

Лабораторная работа N4

Изучение контроллера CAN 2.0

Оглавление

Общие сведения	3
Интерфейс CAN 2.0B	4
Основные характеристики	5
Послойная архитектура CAN - сети согласно модели OSI	5
Сообщения	6
Информационная маршрутизация	6
Скорость передачи информации в битах	7
Приоритеты	7
Удаленный запрос данных	7
Multimaster	7
Арбитраж	7
Безопасность	8
Обнаружение ошибок	8
Эффективность обнаружения ошибок	8
Сигнализация ошибки и время восстановления	8
Типизация ошибок	8
Подключение	8
Канал обмена	8
Уровни сигналов	9
Подтверждение	9
Режим "сна" / пробуждения	9
Генератор	9
Передача сообщений	9
Форматы кадров	9
Типы кадров	9
Кадр данных (Data frame)	10
Начало кадра (Start of frame)	10
Поле арбитража (Arbitration field)	10
Идентификатор	11
Поле управления (Control field)	12
Поле данных (Data field)	13
Поле CRC (CRC field)	13
Поле подтверждения (ACK field)	13
Конец кадра (End of frame)	14
Кадр удаленного запроса данных (Remote frame)	14
Кадр ошибки (Error Frame)	15
Кадр перегрузки (Overload Frame)	16
Межкадровое пространство (Interframe Spacing)	17
Определение передатчика / приемника	18
Фильтрация сообщений	18
Проверка допустимости сообщения	18
Кодирование	19
Обработка ошибок	19
Обнаружение ошибок	19
Передача сигналов ошибки	20
Типизация неисправностей	20
Требования к битовой синхронизации	20

Драйвер CAN для микроконтроллера LPC2292	23
Описание регистров	28
Регистр режима работы CANMOD	29
Командный регистр CANCMR	29
Глобальный регистр статуса CANGSR	30
Регистр разрешения прерывания CANIER	30
Регистр временных характеристик шины CANBTR	31
Регистр статуса CANSR	31
Регистр статуса полученного сообщения CANRFS	32
Регистр данных полученного сообщения CANRDA	32
Регистр информации о передаваемом сообщении CANTFI	32
Литература	33

Общие сведения

CAN (Controller Area Network) является высокотехнологичным средством управления и передачи последовательных данных в распределенных системах, находящихся всё большее применение в различных областях промышленности и имеющих большое количество интеллектуальных датчиков.

Области применения CAN:

- авиационно-космическая промышленность
- автомобильная и сельскохозяйственная промышленность
- военная промышленность
- морской и железнодорожный транспорт
- медицинская промышленность и т.д.

Преимущества CAN:

- надежность
- невысокая стоимость
- гибкость
- высокая помехоустойчивость
- работа в режиме реального времени

CAN сеть имеет сходство с Ethernet, но в отличие от неё имеет гарантированное время доставки пакетов и соответственно нет потери времени при столкновении пакетов от двух различных узлов. Передача пакета имеющего меньший приоритет откладывается до тех пор, пока не освободиться шина.

CAN имеет мощные средства для обнаружения ошибок в виде пяти механизмов, реализованных аппаратно. Кроме того, CAN позволяет отключать узел, являющийся источником фиксированных ошибок с возможностью автоматического подключения его, тем самым CAN сеть невозможно “подвесить” и она остается работоспособной. Даже в случае обрыва одного из проводов дифференциальной пары CAN шины или замыкании одного из них на источник питания или землю CAN по-прежнему остается работоспособным.

В CAN возможна лишь одна не обнаруживаемая ошибка за 1000 лет, при работе системы 8 часов в день, в течение 365 дней и частоте появления ошибок 0,7 в секунду. В CAN сети все узлы участвуют в проверке битстаффинга и целостности фрейма, все также непосредственно участвуют в подтверждении правильности приема фрейма и маркировке неправильного фрейма, применяются коды Хемминга (расстояние Хемминга - 6) и 15-битовая CRC.

Протокол CAN имеет большой потенциал для своего развития благодаря:

- аппаратной поддержке протокола
- различной среде передачи (основная – витая пара, и не основные – оптоволокно, радиоканал, ИК-канал, силовые линии, однопроводная линия и т.д.)
- превосходной обработке ошибок; хорошей поддержке систем реального времени
- хорошей поддержке систем, управляемых событиями
- ориентированности на распределенные системы управления

Интерфейс CAN 2.0B

В любой реализации CAN - носитель (физическая среда передачи данных) интерпретируется как эфир, в котором контроллеры, работают как приемники и передатчики. При этом, начав передачу, контроллер не прерывает слушание эфира, в частности он отслеживает и контролирует процесс передачи текущих, передаваемых им же, данных. Это означает, что все узлы сети одновременно принимают сигналы, передаваемые по шине. Невозможно послать сообщение какому-либо конкретному узлу. Все узлы сети принимают весь трафик, передаваемый по шине. Однако CAN-контроллеры предоставляют аппаратную возможность фильтрации CAN-сообщений.

Область применения – от высокоскоростных сетей до дешевых мультиплексных шин. В автоматике, устройствах управления, датчиках используется CAN со скоростью до 1 Мбит/с.

Задача данной спецификации состоит в том, чтобы достигнуть совместимости между любыми двумя реализациями CAN - систем. Однако, совместимость имеет различные аспекты относительно, например электрических элементов и интерпретации данных, которые будут передаваться.

Для достижения прозрачности проекта и гибкости реализации, CAN был подразделен на различные уровни согласно модели ISO/OSI:

- Уровень передачи данных (Data Link Layer)
- Подуровень логического управления линией (LLC)
- Подуровень управления доступом к среде передачи (MAC)
- Физический Уровень (Physical Layer)

Обратите внимание, что в предыдущих версиях спецификации CAN функции LLC и MAC подуровней, уровня передачи данных, были описаны в уровнях, обозначенных как “объектный уровень” и “канальный уровень”.

Область LLC подуровня:

- 1) обеспечение сервиса для передачи данных и для удалённого запроса данных;
- 2) решение, какие сообщения, полученные LLC подуровнем, должны быть фактически приняты;
- 3) обеспечение средствами для управления восстановлением и уведомления о перегрузке.

Область MAC подуровня главным образом – протокол передачи, то есть: арбитраж, проверка на ошибки, сигнализация и типизация ошибок. Внутри MAC подуровня решается, является ли шина свободной для начала новой передачи или возможен только приём данных.

В MAC подуровень также включены некоторые элементы битовой синхронизации. Всё это находится внутри MAC подуровня и не имеет никакой возможности к модификации.

Область физического уровня – фактическая передача битов между различными узлами с соблюдением всех электрических правил. Физический уровень определяет сопротивление кабеля, уровень электрических сигналов в сети и т.п.

Внутри одной сети, физический уровень одинаков для всех узлов. Однако существует свобода в выборе физического уровня. Цель этой спецификации – определить MAC подуровень и небольшую часть LLC подуровня уровня передачи данных и описать действие протокола CAN на окружающие уровни.

Основные характеристики

- Приоритетность сообщений
- Гарантированное время отклика
- Гибкость конфигурации
- Групповой прием с синхронизацией времени
- Система непротиворечивости данных
- Multimaster
- Обнаружение ошибок и их сигнализация
- Автоматическая ретрансляция испорченных сообщений, как только шина снова станет свободной
- Различие между нерегулярными ошибками и постоянными отказами узлов и автономное выключения дефектных узлов

Послойная архитектура CAN - сети согласно модели OSI

- Физический уровень определяет, как сигналы фактически передаются и, следовательно, имеет дело с описанием битовой синхронизации и кодирования битов. Внутри этой спецификации характеристики передатчика / приемника физического уровня не определены, чтобы позволить среде передачи и реализации уровня сигнала быть оптимизированными для конкретных систем.
- MAC подуровень представляет собой ядро протокола CAN. Он передает сообщения, полученные от LLC подуровня, и принимает сообщения, которые будут переданы к LLC подуровню. MAC подуровень ответственен за арбитраж, подтверждение, обнаружение ошибок и их сигнализацию.
- LLC подуровень имеет отношение к фильтрации сообщений, уведомлению о перегрузке и управлению восстановлением.

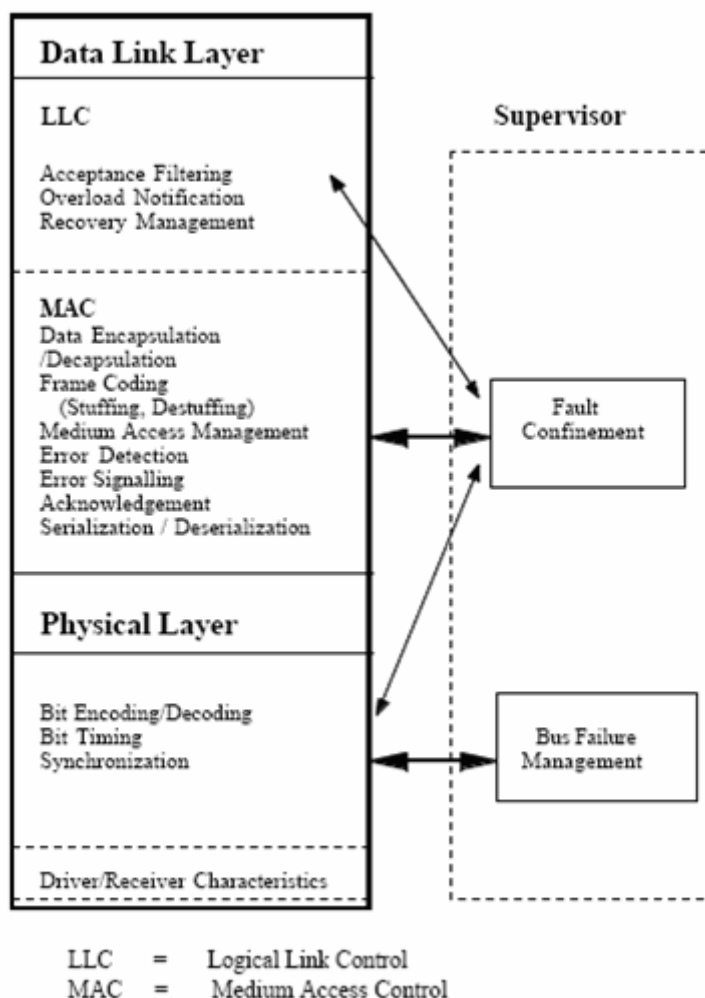


Рисунок 1. Послойная архитектура CAN

Сообщения

Информация на шине представлена в виде фиксированных сообщений различной, но ограниченной длины. Когда шина свободна, любой подключённый узел может начать передавать новое сообщение.

Информационная маршрутизация

В системах CAN нет никакой информации относительно конфигурации системы (например, адрес узла). Это имеет несколько важных следствий:

1. Гибкость системы: узлы могут быть добавлены к CAN - сети без изменения программного обеспечения и аппаратных средств любого узла или прикладного уровня.
2. Маршрутизация сообщения: содержание сообщения определяется идентификатором. Идентификатор не указывает адресата сообщения, но описывает значение данных, так, чтобы все узлы в сети были способны решить фильтрацией, должны ли они воздействовать на данное сообщение или нет.
3. Групповая обработка: любое число узлов может получать и одновременно воздействовать на одно и то же сообщение.
4. Непротиворечивость данных: внутри CAN-сети гарантирует, что сообщение одновременно принято или всеми узлами или ни одним узлом.

Скорость передачи информации в битах

Скорость CAN может быть отлична в различных системах. Однако в конкретной сети скорость передачи информации должна быть фиксированная.

Приоритеты

Идентификатор определяет статический приоритет сообщения в течение доступа к шине. Идентификатор служит уникальным именем для типа сообщения и определяет то, кем будет принято и как будет интерпретировано следующее за ним поле данных.

Более интересным представляется использование идентификаторов в качестве основного инструмента, используемого в процедуре разрешения коллизий. В CAN в качестве основного критерия для разбора коллизий, для принятия решения, кому отдать эфир, используется приоритет сообщений. Если одновременно несколько станций начали передачу, и при этом произошла коллизия, происходит суперпозиция передаваемых идентификаторов. Идентификаторы последовательно, побитно (bitwise), начиная со старшего, налагаются друг на друга и в их “противоборстве” выигрывает тот, у кого меньше арифметическое значение идентификатора, а значит, выше приоритет. Доминантный “ноль” подавит единицы и в любом случае к концу передачи поля идентификатора оно станет равно более приоритетному значению. Таким образом, система позволяет на уровне проектирования (и определения идентификатора) для любого сообщения в системе заранее предопределить его приоритетность в обслуживании.

Приоритетность сообщения, таким образом, определяется значением идентификатора. Приоритет тем больше, чем идентификатор меньше.

Удаленный запрос данных

Посылая кадр удалённого запроса данных, узел может запрашивать данные у другого узла, которые тот должен послать в виде соответствующего кадра данных. Кадр данных и кадр удаленного запроса данных имеют один и тот же идентификатор.

Multimaster

Когда шина свободна, любой узел, может начинать передавать сообщение. Узлу с сообщением более высокого приоритета, будет передано право на доступ к шине.

Арбитраж

Всякий раз, когда шина свободна, любой узел может начинать передавать сообщение. Если два или большее количество узлов начинают передавать сообщения в одно и то же время, конфликт доступа к шине решается поразрядным арбитражем, с использованием идентификатора. Механизм арбитража гарантирует, что ни информация, ни время не будут потеряны. Если кадр данных и кадр удаленного запроса данных с одинаковым идентификатором начинают передаваться одновременно, то кадр данных преобладает над кадром удаленного запроса данных. В течение арбитража каждый передатчик сравнивает уровень переданного бита с уровнем, который пытается передать на шину конкурирующий передатчик. Если значения этих битов равны, оба передатчика пытаются передать следующий бит. И так происходит до тех пор, пока значения передаваемых битов не окажутся различными. Теперь контроллер, который передавал логический ноль (более приоритетный сигнал) будет продолжать передачу (см. “Уровни

сигналов”), а другой (другие) контроллер прервёт свою передачу до того времени, пока шина вновь не освободится. Конечно, если шина в данный момент занята, то контроллер не начнет передачу до момента её освобождения.

Безопасность

Для достижения безопасности передачи данных, в каждом узле CAN имеются мощные средства самоконтроля, обнаружения и сообщения об ошибках.

Обнаружение ошибок

Для обнаружения ошибок применяются следующие меры:

- Текущий контроль (передатчики сравнивают уровни передающихся битов с уровнями, обнаруженными на шине)
- Циклический контроль по избыточности
- Контроль кадра сообщения

Эффективность обнаружения ошибок

Механизмы обнаружения ошибок позволяют обнаруживать:

- 1) все ошибки глобального характера;
- 2) все локальные ошибки передатчика;
- 3) до 5 случайных ошибок в сообщении;
- 4) последовательную группу ошибок длиной до 15;
- 5) любые ошибки нечетности.

Общая остаточная вероятность ошибки для необнаруженных искаженных сообщений меньше, чем частота ошибки сообщения $\times 4.7 \times 10^{-11}$.

Сигнализация ошибки и время восстановления

Искаженные кадры помечаются любым узлом, обнаружившим ошибку. Такие кадры прерываются и должны быть автоматически переданы заново. Время восстановления от обнаружения ошибки до начала следующего кадра – в большинстве случаев равно 31 интервалу передачи бита, если не будет последующей ошибки.

Типизация ошибок

Узлы CAN способны отличить короткие неполадки от постоянных отказов. Дефектные узлы отключаются.

Подключение

Последовательная линия связи CAN является шиной, к которой теоретически может быть подключено любое количество узлов. Фактически общее число узлов будет ограничено временной задержкой и (или) электрической нагрузкой на линии шины.

Канал обмена

Шина состоит из канала обмена, который передаёт биты. Из этих данных может быть получена информация о пересинхронизации. Способ, по которому реализуется этот

канал обмена, в этой спецификации не устанавливается. Например, это может быть однопроводочный провод (плюс земля), два дифференциальных провода, оптическое стекловолокно.

Уровни сигналов

Шина может иметь одно из двух логических значений: “dominant” или “recessive”. При одновременной передаче “dominant” и “recessive” битов, возникающая в результате величина на шине будет “dominant”. Физические состояния (например, электрическое напряжение, свет), которые представляют логические уровни, не даны в этой спецификации (далее считается, что “dominant” уровень эквивалентен нулевому уровню, а “recessive” уровень эквивалентен единичному уровню).

Подтверждение

Все приемники проверяют непротиворечивость получаемого сообщения, подтверждают непротиворечивость сообщения и помечают противоречивые сообщения.

Режим "сна" / пробуждения

Для уменьшения потребляемой мощности системы, CAN - узел может быть переведен в режим “сна” без внутренней активности и с отключенными формирователями шины. Выход из этого режима происходит при проявлении какой-либо активности на шине или внутреннем состоянии системы. При пробуждении, осуществляется внутренний перезапуск, МАС подуровень будет ждать стабилизации генератора системы, и затем будет ожидать самосинхронизации с сигналами на шине (проверяя, одиннадцать последовательных единичных бит), пока выходные формирователи снова не установятся в состояние “подключен к шине”.

Генератор

Требования к синхронизации позволяют использовать керамические резонаторы в системах со скоростями передачи до 125 Кбит/с. Для более высокой скорости шины CAN требуется кварцевый резонатор.

Передача сообщений

Форматы кадров

Имеются два формата, которые отличаются по длине поля идентификатора:

- Кадры с 11-разрядным идентификатором называются стандартными кадрами.
- Кадры, содержащие 29-разрядные идентификаторы, называются расширенными кадрами.

Типы кадров

Данные в CAN передаются короткими сообщениями-кадрами стандартного формата. В CAN существуют четыре типа сообщений:

Кадр данных (Data Frame) передает данные от передатчика к приемнику.

Кадр удаленного запроса данных (Remote Frame) передается узлом, чтобы запросить передачу кадра данных с тем же самым идентификатором.

Кадр ошибки (Error Frame) передается любым узлом при обнаружении ошибки на шине.

Кадр перегрузки (Overload Frame) используется, чтобы обеспечить дополнительную задержку между предшествующим и последующим кадром данных или кадром удаленного запроса данных.

Кадры данных и кадры удаленного запроса данных могут использоваться и в стандартном и в расширенном формате; они отделяются от предшествующих кадров межкадровым пространством.

Кадр данных (Data frame)

Наиболее часто используемый тип сообщения.

Кадр данных состоит из семи различных полей (см. рис. 2): “начало кадра” (start of frame), “поле арбитража” (arbitration field), “поле управления” (control field), “поле данных” (data field), “поле CRC” (CRC field), “поле подтверждения” (ACK field), “конец кадра” (end of frame). Поле данных может иметь нулевую длину.

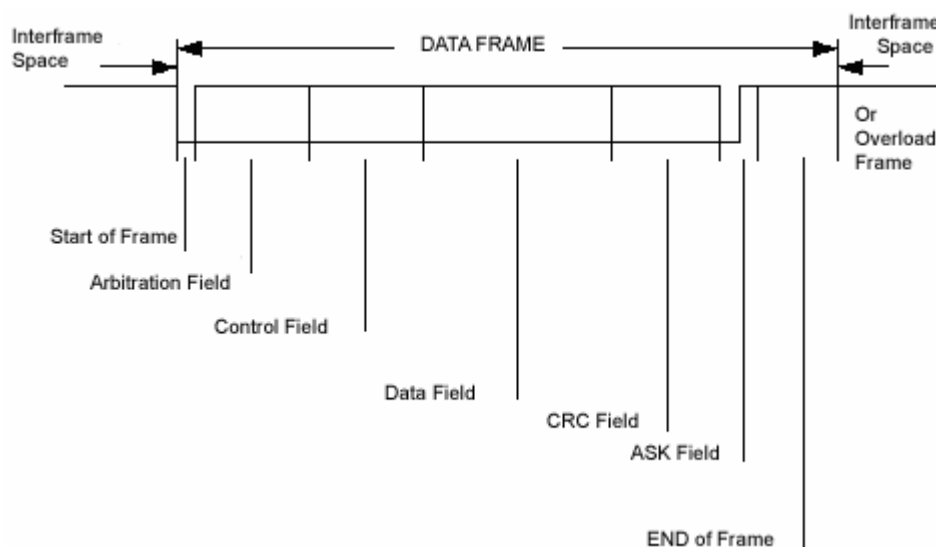


Рисунок 2. Структура кадра данных

Начало кадра (Start of frame)

Начало кадра отмечает начало кадра данных или кадра удаленного запроса данных. Это поле состоит из одиночного нулевого бита. Узлу разрешено начать передачу, когда шина свободна (см. “Межкадровый интервал”). Все узлы должны синхронизироваться по фронту, вызванному передачей поля “начало кадра” (см. “Аппаратная синхронизация”) узла, начавшего передачу первым.

Поле арбитража (Arbitration field)

Определяет приоритет сообщения в случае, когда два или более узлов одновременно пытаются передать данные в сеть. Формат поля арбитража является различным для стандартного и расширенного форматов (см. рис. 3):

- В стандартном формате поле арбитража состоит из 11-разрядного идентификатора и RTR-бита. Биты идентификатора обозначены ID-28 ... ID-18.

- В расширенном формате поле арбитража состоит из 29-разрядного идентификатора, SRR-бита, IDE-бита, и RTR-бита. Биты идентификатора обозначены ID-28 ... ID-0. Чтобы отличать стандартный формат и расширенный формат, зарезервированный в предыдущих спецификациях CAN (версия 1.0-1.2), бит r0 теперь обозначен как IDE бит.

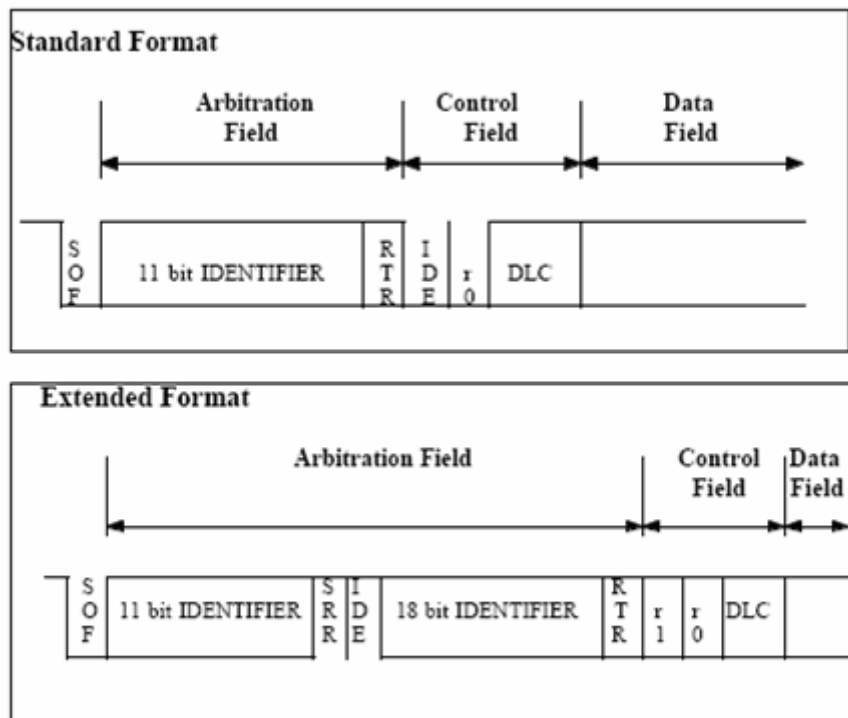


Рисунок 3. Стандартный и расширенный формат

Идентификатор

Поле идентификатора, несмотря на свое название никак, не идентифицирует само по себе ни узел в сети, ни содержимое поля данных.

Идентификатор – стандартный формат: длина идентификатора – 11 бит и соответствует Base ID в расширенном формате. Эти биты передаются в порядке ID-28 ... ID-18. Самый младший бит – ID-18. 7 старших битов (ID-28 – ID-22) не должны быть все единичными битами.

Идентификатор – расширенный формат: в отличие от стандартного идентификатора, расширенный идентификатор состоит из 29 бит. Его формат содержит две секции:

1. Base ID – 11 бит: эта секция передается в порядке от ID-28 до ID-18. Это эквивалентно формату стандартного идентификатора. Base ID определяет базовый приоритет расширенного кадра.
2. Extended ID – 18 бит: эта секция передается в порядке от ID-17 до ID-0. В стандартном кадре идентификатор сопровождается RTR битом.

Бит RTR (стандартный и расширенный формат)

Бит запроса удаленной передачи. В кадрах данных RTR бит должен быть передан нулевым уровнем. Внутри кадра удаленного запроса данных RTR бит должен быть единичным. В расширенном кадре сначала передается Base ID, с последующими битами IDE и SRR. Extended ID передается после SRR бита.

Бит SRR (расширенный формат)

Заменитель бита удаленного запроса. SRR – единичный бит. Он передается в расширенных кадрах в позиции RTR бита. Таким образом, он заменяет RTR бит стандартного кадра. Следовательно, при одновременной передаче стандартного кадра и расширенного кадра, Base ID которого совпадает с идентификатором стандартного кадра, стандартный кадр преобладает над расширенным кадром.

Бит IDE (расширенный формат)

Бит расширения идентификатора. Бит принадлежит:

- Полю арбитража для расширенного формата.
- Полю управления для стандартного формата. IDE бит в стандартном формате передается нулевым уровнем, в то время как в расширенном формате IDE бит – единичный уровень.

Поле управления (Control field)

Поле управления состоит из шести бит. Формат поля управления отличается для стандартного и расширенного формата (рис. 4). Кадры в стандартном формате включают: код длины данных (DLC), бит IDE, который передается нулевым уровнем (см. выше), и зарезервированный бит r0. Кадры в расширенном формате включают код длины данных и два зарезервированных бита r1 и r0. Зарезервированные биты должны быть посланы нулевым уровнем, но приемники принимают единичные и нулевые уровни биты во всех комбинациях.

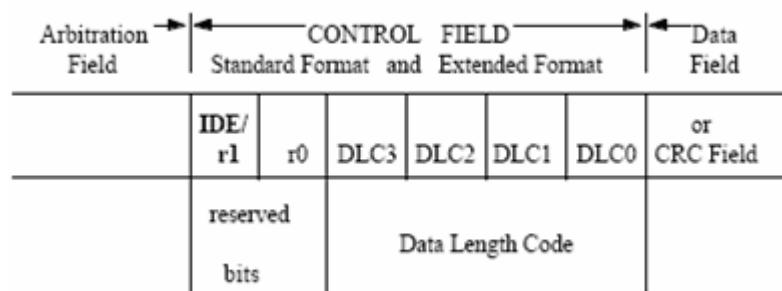


Рисунок 4. Формат поля управления

Число байт в поле данных обозначается кодом длины данных (Data Length Code, DLC). Этот код длины данных, размером 4 бита, передается внутри поля управления. Допустимое число байт данных: {0, 1, ..., 7, 8}. Другие величины использоваться не могут. В таблице d – “dominant”, r – “recessive”.

Число байт данных	Код длины данных (DLC)			
	DLC3	DLC2	DLC1	DLC0
0	d	d	d	d
1	d	d	d	r
2	d	d	r	d
3	d	d	r	r
4	d	r	d	d
5	d	r	d	r
6	d	r	r	d
7	d	r	r	r
8	r	d	d	d

Поле данных (Data field)

Поле данных состоит из данных, которые будут переданы внутри кадра данных. Оно может содержать от 0 до 8 байт, каждый содержит 8 бит, которые передаются, начиная с MSB.

Поле CRC (CRC field)

Содержит 15-битную контрольную сумму сообщения, которая используется для обнаружения ошибок, состоит из последовательности CRC и CRC – разделителя (рис. 5).

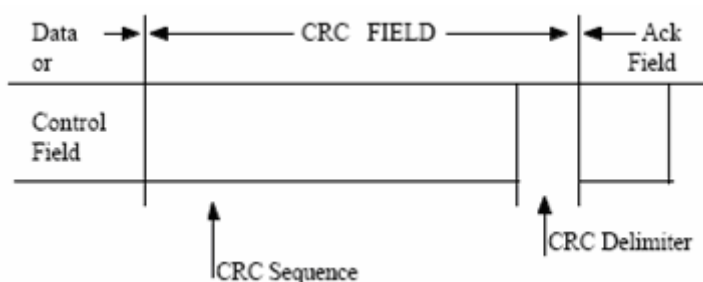


Рисунок 5. Формат поля CRC

Последовательность CRC (CRC Sequence)

Последовательность контроля кадра, получаемая из избыточного циклического кода, лучше всего подходит для кадров с числом битов меньше, чем 127 бит. При вычислении CRC, полином разделяется и определяется как полином, коэффициенты которого заданы последовательностью бит, состоящей из полей: “начало кадра”, “поле арбитража”, “управляющее поле”, “поле данных” (если есть) и, для 15 самых младших коэффициентов, 0. Этот полином делится на сгенерированный полином:

$$X^{15} + X^{14} + X^{10} + X^8 + X^7 + X^4 + X^3 + 1$$

Остаток от этого полиномиального деления и есть последовательность CRC, передаваемая по шине.

Разделитель CRC (CRC Delimiter)

Последовательность CRC сопровождается разделителем CRC, который состоит из одного единичного бита (“recessive”).

Поле подтверждения (ACK field)

Поле подтверждения имеет длину два бита и содержит (см. рис. 6): “область подтверждения” и “разделитель подтверждения”. В поле подтверждения передающий узел посылает два бита с единичным уровнем (“recessive”). Приемник, который получил сообщение правильно, сообщает об этом передатчику, посылая бит с нулевым уровнем (“dominant”) в течение приема поля “область подтверждения”.

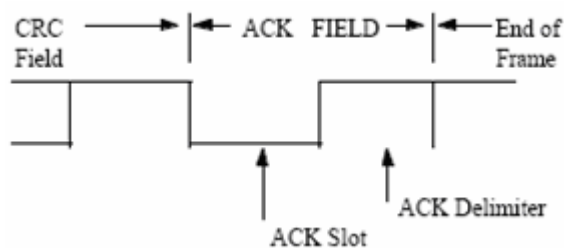


Рисунок 6. Формат поля подтверждения

Область подтверждения (ACK slot)

Все узлы, получившие соответствующую последовательность CRC, сообщают об этом во время приема поля “область подтверждения” путем замены бита с единичным уровнем на бит с нулевым уровнем.

Разделитель подтверждения (ACK delimiter)

Разделитель подтверждения – второй бит поля подтверждения, и он должен быть представлен единичным уровнем.

Конец кадра (End of frame)

Каждый кадр данных и кадр удаленного запроса данных ограничен последовательностью флагов, состоящей, из семи единичных бит (“recessive”).

Кадр удаленного запроса данных (Remote frame)

Инициация одним из узлов сети передачи в сеть данных другим узлом. Такая схема позволяет уменьшить суммарный трафик сети.

Узел, действующий как приемник некоторых данных, может инициировать передачу соответственных данных исходными узлами, посылая кадр удаленного запроса данных.

Кадр удаленного запроса данных существует и в стандартном формате и расширенном формате. В обоих случаях он состоит из шести битовых полей (см. рис. 7): “начало кадра” (Start of frame), “поле арбитража” (Arbitration field), “управляющее поле” (Control field), “поле CRC” (CRC field), “поле подтверждения” (ACK field), “конец кадра” (End of frame).

В отличие от кадра данных, RTR бит кадра удаленного запроса данных – единичный. В этом кадре отсутствует поле данных. При этом значение кода длины данных может принимать любое значение в пределах допустимого диапазона [0,8]. Значение кода длины данных соответствует коду длины данных кадра данных. RTR бит указывает, является ли переданный кадр кадром данных.

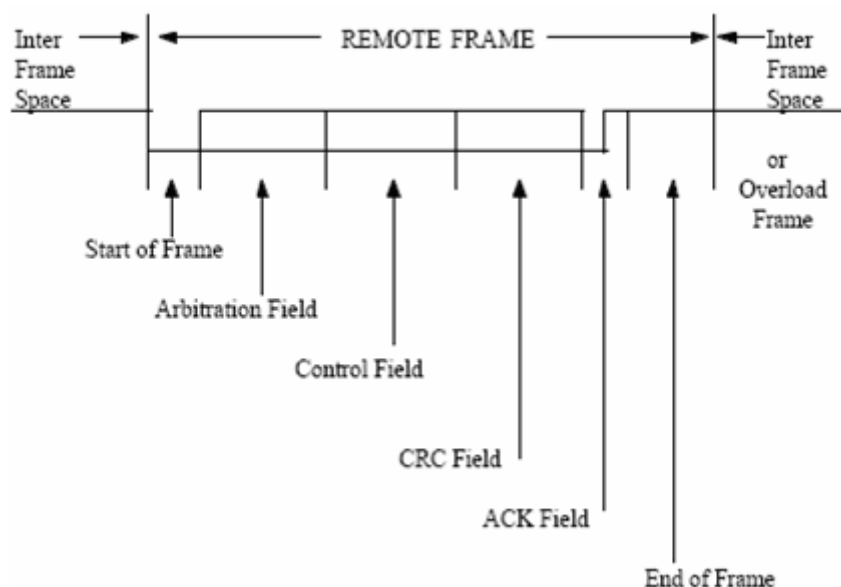


Рисунок 7. Формат кадра удаленного запроса данных

Кадр ошибки (Error Frame)

Явно нарушает формат сообщения CAN. Передача такого сообщения приводит к тому, что все узлы сети регистрируют ошибку формата CAN-кадра, и в свою очередь автоматически передают в сеть Error Frame. Результатом этого процесса является автоматическая повторная передача данных в сеть передающим узлом. Кадр ошибки состоит из двух различных полей (см. рис. 8). Первое поле получается суперпозицией флагов ошибки, полученных от различных узлов. Следующее поле – разделитель ошибки (Error Delimiter).

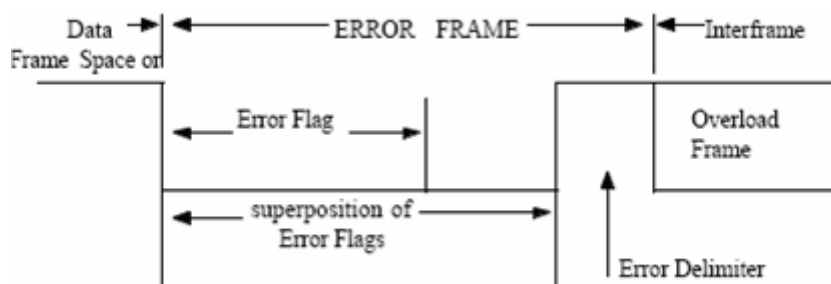


Рисунок 8. Формат кадра ошибки

Для правильного завершения кадра ошибки, узел в состоянии “пассивной ошибки” может нуждаться в доступе к шине, по крайней мере, на 3 битовых интервала (если имеет место локальная ошибка приемника, находящегося в состоянии “пассивной ошибки”). Следовательно, шина не должна быть загружена на 100 %.

Флаг ошибки (Error flag)

Имеются два вида флага ошибки: флаг активной ошибки и флаг пассивной ошибки.

1. Флаг активной ошибки состоит из шести последовательных нулевых бит (“dominant”).
2. Флаг пассивной ошибки состоит из шести последовательных единичных бит (“recessive”), если они не перезаписаны нулевыми битами других узлов.

Узел в состоянии “активной ошибки”, обнаружив условие ошибки, сообщает об этом передачей флага активной ошибки. Форма флага ошибки нарушает закон заполнения

бита (см. “Кодирование”), применяемый ко всем полям от поля “начало кадра” до разделителя CRC, или разрушает фиксированную форму поля подтверждения или поля “конец кадра”. Как следствие, все другие узлы обнаруживают условие ошибки и в свою очередь начинают передачу флага ошибки. Таким образом, последовательность “dominant” битов, которая фактически может появиться на шине, получается суперпозицией различных флагов ошибки, переданных отдельными узлами. Суммарная длина этой последовательности изменяется от шести до двенадцати бит.

Узел, в состоянии “пассивной ошибки”, обнаружив условие ошибки, пробует сообщить об этом передачей флага пассивной ошибки, он ожидает появления шести последовательных одинаковых бит, начинающих флаг пассивной ошибки. Флаг пассивной ошибки завершен после обнаружения этих 6 бит.

Разделитель ошибки (Error Delimiter)

Разделитель ошибки состоит из восьми “recessive” битов. После передачи флага ошибки каждый узел посылает “recessive” биты и проверяет шину, пока не обнаруживает “recessive” бит. Далее он начинает передачу еще семи “recessive” битов.

Кадр перегрузки (Overload Frame)

Кадр перегрузки содержит два битовых поля (см. рис. 9): флаг перегрузки и разделитель перегрузки.

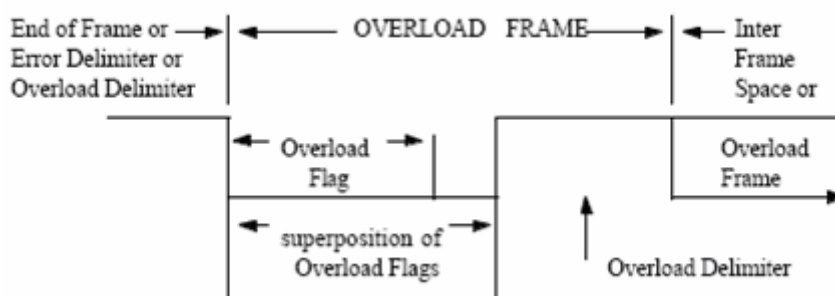


Рисунок 9. Формат кадра перегрузки

Есть условия, которые приводят к передаче флага перегрузки:

1. Внутреннее состояние приемника требует задержки следующего кадра данных или кадра удаленного запроса данных.
2. Обнаружение “dominant” бита при передаче первого и второго битов перерыва.
3. Если узел обнаруживает “dominant” бит на восьмом бите (последнем бите) разделителя ошибки или разделителя перегрузки, это повлечет передачу кадра перегрузки (а не кадра ошибки). Счетчики ошибок не будут инкрементированы.

Передачу кадра перегрузки, обусловленного 1-м видом перегрузки, разрешено начинать в первом битовом интервале предусмотренного перерыва, в то время как кадр перегрузки, обусловленный 2-м и 3-м видами перегрузки, начинает передаваться на один бит позже обнаружения “dominant” бита.

Не больше двух кадров перегрузки может быть сгенерировано, чтобы задержать следующий кадр данных или кадр удаленного запроса данных.

Флаг перегрузки (Overload Flag)

Состоит из шести “dominant” битов. Полная форма соответствует флагу активной ошибки. Форма флага перегрузки нарушает фиксированную форму поля перерыва. Все другие узлы также обнаруживают условие перегрузки и в свою очередь начинают передачу флага перегрузки. В случае если обнаружен “dominant” бит, во время 3-го бита перерыва, то этот бит интерпретируется как начало кадра.

Замечание:

Контроллеры, базирующиеся на CAN 1.0 и 1.1, по-другому интерпретируют 3-й бит перерыва. Если в это время будет обнаружен “dominant” бит, то эти узлы интерпретируют его как поле “начало кадра”; шестой “dominant” бит нарушает правило заполнения бит и вызовет ошибку.

Разделитель перегрузки (Overload Delimiter)

Состоит из восьми “recessive” бит. Разделитель перегрузки имеет такую же форму, как и разделитель ошибки. После передачи флага перегрузки узел контролирует шину, пока не обнаружит переход от “dominant” бита к “recessive” биту. В этот момент времени каждый узел заканчивает передачу флага перегрузки, и все узлы начинают передачу еще 7 “recessive” битов.

Межкадровое пространство (Interframe Spacing)

Кадры данных и кадры удаленного запроса данных отделяются от предшествующих кадров любого типа (кадр данных, кадр удаленного запроса данных, кадр ошибки, кадр перегрузки) битовым полем, называемым межкадровым пространством. В отличие от них, кадрам перегрузки и кадрам ошибки не предшествует межкадровое пространство (кратные кадры перегрузки также не отделяются интервалами).

Межкадровое пространство содержит битовые поля “перерыв” и “простой шины”, а для узла в состоянии “пассивной ошибки”, который был передатчиком предыдущего сообщения, еще и поле “приостановка передачи”.

Рисунок 10 (a) приведен для узла, который был приемником предыдущего сообщения. Рисунок 10 (b) приведен для узла в состоянии “пассивной ошибки”, который был передатчиком предыдущего сообщения.

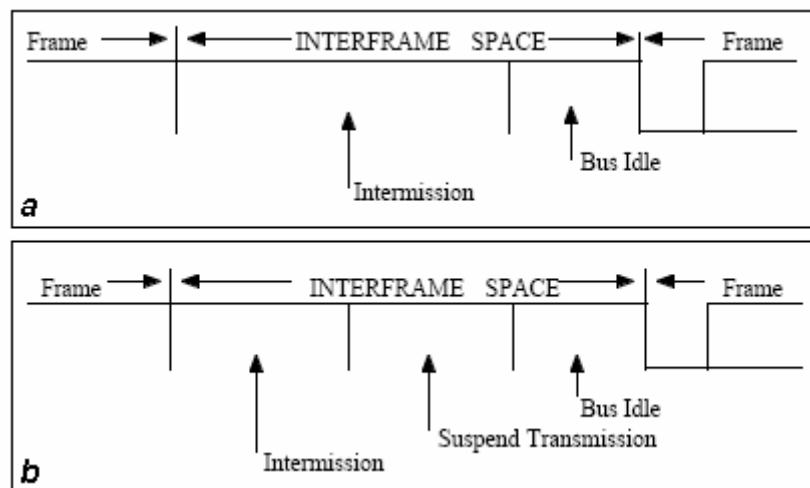


Рисунок 10. Межкадровое пространство

Перерыв (Intermission)

Состоит из трех “recessive” бит. В течение перерыва единственное действие, которое можно предпринять, – сообщить об условии перегрузки, и никакой узел не сможет начать передачу кадра данных или кадра удаленного запроса данных.

Замечание:

Если узел имеет сообщение, которое нужно передать, и он выявляет “dominant” бит в третьем бите перерыва, это интерпретируется, как бит “начало кадра”, и, со следующего бита, он начинает передавать сообщение, с первым битом его идентификатора, не передавая предварительно, бита “начало кадра”.

Простой шины (Bus Idle)

Период простоя шины может иметь произвольную длину. Шина свободна и любой узел, которому нужно что-либо передать, может обращаться к шине. Сообщение, которое ждет передачи во время передачи другого сообщения, начинает передаваться в первом бите после перерыва. Обнаружение “dominant” бита на шине интерпретируется как “начало кадра”.

Приостановка передачи (Suspend Transmission)

После того, как узел в состоянии “пассивной ошибки” передал сообщение, он посылает восемь “recessive” бит, после которых идет перерыв, перед передачей следующего сообщения или распознавания простоя шины. Если тем временем начинается передача, вызванная другим узлом, узел становится приемником этого сообщения.

Определение передатчика / приемника**Передатчик**

Узел, инициирующий сообщение называется передатчиком этого сообщения. Узел остается передатчиком до освобождения шины или потери арбитража.

Приемник

Узел называется приемником сообщения, если он не является передатчиком этого сообщения и шина не простаивает.

Фильтрация сообщений

Фильтрация сообщения основана на идентификации. Существуют специальные регистры масок, которые разрешают установку любого бита идентификатора, могут использоваться для выбора группы идентификаторов, которые будут отображены в буферах.

Если регистры масок есть, то каждый бит регистров маски должен быть программируемым, то есть его можно использовать для фильтрации сообщения. Длина регистра маски может составлять целый идентификатор или только его часть.

Проверка допустимости сообщения

Момент времени, при котором сообщение является допустимым, различен для передатчика и приемников сообщения.

Передатчик

Сообщение допустимо для передатчика, если нет никакой ошибки до конца сообщения. Если сообщение разрушено, повторная передача будет следовать автоматически и согласно приоритетам. Чтобы конкурировать за доступ к шине с другими сообщениями, повторная передача должна начаться, как только, шина станет свободна.

Приемник

Сообщение допустимо для приемников, если нет никакой ошибки до предпоследнего бита “конец кадра”. Значение последнего бита “конец кадра” рассматривается как безразличное, “dominant” значение не ведет к ошибке формы.

Кодирование

Кодирование последовательности битов

Сегменты “начало кадра”, “поле арбитража”, “поле управления”, “поле данных” и “последовательность CRC” кодируются методом разрядного заполнения. Всякий раз, когда передатчик обнаруживает в разрядном потоке, пять последовательных одинаковых бит, он автоматически вставляет дополнительный бит в фактический переданный разрядный поток.

Оставшиеся битовые поля кадра данных или кадра удаленного запроса данных (“разделитель CRC”, “поле подтверждения”, и “конец кадра”) имеют фиксированную форму. Кадр ошибки и кадр перегрузки имеют фиксированную форму также и не кодируются методом разрядного заполнения.

Разрядный поток сообщения кодируется согласно NRZ методу (без возврата к нулю). Это означает, что в течение всего времени передачи битов сгенерированный разрядный уровень является или “dominant”, или “recessive”.

Обработка ошибок

Обнаружение ошибок

Существует 5 различных типов ошибки (взаимно не исключают друг друга):

- Ошибка бита (Bit Error)
Узел, который передает данные на шину, осуществляет контроль шины. Ошибка бита имеет место, если значение бита на шине отличается от переданного значения. Исключение – посылка “recessive” бита в течение заполненного битового потока поля арбитража или в течение поля подтверждения. Передатчик, посылающий флаг пассивной ошибки и обнаруживший “dominant” бит, не интерпретирует это как ошибку бита.
- Ошибка заполнения (Stuff Error)
Ошибка заполнения должна быть обнаружена во время передачи 6-ого бита из последовательности 6-ти одинаковых бит в поле сообщения, которое должно быть закодировано методом разрядного заполнения.
- Ошибка CRC (CRC Error)
Последовательность CRC состоит из результата вычисления CRC передатчиком. Приемники вычисляют CRC таким же образом, как и передатчик. Ошибка CRC имеет место, если расчетный результат не совпадает с результатом, вычисленным из принятых данных.
- Ошибка формы (Form Error)
Ошибка формы обнаруживается, если битовое поле фиксированного формата содержит одни или более запрещенных битов. (Примечание: для приемника “dominant” бит в течение последнего бита “конец кадра” не интерпретируется как ошибка формы).
- Ошибка подтверждения (Acknowledgement Error)
Ошибка подтверждения обнаруживается передатчиком всякий раз, когда он не обнаруживает “dominant” бит в “области подтверждения”.

Передача сигналов ошибки

Узел, обнаруживший ошибку сообщает об этом, передавая флаг ошибки. Для узла в состоянии “активной ошибки” – это флаг активной ошибки, для узла в состоянии “пассивной ошибки” – это флаг пассивной ошибки.

Всякий раз, когда каким-либо узлом обнаружена ошибка бита, ошибка заполнения, ошибка формы или ошибка подтверждения, со следующего бита начинается передача флага ошибки соответствующим узлом.

Всякий раз, когда обнаружена ошибка CRC, передача флага ошибки начинается с бита, следующего после разделителя подтверждения, если не передается флаг ошибки для другого условия.

Типизация неисправностей

Неисправный узел может быть в одном из трех состояний:

- активной ошибки (Error Active);
- пассивной ошибки (Error Passive);
- отключения от шины (Bus Off).

Узел в состоянии “активной ошибки” нормально присоединен к шине и посылает флаг активной ошибки при обнаружении ошибок.

Узел в состоянии “пассивной ошибки” не должен посылать флаг активной ошибки. Он подключен к шине, но при обнаружении ошибок, посылает флаг пассивной ошибки. После передачи, узел в состоянии “пассивной ошибки” будет ждать инициализации дальнейшей передачи (см. “Приостановка передачи”).

Узел в состоянии “отключения от шины” не может работать с шиной.

Для типизации неисправностей в каждом узле есть два счетчика:

1. Счетчик ошибок передачи (Transmit Error Counter).
2. Счетчик ошибок приема (Receive Error Counter).

Эти счетчики увеличиваются или уменьшаются в соответствии с несколькими правилами. Сами правила управления счетчиками ошибок достаточно сложны, но сводятся к простому принципу, ошибка передачи приводит к увеличению Transmit Error счетчика на 8, ошибка приема увеличивает счетчик Receive Error на 1, любая корректная передача/прием сообщения уменьшают соответствующий счетчик на 1. Эти правила приводят к тому, что счетчик ошибок передачи передающего узла увеличивается быстрее, чем счетчик ошибок приема принимающих узлов. Это правило соответствует предположению о большой вероятности того, что источником ошибок является передающий узел.

Когда, значение хотя бы одного из двух счетчиков ошибок превышает предел 127, узел переходит в состояние Error Passive. Когда значение хотя бы одного из двух счетчиков превышает предел 255, узел переходит в состояние Bus Off.

Требования к битовой синхронизации

Номинальная битовая скорость

Номинальная скорость передачи информации в битах – число битов в секунду, переданное в отсутствие пересинхронизации идеальным передатчиком.

Номинальное время передачи бита = 1/Номинальная битовая скорость

Номинальное время передачи бита можно разделить на несколько не перекрывающихся участков (см. рис. 11):

- Сегмент синхронизации (SYNC_SEG)
- Сегмент времени распространения (PROP_SEG)
- Сегмент1 буфера фазы (PHASE_SEG1)
- Сегмент2 буфера фазы (PHASE_SEG2)

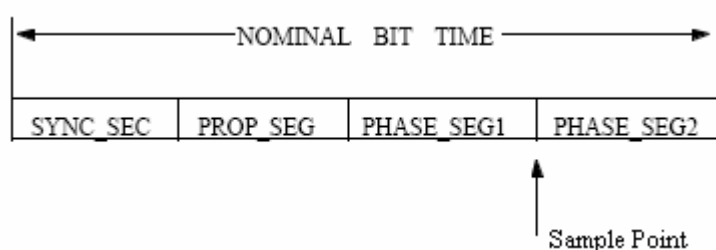


Рисунок 11. Структура времени передачи бита

Сегмент синхронизации (SYNC_SEG)

Используется для синхронизации различных узлов на шине. Предполагается, что фронт сигнала должен находиться в пределах этого сегмента.

Сегмент времени распространения (PROP_SEG)

Используется, чтобы компенсировать физические время запаздывания в пределах сети. Это удвоенная сумма времени распространения сигнала на линии шины, входной задержки компаратора, и задержки выходного формирователя.

Сегменты буферов фазы (PHASE_SEG1, PHASE_SEG2)

Эти сегменты используются, чтобы компенсировать ошибки фазы. Эти сегменты могут быть удлинены или укорочены пересинхронизацией.

Точка считывания (Sample Point)

Момент времени, при котором уровень шины читается и интерпретируется, как значение соответственного бита. Она расположена в конце PHASE_SEG1.

Время обработки информации (Information Processing Time)

Время обработки информации – сегмент времени, начинающийся с точки считывания.

Шаг квантования времени (Time Quantum)

Шаг квантования времени – фиксированная единица времени, полученная из периода генератора. Существует программируемый делитель, со значениями, от 1 до 32 (m). При старте с минимальным шагом квантования времени, шаг квантования времени может иметь длину:

Шаг квантования = $m * \text{минимальный шаг квантования времени}$,
где m – значение делителя.

Длительность сегментов

SYNC_SEG – 1 шаг квантования времени.

PROP_SEG – программируется, может быть 1, 2, ..., 8 квантов времени.

PHASE_SEG1 – программируется, может быть 1, 2... 8 квантов времени.

PHASE_SEG2 – максимум из PHASE_SEG1 и времени обработки информации.

Время обработки информации меньше или равно 2 квантам времени.

Общее число квантов времени в битовом интервале должно быть программируемым, по крайней мере, от 8 до 25.

Аппаратная синхронизация (Hard Synchronization)

После аппаратной синхронизации внутреннее битовое время каждого узла перезапускается с SYNC_SEG.

Переход пересинхронизации (Resynchronization Jump Width)

В результате пересинхронизации PHASE_SEG1 может быть удлинен, или PHASE_SEG2 может быть укорочен. Размер удлинения или укорачивания этих сегментов имеет предел, определяемый переходом пересинхронизации. Переход пересинхронизации должен быть программируемым от 1 до минимума из (4, PHASE_SEG1).

Ошибка фазы границы (Phase Error of an edge)

Ошибка фазы границы определяется положением фронта сигнала относительно SYNC_SEG, измеряется в квантах времени. Знак ошибки фазы определяется следующим образом:

- $e = 0$, если фронт сигнала находится в пределах SYNC_SEG.
- $e > 0$, если фронт сигнала находится перед точкой считывания.
- $e < 0$, если фронт сигнала находится после точки считывания предыдущего бита.

Пересинхронизация (Resynchronization)

Эффект от пересинхронизации такой же как в случае аппаратной синхронизации, когда величина ошибки фазы границы, которая вызывает пересинхронизацию меньше или равна программируемому значению перехода пересинхронизации. Когда величина ошибки фазы большая, чем ширина перехода пересинхронизации и:

- если ошибка фазы положительна, то PHASE_SEG1 удлиняется на ширину перехода пересинхронизации.
- если ошибка фазы отрицательна, то PHASE_SEG2 укорачивается на ширину перехода пересинхронизации.

Правила синхронизации (Synchronization Rules)

Аппаратная синхронизация и пересинхронизация – две формы синхронизации. Они удовлетворяют следующим правилам:

1. Допускается только одна синхронизация в пределах битового интервала.
2. Фронт сигнала будет использоваться для синхронизации только, если значение, обнаруженное при предыдущей точке считывания (предыдущее значение на шине), отличается от значения на шине сразу после фронта.
3. Аппаратная синхронизация происходит всякий раз, когда есть переход от “recessive” бита к “dominant” в течение простоя шины.

Драйвер CAN для микроконтроллера LPC2292

В микроконтроллере LPC2292 два контроллера CAN.

```
// Определение двух CAN-контроллеров
#define CAN0 0
#define CAN1 1

// Определение ошибок (возвращаемое функцией значение)
#define OK 0
#define ERROR 1

#define VIC_CANRX1_bit (1 << VIC_CANRX1) // Формирование маски прерывания (прием по CAN0)
#define VIC_CANRX2_bit (1 << VIC_CANRX2) // Формирование маски прерывания (прием по CAN1)
#define VIC_CANERR_bit (1 << VIC_CANERR) // Формирование маски прерывания (ошибки в CAN)

#define MAX_CAN_PORT 2 // Количество контроллеров CAN
#define MAX_CAN_PIPE_SIZE 16 // Размер буфера (очереди)

// Определение типа данных для хранения сообщений CAN
typedef struct
{
    unsigned int frame; // Биты 16..19: код длины данных (DLC)
                        // Бит 30 устанавливается, как бит запроса удаленной передачи (RTR)
                        // Бит 31: устанавливается, если это сообщение расширенного формата
                        // (29-битный идентификатор)
    unsigned int id; // Идентификатор сообщения CAN (11-битный или 29-битный)
    unsigned int data_a; // Поле данных сообщения CAN (0-3 байты)
    unsigned int data_b; // Поле данных сообщения CAN (4-7 байты)
} CANMSG;

// Определение типа данных, описывающего буфер (очередь) сообщений на прием
typedef struct {
    CANMSG msg[ MAX_CAN_PIPE_SIZE ]; // Очередь сообщений
    unsigned short start; // "Голова" очереди (очередь кольцевая)
    unsigned short size; // Размер активной очереди (т. е. количество сообщений, которые
                        // были не использованы); "хвост" = start+size
} CANPIPE;

// Таблица со смещениями до регистров контроллеров CAN1 и CAN2
static unsigned long offset_table[2] = { 0x00000000L, 0x00001000L };

static char error_handler_install = 0; // =0, если обработчик ошибок CAN не установлен

static int can_baud[2];

static CANPIPE can_pipe[ MAX_CAN_PORT ]; // Очереди сообщений для всех контроллеров CAN

/*-----
init_can
-----*/
Инициализация CAN

Вход: interface - тип CAN (0 или 1),
      baud - скорость передачи по каналу
Выход: нет
Результат: ERROR или OK
-----*/

int init_can( int interface, int baud )
{
    unsigned int *pSFR; // указатель на регистры SFR
    unsigned int offset; // смещение, добавляемое к pSFR

    if( OK != check_can_interface( interface ) ) return ERROR;

    init_can_pipe();

    AFMR = 0x00000002;

    can_baud[ interface ] = baud;

    offset = offset_table[ interface ];
```

```

pSFR = (unsigned int *) &ClMOD + offset;    // Выбрать регистр режима работы
*pSFR = 1;    // Установить режим сброса (перезагрузки)

pSFR = (unsigned int *) &ClIER + offset;    // Выбрать регистр разрешения прерываний
*pSFR = 0;    // Запретить все прерывания

pSFR = (unsigned int *) &ClGSR + offset;    // Выбрать глобальный регистр статуса
*pSFR = 0;    // Очистить глобальный регистр статуса

pSFR = (unsigned int *) &ClBTR + offset;    // Выбрать регистр временных характеристик шины
*pSFR = baud;    // Установить время передачи бита

switch ( interface )
{
    case CAN0:

        VICIntSelect &= ~VIC_CANRX1_bit;    // Тип прерывания IRQ
        // Адрес обработчика прерываний
        VICVectAddr1 = (unsigned int)&CANAll_CANISR_Rx1;
        VICVectCntl1 = 0x20 | VIC_CANRX1;    // Включить векторное прерывание
        VICIntEnable = VIC_CANRX1_bit;    // Разрешить прерывание
        break;

    case CAN1:

        VICIntSelect &= ~VIC_CANRX2_bit;    // Тип прерывания IRQ
        // Адрес обработчика прерываний
        VICVectAddr2 = (unsigned int)&CANAll_CANISR_Rx2;
        VICVectCntl2 = 0x20 | VIC_CANRX2;    // Включить векторное прерывание
        VICIntEnable = VIC_CANRX2_bit;    // Разрешить прерывание
        break;

    default: return ERROR;
}

if ( error_handler_install == 0 )
{

    VICIntSelect &= ~VIC_CANERR_bit;    // Тип прерывания IRQ
    // Адрес обработчика прерываний
    VICVectAddr5 = (unsigned int)&CANAll_CANISR_Err;
    VICVectCntl5 = 0x20 | VIC_CANERR;    // Включить векторное прерывание
    VICIntEnable = VIC_CANERR_bit;    // Разрешить прерывание

    error_handler_install = 1;
}

pSFR = (unsigned int *) &ClIER + offset;    // Выбрать регистр разрешения прерываний
pSFR = 0x1;    // Разрешить прерывание на прием

// Установка нормального режима работы CAN
pSFR = (unsigned int *) &ClMOD + offset;    // Выбрать регистр режима работы
pSFR = 0;    // Режим работы контроллера CAN (выполняет действия)

return OK;
}

/*-----
                                write_can
-----*/
Передача сообщений по CAN

Вход:  interface - тип CAN (0 или 1),
       pTransmitBuf - адрес буфера передачи
Выход: нет
Результат: ERROR или OK
-----*/

int write_can_packet ( int interface, CANMSG *pTransmitBuf )
{
    unsigned int *pAddr;
    unsigned int *pCandata;
    unsigned int offset;

    if ( OK != check_can_interface( interface ) ) return ERROR;

    offset = offset_table[ interface ];

```



```

pAddr = (unsigned int *) &C1SR + offset;    // Выбрать регистр статуса
// Проверка: доступен ли канал передачи
if (!(*pAddr & 0x00000004L))
{
    // Канал передачи не доступен
    return ERROR;
}

// Запись DLC, RTR и FF для передаваемого сообщения
pAddr = (unsigned int *) &C1TF11 + offset; // Регистр информации о кадре передачи
*pAddr = pTransmitBuf->frame & 0xC00F0000L; // Установка DLC (8 байт), RTR и FF
                                           // (расширенный формат сообщения)

// Запись идентификатора сообщения в регистр идентификатора передачи
pAddr++;
*pAddr = pTransmitBuf->id;

// Запись первых 4 байтов сообщения в регистр1 передаваемых данных
pCandata = (unsigned int *) &(pTransmitBuf->data_a);
pAddr++;
*pAddr = *pCandata;

// Запись вторых 4 байтов сообщения в регистр2 передаваемых данных
pCandata++;
pAddr++;
*pAddr = *pCandata;

// Работа с регистром команд
pAddr = (unsigned int *) &C1CMR + offset;    // Выбрать регистр команд
*pAddr = 0x30; // Выбрать Self Transmission Request и буфер1 для передачи

return OK;
}

/*-----
                        read_can
-----*/

Прием сообщений из CAN

Вход:  interface - тип CAN (0 или 1),
       pReceiveBuf - адрес буфера приема

Выход:      нет
Результат:  ERROR или OK
-----*/

int read_can_packet ( int interface, CANMSG *pReceiveBuf )
{
    if ( OK != check_can_interface( interface ) ) return ERROR;

    return can_pipe_read( interface, pReceiveBuf );
}

/*-----
                        check_busoff
-----*/

Проверка нахождения контроллера в состоянии BusOff

Вход:  interface - тип CAN (0 или 1)

Выход:      нет
Результат:  ERROR или OK
-----*/

int check_busoff( int interface )
{
    unsigned int *pSFR;
    unsigned int offset;

    if ( OK != check_can_interface( interface ) ) return ERROR;

    __disable_interrupt();    // Запретить прерывания

    offset = offset_table[ interface ];

    pSFR = (unsigned int *) &C1GSR + offset;    // Выбрать глобальный регистр статуса

    // Проверка: если контроллер находится в состоянии BusOff, то инициализировать CAN
    if ( *pSFR & 0x80 ) init_can( interface, can_baud[interface] );
}

```

```
__enable_interrupt();          // Разрешить прерывания

return OK;

}

/*-----
                        check_can_interface
-----*/
Проверка корректности обращения к CAN (по номеру) .

Вход:  interface - тип CAN (0 или 1)

Выход:      нет
Результат:  ERROR или OK
-----*/

static int check_can_interface( int interface )
{
    if( interface > CAN1 ) return ERROR;
    else return OK;
}

/*-----
                        Interrupt_Service_Routine
-----*/
Обработчик прерывания по ошибкам приема/передачи по CAN
-----*/
static void CANAll_CANISR_Err ( void )
{
    // Проверка на переполнение счетчика ошибок передачи (> 255)
    if ( C1SR_bit.BS )
    {
        C1MOD = 0;
    }

    if ( C2SR_bit.BS )
    {
        C2MOD = 0;
    }

    VICVectAddr = 0xFFFFFFFFL;    // Подтверждение прерывания
}

/*-----
                        Interrupt_Service_Routine
-----*/
Обработчик прерывания по приему сообщений CAN0.
-----*/
static void CANAll_CANISR_Rx1 ( void )
{
    CANMSG msg;

    msg.frame = C1RFS;    // Чтение характеристик принятого сообщения (DLC, RTR, FF,...)
    msg.id     = C1RID;    // Чтение идентификатора принятого сообщения
    msg.data_a = C1RDA;    // Чтение первой половины сообщения (0-4 байта)
    msg.data_b = C1RDB;    // Чтение второй половины сообщения (5-7 байта)

    // Запись сообщения в очередь первого контроллера CAN
    can_pipe_write( CAN0, &msg );

    C1CMR = 0x04; // Освобождение буфера приема
    VICVectAddr = 0xFFFFFFFFL;    // Подтверждение прерывания
}

/*-----
                        Interrupt_Service_Routine
-----*/
Обработчик прерывания по приему сообщений CAN1.
-----*/
static void CANAll_CANISR_Rx2 ( void )
{
    CANMSG msg;

    msg.frame = C2RFS;    // Чтение характеристик принятого сообщения (DLC, RTR, FF,...)
    msg.id     = C2RID;    // Чтение идентификатора принятого сообщения
    msg.data_a = C2RDA;    // Чтение первой половины сообщения (0-4 байта)
```

```
msg.data_b = C2RDB;    // Чтение второй половины сообщения (5-7 байта)

// Запись сообщения в очередь второго контроллера CAN
can_pipe_write( CAN1, &msg ) ;

C2CMR = 0x04; // Освобождение буфера приема
VICVectAddr = 0xFFFFFFFFL;    // Подтверждение прерывания
}

/*-----
                        can_pipe_write
-----*/

Запись сообщений в очередь CAN (по номеру).

Вход:  interface - тип CAN (0 или 1),
       msg - записываемое сообщение

Выход:      нет
Результат:  ERROR или OK
-----*/

static int can_pipe_write( int interface, CANMSG * msg )
{
    unsigned short index;

    if ( OK != check_can_interface( interface ) ) return ERROR;

    // Проверка: размер активной очереди не должен быть больше максимального размера очереди
    if ( can_pipe[ interface ].size < MAX_CAN_PIPE_SIZE )
    {
        // Определение номера свободного элемента очереди для записи нового сообщения
        index = ( can_pipe[ interface ].start + can_pipe[ interface ].size ) & (
            MAX_CAN_PIPE_SIZE - 1 );

        can_pipe[ interface ].msg[ index ] = *msg;    // Запись сообщения в очередь

        can_pipe[ interface ].size++;                // Увеличение размера активной очереди

        return OK;
    }
    else return ERROR;
}

/*-----
                        can_pipe_read
-----*/

Чтение сообщения из очереди CAN (по номеру).

Вход:  interface - тип CAN (0 или 1)

Выход:      msg - прочитанное сообщение (принятое по CAN)
Результат:  ERROR или OK
-----*/

static int can_pipe_read( int interface, CANMSG * msg )
{
    if( OK != check_can_interface( interface ) ) return ERROR;

    // Чтение возможно только тогда, когда еще не все сообщения были прочитаны из нее
    if ( can_pipe[ interface ].size == 0 ) return ERROR;

    // Чтение сообщения
    *msg = can_pipe[ interface ].msg[ can_pipe[ interface ].start ];

    can_pipe[ interface ].start++;    // Смещение "головы" на одну позицию
    can_pipe[ interface ].start &= ( MAX_CAN_PIPE_SIZE - 1 );
    can_pipe[ interface ].size--;    // Уменьшение размера активной очереди

    return OK;
}

/*-----
                        init_can_pipe
-----*/

Инициализация очередей контроллеров CAN.

Вход:      нет
Выход:      нет
```

Результат: **нет**

-----*/

```

static void init_can_pipe( void )
{
    int i;

    for( i = 0; i < MAX_CAN_PORT; i++ )
    {
        can_pipe[i].start = 0;
        can_pipe[i].size  = 0;
    }
}

```

Описание регистров

Название	Описание	Доступ	Адрес и название	
			CAN1	CAN2
CANMOD	Управляет режимом работы контроллера CAN	Чтение/запись	0xE0044000 C1MOD	0xE0048000 C2MOD
CANCMR	Командные биты, влияющие на состояние контроллера CAN	Только запись	0xE0044004 C1CMR	0xE0048004 C2CMR
CANGSR	Глобальный регистр статуса и счетчиков ошибок	Только чтение	0xE0044008 C1GSR	0xE0048008 C2GSR
CANIER	Разрешение прерываний	Чтение/запись	0xE0044010 C1IER	0xE0048010 C2IER
CANBTR	Временные характеристики шины	Чтение/запись	0xE0044014 C1BTR	0xE0048014 C2BTR
CANSR	Регистр статуса	Только чтение	0xE004401C C1SR	0xE004801C C2SR
CANRFS	Статус полученного сообщения	Чтение/запись	0xE0044020 C1RFS	0xE0048020 C2RFS
CANRID	Идентификатор в принятом сообщении	Чтение/запись	0xE0044024 C1RID	0xE0048024 C2RID
CANRDA	Поле данных (1-4 байт) в принятом сообщении	Чтение/запись	0xE0044028 C1RDA	0xE0048028 C2RDA
CANRDB	Поле данных (5-8 байт) в принятом сообщении	Чтение/запись	0xE004402C C1RDB	0xE004802C C2RDB
CANTFI1	Информация о передаваемом сообщении	Чтение/запись	0xE0044030 C1TFI1	0xE0048030 C2TFI1
CANTID1	Идентификатор передаваемого сообщения	Чтение/запись	0xE0044034 C1TID1	0xE0048034 C2TID1
CANTDA1	Поле данных (1-4 байт) в передаваемом сообщении	Чтение/запись	0xE0044038 C1TDA1	0xE0048038 C2TDA1
CANTDB1	Поле данных (5-8 байт) в передаваемом сообщении	Чтение/запись	0xE004403C C1TDB1	0xE004803C C2TDB1

Регистр режима работы CANMOD

Биты	Название	Функция
0	RM	0 – контроллер CAN работает, некоторые регистры недоступны для записи; 1 – режим перезагрузки (Reset Mode) – CAN недоступен, в регистры, доступные для записи, можно записывать.
1	LOM	0 – контроллер CAN подтверждает, что сообщение было успешно принято; 1 – режим прослушивания (Listen Only Mode) – контроллер не дает подтверждений того, что сообщение было успешно принято. Сообщения не могут посылаться, контроллер находится в состоянии “пассивной ошибки”. Этот режим предусмотрен для программного определения скорости передачи информации (битовой скорости) и “горячего включения”.
2	STM	0 – переданное сообщение должно быть подтверждено для того, чтобы передача считалась успешной; 1 – режим самостоятельного тестирования (Self Test Mode) – контроллер будет считать переданное сообщение успешным, если не будет подтверждений. Этот режим можно использовать вместе с установкой бита SRR в регистре CANCMR.
3	TPM	0 – очередность трех буферов передачи зависит от идентификаторов их CAN; 1 – очередность трех буферов передачи зависит от соответствующих полей (Tx Priority Fields в регистре CANTFI).
4	SM	0 – нормальный режим работы; 1 – режим “сна” (Sleep Mode) – контроллер CAN находится в таком режиме, если нет запросов на прерывание и нет активности на шине.
5	RPM	0 – на выводах шины (RX и TX) под уровнем “dominant” подразумевается нулевой сигнал; 1 – режим смены полярности сигнала (Reverse Polarity Mode) – на выводе RX под уровнем “dominant” подразумевается единичный сигнал.
7	TM	0 – нормальный режим работы; 1 – режим тестирования (Test Mode) – вывод RX замыкается на вывод TX.

Командный регистр CANCMR

Запись в этот регистр вызывает выполнение какого-либо действия.

Биты	Название	Функция
0	TR	1 – запрос передачи (Transmission Request) – сообщение, уже записанное в регистры CANTFI, CANTID и CANTDA, CANTDB (необязательно), помещается в очередь на передачу.
1	AT	1 – прекращение передачи (Abort Transmission) – невыполненный запрос на передачу отменяется.
2	RRB	1 – освобождение буферов приема (Release Receive Buffer) – регистры CANRFS, CANRID и CANRDA, CANRDB (если используются) освобождаются, и в них может быть записан следующий принятый кадр. Если следующий принятый кадр не доступен, то выполнение этой команды очищает бит RBS в регистре CANSR..
3	CDO	1 – очищение бита переполнения данных (Data Overrun Status) в регистре CANSR.
4	SRR	1 – Self Reception Request – сообщение, уже записанное в регистры

		CANTFI, CANTID и CANTDA, CANTDB (необязательно), помещается в очередь на передачу. В отличие от бита TR в данном случае приемник является доступным во время передачи и может принимать сообщения.
5	STB1	1 – выбор первого буфера для передачи (Select Tx Buffer1).
6	STB2	1 – выбор второго буфера для передачи (Select Tx Buffer2).
7	STB3	1 – выбор третьего буфера для передачи (Select Tx Buffer3).

Глобальный регистр статуса CANGSR

Регистр доступен только для чтения, но если RM (CANMOD) = 1, поля RXERR и TXERR доступны и для записи.

Биты	Название	Функция
0	RBS	1 – статус буфера приема (Receive Buffer Status) – принятое сообщение доступно в регистрах CANRFS, CANRID и CANRDA CANTDB (если используются). Этот бит очищается командой RRB в регистре CANCMR, если последующие принятые сообщения не доступны.
1	DOS	1 – переполнение данных (Data Overrun Status) – сообщение было потеряно, потому что предыдущее сообщение не было прочитано и соответствующие регистры очень быстро освободились от него; 0 – никакого переполнения данных не произошло с момента последнего выполнения команды CDO (регистр CANCMR) или перезагрузки.
2	TBS	1 – статус буфера передачи (Transmit Buffer Status) – все сообщения переданы для данного контроллера CAN (из всех трех буферов передачи) и можно программно записывать в регистры любого буфера CANTFI, CANTID и CANTDA, CANTDB; 0 – как минимум одно сообщение находится в очереди для передачи и все еще не послано, поэтому не следует записывать в регистры этого занятого буфера передачи. Подробнее в регистре CANSR.
3	TCS	1 – статус выполненной передачи (Transmit Complete Status) – все сообщения были успешно переданы; 0 – как минимум одно сообщение не было успешно отправлено. Подробнее в регистре CANSR.
4	RS	1 – статус приема (Receive Status) – контроллер CAN принимает сообщение.
5	TS	1 – статус передачи (Transmit Status) – контроллер CAN отправляет сообщение. Подробнее в регистре CANSR.
6	ES	1 – статус ошибки (Error Status) – один или оба счетчика ошибок по приему (передаче) достигли максимального значения.
7	BS	1 – статус шины (Bus Status) – выбор третьего буфера для передачи (Select Tx Buffer3).
23:16	RXERR	Текущее значение счетчика ошибок приема (Rx Error Counter).
31:24	TXERR	Текущее значение счетчика ошибок передачи (Tx Error Counter).

Регистр разрешения прерывания CANIER

Биты	Название	Функция
------	----------	---------

0	RIE	Разрешение прерывания от приемника.
1	TIE1	Разрешение прерывания от передатчика (первый буфер передачи).
2	EIE	Разрешение прерывания предупреждения ошибки.
3	DOIE	Разрешение прерывания переполнения данных.
4	WUIE	Разрешение прерывания “пробуждения”.
5	EPIE	Разрешение прерывания “пассивной ошибки”.
6	ALIE	Разрешение прерывания потери арбитража.
7	BEIE	Разрешение прерывания ошибки на шине.
8	IDEI	Разрешение прерывания готовности идентификатора (ID).
9	TIE2	Разрешение прерывания от передатчика (второй буфер передачи).
10	TIE3	Разрешение прерывания от передатчика (третий буфер передачи).

Регистр временных характеристик шины CANBTR

Доступен для записи только при RM (CANMOD) = 1.

Биты	Название	Функция
0:9	BRP	Делитель частоты (Baud Rate Prescaler). Частота шины CAN = частота VPB/(BRP+1).
15:14	SJW	Переход синхронизации (Synchronization Jump Width).
19:16	TSEG1	Сегмент буфера1 фазы (PHASE_SEG1)
22:20	TSEG2	Сегмент буфера2 фазы (PHASE_SEG2)
23	SAM	1 – значение на шины считывается трижды (рекомендуется для шин с низкой и средней скоростью передачи); 0 – значение шины считывается один раз (рекомендуется для высокоскоростных шин).

Регистр статуса CANSR

Регистр содержит три байта статуса, в каждом из которых биты, не касающиеся передачи сообщения, идентичны битам в регистре CANGSR; остальные биты отражают статус каждого из трех буферов передачи (Tx Buffers).

Биты	Название	Функция
0, 8, 16	RBS	Эти биты идентичны биту RSB в регистре CANGSR.
1, 9, 17	DOS	Эти биты идентичны биту DOS в регистре CANGSR.
2, 10, 18	TBS1, TBS2, TBS3	1 – можно программно записывать сообщения в регистры выбранного буфера CANTFI, CANTID и CANTDA, CANTDB; 0 – не следует записывать в регистры этого занятого буфера передачи.
3, 11, 19	TCS1, TCS2, TCS3	1 – запрошенная ранее передача для выбранного буфера передачи была успешно выполнена; 0 – запрошенная ранее передача для выбранного буфера передачи была не выполнена.
4, 12, 20	RS	Эти биты идентичны биту RS в регистре CANGSR.
5, 13, 21	TS1, TS2, TS3	1 – контроллер CAN отправляет сообщение из выбранного буфера передачи.
6, 14, 22	ES	Эти биты идентичны биту ES в регистре CANGSR.
7, 15, 23	BS	Эти биты идентичны биту BS в регистре CANGSR.

Регистр статуса полученного сообщения CANRFS

Регистр определяет характеристики текущего принятого сообщения. Доступен для записи только при RM (CANMOD) = 1.

Биты	Название	Функция
9:0	ID Index	См. в руководстве пользователя по LPC2292.
10	BP	См. в руководстве пользователя по LPC2292.
19:16	DLC	Содержит код длины данных для принятого сообщения: 0000-0111 = 0-7 байт, 1000-1111 = 8 байт
30	RTR	Бит запроса удаленной передачи. Если RTR = 0, то это кадр данных, при этом сами данные можно прочитать из регистров CANRDA и CANRDB (если DLC ≠ 0). Если RTR = 1, то это кадр удаленного запроса данных и DLC в этом случае определяет количество запрашиваемых байтов для посылки.
31	FF	0 – стандартное сообщение (11-битное поле идентификатора); 1 – расширенное сообщение (29-битное поле идентификатора). Влияет на содержимое регистра идентификатора (CANRID).

Регистр данных полученного сообщения CANRDA

Доступен для записи только при RM (CANMOD) = 1 как и регистр CANRID.

Биты	Название	Функция
7:0	Data 1	Если DLC (CANRFS) ≥ 0001, то это поле содержит первый байт данных полученного сообщения.
15:8	Data 2	Если DLC (CANRFS) ≥ 0010, то это поле содержит второй байт данных полученного сообщения.
23:16	Data 3	Если DLC (CANRFS) ≥ 0011, то это поле содержит третий байт данных полученного сообщения.
31:24	Data 4	Если DLC (CANRFS) ≥ 0100, то это поле содержит четвертый байт данных полученного сообщения.

Регистр данных полученного сообщения CANRDB аналогично CANRDA для 5-8 байтов данных сообщения.

Регистр информации о передаваемом сообщении CANTFI

Если бит TBS = 1 в регистре CANSR, то можно записать информацию в этот регистр для определения формата передаваемого сообщения (для выбранного буфера передачи).

Биты	Название	Функция
7:0	PRIO	Если бит TPM (CANMOD) = 1, то выбрать очередность буферов передачи можно при помощи этого поля. Чем меньше значение этого поля, тем приоритетнее буфер передачи.
19:16	DLC	Аналогично полю DLC в регистре CANRFS.
30	RTR	Аналогично полю RTR в регистре CANRFS.
31	FF	Аналогично полю FF в регистре CANRFS.

Регистры CANTID, CANTDA, CANTDB аналогичны регистрам CANRID, CANRDA, CANRDB (нет ограничения на запись).

Литература

1. Учебный стенд SDK-2.0. Инструкция по эксплуатации
2. LPC2119/2129/2194/2292/2294 USER MANUAL. Philips Semiconductors.
3. Can Specification 2.0, Part B.
4. http://www.triton.ru/site/el_can.shtml
5. http://www.itt-ltd.com/reference/ref_can.html