_{min} определяется максимально возможная погрешность, при бóльших k погрешность будет меньше. С уменьшением скорости увеличивается k . Если время одного импульса становится больше чем τ , то $k = T_{imp} / T_d$.

После прихода первого фронта (начало импульса) начинается отсчёт k , который должен быть остановлен с приходом второго фронта (конец импульса). Но как долго ждать прихода второго фронта, т. е. каково k_{max} ? Чем больше k_{max} , тем меньшую скорость можно зарегистрировать, но тем большее время нужно

ожидать, чтобы удостовериться, что произошла полная остановка. В программе $k_{max} = 43085$, т. е. импульсы должны отсутствовать в течение 4,3 с, чтобы $V = 0$.

3.4 Вычисление ускорения

Обыкновенно, в физике, ускорением называется производная от скорости по времени. Как видно из предыдущего раздела, значение скорости имеет дискретные значения и квантовано по времени. Вычисление производной обычными методами в таких условиях становится невозможным. Решение подобных задач изучается наукой под названием «Цифровая обработка сигналов» (Digital Signal Processing).

Наиболее естественным было бы применить в рассматриваемом случае цифровой дифференцирующий фильтр. Однако практически не удалось подобрать оптимальный КИХ-дифференциатор длиной импульсной характеристики менее 16 точек. Выполнение свёртки более, чем на 16 точках оказалось слишком затратным с точки зрения использования машинных ресурсов.

Удовлетворительных результатов удалось добиться рассчитывая ускорение синхронно со скоростью как $\Delta V / \Delta T$ ($\Delta V = V_{\text{текущий расчет}} - V_{\text{предыдущий расчет}}$, $\Delta T = kT_{imp}$) и применяя к так рассчитанному значению ускорения фильтр нижних частот. Используется цифровой эллиптический НЧ-фильтр 3-го порядка с разностным уравнением

$$y(n) = \sum_{i=0}^p b_i(n-i) - \sum_{k=1}^Q a_k y(n-k), \quad (5)$$

где $b_0 = 1, b_1 = 1, a_1 = 3/4$.

Как видно фильтр имеет всего 3 коэффициента, только один из которых не равен 1. Это делает его реализацию чрезвычайно простой. В прямой канонической форме необходим всего один элемент задержки, один множитель и 2 сумматора (рисунок 1).

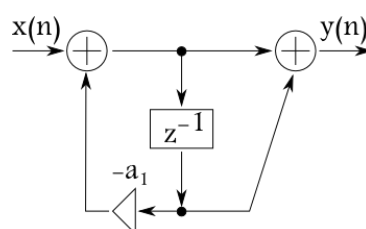


Рисунок 1

Такой фильтр позволяет определять ускорение с погрешностью в диапазоне скоростей от 0 до 512 км/ч при постоянном значении ускорения. Условие постоянства ускорения возникает из того факта, что любой подобный фильтр будет давать задержку.

Не составляет труда подобрать простой в реализации и очень эффективный с точки зрения подавления пульсаций БИХ-фильтр, однако вносимые им задержки будут неприемлемо велики. Поэтому, подбирая фильтр, необходимо искать баланс между эффективностью подавления пульсаций и скоростью работы.

На графике переходная характеристика используемого фильтра, т.е. реакция на ступеньку ускорения с 0 до 1 (рисунок 2).

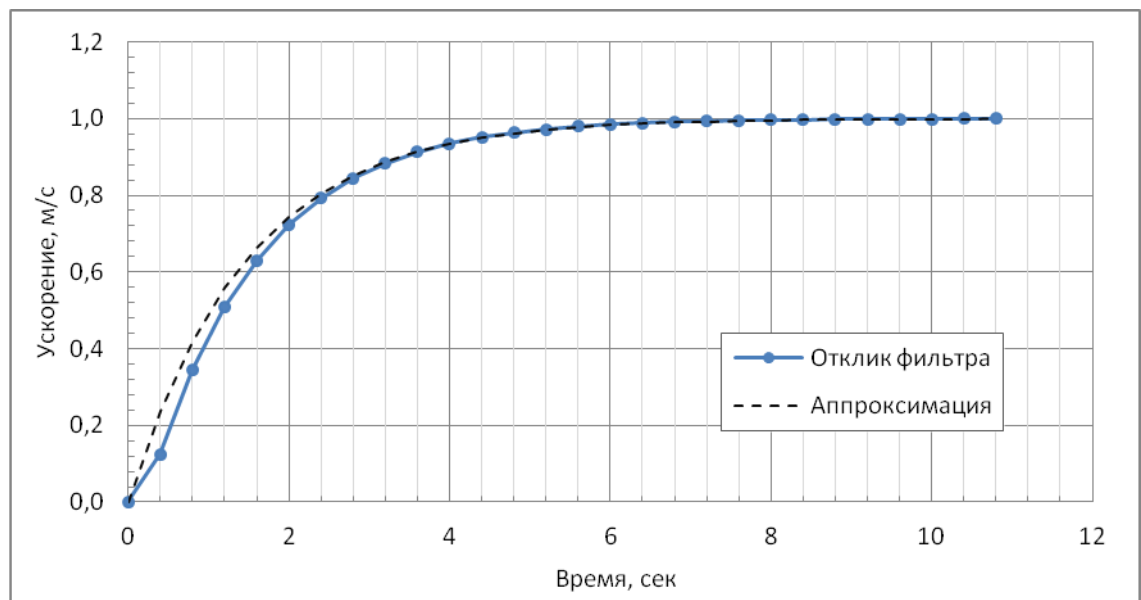


Рисунок 2

Как видно, отклик фильтра хорошо аппроксимируется экспоненциальной функцией:

$$x(t) = 1 - e^{-\frac{t}{k}}, \quad (6)$$

где $k = 0,68$.

Это означает, что задержки можно охарактеризовать величиной k следующим образом. Если значение ускорения изменилось скачком на величину

$\Delta A = A_2 - A_1$, то на выходе системы получится значение A_2 с погрешностью δA по формуле

$$t_{\text{задержки}} = k \cdot \ln\left(\frac{\Delta A}{\delta A}\right) \quad (7)$$

3.5 Диагностика неисправностей

Как уже отмечалось выше, датчик ДПС имеет 2 канала. Хотя для определения характеристик движения достаточно одного, второй канал необходим для определения направления вращения колеса. Дополнительно, наличие второго канала позволяет диагностировать неисправности.

В момент прихода фронта импульса, который начинает новый отсчёт, сбрасываются счётчики импульсов и времени не только текущего канала, но и дополнительно. Текущий канал начинает отсчёт времени уже с этого момента, а дополнительный – только после прихода фронта по своему каналу.

В момент прихода фронта импульса, заканчивающего текущий отсчёт, происходит сравнение значений счётчиков импульсов двух каналов. Если они различаются более, чем на 1 импульс, то делается вывод о том, что полученное в ходе измерения значение скорости недостоверно.

Сделать вывод о том, какой из этих 2 каналов неисправен, невозможно. Возможная пропажа импульсов, а возможны помехи и, как следствие, лишние импульсы.

Помимо того, что каждый датчик имеет 2 канала, на локомотиве установлено 2 датчика на разных колёсных парах. Это даёт возможность в некоторых случаях определять шлифование и блокировку колеса.

Если один датчик неисправен (выдаёт данные, помеченные как «недостоверные»), то скорость берётся с другого. Если оба датчика неисправны, то выдаётся наибольшее значение. Если оба датчика исправны, то в режиме тяги выдаётся наименьшее значение, а без тяги (торможение) – наибольшее значение.

Если скорость по датчикам расходиться больше чем на 25 % в течении 90 с и абсолютное значение скорости по быстрому датчику превышает 10 км/ч, то более медленный считается неисправным, а скорость берётся с более быстрого.

Если скорости сравниваются в пределах 25 %, то по истечении 30 с восстанавливается исправность датчика.

Если в течении 70 с есть тяга, но скорость не появилась, то неисправность выставляется обоим ДПС.

Дополнительной функциональностью блока является определение местоположения изостыка, основанное на расшифровке кодов КПТ по огибающей АЛСН. Изостыком считается место, в котором произошла смена типа КПТ.

Каждая посылка КПТ представляет собой последовательность импульсов определённой длительности с определёнными паузами между ними. Посылки идут непрерывно. На рисунке 3 представлены все возможные виды посылок:

Тип кодов	Обозначение кода	Продолжительность импульсов и интервалов
5	«3»	
	«Ж»	
	«КЖ»	
7	«3»	
	«Ж»	
	«КЖ»	

Рисунок 3

Отсутствие импульсов более 1,86 с означает «БЕЛЫЙ» или «КРАСНЫЙ» цвет: если до этого был «КЖ», тогда определяется «К», если до этого был любой другой цвет – «Б».

Посылка может быть достоверно расшифрована только в тот момент, когда она будет принята до конца. Поэтому изостык можно локализовать только с

точностью до сбойной посылки. Если в качестве местоположения изостыка брать конец сбойной посылки, то это может привести к проезду светофора, т.к. система будет считать, что блок-участок начался позже, чем это произошло на самом деле. Поэтому в качестве местоположения изостыка берётся координата в момент прихода конца последней корректной посылки (рисунок 4).



Рисунок 4

На практике длительности импульсов значительно отличаются от изображённых на рисунке 3. Импульсы зачастую короче, а паузы – длиннее. В связи с этим для расшифровки посылки применяется достаточно лояльный алгоритм.

Для определения типа КПТ измеряется полное время посылки как время между соседними длинными паузами. Длинной считается пауза более 0,35 с. Если между двумя длинными паузами не было короткой, то время удваивается. Посылка распознается как КПТ-5, если её длительность от 1,47 до 1,73 с, а как КПТ-7 при длительности от 1,73 до 1,99 с. В остальных случаях посылка считается некорректной. Код (цвет) определяется по количеству коротких импульсов. Если их количество больше трех, то посылка считается некорректной.

При проезде изостыка возможно наложение начала одной посылки на конец другой таким образом, что получится третья, отличная от этих двух и признанная «корректной». Чтобы исключить эту ситуацию, корректной будет считаться только та посылка, которая тождественна предыдущей. Побочный эффект – признание первой правильной после изостыка посылки «некорректной», не критичен, поскольку изостык определяется по концу последней корректной посылки.

4 ИСПОЛЬЗУЕМЫЕ ТЕХНИЧЕСКИЕ СРЕДСТВА

Блок БС-ДПС работает в составе ПО БЛОК-М и иных локомотивных систем.

Используемый микроконтроллер - AT90CAN128 (ЦП – 16 МГц, ОЗУ – 4 Кб, ПЗУ – 128 Кб).

5 ВЫЗОВ И ЗАГРУЗКА

Программа начинает автоматически выполняться при подаче электропитания на систему БЛОК-М.

6 ВХОДНЫЕ И ВЫХОДНЫЕ ДАННЫЕ

Модуль выдаёт и принимает информацию по двум каналам связи: по линии связи RS-485 и по CAN-интерфейсу.

6.1 Линия связи RS-485

Взаимодействие по линии связи происходит в соответствии с протоколом, описанным в документе «Форматы данных САУТ-485».

6.1.1 Пройденный путь.

Программа наблюдает за всеми проходящими посылками и в каждую посылку в качестве 3-го байта передаёт текущее значение пройденного пути в дециметрах, если на то есть разрешение. Так как половина байта используется для кода Хэмминга, то отправляются только 4 младших бита счётчика пройденного пути.

Запрет на отправку 3-го байта модуль получает в 7-ом бите старшего байта посылки к устройству 1 по адресу 0 на запись. Разрешение на отправку появляется автоматически после получения 10 неоконченных посылок без 3-го байта.

6.1.2 Устройство № 1 (БС-ДПС).

Модуль принимает «команды», переводящие его в различные режимы работы, - это сообщения на запись по адресу 0 устройства 1 в соответствии с протоколом.

Существуют следующие режимы работы:

- основной. Измерение характеристик движения. Этот режим активируется, если принята команда, в которой отсутствуют запросы на другие режимы. В этом режиме модуль производит измерение характеристик движения и выдаёт результат в линию связи, обновляя его каждые 0,48 с;

- чтение идентификационной информации. В этом режиме измерение характеристик останавливается и выдаётся информация в соответствии с таблицей 1:

Таблица 1

Адрес	Данные	
	Старший байт	Младший байт
0	Версия	Год
1	Модификация	Месяц
2	Номер блока	
3	Контрольная сумма	

Эта информация хранится в специальной области flash-памяти по адресу и заносится туда программатором;

- чтение Еергом. В этом режиме измерение характеристик останавливается и выдаётся запрошенная информация из еергом. Номер запрашиваемого 8-байтного блока передаётся в младшем байте посылки с запросом. Данные передаются старшим байтом вперёд;

- режим программирования соседнего полукомплекта. Переход в этот режим осуществляется, если в качестве команды принята посылка с содержанием 0x0EFE. После этого взаимодействие происходит в определённом формате, который будет когда-нибудь описан.

Сообщения на адрес 1 игнорируются.

6.1.3 Устройство № 10 (БС-КЛУБ).

От имени устройства под номером №10 выдаётся следующая информация:

Данные, получаемые из CAN-интерфейса:

Передаются посредством, так называемых, страниц КЛУБа, содержание которых описано в «Форматы данных САУТ-485». 3 байта текущей страницы находятся по адресам 2 и 3. Переключение на следующую страницу происходит по приходу любого сообщения на адрес 1. В данный момент заполняются только данные страницы 0 (исключая тип препятствия), расстояние до цели и реальное

время. Остальные данные всегда = 0. Данные берутся из CAN-сообщений MCO_STATE и IPD_DATE. Дополнительно передаются нажатия кнопок «ОС», «ОТПР», «ПОДТ», «K20». Информация об их нажатии берётся из сообщения SYS_KEY, после нажатия флаги удерживаются в течение 1 с.

Расшифровка посылки КПТ:

Расшифровка сигналов КПТ выполняется двумя модулями МП-АЛС и БС-ДПС. Результат расшифровки МП-АЛС берётся из CAN-сообщения MCO_STATE и передаются в старшем байте адреса 0. Результат расшифровки модулем БС-ДПС находится в младшем байте адреса 1. Флаг «ИФ» передаёт состояние огибающей АЛСН в текущий момент.

Положение изостыка:

При проезде изостыка включается счётчик пройденного пути в единицах 1,6 метров, выдающий значение в старший байт 1 адреса. По нему система может определить положение изостыка. Для корректного включения счётчика необходимо, чтобы по адресу 0 в старший байт был записан флаг «Разр. dS», а флаг «Сбр. dS» до проезда изостыка претерпел изменение с 1 на 0. Почему используется такой странный способ, никто объяснить не может.

6.1.4 Эмуляция движения по запросу от устройства №13 (БПР).

Модуль анализирует исходящие сообщения от устройства №13 по адресу 0 и по адресу 3. По адресу 0 передаётся флаг запроса на эмуляцию движения. Если этот флаг установлен и по адресу 3 не реже чем раз в секунду передаётся скорость, то модуль эмулирует сигналы от датчиков ДПС.

В режиме эмуляции система берёт информацию о состоянии каналов не с внешних входов, а из внутренней переменной, значение которой изменяется с определённой частотой, необходимой для обеспечения требуемой скорости. Подбирается наиболее подходящая частота, которую может обеспечить Timer/Counter микроконтроллера, однако она не в точности соответствует требуемой. При скоростях от 4 до 160 км/ч погрешность не превышает ± 1 км/ч.

6.2 CAN-интерфейс

6.2.1 Модуль измерения параметров движения.

Модуль измерения параметров движения приведён в таблице 2.

Таблица 2

Входящие		Исходящие	
Дескриптор	Данные	Дескриптор	Данные
MCO_STATE_A(B)	0-контроллер	IPD_STATE_A(B)	Скорость, пройденный путь, исправность, направление
MCO_LIMITS_A(B)	Двойная тяга		
MM_DATA	Пройденный путь по ЭК, флаг «в карте»	SAUT_INFO_A(B)	Ускорение, скорость, информация, диаметры бандажей

Если при наличии тяги (положение ноль-контроллер = 0) в течение 70 с отсутствует скорость и нет признака двойной тяги, то обоим датчика выставляется неисправность.

В IPD_STATE передаётся скорректированный под ЭК пройденный путь. Корректировка происходит в соответствии со следующим алгоритмом. Если от карты пришли данные MM_DATA, есть признак «в карте» и отсутствует признак «недостоверность», если скорость более 2 км/ч, то выполняется сравнение пройденного пути по данным от БС-ДПС в текущий момент и пришедшим данным от ЭК. Если пути расходятся более, чем на 500 м, то в IPD_STATE выставляется флаг о критическом расхождении. Если пути расходятся менее чем на 25 м, то никаких действий не выполняется. В остальных случаях в качестве текущего значения пройденного пути БС-ДПС берётся среднее значение.

6.2.2 Модуль постоянных характеристик.

Модуль постоянных характеристик приведён в таблице 3.

Таблица 3

Входящие		Исходящие	
Дескриптор	Данные	Дескриптор	Данные
INPUT_DATA	Номер ячейки и данные для записи	SYS_DATA	Номер ячейки и её содержание
Дескриптор	Данные	Дескриптор	Данные
MCO_DATA	Номер ячейки и данные для записи	SYS_DATA_STATE	Конфигурация (младший байт), категория, допустимые скорости, длина блок-участка, длина поезда
BKSI_DATA	Номер ячейки и данные для записи		
SYS_DATA_QUERY	Номер запрашиваемой ячейки	SYS_DATA_STATE2	Конфигурация и тип локомотива

По приходу дескрипторов INPUT_DATA, MCO_DATA, BKSI_DATA происходить запись данных в нужную ячейку МПХ, после этого для контроля данные из этой ячейки считываются и отправляются в SYS_DATA.

По приходу SYS_DATA_QUERY запрашиваемые данные считываются и отправляются в SYS_DATA.

SYS_DATA_STATE и SYS_DATA_STATE2 отправляются 2 раза в секунду.

Запросы на чтение и запись в ячейки с адресами 1 и 9 игнорируются.

6.2.3 Диагностика.

Диагностика приведена в таблице 4.

Таблица 4

Входящие		Исходящие	
Дескриптор	Данные	Дескриптор	Данные
SYS_DIAGNOSTICS	Код модуля и требуемая операция	AUX_RESOURCE_IPD_A(B)	Ответ на запрашиваемую операцию
		AUX_RESOURCE_BS_A(B)	Ответ на запрашиваемую операцию

Модуль отвечает на запросы, адресованные модулю ИПД (код 4) и БС-ДПС (код 16). На данный момент реализован вывод версии и чтение/запись текущего значения координаты. Эти данные хранятся в соответствующей ячейке МПХ.

6.2.4 Данные, необходимы для работы с линией связи RS-485.

Данные, необходимы для работы с линией связи RS-485, приведены в таблице 5.

Таблица 5

Входящие	
Дескриптор	Данные
MP_ALS_ON_A(B)	Длительность паузы КПП
MP_ALS_OFF_A(B)	Длительность импульса КПП
MCO_STATE_A(B)	Расшифрованная МП-АЛС посылка КПП, нажатие кнопок РБ и РБС, допустимая скорость, целевая скорость, расстояние до цели, ключ ЭПК 1-ой кабины
IPD_DATE	Дата/время
SYS_KEY	Нажатие кнопок «ОС», «ОТПР», «ПОДТ», «К20»

ПРИЛОЖЕНИЕ 1

(Справочное)

ПЕРЕЧЕНЬ ПРИНЯТЫХ СОКРАЩЕНИЙ

АЛСН – автоматическая локомотивная сигнализация непрерывного действия;

«Б» – белый;

БЛОК-М – безопасный локомотивный объединенный комплекса – масштабируемого;

БС-ДПС – блок связи с датчиком пути и скорости;

БС-КЛУБ – блок связи с комплексным локомотивным устройством безопасности;

БПР – блок проверочный;

ДПС – датчик пути и скорости;

ИПД – измеритель параметров движения;

«К» – красный;

«КЖ» – красно-жёлтый;

КПТ – кодовый путевой транзиттер;

КПТ-5 – кодовый путевой транзиттер, тип 5;

КПТ-7 – кодовый путевой транзиттер, тип 7;

МПХ – модуль постоянных характеристик;

ПО – программное обеспечение;

САУТ – система автоматического управления торможением;

ЭК – электронная карта.

ПРИЛОЖЕНИЕ 2

(Обязательное)

СБОРКА ПРОГРАММНОГО МОДУЛЯ БС-ДПС

1 Цель сборки

В результате сборки будет получен исполняемый файл программного модуля БС-ДПС:

- версии «8», с наименованием «DPS_CAN.FLA», длиной 54452 байт, и контрольной суммой CRC32 равной «6964D61B», предназначенный для исполнения на микроконтроллере «Atmel AVR AT90CAN128»;
- версии «14», с наименованием «DPS_CAN.FLA», длиной 68348 байт, и контрольной суммой CRC32 равной «A087FD56», предназначенный для исполнения на микроконтроллере «Atmel AVR AT90CAN128».

2 Необходимые технические средства

Для сборки исполняемого файла программного модуля необходим компьютер, способный работать под операционной системой «Windows 7».

3 Необходимые программные средства

Сборка исполняемого файла выполняется в операционной системе Windows.

Предполагается, что исходные тексты получены посредством системы контроля версий Git.

Сборка осуществляется системой сборки CMAKE, возможно использование среды JetBrains CLion.

Компилятор в виде бинарных файлов поставляется вместе с исходными кодами.

4 Подготовка к сборке

Клонировать проект из Git-репозитория.

Переключиться на тег в зависимости от нужной версии.

5 Выполнение сборки

С помощью CMAKE или JetBrains CLion скомпилировать проект.

6 Получение исполняемого файла

В результате сборки в папке «Debug» будет получен файл «DPS_CAN.FLA».

[illegible]