**CAN.**

**Введение.**

Базовая расширенная CAN периферия, именуемая как bxCAN, является средством взаимодействия (интерфейсом) с CAN сетью. Это поддерживает CAN протоколы версии 2.0A и B. Модуль был разработан для обработки большого числа входящих сообщений эффективно, с минимальной загрузкой CPU. Он также удовлетворяет требованиям приоритета для передаваемых сообщений.

Для критичных к сохранности (безопасности) приложений, CAN контроллер предоставляет все аппаратные функции для поддержки опции CAN Time Triggered Communication.

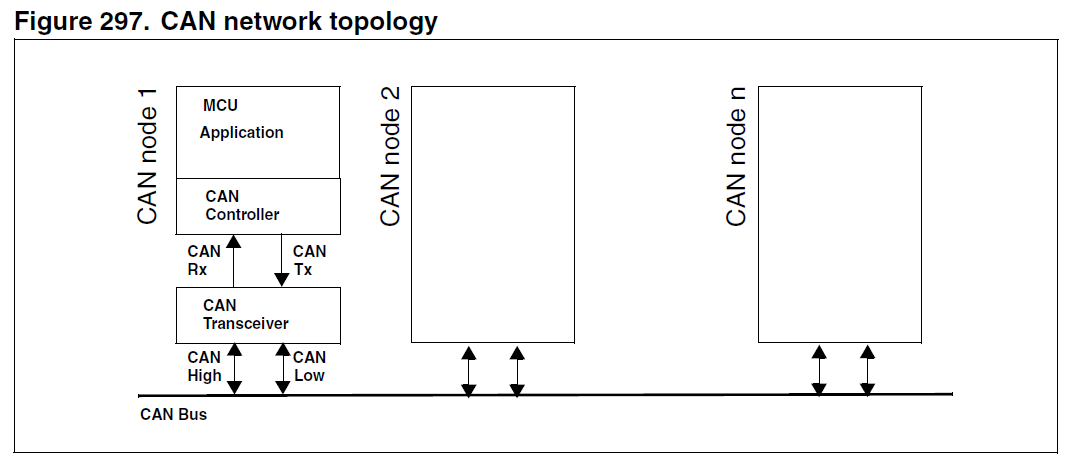
**Общее описание.**

В современных CAN приложениях, число узлов в сети постоянно возрастает и часто несколько сетей соединены вместе через шлюзы. Обычно число сообщений в системе (которые перехватываются каждым узлом) значительно выше. В дополнение к сообщениям приложения, были введены управление сетью, диагностические сообщения.

Улучшенный механизм фильтрации требуется для обработки каждого типа сообщения. Кроме того, задачи приложения требуют больше процессорного времени, поэтому ограничения в реальном времени, вызванные приемом сообщений должны быть уменьшены.

Приемное FIFO позволяет процессору отвлекаться на задачи приложения в течении длительного периода времени без потерь сообщений.

Стандарт HLP (протокол верхнего уровня), основанный на стандартных CAN драйверах, требует эффективного CAN контроллера.

**Активное ядро CAN 2.0B.**

Модуль bxCAN перехватывает передачу и прием CAN сообщений полностью автономно. Стандартные идентификаторы (11 бит) и расширенные идентификаторы (29 бит) полностью поддерживаются аппаратно.

**Регистры управления, статуса и конфигурации.**

Приложение использует эти регистры для:

1. Настройки параметров CAN, например скорости.
2. Запрос передач.
3. Обработка приема.
4. Управление прерываниями.
5. Получение диагностической информации.

**Почтовые ящики Tx.**

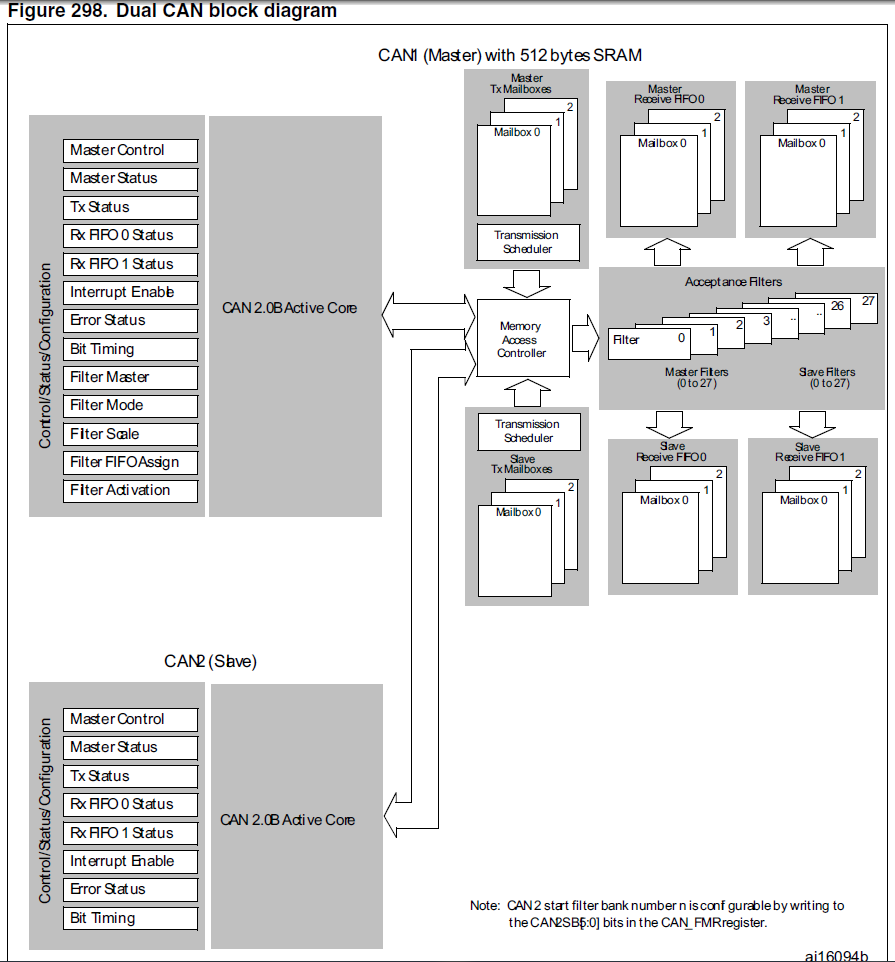
Три передающих почтовых ящика предоставлены программе для установки сообщений. Планировщик передач решает, какой почтовый ящик должен быть отправлен в первую очередь.

**Приемные фильтры.**

bxCAN предоставляет 28 масштабируемых/настраиваемых банков фильтров идентификаторов входящих сообщений, отбирающих нужные программе сообщения и отбрасывающие остальные. В других устройствах есть 14 масшабируемых/настраиваемых банков фильтров идентификаторов.

**Приемное FIFO.**

Два приемных FIFO используется аппаратно для сохранения входящих сообщений. В каждом FIFO может быть сохранено 3 полных сообщения. FIFO полностью управляются аппаратными средствами.



**Режимы работы bxCAN.**

bxCAN имеет три основных режимов работы: инициализация, нормальная работа, спящий режим. После аппаратного сброса, bxCAN находится в спящем режиме для уменьшения энергопотребления и внутренняя подтяжка по питанию активна на CANTX. Программа запрашивает вход bxCAN в режим инициализации, или спящий режим установкой INRQ или SLEEP битов в регистре CAN\_MCR. Как только модуль вошел в режим, bxCAN подтверждает это установкой INAK или SLAK битов в регистре CAN\_MSR и внутренняя подтяжка по питанию отключается. Когда не установлены ни INAK, ни SLAK, bxCAN находится в нормальном режиме. Перед входом в нормальный режим, bxCAN всегда должен быть синхронизирован с CAN шиной. Чтобы засинхронизироваться, bxCAN ожидает, пока CAN шина не будет в idle состоянии, это предполагает 11 последующих recessive битов, отслеживаемых на выводе CANRX.

**Режим инициализации.**

Программная инициализация может быть выполнена, когда модуль находится в режиме инициализации. Для входа в этот режим программа устанавливает INRQ бит в регистре CAN\_MCR и ждет до тех пор, пока аппаратура не подтвердит запрос установкой бита INAK в регистре CAN\_MSR.

Чтобы покинуть режим инициализации, программа очищает INRQ бит. bxCAN должен выйти из режима инициализации, сразу, как только INAK бит будет очищен аппаратурой.

Пока модуль находится в режиме инициализации, все передачи сообщений в и из CAN шины останавливаются и состояние вывода CANTX становится recessive (высокий).

Вхождение в режим инициализации не изменяет каких-либо регистров конфигурации.

Для инициализации CAN контроллера, программа должна установить Bit Timing (CAN\_BTR) и регистры опций CAN (CAN\_MCR).

Для инициализации регистров, связанных с банками CAN фильтров (режим, масштаб, FIFO назначение, активация и значения фильтра), программа должна установить FINIT бит (CAN\_FMR). Инициализация фильтра также может быть выполнена вне режима инициализации.

Замечание. Когда FINIT=1, CAN прием деактивируется. Значения фильтров также могут быть изменены, путем деактивирования соответствующих битов активации фильтра (в регистре CAN\_FA1R). Если банк фильтра не используется, рекомендуется оставить его в неактивном состоянии (оставить соответственный бит FACT очищенным).

**Нормальный режим.**

Как только инициализация завершилась, программа должна запросить аппаратуру войти в нормальный режим, для возможности синхронизации с CAN шиной и старта приема и передачи.

Запрос на вход в нормальный режим работы формируется по сбросу INRQ бита в CAN\_MCR регистре. bxCAN входит в нормальный режим и готов принять участие в шинной деятельности, когда он синхронизируется с передачей данных на CAN шине. Это достигается ожиданием возникновения последовательности из 11 recessive битов (Idle состояние). Переключение в нормальный режим подтверждается аппаратно очисткой INAK бита в регистре CAN\_MSR.

Инициализация значений фильтра не зависит от того, находится ли модуль в режиме инициализации, но она должна быть выполнена, пока фильтр неактивен (соответствующий бит FACTx очищен). Масштаб фильтра и режим конфигурации должен быть настроен перед входом в нормальный режим.

**Спящий режим (низкое энергопотребление).**

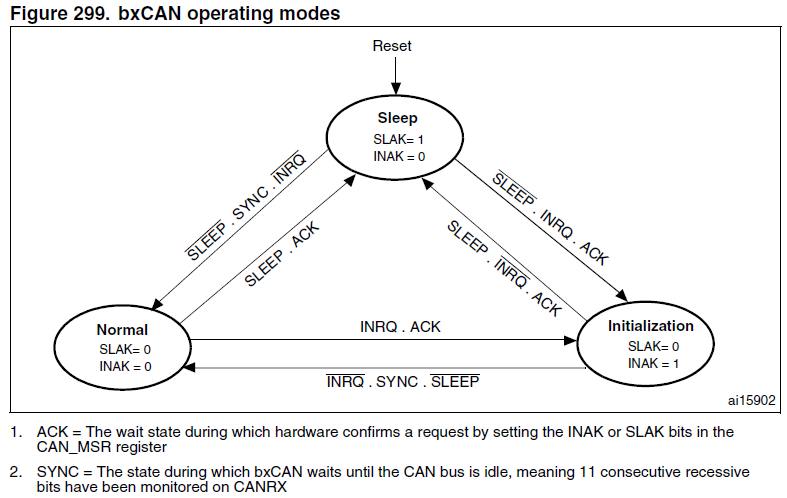
Для уменьшения энергопотребления, bxCAN имеет энергосберегающий режим, называемый Спящим режимом. Модуль входит в этот режим, когда программа затребует установку SLEEP бита в регистре CAN\_MCR. В этом режиме тактирование bxCAN останавливается, однако программа может все еще получить доступ к почтовым ящикам bxCAN.

Если программа требует входа в режим инициализации установкой бита INRQ, пока bxCAN модуль находится в спящем режиме, она должна также очистить SLEEP бит. bxCAN может быть разбужен (выведен из спящего режима) либо программной очисткой SLEEP бита или по определению активности CAN шины.

При возникновении активности на CAN шине, аппаратура автоматически выполняет последовательность пробуждения путем очистки бита SLEEP, если AWUM бит в регистре CAN\_MCR установлен. Если AWUM бит очищен, программа должна очистить бит SLEEP, когда происходит прерывание пробуждения, для того, чтобы вывести модуль из спящего режима.

Замечание: Если прерывание пробуждения разрешено (WKUIE бит установлен в регистре CAN\_IER) прерывание пробуждения будет сгенерировано по детектированию активности шины CAN, даже если bxCAN автоматически выполнит последовательность пробуждения.

После того, как SLEEP бит очищается, модуль выводится из спящего режима, как только синхронизируется с CAN шиной. Модуль выходит из спящего режима, как только SLAK бит очищается аппаратно.

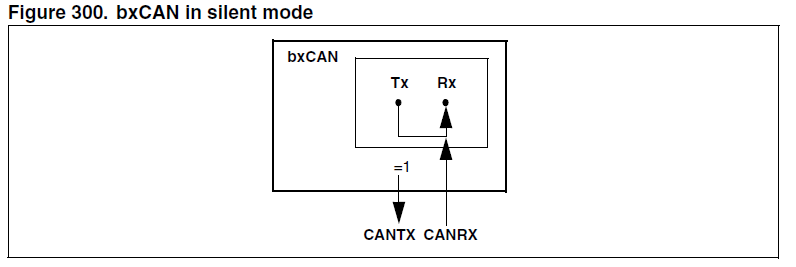


**Тестовый режим.**

Тестовый режим может быть выбран битами SILM и LBKM в регистре CAN\_BTR. Эти биты должны быть сконфигурированы пока bxCAN находится в режиме инициализации. Как только будет выбран тестовый режим, INRQ бит в регистре CAN\_MCR должен быть сброшен для входа в нормальный режим.

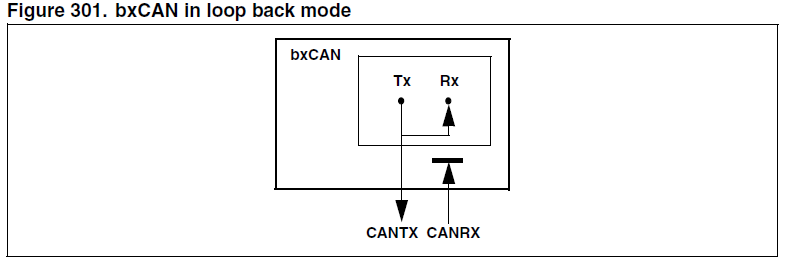
**Тихий режим.**

bxCAN может быть помещен в тихий режим, установкой бита SILM в регистре CAN\_BTR. В тихом режиме, bxCAN способен получать правильные кадры данных и правильные кадры удаленного запроса, но он посылает только recessive биты на CAN шину и это не может начать передачу. Если bxCAN должен послать dominant бит (ACK бит, флаг перегрузки, флаг активной ошибки), то бит перенаправляется внутри контроллера так, что ядро CAN отслеживает этот dominant бит, хотя и CAN шина может оставаться в recessive состоянии. Тихий режим может быть использован для анализа трафика на CAN шине без влияния на нее передачей данных и dominant битами (биты подтверждения, кадровые ошибки).



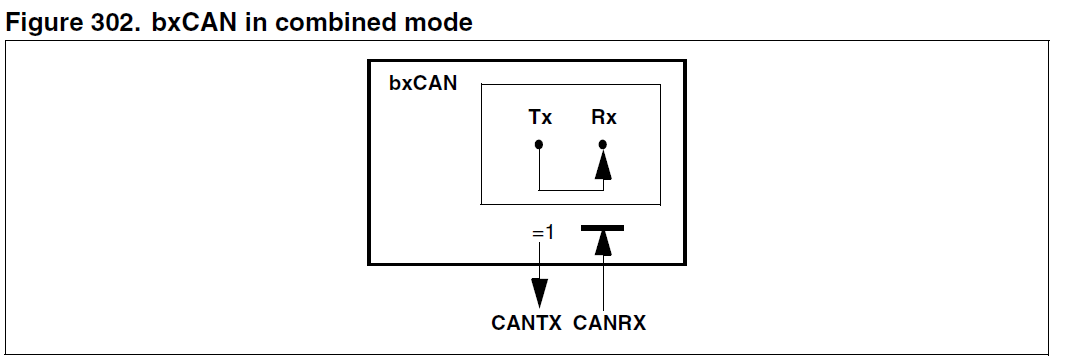
**Режим обратной петли.**

bxCAN может быть установлен в режим обратной петли, установив бит LBKM в регистре CAN\_BTR. В режиме обратной петли, bxCAN обрабатывает свои собственные переданные сообщения как полученные сообщения и сохраняет их (если они прошли приемный фильтр) в приемном почтовом ящике.



**Режим обратной петли, в комбинации с тихим режимом.**

Возможно также скомбинировать режим Loop Back и Silent, установкой бита LBKM и SILM в регистре CAN\_BTR. Этот режим может быть использован для «горячей самодиагностики», что означает, что bxCAN может быть протестирован подобно как в режиме обратной петли, но не вмешиваясь в работающую CAN систему, подключенную к CANTX и CANRX выводам. В этом режиме, вывод CANRX отключен от bxCAN и CANTX вывод удерживается в recessive.



**Режим отладки.**

Когда микроконтроллер входит в отладочный режим, bxCAN продолжает работать нормально или останавливается, в зависимости от:

* DBG\_CAN1\_STOP бита для CAN1 или DBG\_CAN2\_STOP бита для CAN2 в DBG модуле. Для большей информации обратитесь к секции 32.16.2.
* DBF бита в регистре CAN\_MCR. Для большей информации обратитесь к секции 27.9.2.

**Функциональное описание.**

**Обработка передач.**

Для того, чтобы передать сообщение, приложение должно выбрать один пустой почтовый ящик для передачи, установить идентификатор, код длины пакета (DLC) и данные, перед запросом передачи установкой соответствующего бита TXRQ в регистре CAN\_TIxR. Как только почтовый ящик стал непустым, программа больше не может ничего записывать в регистры почтового ящика. Сразу после установки бита TXRQ, почтовый ящик входит в ожидающее состоянии и ждет, пока его приоритет не станет наивысшим. Вскоре, после того, как почтовый ящик получит наивысший приоритет, он будет запланирован для передачи. Передача сообщения планируемого почтового ящика начнется, когда шина CAN станет неактивной (idle). Как только почтовый ящик был успешно передан, он снова станет пустым. Аппаратура индицирует успешную передачу установкой RQCP и TXOK битов в регистре CAN\_TSR.

Если передача провалилась, причина этого отражена битом ALST в регистре CAN\_TSR в случае потери арбитража, и/или битом TERR в случае детектирования ошибки передачи.

**Приоритет передачи.**

**По идентификатору.**

Когда более чем один почтовый ящик ожидает отправки, порядок передачи определяется по идентификатору сообщения, сохраненного в почтовом ящике. Сообщение с низшим значением значением идентификатора имеет наивысший приоритет в соответствии с арбитражем CAN протокола. Если значения идентификаторов равны, почтовый ящик с низшим номером будет планироваться первым.

**По порядку запроса на передачу.**

Почтовые ящики для передачи могут быть настроены как передающее FIFO, установкой бита TXFP в регистре CAN\_MCR. В этом режиме порядок приоритета определяется порядком запроса на передачу. Этот режим очень полезен для сегментированных передач.

**Отмена.**

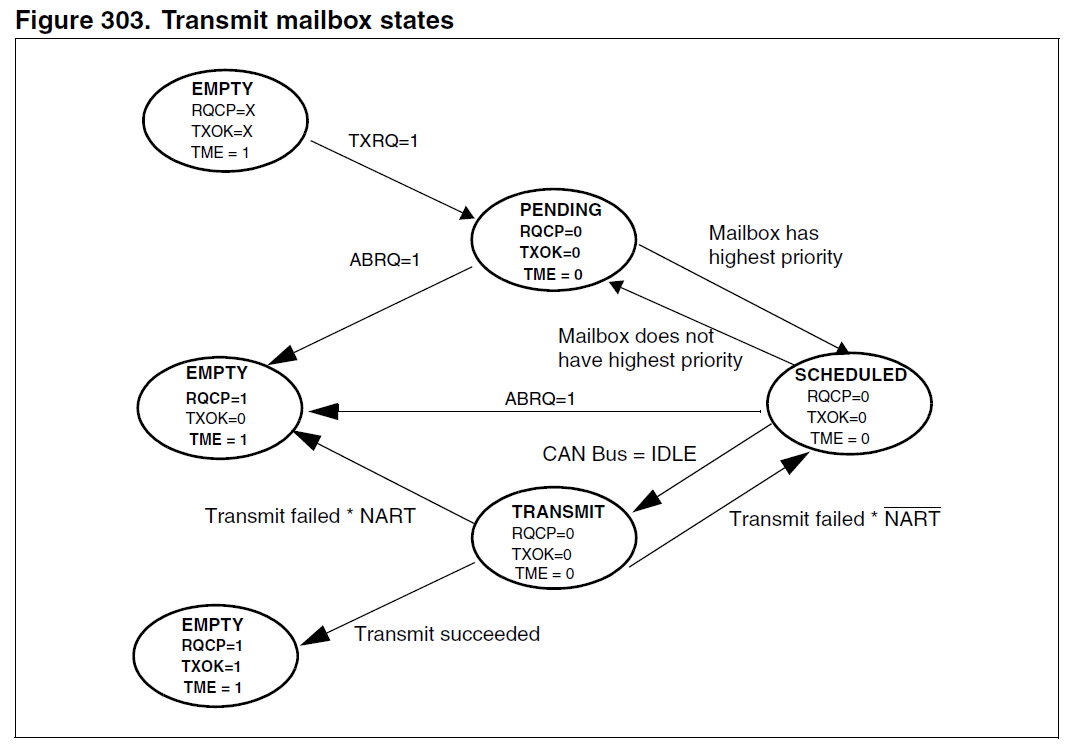
Запрос на передачу может быть отменен пользовательской установкой бита ABRQ в регистре CAN\_TSR. В ожидающем или планируемом состоянии, почтовый ящик отменяется мгновенно. Запрос отмены во время передачи почтового ящика может дать два результата. Если почтовый ящик успешно передан, почтовый ящик становится пустым с установленным битом TXOK в регистре CAN\_TSR. Если передача проваливается, почтовый ящик становится запланированным, передача отменяется и почтовый ящик становится пустым с очищенным TXOK битом. Во всех случаях почтовый ящик станет снова пустым по крайней мере в конце текущей передачи.

**Ручной режим повторной передачи.**

Этот режим был реализован для того, чтобы выполнить требование опции Time Triggered Communication стандарта CAN. Для настройки аппаратного обеспечения в этом режиме бит NART в регистре CAN\_MCR должен быть установлен.

В этом режиме, каждая передача стартует только раз. Если первая попытка проваливается, из за потери арбитража или ошибки, аппаратура не будет автоматически повторять передачу сообщения.

В конце первой попытки передачи, аппаратура считает запрос завершенным и устанавливает бит RQCP в регистре CAN\_TSR. Результат передачи индицируется в регистре CAN\_TSR битами TXOK, ALST и TERR битами.



**Режим time triggered communication.**

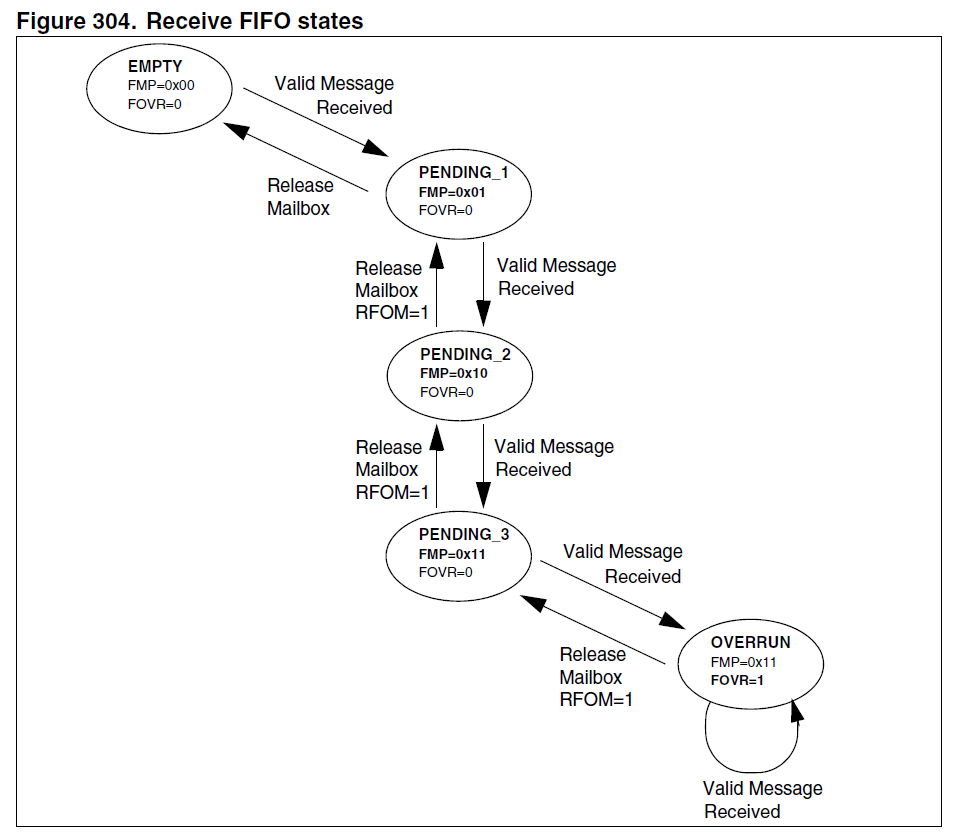
В этом режиме активируется внутренний счетчик CAN аппаратуры, который используется для генерации значения Time Stamp (временной метки), сохраненным в регистрах CAN\_RDTxR/CAN\_TDTxR, соответственно (для Rx и Tx почтовых ящиков). Внутренний счетчик инкрементируется каждый битовый интервал (обратитесь к секции бит тайминга). Внутренний счетчик захватывается на точке захвата бита стартового кадра передачи и приема.

**Обработка приема.**

Для приема CAN сообщений, три почтовых ящика организованы как FIFO. Для экономии нагрузки на процессор, упрощения программного обеспечения и гарантии непротиворечивости данных, FIFO полностью управляется аппаратно. Приложение получает доступ к сообщениям, сохраненным в FIFO через выходной почтовый ящик FIFO.

**Корректность сообщения.**

Принятое сообщение считается допустимым, когда оно было принято корректно, в соответствии с CAN протоколом (нет ошибок до предпоследнего бита EOF) и успешно пропущено через фильтр.



**Управление FIFO.**

Начиная с пустого состояния, первое принятое корректное сообщение сохраняется в FIFO, которое становится pending\_1. Аппаратура сигнализирует о событии, устанавливая биты FMP[1:0] в регистре CAN\_RFR в значение 01b. Сообщение доступно в выходном почтовом ящике FIFO. Программа считывает содержимое почтового ящика и освобождает его, устанавливая бит RFOM в регистре CAN\_RFR. FIFO снова становится пустым. Если тем временеи было принято новое корректное сообщение, FIFO остается в состоянии pending\_1 и новое сообщение доступно в выходном почтовом ящике.

Если приложение не освобождает почтовый ящик, следующее корректное сообщение будет сохранено в FIFO, который входит в состояние pending\_2 (FMP[1:0] = 10b). Процесс сохранения повторяется для следующего корректного сообщения, помещаемого в FIFO, который входит в состояние pending\_3 (FMP[1:0] = 11b). На этой точке, программа должна освободить почтовый ящик, установив бит RFOM, чтобы почтовый ящик стал свободным для сохранения следующего корректного сообщения. Иначе, следующее корректное принятое сообщение будет потеряно.

**Переполнение.**

Если FIFO находится в состоянии pending\_3 (т.е. три почтовых ящика заполнены), прием следующего корректного сообщения приведет к переполнению и сообщение будет потеряно. Аппаратура сигнализирует о возникновения переполнения, устанавливая бит FOVR в регистре CAN\_RFR. Какое сообщение терять, зависит от настройки FIFO:

* Если блокирующая функция FIFO отключена (RFLM бит в регистре CAN\_MCR очищен), то последнее сохраненное сообщение в FIFO будет перезаписано новым входящим сообщением. В этом случае последние прибывшие сообщения будут всегда доступны приложениею.
* Если блокирующая функция FIFO разрешена (RFLM бит в регистре CAN\_MCR установлен) последнее пришедшее сообщение будет отброшено и программа сможет получить доступ к трем старым сообщениям в FIFO.

**Прерывания, относящиеся к приему.**

Как только сообщение было сохранено в FIFO, биты FMP[1:0] обновляются и генерируется запрос прерывания, если FMPIE бит в регистре CAN\_IER установлен.

Когда FIFO заполнено (т.е. третье сообщение сохранилось), бит FULL в регистре CAN\_RFR устанавливается и прерывание генерируется, если бит FFIE в регистре CAN\_IER установлен.

При переполнении, бит FOVR устанавливается и прерывание генерируется, если FOVIE бит регистра CAN\_IER установлен.

**Фильтрация идентификаторов.**

В CAN протоколе идентификатор сообщения не связан с адресом узла, но связан с самим сообщением. Следовательно, передатчик передает сообщения его узла широковещательно ко всем приемникам. После приема сообщения принимающий узел решает на основе идентификатора принятого сообщения, нужно ли программе это сообщение или нет. Если сообщение нужно, оно копируется в SRAM. Если нет, сообщение должно быть отброшено без вмешательства программы.

Для выполнения этого требования, bxCAN контроллер предоставляет 28 настраиваемых и масштабируемых банков фильтров (27-0) приложению. В других устройствах bxCAN контроллер предоставляет 14 настраиваемых и масштабируемых банков фильтров (13-0) приложению для того, чтобы получать только те сообщения, которые нужны программе. Такая аппаратная фильтрация сохраняет ресурсы ЦП, которые могли бы в противном случае отфильтровывать сообщения программно. Каждый банк фильтра состоит из двух 32 битных регистров, CAN\_FxR0 и CAN\_FxR1.

**Масштабируемая ширина.**

Для оптимизации и адаптирования фильтров для нужд приложения, каждый банк фильтра может быть отмасштабирован независимо. В зависимости от масштаба фильтра, банк фильтра предоставляет:

* Один 32-битный фильтр для STDID[10:0], EXTID[17:0], IDE и RTR битов.
* Два 16-битных фильтра для STDID[10:0], RTR, IDE и EXTID[17:15] битов.

См. рисунок 305.

Более того, фильтры могут быть настроены в режиме маскирования, или в режиме списка идентификаторов.

**Режим маскирования.**

В режиме маскирования, регистры идентификаторов связаны с регистрами маски, которые определяют, какие биты идентификатора обрабатывать как «должен соответствовать» или «оставить без внимания».

**Режим списка идентификаторов.**

В режиме списка идентификаторов, регистры маски используются как регистры идентифкаторов. Таким образом, вместо определения идентификатора и маски, определены два идентификатора. Все биты входящего идентификатора должны соответствовать битам, определенным в регистрах фильтра.

**Масштаб банка фильтра и конфигурация режима.**

Банки фильтров настраиваются посредством соответствующих CAN\_FMR регистров. Для настройки банка фильтра он должен быть деактивирован очисткой бита FACT в регистре CAN\_FAR. Масштаб фильтра настраивается посредством соответствующего FSCx бита в регистре CAN\_FS1R. (различные конфигурации банка фильтра см. на рисунке 305). Режим списка идентификаторов или режим маски идентификатора настраивается посредством FBMx битов в регистре CAN\_FMR.

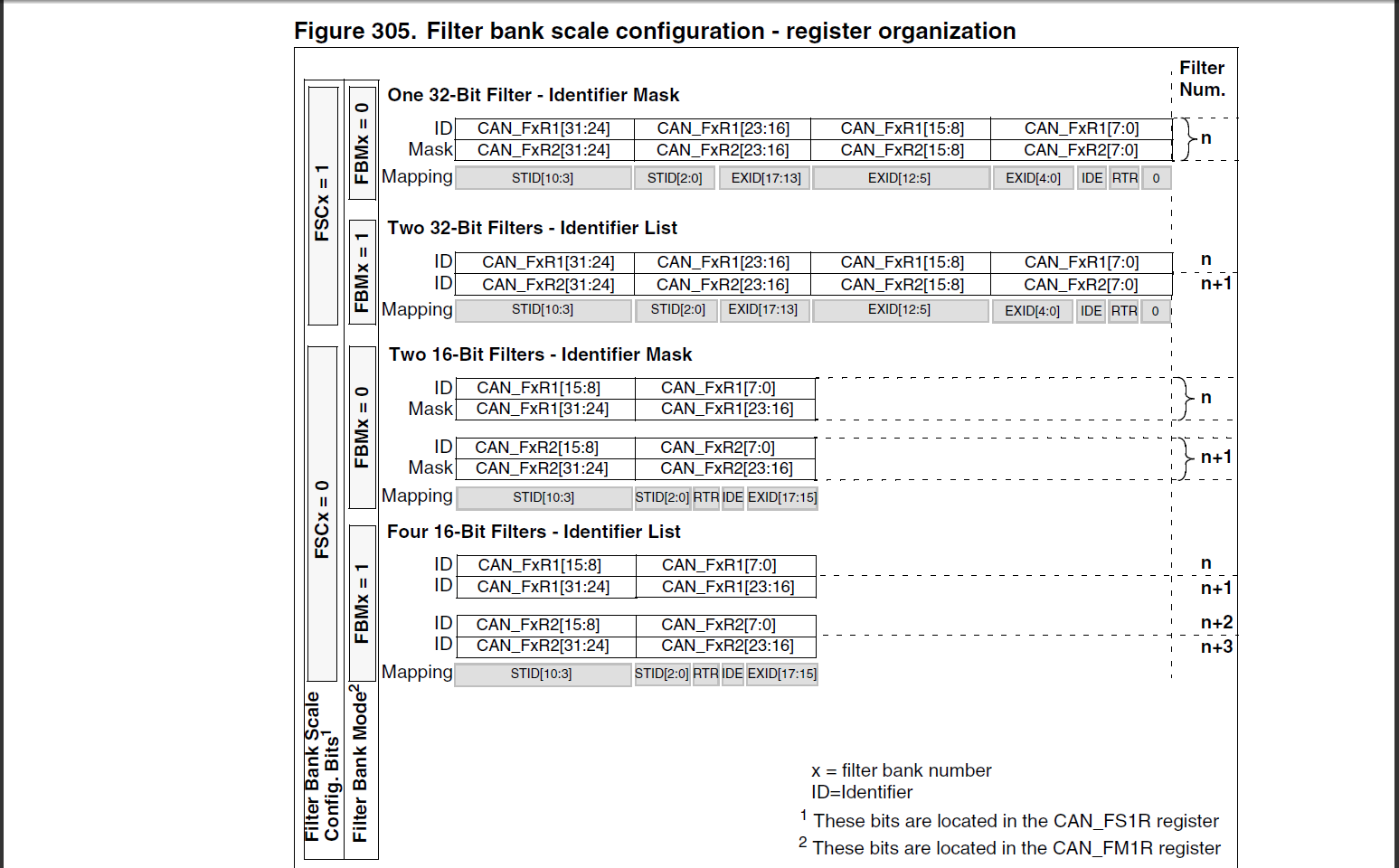
Для фильтрации группы идентификаторов, настройте регистры маски/идентификаторов в режим маскирования.

Для выбора одиночных идентификаторов, настройте регистры маски/идентификаторов в режим списка идентификаторов.

Фильтры не используемые приложением должны быть оставлены неактивными.

Каждый фильтр в пределах банка фильтра пронумерован (так называемым номером фильтра) от 0 до максимума в зависимости от режима и масштаба каждого из банков фильтров.

Относительно настройки фильтра, обратитесь к рисунку 305.



**Индекс совпавшего фильтра.**

Как только сообщение было принято в FIFO, оно становится доступным приложению. Обычно, данные приложения копируются в SRAM память. Для копирования данных в правильную позицию, приложение должно идентифицировать данные посредством идентификатора. Для избежания этого, и упрощения доступа к SRAM, CAN контроллер предоставляет индекс совпавшего фильтра.

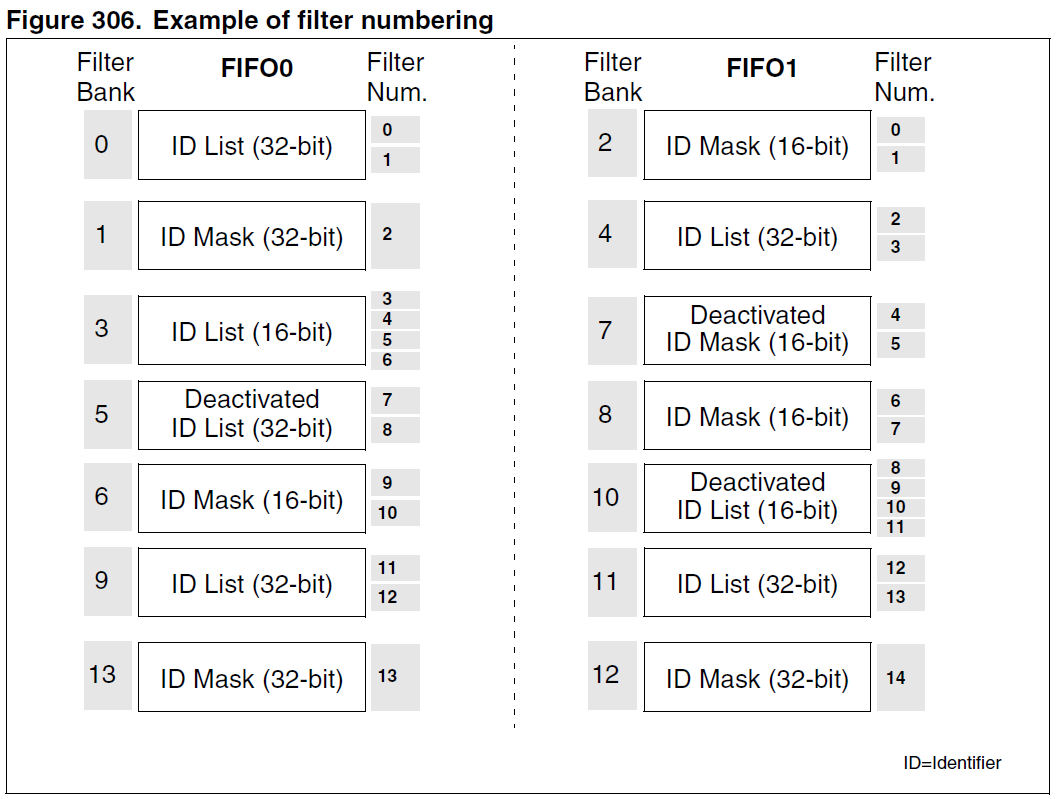
Этот индекс сохраняется в почтовом ящике вместе с сообщением, согласно правилам приоритета фильтра. Таким образом, каждое принятое сообщение связано с его индексом совпадения. Индекс совпадения фильтра может быть использован двумя способами:

* Сравнение индекса совпадения фильтра со списком ожидаемых значений.
* Использовать индекс совпадения фильтра как индекс массива принятых пакетов.

Для немаскируемых фильтров, программа больше не должна сравнивать идентификатор.

Если фильтр замаскирован, программа сравнивает только маскированные биты.

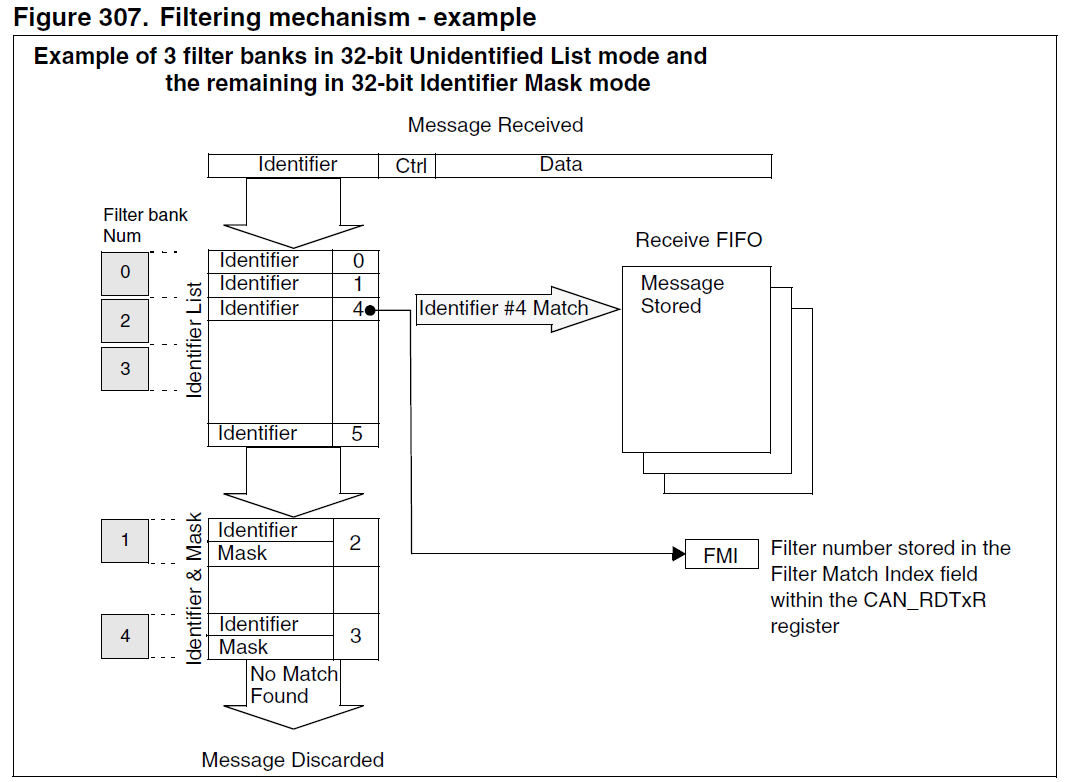
Значение индекса фильтра не принимает во внимание состояние активности банков фильтров. Кроме этого, используется две независимых схемы нумерации, по одной на FIFO. Смотри рисунок 306 для примера.



**Правила приоритета фильтрации.**

В зависимости от комбинации фильтрации, может произойти событие, при котором идентификатор передается успешно сквозь несколько фильтров. В этом случае значение совпавшего фильтра сохраняется в почтовом ящике и выбирается в соответствии со следующими правилами:

* 32 битный фильтр имеет приоритет перед 16 битным.
* Для фильтров равного масштаба, приоритет имеет режим списка идентификаторов относительно режима маси идентификатора.
* Для фильтров равных масштабов и режимов, приоритет задается номером фильтра (младший номер имеет более высокий приоритет).



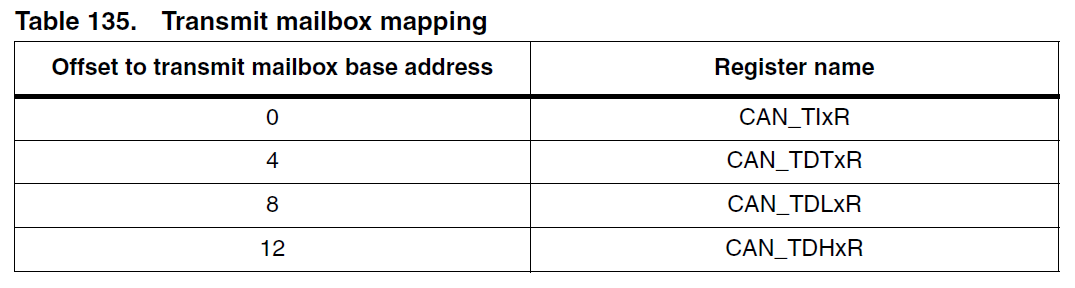
Вышеприведенный пример показывает принцип фильтрации. При приеме сообщения, идентификатор сравнивается сперва с фильтрами, настроенными в режиме списка идентификаторов. Если есть совпадение, сообщение сохраняется в соответствующем FIFO и индекс совпавшего фильтра сохраняется вместе с ним. Как показано в примере, совпадение идентификатора произошло во втором банке. Таким образом, содержимое сообщения и FMI 2 сохраняется в FIFO. Если совпадения не было, входящий идентификатор сравнивается с фильтрами, настроенными в режиме маски. Если идентификатор не соответствует фильтрам, сообщение отбрасывается аппаратно без участия программы.

**Хранилище сообщения.**

Интерфейс между программным обеспечением и аппаратным для CAN сообщений реализован посредством почтовых ящиков. Почтовый ящик содержит всю информацию, относящуюся к сообщению; идентификатор, данные, управляющую информацию, статус и временную отметку.

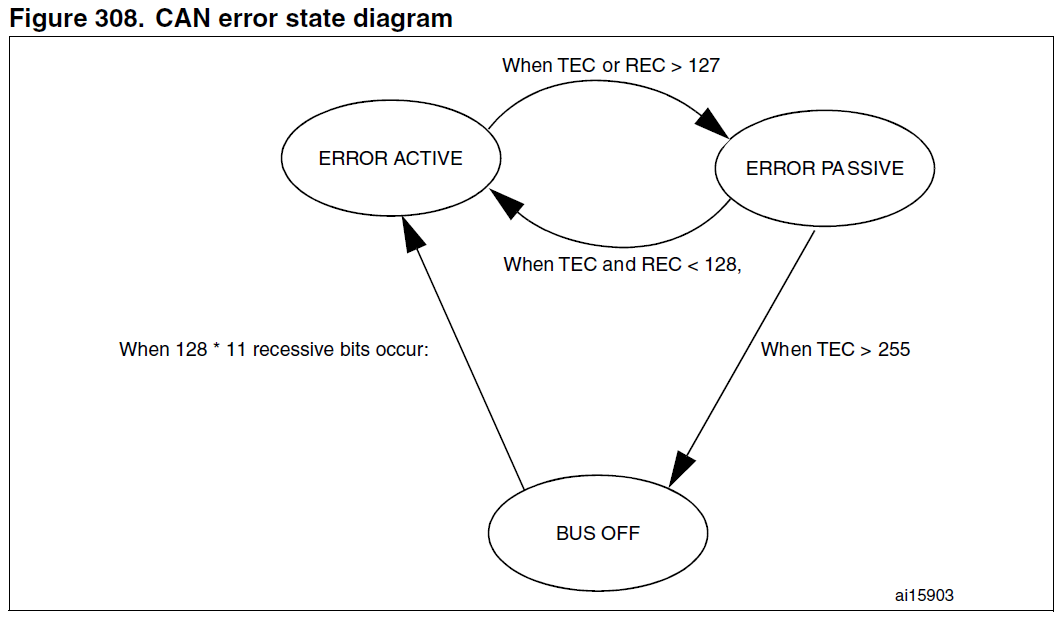
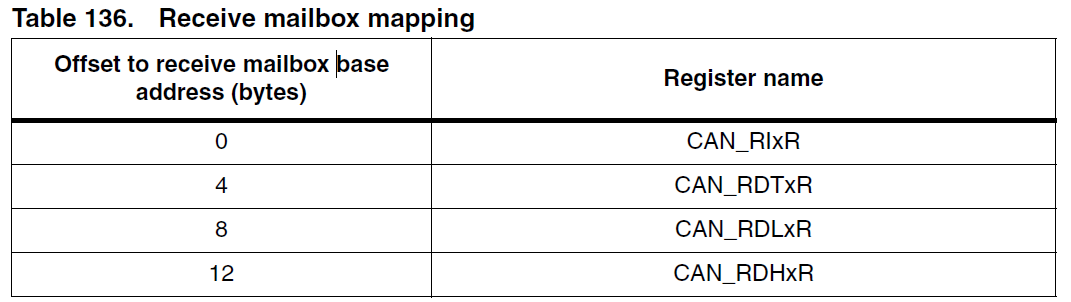
**Передача почтового ящика.**

Программа устанавливает сообщение для передачи в пустой почтовый ящик. Статус передачи индицируется аппаратно в регистре CAN\_TSR.



**Прием почтового ящика.**

Когда сообщение было принято, оно доступно для программы в выходном почтовом ящике FIFO. Как только программа обработала сообщение (например, прочитала его), программа должна освободить выходной почтовый ящик посредством бита RFOM в регистре CAN\_RFR чтобы дать ему возможность принимать следующее входящее сообщение. Индекс совпавшего фильтра сохраняется в поле MFMI регистра CAN\_RDTxR. 16-битное значение временной метки сохраняется в поле TIME[15:0] регистра CAN\_RDTxR.



**Управление ошибками сети.**

Управление ошибки как описано в CAN протоколе обрабатывается полностью аппаратно, используя счетчик ошибок передач (TEC) в регистре CAN\_ESR и счетчик ошибок приема (REC) в регистре CAN\_ESR, который инкрементируется или декрементируется в соответствии с условием ошибки. Для детальной информации об управлении TEC и REC, пожалуйста, обратитесь к стандарту CAN.

Оба из них могут быть прочитаны программно для определения стабильности сети. Более того, аппаратные средства CAN модуля предоставляют детальную информацию о текущей ошибке в регистре CAN\_ESR. Посредством CAN\_IER регистра (ERRIE бита и других), программа может настроить генерацию прерывания по определению ошибки весьма гибким способом.

**Восстановление шины после отключения.**

Состояние отключения от шины достигается, когда TEC счетчик превышает 255. Это состояние индицируется битом BOFF в регистре CAN\_ESR. В состоянии отключения от шины, bxCAN больше не может передавать и принимать сообщения.

В зависимости от бита ABOM в регистре CAN\_MCR, bxCAN восстановит подключение к шине (снова вернет состояние активной ошибки) либо автоматически, либо по запросу программы. Но в обоих случаях, bxCAN должен подождать как минимум втечение последовательности восстановления, определенной в стандарте CAN (128 возникновений 11 последовательных recessive битов, отслеживаемых на CANRX выводе).

Если ABOM бит установлен, bxCAN начнет отслеживать последовательность восстановления автоматически, после того, как войдет в режим отключения от шины.

Если ABOM бит сброшен, программа должна инициировать последовательность восстановления, запрашивая bxCAN войти и оставить режим инициализации.

Замечание: в режиме инициализации, bxCAN не следит за сигналом CANRX, поэтому он не может завершить последовательность восстановления. Для восстановления, bxCAN должен быть в нормальном режиме!

**Бит-тайминг.**

Логика бит-тайминга отслеживает последовательную шину и выполняет сэмплинг (захват) и подстройку точки захвата путем синхронизации по фронту стартового бита и ресинхронизации по последующим фронтам.

Эта операция может быть просто пояснена, если разбить номинальный битовый интервал на три сегмента:

* Сегмент синхронизации (SYNC\_SEG): ожидается, что изменение бита в пределах этого временного сегмента. Он имеет фиксированную длину, равную одной кванте времени (1 x tcan).
* Битовый сегмент 1 (BS1): определяет расположение точки захвата. Это включает сегменты PROP\_SEG и PHASE\_SEG1 стандарта CAN. Его длительность программируется от 1 до 16 квант времени, но также может быть автоматически удлинена для компенасции положительных фазовых сдвигов из за разницы в частоте различных узлов сети.
* Битовый сегмент 2 ( BS2): определяет расположение точки передачи (перехода). Он представляет собой PHASE\_SEG2 стандарта CAN. Его длительность программируется от 1 до 8 временных квант, но может также быть автоматически сокращена для компенсации негативных фазовых сдвигов.

Ширина прыжка ресинхронизации (SJW) определяет верхнюю границу величны удлинения или сокращения битовых сегментов. Она программируется от 1 до 4 квант времени.

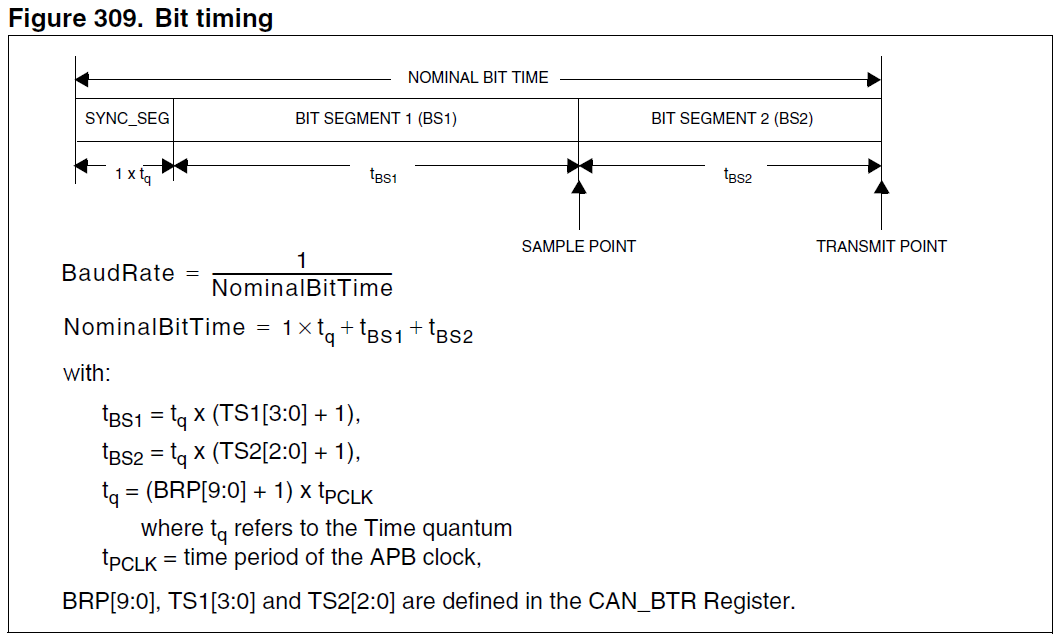
Корректный фронт определен как первый переход в битовом интервале из dominant состояния в recessive, обеспечиваемый самим контроллером (не отсылающий recessive бит).

Если корректный фронт определился в BS1 вместо SYNC\_SEG, BS1 расширяется до SJW (на SJW) так, что точка захвата задерживается.

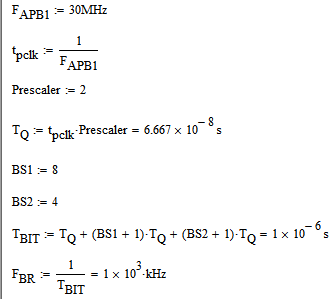
Напротив, если корректный фронт детекируется в BS2 вместо SYNC\_SEG, BS2 укорачивается на SJW величину, так что точка передачи (перехода) появляется раньше.

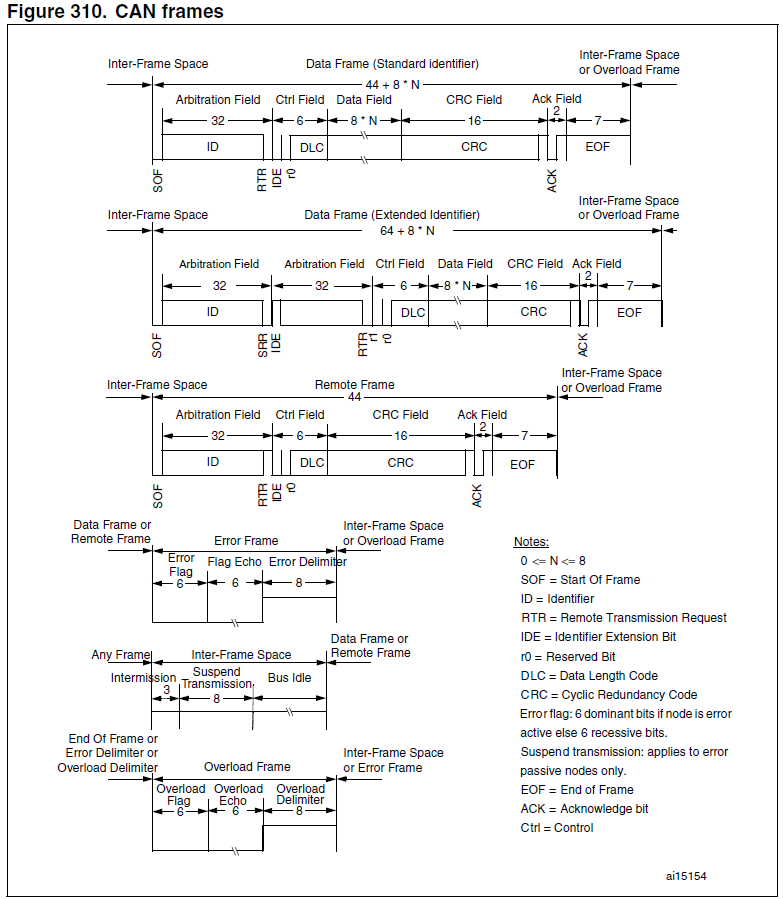
Для гарантии защиты от программных ошибок настройка бит-тайминга CAN\_BTR возможна только пока устройства работает в ждущем (Standby) режиме.

Замечание: для детального описания CAN бит тайминга и механизма ресинхронизации, пожалуйста обратитесь к стандарту ISO 11898.



Пример настройки для скорости в 1 Мб/с.





**Прерывания bxCAN.**