Глава XII

Controller Area Network

Цель работы:

– получение представлений о сети CAN;

– инициализация модуля CAN в микроконтроллерах серии 1986ВЕ9х;

– обмен данными между микроконтроллерами посредством сети CAN.

Оборудование:

– отладочный комплект для микроконтроллера 1986ВЕ92У;

– программатор-отладчик MT-LINK;

– персональный компьютер.

Программное обеспечение:

– операционная система Windows 7 / 8 / 10;

– среда программирования Keil µVision MDK-ARM 5.24;

– драйвер программатора MT-LINK;

– примеры кода программ.

12.1. Общие сведения

CAN (англ. Controller Area Network – местная контроллерная сеть) – стандарт промышленной сети, ориентированный на объединение блоков управления, исполнительных устройств и датчиков в систему.

Стандарт CAN разработан в середине 1980-х в результате совместных работ компаний Robert Bosch GmbH и Intel по проектированию высокоскоростной сети с простым подключением узлов и блоков к автомобильной системе. Впоследствии CAN вошел во все сферы промышленного управления, в том числе автомобильную, железнодорожную и авиационную технику, судостроение, производственную автоматику.

CAN представляет собой высоконадежный протокол связи **равноправных устройств** по **последовательному каналу** в **широковещательном режиме**. Это означает, что каждый отправленный кадр данных получат все узлы сети; невозможно послать сообщение только одному узлу. Поэтому каждый узел должен содержать встроенный конфигурируемый фильтр, блокирующий нежелательные кадры данных [x].

Необходимо отметить, что стандарт CAN определяет только протокол передачи данных в отрыве от физического уровня. Это позволяет организовывать сеть с использованием различных носителей сигнала – радиоканала, оптоволокна, медного провода. На практике, однако, под CAN обычно понимается сеть с топологией типа «шина», организованная на базе дифференциальной пары.

Для абстрагирования от среды передачи, стандарт CAN не использует такие понятия, как логические «0» и «1» при определении сигналов. Вместо этого применяются термины «**доминантный**» и «**рецессивный**», смысл которых состоит в том, что при одновременной передаче двумя источниками доминантного и рецессивного сигналов, принят будет только доминантный.

Стандарт сети требует от физического уровня, фактически, единственного условия: подавление доминантным сигналом рецессивного (но не наоборот). Так, например, в радиоканале отсутствие сигнала соответствует рецессивному состоянию, наличие – доминантному; в оптическом волокне доминантному сигналу соответствует свет, рецессивному – темнота. Однако в электрическом проводе доминантным сигналом обычно является низкий уровень напряжения, так как сигнальная линия подтягивается к земле; высокий уровень напряжения, соответственно, является рецессивным сигналом. В таком случае, если линия находится в рецессивном состоянии, то перевести ее в доминантное может любое устройство сети, передав радиосигнал, световое излучение или подтянув линию к земле.

Все устройства в сети CAN **должны работать на одной скорости**. Стандарт не определяет конкретных значений, но большинство CAN-контроллеров поддерживают диапазон скоростей от 10 Кбит/с до 1 Мбит/с. При этом существуют проектные решения, выходящие далеко за рамки указанного диапазона.

Протокол CAN имеет **встроенный арбитраж** и несколько механизмов **обработки ошибок**, которые требуют, чтобы изменение сигнала при передаче успело распространиться по всей сети к моменту регистрации значения. Это ставит **максимальную длину сети в обратную зависимость от скорости передачи**. Например, в сети длиной 40 метров скорость передачи данных может составлять 1 Мбит/с, а для организации сети длиной 500 метров необходимо снизить скорость до 125 Кбит/с.

Преимуществами сети CAN являются:

– простота реализации, минимальные затраты на использование;

– высокая устойчивость к помехам;

– арбитраж доступа к сети без потерь пропускной способности;

– надежный контроль ошибок передачи и приема;

– широкий диапазон скоростей работы;

– большое распространение технологии.

К недостаткам следует отнести большой объем служебных данных в кадре (по отношению к полезным данным) [x].

12.2. Аппаратная реализация

Очевидно, что аппаратная реализация сети целиком зависит от выбранного физического уровня, поэтому имеет смысл сразу перейти к конкретике. Наиболее распространенной средой для сети CAN является дифференциальная пара стандарта ISO 11898-2.

Для преобразования сигнала, формируемого модулем CAN, в дифференциальный вид (и обратно), на отладочной плате микроконтроллеров серии 1986BE9x установлена микросхема ATA6660 компании Atmel [x] (предусмотрено посадочное место для интерфейсной микросхемы 5559ИН14 компании «Миландр»). Выводы микросхемы подключены к линиям приема и передачи модуля CAN микроконтроллера (PA6, PA7) с одной стороны, и к разъему типа DE-9 – с другой (рисунок 12.1).

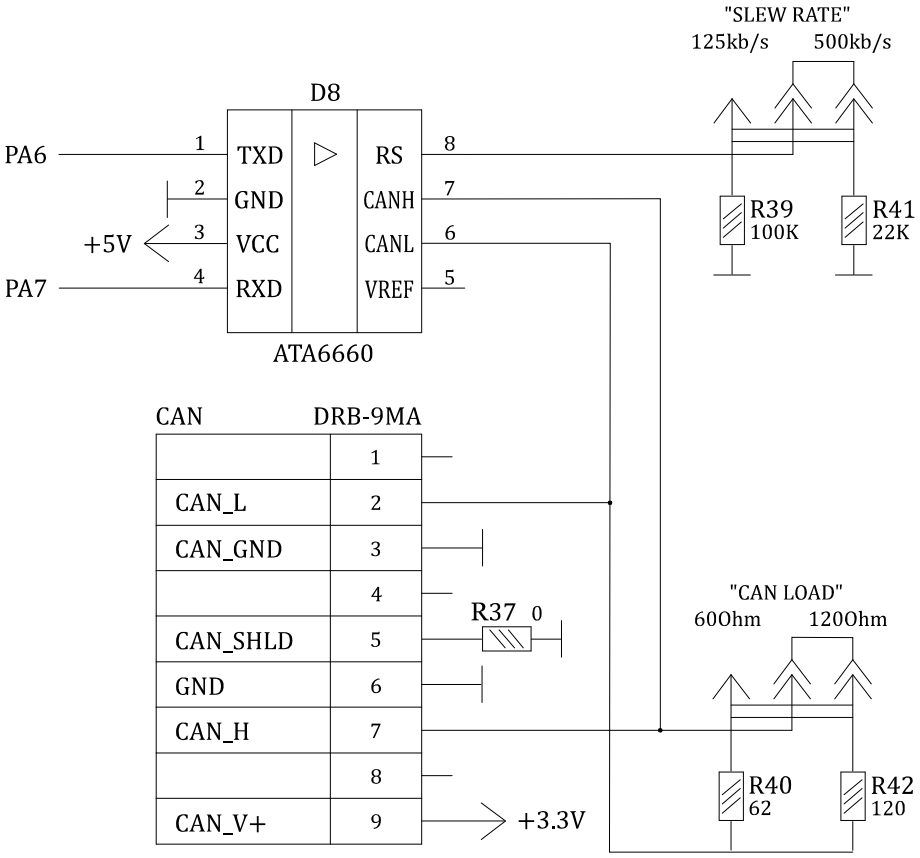

Рисунок 12.1 – Схема подключения микросхемы ATA6660

Дифференциальная передача сигнала ведется по двум электрическим линиям – *CANL* и *CANH*. Изменения напряжений на этих линиях в каждый момент времени равны по модулю, но противоположны по фазе. При этом приемник сигнала реагирует на разность между напряжениями линий. Если разность близка к нулю, то принимается рецессивный сигнал, если достигает ~2 В – доминантный.

Достоинством дифференциальной передачи сигнала является высокая помехоустойчивость, так как электромагнитные помехи оказывают влияние одновременно на оба проводника, и при вычислении разности напряжений будут полностью нивелированы (при условии равенства импеданса источника и приемника).

Шина CAN **обязательно должна терминироваться** вне зависимости от ее скорости. Это достигается путем установки резисторов на концах шины. В частности, на отладочной плате размещен разъем «*CAN LOAD*», определяющий сопротивление терминирующих резисторов в зависимости замкнутых контактов (60 / 120 Ом).

Разъем «*SLEW RATE*» позволяет выбрать крутизну фронтов передаваемых импульсов путем изменения нагрузки на линии RS.

12.3. Протокол передачи данных

12.3.1. Кадры

Информация в сети CAN передается **кадрами** (англ. frame), имеющими определенную структуру. В кадрах отсутствует явный адрес, их можно назвать контентно-адресованными: адресата имплицитно определяет содержимое кадра. Всего существует четыре типа кадров:

– кадр данных;

– кадр удаленного запроса данных;

– кадр ошибки;

– кадр перегрузки.

Кадр данных

Кадр данных (англ. Data Frame) является наиболее используемым типом кадров в сети CAN. Он состоит из семи полей:

1. Начало кадра.
2. Поле арбитража.
3. Поле контроля.
4. Поле данных.
5. Поле CRC.
6. Область подтверждения (ACK).
7. Конец кадра.

Существует два формата кадра данных – **стандартный** и **расширенный**. Их различие заключается в структуре полей арбитража и контроля (рисунки 12.2, 12.3).

1. Начало кадра (англ. Start Of Frame) представляет собой одиночный доминантный бит. Все узлы сети синхронизируются по его фронту. Любой узел может инициировать передачу таким образом, но лишь в том случае, если линия свободна, т.е. находится в рецессивном состоянии.

2. Поле арбитража (англ. Arbitration Field) определяет очередность кадров в том случае, если к шине одновременно обращаются два или более узла.

В стандартном формате кадра поле арбитража состоит из 11-битного идентификатора и бита RTR.

Идентификатор определяет приоритет кадра. При одновременной передаче кадров несколькими узлами арбитраж выигрывает узел, передающий кадр с наименьшим идентификатором (раздел 12.3.2). 7 старших битов идентификатора не должны одновременно иметь рецессивное значение.

Бит RTR (англ. Remote Transmission Request – удаленный запрос данных) определяет тип кадра. Для кадра данных он должен иметь доминантное значение.

В расширенном формате кадра поле арбитража состоит из 29-битного идентификатора, битов SSR, IDE и RTR.

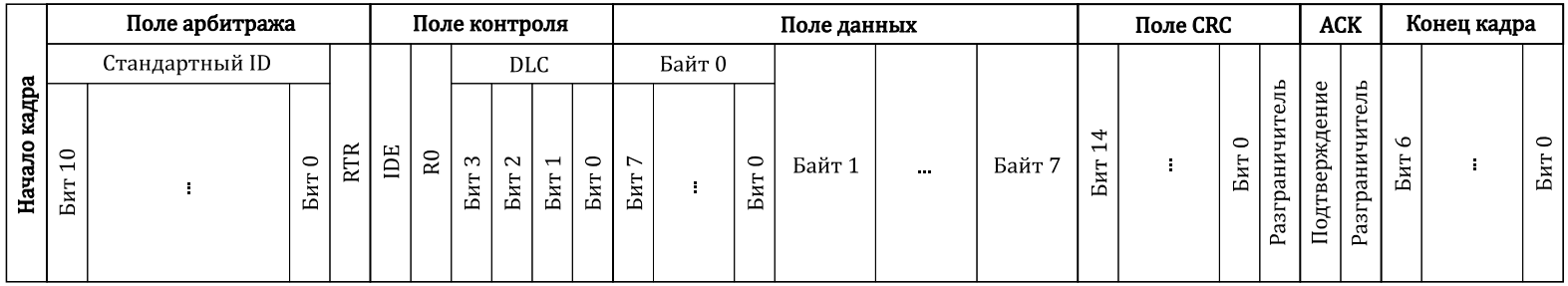

Рисунок 12.2 – Структура стандартного кадра данных

Рисунок 12.3 – Структура расширенного кадра данных

29-битный идентификатор состоит из двух секций:

– 11-битный стандартный идентификатор;

– 18-битный расширенный идентификатор.

Бит SSR (англ. Substitute Remote Request – подмена удаленного запроса) необходим для совместимости стандартного и расширенного форматов. Он всегда должен иметь рецессивное значение.

Бит IDE (англ. Identifier Extension – формат идентификатора) определяет формат кадра данных. Для стандартного формата он должен иметь доминантное значение, для расширенного – рецессивное.

Обратите внимание, что бит IDE находится в поле контроля стандартного кадра, и в поле арбитража – расширенного.

3. **Поле контроля** (англ. Control Field) содержит информацию об объеме передаваемых данных.

В стандартном формате кадра поле контроля состоит из одиночных битов IDE, R0 и четырех битов DLC.

Бит R0 (англ. Reserved – зарезервированный) согласно стандарту протокола должен иметь доминантное значение, однако практически может быть и доминантным, и рецессивным.

Биты DLC (англ. Data Length Code – код длины данных) определяют, какой объем данных содержит кадр (в байтах). Допустимый диапазон значения – от 0 до 8.

В расширенном формате кадра поле контроля состоит из зарезервированных битов R0, R1 и четырех битов DLC.

4. **Поле данных** (англ. Data Field) содержит передаваемые в кадре данные. Оно может иметь длину от 0 до 8 байт в зависимости от выбранного значения DLC. Каждый байт содержит 8 бит, которые передаются, начиная со старшего.

5. **Поле CRC** (англ. Cyclic Redundancy Check – циклическая проверка избыточности) содержит 15-битную контрольную сумму и разграничителя.

Контрольная сумма (CRC-последовательность) включает в себя информацию обо всех описанных выше полях, в том числе «начало кадра», «поле арбитража», «поле контроля» и «поле данных».

Разграничитель представляет собой одиночный рецессивный бит.

6. **Область подтверждения** (англ. Acknowledge Slot) состоит из бита подтверждения (ACK) и разграничителя.

Для получения подтверждения передатчик должен отправить рецессивный бит, а приемник – доминантный. Если подтверждение не получено, то передатчик повторно инициирует передачу кадра.

Наличие бита подтверждения на шине означает только то, что по крайней мере один узел сети получил данные.

7. **Конец кадра** (англ. End Of Frame) состоит из 7 рецессивных битов для отделения кадров друг от друга при плотном трафике.

Кадр удаленного запроса данных

Кадр удаленного запроса данных (англ. Remote Frame) структурно подобен кадру данных с тремя принципиальными отличиями:

– бит RTR в поле арбитража имеет рецессивное значение;

– отсутствует поле данных;

– биты DLC должны соответствовать длине ожидаемого ответного кадра.

Как понятно из названия кадра, его основной задачей является запрос определенных данных. Например, если узел A передает кадр удаленного запроса данных с идентификатором 234, то узел B (если он правильно инициализирован) должен в ответ передать кадр данных с тем же идентификатором 234.

Такая схема взаимодействия узлов сети гипотетически позволяет снизить суммарный трафик. На практике, однако, кадры удаленного запроса данных используются редко.

Кадр ошибки

Кадр ошибки (англ. Error Frame) передается узлом в случае обнаружения сбоя в сети. Он имеет структуру, нарушающую правила формирования кадров в сети CAN, помогая таким образом и другим узлам обнаружить сбой. При регистрации сбоя передатчик автоматически инициирует повторную передачу данных.

Кадр ошибки содержит два поля: «флаг ошибки» и «разграничитель».

1. **Поле флага ошибки** (англ. Error Flag). Существует два вида флагов ошибки: активный и пассивный. Использование того или иного флага определяется внутренним счетчиком ошибок отдельного узла (раздел 12.3.3).

Активный флаг ошибки состоит из 6 доминантных битов, нарушающих правило вставки битов.

Пассивный флаг ошибки состоит из 6 рецессивных битов.

2. **Поле разграничителя ошибки** (англ. Error Delimiter) состоит из 8 рецессивных битов и служит в качестве пространства для флагов ошибки других узлов.

Кадр перегрузки

Кадр перегрузки (англ. Overload Frame) формируется приемником с целью задержки следующего кадра данных. Его формат идентичен формату кадра ошибки.

Кадр перегрузки в настоящее время практически не используется в силу высокой производительности современных контроллеров.

Межкадровое пространство

Кадры данных отделяются друг от друга полем, именуемым межкадровое пространство (англ. Interframe Spacing). Это пространство состоит из двух полей: «перерыв» и «простой шины».

1. **Перерыв** (англ. Intermission) состоит из трех рецессивных битов. Единственное действие, которое можно выполнить во время перерыва – это сообщить о перегрузке, чтобы отложить передачу кадров другими узлами.

2. Простой шины (англ. Bus Idle) может иметь произвольную длину. В этот период шина считается свободной и любой узел может получить контроль над ней.

12.3.2. Арбитраж

Арбитраж сети CAN предотвращает коллизии при одновременной передаче нескольких кадров, а также гарантирует, что наиболее важный кадр будет передан без задержек.

Перед началом каждой передачи узел проверяет состояние шины. Если шина свободна (находится в рецессивном состоянии), то данный узел получает контроль над ней, а другие узлы переходят в режим приема. Если шина занята (активна), то прежде чем начать передачу, узлу необходимо дождаться ее освобождения. Если несколько узлов планировали передачу кадров, то при освобождении шины они начнут передавать их одновременно, синхронизируясь по биту начала кадра. В этом случае на шине начнется процесс арбитража, задача которого состоит в определении того, какой именно кадр будет передан.

Разрешение коллизий кадров в сети CAN осуществляется методом **неразрушающего арбитража**. Суть метода заключается в том, что каждый передающий узел сравнивает передаваемое значение с фактическим значением на шине. Если эти значения равны, передача продолжается. Если они отличаются, то узел, зафиксировавший различие, останавливает передачу и переходит в режим приема.

На рисунке 12.4 показан пример процесса неразрушающего арбитража.

Узел X завершает передачу кадра, при этом узлы A, B и C готовы к передаче с моментов времени T_A, T_B и T_C соответственно. Каждый из узлов A, B и C принял кадр узла X, о чем свидетельствует доминантный бит в интервале подтверждения.

После освобождения шины узлом X остальные узлы синхронизируются в интервале начала кадра и начинают передачу своих идентификаторов. При передаче первых двух битов идентификатора все три узла отправляют доминантные значения, фиксируют на шине идентичные состояния и продолжают передачу. Однако при передаче третьего бита узел A отправляет рецессивное значение, но при этом фиксирует на шине доминантное. Это заставляет узел A освободить шину и вернуться в состояние мониторинга. Узлы B и C продолжают передачу. На седьмом бите идентификатора ситуация повторяется: узел C отправляет рецессивное значение, а узел B – доминантное. Тогда узел C переходит в режим мониторинга, а узел B получает контроль над шиной. Арбитраж, таким образом, выполнен.

По завершении передачи кадра узлом B, узлы A и C повторно начинают передачу, при этом контроль над шиной получает узел C. Следует отметить, что если бы узлу B снова потребовалось передать кадр с тем же идентификатором, то он опять бы получил контроль над шиной. Таким образом, **приоритет в сети CAN имеет кадр с наименьшим идентификатором**.

Преимущество неразрушающего арбитража заключается в том, что он не снижает пропускную способность шины и не задерживает приоритетные кадры.

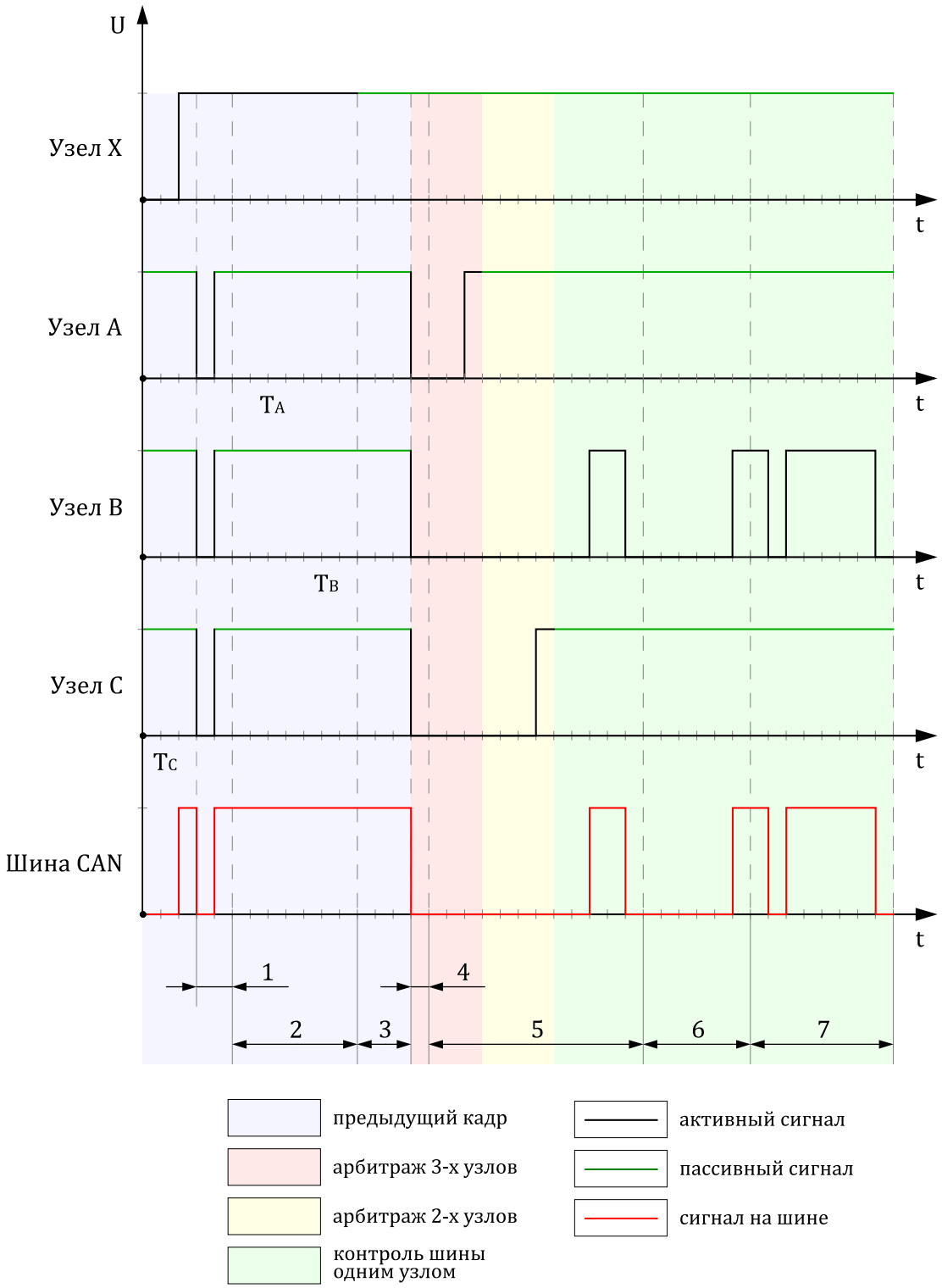

Рисунок 12.4 – Процесс арбитража на шине CAN
(интервалы: 1 – подтверждение; 2 –конца кадра; 3 – межкадровое пространство; 4 – начало кадра; 5 – арбитраж; 6 – контроль; 7 – данные)

12.3.3. Обнаружение ошибок

Протокол CAN имеет пять механизмов обнаружения ошибок в сети:

1. Контроль передаваемых значений.
2. Вставка битов.
3. Проверка формы кадра.
4. Подтверждение приема.
5. Контрольная сумма.

1. Контроль передаваемых значений (англ. Bit Monitoring). При передаче данных в сеть каждый узел сравнивает передаваемое значение с фактическим значением на шине. Если эти значения не совпадают, то узел генерирует **ошибку бита** (англ. Bit Error). Данный механизм контроля отключается на время арбитража и подтверждения.

2. Вставка битов (англ. Bit Stuffing). При последовательной передаче узлом пяти битов с одинаковым значением, должен быть добавлен шестой бит с противоположным значением. Этот бит является служебным и удаляется из кадра принимающими узлами. Если какой-либо узел фиксирует на шине более 5 битов с одинаковым значением, то он генерирует **ошибку вставки** (англ. Stuff Error). Данное правило не распространяется на поле конца кадра.

Механизм вставки битов позволяет предотвратить появление на шине постоянных уровней и обеспечивает достаточное количество переходов, требуемых для повторной синхронизации (раздел 12.3.5).

3. Проверка формы кадра (англ. Frame Check). Некоторые части кадра, такие как разграничители, конец кадра и перерыв в межкадровом пространстве, должны иметь фиксированные значения. Если какой-либо узел обнаружит неверное значение в одном из этих полей, то он генерирует **ошибку формы** (англ. Form Error).

4. Контрольная сумма (англ. Cyclic Redundancy Check). Каждый кадр содержит 15-битную контрольную сумму, зависящую от его содержания. Если узел, принимающий кадр, обнаруживает, что контрольная сумма кадра отличается от посчитанной им, то он генерирует **ошибку контрольной суммы** (англ. CRC Error).

5. Подтверждение приема (англ. Acknowledgement Check). Все узлы шины, корректно получившие кадр, должны отправить доминантное значение в области подтверждения. Если передающий узел не обнаруживает доминантного значения в этой области, то он генерирует **ошибку подтверждения** (англ. Acknowledgement Error).

Механизм ограничения ошибок

Каждый узел сети CAN будет пытаться обнаружить описанные выше ошибки в каждом кадре. Если ошибка обнаружена, то нашедший ее узел немедленно передает на шину кадр ошибки, нарушая таким образом процесс передачи данных и помогая остальным узлам обнаружить ошибку.

Каждый CAN-контроллер содержит два счетчика ошибок: **счетчик ошибок приема** и **счетчик ошибок передачи**. Увеличение значений этих счетчиков происходит при возникновении ошибок при приеме и передаче кадра соответственно. При этом счетчик ошибок передачи увеличивается на 8 единиц за каждую ошибку, а счетчик ошибок приема – на 1 единицу. Уменьшение значений счетчиков происходит при корректном приеме и передаче кадров по тем же правилам [x].

Если значение каждого из счетчиков строго меньше 128, то узел находится в **состоянии активной ошибки** (англ. Error Active), при котором его флаг ошибки состоит из 6 доминантных битов. Если значение любого из счетчиков достигает 128, то узел переходит в **состояние пассивной ошибки** (англ. Error Passive), при котором его флаг ошибки состоит из 6 рецессивных битов и не нарушает передачу данных другими узлами. Если значение любого из счетчиков превысит 255, то узел **прекращает работу с шиной** (состояние «Bus Off»). Для возврата узла в состояние активной ошибки необходимо либо вмешательство процессорного ядра для повторной инициализации CAN-контроллера, либо отсутствие на шине сигнала в течение времени, равного времени передачи 128 x 11 = 1408 битов.

12.3.4. Структура битового интервала

Все узлы сети CAN должны работать на одной скорости передачи кадров.

Протокол CAN использует кодирование без возврата в ноль (англ. Non Return to Zero, NRZ), в котором отсутствует синхронизация. Поскольку все узлы сети имеют собственные генераторы, а время распространения сигнала от узла к узлу различно, каждый CAN-контроллер содержит блок **цифровой фазовой автоподстройки** (англ. Digital Phase-Locked Loop, DPLL). Блок DPLL разбивает битовый интервал на сегменты, которые в свою очередь состоят из **квантов времени** (англ. Time Quanta, TQ). Структурно битовый интервал состоит из четырех сегментов (рисунок 12.5):

– сегмент синхронизации;

– сегмент распространения;

– два фазовых сегмента.

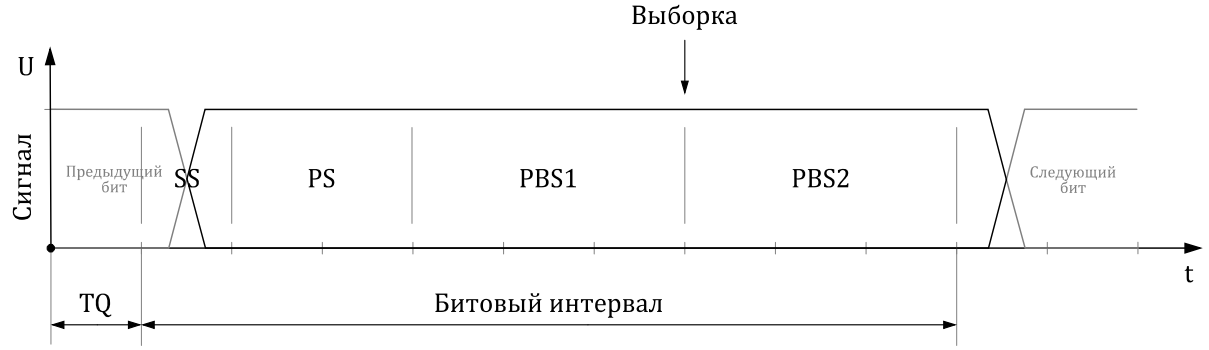

Рисунок 12.5 – Структура битового интервала
(SS – сегмент синхронизации; PS – сегменты распространения;
PBS1, PBS2 – фазовые сегменты)

1. Сегмент синхронизации (англ. Synchronization Segment) предназначен для синхронизации приемника и передатчика. Признаком успешной синхронизации является изменение уровня сигнала в данном сегменте. Длительность сегмента составляет 1 TQ.

2. Сегмент распространения (англ. Propagation Segment) используется для компенсации физических задержек распространения сигнала в сети и внутренних задержек в узлах. Длительность сегмента может составлять от 1 до 8 TQ.

3. Фазовые сегменты (англ. Phase Buffer Segments) определяют положение выборки полученного сигнала. Точка выборки располагается между сегментами, т.е. в конце первого сегмента или начале второго. Длительность каждого сегмента может составлять от 1 до 8 TQ.

В условиях медленной синхронизации возможна работа CAN-контроллера в режиме трехкратной выборки, при котором значение каждого бита считывается три раза с периодом TQ / 2. В качестве истинного значения бита принимается логический уровень, зафиксированный по крайней мере дважды.

Длительность битового интервала равна сумме длительностей всех сегментов:

$$T_{\text{бит}} = \text{TQ} \times (\text{SS} + \text{PS} + \text{PBS1} + \text{PBS2}). \quad (12.1)$$

Согласно стандарту, битовый интервал должен иметь длительность от 8 TQ до 25 TQ.

Максимальная скорость передачи S в сети CAN составляет 1 Мбит/с и определяется величиной битового интервала:

$$S = \frac{1}{T_{\text{бит}}}. \quad (12.2)$$

12.3.5. Синхронизация

Существует два метода синхронизации: жесткая синхронизация и повторная синхронизация.

1. Жесткая синхронизация (англ. Hard Synchronization) выполняется один раз в течение передачи кадра при первом переходе сигнала из рецессивного состояния в доминантное, т.е. в поле начала кадра. Жесткая синхронизация гарантирует, что на начальном этапе фронт сигнала будет расположен в сегменте синхронизации.

2. Повторная синхронизация (англ. Soft Synchronization) производится при отклонении фронта сигнала от сегмента синхронизации. Восстановление битовых интервалов осуществляется за счет увеличения длительности первого фазового сегмента или уменьшения второго. Допустимое значение изменения фазовых сегментов задается программно и находится в диапазоне от 1 TQ до 4 TQ. Синхронизация выполняется только при переходе сигнала из рецессивного состояния в доминантное.

Механизм вставки битов гарантирует достаточное количество изменений сигнала для своевременного восстановления синхронизации.

*
**

На рисунке 12.6 показан пример процесса передачи кадра данных стандартного формата по дифференциальной паре с учетом механизма вставки битов.

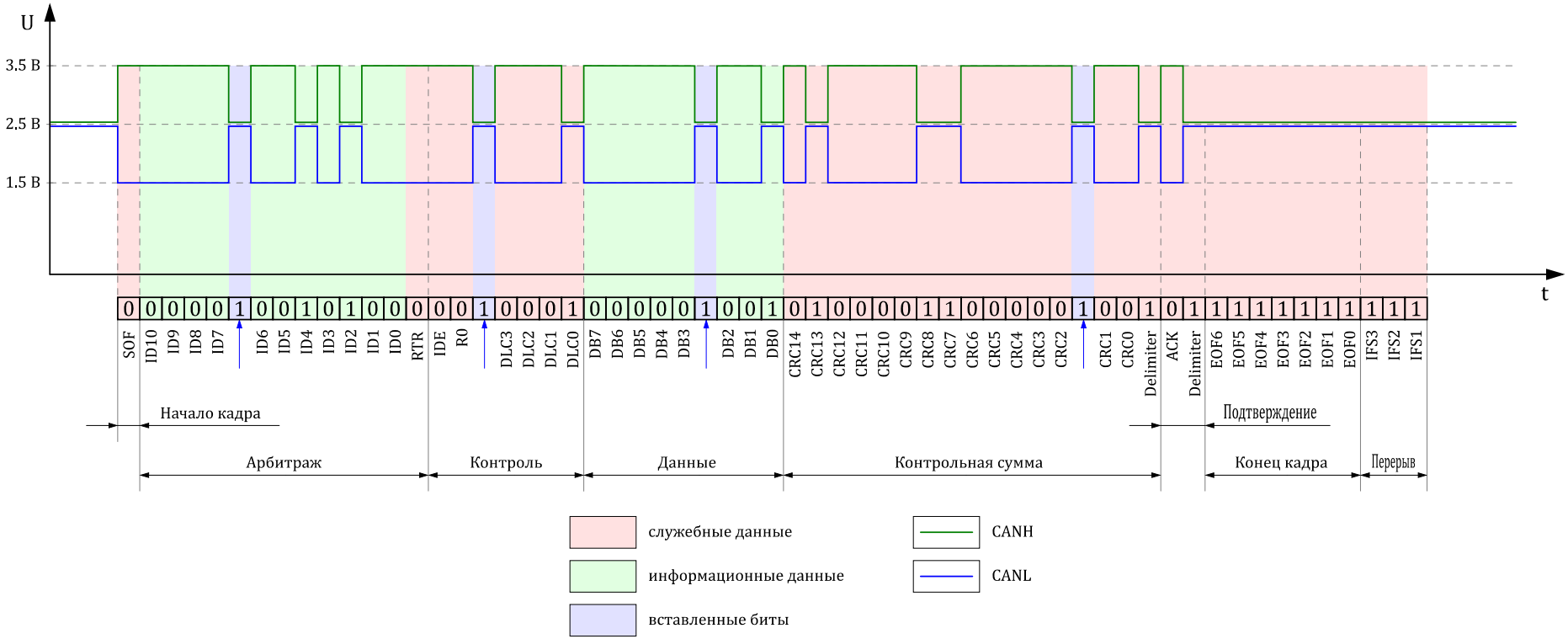

Рисунок 12.6 – Процесс передачи кадра данных стандартного формата по дифференциальной паре (идентификатор – 00000010100; один байт данных – 00000001)

12.4. Программирование контроллера CAN в микроконтроллерах серии 1986BE9x

12.4.1. Особенности контроллера

В микроконтроллерах серии 1986BE9x реализовано два независимых полнофункциональных CAN-контроллера, отвечающих требованиям к активным и пассивным устройствам стандартов CAN 2.0A и CAN 2.0B.

В каждом контроллере реализованы 32 буфера для приема/передачи данных, 32 фильтра входных кадров и система маскирования прерываний.

В работе будет использован модуль CAN1, т.к. его выводы подведены к микросхеме ATA6660 и разъему «CAN» отладочной платы.

12.4.2. Инициализация

Инициализация модуля CAN состоит из следующих шагов:

1. Включение тактирования.
2. Деинициализация.
3. Конфигурация порта ввода-вывода.
4. Конфигурация CAN-контроллера.
5. Конфигурация буферов приема и передачи.
6. Настройка аппаратных прерываний.

1. Включение тактирования контроллера производится тривиально:

```
// Включение тактирования контроллера CAN1
MDR_RST_CLK->PER_CLOCK |= (1 << 0);
MDR_RST_CLK->CAN_CLOCK |= (1 << 24);
```

2. Деинициализация состоит из сброса всех регистров CAN-контроллера, в том числе регистров управления буферами и регистров фильтрации входных кадров:

```
// Деинициализация контроллера CAN
void CAN_Reset(MDR_CAN_TypeDef *MDR_CANx)
{
  uint32_t i;

  MDR_CANx->CONTROL = 0;
  MDR_CANx->STATUS  = 0;
  MDR_CANx->BITTMNG = 0;
  MDR_CANx->INT_EN  = 0;
  MDR_CANx->OVER    = 0;
  MDR_CANx->INT_RX  = 0;
  MDR_CANx->INT_TX  = 0;

  // Сброс регистров управления буферами
  for (i = 0; i < 32; i++)
    MDR_CANx->BUF_CON[i] = 0;

  // Сброс фильтров
  for (i = 0; i < 32; i++)
  {
    MDR_CANx->CAN_BUF_FILTER[i].FILTER = 0;
    MDR_CANx->CAN_BUF_FILTER[i].MASK   = 0;
  }
}
```

3. Конфигурация порта ввода-вывода. CAN-контроллер использует две линии для обмена данными: одну для приема (*CAN_RX*) и одну для передачи (*CAN_TX*). Для корректной работы контроллера линии PA6 и PA7 должны быть сконфигурированы в качестве цифровых линий входа и выхода в соответствии с таблицей П.1:

```
// Конфигурация порта для работы контроллера CAN1
void CAN_PortCfg(void)
{
  // Включение тактирования порта A
  MDR_RST_CLK->PERCLOCK |= (1 << 21);

  // Конфигурация линий PA6 (TX) и PA7 (RX)
  MDR_PORTA->OE     |=  ((1 << 6));               // Направление линии PA6 (выход)
  MDR_PORTA->OE     &= ~((1 << 7));               // Направление линии PA7 (вход)
  MDR_PORTA->PULL   &= ~((1 << 6)  | (1 << 7));
  MDR_PORTA->PULL   &= ~((1 << 22) | (1 << 23));
  MDR_PORTA->ANALOG |=  ((1 << 6)  | (1 << 7));
  MDR_PORTA->FUNC   &= ~((3 << 12) | (3 << 14));
  MDR_PORTA->FUNC   |=  ((2 << 12) | (2 << 14));  // Функция линий (альтер.)
  MDR_PORTA->PD     &= ~((1 << 6)  | (1 << 7));
  MDR_PORTA->PD     &= ~((1 << 22) | (1 << 23));
  MDR_PORTA->PWR    |=  ((3 << 12) | (3 << 14));  // Длительность фронтов (низкая)
  MDR_PORTA->GFEN   &= ~((1 << 6)  | (1 << 7));
}
```

4. Конфигурация CAN-контроллера производится в двух регистрах – `CONTROL` и `BITTMNG`, описание которых приведено в таблицах 12.1 и 12.2 соответственно.

Таблица 12.1 – Описание регистра `MDR_CANx->CONTROL`

№ битов	Дескрипция
31…5	–
4	Прием собственных кадров: 0 – запрещен; 1 – разрешен.
3	Подтверждение приема собственных кадров: 0 – запрещено; 1 – разрешено.
2	Режим самотестирования: 0 – нормальный режим; 1 – режим самотестирования.
1	Режим «только чтение»: 0 – нормальный режим; 1 – режим «только чтение».
0	Запуск CAN-контроллера: 0 – отключен; 1 – включен.

Таблица 12.2 – Описание регистра MDR_CANx->BITTMNG

№ битов	Дескрипция
31…28	–
27	Режим выборки: 0 – однократная; 1 – трехкратная с мажоритарным контролем.
26…25	Допустимый интервал подстройки: 00 – 1 TQ; 01 – 2 TQ; 10 – 3 TQ; 11 – 4 TQ.
24…22	Длительность второго фазового сегмента: 000 – 1 TQ; 001 – 2 TQ; … 111 – 8 TQ.
21…19	Длительность первого фазового сегмента: 000 – 1 TQ; 001 – 2 TQ; … 111 – 8 TQ.
18…16	Длительность сегмента распространения: 000 – 1 TQ; 001 – 2 TQ; … 111 – 8 TQ.
15…0	Делитель тактовой частоты CAN-контроллера: $CAN\_CLK = \frac{HCLK}{DIV + 1}$.

В регистре CONTROL задается режим работы контроллера CAN:

– **режим «только чтение»** отключает от шины линию передачи, не позволяя контроллеру отправлять кадры; в пределах контроллера, однако, все сигнал проходят в штатном режиме;

– **режим самотестирования** соединяет линию передачи контроллера с линией приема, создавая замкнутую петлю;

– **прием собственных кадров** может иметь место ввиду того, что шина CAN является широковещательной; при этом отправление подтверждения приема таких кадров настраивается отдельно.

Регистр BITTMNG содержит конфигурацию битового интервала.

Значение кванта времени задается контроллером по следующей формуле:

$$\mathrm{TQ} = \frac{DIV + 1}{HCLK}. \tag{12.3}$$

Поскольку длительность битового интервала определяется суммой квантов времени всех сегментов (формула 12.1), а скорость передачи обратно пропорциональна значению битового интервала (формула 12.2), то согласно формуле 12.3 в общем виде можно вывести такую зависимость:

$$S = \frac{HCLK}{N \times (DIV + 1)}, \tag{12.4}$$

где N – сумма всех сегментов в квантах времени.

Допустимый интервал подстройки задает максимальное изменение длительности фазовых сегментов для восстановления синхронизации (раздел 12.3.5) В типовых приложениях, как правило, достаточно 1-2 TQ.

Для примера рассмотрим конфигурацию контроллера CAN в режиме самотестирования для обеспечения скорости передачи данных 0.5 Мбит/с:

```
// Конфигурация контроллера CAN
MDR_CAN1->CONTROL = (1 << 0)   // Работа контроллера (включен)
                  | (0 << 1)   // Режим "только чтение" (отключен)
                  | (1 << 2)   // Режим самотестирования (включен)
                  | (1 << 3)   // Подтв. приема собственных кадров (включено)
                  | (1 << 4);  // Прием собственных кадров (включен)

// Конфигурация битового интервала
MDR_CAN1->BITTMNG = (15 << 0)   // Делитель тактовой частоты (16 – 1)
                  | (1 << 16)   // Длительность сегмента распространения (2 TQ)
                  | (3 << 19)   // Длительность первого фазового сегмента (4 TQ)
                  | (2 << 22)   // Длительность второго фазового сегмента (3 TQ)
                  | (1 << 25)   // Допустимый интервал подстройки сегментов (2 TQ)
                  | (0 << 27);  // Режим выборки (однократная)
```

Сумма всех сегментов в приведенном фрагменте составляет 10 TQ (сегмент синхронизации не задан явно – его значение всегда равно 1 TQ). Поэтому для обеспечения скорости передачи данных 0.5 Мбит/с по формуле 12.4 требуется задать делитель равным:

$$DIV = \frac{HCLK}{S \times N} - 1 = \frac{80 \times 10^6}{0.5 \times 10^6 \times 10} - 1 = 15.$$

5. Конфигурация буферов приема и передачи. Для правильной работы контроллера CAN целевые буферы проекта также должны быть настроены соответствующим образом. Настройка буферов осуществляется в регистрах `BUF_CON[x]`, где `x` – номер буфера (0…31). Описание регистров приведено в таблице 12.3.

Таблица 12.3 – Описание регистра MDR_CANx->BUF_CON[x]

№ битов	Дескрипция
31…8	–
7	Флаг перезаписи принятого кадра: 0 – перезапись не зафиксирована; 1 – кадр записан поверх предыдущего.
6	Флаг наличия принятого кадра в буфере: 0 – буфер пуст; 1 – буфер содержит принятый кадр.
5	Запрос на передачу кадра: 0 – запрос отсутствует (либо кадр уже отправлен); 1 – запрос на передачу кадра.
4	Приоритет буфера: 0 – высокий; 1 – обычный.
3	Ответ на удаленный запрос (RTR): 0 – запрещен; 1 – разрешен.
2	Перезапись принятого кадра: 0 – запрещена; 1 – разрешена.
1	Режим работы буфера: 0 – передача; 1 – прием.
0	Работа буфера: 0 – отключен; 1 – включен.

Приоритет буфера определяет порядок отправления кадров по шине CAN при одновременном запросе на передачу с нескольких буферов. В этой ситуации в первую очередь будет обработан буфер с высоким приоритетом. При равном приоритете очередность обработки определяется порядковыми номерами буферов: буфер с меньшим номер имеет больший приоритет.

Перезапись кадров используется в случае, когда для приема очередного кадра отсутствует свободный буфер; тогда кадр будет сохранен в иной буфер, для которого разрешена перезапись. Следует с осторожностью использовать данный, т.к. процесс перезаписи может попасть на период считывания кадра процессорным ядром, и в таком случае первая часть полученных данных будет принадлежать одному кадру, а вторая – другому, т.е. практически два кадра будут утрачены, а в распоряжении ядра окажется некорректная информация. Чтобы избежать дальнейшей обработки некорректных данных рекомендуется перед считыванием кадра сбросить флаг перезаписи (бит 7 регистра BUF_CON[x]), а после считывания проверить его. Если при проверке он все еще сброшен, то считанный кадр можно считать корректным; если установлен, то в процессе чтения была произведена перезапись данных и этот кадр следует игнорировать.

Так, например, для конфигурации нулевого буфера на прием данных, а первого – на передачу, нужно написать:

```
// Конфигурация буфера приема
MDR_CAN1->BUF_CON[0] = (1 << 0)   // Работа буфера (включен)
                     | (1 << 1)   // Режим работы буфера (прием)
                     | (0 << 2)   // Разрешение перезаписи кадра (запрещено)
                     | (0 << 3)   // Разрешение ответа на RTR (не используется)
                     | (1 << 4);  // Приоритет буфера (обычный)

// Конфигурация буфера передачи
MDR_CAN1->BUF_CON[1] = (1 << 0)   // Работа буфера (включен)
                     | (0 << 1)   // Режим работы буфера (передача)
                     | (0 << 2)   // Разрешение перезаписи кадра (не используется)
                     | (0 << 3)   // Разрешение ответа на RTR (запрещен)
                     | (1 << 4);  // Приоритет буфера (обычный)
```

6. Настройка аппаратных прерываний, как обычно, состоит из разрешения отправления запросов от контроллера CAN, конфигурации контроллера NVIC и описания обработчика прерываний.

Запросы на обработку прерываний от контроллера CAN регулируется регистром `INT_EN` (таблица 12.4).

Таблица 12.4 – Описание регистра `MDR_CANx->INT_EN`

№ битов	Дескрипция
31…5	–
4	Прерывание при превышении допустимого значения счетчика ошибок: 0 – запрещено; 1 – разрешено.
3	Прерывание при возникновении ошибки на шине: 0 – запрещено; 1 – разрешено.
2	Прерывание при передаче кадра: 0 – запрещено; 1 – разрешено.
1	Прерывание при приеме кадра: 0 – запрещено; 1 – разрешено.
0	Общий флаг прерываний контроллера: 0 – прерывания запрещены; 1 – прерывания разрешены.

Если в проекте предполагается использование какого-либо прерывания, то **общий флаг прерываний** (бит 0 регистра `INT_EN`) также должен быть установлен.

При работе с **прерываниями при приеме** или **передаче** также нужно задать буферы, для которых будут работать прерывания с использованием регистров `INT_RX` и `INT_TX` соответственно.

Прерывание при возникновении ошибки может быть подробно обработано с помощью регистра STATUS (стр. 260 Спецификации). Данный регистр хранит всю информацию механизма обработки ошибок (раздел 12.3.3).

Допустимое значение счетчика ошибок для формирования прерывания (в пределах 255) задается в регистре OVER.

Таким образом, чтобы организовать прерывание, к примеру, при приеме кадра в нулевой буфер можно написать:

```
// Настройка запросов на обработку прерываний от контроллера CAN
MDR_CAN1->INT_EN = (1 << 0)   // Общий флаг прерываний
                 | (1 << 1);  // Прерывание при приеме кадра

// Настройка запросов на обработку прерываний при приеме кадра в определенный буфер
MDR_CAN1->INT_RX = (1 << 0);

// Назначение приоритета прерывания от контроллера CAN
NVIC_SetPriority(CAN1_IRQn, 1);

// Разрешение обработки прерывания от контроллера CAN
NVIC_EnableIRQ(CAN1_IRQn);

...

// Обработчик прерываний от контроллера CAN1
void CAN1_IRQHandler(void)
{
  uint32_t data;

  // Чтение данные из буфера
  data = MDR_CAN1->CAN_BUF[0].DATAL;

  // Передача данных
  osMessageQueuePut(MsgQueId_CAN, &data, NULL, NULL);

  // Сброс флага прерывания при приеме кадра
  MDR_CAN1->BUF_CON[0] &= ~(1 << 6);
}
```

В описанном выше обработчике выполняется извлечение минимального объема данных из буфера. Получение полной информационной составляющей кадра будет рассмотрено в разделе 12.4.4.

12.4.3. Передача данных

Для передачи кадра данных его необходимо предварительно сформировать. Для каждого буфера предусмотрена структура данных CAN_BUF[x] (где x – номер буфера), поля которой соответствуют формату кадра. Заполнение такой структуры может выглядеть так:

```
// Формирование кадра
MDR_CAN1->CAN_BUF[1].ID    = 234;         // Идентификатор кадра
MDR_CAN1->CAN_BUF[1].DLC   = (1 << 0)     // Биты DLC (1 байт)
                           | (0 << 8)     // Бит RTR (нет запроса)
                           | (1 << 9)     // Бит R1 (всегда должен быть равен 1)
                           | (0 << 10)    // Бит R0 (всегда должен быть равен 0)
                           | (1 << 11)    // Бит SSR (всегда должен быть равен 1)
                           | (1 << 12);   // Бит IDE (расширенный формат)
```

```
MDR_CAN1->CAN_BUF[1].DATAL = 0x000000AA;  // Байты данных 1...4
MDR_CAN1->CAN_BUF[1].DATAH = 0x00000000;  // Байты данных 5...8
```

Параметр DLC определяет количество байт данных в кадре вне зависимости от значений DATAL и DATAH. Это означает, что если значение DATAL будет составлять, например, 0xFFFFFFFF, но DLC при этом по-прежнему будет соответствовать одному байту, то отправлен будет лишь один байт 0xFF.

Передача кадра инициируется путем запроса (бит 5 регистра BUF_CON[x]):

```
// Запрос на передачу сформированного кадра
MDR_CAN1->BUF_CON[1] |= (1 << 5);
```

12.4.4. Прием данных

Принятые кадры располагаются в регистрах CAN_BUF[x] аналогичным уже рассмотренному образом. Для удобства работы с данными кадрами рекомендуется создать специальную структуру и функцию для ее заполнения:

```
// Структура кадра
typedef struct
{
  uint32_t ID;        // Идентификатор
  uint8_t  IDE;       // Формат кадра
  uint8_t  DLC;       // Код длины данных
  uint32_t DATAL;     // Байты данных 1...4
  uint32_t DATAH;     // Байты данных 5...8
} CAN_FrameTypeDef;

// Извлечение принятого кадра из буфера
CAN_FrameTypeDef CAN_ExtractFrame(uint8_t buffer_number)
{
  CAN_FrameTypeDef frame;

  frame.ID    = MDR_CAN1->CAN_BUF[buffer_number].ID;
  frame.IDE   = MDR_CAN1->CAN_BUF[buffer_number].DLC >> 12 & 0x1;
  frame.DLC   = MDR_CAN1->CAN_BUF[buffer_number].DLC & 0xF;
  frame.DATAL = MDR_CAN1->CAN_BUF[buffer_number].DATAL;
  frame.DATAH = MDR_CAN1->CAN_BUF[buffer_number].DATAH;

  return frame;
}
```

Такой подход позволит с легкостью извлекать требуемые данные из кадра, а также организовать передачу этих данных между потоками.

12.4.5. Фильтрация принимаемых кадров

Для уменьшения затрат процессорного времени на обработку принимаемых кадров, в CAN-контроллере реализована специальная система фильтрации. Данная система позволяет задать для каждого буфера значение маски CAN_BUF_FILTER[x].MASK и фильтра CAN_BUF_FILTER[x].FILTER таким образом, что в данный буфер будут приниматься только кадры, удовлетворяющие условию:

```
ID & CAN_BUF_FILTER[x].MASK == CAN_BUF_FILTER[x].FILTER
```

Если принимаемый кадр не может быть помещен ни в один из буферов, то он игнорируется. Если же кадр может быть принят более чем одним буфером, то он помещается в буфер с наименьшим порядковым номером.

Так, например, если в нулевой буфер должны попадать лишь кадры с идентификатором 234 (0xEA), то для маски и фильтра этого буфера следует указать значения:

```
MDR_CAN1->CAN_BUF_FILTER[0].MASK = 0xFFFFFFFF;
MDR_CAN1->CAN_BUF_FILTER[0].FILTER = 0x000000EA;
```

А если в первый буфер требуется принимать только кадры, идентификатор которых меньше 255 (0xFF), то нужно указать значения:

```
MDR_CAN1->CAN_BUF_FILTER[1].MASK = 0xFFFFFF00;
MDR_CAN1->CAN_BUF_FILTER[1].FILTER = 0x00000000;
```

Следует отметить, что по умолчанию после включения питания (или сброса) микроконтроллера регистры масок и фильтров могут иметь произвольные значения, поэтому перед началом работы необходима их деинициализация. Такая процедура уже описана нами в разделе 12.4.2, пункт 2:

```
// Сброс фильтров
for (i = 0; i < 32; i++)
{
  MDR_CANx->CAN_BUF_FILTER[i].FILTER = 0;
  MDR_CANx->CAN_BUF_FILTER[i].MASK   = 0;
}
```

При этом буферы будут принимать все кадры без фильтрации.

Описание программного проекта

В проекте **Sample 12.1** производится инициализация контроллера CAN в режиме самотестирования. При нажатии на кнопку «SELECT» выполняется передача и прием кадра. Содержание принятого кадра отображается на дисплей. Идентификатор каждого последующего кадра на единицу больше предыдущего.

Задачи для самостоятельной работы

1. Задание один.
2. Задание два.
3. Задание три.

Контрольные вопросы

1. На каком физическом уровне может быть организована шина CAN?
2. Что такое дифференциальный сигнал?
3. Опишите структуру кадра данных, используемую стандартом CAN.
4. Что такое неразрушающий арбитраж?
5. Какие механизмы обнаружения ошибок реализованы в стандарте CAN?
6. Из каких сегментов состоит битовый интервал в стандарте CAN?
7. Как производится синхронизация при передаче информации?
8. Как задать определенную скорость передачи данных для контроллера CAN?
9. Как настроить фильтрацию входящих кадров таким образом, чтобы в буфер попадали лишь кадры, старший бит идентификатора которых равен единице?