МІНІСТЕРСТВО ОСВІТИ І НАУКИ, МОЛОДІ ТА СПОРТУ УКРАЇНИ

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ „КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ”

КАФЕДРА АВТОМАТИКИ ТА УПРАВЛІННЯ У ТЕХНІЧНИХ СИСТЕМАХ

Навчальний посібник з дисципліни «Проектування мікропроцесорних систем та мереж» Розділ «Проектування CAN-мережі»

Київ НТУУ “КПІ” 2011

Навчальний посібник з дисципліни «Проектування мікропроцесорних систем та мереж», розділ «Проектування CAN-мережі» для студентів спеціальності 8.050201.01 «Комп'ютеризовані системи управління та автоматика» кафедри Автоматики та управління у технічних системах / Автор: А.О. Новацький – К: НТУУ „КПІ", 2016– 184 с.

Автор: А.О. Новацький
Відповідальний за випуск: Л.Ю. Юрчук
Рецензенти: А.Ю.Дорошенко, О.А.Чемерис

Зміст

1 ЗАГАЛЬНІ ВІДОМОСТІ

Розроблений у середині 1980-х фірмою “Bosch” для систем керування вузлами автомобіля, протокол CAN (Controller Area Network – мережа контролерів) є послідовним протоколом високошвидкісної та високонадійної передачі даних у широкомовному (broadcast) режимі в мультимайстерному середовищі. Вдале сполучення низької вартості під’єднання, простоти й надійності з доступністю елементної бази та інструментальних засобів розробки – одні з основних переваг CAN-технології. Положення стандарту, закріплені у використовуваній на сьогоднішній день специфікації 2.0 A/B фірми “Bosch” та міжнародному стандарті ISO 11898, відповідають двом початковим рівням (фізичному та канальному) семирівневої моделі взаємодії відкритих систем ISO/OSI. Ряд оригінальних технічних рішень, реалізованих при розробці протоколу, якнайкраще дозволили зорієнтувати його на побудову мереж.

В даній роботі ми будемо вести розгляд промислових мереж на прикладі CAN-мережі. Основними перевагами цього стандарту є:

- зрілість стандарту. Протокол вже більше 20 років активно використовується в цілому світі. На сучасному ринку представлені тисячі різних CAN-виробів;
- апаратна підтримка протоколу;
- високонадійний алгоритм передачі та обробки помилок, механізм, що дозволяє від’єднувати несправний віддалений вузол і тим самим не допускати блокування всієї мережі;
- хороша підтримка систем реального часу й систем, керованих подіями.

CAN має асинхронну послідовну структуру шини з одним логічним сегментом мережі. CAN-мережа може складатися з двох або більше вузлів, з можливістю під’єднання та від\єднання вузлів від шини без переналагоджування інших пристроїв. Логіка шини працює за механізмом «монтажне І», у якому «рецесивний» біт відповідає логічній одиниці, а «домінантний» логічному нулю.

Поки жоден вузол не формує «домінантний» біт, шина перебуває в «рецесивному» стані, але «домінантний» біт від будь-якого вузла створює «домінантний» стан шини. Тому при виборі середовища передачі даних необхідно визначити який стан буде «домінантний», а який – «рецесивний». Одним із найпоширеніших і дешевих варіантів є пара скручених дротів (кручена пара). Лінії шини тоді називаються CANH і CANL і можуть бути під'єднані безпосередньо до пристроїв. Ніякого додаткового стандарту для середовища передачі даних не існує. Через диференціальний характер лінії зв'язку, шина CAN мало чутлива до електромагнітних завад. У випадку підвищених вимог до надійності у високошвидкісних режимах роботи застосовують екранування шини. Двійкова інформація кодується NRZ-кодом. Для гарантованої синхронізації даних всіма вузлами використовується додатковий наповнюючий біт (виконується біт-стаффінг). При послідовній передачі п'яти біт однакової полярності, передавач вставляє один додатковий біт протилежної полярності після п'яти однакових, які ідуть підряд. Приймач також перевіряє полярність та знищує додаткові біти (виконує дебіт-стаффінг). Структура типової CAN-мережі в складі мікропроцесорної системи керування показана на рисунку 1.1.

Код NRZ (Non Return to Zero – без повернення до нуля) – це найпростіший двійковий код, що передається звичайним цифровим сигналом. Логічному нулю відповідає високий рівень напруги в кабелі, логічній одиниці – низький рівень напруги (або навпаки, що не принципово). Рівні можуть бути різної полярності (додатної та від'ємної) або ж однієї полярності (додатної або від'ємної). Протягом бітового інтервалу (bit time, BT), тобто часу передачі одного біта ніяких змін рівня сигналу в кабелі не відбувається.

До беззаперечних переваг коду NRZ відносяться його досить проста реалізація (вихідний сигнал не треба ні спеціально кодувати на передавальному кінці, ні декодувати на приймальному кінці), а також потрібна мінімальна серед інших кодів пропускна здатність лінії зв'язку. Так як найбільш часта зміна сигналу в мережі буде при безперервному чергуванні одиниць і нулів, тобто при послідовності 1010101010, тому при швидкості передачі, що дорівнює 1 Мбіт/с

(тривалість одного біта дорівнює 1 мс) частота зміни сигналу і відповідно необхідна пропускна здатність лінії становитиме 1/2000нс = 500 КГц.

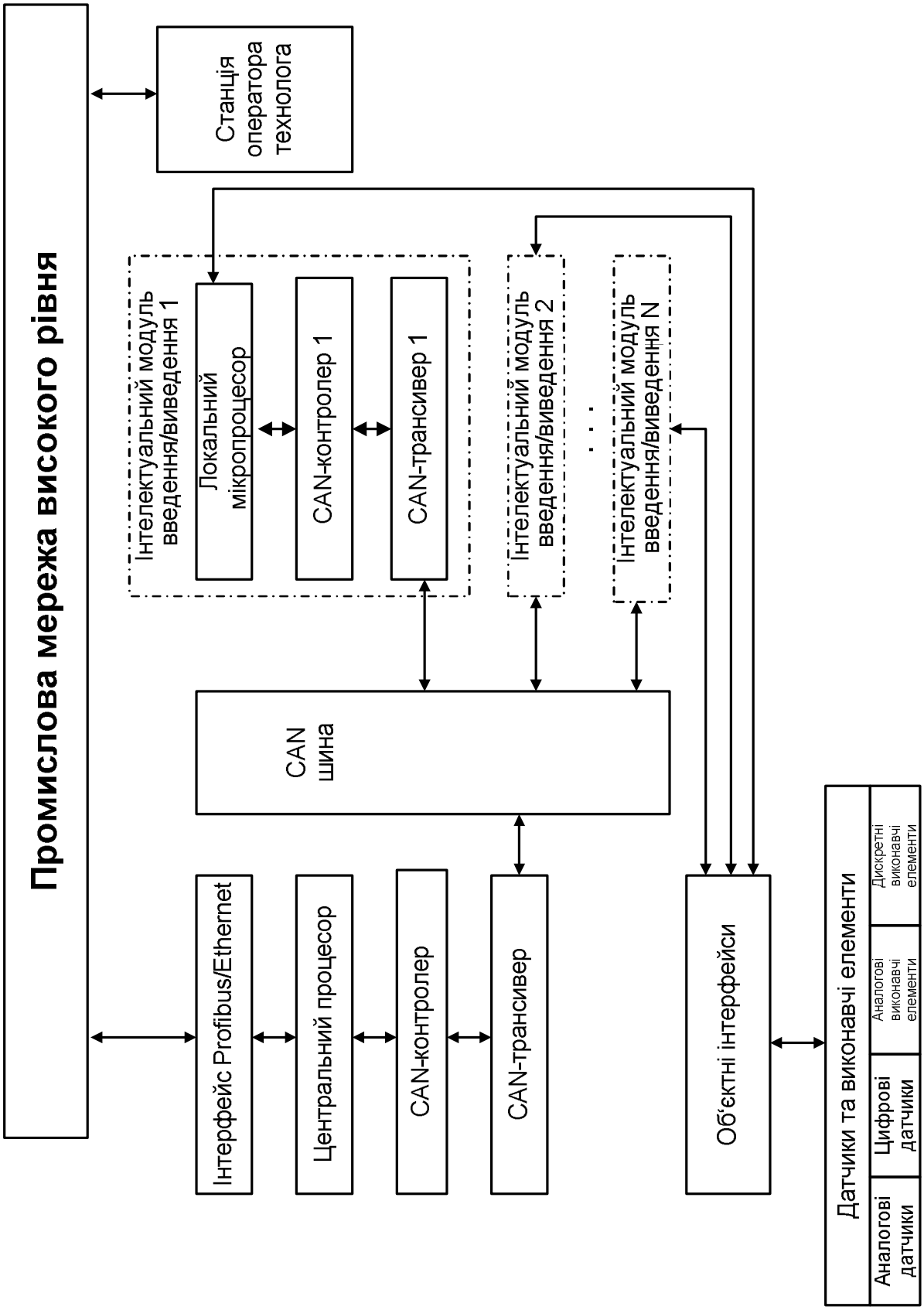


Рисунок 1.1 – Мікропроцесорна система керування

При колективному доступі до шини використовується неруйнівний арбітраж з опитуванням стану шини. Перед початком передачі даних вузол перевіряє стан шини (відсутність активності на шині). Як тільки починається передача повідомлення, вузол стає керуючим для шини, а всі інші вузли переходять у режим прийому. Кожен вузол видає підтвердження прийому, перевіряє ідентифікатор повідомлення, обробляє або знищує прийняті дані. Якщо два або більше вузли починають передачу одночасно, порозрядний арбітраж дозволяє уникнути конфлікту на шині. Кожен вузол видає на шину свій ідентифікатор і контролює її стан. Якщо вузол посилає «рецесивний» біт, а читає «домінантний», значить арбітраж втрачено і вузол перемикається в стан прийому. Таким чином, вузол з більш високим пріоритетом виграє арбітраж без необхідності повторювати передачу повідомлення. Всі інші вузли будуть намагатися передати повідомлення після того, як шина звільниться. Даний механізм не дозволяє передавати одночасно повідомлення з однаковим ідентифікатором, оскільки пізніше можуть виникати помилки. Оригінальна специфікація CAN визначає довжину ідентифікатора тільки 11 біт (2048 можливих варіанти). Щоб уникнути зазначеного обмеження, пізніше технічні вимоги були змінені, і у версії CAN 2.0B довжина ідентифікатора може бути 11 або 29 біт (536 мільйонів варіантів). Повідомлення, що містять 11-розрядний ідентифікатор, називаються стандартними повідомленнями та відповідають специфікації CAN 2.0A.

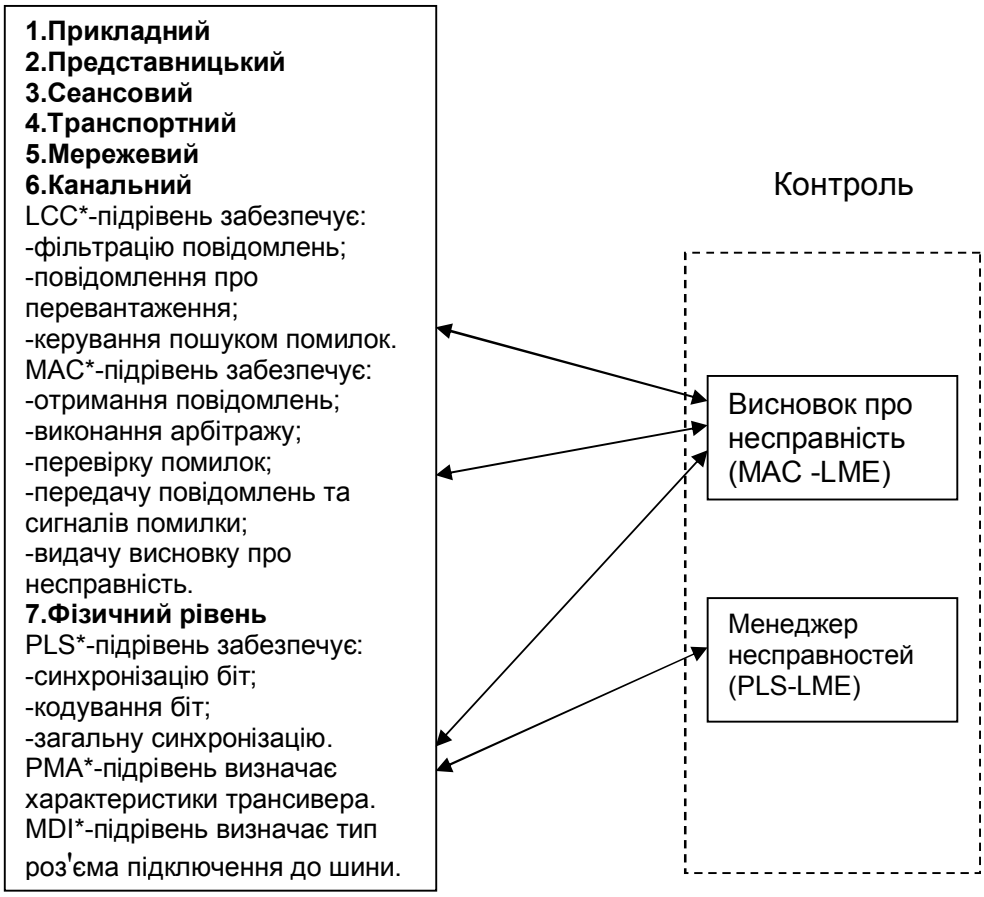
При цьому можливо використовувати лише 2048 різних ідентифікаторів, з яких 16 ідентифікаторів з найменшим пріоритетом - зарезервовані (2032-2047). CAN-модулі версії 2.0A можуть тільки передавати або приймати дані. Версія 2.0B дозволяє активним вузлам робити віддалений стандартний або розширений запит та отримувати дані від потрібного пристрою, використовуючи відповідно стандартне або розширене повідомлення.

Міжнародною організацією по стандартизації ISO (International Standards Organization) була розроблена еталонна модель взаємозв'язку відкритих систем OSI (Open System Interconnection). ISO/OSI еталонна модель використовується для того, щоб визначити рівні протоколу системи зв'язку. На верхньому рівні

моделі перебувають додатки програми (застосовання), які повинні зв'язуватися одна з одною. На найнижчому рівні моделі розташоване фізичне середовище, що забезпечує передачу інформації між вузлами. ISO/OSI еталонна модель CAN-шини зображена на рисунку 1.2.

Верхні рівні мережевої моделі CAN реалізуються програмно. У технічних вимогах до CAN-протоколу немає ніяких додаткових вимог до змісту повідомлень.

Рівні 1–5 ISO/OSI моделі реалізуються програмно, рівень 6 та PLS*-підрівень апаратно реалізує CAN-модуль, PMA*-підрівень є мікросхемою CAN-трансивера, а MDI*-підрівень – роз'єм для під'єднання до шини.



*Примітка: LLC (logical link control) – підрівень контролю логічного з'єднання, описує верхні компоненти канального рівня ISO/OSI; MAC (medium access control) – підрівень контролю доступу до комунікаційного середовища, представляє нижні компоненти канального рівня ISO/OSI; PLS (physical signaling) – підрівень фізичного рівня. Приймає і передає в схему трансивера бітовий потік. Виконує кодування/декодування, контролює синхронізацію; PMA (physical medium attachment) підрівень фізичного рівня, що відповідає за керування передачею, прийомом, виявлення колізій, синхронізацію і корекцію розсинхронізації. MDI (medium dependent interface) – підрівень, що визначає вимоги до роз'ємів, кабелів та термінаторів.

Рисунок 1.2 – ISO/OSI еталонна модель CAN-шини

2 ОСНОВНІ ХАРАКТЕРИСТИКИ CAN-ПРОТОКОЛУ

До основних характеристик CAN-протоколу можна віднести:

- пріоритетність повідомлень;
- гарантований час відгуку;
- гнучкість конфігурації;
- груповий прийом із синхронізацією часу;
- система несуперечності даних;
- робота у мультимайстерному режимі;
- виявлення помилок та сигналізація про їх наявність;
- автоматична ретрансляція пошкоджених повідомлень, як тільки шина знову стане вільною;
- можливість відрізняти нерегулярні помилки та постійні відмови вузлів, а також автономне вимикання дефектних вузлів.

Зупинимося на названих характеристиках більш докладно.

2.1 Повідомлення

Інформація на шині подається у вигляді фіксованих повідомлень різної, але обмеженої довжини. Коли шина вільна, будь-який підключений вузол може почати передавати нове повідомлення.

2.2 Інформаційна маршрутизація

У системах CAN, немає ніякої інформації щодо конфігурації системи (наприклад, адреса вузла). Це має кілька важливих наслідків, які розглянуті нижче.

2.2.1 Гнучкість системи

Вузли можуть бути додані до CAN-мережі без зміни програмного забезпечення та апаратних засобів будь-якого вузла або прикладного рівня.

2.2.2 Маршрутизація повідомлення

Зміст повідомлення визначається ідентифікатором. Ідентифікатор не вказує адресата повідомлення, але описує значення даних, так, щоб всі вузли в мережі мали здатність вирішити фільтрацією, чи повинні вони реагувати на дане повідомлення чи ні.

2.2.3 Групова обробка

Будь-яке число вузлів може отримувати та одночасно реагувати на одне і теж повідомлення.

2.2.4 Несуперечність даних

В середині CAN-мережі гарантується, що повідомлення одночасно буде прийняте або всіма вузлами або жодним вузлом.

2.3 Швидкість передачі інформації в бітах за секунду

Швидкість CAN може бути різною в різних системах. Однак у конкретній мережі швидкість передачі інформації повинна бути фіксована.

2.4 Пріоритети

Ідентифікатор визначає статичний пріоритет повідомлення протягом доступу до шини.

2.5 Віддалений запит даних

Посилаючи кадр віддаленого запиту даних, вузол може виконувати запит даних у іншого вузла, які той повинен передати у вигляді відповідного кадру даних. Кадр даних і кадр віддаленого запиту даних мають однаковий ідентифікатор.

2.6 Мультимайстер

Коли шина вільна, будь-який вузол, може починати передавати повідомлення. Вузлу з повідомленням більш високого пріоритету, буде передане право на доступ до шини.

2.7 Арбітраж

Кожен раз, коли шина вільна, будь-який вузол може починати передавати повідомлення. Якщо два або більше вузлів починають передавати повідомлення в той самий час, конфлікт доступу до шини вирішується порозрядним арбітражем з використанням ідентифікатора. Механізм арбітражу гарантує, що ні інформація, ні час не будуть втрачені. Якщо кадр даних і кадр віддаленого запиту даних з однаковим ідентифікатором починають передаватися одночасно, то кадр даних має перевагу над кадром віддаленого запиту даних. Протягом арбітражу кожен передавач порівнює рівень переданого біта з рівнем, що спостерігається на шині. Якщо ці рівні однакові, вузол може продовжувати передачу. Коли переданий

одиничний рівень, а спостерігається нульовий рівень, це означає, що вузол втратив право арбітражу і не повинен посилати більше жодного біта.

2.8 Безпека

Для досягнення безпеки передачі даних у кожному вузлі CAN є потужні засоби самоконтролю, виявлення та повідомлення про помилки.

2.9 Виявлення помилок

Для виявлення помилок застосовуються наступні засоби:

- поточний контроль (передавачі порівнюють рівні бітів, що передаються, з рівнями, виявленими на шині);
- контроль правильності прийнятих повідомлень із використанням надлишкового циклічного коду;
- контроль формату переданого кадру.

2.10 Ефективність виявлення помилок

Механізми виявлення помилок дозволяють виявляти:

- всі помилки глобального характеру;
- всі локальні помилки передавача;
- до 5 випадкових помилок у повідомленні;
- послідовну групу помилок завдовжки до 15 біт;
- будь-які помилки непарності.

Загальна залишкова ймовірність помилки для невиявлених спотворених повідомлень: $\leq 4{,}7 \cdot 10^{-11} \cdot \text{ErrorRate}$,

де ErrorRate – частота появи помилки в кадрі [1].

Тобто помилки передач будуть виявлятися з досить високою ймовірністю за допомогою вбудованого механізму виявлення помилок CAN-протоколу.

2.11 Сигналізація помилки та час відновлення

Пошкоджені кадри позначаються будь-яким вузлом, що виявив помилку. Такі кадри перериваються й повинні бути автоматично передані заново. Час відновлення від виявлення помилки до початку наступного кадру – у більшості випадків дорівнює 31 інтервалу передачі біта, якщо не буде наступної помилки.

2.12 Типізація помилок

Вузли CAN здатні відрізнити короткі неполадки від постійних відмов. Дефектні вузли від’єднуються від шини CAN-модуля.

2.13 Під’єднання

Послідовна лінія зв'язку CAN є шиною, до якої теоретично може бути під’єднана будь-яка кількість вузлів. Фактично загальне число вузлів буде обмежено часовою затримкою та (або) електричним навантаженням на лінії шини.

2.14 Канал обміну

Шина складається з каналу обміну, що передає біти. Із цих даних може бути отримана інформація про пересинхронізацію. Спосіб, у який реалізується канал обміну, у цій роботі більш докладно розглядається у розділі 7. Наприклад, це може бути один дріт (плюс земля), два диференціальних дроти, оптичне скловолокно і т.ін.

2.15 Рівні сигналів

Шина може мати одне із двох логічних значень: "домінантне" або "рецесивне". При одночасній передачі "домінантного" та "рецесивного" бітів, значення, що виникає на шині буде "домінантне". Фізичні стани (наприклад, електрична напруга, світло), які представляють фізичні рівні, розглянуті далі. Вважається, що "домінантний" рівень еквівалентний нульовому рівню, а "рецесивний" рівень еквівалентний одиничному рівню.

2.16 Підтвердження

Всі приймачі перевіряють несуперечність отриманого повідомлення, підтверджують несуперечність повідомлення та позначають суперечливі повідомлення.

2.17 Режим "сну/пробудження"

Для зменшення кількості енергії, яка споживається від джерела живлення, CAN-вузол може бути переведений у режим „сну” без внутрішньої активності та з відключеними формувачами шини. Вихід із цього режиму відбувається при появі будь-якої активності на шині або внутрішньому стані системи. При пробудженні, здійснюється внутрішнє перезавантаження. MAC-підрівень буде чекати стабілізації генератора системи, і потім буде очікувати самосинхронізації із сигналами на шині (перевіряючи, одинадцять послідовних одиничних біт), поки вихідні формувачі знову не встановляться в стан „під’єднаний до шини”.

2.18 Генератор

Вимоги до синхронізації дозволяють використовувати керамічні резонатори в системах зі швидкостями передачі до 125 Кбіт/сек. Для більш високої швидкості шини CAN, потрібен кварцовий резонатор.

3 СТРУКТУРА ПОВІДОМЛЕНЬ CAN-МЕРЕЖІ

Повідомлення, що передаються CAN-шиною, називаються кадрами (фреймами). Формати кадрів наведені на рисунках 3.1…3.5. Залежно від ініціатора передачі і її мети існують чотири типи кадрів:

1) Data Frame – кадр даних;
2) Remote Frame – кадр запиту даних;
3) Error Frame – кадр помилки;
4) Overload Frame – кадр перевантаження.

Є два формати повідомлень, які передаються, що відрізняються довжиною поля ідентифікатора:

- кадри з 11-розрядним ідентифікатором, які називаються стандартними кадрами;
- кадри, що містять 29-розрядні ідентифікатори та називаються розширеними кадрами.

Кадр даних передає дані від передавача до приймача. Кадр віддаленого запиту даних передається вузлом, щоб запросити передачу кадру даних від іншого вузла з тим же самим ідентифікатором. Кадр помилки передається будь-яким вузлом при виявленні помилки на шині. Кадр перевантаження використовується, щоб забезпечити додаткову затримку між попереднім і наступним кадром даних або кадром віддаленого запиту даних. Кадри даних і кадри віддалених запитів даних можуть використовуватися і у стандартному і у розширеному форматі. Вони відокремлюються від попередніх кадрів міжкадровим простором.

3.1 Кадр даних (Data frame)

Кадр даних складається із семи різних полів: "початок кадру (повідомлення)" (start of frame), "поле арбітражу" (arbitration field), "поле керування" (control field), "поле даних" (data field), "поле CRC" (CRC field), "поле підтвердження" (ACK

field), "кінець кадру (повідомлення)" (end of frame). Поле даних може мати нульову довжину. На рисунках 3.1, 3.2 наведені структури, відповідно стандартного та розширеного повідомлень.

Нижче наведений опис окремих полів цих повідомлень.

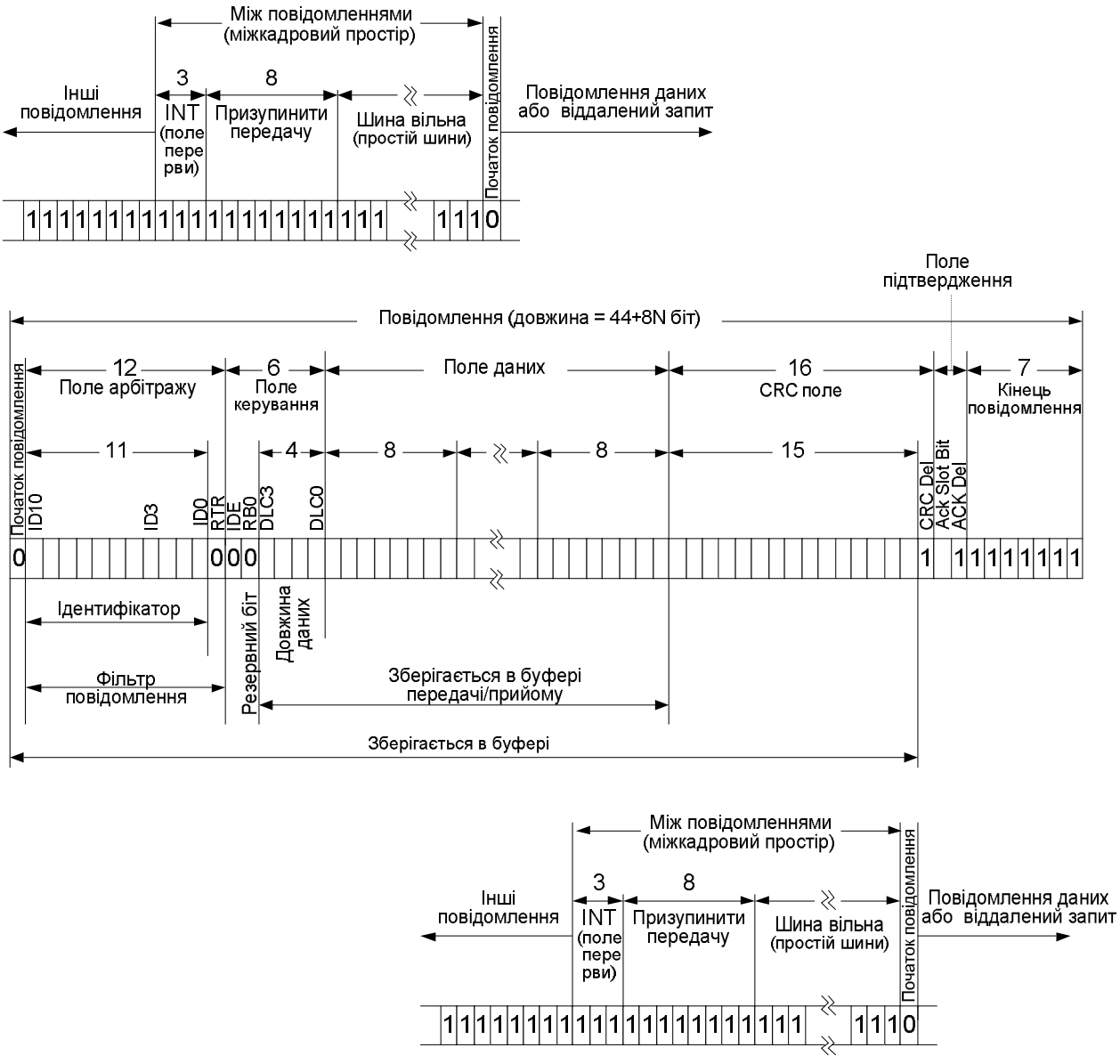


Рисунок 3.1 – Стандартне повідомлення

3.1.1 Початок кадру (стандартний або розширений формат) (Start of frame)

Початок кадру відмічає початок кадру даних або кадру віддаленого запиту даних. Це поле складається з одиночного нульового біта. Вузлу дозволено почати передачу, коли шина вільна (див. міжкадровий простір). Всі вузли повинні синхронізуватися за фронтом, який викликаний передачею поля "початок кадру" (див. апаратна синхронізація) вузлом, що почав передачу першим.

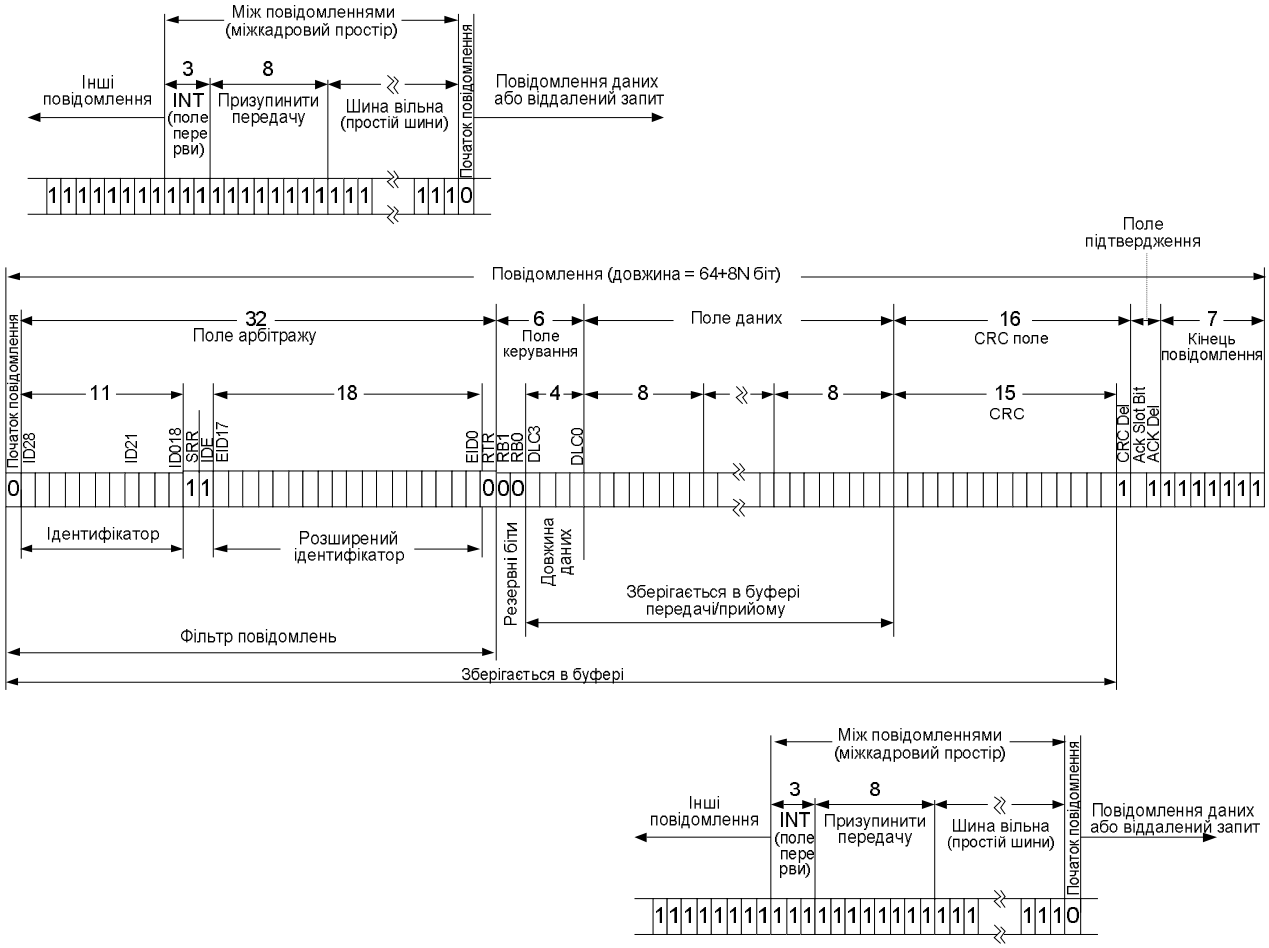


Рисунок 3.2 – Розширене повідомлення

3.1.2 Поле арбітражу (Arbitration field)

Формат поля арбітражу відрізняється для стандартного та розширеного форматів:

- у стандартному форматі поле арбітражу, складається з 11-розрядного ідентифікатора та RTR-біта. Біти ідентифікатора позначені ID10...ID0;
- у розширеному форматі поле арбітражу складається з 29-розрядного ідентифікатора, SRR-біта, IDE-біта, та RTR-біта. Біти ідентифікатора позначені ID28...ID0.

Ідентифікатор (Identifier)

Ідентифікатор стандартного повідомлення.

Довжина ідентифікатора – 11 біт і відповідає BASE ID у розширеному форматі. Ці біти передаються в порядку ID10...ID0. Молодший біт – ID0. Сім старших бітів (ID10 – ID4) не повинні бути одиничними бітами. У стандартному кадрі ідентифікатор супроводжується RTR-бітом, який дорівнює нулю.

Ідентифікатор розширеного повідомлення.

На відміну від стандартного ідентифікатора, розширений ідентифікатор складається з 29 біт. Його формат містить дві секції:

1. Base ID - 11 біт
2. Extended ID - 18 біт

Base ID

Base ID складається з 11 біт. Ця секція передається в порядку від ID28 до ID18. Це еквівалентно формату стандартного ідентифікатора. Base ID визначає базовий пріоритет розширеного кадру.

Extended ID

Extended ID складається з 18 біт. Ця секція передається в порядку від ID17 до ID0. У розширеному кадрі ідентифікатор супроводжується RTR бітом, який дорівнює нулю.

Біт RTR (стандартний і розширений формат)

Біт віддаленого запиту. В кадрах даних RTR біт повинен бути переданий нульовим рівнем. У кадрі віддаленого запиту даних RTR-біт повинен бути одиничним. У розширеному кадрі спочатку передається Base ID, а наступними бітами передаються SRR та IDE. Extended ID передається після IDE біта.

Біт SRR (розширений формат)

Замінник біта віддаленого запиту в стандартному повідомленні. SRR – одиничний біт. Він передається в розширених кадрах на позиції RTR біта. Таким чином, він заміняє RTR – біт стандартного кадру. Отже, при одночасній передачі стандартного кадру та розширеного кадру, Base ID якого збігається з ідентифікатором стандартного кадру, стандартний кадр має перевагу над розширеним кадром.

Біт IDE (розширений формат)

Біт розширеного ідентифікатора IDE. Біт належить:

- полю арбітражу для розширеного формату;
- полю керування для стандартного формату. IDE-біт у стандартному форматі передається нульовим рівнем, у той час як у розширеному форматі IDE біт передається одиничним рівнем.

3.1.3 Поле керування (стандартний формат і розширений формат) (Control field)

Поле керування складається із шести біт. Формат поля керування відрізняється для стандартного й розширеного формату. Кадри в стандартному форматі включають: код довжини даних (DLC), біт IDE, що передається нульовим рівнем (див. вище), та зарезервований біт RB0. Кадри в розширеному форматі включають код довжини даних і два зарезервованих біти RB1 й RB0. Зарезервовані біти повинні бути передані нульовим рівнем.

Код довжини даних (стандартний і розширений формати) (Data length code)

Число байт у полі даних визначається кодом довжини даних. Цей код довжини даних, розміром 4 біти, передається усередині поля керування (таблиця 3.1).

Таблиця 3.1 – Кодування довжини даних

Число байт даних	Код довжини даних			
	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
d – “домінантний” r – “рецесивний”				

Допустиме число байт даних, що передаються: {0,1,...,7,8}.

Інші величини використовуватися не можуть.

3.1.4 Поле даних (стандартний і розширений формати) (Data field)

Поле даних складається з даних, які будуть передані всередині кадру даних. Воно може містити від 0 до 8 байт, кожен з яких містить 8 біт, які передаються, починаючи зі старшого значущого розряду (СЗР).

3.1.5 Поле CRC (стандартний і розширений формати) (CRC field)

Містить послідовність CRC та CRC-роздільник.

Послідовність CRC (стандартний і розширений формати) (CRC Sequence)

Послідовність контролю кадру, яка отримується за правилом надлишкового циклічного коду, найкраще підходить для кадрів із числом бітів менше ніж 127 біт. При обчисленні CRC береться початковий поліном, що визначається як поліном, коефіцієнти якого задані послідовністю біт, які складаються з полів: "початок кадру", "поле арбітражу", "керуюче поле", "поле даних" (якщо є) і, для 15 молодших коефіцієнтів, 0. Цей поліном ділиться на поліном:

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

Залишок від цього поліноміального ділення і є послідовність CRC, яка передається шиною.

Приклад розрахунку поля CRC в кадрі данних в якому передається 1 байт даних (00001111) по ідентифікатору 00000000001 (рис.1)

Довжина поля CRC рівна 15 бітам. В доцільності вибору такої довжини можна переконатися скориставшись наступною формулою:

$$k \geq \frac{d-1}{2}\log_2(m+k+1)$$

де $m = 103$ – довжина інформаційних символів, k – довжина перевірочних символів(поля CRC),

$d = 5$ – мінімальна кодова відстань.

$$15 \geq \frac{5-1}{2}\log_2(103+15+1)$$

Початок повідомлення	Ідентифікатор	RTR	IDE	RB0	Довжина данних	Поле даних	CRC поле	CRC Del	ACK Slot Bit ACK Del	Кінець кадру
0	0 0 0 0 0 0 0 0 0 0 0 1	0	0	0	0 0 0 1	0 0 0 0 1 1 1 1	x x x x x x x x x x x x x x x	1	1	1 1 1 1 1 1 1

Рисунок 3.3 – Повідомлення яке потрібно передати

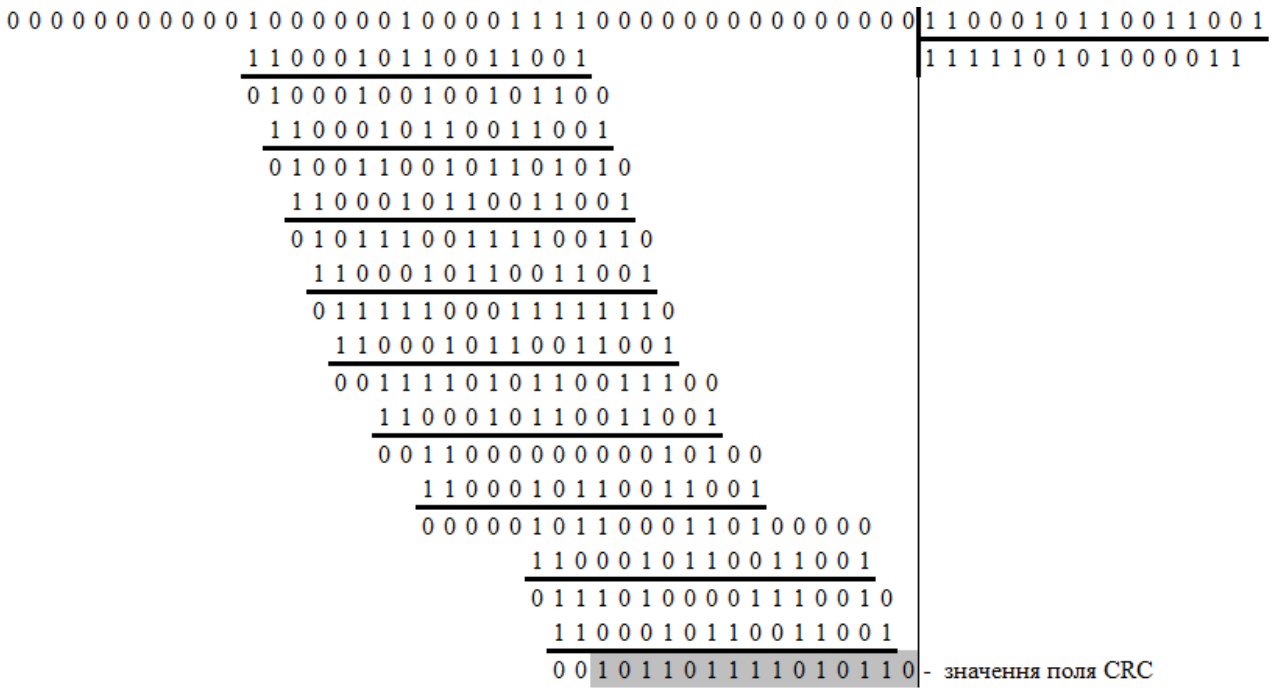


Рисунок 3.4 – Поліноміальне ділення

Початок повідомлення	Ідентифікатор	RTR	IDE	RB0	Довжина данних	Поле даних	CRC поле	CRC Del	ACK Slot Bit	ACK Del	Кінець кадру
0	0 0 0 0 0 0 0 0 0 0 0 1	0	0	0	0 0 0 1	0 0 0 0 1 1 1 1	1 0 1 1 0 1 1 1 1 0 1 0 1 1 0	1		1	1 1 1 1 1 1 1

Рисунок 3.5 – Повідомлення з розрахованим полем CRC

Роздільник CRC (стандартний формат, а також розширений формат) (CRC-Delimiter)

Послідовність CRC супроводжується роздільником CRC, що складається з одного одиничного біта.

Циклічні коди (CRC) названі так тому, що в них частина комбінацій коду або всі комбінації можуть бути отримані шляхом циклічного зсуву однієї або декількох комбінацій коду. Циклічний зсув здійснюється справа наліво, причому крайній лівий символ щоразу переноситься в кінець комбінації. Практично всі циклічні коди відносяться до систематичних кодів – у них контрольні та інформаційні розряди розташовані на чітко визначених місцях. Крім того, циклічні коди відносяться до числа блочних кодів. Кожен блок (одна літера є

окремим випадком блоку) кодується самостійно. Ідея побудови циклічних кодів базується на використанні многочленів, що не приводяться в полі двійкових чисел. Многочлени, що не приводяться – це такі многочлени, які діляться без залишку тільки на себе і на одиницю.

3.1.6 Поле підтвердження (стандартний формат і розширений формат) (ACK-field)

Поле підтвердження має довжину два біти й містить: біт "область підтвердження" і роздільник підтвердження. У поле підтвердження вузол, що передає, посилає два біти з одиничним рівнем. Приймач, що отримав повідомлення правильно, сповіщає про це передавачу, посилаючи біт з нульовим рівнем протягом прийому біта "область підтвердження".

Біт "Область підтвердження" (ACK Slot Bit)

Всі вузли, що отримали повідомлення з відповідною послідовністю CRC, сповіщають про це під час прийому біта "область підтвердження" шляхом заміни біта з одиничним рівнем на біт з нульовим рівнем.

Роздільник підтвердження (ACK-delimiter):

Роздільник підтвердження – другий біт поля підтвердження, і він повинен бути представлений одиничним рівнем.

3.1.7 Кінець кадру (стандартний формат і розширений формат) (End of frame)

Кожен кадр даних і кадр віддаленого запиту даних закінчується послідовністю прапорців, яка складається із семи одиничних біт.

3.2 Кадр віддаленого запиту даних (Remote frame)

Вузол, що діє як приймач деяких даних, може ініціювати передачу відповідних даних потрібними вузлами, посилаючи кадр віддаленого запиту даних (рисунок 3.6).

Кадр віддаленого запиту даних існує і у стандартному форматі і розширеному форматі. В обох випадках він складається із шести бітових полів: "початок кадру (повідомлення)" (Start of frame), "поле арбітражу" (Arbitration field), "поле керування " (Control field), "поле CRC" (CRC - field), "поле підтвердження" (ACK-field), "кінець кадру" (End of frame). На відміну від кадру даних, RTR біт кадру віддаленого запиту даних – одиничний. У цьому кадрі відсутнє поле даних. Значення коду довжини даних відповідає коду довжини даних кадру даних, який запитується. RTR біт вказує – чи є переданий кадр кадром даних чи кадром віддаленого запиту.

В таблиці 3.2 показаний вплив бітів RTR, SRR и IDE на тип повідомлення, що передається.

Таблиця 3.2 – Вплив бітів RTR, SRR та IDE на тип кадру (повідомлення)

№ п.п.	Значення біта			Тип кадра
	RTR в стандартному повідомленні (SRR в розширеному повідомленні)	IDE	RTR в розширеному повідомленні	
1	0	0	–	Стандартне повідомлення
2	1	1	0	Розширене повідомлення
3	1	0	–	Віддалений запит (стандартний)
4	1	1	1	Віддалений запит (розширений)

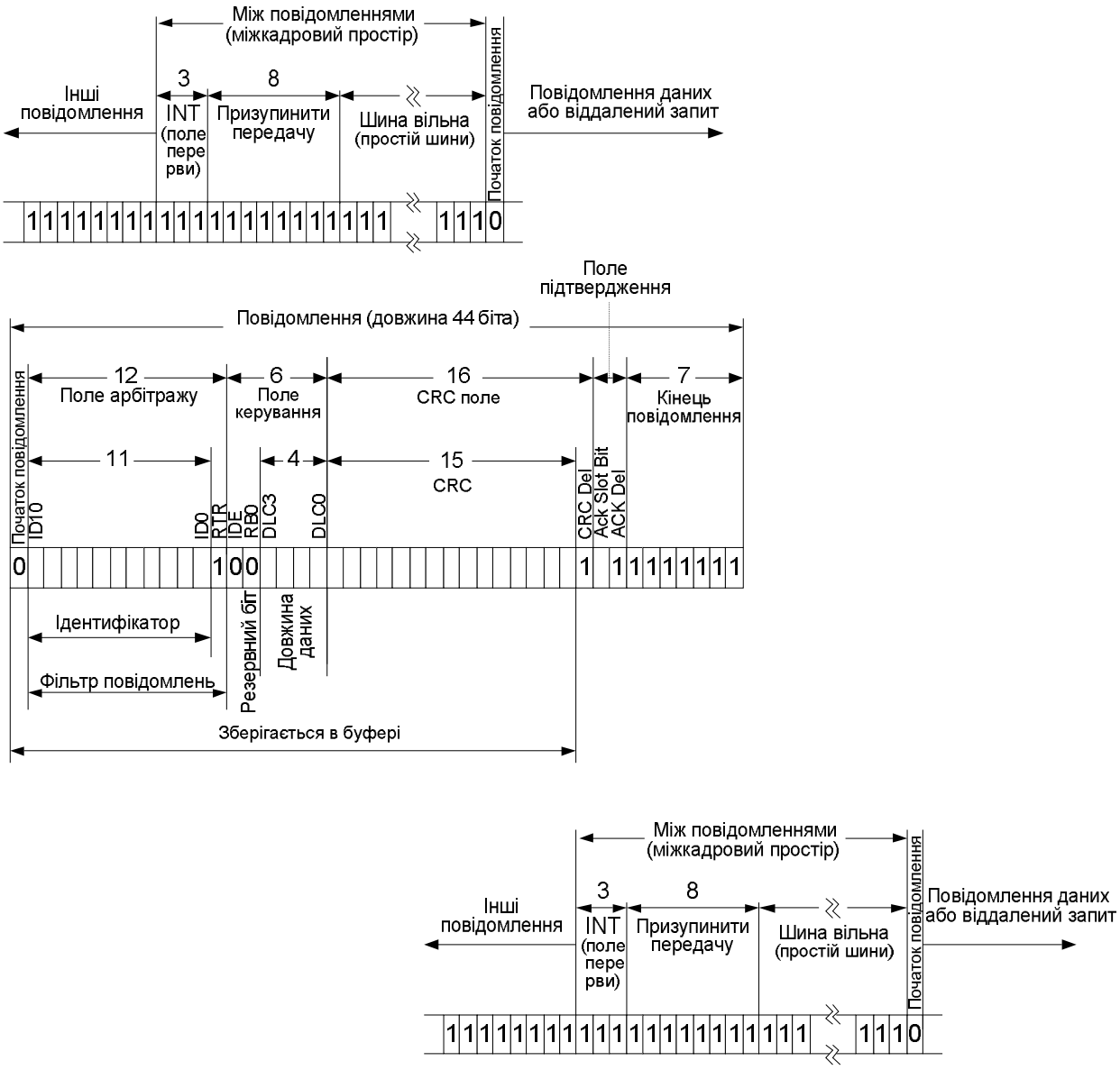


Рисунок 3.6 – Віддалений запит (стандартний)

3.3 Кадр помилки (Error frame)

Кадр (повідомлення) помилки (рисунок 3.7) складається із двох різних полів. Перше поле є суперпозицією прапорців помилки, отриманих від вузла, який передає кадр помилки, та інших вузлів. Наступне поле – роздільник помилки (Error Delimiter).

Для правильного закінчення кадру помилки, вузол у стані "пасивної помилки" може мати потребу в доступі до шини, принаймні, на 3 бітових

інтервали (якщо має місце локальна помилка приймача, що перебуває в стані "пасивної помилки"). Отже, шина не повинна бути завантажена на 100 %.

Прапорець помилки (Error flag)

Є два види прапорця помилки: прапорець активної помилки та прапорець пасивної помилки.

1. Прапорець активної помилки складається із шести послідовних нульових біт.
2. Прапорець пасивної помилки складається із шести послідовних одиничних біт, якщо вони не перезаписані нульовими бітами інших вузлів.

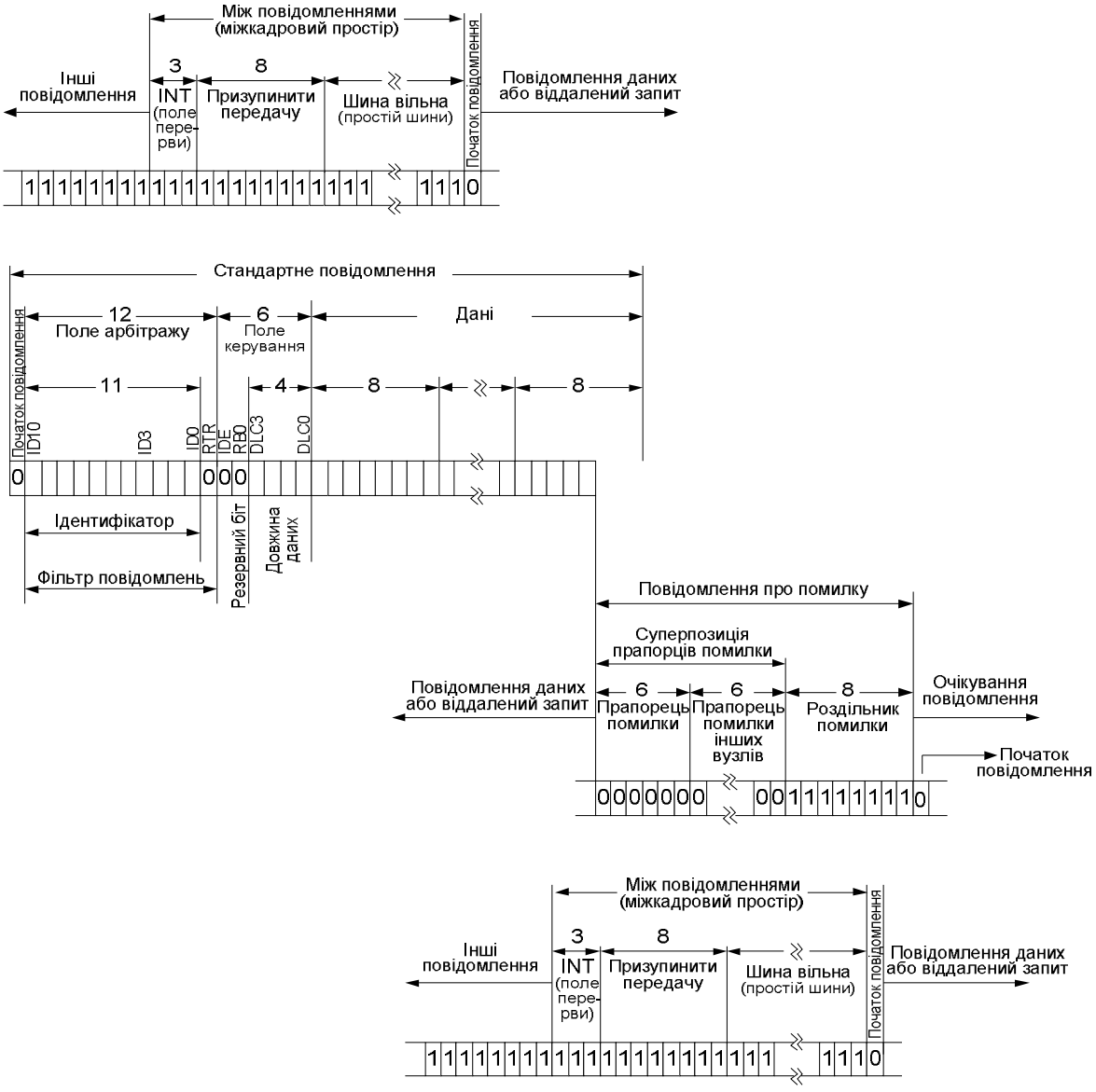


Рисунок 3.7 – Повідомлення про помилку

Вузол у стані "активної помилки", виявивши умову помилки, сповіщає про це передачею прапорця активної помилки. Форма прапорця помилки порушує закон заповнення біта (біт-стаффінг), який застосовується до всіх полів від поля "початок кадру" до роздільника CRC, або руйнує фіксовану форму поля підтвердження або поля "кінець кадру". Як наслідок, всі інші вузли виявляють умову помилки і у свою чергу починають передачу прапорця помилки. Таким чином, послідовність "домінантних" бітів, що фактично може з'явитися на шині, отримується суперпозицією різних прапорців помилки, які передані окремими вузлами. Сумарна довжина цієї послідовності змінюється від шести до дванадцяти біт. Вузол, у стані "пасивної помилки", виявивши умову помилки, робить спробу сповістити про це передачею прапорця пасивної помилки. Він очікує появи шести послідовних однакових одиничних біт, які починають прапорець пасивної помилки. Прапорець пасивної помилки завершується після виявлення цих 6 біт.

Роздільник помилки (Error Delimiter)

Роздільник помилки складається з восьми "рецесивних" бітів. Після передачі прапорця помилки кожен вузол посилає "рецесивні" біти та перевіряє шину, поки не виявляє "рецесивний" біт. Далі він починає передачу ще семи "рецесивних" бітів.

3.4 Кадр перевантаження (Overload Frame)

Кадр перевантаження містить два бітових поля: прапорець перевантаження та роздільник перевантаження (рисунок 3.8).

Є три види перевантаження, які приводять до передачі прапорця перевантаження:

- внутрішній стан приймача, який потребує затримки наступного кадру даних або кадру віддаленого запиту даних;
- виявлення "домінантного" біта при передачі першого та другого бітів поля перерви;

- якщо вузол виявляє "домінантний" біт на восьмому біті (останньому біті) роздільника помилки або роздільника перевантаження, це спричиняє передачу кадру перевантаження (а не кадру помилки). Лічильники помилок не будуть збільшені.

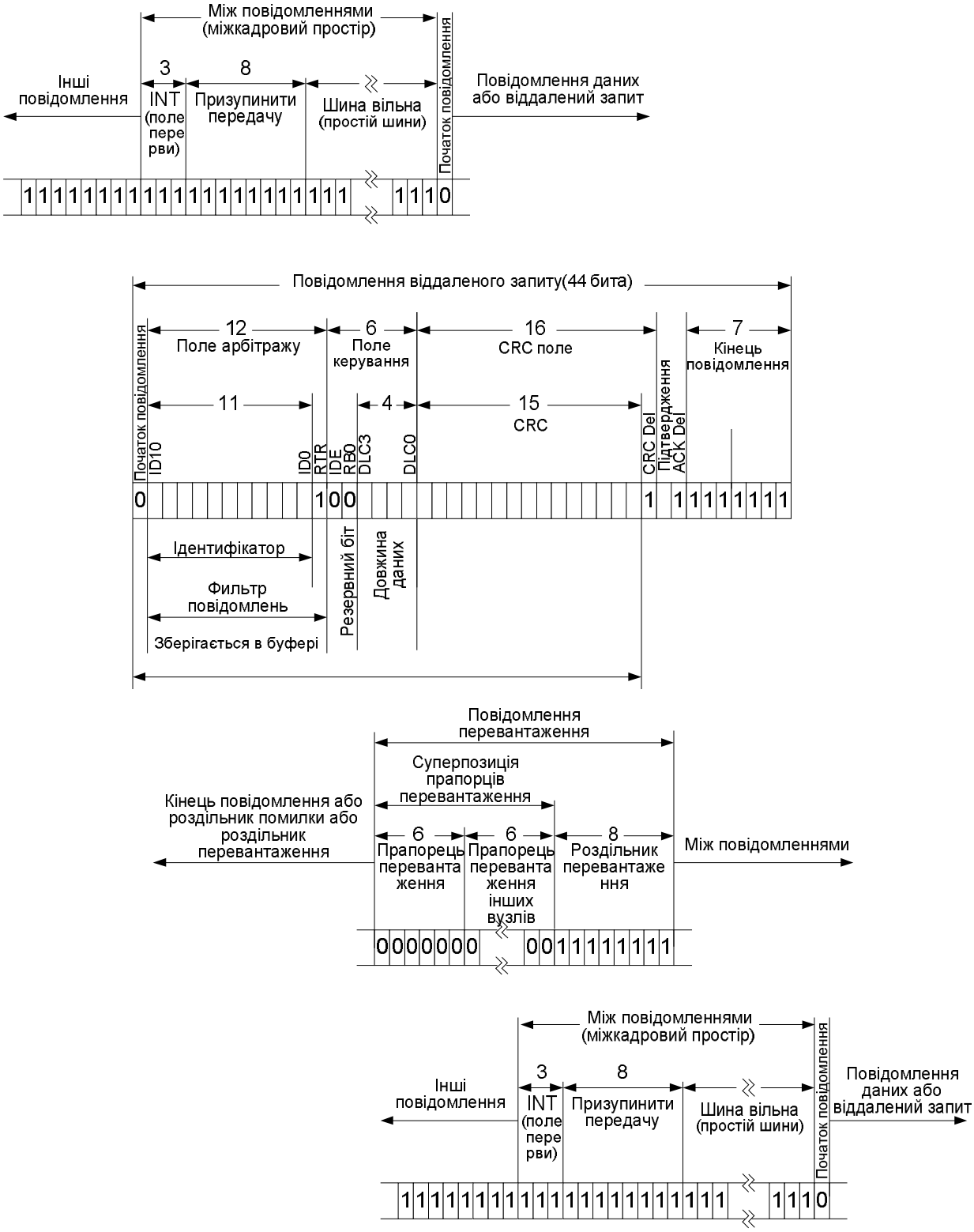


Рисунок 3.8 – Повідомлення перевантаження

Передачу кадру перевантаження, обумовленого 1-м видом перевантаження, дозволено починати в першому бітовому інтервалі передбаченого поля перерви, у той час як кадр перевантаження, обумовлений 2-м й 3-м видами перевантаження, починає передаватися на один біт пізніше виявлення "домінантного" біта. Для того щоб затримати наступний кадр даних або кадр віддаленого запиту даних, може бути згенеровано не більше ніж два кадри перевантаження підряд.

Прапорець перевантаження (Overload Flag)

Складається із шести "домінантних" бітів. Повна форма відповідає прапорцю активної помилки. Форма прапорця перевантаження порушує фіксовану форму поля перерви. Всі інші вузли також виявляють умову перевантаження і у свою чергу починають передачу прапорця перевантаження. Таким чином, послідовність "домінантних" бітів, що фактично може з'явитися на шині, отримується суперпозицією прапорців перевантаження, які передані окремими вузлами. Сумарна довжина цієї послідовності змінюється від шести до дванадцяти біт. У випадку, якщо виявлено "домінантний" біт під час 3-го біта перерви, то цей біт інтерпретується як початок кадру.

Роздільник перевантаження (Overload Delimiter)

Складається з восьми "рецесивних" біт. Роздільник перевантаження має таку ж форму, як і роздільник помилки. Після передачі прапорця перевантаження вузол контролює шину, поки не виявить перехід від "домінантного" біта до "рецесивного" біта. У цей момент часу кожен вузол закінчує передачу прапорця перевантаження, і всі вузли починають передачу ще 7 "рецесивних" біт.

3.5 Міжкадровий простір (Interframe Space)

Кадри даних і кадри віддаленого запиту даних відокремлюються від попередніх кадрів будь-якого типу (кадр даних, кадр віддаленого запиту даних, кадр помилки, кадр перезавантаження) бітовим полем, яке називається міжкадровим простором (рисунок 3.1, 3.2, 3.6, 3.7, 3.8). На відміну від них,

кадрам перевантаження й кадрам помилки не передує міжкадровий простір (кратні кадри перевантаження також не відокремлюються інтервалами).

Для вузла, що був джерелом попереднього повідомлення, міжкадровий простір містить бітові поля: перерва та простій шини. Для вузла в стані "пасивної помилки", що був джерелом попереднього повідомлення, міжкадровий простір містить додаткове поле «призупинити передачу». Це поле має бути між полями «перерва» і «простій» шини.

Поле перерви (Intermission)

Складається із трьох "рецесивних" біт. Протягом перерви єдина дія, яку можна виконати – повідомлення перевантаження й ніякий вузол не може почати передачу кадру даних або кадру віддаленого запиту даних.

Зауваження: якщо вузол має повідомлення, яке потрібно передати, і він виявляє "домінантний" біт у третьому біті перерви, це інтерпретується, як біт "початок кадру", і, з наступного біта, він починає передавати повідомлення, з першого біта його ідентифікатора, не передаючи попередньо, біт "початок кадру".

Простій шини (Bus Idle)

Період простою шини може мати довільну довжину. Шина вільна й будь-який вузол, якому потрібно що-небудь передати, може звертатися до шини. Повідомлення, що чекає передачі під час передачі іншого повідомлення, починає передаватися в першому біті після перерви. Виявлення "домінантного" біта на шині інтерпретується як "початок кадру".

Призупинення передачі (Suspend Transmission)

Після того, як вузол у стані "пасивної помилки" передав повідомлення, він посилає вісім "рецесивних" бітів після поля перерви, перед передачею наступного повідомлення або розпізнавання простою шини. Якщо тим часом починається передача, яка викликана іншим вузлом, вузол перетворюється в приймач цього повідомлення.

4 CAN-МОДУЛЬ МІКРОКОНТРОЛЕРА AT90CAN128

4.1 Загальна характеристика

CAN-модуль є контролером зв'язку, що підтримує протокол зв'язку CAN 2.0 A/B. Повнофункціональний CAN-модуль у складі AT90CAN128, апаратно підтримує протоколи CAN-2.0A і CAN-2.0B. Основні характеристики контролера зв’язку мікроконтролера AT90CAN128:

- стандартний і розширений типи повідомлень;
- довжина даних у повідомленні від 0 до 8 байт;
- програмована швидкість передачі інформації до 1 Мбіт/с;
- підтримка віддаленого запиту даних;
- 15 повних об'єктів повідомлень з окремими ідентифікаторами і масками;
- можливість указання пріоритету та аварійного припинення передачі;
- можливість пробудження з SLEEP-режиму;
- гнучка система переривань від CAN-модуля;
- низьке енергоспоживання в режимі SLEEP.

4.2 Структура CAN-модуля

CAN-модуль забезпечує повну апаратну підтримку фільтрації повідомлень та керування ними. Для кожного повідомлення, яке повинно бути передане або отримане, даний модуль містить один так званий об'єкт повідомлення MOb (рисунок 4.1). MOb – це дескриптор CAN-кадру. Він містить всю інформацію для роботи з CAN-кадром. Інакше кажучи, MOb був розроблений для опису CAN-кадру, як об'єкта.

Кожен MOb має свої власні поля, щоб керувати поточним режимом. Після перезавантаження мікроконтролера жодний з MOb не має вибраного режиму роботи. Перед активацією периферійних пристроїв, під’єднаних до модуля CAN, кожен MOb повинен бути налаштований відповідним чином на поточний режим

(виняток лише для дезактивованого режиму – біти CONMOB0,1 регістра CANCDMOB дорівнюють нулю). Можливі режими наведено у таблиці 4.1.

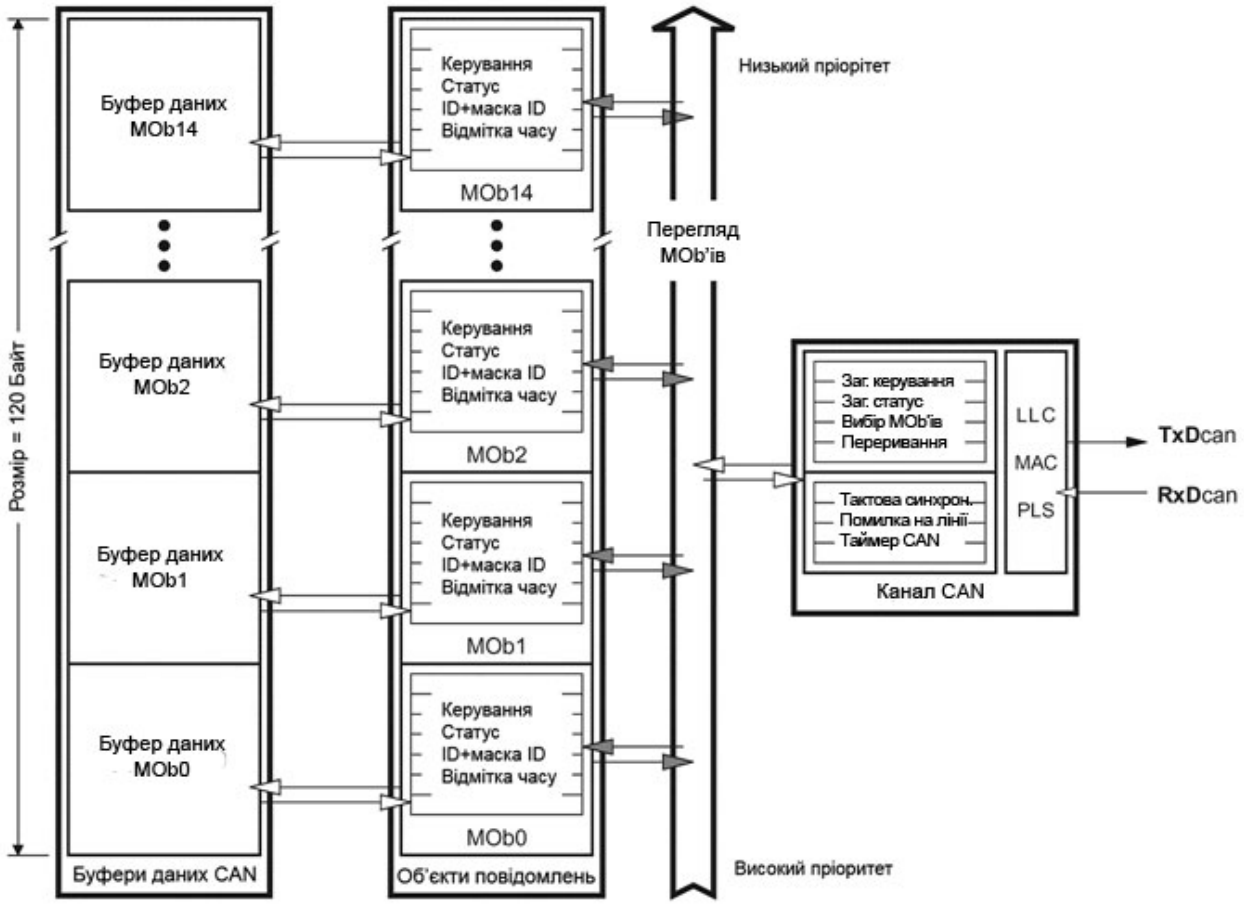


Рисунок 4.1 – Структура CAN-модуля

Таблиця 4.1 – Конфігурація об'єктів повідомлення (MOb)

Конфігурація MOb		Відповідь	RTR Tag	Поточний режим
CONMOB0	CONMOB1	Біт RPLV регістра CANCDMOB		
0	0	x	x	Дезактивований
0	1	x	0	Передача кадру даних
		x	1	Передача кадру віддаленого запиту
1	0	x	0	Прийом кадру даних
		0	1	Прийом кадру віддаленого запиту
		1		Прийом кадру віддаленого запиту та авт. відповідь
1	1	x	x	Прийом кадрів в буфер

4.3 Режими роботи CAN-контролера

CAN-контролер визначає, які повідомлення повинні передаватися, а які прийматися. Тільки тоді, коли ідентифікатор отриманого повідомлення збігається з одним із запрограмованих ідентифікаторів, дозволених до прийому повідомлень, об'єкт повідомлення зберігається і програма інформується перериванням. Також на повідомлення віддаленого запиту можуть автоматично надсилатися відповідні дані. Таким чином, значно знижується завантаження процесора в порівнянні з базовим рішенням CAN-контролера.

4.3.1 Дезактивований режим

У цьому режимі MOb залишається "вільним" (таблиця 4.1).

4.3.2 Передача кадру (повідомлення) або віддаленого запиту (Tx-конфігурація)

1. Перед відправкою повідомлення або кадру віддаленого запиту деякі поля MOb повинні бути ініціалізовані:
   - IDT (регістри CANIDT1…CANIDT4) – 11 бітний або 29 бітний ідентифікатор;
   - IDE (регістр CANCDMOB) – біт ознаки розширеного ідентифікатора;
   - RTRTAG (регістр CANIDT4) – біт ознаки віддаленого запиту на передачу;
   - DLC (регістр CANCDMOB) – розмір поля даних;
   - RBnTAG (регістр CANIDT4) – зарезервовані поля;
   - MSG (регістри CANMSG, CANPAGE) – поле даних.

2. MOb готовий відіслати кадр даних або віддаленого запиту, після встановлення Tx-конфігурації (біти CONMOB0,1 регістра CANCDMOB) (таблиця 4.1).

3. Далі CAN-контролер переглядає (сканує) всі MOb в Tx-конфігурації, знаходить MOb, що має найбільший пріоритет і намагається надіслати його. Слід зауважити, що можливе виникнення ситуації, коли декілька MOb очікують передачі. Наприклад:

- в наслідок втрати арбітражу шини або невдалої попередньої передачі;
- ініціювання передачі декількох наступних MOb до завершення передачі поточного;
- активування CAN-контролера за попереднім налаштуванням на передачу декількох MOb.

4. Коли передача завершена, встановлюється біт TXOK (регістр CANSTMOB) (при цьому можливе переривання).

5. Всі параметри і дані залишаються доступними в MOb до нової ініціалізації.

Схема алгоритму роботи CAN-модуля у цьому режимі наведена в додатку В.

4.3.3 Прийом кадру даних або віддаленого запиту (Rx-конфігурація)

1. Перед отриманням повідомлення деякі поля MOb повинні бути ініціалізовані:

- IDT – 11-бітний або 29-бітний ідентифікатор (регістри CANIDT1…CANIDT4);
- IDMSK – маска IDT (регістри CANIDM1…CANIDM4);
- IDE – біт ознаки розширеного ідентифікатора (регістр CANCDMOB);
- IDEMSK – маска розширеного ідентифікатора IDE (регістр CANIDM4);
- RTRTAG – біт ознаки віддаленого запиту на передачу (регістр CANIDT4);
- RTRMSK – біт маски RTRTAG (регістр CANIDM4);
- DLC – розмір поля даних (регістр CANCDMOB);
- RBnTAG – резервні поля (регістр CANIDT4).

2. MOb готовий прийняти кадр даних або віддаленого запиту, коли Rx-конфігурація MOb встановлена (біти CONMOB0,1 регістра CANCDMOB) (таблиця 4.1).

3. Коли ідентифікатор кадру отриманий з CAN-мережі, CAN-контролер переглядає всі MOb намагаючись знайти відповідний MOb з найбільшим пріоритетом.

4. IDT, IDE, DLC поля відповідного MOb оновлюються значеннями із прийнятого кадру.

5. Як тільки прийом закінчено, байти даних отриманого повідомлення зберігаються (не для кадру віддаленого запиту) в буфері даних відповідного MOb (рисунок 4.1) і біт RXOK встановлюється (при цьому можливе переривання).

6. Всі параметри і дані залишаються доступними в MOb до наступної ініціалізації.

Схема алгоритму роботи CAN-модуля у цьому режимі наведена в додатках Г та Д.

4.3.4 Автоматична відповідь

Відповідь (кадр даних) на кадр віддаленого запиту може бути автоматично відправлена після прийому очікуваного кадру віддаленого запиту, якщо біт RPLV регістра CANCDMOB встановлений у одиницю (таблиця 4.1).

1. Для цього режиму деякі поля MOb повинні бути ініціалізовані:
   - IDT (регістри CANIDT1…CANIDT4) – 11-бітний або 29-бітний ідентифікатор;
   - IDE (регістр CANCDMOB) – біт ознаки розширеного ідентифікатора;
   - RTRTAG (регістр CANIDT4) – біт ознаки віддаленого запиту на передачу;
   - DLC (регістр CANCDMOB) – розмір поля даних;
   - RBnTAG (регістр CANIDT4) – резервні поля;
   - MSG (регістри CANMSG, CANPAGE) – поле даних.

2. Коли кадр віддаленого запиту співпадає з MOb, біти RTRTAG і RPLV автоматично скидаються. Так як CAN-буфер даних не буде заповнений даними з вхідного кадру, MOb готовий відразу відповісти без додаткових налаштувань. IDT, IDE, DLC та деякі інші поля прийнятого кадру віддаленого запиту використовуються для автоматичної відповіді.

3. Коли передача відповіді завершена, встановлюється біт TXOK (при цьому можливе переривання).

4. Всі параметри і дані залишаються доступними в MOb до наступної ініціалізації.

Схема алгоритму роботи CAN-модуля у цьому режимі наведена в додатку Г.

4.3.5 Режим прийому кадрів в буфер

Цей режим використовується для отримання мультикадрів. Пріоритет між MOb дозволяє керувати вхідними кадрами такого типу. Лише один набір MOb (включаючи непослідовні MOb) настроюється у цьому режимі. Біт завершення прийому кадрів в буфер (чи переривання) BXOK буде встановлено, тільки коли всі MOb набору отримають, належні кадри.

1. MOb в режимі прийому кадрів в буфер повинні бути ініціалізовані як MOb в Rx-конфігурації (див. 4.3.3).

2. MOb готові отримувати кадри даних (або віддаленого запиту), коли їх відповідна конфігурація встановлена (таблиця 4.1).

3. Коли ідентифікатор кадру отриманий з CAN-мережі, CAN-контролер перебирає всі MOb намагаючись знайти відповідний MOb з найбільшим пріоритетом.

4. IDT, IDE, DLC відповідного MOb оновлюються вхідними значеннями (кадру).

5. Як тільки прийом закінчено, байти даних отриманого повідомлення зберігаються (не для кадру віддаленого запиту) в буфері даних відповідного MOb і біт RXOK встановлюється (при цьому можливе переривання).

6. Коли прийом для останнього MOb з набору завершений, встановлюється біт завершення прийому кадрів в буфер BXOK (регістр CANGIT), при цьому можливе переривання. Біт BXOK може бути очищений тоді, коли всі біти CONMOB0,1 регістра CANCDMOB для всіх MOb набору були перезавантажені.

7. Всі параметри і дані залишаються доступними в MOb до наступної ініціалізації.

Схема алгоритму роботи CAN-модуля у цьому режимі наведена в додатку Е.

4.4 Структура блоку фільтрації

У блоці фільтрації (рисунок 4.2) ідентифікатор вхідного повідомлення побітно підсумовується за модулем два з регістром фільтру, який складається з 13 (32) бітів регістрів CANIDT1, CANIDT2, CANIDT3, CANIDT4 (залежно від формату кадру версія 2.0 A/B) і біта IDE регістра CANCDMOB. Після цього результат інвертується і на нього накладається маска з регістрів масок ідентифікатора CANIDM1, CANIDM2, CANIDM3, CANIDM4. Якщо після цих операцій ми отримуємо послідовність одиничних біт, то повідомлення записується у відповідний об'єкт повідомлення (MOb).

Маска застосовується для того, щоб вказати які біти фільтра будуть використовуватися для перевірки ідентифікатора вхідного повідомлення. Якщо біт маски дорівнює нулю, то відповідний біт ідентифікатора повідомлення буде прийнятий, незалежно від значення біта фільтра.

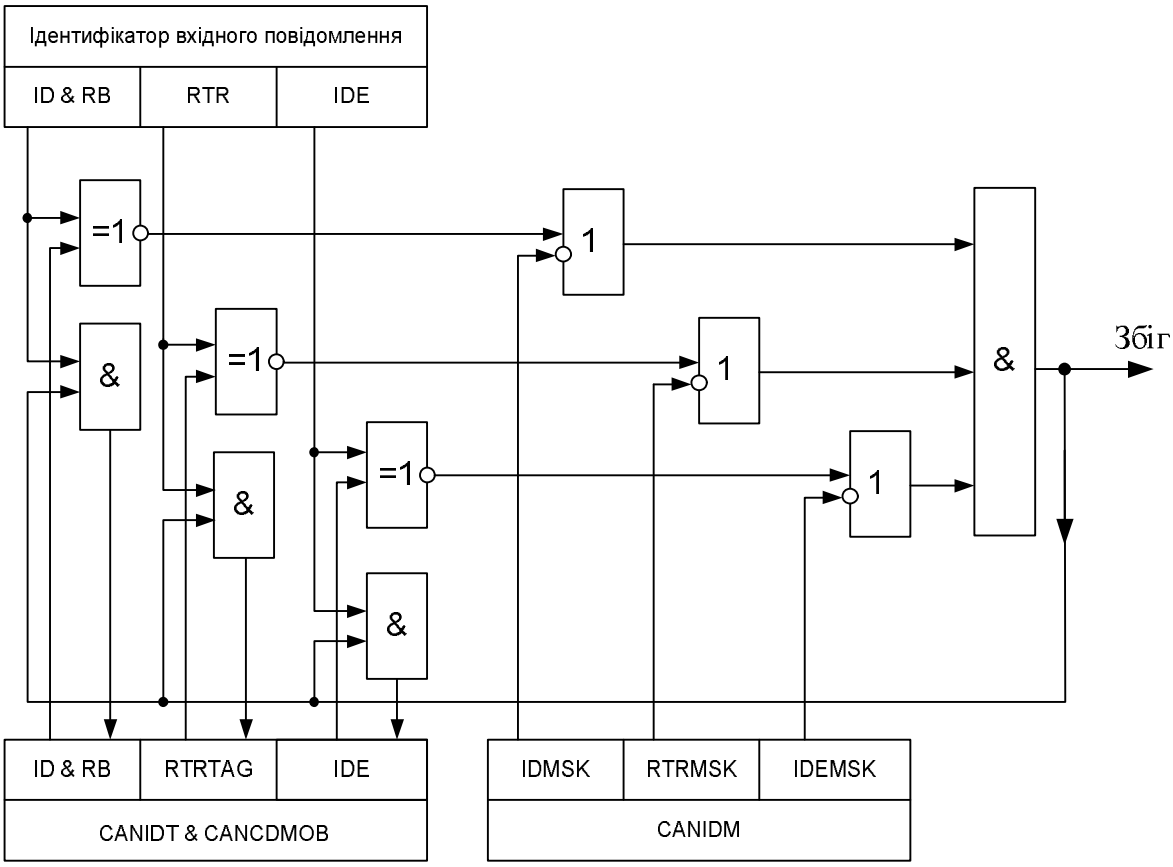


Рисунок 4.2– Структура блоку фільтрації повідомлень

Приклад для стандартних повідомлень:

Прийом тільки ID = 0x317 (повна фільтрація):

ID MSK = 111 1111 1111 b;

ID TAG = 011 0001 0111 b.

Прийом ID від 0x310 до 0x317 (часткова фільтрація):

ID MSK = 111 1111 1000 b;

ID TAG = 011 0001 0xxx b.

4.5 Структура переривань від CAN-модуля

AVR забезпечує кілька різних джерел переривань. Ці переривання мають окремий програмний вектор в пам'яті програм (підпрограма обробки CAN-переривання має адресу 0x0024, а обробки переривання від переповнення CAN-

таймера – адресу 0x0026). Всі переривання визначаються індивідуальними бітами дозволу, які повинні бути встановлені у 1 разом з бітом глобального дозволу переривань у регістрі статусу, щоб дозволити переривання. Залежно від значення програмного лічильника, переривання можна автоматично заборонити, коли комірки захисту (блокування завантаження) BLB02 або BLB12 запрограмовані (скинуті). Ця опція покращує безпеку програмного забезпечення [2, 3].

У CAN-модулі мікроконтролера AT90CAN128 можлива обробка переривань як для всіх повідомлень, так і для кожного конкретного об'єкта повідомлень (MOb). Структура переривань представлена на рисунку 4.3.

Існують такі переривання:

- успішне отримання повідомлення;
- успішне відправлення повідомлення;
- виявлено помилку;
- заповнення буфера зберігання кадрів;
- установка стану «відключення від шини»;
- переповнення лічильника CAN-таймера.

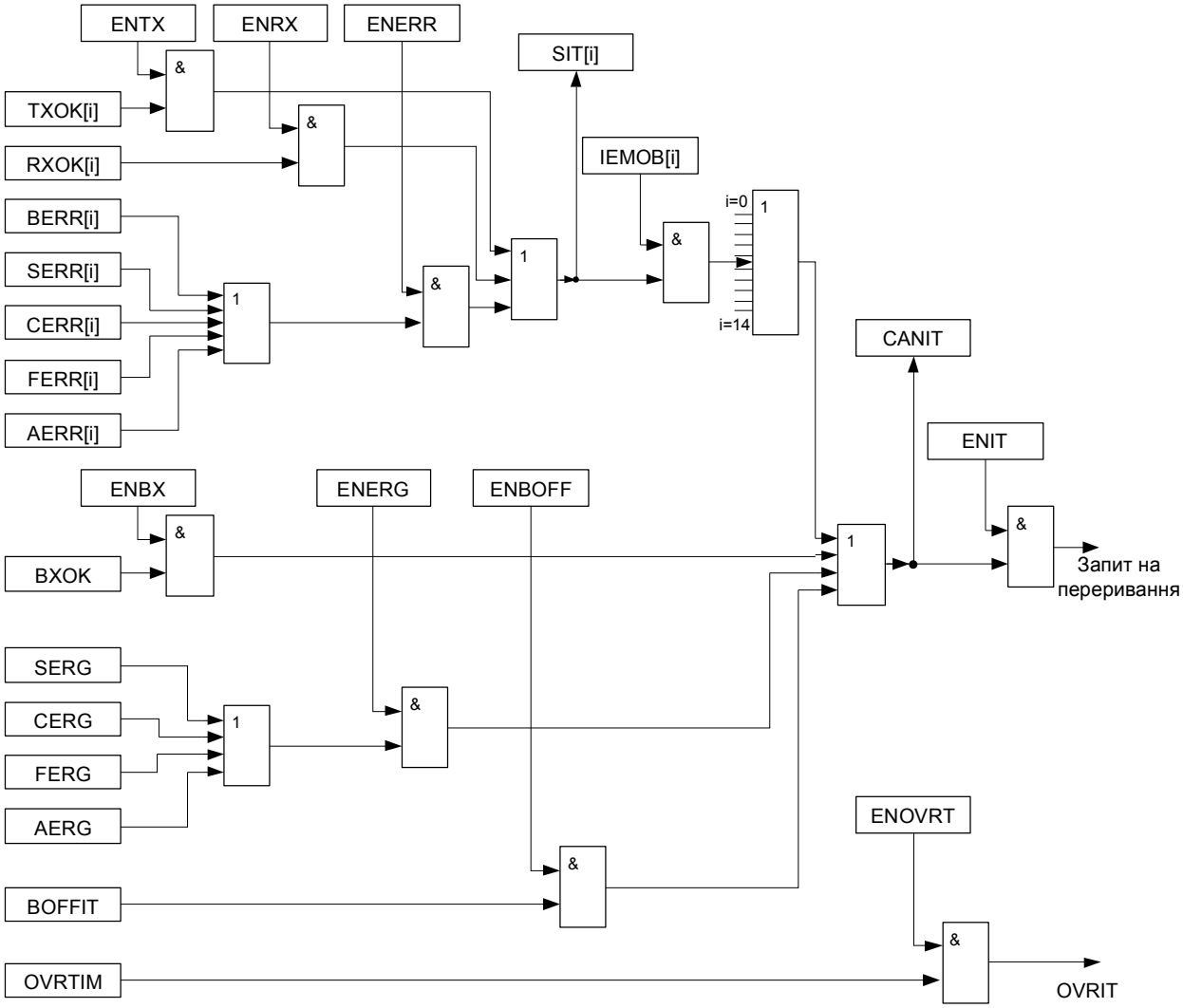


Рисунок 4.3 – Структура переривань

Розпишемо умовні позначення на рисунку 4.3.

CANSTMOB - регістр статусу об'єкта повідомлення:

- CANSTMOB.6 (TXOK) – успішна передача;
- CANSTMOB.5 (RXOK) – успішний прийом;
- CANSTMOB.4 (BERR) – помилка біта;
- CANSTMOB.3 (SERR) – помилка наповнення;
- CANSTMOB.2 (CERR) – помилка CRC;
- CANSTMOB.1 (FERR) – помилка форми;
- CANSTMOB.0 (AERR) – помилка підтвердження.

CANGIT – загальний регістр переривань:

- CANGIT.7 (CANIT) – загальний прапорець переривань (відображає наявність будь-якого переривання крім OVRTIM);
- CANGIT.6 (BOFFIT) – прапорець переривання відключення від шини;
- CANGIT.5 (OVRTIM) – прапорець переповнення лічильника CAN-таймера;
- CANGIT.4 (BXOK) – прапорець отримання відповідного набору кадрів в буфер;
- CANGIT.3 (SERG) – прапорець помилки наповнення;
- CANGIT.2 (CERG) – прапорець CRC-помилки;
- CANGIT.1 (FERG) – прапорець помилки форми;
- CANGIT.0 (AERG) – прапорець помилки підтвердження.

CANGIE – загальний регістр дозволу переривань:

- CANGIE.7 (ENIT) – дозвіл всіх переривань (крім переривання від переповнення CAN-таймера);
- CANGIE.6 (ENBOFF) – дозвіл переривання від’єднання від шини;
- CANGIE.5 (ENRX) – дозвіл переривання прийому;
- CANGIE4 (ENTX) – дозвіл переривання передачі;
- CANGIE.3 (ENERR) – дозвіл переривань від “помилки MOb”;
- CANGIE.2 (ENBX) – дозвіл переривання від отримання відповідного набору кадрів в буфер (BXOK);
- CANGIE.1 (ENERG) – дозвіл переривань від загальних помилок
- CANGIE.0 (ENOVRT) – дозвіл переривання від переповнення лічильника CAN-таймера.

CANIE1 і CANIE2 – регістри дозволу переривань від MOb:

- відповідні біти (IEMOB[i]), які встановлені в 1, показують, для яких MOb дозволені переривання (усього біт в регістрах 15, як і об'єктів повідомлення).

При виникненні загального переривання відповідний біт встановлюється в регістрі CANGIT. Якщо відповідний йому "дозволяючий біт" в регістрі CANGIE встановлений і переривання дозволені (регістр CANGIE), то встановлюється загальний прапорець переривань CANGIT.7 (CANIT), регістр CANGIT.

При виникненні переривання від MOb відповідний біт встановлюється в регістрі CANSTMOB і записується в CANIT тільки при встановленому "дозволяючому біті" для даного переривання в регістрі CANGIE (CANGIE.3, CANGIE.4, CANGIE.5) та встановленому біті, що дозволяє переривання від даного MOb, в регістрах CANIE1 або CANIE2.

Для правильного визначення переривань від MOb відповідні біти регістра CANSTMOB спочатку повинні бути очищені. Для правильного визначення загальних переривань відповідні біти регістра CANGIT повинні бути також спочатку очищені.

4.6 Структура блоку CAN-таймера

Структура блоку таймера CAN-модуля наведена на рисунку 4.4. Нижче описане призначення основних вузлів блоку.

Дільник частоти (CANTCON).

8-ми бітний дільник ініціалізується регістром CANTCON. Він отримує сигнал з частотою f_{CLKIO}, поділеною на 8, та забезпечує надходження сигналу з частотою $f_{CLKCANTIM}$ на CAN-таймер (CANTIM), якщо CAN-контролер активований (біт ENFG регістра CANGSTA встановлений у 1):

$T_{CLKCANTIM} = T_{CLKIO}$ x 8 x (CANTCON [7:0] + 1), де T_{CLKIO} – період тактової частоти підсистеми введення/виведення МК-ра [2, 3].

16-бітний таймер (CANTIM).

Цей таймер починає рахувати з 0x0000, коли CAN-контролер переходить в активний стан (ENFG біт = 1).

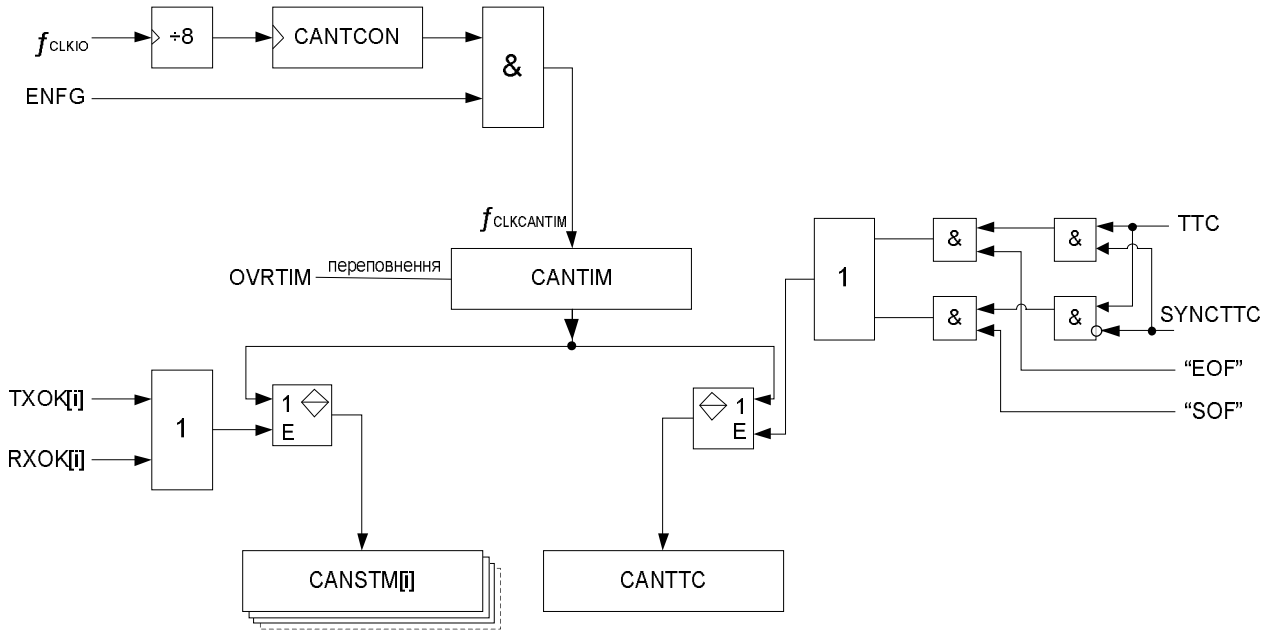


Рисунок 4.4 – Структура блоку CAN-таймера

Коли відбувається переповнення лічильника таймера і скидання зі значення 0xFFFF на 0x0000, генерується переривання (OVRTIM = 1).

Режим TTC.

Передбачено два режими синхронізації для TTC (TTC біт):

- синхронізація за початком кадру (SYNCTTC = 0, SOF = 1);
- синхронізація за кінцем кадру (SYNCTTC = 1, EOF = 1).

Цей режим використовується програмним забезпеченням, якщо в мережі використовується взаємодія по стандарту TTCAN (Time-Triggered CAN — CAN, керований часом). Цей стандарт дозволяє організувати взаємодію вузлів в мережі таким чином, що зникає невизначеність у часі передачі повідомлень (яка існує у CAN — наприклад, повідомлення з низьким пріоритетом може бути надіслано не з першого разу через втрату арбітражу вузлом, що намагається його передати і завчасно невідомо, чи це трапиться) та підвищується ефективність використання пропускної здатності шини. У цьому режимі кожен вузол передає повідомлення у певний виділений для нього часовий інтервал, інші вузли в цей час не намагатимуться передавати дані. Це дозволяє уникнути колізій і втрати арбітражу. Також у цьому режимі передавач не намагається автоматично удруге

передати повідомлення у разі виникнення помилки (оскільки тоді він буде використовувати не свій часовий інтервал, що у цьому режимі неприпустимо).

Щоб контролер міг брати участь у роботі CAN-шини, де використовується режим TTCAN, необхідно додатково програмно реалізовувати відповідні алгоритми взаємодії [13].

Часові позначки повідомлень.

У CAN-модулі відбувається збереження значення таймера в об'єкті повідомлення (MOb), який отримує або відсилає кадр. Всі керовані об'єкти повідомлень отримують позначки часу. Це відбувається за подіями RXOK і TXOK (регістр CANSTMOB (успішний прийом, успішна передача)).

Якщо ви встановили режим TTC (біт TTC = 1) і SYNCTTC = 0, то застосовується синхронізація за початком кадру ("SOF" – початок кадру (Start of Frame)) і при виявленні відповідного поля (яке складається з одного нульового біта) значення з регістра таймера CANTIM записується в регістр CANTTC.

Якщо ви встановили режим TTC (біт TTC = 1) і SYNCTTC = 1, то застосовується синхронізація за кінцем кадру ("EOF" – кінець кадру (End of Frame)) і при виявленні останнього одиничного біта поля "кінець кадру" (яке складається з 7 одиничних біт) значення з регістра таймера CANTIM записується в регістр CANTTC.

4.7 Організація регістрів CAN-модуля

Склад регістрів CAN-модуля, наведений на рисунку 4.6. Ці регістри поділяються на дві групи: загальні регістри та регістри MOb.

4.7.1 Загальні регістри CAN-модуля

Для програмування CAN-модуля використовують загальні регістри, які описані нижче.

CANGCON – регістр загального керування (рисунок 4.5)

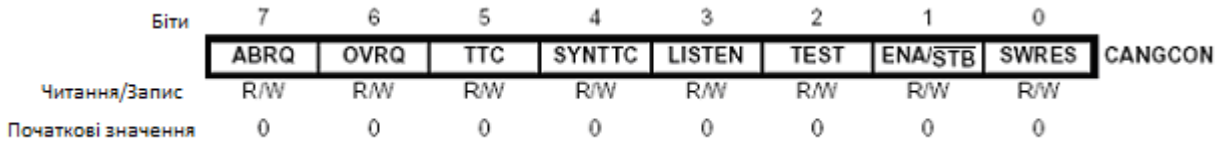


Рисунок 4.5 – Регістр CANGCON

1) Bit 7 – ABRQ: Запит аварійного припинення обміну

Цей біт не скидається автоматично:

- 0 – немає запиту;
- 1 – запит аварійного припинення обміну: скидання бітів CONMOB регістра CANCDMOB для кожного об'єкта повідомлення. Заплановані передачі негайно заморожуються, а поточна передача коректно припиняється з установкою відповідних прапорців статусу.

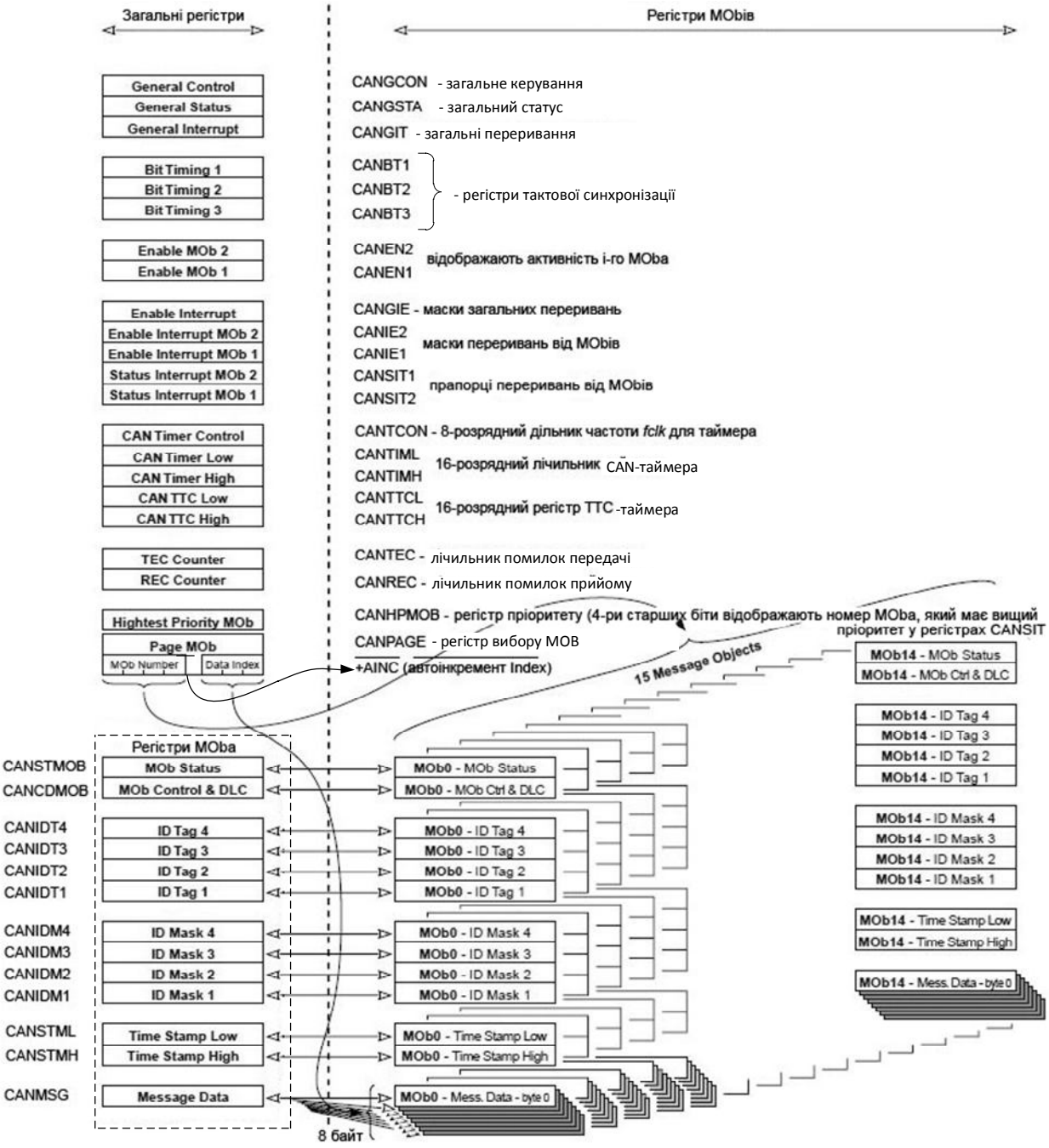


Рисунок 4.6 – Організація регістрів

2) Bit 6 – OVRQ: Запит кадру перевантаження

Цей біт не скидається автоматично:

- 0 – немає запиту;
- 1 – запит кадру перевантаження: надсилається кадр перевантаження після наступного отриманого повідомлення. Завершення передачі кадру перевантаження (рисунок 3.8) може бути відстежене за допомогою спостереження прапорця OVFG в регістрі CANGSTA.

3) Bit 5 – TTC: Режим TTC

У цьому режимі кадр надсилається один раз, навіть якщо відбулася помилка:

- 0 – TTC режим вимкнено;
- 1 – режим TTC увімкнено.

4) Bit 4 – SYNTTC: Тип синхронізації TTC

Цей біт використовується тільки в режимі TTC:

- 0 – таймер TTC (CANTTC) перезаписується значенням CANTIM (рисунок 4.4) при надходженні сигнала SOF (початок кадру);
- 1 – таймер TTC перезаписується значенням CANTIM (рисунок 4.4) при надходженні біта EOF (кінець кадру).

5) Bit 3 – LISTEN: Режим прослуховування

У цьому режимі CAN-контролер прозорий для мережі: забезпечує рецесивний рівень на виході, не відключає вхід RXDCAN, заморожує лічильники помилок:

- 0 – дезактивовано;
- 1 – активовано.

6) Bit 2 – TEST: Режим тестування:

- 0 – дезактивовано;
- 1 – режим тестування: призначений для тестування на виробництві, але не для використання користувачем.

7) Bit 1 – ENA / $\overline{\text{STB}}$: Вибір режиму: активний/очікування

Оскільки цей біт – керуючий і не діє негайно, дійсний стан контролера можна подивитися за бітом ENFG в регістрі CANGSTA:

- 0 – режим очікування: поточна передача нормально завершується і CAN-канал заморожується (біти CONMOB0,1 регістрів CONCDMOB для всіх об'єктів повідомлень не змінюються). Передавач постійно передає рецесивний рівень. У цьому режимі приймач неактивний, але всі регістри залишаються доступними для процесора;
- 1 – активний режим: канал CAN переходить в активний режим, після того як 11 поспіль рецесивних біт буде прийнято.

8) Bit 0 – SWRES: Запит програмного скидання

Цей біт скидає тільки CAN-контролер:

- 0 – немає скидання;
- 1 – скидання (при описі кожного регістра модуля на відповідних рисунках наведені початкові значення кожного біта після “скидання”).

CANGSTA – загальний регістр статусу

Біти	7	6	5	4	3	2	1	0	
	-	OVFG	-	TXBSY	RXBSY	ENFG	BOFF	ERRP	CANGSTA
Читання/Запис	-	R	-	R	R	R	R	R	
Початкові значення	-	0	-	0	0	0	0	0	

Рисунок 4.7 – Регістр CANGSTA

1) Bit 7 – Зарезервований біт

Цей біт зарезервований для майбутнього використання.

2) Bit 6 – OVFG: Overload Frame Flag (прапорець передачі кадру перевантаження)

Цей прапорець не генерує переривання:

- 0 – немає передачі кадру перевантаження;
- 1 – кадр перевантаження: встановлюється апаратно після передачі кадру перевантаження.

3) Bit 5 – Зарезервований біт

Цей біт зарезервований для майбутнього використання.

4) Bit 4 – TXBSY: Зайнятість передавача

Цей прапорець не генерує переривання:

- 0 – передавач вільний;
- 1 – передавач зайнятий: встановлюється апаратно під час відсилання кадрів або установки поля підтвердження (ACK). Також встановлюється під час посилки поля IFS (міжкадровий проміжок).

5) Bit 3 – RXBSY: Зайнятість приймача:

- 0 – приймач вільний;
- 1 – приймач зайнятий: встановлюється апаратно під час прийому кадру

6) Bit 2 – ENFG: Прапорець активності CAN-контролера

Цей прапорець не генерує переривання:

- 0 – CAN-контролер неактивний: тому що команда активації (деактивації) контролера не виконується негайно, в цьому біті зберігається реальний стан контролера;
- 1 – CAN-контролер активний (для переведення в активний режим біт ENA/$\overline{\text{STB}}$ регістра CANGCON треба встановити в одиницю).

7) Bit 1 – BOFF: Режим відключення від шини

BOFF містить інформацію про стан каналу CAN. Тільки вхід в стан від'єднання від шини встановлює прапорець BOFFIT регістра CANGIT, що може викликати переривання:

- 0 – нормальний режим роботи (режими активної або пасивної помилки);
- 1 – режим від'єднання від шини.

8) Bit 0 – ERRP: Режим пасивної помилки

ERRP містить інформацію про стан каналу CAN. Цей прапорець не генерує переривання:

- 0 –режим відсутності пасивної помилки;
- 1 – режим пасивної помилки.

CANGIT – загальний регістр переривань

Біти	7	6	5	4	3	2	1	0	
	CANIT	BOFFIT	OVRTIM	BXOK	SERG	CERG	FERG	AERG	CANGIT$_A$
Читання/Запис	R	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Початкові значення	0	0	0	0	0	0	0	0	

Рисунок 4.8 – Регістр CANGIT

1) Bit 7 – CANIT: Загальний прапорець переривань

Біт тільки для читання:

- 0 – немає переривання;

- 1 – CAN-переривання: відображає наявність будь–якого переривання крім OVRTIM (рисунок 4.3). Може використовуватися для реалізації механізму опитування окремих прапорців переривань.

2) Bit 6 – BOFFIT: Прапорець переривання відключення від шини

Запис логічної одиниці скидає прапор. Прапорець BOFFIT встановлюється тільки тоді, коли CAN переходить в режим відключення від шини з режиму пасивної помилки:

- 0 – немає переривання;
- 1 – відбувся перехід в стан відключення від шини.

3) Bit 5 – OVRTIM: Переповнення таймера

Запис логічної одиниці скидає прапорець. Прапорець також скидається сам при виконанні обробника переривання:

- 0 – немає переривання;
- 1 – переповнення таймера CAN: встановлюється при скиданні таймера CANTIM зі значення 0xFFFF в 0x0000 (рисунок 4.4).

4) Bit 4 – BXOK: Переривання отримання потрібного набору кадрів в буфер

Запис логічної одиниці скидає прапорець. Прапорець BXOK може бути очищений тільки після того, як всі поля CONMOB регістра CANCDMOB для поточного об'єкта повідомлення були перезавантажені:

- 0 – немає переривання;
- 1 – отримання пакета: встановлюється, коли отримання потрібного набору кадрів в буфер завершено.

5) Bit 3 – SERG: Помилка наповнення

Запис логічної одиниці скидає прапорець:

- 0 – немає переривання;
- 1 – помилка наповнення: виявлення більше ніж 5 послідовних однакових біт.

6) Bit 2 – CERG: CRC-помилка

Запис логічної одиниці скидає прапорець:

- 0 – немає переривання;

- 1 – CRC-помилка.

7) Bit 1 – FERG: Помилка форми

Запис логічної одиниці скидає прапорець:

- 0 – немає переривання;
- 1 – помилка форми: одне або більше порушень форми кадру у роздільниках CRC, ACK або EOF.

8) Bit 0 – AERG: Помилка підтвердження

Запис логічної одиниці скидає прапорець:

- 0 – немає переривання;
- 1 – помилка підтвердження: не виявлено домінантний біт у полі підтвердження.

CANGIE – загальний регістр дозволу переривань

Біти	7	6	5	4	3	2	1	0	
	ENIT	ENBOFF	ENRX	ENTX	ENERR	ENBX	ENERG	ENOVRT	CANGIE A
Читання/Запис	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Початкові значення	0	0	0	0	0	0	0	0	

Рисунок 4.9 – Регістр CANGIE

1) Bit 7 – ENIT: Дозвіл всіх переривань (крім переривання від переповнення таймера):
   - 0 – переривання заборонені;
   - 1 – переривання від встановлення прапорця CANIT (регістр CANGIT) дозволені.
2) Bit 6 – ENBOFF: Дозвіл переривання відключення від шини:
   - 0 – переривання заборонене;
   - 1 – переривання від встановлення прапорця BOFFIT (регістр CANGIT) дозволене.
3) Bit 5 – ENRX: Дозвіл переривання завершення прийому:
   - 0 – переривання заборонене;
   - 1 – переривання про завершення прийому дозволене.

4) Bit 4 – ENTX: Дозвіл переривання завершення передачі:
   - 0 – переривання заборонене;
   - 1 – переривання про завершення передачі дозволене.
5) Bit 3 – ENERR: Дозвіл переривань від помилки MOb:
   - 0 – переривання заборонене;
   - 1 – переривання від «помилки об’єкта повідомлення» дозволене.
6) Bit 2 – ENBX: Дозвіл переривання від отримання потрібного набору кадрів у буфер (BXOK=1):
   - 0 – переривання заборонене;
   - 1 – BXOK переривання дозволене.
7) Bit 1 – ENERG: Дозвіл переривань від загальних помилок:
   - 0 – переривання заборонене;
   - 1 – переривання від «загальних помилок» дозволене.
8) Bit 0 – ENOVRT: Дозвіл переривання від переповнення таймера:
   - 0 – переривання заборонене;
   - 1 – переривання від переповнення OVRTIM дозволене.

CANEN2 і CANEN1 – регістри дозволу доступу до MOb

Біти	7	6	5	4	3	2	1	0	
	ENMOB7	ENMOB6	ENMOB5	ENMOB4	ENMOB3	ENMOB2	ENMOB1	ENMOB0	CANEN2
	-	ENMOB14	ENMOB13	ENMOB12	ENMOB11	ENMOB10	ENMOB9	ENMOB8	CANEN1
Біти	15	14	13	12	11	10	9	8	
Читання/Запис	R	R	R	R	R	R	R	R	
Початкові значення	0	0	0	0	0	0	0	0	
Читання/Запис	-	R	R	R	R	R	R	R	
Початкові значення	-	0	0	0	0	0	0	0	

Рисунок 4.10 – Регістри CANEN2 і CANEN1

1) Bits 14:0 –ENMOB[14: 0]: Дозвіл доступу до MOb

Ці біти визначають доступність об'єктів повідомлень.

Біт встановлюється в одиницю, якщо об'єкт повідомлення активний, тобто біти CONMOB0,1 регістра CANCDMOB[7:6] не дорівнюють 00. При установці TXOK або RXOK в одиницю (також TXOK після автоматичні відповіді на віддалений запит), відповідний біт ENMOB скидається. ENMOB також встановлюється в нуль, переводячи об'єкт повідомлення в неактивний стан при аварійному завершенні передачі або при переході в режим очікування:

- 0 – об'єкт повідомлення неактивний: об'єкт повідомлення доступний для передачі або прийому;
- 1 – об'єкт повідомлення активний: об'єкт в даний момент використовується.

2) Bit 15 – Зарезервований біт

Цей біт зарезервований для майбутнього використання.

CANIE2 і CANIE1 –регістри дозволу переривань від MOb

Біти	7	6	5	4	3	2	1	0	
	IEMOB7	IEMOB6	IEMOB5	IEMOB4	IEMOB3	IEMOB2	IEMOB1	IEMOB0	CANIE2
	-	IEMOB14	IEMOB13	IEMOB12	IEMOB11	IEMOB10	IEMOB9	IEMOB8	CANIE1
Біти	15	14	13	12	11	10	9	8	
Читання/Запис	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Початкові значення	0	0	0	0	0	0	0	0	
Читання/Запис	-	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Початкові значення	-	0	0	0	0	0	0	0	

Рисунок 4.11 – Регістри CANIE2 і CANIE1

1) Bits 14:0 – IEMOB[14:0] : Дозвіл переривань від відповідного MOb:

- 0 – переривання заборонене;
- 1 – переривання для відповідного об'єкта повідомлення дозволене.

Приклад: CANIE2 = 0000 1100b: дозволяє переривання для MOb 2 і 3.

2) Bit 15 – Зарезервований біт

Цей біт зарезервований для майбутнього використання. Для сумісності з майбутніми пристроями повинен скидатися в 0.

CANSIT2 і CANSIT1 – регістри статусу переривань від MOb

1) Bits 14:0 – SIT14:0] : Статус переривань від MOb:
   - 0 – переривання відсутнє;
   - 1 – переривання у відповідному MOb присутнє.
2) Bit 15 – Зарезервований біт

Цей біт зарезервований для майбутнього використання. Для сумісності з майбутніми пристроями повинен скидатися в 0.

Біти	7	6	5	4	3	2	1	0	
	SIT7	SIT6	SIT5	SIT4	SIT3	SIT2	SIT1	SIT0	CANSIT2
	-	SIT14	SIT13	SIT12	SIT11	SIT10	SIT9	SIT8	CANSIT1
Біти	15	14	13	12	11	10	9	8	
Читання/Запис	R	R	R	R	R	R	R	R	
Початкові значення	0	0	0	0	0	0	0	0	
Читання/Запис	-	R	R	R	R	R	R	R	
Початкові значення	-	0	0	0	0	0	0	0	

Рисунок 4.12 – Регістри CANSIT2 і CANSIT1

CANBT1 –регістр тактової синхронізації 1

Біти	7	6	5	4	3	2	1	0	
	-	BRP5	BRP4	BRP3	BRP2	BRP1	BRP0	-	CANBT1
Читання/Запис	-	R/W	R/W	R/W	R/W	R/W	R/W	-	
Початкові значення	-	0	0	0	0	0	0	-	

Рисунок 4.13 – Регістр CANBT1

1) Bit 7 – Зарезервований біт

   Цей біт зарезервований для майбутнього використання. Для сумісності з майбутніми пристроями повинен скидатися в 0.
2) Bit 6:1 – BRP[5:0] : Програмування швидкості передачі

Програмує квант часу T_{SCL} і визначає тривалість передачі одного біта (див. 6.1):

$$T_{SCL} = (BRP[5:0] + 1) \cdot T_{CLKIO},$$

$$t_{\text{БІТА}} = T_{SCL} \cdot n_{\text{кв. ч}},$$

де $n_{кв.ч}$ – програмно задане число квантів часу в одному біті, T_{CLKIO} – період тактової частоти підсистеми введення/виведення МК-ра [2, 3].

3) Bit 0 – Зарезервований біт

Цей біт зарезервований для майбутнього використання. Для сумісності з майбутніми пристроями повинен скидатися в 0.

CANBT2 – регістр тактової синхронізації 2

Біти	7	6	5	4	3	2	1	0	
	-	SJW1	SJW0	-	PRS2	PRS1	PRS0	-	CANBT2
Читання/Запис	-	R/W	R/W	-	R/W	R/W	R/W	-	
Початкові значення	-	0	0	-	0	0	0	-	

Рисунок 4.14 – Регістр CANBT2

1) Bit 7 – Зарезервований біт

Цей біт зарезервований для майбутнього використання. Для сумісності з майбутніми пристроями повинен скидатися в 0.

2) Bits 6:5 – SJW[1:0] : Довжина періоду ресинхронізації (пересинхронізації)

Щоб компенсувати фазові зсуви між кварцовими генераторами різних контролерів, контролер повинен проводити синхронізацію за перепадом з 1 в 0 в поточній передачі. Довжина періоду пересинхронізації визначає максимальне число квантів часу, на яку загальна тривалість біта може бути збільшена або зменшена за рахунок ресинхронізації (див 6.6.2).

3) Bit 4 – Зарезервований біт

Цей біт зарезервований для майбутнього використання. Для сумісності з майбутніми пристроями повинен скидання в 0.

4) Bits 3:1 – PRS[2:0] : Тривалість сегмента часу розповсюдження

Ця частина тривалості передачі біта (рисунок 6.1) необхідна для компенсації фізичної затримки в мережі. Це подвійна сума часу поширення сигналу шиною в один бік, затримки вхідного компаратора і вихідного пристрою.

5) Bit 0 – Зарезервований біт

Цей біт зарезервований для майбутнього використання. Для сумісності з майбутніми пристроями повинен скидатися в 0.

CANBT3 – регістр тактової синхронізації 3

Біти	7	6	5	4	3	2	1	0	
	-	PHS22	PHS21	PHS20	PHS12	PHS11	PHS10	SMP	CANBT3
Читання/Запис	-	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Початкові значення	-	0	0	0	0	0	0	0	

Рисунок 4.15 – Регістр CANBT3

1) Bit 7 – Зарезервований біт

Цей біт зарезервований для майбутнього використання. Для сумісності з майбутніми пристроями повинен скидатися в 0.

2) Bits 6:4 – PHS[22:20] : Довжина фазового сегмента 2

Цей фазовий сегмент (рисунок 6.1) потрібен для компенсації фазових зсувів границі (фронту) вхідного сигналу відносного періоду тактової частоти приймача CAN-модуля. Цей сегмент може зменшуватися в розмірах при ресинхронізації (див. 6.6.2).

3) Bits 3:1 – PHS[12:10] : Довжина фазового сегмента 1

Цей фазовий сегмент (рисунок 6.1) потрібен для компенсації фазових зсувів границі (фронту) вхідного сигналу відносного періода тактової частоти приймача CAN-модуля. Цей сегмент може збільшуватися при ресинхронізації.

4) Bit 0 – SMP: Точка(и) вибірки:

- 0 – одноразова вибірка;
- 1 – значення зчитується тричі, один раз в точці вибірки і по одному разу на відстані 1/2 тривалості T_{SCL}. Отриманий результат відповідає мажоритарному принципу (рисунок 6.1).

CANTCON – керуючий регістр таймера

Біти	7	6	5	4	3	2	1	0	
	TPRSC7	TPRSC6	TPRSC5	TPRSC4	TPRSC3	TPRSC2	TRPSC1	TPRSC0	CANTCON
Читання/Запис	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Початкові значення	0	0	0	0	0	0	0	0	

Рисунок 4.16 – Регістр CANTCON

1) Bits 7:0 – TPRSC[7:0] : Дільник частоти для таймера (рисунок 4.4)

В результаті роботи дільника частоти для таймера, період частоти на вході таймера встановлюється відповідно до наступного виразу:

$$T_{CLKCANTIM} = T_{CLKIO} \cdot 8 \cdot \left(CANTCON\left[7:0\right] + 1 \right),$$

де T_{SCKIO} – період тактової частоти підсистеми введення/виведення МК-ра [2, 3].

CANTIML і CANTIMH – *регістри CAN-таймера*

Біти	7	6	5	4	3	2	1	0	
	CANTIM7	CANTIM6	CANTIM5	CANTIM4	CANTIM3	CANTIM2	CANTIM1	CANTIM0	CANTIML
	CANTIM15	CANTIM14	CANTIM13	CANTIM12	CANTIM11	CANTIM10	CANTIM9	CANTIM8	CANTIMH
Біти	15	14	13	12	11	10	9	8	
Читання/Запис	R	R	R	R	R	R	R	R	
Початкові значення	0	0	0	0	0	0	0	0	

Рисунок 4.17 – Регістри CANTIML і CANTIMH

1) Bits 15:0 – CANTIM[15:0] : Значення лічильника CAN-таймера CANTIM (рисунок 4.4).

Діапазон значень лічильника CAN-таймера: від 0 до 65535.

CANTTCL і CANTTCH – регістри TTC-таймера

Біти	7	6	5	4	3	2	1	0	
	TIMTTC7	TIMTTC6	TIMTTC5	TIMTTC4	TIMTTC3	TIMTTC2	TIMTTC1	TIMTTC0	CANTTCL
	TIMTTC15	TIMTTC14	TIMTTC13	TIMTTC12	TIMTTC11	TIMTTC10	TIMTTC9	TIMTTC8	CANTTCH
Біти	15	14	13	12	11	10	9	8	
Читання/Запис	R	R	R	R	R	R	R	R	
Початкові значення	0	0	0	0	0	0	0	0	

Рисунок 4.18 – Регістри CANTTCL і CANTTCH

1) Bits 15:0 – TIMTTC[15:0] : Значення лічильника TTC-таймера

Діапазон значень TTC-таймера: від 0 до 65535.

CANTEC – лічильник помилок передачі

Біти	7	6	5	4	3	2	1	0	
	TEC7	TEC6	TEC5	TEC4	TEC3	TEC2	TEC1	TEC0	CANTEC
Читання/Запис	R	R	R	R	R	R	R	R	
Початкові значення	0	0	0	0	0	0	0	0	

Рисунок 4.19 – Регістр CANTEC

1) Bits 7:0 – TEC[7:0] : Кількість помилок передачі

Діапазон значень лічильника помилок передачі CAN: від 0 до 255.

CANREC – лічильник помилок прийому

Біти	7	6	5	4	3	2	1	0	
	REC7	REC6	REC5	REC4	REC3	REC2	REC1	REC0	CANREC
Читання/Запис	R	R	R	R	R	R	R	R	
Початкові значення	0	0	0	0	0	0	0	0	

Рисунок 4.20 – Регістр CANREC

1) Bits 7:0 – REC[7:0] : Кількість помилок прийому

Діапазон значень лічильника помилок прийому CAN: від 0 до 255.

CANHPMOB – регістр пріоритету MOb

Біти	7	6	5	4	3	2	1	0	
	HPMOB3	HPMOB2	HPMOB1	HPMOB0	CGP3	CGP2	CGP1	CGP0	CANHPMOB
Читання/Запис	R	R	R	R	R/W	R/W	R/W	R/W	
Початкові значення	1	1	1	1	0	0	0	0	

Рисунок 4.21 – Регістр CANHPMOB

1) Bits 7:4 – HPMOB[3:0] : Відображають номер MOb, який має найбільший пріоритет у регістрах CANSIT (має менший номер)

Якщо CANSIT = 0 (немає MOb), то повертає значення 0xF.

2) Bits 3:0 – CGP[3:0] : Біти загального призначення

Ці біти можуть бути перепрограмовані для відповідності з потрібним значенням бітів AINC і INDX[2:0] регістра CANPAGE.

CANPAGE – регістр вибору MOb

Біти	7	6	5	4	3	2	1	0	
	MOBNB3	MOBNB2	MOBNB1	MOBNB0	AINC	INDX2	INDX1	INDX0	CANPAGE
Читання/Запис	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Початкові значення	0	0	0	0	0	0	0	0	

Рисунок 4.22 – Регістр CANPAGE

1) Bit 7:4 – MOBNB[3:0] : Номер MOb

Вибір потрібного MOb, доступні номери від 0 до 14.

2) Bit 3 – $\overline{\text{AINC}}$: Автоінкремент покажчика на байт даних в буфері MOb:
   - 0 – автоінкремент покажчика (значення за замовчуванням);
   - 1 – немає автоінкремента.
3) Bit 2:0 – INDX[2:0] : Покажчик на байт даних в буфері MOb (працює як стек FIFO):

Покажчик на байт даних для заданого буфера об'єкта повідомлення (буфера MOb).

4.7.2 Регістри об'єктів повідомлень (регістри MOb)

Нижче розглядаються регістри MOb, розряди яких не мають відповідного значення за замовчуванням після скидання.

CANSTMOB – регістр статусу відповідного MOb

Біти	7	6	5	4	3	2	1	0	
	DLCW	TXOK	RXOK	BERR	SERR	CERR	FERR	AERR	CANSTMOB
Читання/Запис	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Початкові значення	-	-	-	-	-	-	-	-	

Рисунок 4.23 – Регістр CANSTMOB

1) Bit 7 – DLCW: Попередження невідповідності коду довжини повідомлення

Встановлюється, коли вхідне повідомлення не має очікуваного коду довжини повідомлення (DLC). Яким би не був тип кадру, поле DLC в регістрі CANCDMOB перезаписується отриманим кодом DLC (не може згенерувати переривання).

2) Bit 6 – TXOK: Успішна передача

Цей прапорець може згенерувати переривання. Він повинен очищатися за допомогою програмної послідовності читання-модифікація-запис, що здійснюється над усім регістром CANSTMOB.

Встановлюється, коли ініційований передачею сеанс зв'язку закінчено. Коли контролер готовий послати кадр, якщо два або більше об'єктів повідомлень одночасно налаштовані на передачу, виграє той об’єкт, у якого менше MOb-індекс (від 0 до 14).

3) Bit 5 – RXOK: Успішний прийом

Цей прапорець може згенерувати переривання. Він повинен очищатися за допомогою програмної послідовності читання-модифікація-запис, що здійснюється над усім регістром CANSTMOB.

Встановлюється, коли ініційований прийомом сеанс зв'язку успішно закінчено. У випадку, коли 2 або більше об'єктів повідомлень відповідають умові «успішний прийом», виграє той, у якого менше MOb-індекс (він оновлюється першим).

4) Bit 4 – BERR: Помилка біта

Цей прапорець може згенерувати переривання. Він повинен очищатися за допомогою програмної послідовності читання-модифікація-запис, що здійснюється над усім регістром CANSTMOB.

Встановлюється у тому разі, якщо прочитаний біт на виході відрізняється від того, що був переданий.

Винятки: рецесивний біт замінюється домінантним у полі арбітражу або домінантним бітом у полі підтвердження прийому при посилці кадру помилки.

5) Bit 3 – SERR: Помилка наповнення

Цей прапорець може згенерувати переривання. Він повинен очищатися за допомогою програмної послідовності читання-модифікація-запис, що здійснюється над усім регістром CANSTMOB.

Встановлюється при виявленні більше ніж п’яти послідовних однакових біт.

6) Bit 2 – CERR: Помилка CRC

Цей прапорець може згенерувати переривання. Він повинен очищатися за допомогою програмної послідовності читання-модифікація-запис, що здійснюється над усім регістром CANSTMOB.

Приймач проводить перевірку CRC для кожного отриманого повідомлення від початку кадру до закінчення поля даних. Якщо розраховане значення не збігається з розпізнаним полем CRC, біт встановлюється.

7) Bit 1 – FERR: Помилка форми

Цей прапорець може згенерувати переривання. Він повинен очищатися за допомогою програмної послідовності читання-модифікація-запис, що здійснюється над усім регістром CANSTMOB.

Помилка форми – це результат одного або більше порушень у наступних бітових полях:

- роздільник CRC (CRC delimiter);
- роздільник ACK (Acknowledgment delimiter);
- кінець кадру (EOF).

8) Bit 0 – AERR: Помилка підтвердження

Цей прапорець може згенерувати переривання. Він повинен очищатися за допомогою програмної послідовності читання-модифікація-запис, що здійснюється над усім регістром CANSTMOB.

Встановлюється, якщо не знайдено домінантний біт у полі підтвердження.

CANCDMOB – керуючий і DLC-регістр MOb

Біти	7	6	5	4	3	2	1	0	
	CONMOB1	CONMOB0	RPLV	IDE	DLC3	DLC2	DLC1	DLC0	CANCDMOB
Читання/Запис	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Початкові значення	-	-	-	-	-	-	-	-	

Рисунок 4.24 – Регістр CANCDMOB

1) Bit 7:6 – CONMOB[1:0] : Задає конфігурацію (режим роботи) MOb (таблиця 4.1).

Ці біти встановлюють режим, у якому перебуває відповідний MOb (після скидання немає значення за замовчуванням):

- 00 – дезактивовано;
- 01 – режим передачі;
- 10 – режим прийому;
- 11 – режим прийому відповідного набору кадрів в буфер.

Ці біти не очищуються, після того, як дія була вчинена. Користувач повинен перезавантажити конфігурацію, щоб дозволити нове з'єднання, а також для того, щоб мати можливість скинути прапорець BXOK. Якщо MOb активний, встановлюється відповідний біт в регістрах CANEN.

2) Bit 5 – RPLV: Статус відповіді

Використовується в режимі автоматичної відповіді після отримання віддаленого запиту кадру.

- 0 – відповідь не готова;
- 1 – відповідь готова і є дійсною (відбувається автовідповідь на кадр віддаленого запиту (таблиця 4.1)).

3) Bit 4 – IDE: Вид ідентифікатора

Біт IDE, який повинен бути переданий, або отриманий.

Цей біт оновлюється відповідним значенням отриманого кадру.

- 0 – CAN специфікація 2.0 A (довжина ідентифікатора 11 біт);
- 1 – CAN специфікація 2.0 B (довжина ідентифікатора 29 біт).

4) Bit 3:0 – DLC[3:0] : Код довжини даних

Кількість байт у полі даних повідомлення.

Поле DLC-кадру, який повинен бути переданий, або отриманий, діапазон значень від 0 до 8 (у двійковому коді). Якщо значення DLC-поля більше 8, то вважається рівним 8.

Це поле перезаписується відповідним полем отриманого кадру. Якщо очікуване значення DLC відрізняється від вхідного DLC, встановлюється біт DLCW в регістрі CANSTMOB.

CANIDT1, CANIDT2, CANIDT3, CANIDT4 – регістри ідентифікатора

V2.0 part A

Біти	15/7	14/6	13/5	12/4	11/3	10/2	9/1	8/0	
	-	-	-	-	-	RTRTAG	-	RB0TAG	CANIDT4
	-	-	-	-	-	-	-	-	CANIDT3
	IDT2	IDT1	IDT0	-	-	-	-	-	CANIDT2
	IDT10	IDT9	IDT8	IDT7	IDT6	IDT5	IDT4	IDT3	CANIDT1
Біти	31/23	30/22	29/21	28/20	27/19	26/18	25/17	24/16	
Читання/Запис	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Початкові значення	-	-	-	-	-	-	-	-	

V2.0 part B

Біти	15/7	14/6	13/5	12/4	11/3	10/2	9/1	8/0	
	IDT4	IDT3	IDT2	IDT1	IDT0	RTRTAG	RB1TAG	RB0TAG	CANIDT4
	IDT12	IDT11	IDT10	IDT9	IDT8	IDT7	IDT6	IDT5	CANIDT3
	IDT20	IDT19	IDT18	IDT17	IDT16	IDT15	IDT14	IDT13	CANIDT2
	IDT28	IDT27	IDT26	IDT25	IDT24	IDT23	IDT22	IDT21	CANIDT1
Біти	31/23	30/22	29/21	28/20	27/19	26/18	25/17	24/16	
Читання/Запис	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Початкові значення	-	-	-	-	-	-	-	-	

Рисунок 4.25 – Регістри CANIDT1, CANIDT2, CANIDT3 і CANIDT4

V2.0 part A

1) Bits 31:21 – IDT[10:0] : Ідентифікатор

Ідентифікатор кадру, який має бути переданий, або прийнятий.

Поле перезаписується відповідним значенням отриманого кадру.

2) Bits 20:3 – Зарезервовані біти

Зарезервовані. Але для майбутньої сумісності повинні скидатися в 0. При отриманні кадру ці біти перезаписуються випадковими значеннями.

3) Bit 2 – RTRTAG: Біт RTR

Біт кадру RTR, що повинен бути переданий, або прийнятий.

Поле перезаписується відповідним значенням отриманого кадру.

4) Bit 1 – Зарезервований біт

Зарезервований. Але для майбутньої сумісності повинен скидатися в 0. При отриманні кадру цей біт перезаписуються випадковим значенням.

5) Bit 0 – RB0TAG: Біт RB0

Біт RB0 кадру, який повинен бути переданий, або прийнятий.

Поле перезаписується відповідним значенням отриманого кадру.

V2.0 part B

1) Bits 31:3 –IDT[28:0] : Ідентифікатор

Ідентифікатор кадру, який має бути переданий, або прийнятий.

Поле перезаписується відповідним значенням отриманого кадру.

2) Bit 2 – RTRTAG: Біт RTR

Біт RTR кадру, що повинен бути переданий, або прийнятий.

Поле перезаписується відповідним значенням отриманого кадру.

3) Bit 1 – RB1TAG: Біт RB1

Біт RB1 кадру, який повинен бути переданий, або прийнятий.

Поле перезаписується відповідним значенням отриманого кадру.

4) Bit 0 – RB0TAG: Біт RB0

Біт RB0 кадру, який повинен бути переданий, або прийнятий.

Поле перезаписується відповідним значенням отриманого кадру.

CANIDM1, CANIDM2, CANIDM3, CANIDM4 – регістри масок ідентифікатора

V2.0 part A

1) Bit 31:21 – IDMSK[10:0] : Маска ідентифікатора:
   - 0 – порівняння відповідного біта вимкнене;
   - 1 – порівняння відповідного біта увімкнене.
2) Bit 20:3 – Зарезервовані біти

Зарезервовані. Але для майбутньої сумісності повинні скидатися в 0.

3) Bit 2 – RTRMSK: Маска біта RTR:
   - 0 – порівняння RTR-біта вимкнене;

- 1 – порівняння RTR-біта ввімкнене.

4) Bit 1 – Зарезервований біт

Зарезервований. Але для майбутньої сумісності повинен скидатися в 0.

5) Bit 0 – IDEMSK: Маска біта IDE:

- 0 – порівняння IDE-біта вимкнене;
- 1 – порівняння IDE-біта ввімкнене.

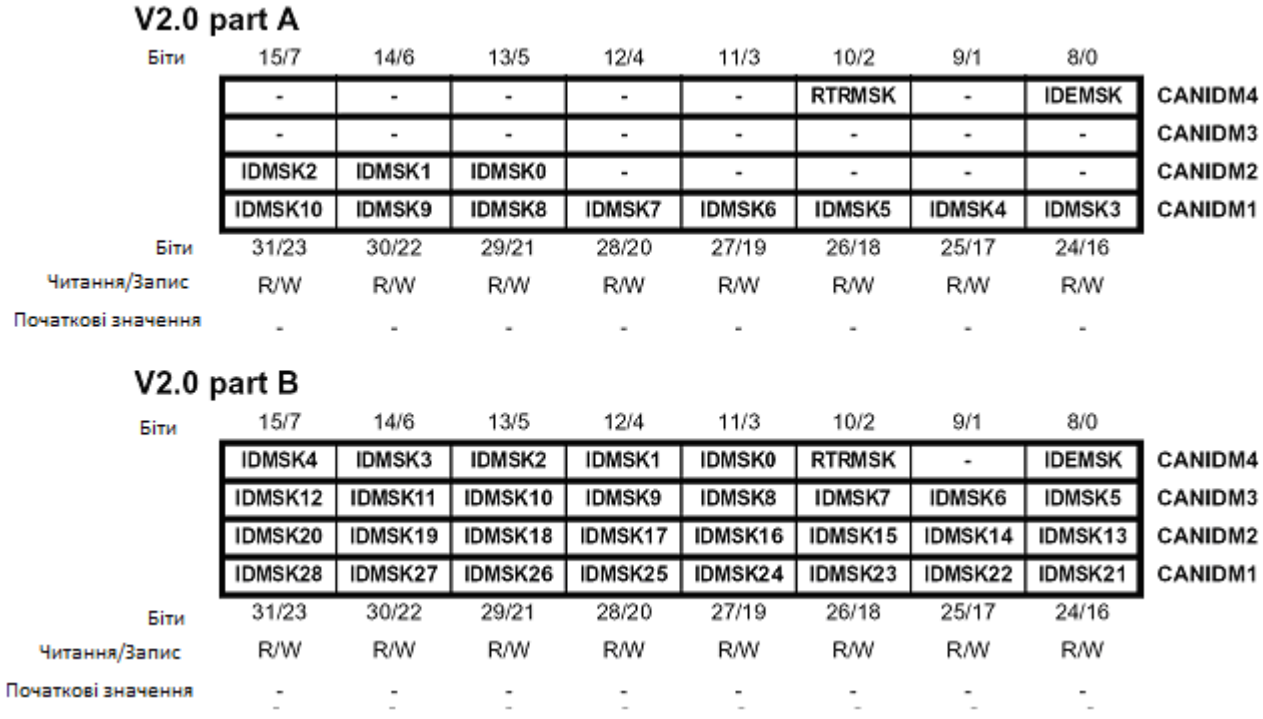


Рисунок 4.26 – Регістри CANIDM1, CANIDM2, CANIDM3 і CANIDM4

V2.0 part B

1) Bits 31:3 – IDMSK[28:0] : Маска ідентифікатора:

- 0 – порівняння відповідного біта вимкнене;.
- 1 – порівняння відповідного біта ввімкнене.

2) Bit 2 – RTRMSK: Маска біта RTR:

- 0 – порівняння RTR-біта вимкнене;
- 1 – порівняння RTR-біта ввімкнене.

3) Bit 1 – Зарезервований біт

Зарезервований. Але для майбутньої сумісності повинен скидатися в 0.

4) Bit 0 – IDEMSK: Маска біта IDE:

- 0 – порівняння IDE-біта вимкнене;
- 1 – порівняння IDE-біта ввімкнене.

CANSTML і CANSTMH – регістр часових відміток

Біти	7	6	5	4	3	2	1	0	
	TIMSTM7	TIMSTM6	TIMSTM5	TIMSTM4	TIMSTM3	TIMSTM2	TIMSTM1	TIMSTM0	CANSTML
	TIMSTM15	TIMSTM14	TIMSTM13	TIMSTM12	TIMSTM11	TIMSTM10	TIMSTM9	TIMSTM8	CANSTMH
Біти	15	14	13	12	11	10	9	8	
Читання/Запис	R	R	R	R	R	R	R	R	
Початкові значення	-	-	-	-	-	-	-	-	

Рисунок 4.27 – Регістри CANSTML і CANSTMH

1) Bits 15:0 – TIMSTM15:0] : Лічильник часових відміток

Діапазон значень відміток часу: від 0 до 65535.

CANMSG – регістр даних

Біти	7	6	5	4	3	2	1	0	
	MSG 7	MSG 6	MSG 5	MSG 4	MSG 3	MSG 2	MSG 1	MSG 0	CANMSG
Читання/Запис	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Початкові значення	-	-	-	-	-	-	-	-	

Рисунок 4.28 – Регістр CANMSG

1) Bit 7:0 – MSG7:0] : Дані повідомлення

Регістр містить байт даних в буфері MOb, на який вказують 3 молодших біта регістра CANPAGE.

5 ОБРОБКА ПОМИЛОК

Існує 5 різних типів не взаємовиключаючих (можуть виникати одночасно) помилок.

5.1 Помилка біта

Вузол, що передає дані на шину, одночасно її контролює. Помилка біта виникає, якщо значення біта на шині відрізняється від значення, що передається. Виняток становить поле арбітражу, поле підтвердження та передачі додаткового 6-го біта протилежної полярності, після передачі п’яти біт однакової полярності.

Передавач, що посилає прапорець пасивної помилки і виявляє “домінантний” біт, не інтерпретує це як помилку біта.

5.2 Помилка заповнення

Помилка заповнення виникає, якщо при прийомі між початком повідомлення та роздільником CRC виявлено підряд 6 біт однакової полярності.

5.3 Помилка CRC

Послідовність CRC обчислюється передавачем і передається в кадрі повідомлення, а також обчислюється приймачем при прийомі повідомлення. Помилка CRC виникає, якщо значення, що обчислене приймачем, не збігається із прийнятим від передавача.

5.4 Помилка форми

Помилка форми виявляється, якщо бітове поле фіксованого формату (кінець повідомлення, роздільники CRC і підтвердження) містять один або більше заборонених біти. Для приймача “домінантний” біт протягом останнього біта “кінець кадру” не інтерпретується як помилка форми.

5.5 Помилка підтвердження

Помилка підтвердження виявляється передавачем щораз, коли він не виявляє “домінантний” біт в області підтвердження, що говорить про те, що жоден з вузлів не одержав повідомлення правильно. При цьому повідомлення повинне бути передане повторно.

Лічильники помилок змінюються відповідно до таких правил:

1. При виявленні помилки вузлом-одержувачем (приймачем) лічильник помилок приймача збільшується на 1, крім випадків, коли виявлена помилка була помилкою під час передачі приймачем прапорця активної помилки та прапорця перевантаження.

2. Якщо вузол-приймач (одержувач) виявляє помилку біта при передачі приймачем прапорця активної помилки або прапорця перевантаження, лічильник помилок приймача збільшується на 8.

3. Кожен вузол допускає до 7 послідовних нульових (домінантних) біт після передачі прапорця активної помилки, пасивної помилки та прапорця перевантаження. У випадку виявлення 14-ти послідовних домінантних біт у випадку передачі прапорця активної помилки або прапорця перевантаження або після виявлення 8-ми послідовних домінантних біт, які слідують за прапорцем пасивної помилки, після кожної послідовності з додаткових 8-ми нульових (домінантних) біт кожен вузол-передавач збільшує лічильник помилок на 8, а також кожен вузол-приймач збільшує лічильник помилок, що приймаються, на 8.

4. Коли вузол-передавач передає прапорець помилки, лічильник помилок передавача збільшується на 8.

5. Якщо вузол-передавач виявляє помилку біта при передачі прапорця активної помилки або прапорця перевантаження, то лічильник помилок передавача збільшується на 8.

6. Після успішної передачі повідомлення (одержання АСК і не було помилки до кінця кадру (передачі роздільника АСК (АСК DEL = 1) і поля кінця повідомлення (сім одиничних біт))) лічильник помилок передавача зменшується на 1, якщо він не дорівнює 0.

7. Після успішного отримання повідомлення (прийом повідомлення без помилки та успішна передача біта АСК), лічильник помилок приймача зменшується на 1, якщо його значення було від 1 до 127. Якщо значення лічильника помилок приймача дорівнює 0, то воно не змінюється. Якщо значення лічильника помилок приймача було більше 127, воно буде встановлено в значення від 119 до 127.

8. Вузол перебуває в стані пасивної помилки, якщо лічильник помилок передавача або приймача дорівнює або більше 128. При переході в стан пасивної помилки вузол посилає повідомлення активної помилки.

9. Вузол переходить у стан відключення від шини, коли значення лічильника помилок передавача більше або дорівнює 256.

10. Вузол зі стану пасивної помилки переходить у стан активної помилки, коли значення обох його лічильників менше або дорівнює 127.

11. Вузлу зі стану відключення від шини дозволяється перейти в стан активної помилки з обома лічильниками, які встановлені в нуль, після виявлення на шині 128-ми послідовностей з 11 одиничних біт (рисунок 5.1).

Вузол у стані "активної помилки" при виявленні помилки передає активний прапор помилки. Форма прапора помилки порушує закон кодування бітового потоку методом розрядного заповнення. Внаслідок цього всі вузли виявляють умову помилки і починають передавати прапор помилки. В результаті, послідовність біт з лог. "0", контрольована на шині є суперпозицією прапорів

помилок окремих вузлів. Загальна довжина цієї послідовності - від 6 до 12 біт з лог. "0".

Вузол у стані "пасивної помилки" при виявленні помилки передає пасивний прапор помилки, він чекає послідовності з 6 однакових біт, що визначають початок прапора пасивної помилки. Коли ця послідовність буде виявлена, прапор пасивної помилки буде завершений.

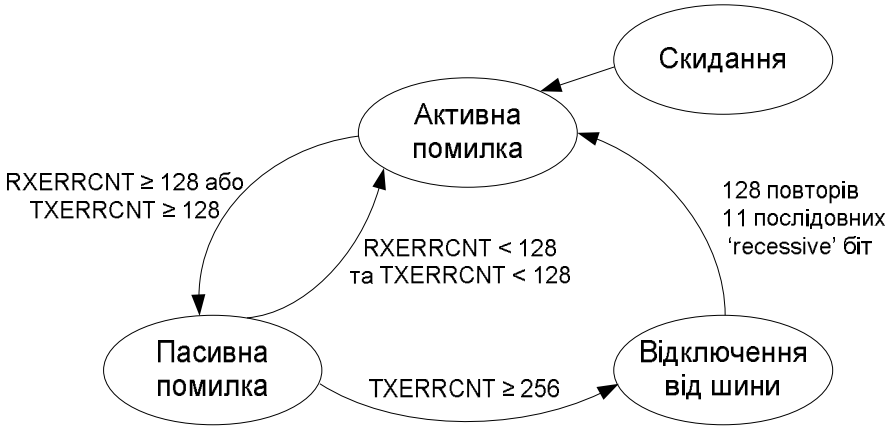


Рисунок 5.1 – Стан лічильників помилок

6 БІТОВА СИНХРОНІЗАЦІЯ

6.1 Номінальна швидкість передачі та номінальна тривалість біта

Номінальна швидкість передачі інформації визначається числом бітів у секунду, які передаються ідеальним передавачем при відсутності пересинхронізації.

Номінальний час передачі біта =

=1/ номінальна швидкість передачі інформації в бітах в секунду.

Номінальний час передачі біта можна розділити на кілька ділянок, що не перекриваються (рисунок 6.1):

- сегмент синхронізації (SYNC_SEG);
- сегмент часу розповсюдження (PROP_SEG);
- фазовий сегмент 1 (PHASE_SEG1);
- фазовий сегмент 2 (PHASE_SEG2).

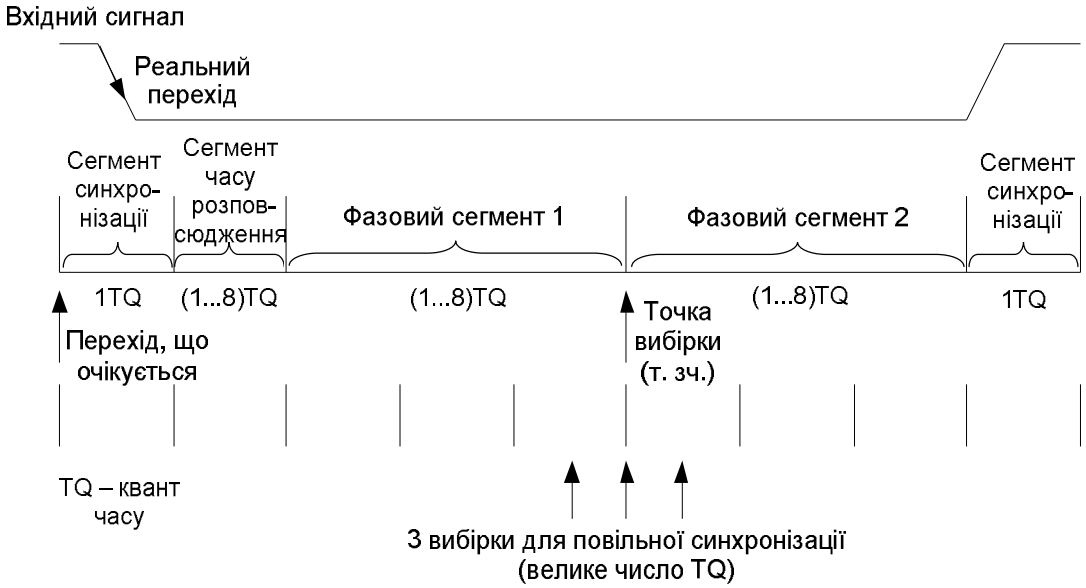


Рисунок 6.1 – Номінальний час біта

Кожен сегмент складається із цілого числа відрізків часу, які називаються квантами часу TQ. Номінальний час передачі біта коливається від 8TQ до 25TQ (включно). Мінімальний номінальний час передачі біта дорівнює 1мкс, що відповідає швидкості передачі 1 Мбіт/с.

Тривалість одного кванта часу, як правило, визначається тактовою частотою мікроконтролера і значенням коефіцієнта ділення цієї частоти, що може програмно змінюватися:

$$T_{SCL} = (BRP[5:0] + 1) \cdot T_{CLKIO},$$

де T_{CLKIO} – період тактової частоти підсистеми введення/виведення МК-ра [2, 3], *BRP[5:0]* – десятковий еквівалент шести біт регістра CANBT1.

6.2 Сегмент синхронізації (SYNC_SEG)

Сегмент синхронізації (SYNC_SEG) використовується для синхронізації різних вузлів на шині. Передбачається, що фронт сигналу, що визначає початок передачі, повинен перебувати в межах цього сегмента.

6.3 Сегмент часу розповсюдження

Використовується, щоб компенсувати час фізичного запізнювання в мережі. Цей час дорівнює подвоєній сумі часу розповсюдження сигналу на лінії шини, затримки вхідного компаратора, і затримки вихідного формувача. Значення цього сегмента може програмуватися.

6.4 Фазові сегменти 1, 2

Ці сегменти використовуються, щоб компенсувати помилки фази при прийомі (фазові зсуви). Фазовий сегмент 1 може бути подовжений, а фазовий 2 - вкорочений пересинхронізацією (ресинхронізацією).

Фазові сегменти призначені для оптимального розташування точки вибірки отриманого біта у межах часу біта, що передається. Точка вибірки розташовується між фазовим сегментом 1 і фазовим сегментом 2.

Фазовий сегмент 1 визначає точку вибірки в межах біта, що передається. Фазовий сегмент 2 забезпечує затримку до початку наступного біта. Тривалість обох цих сегментів може програмуватися.

6.5 Точка зчитування (вибірки) (Sample point)

Точка зчитування (вибірки) (Sample point) т. зч. – це момент часу, при якому рівень на шині читається та інтерпретується, як значення відповідного біта. Вона розташована наприкінці фазового сегмента 1 (рисунок 6.1).

Якщо синхронізація біта повільна (містить велику кількість TQ), то можливо задати багаторазовий вибірковий контроль стану шини. В цьому випадку CAN-модуль робить вибірку три рази для кожного біта, який прийнятий, з періодичністю TQ/2. Значення, отримане за мажоритарним принципом (два або три рази), приймається як правильне. Увімкнення багаторазової вибірки

здійснюється установкою спеціального програмованого біта (біт SMP в регістрі CANBT3).

6.6 Синхронізація

Щоб компенсувати зсув фази між частотами генераторів різних вузлів шини та при зміні ведучого при арбітражі, кожен CAN-модуль повинен синхронізуватися до переходу рівня вхідного сигналу з 1 на 0 (рисунок 6.1). Як тільки перехід виявлений, то логіка синхронізації порівнює розміщення переходу з переходом, що очікується при прийомі, та виконує налаштування значень фазових сегментів 1 та 2. Є два механізми синхронізації:

- апаратна синхронізація;
- синхронізація з відновленням тактових інтервалів (пересинхронізація/ ресинхронізація).

6.6.1 Апаратна синхронізація

Апаратна синхронізація виконується при переході від “рецесивного” до “домінантного” біта протягом холостого ходу шини, вказуючи початок повідомлення. При апаратній (жорсткій) синхронізації тактові інтервали (тривалість сегментів) не змінюються протягом усього повідомлення.

Після апаратної синхронізації внутрішній бітовий час кожного вузла перезапускається з сегмента синхронізації SYNC_SEG (рисунок 6.1).

6.6.2 Синхронізація з відновленням тактових інтервалів (пере-синхронізація, ресинхронізація)

Синхронізація з відновленням тактових інтервалів призначена для зменшення фазових спотворень при прийомі та виконується автоматичним

подовженням фазового сегмента 1 або скороченням фазового сегмента 2. Максимальне значення зміни фазових сегментів коливається в межах від 1TQ до 4TQ. Синхронізація виконується тільки при переходах від “рецесивного” до “домінантного” біта протягом прийому кадру. Фіксоване значення максимального числа послідовних біт однакової полярності (“біт-стаффінг“) гарантує своєчасне відновлення синхронізації.

Фазове спотворення (помилка фази) визначається положенням фронту вхідного сигналу (переходу з одиниці в нуль) відносно точки зчитування бітового інтервалу приймача, в якому з’явився перехід з «1» в «0» і вимірюється у квантах часу TQ. Знак помилки фази (e) визначається в такий спосіб:

- e = 0, якщо фронт вхідного сигналу (перехід з 1 в 0) перебуває в межах сегмента синхронізації SYNC_SEG (ресинхронізація не проводиться);
- e > 0, якщо фронт вхідного сигналу (перехід з 1 в 0) не перебуває в межах SYNC_SEG та знаходиться перед точкою зчитування бітового інтервалу приймача, протягом якого з’явився перехід вхідного сигналу з «1» в «0»;
- e < 0, якщо фронт сигналу (перехід з 1 в 0) не перебуває в межах SYNC_SEG та знаходиться після точки зчитування бітового інтервалу приймача, протягом якого з’явився перехід вхідного сигналу з «1» в «0».

Значення e визначається в квантах TQ як різниця часу точки зчитування ($t_{т.зч.}$) бітового інтервалу приймача, протягом якого з’явився перехід вхідного сигналу з «1» в «0», та часу появи цього переходу ($t_{ф}$).

Ефект від пересинхронізації – такий самий як і у випадку апаратної синхронізації, коли величина помилки фази (e) менше або дорівнює значенню ширини періоду пересинхронізації, що програмується.

Коли величина помилки фази (e) більша, ніж ширина переходу пересинхронізації, що програмується, тоді:

- якщо помилка фази (e) додатна, то фазовий сегмент 1 подовжується на ширину періоду пересинхронізації, що програмується;

– якщо помилка фази (e) від’ємна, то фазовий сегмент 2 скорочується на ширину періоду пересинхронізації, що програмується.

6.6.3 Правила синхронізації

Апаратна синхронізація та пересинхронізація – дві форми синхронізації. Вони задовольняють наступним умовам:

1. Допускається тільки один тип пересинхронізації в межах бітового інтервалу.
2. Фронт сигналу буде використовуватися для пересинхронізації тільки тоді, якщо значення, яке виявлене при попередній точці зчитування (попереднє значення на шині), відрізняється від значення на шині відразу після фронту.
3. Апаратна синхронізація відбувається щораз, коли є перехід від "рецесивного" біта до "домінантного" протягом простою шини.

На рисунках 6.2, 6.3 наведені приклади пересинхронізації.

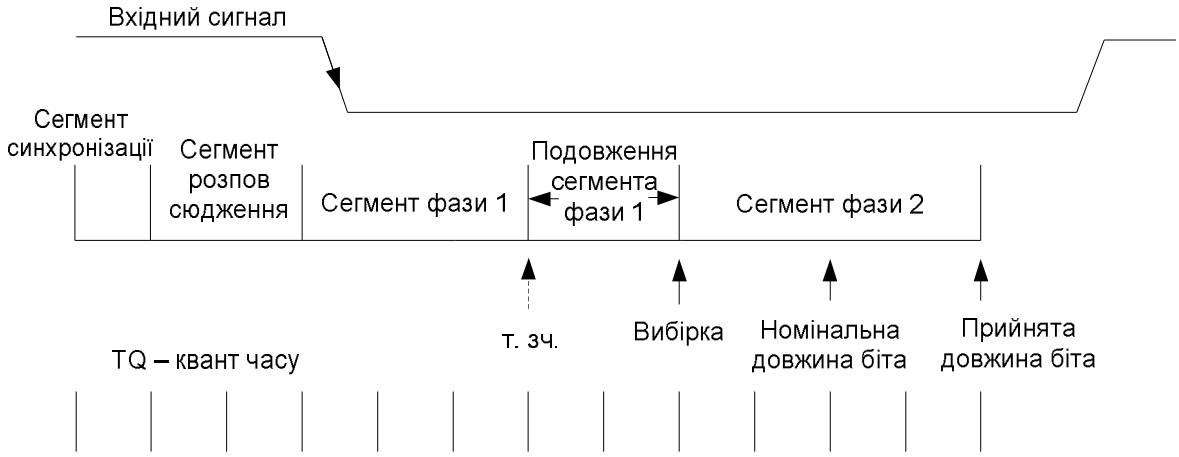


Рисунок 6.2 – Синхронізація з відновленням тактових інтервалів за рахунок автоматичного подовження фазового сегмента (сегмента фази) 1

6.6.4 Тривалість сегментів

SYNC_SEG – 1 квант часу.

PROP_SEG – програмується, може бути 1,2...8 квантів часу.

PHASE_SEG1 – програмується, може бути 1,2...8 квантів часу.

PHASE_1 – максимум з PHASE_SEG1 і часу обробки інформації.

Час обробки інформації – менше або дорівнює 2 квантам часу.

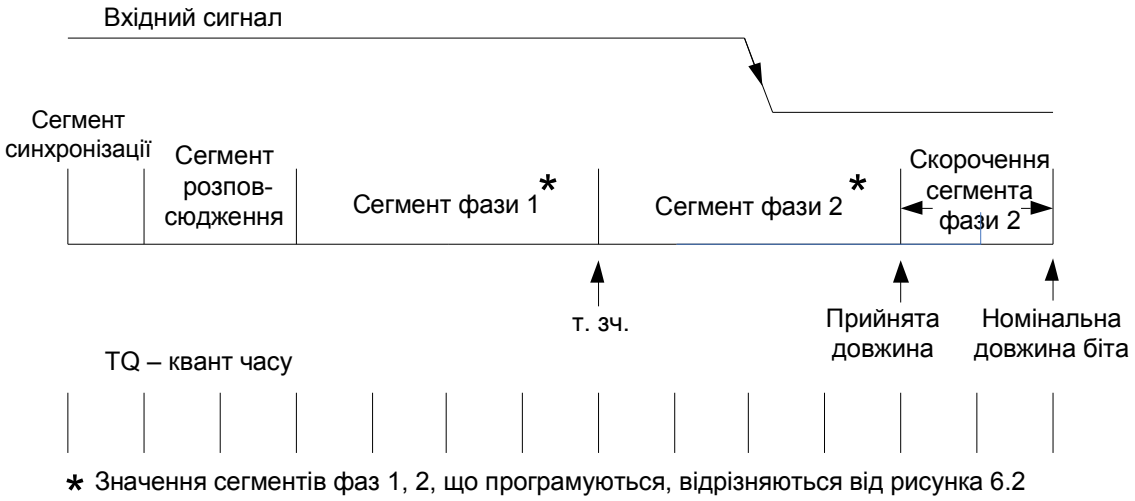


Рисунок 6.3 – Синхронізація з відновленням тактових інтервалів за рахунок автоматичного скорочення фазового сегмента (сегмента фази) 2

7 ФІЗИЧНИЙ РІВЕНЬ ПРОТОКОЛУ CAN

Фізичний рівень (Physical Layer) протоколу CAN визначає опір кабелю, рівень електричних сигналів у мережі і т. ін. Існує кілька фізичних рівнів протоколу CAN (ISO 11898, ISO 11519, SAE J2411) [4].

У переважній більшості випадків використовується фізичний рівень CAN, який описаний у стандарті ISO 11898. ISO 11898 в якості середовища передачі визначає двопровідну диференціальну лінію з імпедансом (термінатори) 120 Ом (допускається коливання імпедансу в межах від 108 Ом до 132 Ом). Фізичний рівень CAN реалізований на спеціальних мікросхемах – CAN-приймачах-

передавачах (transceivers), які перетворюють звичайні TTL-рівні сигналів, що використовуються CAN-контролерами в рівні сигналів на шині CAN. Найпоширеніший CAN-приймач-передавач (трансивер) – Phillips 82C250, який повністю відповідає стандарту ISO 11898.

Максимальна швидкість передачі даних CAN-мережею відповідно до протоколу дорівнює 1Мбіт/с. При швидкості в 1Мбіт/с максимальна довжина кабелю дорівнює приблизно 40 метрам. Обмеження на довжину кабелю пов'язане з кінцевою швидкістю світла та механізмом побітового арбітражу (під час арбітражу всі вузли мережі повинні одержувати поточний біт передачі практично одночасно, тобто сигнал повинен встигнути поширитися по всьому кабелю за один квант часу в мережі). Співвідношення між швидкістю передачі й максимальною довжиною кабелю наведені в таблиці 7.1.

Таблиця7.1 – Залежність швидкості передачі даних від довжини шини

Швидкість передачі	Час передачі біта	Довжина лінії зв'язку
1 Мбіт/с	1 мкс	40 м
800 Кбіт/с	1,25 мкс	50 м
500 Кбіт/с	2 мкс	100 м
250 Кбіт/с	4 мкс	250 м
125 Кбіт/с	8 мкс	500 м
62,5 Кбіт/с	20 мкс	1000 м
20 Кбіт/с	50 мкс	2500 м
10 Кбіт/с	100 мкс	5000 м

При необхідності передавати дані на більші відстані рекомендується використовувати вузли-ретранслятори.

Роз'єми для CAN-мережі до цих пір не стандартизовані. Кожен протокол високого рівня зазвичай визначає свій тип роз'ємів для CAN-мережі. Але міжнародна організація CiA (The CAN in Automation international users and manufacturers group) рекомендує у своєму стандарті CiA DS-102 для з'єднання окремих вузлів з CAN-шиною використовувати наведену на рисунку 7.1 розводку

виводів 9-контактного роз'єму D-Sub. Дану угоду використовують деякі протоколи більш високих рівнів CAN (CANopen, Smart Distributed System).

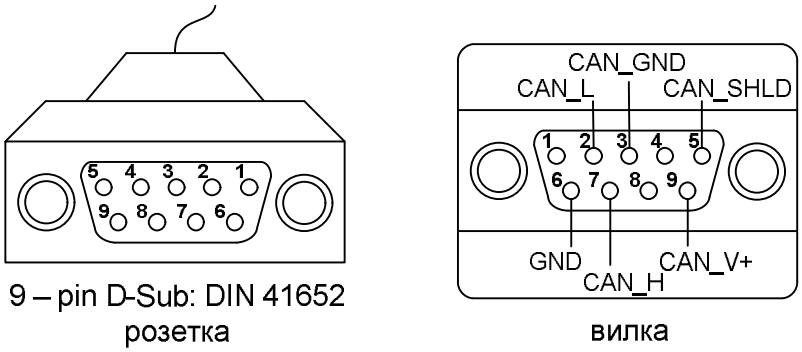


Рисунок 7.1 – CAN- роз'єм згідно зі стандартом CiA DS-102

Таблиця 7.2 – Призначення виводів CAN-роз'єму згідно зі стандартом CiA DS-102

Контакт	Ланцюг	Примітка
1	–	–
2	CAN_L	Низький рівень – домінантний стан
3	CAN_GND	Земля
4	–	–
5	(CAN_SHLD)	Екран (необов'язковий)
6	GND	Земля (необов'язковий)
7	CAN_H	Високий рівень – рецесивний стан
8	–	–
9	(CAN_V+)	Живлення (необов'язковий)

8 CAN - ТРАНСИВЕР

Фізичне середовище передачі даних CAN-інтерфейса, характеристики приймачів-передавачів, розводка роз'ємів, властивості кабелів, топологія мережі не обумовлюється специфікацією CAN. Для організації дротового з'єднання на CAN-шині найбільш широке поширення одержали два типи трансиверів (приймачів-передавачів): «High Speed» (ISO 11898-2) та «Fault Tolerant»

приймачі-передавачі. Трансивери, виконані у відповідності зі стандартом «High Speed» (ISO 11898-2), найбільш прості, дешеві й дають можливість передавати дані зі швидкістю до 1 Мбіт/с. «Fault Tolerant» приймачі-передавачі не дуже чутливі до ушкоджень на шині та дозволяють побудувати високонадійну мережу, що мало споживає, зі швидкостями передачі даних не вище 125 Кбіт/с. Передача живлення через CAN-інтерфейс не суперечить стандарту, але реально не використовується через високу вартість передавачів.

Всі перераховані вище способи організації фізичного середовища передачі даних повинні задовольняти набору обов'язкових вимог, продиктованих способами арбітражу та синхронізації, прийнятими для CAN-інтерфейсу. Фізичне середовище передачі даних повинне мати можливість перебувати у двох станах - «рецесивному» та «домінантному». CAN-шина перебуває в «рецесивному» стані, якщо жоден вузол не передає «домінантний» біт, і в «домінантному» стані, якщо хоча б один вузол встановив «домінантний» біт. Іншими словами на шині не повинен відбуватися конфлікт, якщо одночасно кілька пристроїв виставили різні логічні рівні. На шині в цьому випадку повинен встановлюватися «домінантний» стан.

«High Speed» трансивери стандарту ISO 11898-2 отримали найбільш широке поширення. Вони з'єднуються із шиною CAN так, як показано на рисунку 8.1.

Напруга живлення трансивера становить 5 В. Рівні напруг, що відповідають «рецесивному» й «домінантному» для стандарту ISO 11898-2 показані на рисунку 8.2.

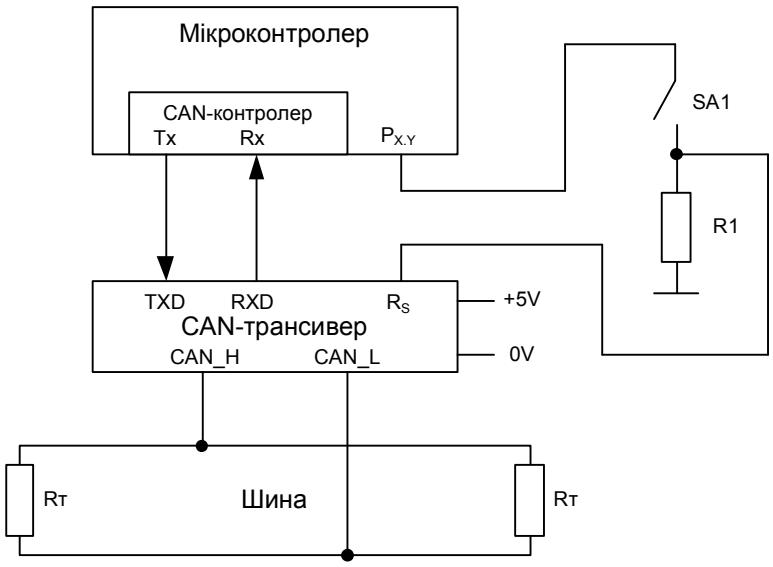


Рисунок 8.1 – З'єднання із шиною «High Speed» трансиверів

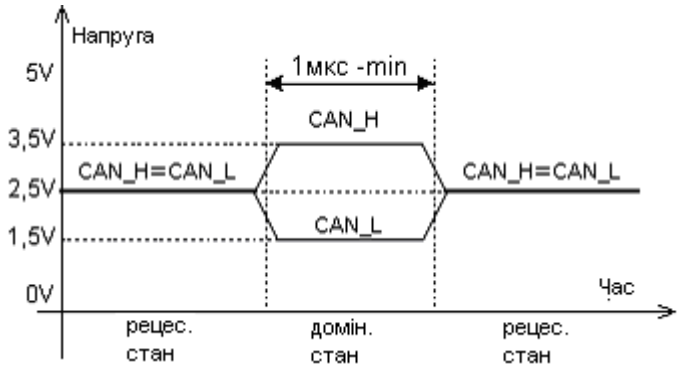


Рисунок 8.2 – Фізичні рівні "high speed" трансиверів (ISO 11898-2)

Приймачі мають диференціальну схему включення та сприймають стан шини як «рецесивний», якщо різниця напруг $U_{CAN\_H} - U_{CAN\_L} < 0.5B$, і як «домінантний» , якщо $U_{CAN\_H} - U_{CAN\_L} > 0.9B$. Номінальні значення напруг на лініях при «домінантний» стані становлять 3,5В та 1,5В відповідно для ліній CAN_H та CAN_L.

На рисунку 8.3 наведена структура мікросхеми MCP2551.

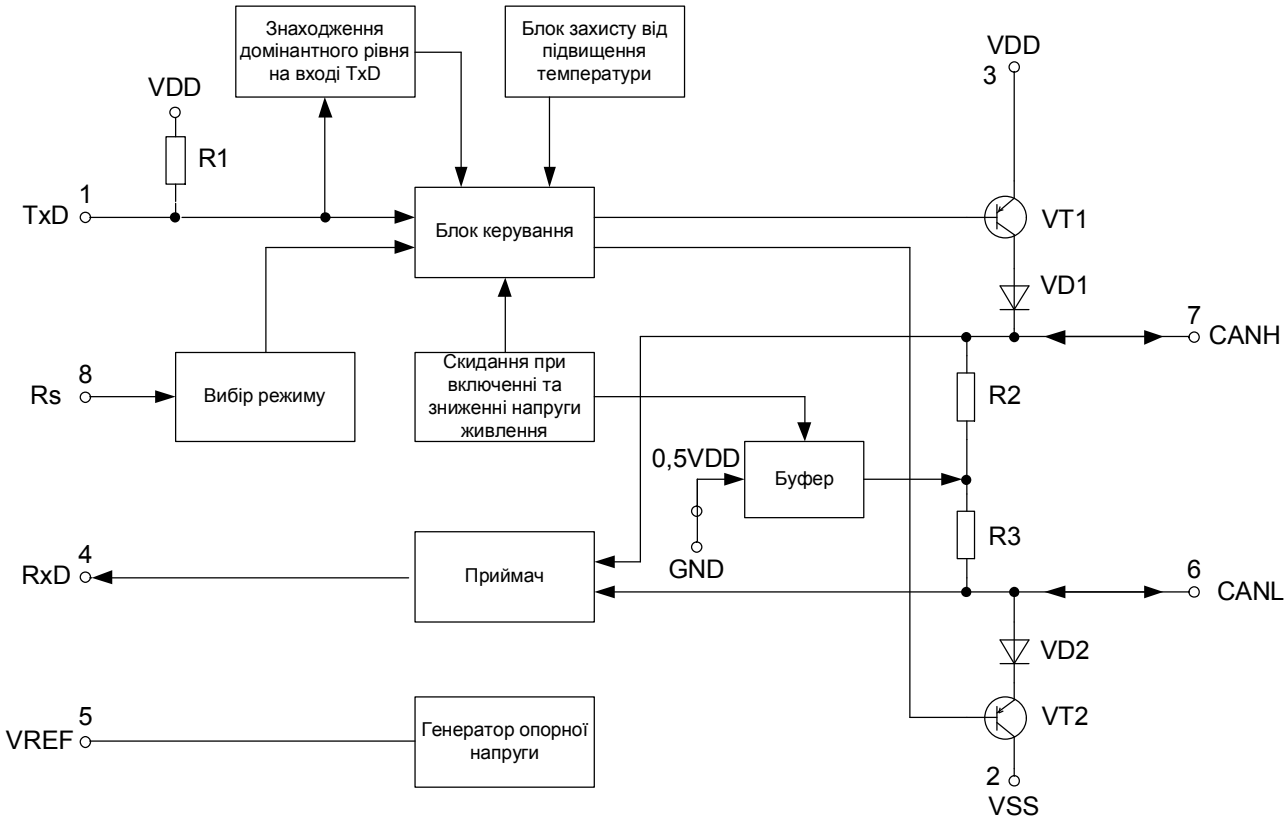


Рисунок 8.3 – Структурна схема «High-Speed» CAN-трансивера

Існує три режими роботи, які налаштовуються ззовні через вивід Rs:

1. Високошвидкісний режим (High-Speed).
2. Режим регулювання швидкості наростання сигналу, що передається в CAN-шину (Slope-Control).
3. Режим очікування (Standby)

Високошвидкісний режим вибирається підключенням виводу Rs до VSS. У цьому режимі вихідні драйвери мають швидкий час наростання і спаду, що забезпечує найвищі швидкості передачі до 1 Мбіта/с і/або максимальну довжину шини, а також забезпечуючи мінімальні циклічні затримки приймача.

Якщо потрібно зменшити випромінювані драйвером електромагнітні завади, MCP2551 можна встановити в режим регулювання швидкості наростання сигналу, що передається в CAN-шину, підключенням резистора від виводу Rs на спільний мінус (на рисунку 8.1 – резистор R1). У цьому режимі швидкість наростання вихідної напруги на одному виводі (на CANH або CANL) в

основному пропорційна вихідному струму на виводі Rs. Струм повинен бути в діапазоні від 10 мкА < I_{RS} < 200 мкА, який відповідає напрузі на виводі Rs в діапазоні 0.4·VDD < VRS < 0.6·VDD відповідно (або зазвичай 0.5·VDD).

Зменшення швидкості наростання вихідної напруги призводить до зменшення швидкості передачі даних CAN при заданій довжині шини, або до скорочення довжини шини при заданій швидкості передачі даних.

Режим очікування (або режим сну (sleep)) встановлюється під'єднанням виводу Rs до VDD (на рисунку 8.1 для цього використані перемикач SA1 та лінія $P_{X.Y}$ одного із вільних портів введення-виведення МК-ра). У режимі сну передавач від'єднаний, а приймач працює в режимі зниженого енергоспоживання. Вивід RxD, який приймає, як і раніше функціонує, але на нижчій швидкості.

Режим очікування можна використовувати для установки пристрою в режим низького енергоспоживання і від'єднання передавача у випадку, якщо контролер CAN несправний і видає на шину непередбачувані дані.

«Fault Tolerant» приймачі-передавачі спочатку були призначені для застосування в легкових автомобілях, де не була потрібна висока швидкість передачі даних, але велике значення мали такі якості шини, як висока надійність і мінімальне енергоспоживання. «Fault Tolerant» трансивери мають два входи живлення: основне 5В та резервне 6...12В. Мікроконтролер за допомогою керуючих сигналів може переводити трансивер у режими зниженого енергоспоживання (Standby і Sleep). При переході в режим Sleep приймач-передавач формує на одному з виходів сигнал, за допомогою якого можна від'єднати джерело живлення всього пристрою. При цьому сам приймач-передавач продовжує живитися від резервного батарейного живлення, споживаючи дуже маленький струм: 15-30 мкА. Цього живлення трансиверу досить для того, щоб при виявленні активності на шині увімкнути джерело живлення та повідомити мікроконтролеру про необхідність повернутися в нормальний режим роботи.

«Fault Tolerant» трансивери є більш інтелектуальними приймачами-передавачами ніж «High Speed». У нормальному режимі роботи вони використовують диференціальний режим передачі даних, але якщо вони виявляють на шині пошкодження (обрив або замикання одного із дротів), то приймачі-передавачі переходять у режим однопровідної передачі тією лінією, що зберегла працездатність та застосовують необхідні переключення для обмеження струму, який викликаний ушкодженням, що відбулося, на лінії. При цьому трансивери увесь час продовжують стежити за станом шини, і, якщо виявляють, що ушкодження усунуте, автоматично переходять до нормального диференціального режиму передачі даних, що має більш високу завадостійкість.

Рівні напруг, що відповідають станам «рецесивний» та «домінантний» для «Fault Tolerant» трансивера показані на рисунку 8.4.

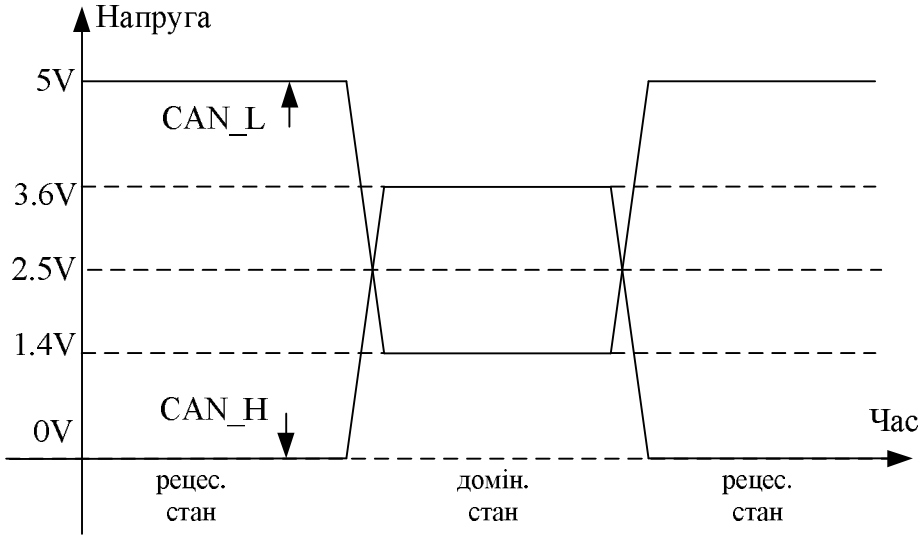


Рисунок 8.4 – Фізичні рівні «Fault Tolerant» трансиверів

Стан «домінантний» представляється значенням диференціального сигналу, що дорівнює 2,2В (CAN_H=3,6В й CAN_L=1,4В), а «рецесивний» - значенням: -5В (CAN_H=0В та CAN_L=5В). Поріг спрацьовування диференціального приймача має значення: -2,9В.

Розглянемо трансивер «Fault Tolerant», виконаний на мікросхемі Motorola MC33388. На рисунку 8.5 зображений зовнішній вигляд мікросхеми з позначенням виводів.

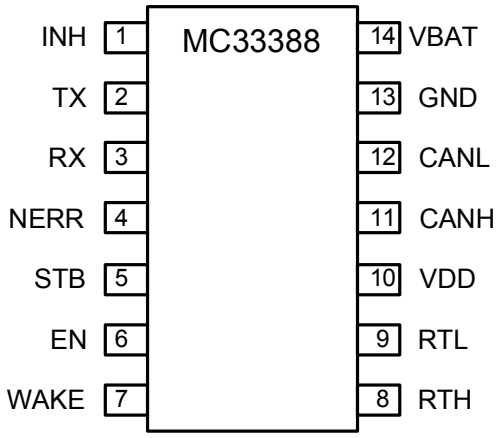


Рисунок 8.5 – Мікросхема Motorola MC33388

Мікросхема може керувати зовнішнім стабілізатором напруги системи через вивід INH. Це дозволяє перемикати пристрій у режим зниженого енергоспоживання. Вихід із цього стану відбувається при виявленні активності на шині або здійснюється за допомогою ключа. На рисунку 8.6 наведена структура мікросхеми MC33388, а на рисунку 9.1 – схема його під'єднання.

Пристрій має чотири режими роботи: нормальний, тільки прийом, Standby VBAT (живлення від батарейки), Sleep. Режими роботи визначаються станом сигналів на вході EN та STB. У нормальному режимі всі функції доступні і вихід NERR сигналізує про помилки, виявлені пристроєм. У режимі «Тільки прийом» передавач від'єднаний, тому пристрій не керує шиною, а функції прийому здійснюються як у нормальному режимі. У режимі «Sleep» функції передачі та прийому відключені. Вивід CANL через зовнішній опір, підключений до виводу RTL, і через внутрішній опір 12,5 кОм під'єднаний до VBAT. Сигналом низького рівня, який формується на виході INH, основне джерело живлення від'єднується, а мікросхема живиться від резервного джерела, споживаючи маленький струм. MC33388 відслідковує активність на шині та сигнал на вході WAKE. Якщо

досягаються умови виходу зі стану «Sleep», пристрій переходить у режим Standby VBAT і через вивід INH під'єднується основна напруга живлення. Режим Standby VBAT схожий на режим «Sleep», але на виході INH формується високий рівень, відповідно забезпечується під'єднання зовнішньої напруги живлення +5В.

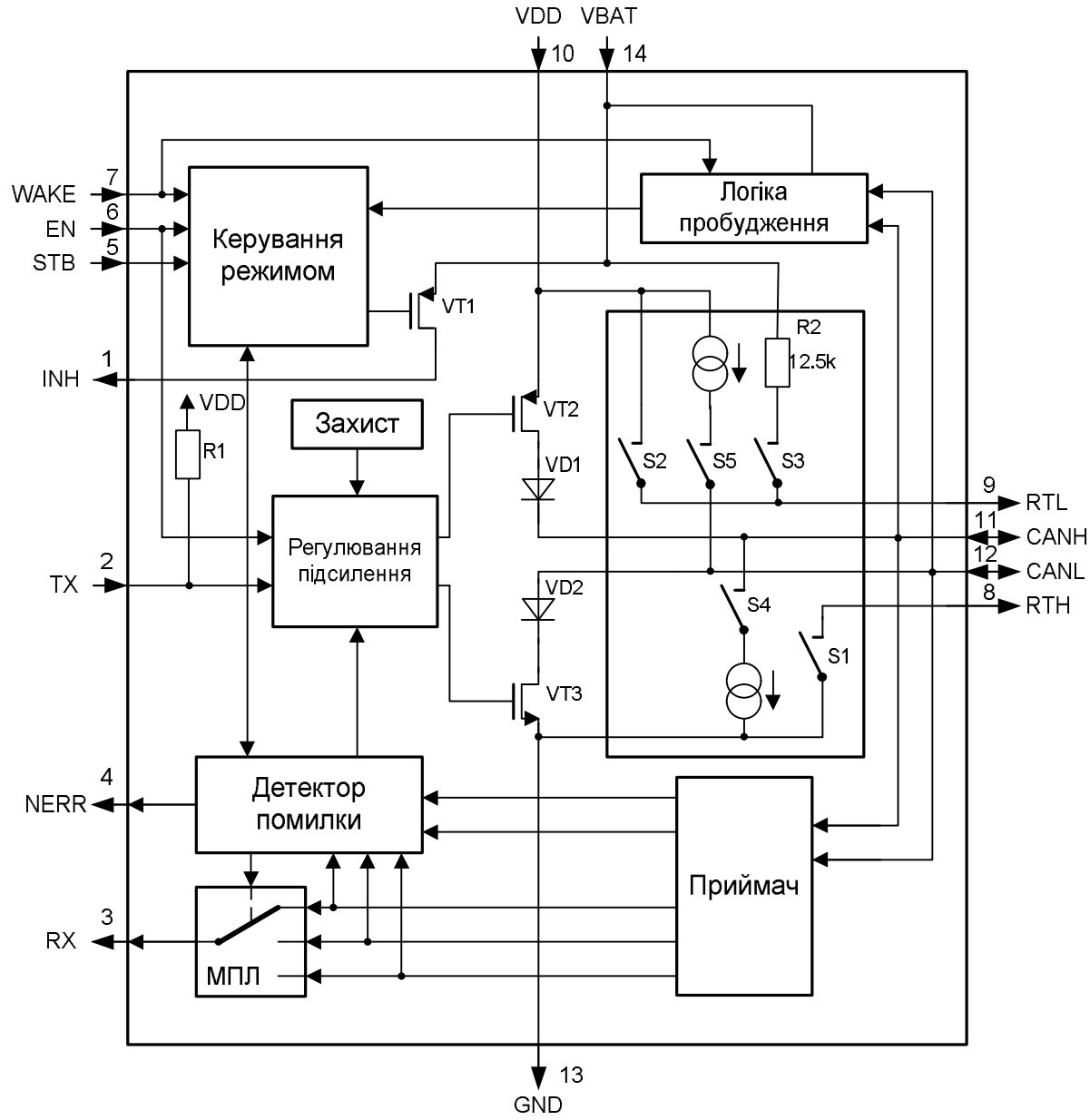


Рисунок 8.6 – Структура трансивера Motorola MC33388

Призначення виводів мікросхеми описані в таблиці 8.1.

Таблиця 8.1 – Призначення виводів трансивера MC33388

Назва	№	Опис
VBAT	14	Вхід для подачі резервної напруги живлення від акумулятора
VDD	10	Вхід для подачі напруги живлення +5В.
CANH	11	Вивід для під’єднання до лінії CAN_H.
RTH	8	Використовується для під’єднання зовнішнього кінцевого опору. Структура виходу RTH – низькорівневий перемикач, увімкнений у нормальному стані, вимикається при виникненні будь-яких збоїв. Коли лінія CAN_H відключена, CANH під’єднується до землі через зовнішній RTH-резистор (резистор R5 на рисунку 9.1).
CANL	12	Вивід для під’єднання до лінії CAN_L.
RTL	9	Використовується для під’єднання зовнішнього кінцевого опору. Структура виходу RTL – високорівневий перемикач, увімкнений у нормальному стані, вимикається при виникненні будь-яких збоїв. Коли лінія CAN_L від’єднана, CANL під’єднується до VDD через зовнішній RTL-резистор (резистор R4 на рисунку 9.1). У режимах Standby VBAT та Sleep вивід під’єднується до лінії VBAT через внутрішній перемикач та опір 12.5кОм.
STB	5	Входи використовуються для переведення пристрою в один із чотирьох робочих режимів. Ці ніжки є КМОН-сумісними та підключаються до виходів мікроконтролера.
EN	6	
INH	1	Вихід, який застосовується для керування зовнішнім стабілізатором напруги. Структура INH – високорівневий перемикач. Високий рівень напруги підтримується на виході в нормальному, тільки прийом, Standby VBAT режимах. У режимі Sleep на виході встановлюється низький рівень який від’єднує основне джерело живлення.

Продовження таблиці 8.1

WAKE	7	Вхід використовується для виводу пристрою з режимів Standby VBAT та Sleep. Зазвичай під'єднується до зовнішнього перемикача (рисунок 9.1). Стандартна напруга Vbat/2. Увага: вхід повинен залишатися під'єднаним . Якщо його функції не використовуються, необхідно під'єднати до землі.
TX	2	Вхід передавача. Зазвичай прямо під'єднується до виходу TXCAN мікроконтролера. Якщо на вході TX високий рівень – лінії CAN_H й CAN_L у рецесивному стані, інакше в домінантному.
RX	3	Вихід приймача. Повідомляє стан шини на вхід RXCAN мікроконтролера. На виході RX високий рівень, коли на шині рецесивний сигнал, і низький, коли домінантний.
NERR	4	Вихід який повідомляє про пошкодження шини. Коли на виході NERR високий рівень ніяких помилок немає.
GND	13	Вихід для підключення до землі.

9 ПРИКЛАД ПРИНЦИПОВОЇ СХЕМИ ВУЗЛА CAN-МЕРЕЖІ

Принципова схема вузла CAN-мережі, наведена на рисунку 9.1. Основним елементом схеми є сучасний мікроконтролер AT90CAN128, до складу якого входить CAN-модуль, що підтримує протокол CAN 2.0A/B, який описаний у розділі 4.

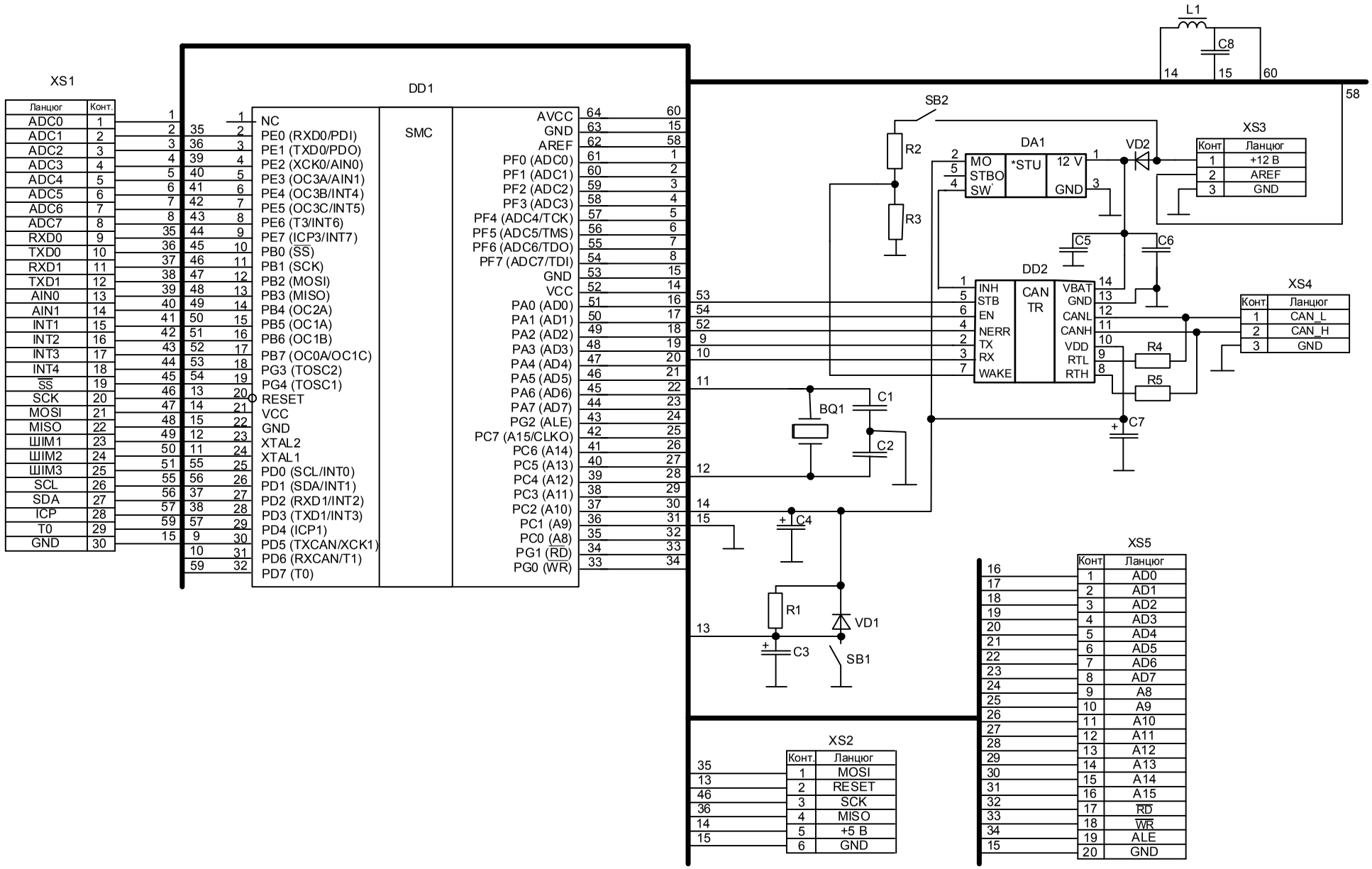


Рисунок 9.1 – Принципова схема вузла CAN-мережі

Лінії TXCAN й RXCAN мікроконтролера є відповідно виходом та входом CAN-модуля мікроконтролера. Вони підключаються до відповідних ліній трансивера, у якості якого використовується мікросхема MC33388 (DD2).

Мікроконтролер AT90CAN128 є досить потужним пристроєм, здатним виконувати безліч різноманітних функцій, пов'язаних з обробкою та керуванням обміном даними в різних протоколах з безліччю периферійних (зовнішніх) пристроїв.

Згідно з принциповою схемою, яка розглядається, окрім CAN-модуля, можна використовувати наступні модулі МК-ра:

- аналого-цифровий перетворювач (АЦП). Вхідні сигнали на АЦП подаються з контактів 1…8 роз'єму XS1 на входи ADC0…ADC7 МК-ра;
- універсальний асинхронний приймач-передавач1 (УАПП1) (лінії RXD0, TXD0);
- універсальний асинхронний приймач-передавач2 (УАПП2) (лінії RXD1, TXD1);
- аналоговий компаратор (АК) (лінії AIN0, AIN1);
- чотири зовнішніх переривання (лінії INT4, INT5, INT6, INT7);
- синхроний послідовний інтерфейс (SPI) (лінії $\overline{SS}$, SCK, MOSI, MISO);
- трьохканальний широтно-імпульсний модулятор (ШІМ) (лінії OC2A, OC1A, OC1B);
- синхроний послідовний інтерфейс (I^2C) (лінії SCL, SDA);
- таймер Т1, який використовується у режимі "Захоплення" (лінія ICP1);
- таймер Т0, який використовується для підрахунку зовнішніх подій (лінія Т0);
- лінії AD0… AD7, A8… A15, ALE, $\overline{RD}$, $\overline{WR}$ можуть використовуватися для підключення до МК-ра зовнішньої пам'яті даних, об'ємом 64 Кбайт.

Для керування передачею даних CAN-мережею на фізичному рівні, використовується трансивер (приймач-передавач типу «Fault Tolerant» (див. розділ 8).

У якості трансивера подібного типу в роботі обрана мікросхема Motorola MC33388 (DD2).

Мікросхема може керувати зовнішнім стабілізатором напруги (DA1) через вивід INH. Це дозволяє перемикати пристрій у режим зниженого енергоспоживання. Вихід з цього стану відбувається при виявленні активності на шині в «сплячому» режимі або здійснюється за допомогою ключа SB2.

Трансивер має чотири режими роботи, що програмно задаються: нормальний, тільки прийом, живлення від батарейки (Stanby VBAT) і сну (Sleep). Перехід в один з названих режимів здійснюється подачею керуючих сигналів на виводи STB та EN (DD2), які пов'язані з виходами PG3, PG4 порту G МК-ра (DD1).

У нормальному режимі всі функції трансивера доступні та вихід NERR трансивера, зв'язаний із входом PB7 МК-ра, сигналізує про пошкодження шини. У випадку відсутності пошкодження на виході NERR присутній високий рівень напруги (логічна 1).

У режимі «Тільки прийом» частина, яка передає, відключена, тому пристрій не керує шиною, а функції прийому здійснюються як у нормальному режимі.

В режимі «Sleep» функції передачі та прийому відключені. Вивід CANL через опір R4 і внутрішній опір 12,5 кОм (рисунок 8.6) підключений до VBAT. Зовнішній стабілізатор напруги (DA1) низьким рівнем напруги, що з'являється на виході INH (DD2), вимикається, і до входу VDD трансивера (DD2) не підводиться напруга живлення.

MC33388 відслідковує активність на шині й сигнал на вході WAKE, що може бути сформований за допомогою зовнішнього перемикача SB2. Якщо виконуються умови виходу зі стану «Sleep», пристрій переходить у режим Stanby VBAT, схожий на режим «Sleep», але на виході INH підтримується високий

рівень, відповідно забезпечується зовнішня напруга живлення для трансивера: +5В.

До виводів XTAL1, XTAL2 МК-ра підключається кварцовий резонатор (кварц), що задає тактову частоту роботи мікроконтролера f_{CLK}, яка дорівнює 16 МГц.

Для підвищення стабільності частоти, яка формується генератором тактових імпульсів, у схемі використовуються два конденсатори однакової ємності. Значення конденсаторів обрані відповідно до рекомендацій, які наведені в [5] і становлять 30 пФ.

Одним з функціональних елементів розглянутої принципової схеми є вузол скидання, що формує зовнішнє скидання ($\overline{\text{RESET}}$), який подається на вхід RESET МК-ра у нормальному режимі роботи.

Основним елементом цього вузла є інтегруючий RC-ланцюг, виконаний на резисторі R1 і конденсаторі C3. В початковому стані (до подачі напруги живлення) ємність C3 розряджена. При увімкненні напруги VDD=+5В, ця ємність не може миттєво зарядитися і на вхід RESET (СКИДАННЯ) мікроконтролера надходить низький рівень напруги. Через заряджання конденсатора C3 через резистор R1 напруга на вході RESET збільшується. В середині МК-ра на вході лінії RESET включений пороговий елемент (тригер Шмітта), що спрацьовує після того, як напруга на конденсаторі досягне порогу спрацьовування цього тригера. У момент спрацьовування тригера дія сигналу «$\overline{\text{RESET}}$» припиняється і контролер переходить у нормальний режим роботи.

Для виконання операції «СКИДАННЯ» МК-ра при увімкненій напрузі живлення VDD необхідно короткочасно натиснути кнопку SB1. При цьому ємність C3 швидко розрядиться й при відпусканні кнопки повторно відбудеться скидання МК-ра. Діод VD1 служить для швидкого розряду ємності C3 при вимиканні напруги живлення та розімкнутій кнопці SB1. Цей розряд здійснюється через внутрішній опір джерела живлення.

10 ТЛУМАЧНИЙ СЛОВНИК ОСНОВНИХ ЗАГАЛЬНОПРИЙНЯТИХ ТЕРМІНІВ CAN-ТЕХНОЛОГІЙ

У додатку З наведений тлумачний словник основних загальноприйнятих термінів CAN-технологій [6].

11 МОДЕЛЮВАННЯ CAN-ІНТЕРФЕЙСУ

11.1 Необхідне програмне забезпечення

Для моделювання CAN-інтерфейсу знадобиться наступне програмне забезпечення:

- AvrStudio4.18 (AvrStudio версії 18) Офіційний сайт з якого можна загрузити програмне забезпечення: http://www.atmel.com/forms/software_download.asp?category_id=163&family_id=607&subfamily_id=760&fn=dl_AvrStudio4Setup.exe

Якщо через несумісність AvrStudio4.18 з Windows Vista дане програмне забезпечення працювати коректно не буде тоді необхідно видалити встановлений AvrStudio4.18, завантажити та встановити наступні програмні засоби:

- AvrStudio4.16 (AvrStudio версії 16: http://www.atmel.com/dyn/resources/prod_documents/AvrStudio416Setup. exe);
- AvrStudio4.18SP1 (перший патч оновлення AvrStudio версії 16 до версії 18): http://www.atmel.com/dyn/resources/prod_documents/AVRStudio4.18SP1. exe);
- AvrStudio4.18SP2 (другий патч оновлення AvrStudio версії 16 до версії 18: http://www.atmel.com/dyn/resources/prod_documents/AVRStudio4.18SP2. exe);
- AvrStudio4.18SP3 (третій патч оновлення AvrStudio версії 16 до версії 18: http://www.atmel.com/dyn/resources/prod_documents/AVRStudio4.18SP3. exe).

Далі потрібно встановити наступні програми:

- AT90can128 Plug-in (плагін установки мікроконтролера AT90CAN128): http://www.atmel.com/dyn/resources/prod ... n_v106.zip, або можна скористатися пошуковою системою Google, передавши назву файлу at90can128_plugin_v106);
- WinAvr (для встановлення плагіну AVR GCC – компілятора мови C): http://sourceforge.net/projects/winavr/files/WinAVR/20100110/WinAVR-20100110-install.exe/download).

Примітка 1. **AvrStudio 4.16 (4.18)** — інтегроване програмне середовище (IDE) для розробки 8 - бітових застосувань AVR від компанії Atmel, що працює в операційних системах Windows NT/2000/XP/Vista/7. AvrStudio містить асемблер і засоби, що дозволяють відстежити виконання програми. Поточна версія підтримує всі контролери AVR і інструменти розробки, що випускаються на сьогоднішній день. AvrStudio містить в собі менеджер проектів, редактор коду, інструменти віртуального імітування і внутрішньосхемного налагодження, дозволяє писати програми на асемблері або на C/C++. Тому AvrStudio може вважатися зручним середовищем розробки програм для мікроконтролерів AVR.

Примітка 2. **Плагін** — додаток до вже встановленого програмного засобу, незалежно компільований програмний модуль, що динамічно підключається до основної програми та призначений для розширення або використання її можливостей.

Примітка 3. **Патч** — окремий програмний модуль, що використовується для усунення проблем в програмному забезпеченні або зміни його функціонала. Виправлення може застосовуватися до вже встановленої програми, або до її вихідних кодів. Сюди входить виправлення помилок, зміна зовнішнього вигляду, поліпшення ергономічності або продуктивності програм, а також будь-які інші зміни.

Для встановлення plug-in AT90can128 потрібно подвійним натиском запустити інсталяційний завантажувальний файл з сайту www.atmel.com (або будь-якого іншого сайту).

11.2 Створення проекту

Для створення проекту необхідно виконати описані нижче дії:

1) Запустити AvrStudio (рисунок 11.1)

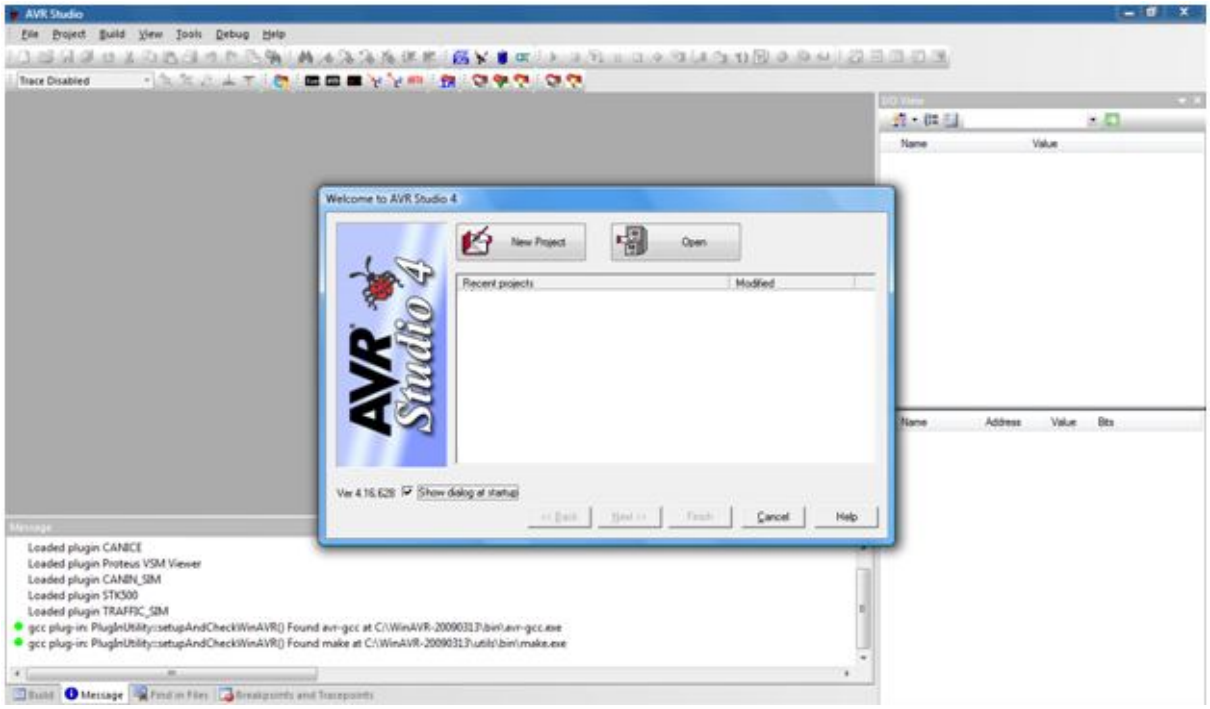


Рисунок 11.1 – Вікно AvrStudio, який був запущений

2) Для створення нового проекту натиснути New Project (рисунок 11.2).
3) При програмуванні мовою С вибрати тип проекту AVR GCC (рисунок 11.3). Бачимо, що в полі Location відображається поточна директорія – C:\Users\Owner\Documents\Project. Директорія може бути змінена при бажанні користувача.

Примітка 1. В діалоговому вікні на рисунку 11.3 можна вибрати тип проекту: Atmel AVR Assembler – для написання програм на асемблері,
та AVR GCC – для програмування на С (саме для написання програм на С необхідно встановити програму WinAvr).

4) Ввести назву проекту Project name (рисунок 11.4). Назвемо цей проект AVR_CAN128. Слід зазначити що в залежності від Project name, файл лістингу програми (рисунок 11.8) буде автоматично названо, як в Project name.

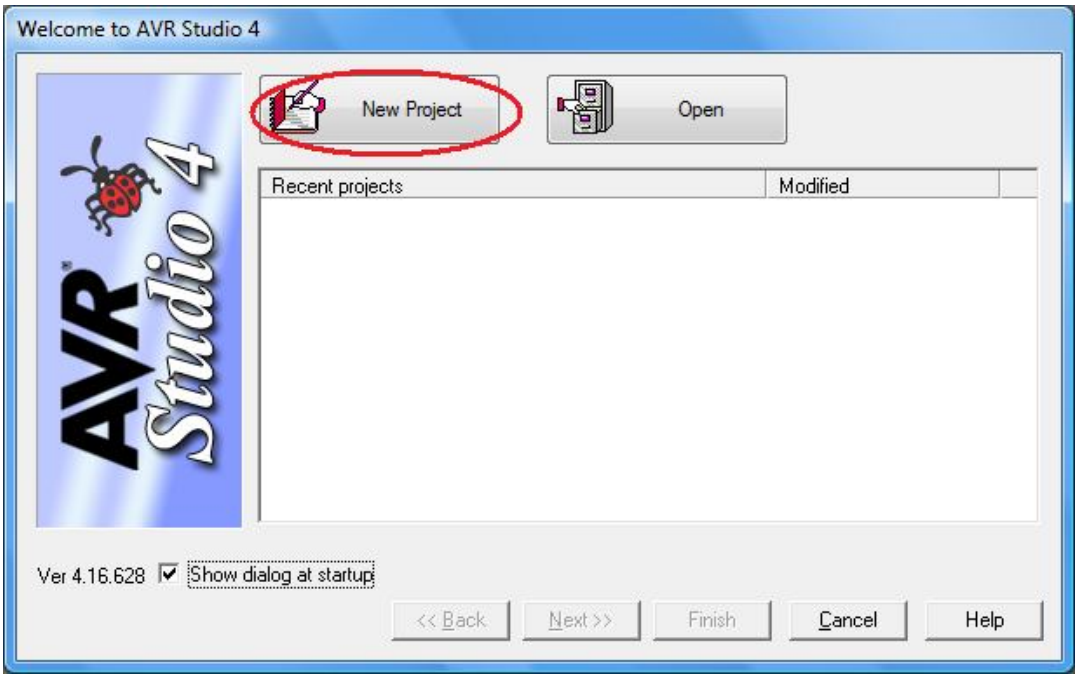


Рисунок 11.2 – Створення нового проекту

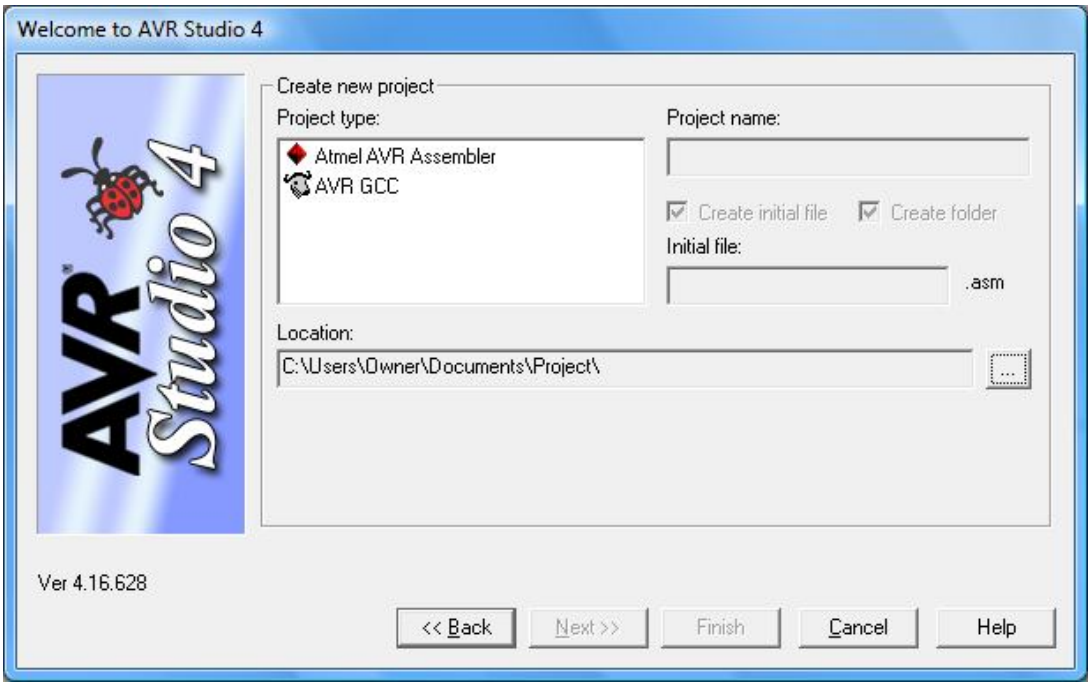


Рисунок 11.3 – Вибір необхідного типу проекту при програмуванні мовою С

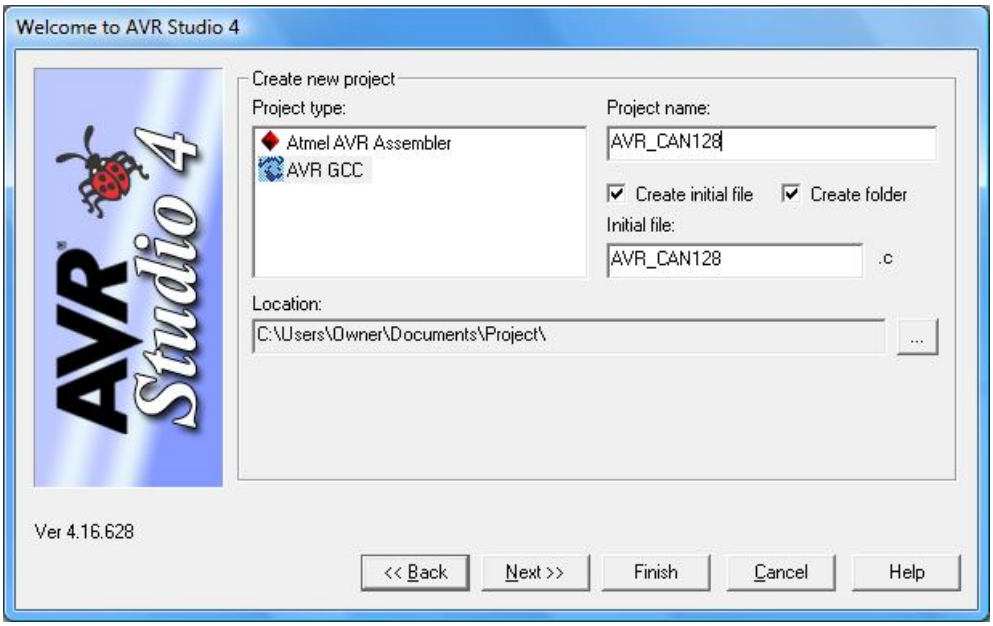


Рисунок 11.4 – Введення назви проекту

5) У лівому вікні (Debug platform) вибрати платформу AVR Simulator, а у правому – контролер (Device) – AT90CAN128 (рисунок 11.5).

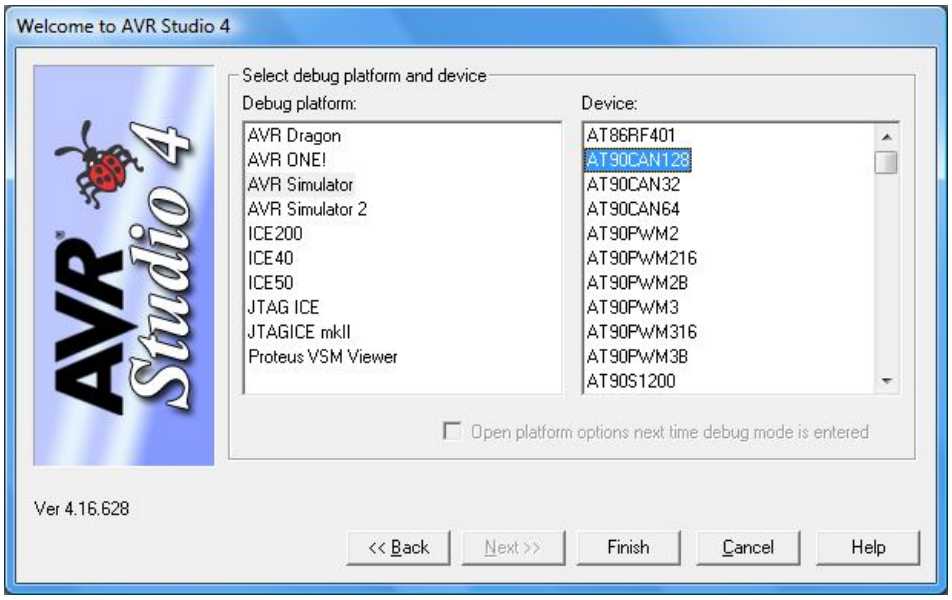


Рисунок 11.5 – Вибір платформи та контролера, який будемо симулювати

Примітка 2. У діалоговому вікні на рисунку 11.5 пропонується вибрати налагоджувальну платформу і модель мікроконтролера, для якої розроблятиметься програма. Налагоджувальною платформою може бути як програматор – емулятор (AVR Dragon, AVR One, ICE 200 і т. ін.), так і віртуальний налагоджувач – AVR Simulator, що дозволяє налагоджувати програму засобами AVR Studio. Якщо на комп'ютері встановлена САПР Proteus

компанії Labcenter Electronics, то як платформа може бути вибраний Proteus VSM Viewer. Це означає, що можна в Proteus створити схему пристрою, для якого розробляється програма, а потім цей віртуальний пристрій підключити до AVR Studio і перевірити на ньому роботу програми.

Для закінчення етапу «створення проекту» натиснути кнопку Finish (рисунок 11.6).

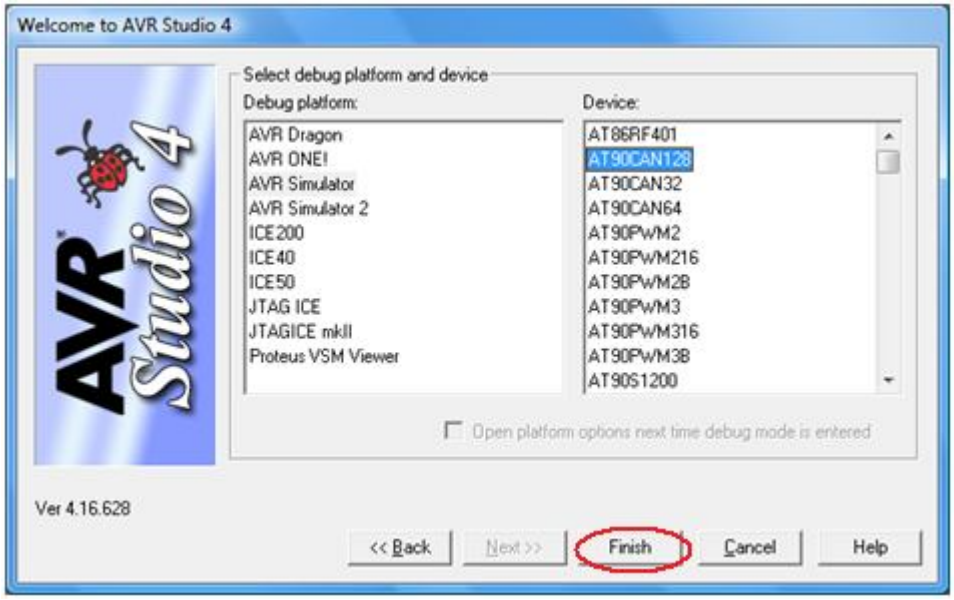


Рисунок 11.6 – Закінчення етапу «створення проекту»

6) До файлу AVR_CAN128.c (назва файлу має такий самий вигляд як і назва проекту) скопіювати програму наведену в ДОДАТКУ К. Для цього 2 рази натиснути AT90CAN128.c та переписати до файлу програму, написаною мовою С (рисунок 11.8).
7) Програму потрібно скомпілювати, натиснувши кнопку Build Active Configuration (рисунок 11.7) або F7.

Рисунок 11.7 – Етап компіляції програми

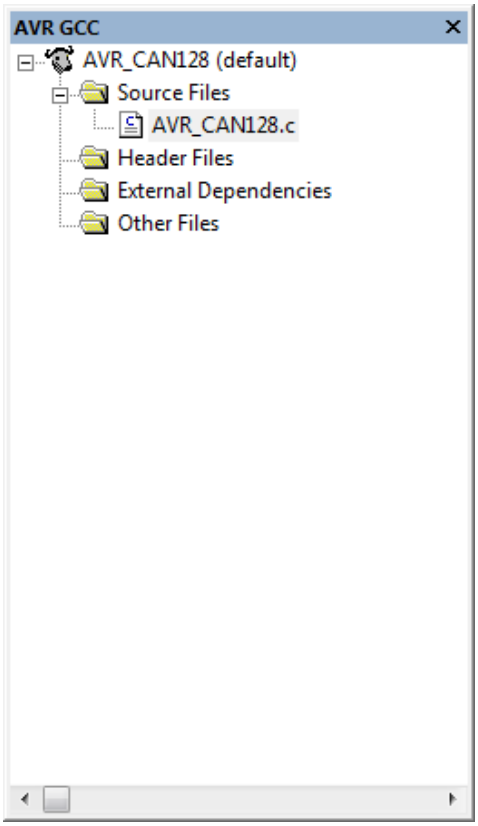


Рисунок 11.8 – Копіювання лістингу програми в файл AVR_CAN128.c

Плагін для роботи з CAN-модулем оформлений у вигляді панелей, де для нас особливий інтерес представляють наступні панелі:

- MAILBOX_SIM, де можна спостерігати поточний стан Mob (рисунок 11.9);
- TRAFFIC_SIM, що дає можливість переглянути присутні в CAN-мережі повідомлення (рисунок 11.10);
- CANIN_SIM. У цій панелі подвійним натисканням лівої кнопки миші на повідомленні можна імітувати його надсилання контролеру через CAN-інтерфейс. Список доступних для надсилання повідомлень від середовища моделювання до CAN-модуля МК-ра задається у файлі “C:\Program Files\Atmel\AVR Tools\AvrStudioPlugin\ AT90CAN128.canin.txt” (рисунок 11.12). Наведений список можна розширювати.

Текстовий файл *canin.txt,* з яким працює програма керування CAN-модулем, запропонована у цьому посібнику, має вміщувати такі рядки:

```
#ID      DLC  Data                    CycleStart    CycleRepeat     Error
12CEx;   4;   11 22 33 44;             180;          0;              ;
0123x;   4;   11 22 33 44;             180;          0;              ;
22Bx;    8;   10 0E 21 22 23 24 25 26; 0;            0;              ;
22Bx;    8;   21 27 28 29 30 31 32 33; 0;            0;              ;
22Bx;    8;   31 34 FF FF FF FF FF FF; 0;            0;              ;
```

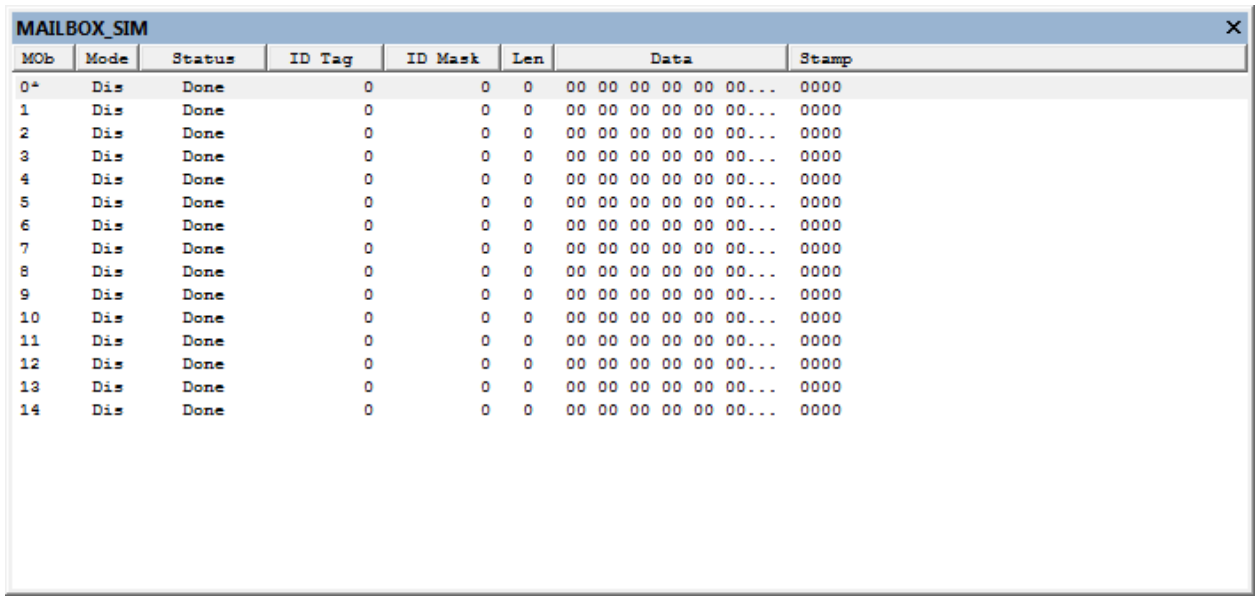


Рисунок 11.9 – Вигляд панелі MAILBOX_SIM

Основні позначення на рисунку 11.9 мають наступний зміст:

- Mob – номер Mob (всього кількість Mob дорівнює 15);
- Mode – режим передачі. Поле Mode може приймати наступні значення:
    1) dis – скорочення від disable, означає, що режим не встановлено;
    2) Rx – режим «прийом»;
    3) Tx – режим «передача»;
- Status – статус повідомлення. Поле Status може приймати наступні значення:

1) Pending – очікується прийняття повідомлення або віддаленого запиту;
2) Done – повідомлення прийнято або передано;
3) N% processed – передано N-процентів повідомлення;

- ID Tag – ідентифікатор повідомлення;
- ID Mask – маска ідентифікатора повідомлення;
- Len – довжина повідомлення;
- Data – поле даних повідомлення;
- Stamp – відмітка часу.

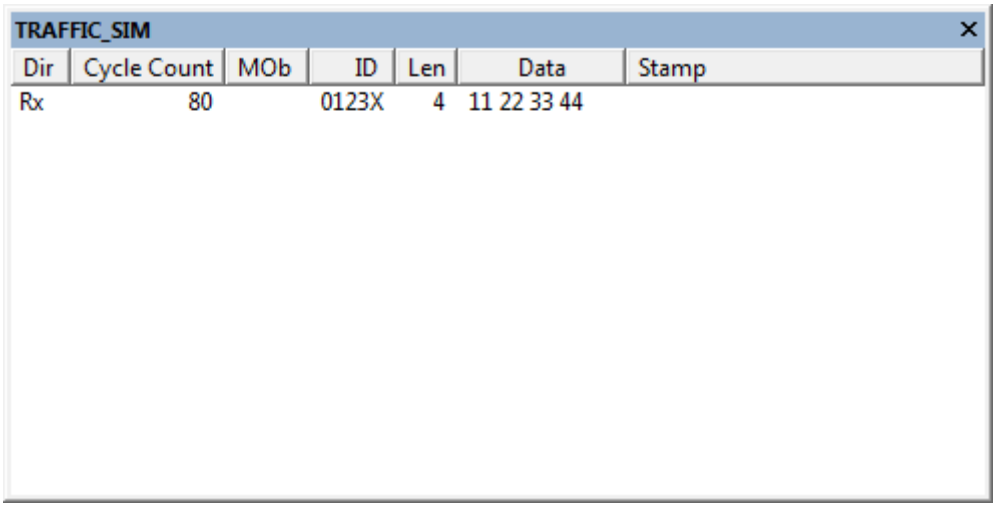


Рисунок 11.10 – Вигляд панелі TRAFFIC_SIM

Основні позначення на рисунку 11.10 мають наступний зміст:

- Mob – номер Mob;
- ID – ідентифікатор повідомлення.
- Len – довжина повідомлення;
- Data – поле даних повідомлення;
- Dir – направлення передачі;
- Cycle Count – лічильник циклів, які МК-р витратив на передачу або прийом повідомлення;
- Stamp – відмітка часу.

Якщо автоматично при створені проекту панель TRAFFIC_SIM закрита необхідно в головному меню натиснути кнопку Tools, а потім послідовно вибрати: CAN Simulator -> View CAN Simulator Traffic (рисунок 11.11).

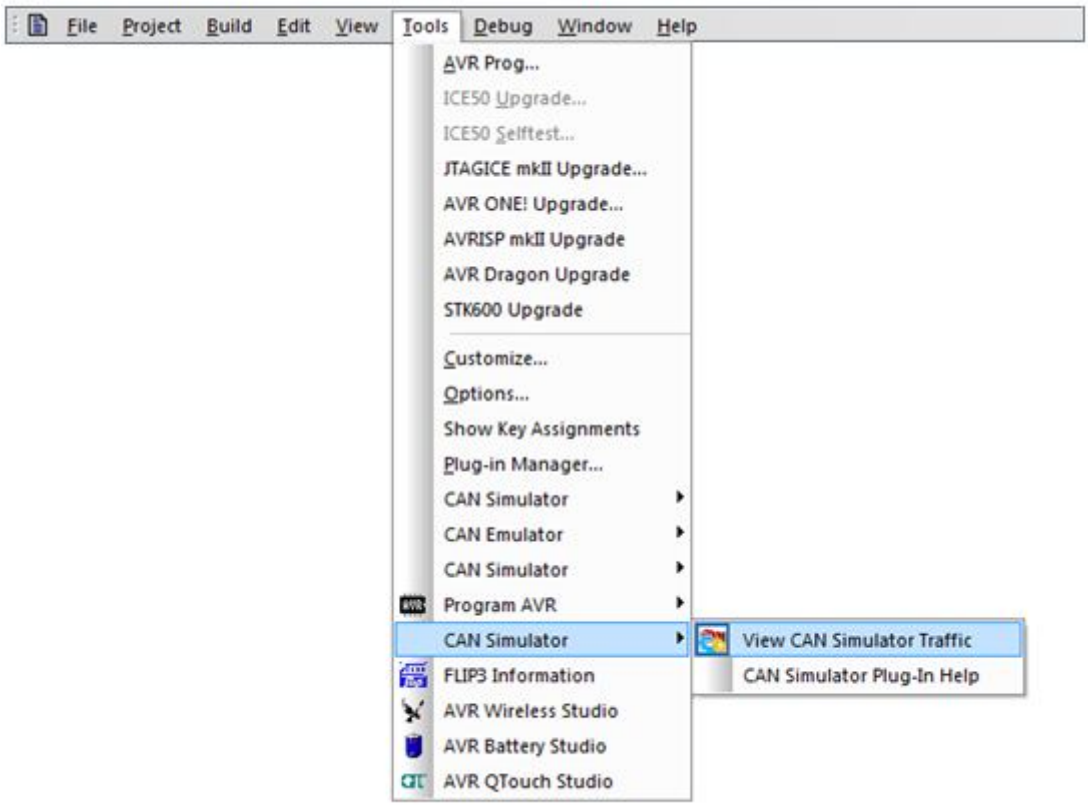


Рисунок 11.11 - Відкриття панелі TRAFFIC_SIM через головне меню

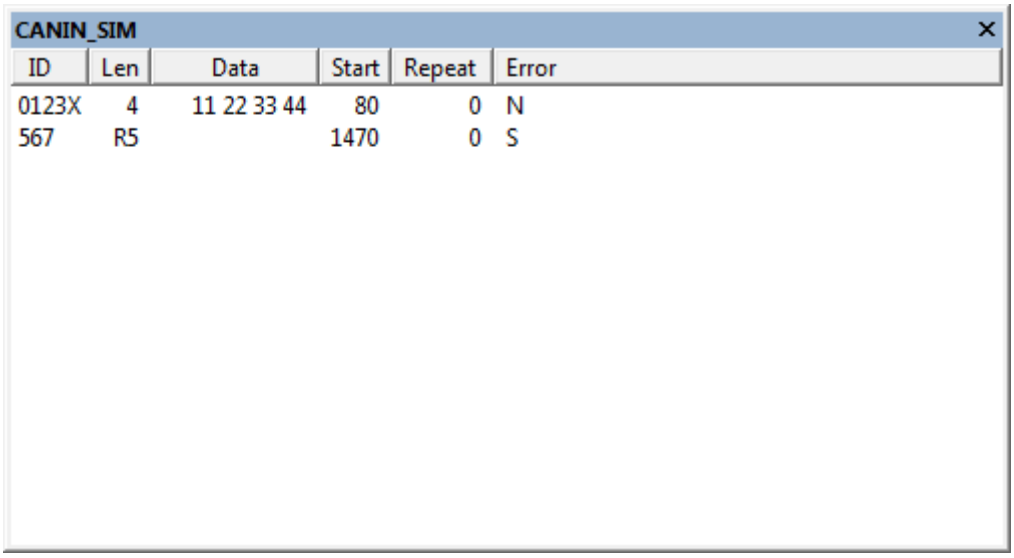


Рисунок 11.12 – Вигляд панелі CANIN_SIM

Основні позначення на рисунку 11.12 мають наступний зміст:

- ID – ідентифікатор повідомлення.
- Len – довжина повідомлення;

- Data – поле даних повідомлення;
- Repeat – кількість повторних передач;
- Start – час у кількості машинних циклів від запуску симуляції до передачі повідомлення через CAN-інтерфейс;
- Error – тип помилки або її відсутність (N).

Якщо автоматично при створені проекту панель CANIN _SIM закрита необхідно в головному меню натиснути кнопку Tools, а потім послідовно вибрати: CAN Simulator -> View CAN Input Frames (рисунок 11.13).

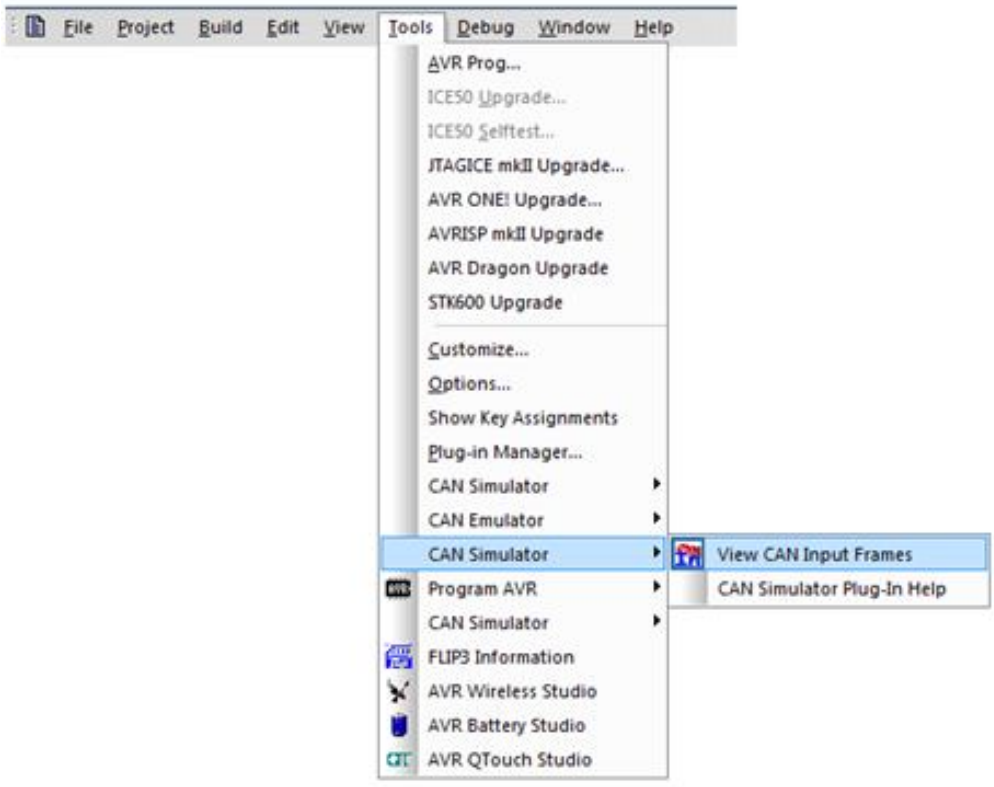


Рисунок 11.13 – Відкриття панелі CANIN _SIM через головне меню

11.3 Процес моделювання

Змоделюємо ситуацію, коли ми маємо один з модулів CAN-мережі (контролер, який підтримує CAN-інтерфейс), та необхідно прийняти та передати дані. Вважається, що дані ми отримуємо за допомогою давачів. Слід зазначити,

що на даний момент для САПР Proteus не існує моделі контролера, який би імітував CAN інтерфейс.

Продемонструємо роботу одного контролера з підтримкою CAN-інтерфейсу (інші контролери CAN-мережі працюють аналогічним чином) на основі моделі в AVR Studio.

До початку моделювання вміст панелі MAILBOX_SIM показаний на рисунку 11.14.

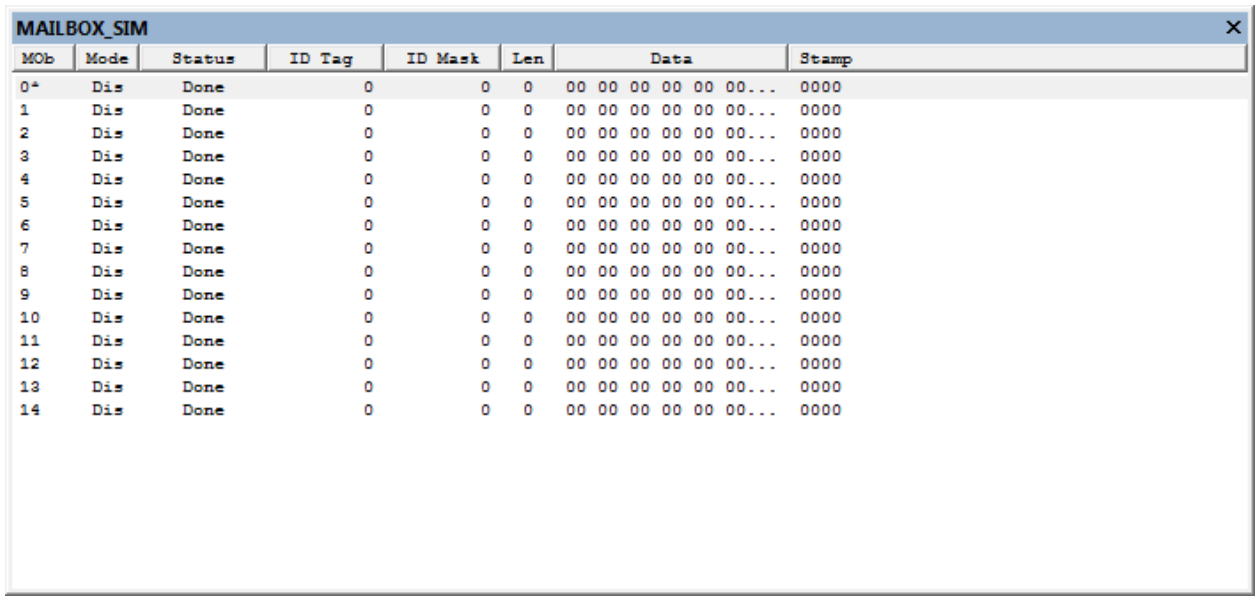


Рисунок 11.14 – Вміст панелі MAILBOX_SIM до початку виконання програми

Щоб розпочати моделювання необхідно в головному меню натиснути кнопку Debug, а потім вибрати Start Debugging (рисунок 11.15).

Слід зазначити, що виконання однієї команди на мові С моделюється та здійснюється за одне натискання клавіші F11. Отож, щоб перейти до наступної команди треба натиснути F11. Потрібно зазначити, що одній команді на мові С може відповідати декілька команд асемблера. Якщо користувач бажає не виконувати одну операцію за один натиск F11, можливо скористатися клавішою F10, яка дозволяє за один натиск виконати всю функцію. Процес прийому або передачі та виконання програми – паралельні. Це означає що динаміку прийому/передачі можливо буде побачити натискаючи F10.

Якщо панель MAILBOX_SIM закрита необхідно в головному меню натиснути кнопку Tools, а потім послідовно вибрати: CAN Simulator -> Edit CAN Simulator Mailbox (рисунок 11.16).

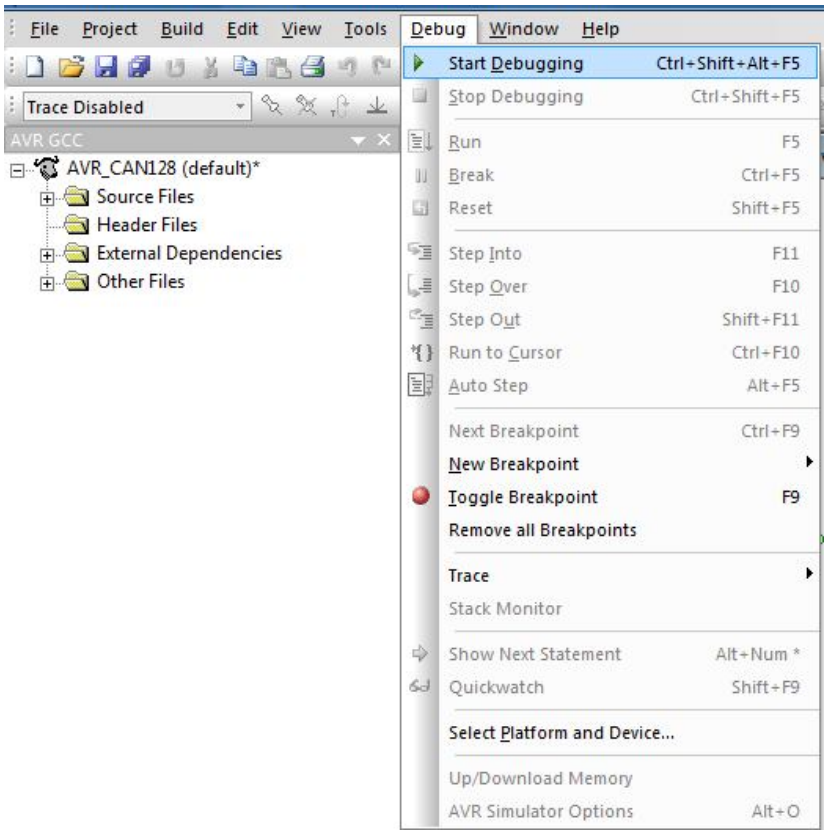


Рисунок 11.15 – Початок моделювання

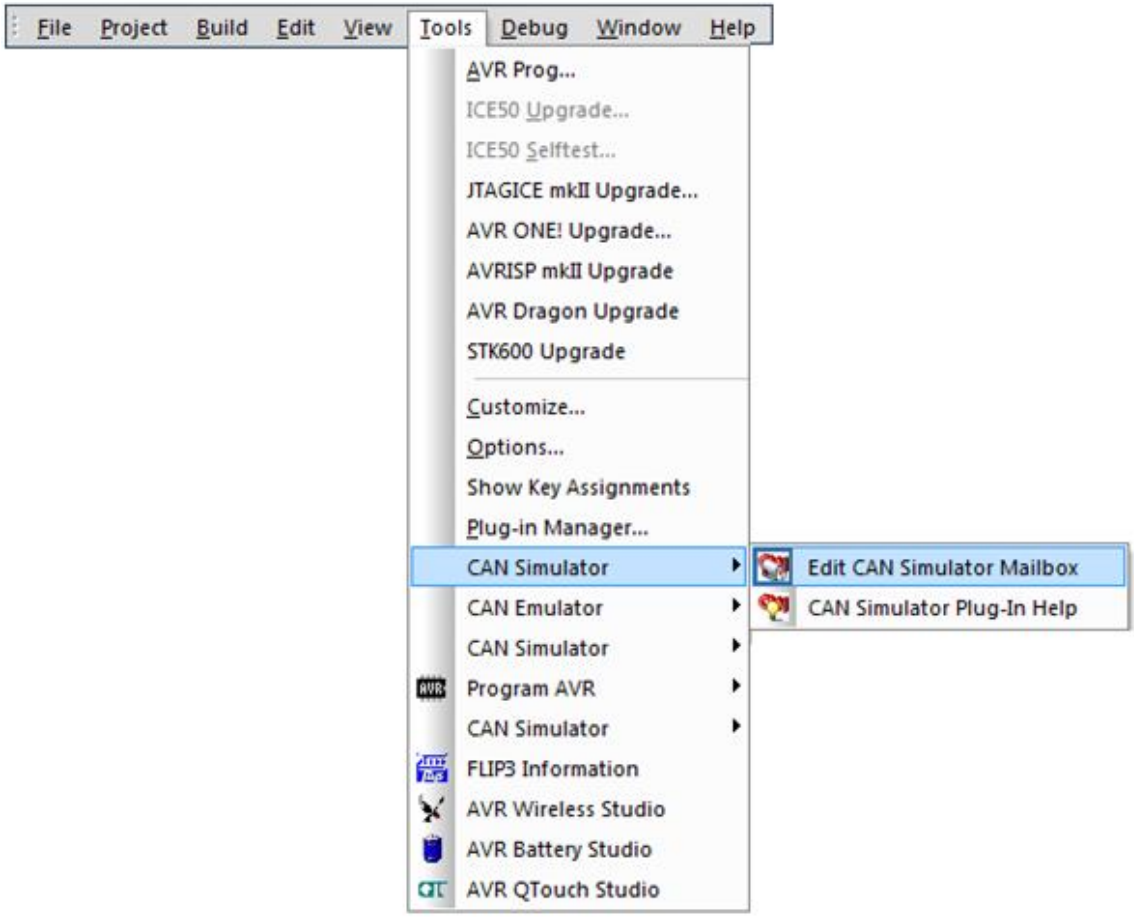


Рисунок 11.16 - Відкриття панелі MAILBOX_SIM через головне меню

У середовищі програмування AVRStudio є можливість спостерігати за вмістом усіх внутрішніх регістрів модельованого мікроконтролеру. У рамках цього посібника нас цікавлять регістри, що відповідають за керування CAN-модулем.

Щоб відкрити панель вмісту внутрішніх регістрів, необхідно в головному меню натиснути кнопку View і у меню Toolbars відмітити пункт I/O. В інтерфейсі середовища розробки відкриється панель I/O View, яку можна побачити на рисунку нижче:

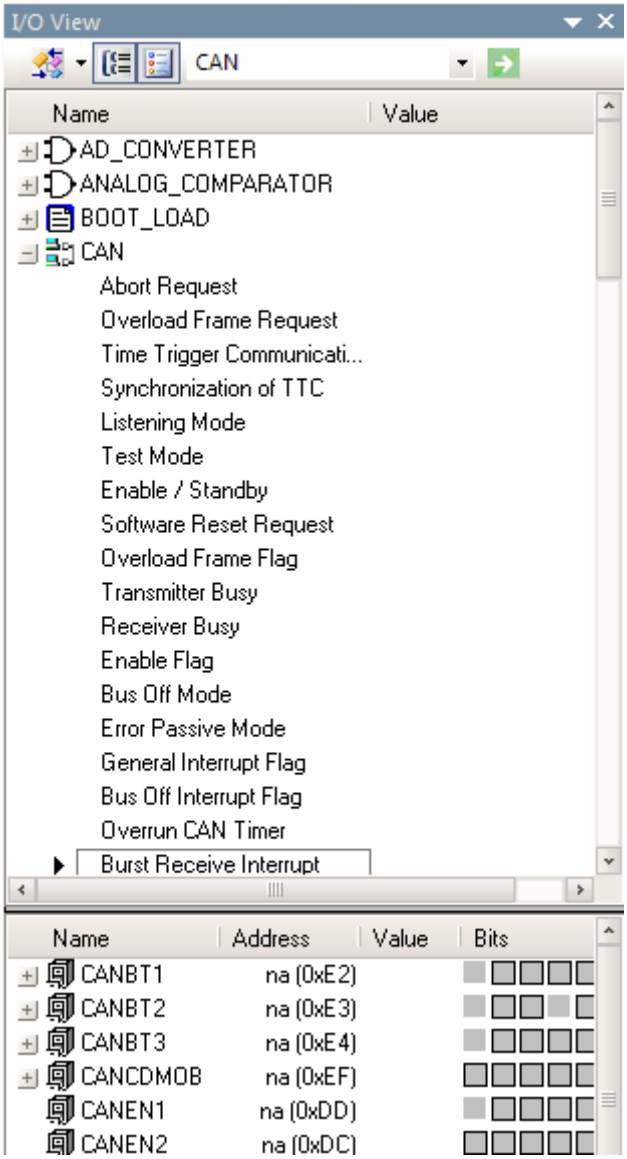


Рисунок 11.17 – Вигляд панелі I/O View з обраним CAN-модулем

У верхній частині панелі I/O View представлені всі доступні модулі модельованого мікроконтроллера. Необхідний модуль можна вибрати або натиснувши на знак «+» біля його назви у переліку модулей, або з випадаючого списку зверху. При цьому буде розгорнуто список усіх прапорців і регістрів під назвою обраного модуля. У зазначеному списку представлені логічні назви компонентів модуля, а якщо компонент є прапорцем, то і його значення (встановлено/скинуто). Нижня частина панелі I/O View надає доступ до вмісту регістрів обраного модуля. Тут зазначені назви регістрів та значення бітів у них. Під час покрокового виконання програми можна спостерігати, як кожна інструкція змінює вміст регістрів, при цьому дані, що було змінено, підсвічуються

червоним. Вміст регістрів можна змінювати у процесі покрокового виконання програми незалежно від алгоритму. Необхідно зазначити, що змінене значення регістру буде доступне лише у наступному після зміни кроці покрокового виконання.

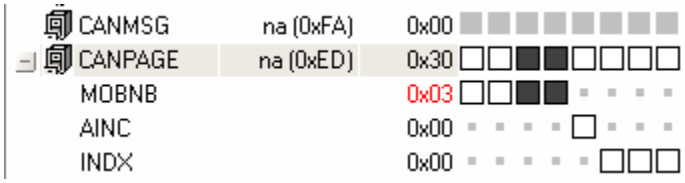


Рисунок 11.18 – Підсвітка зміненого регістру CANPAGE у панелі I/O View

Середовище AVRStudio також дозволяє проглядати вміст пам'яті даних, програм та зовнішньої пам'яті під час покрокового виконання програми. Щоб відкрити панель Memory необхідно на панелі інструментів натиснути на кнопку Toggle Memory Window або Alt+4:

Рисунок 11.19 – Кнопка Toggle Memory Window на панелі інструментів

У панелі Memory можна вибирати необхідну область пам'яті з випадаючого списоку з усіма доступними типами пам'яті, а також побачити у змінені області за допомогою підсвітки червоного кольору, подібно до описаної раніше панелі I/O View. Є можливість також змінювати вміст пам'яті розміром в один байт, після подвійного натискання на обрану комірку пам'яті.

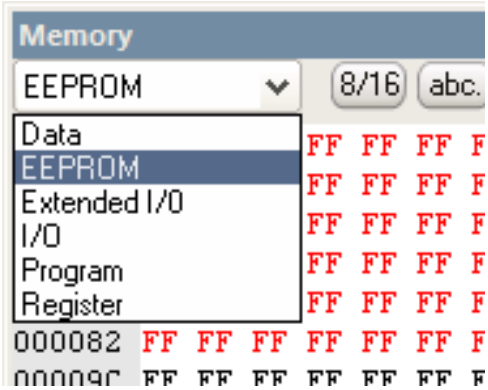


Рисунок 11.20 – Доступні у панелі Memory типи пам'яті та підсвітка змінених областей

Розглянемо основні частини програми, яка знаходиться в ДОДАТКУ К. Її виконання починається з функції int main(void). Слід зазначити що будь – яка програма на мові С починається з функції main . Запис int main(void)означає, що ми в функцію не передаємо жодних параметрів, але функція повертає число типу int (ціле число від 0 до 65535).

Головна функція main викликає шість додаткових функцій:

– void init_can(void) (ініціалізація CAN – модуля та скидання всіх Mob). Дана функція виконує наступні дії:

1) встановлює швидкість інтерфейсу та параметри імпульсів:

```
  //Блок 1.1
CANGCON = (1 << SWRES);

/* Швидкість інтерфейсу та параметри імпульсів. Блок 1.2 */
CANBT1 = 0x01;
CANBT2 = 0x02;
CANBT3 = 0x03;
//Блок 1.5
CANTCON = 0x00;
```

2) очищає регістри статусу, керуючі та DLC усіх MOb:

```
/*
   * Очистити регістри статусу, керування та DLC для усіх MOb. Це
   * робити обов'язково, оскільки ці біти значень за замовчанням не мають. Блок
1.6 */

for (uint8_t mob = 0; mob < 15; mob++) {
  CANPAGE = mob << 4;
  CANSTMOB = 0;
  CANCDMOB = 0;
}
```

3) встановлює активний режим роботи CAN-модуля:

```
/* Встановити активний режим роботи CAN-модуля. Блок 1.7*/
  CANGCON |= (1 << ENASTB);
```

4) дозволяє переривання завершення прийому та передачі:

```
/*
   * Дозволити переривання завершення прийому та передачі. Помилки
   * оброблювати ми не будемо.
   */
```

```
//Блок 1.4
   CANGIE = (1 << ENIT) | (1 << ENTX) | (1 << ENRX) | (1 << ENBX);
```

5) очищає регістри дозволу та статусу переривань:

```
/*
    * Заборонити переривання від усіх MOb, очистивши регістри
    * дозволу переривань. Блок 1.3*/
   CANIE2 = 0x00;
   CANIE1 = 0x00;
```

- void set_frame_buffer(CAN_buffer *buf) – функція для підготовки обраних Mob'ів до прийому мультикадру;
- sei() (функція, яка дозволяє переривання). Дозвіл переривань шляхом установки маски дозволу глобального переривання. Конктерні переривання під час прийому та передачі дозволяються в функції;
- send_multiframe() – функція передачи мультикадру – три послідовних повідомлення з ідентифікатором 0x0111, що містять 24 байти даних;
- void set_mob0_rtr_tx(void) (функція налаштування MOb0 на автоматичну відповідь на кадр віддаленого запиту);
- void set_mob1_rx(void) (функція налаштування MOb1 на прийом);
- void set_mob2_rx(void) (функція налаштування MOb2 на прийом);
- void set_3_mobs_tx(CAN_message *message) (функція, яка надсилає 3 повідомлення, з 3 різних мобів);

Функція main в нашій програмі на мові С має наступний вигляд:

```
/*
 * Головна програма.
 */
int main(void)
{
 /* Ініціалізація CAN. */
   init_can(); //Блок 1

   /*
    * Підготовка буфера для прийому мультикадру.
    * Зчитування прийнятого мультикадру відбувається в обробнику
    * переривань від CAN-модуля.
    */
```

```
    /* Обрання Mob'ів для прийому мультикадра */
    buffer.buffered_mobs = (uint16_t) (1 << 6) | (1 << 7)| (1 << 8); //Блок 2
    /* Встановлення ідентифікатору повідомлень мультикадра*/
    buffer.id = MULTIFRAME_ID;
    /* Ініціалізація Mob'ів для прийому мультикадра */
    set_frame_buffer(&buffer);

    /* Дозволити переривання. */
    sei();

    /*
     * Передача мультикадру - три послідовних повідомлення
     * з ідентифікатором 0x0111, що містять 24 байти даних.
     * Передача мультикадру буде вважатись завершеною, коли
     * буде передано повідомлення з останнього MOb'a.
     */
    send_multiframe();

  /* Підготовка MOb до прийому. */
    set_mob0_rtr_tx();
    set_mob1_rx();
    set_mob2_rx();

    /* Сворення нового повідомлення. */
    tx_message.id = 0x0001;
    tx_message.length = 3;
    tx_message.data[0] = 0xaa;
    tx_message.data[1] = 0xbb;
    tx_message.data[2] = 0xcc;
    /* Надіслати повідомлення з MOb3. */
    set_mob3_tx(&tx_message);

     /* Нескінченний цикл. */
    while ((CANGSTA & 4) == 4) {
       uint8_t a = 3;
    }
}
```

Розглянемо функцію set_MOb0_rtr_tx() (функцію налаштування MOb0 на автоматичну відповідь).

Відповідь (кадр даних) на кадр віддаленого запиту може бути автоматично відправлений після прийому очікуваного кадру віддаленого запиту, якщо біт RPLV регістра CANCDMOB встановлений у одиницю.

Перед відправкою повідомлення деякі поля MOb повинні бути ініціалізовані:

- IDT (регістри CANIDT1…CANIDT4) – 29 бітний ідентифікатор;

```
/*
```

```
 * Функція налаштування MOb0 на автоматичну відповідь.Блок 2
 */
void set_mob0_rtr_tx(void)
{
   uint8_t mob = 0;
   /* Зберегти поточний MOb. Блок 2.1 */
   uint8_t save_canpage = CANPAGE;

   /* Встановити MOb0 у якості поточного. */
   CANPAGE = mob << 4;
   /* Встановити ідентифікатор та біт RTR. Блоки 2.3-2.11 */
   CANIDT4 = (uint8_t) ((MOB0_ID << 3) | (1 << RTRTAG));
   CANIDT3 = (uint8_t) (MOB0_ID >> 5);
   CANIDT2 = (uint8_t) (MOB0_ID >> 13);
   CANIDT1 = (uint8_t) (MOB0_ID >> 21);
   /* Встановити маску для перевірки IDE, біта RTR та ідентифікатора. */
   CANIDM4 = (uint8_t) ((MOB0_MASK << 3) | (1 << RTRMSK) | (1 << IDEMSK));
   CANIDM3 = (uint8_t) (MOB0_MASK >> 5);
   CANIDM2 = (uint8_t) (MOB0_MASK >> 13);
   CANIDM1 = (uint8_t) (MOB0_MASK >> 21);

   /*
    * Записати дані повідомлення у буфер.
    * Оскільки біт CANPAGE.AINC скинуто
    * (тобто встановлено автоінкремент
    * номеру байта в буфері MOb),
    * послідовно записати у регістр CANMSG
    * до 8-ми байтів даних. Блок 2.13-2.15
    */
   for (uint8_t i = 0; i < MOB0_DLC; i++) {
      CANMSG = mob0_data[i];
   }
   /* Дозволити переривання для MOb0. Блок 2.18*/
   CANIE2 |= (1 << IEMOB0);
   /* Встановити режим прийому, автовідповіді, IDE та довжину повідомлення. Блоки 2.12,
2.16, 2.19*/
   CANCDMOB = (1 << CONMOB1) | (1 << RPLV) | (1 << IDE) | (MOB0_DLC << DLC0);
   /* Відновити поточний MOb. */
   CANPAGE = save_canpage;
}
```

Після виконання функції set_MOb0_rtr_tx()вміст панелі MAILBOX_SIM зміниться як показано на рисунку 11.21.

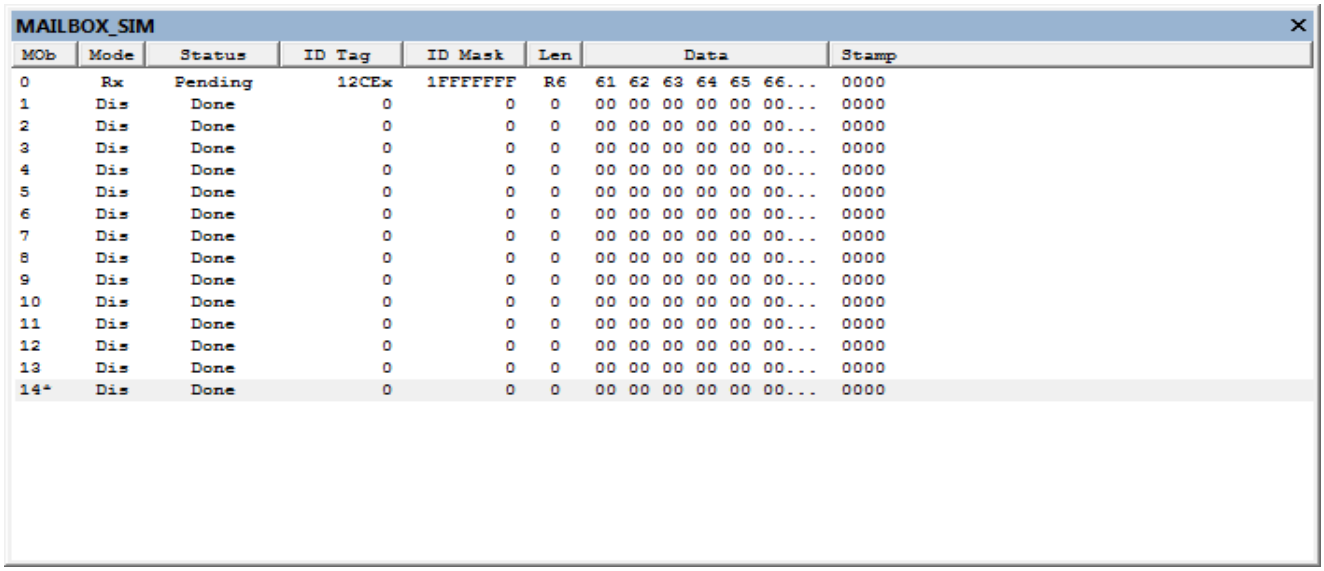


Рисунок 11.21– Моделювання функції set_MOb0_rtr_tx()

Бачимо, що програма встановила ідентифікатор 0x12CE (поле ID Tag). Слід зазначити, що ідентифікатор 0x12CE ми обираємо самостійно. Може бути вибраний будь-який інший ідентифікатор. Рекомендація користувачу змінити ідентифікатор та проаналізувати зміни в регістрі. При ініціалізації регістрів маски, яка була описана вище, прийом повідомлення віддаленого запиту був налаштований з маскою: 1FFFFFFF, що й відображено на рисунку 11.21. Дані (поле Data), які будуть передані в автоматичному режимі, саме ті, які ми вище запрограмували. Довжина поля даних – 6 байтів. Тобто, таким чином, ми підготували кадр даних для відправки на кадр віддаленого запиту.

За допомогою функцій set_MOb1_rx() та set_MOb2_rx() ми налаштовуємо два MOB: MOb1 та MOb2 на прийом.

Розглянемо функцію set_MOb1_rx(). Вона ідентична функції set_MOb2_rx(), оскільки і перша і друга встановлює відповідні MOb у режим прийому.

Перед отриманням повідомлення деякі поля MOb повинні бути ініціалізовані:

```
/*
 * Функція налаштування MOb1 на прийом.
 */
void set_mob1_rx(void)
{
```

```
    uint8_t mob = 1;
    /* Зберегти поточний MOb. Блок 4.1*/
    uint8_t save_canpage = CANPAGE;

    /* Встановити MOb1 у якості поточного. */
    CANPAGE = mob << 4;
    /* Встановити ідентифікатор та біт RTR. Блок 4.7 */
    CANIDT4 = (uint8_t) (MOB1_ID << 3);
    CANIDT3 = (uint8_t) (MOB1_ID >> 5);
    CANIDT2 = (uint8_t) (MOB1_ID >> 13);
    CANIDT1 = (uint8_t) (MOB1_ID >> 21);
    /* Встановити маску для перевірки IDE, біта RTR та ідентифікатора. Блок 4.5*/
    CANIDM4 = (uint8_t) ((MOB1_MASK << 3) | (1 << IDEMSK));
    CANIDM3 = (uint8_t) (MOB1_MASK >> 5);
    CANIDM2 = (uint8_t) (MOB1_MASK >> 13);
    CANIDM1 = (uint8_t) (MOB1_MASK >> 21);
    /* Дозволити переривання для MOb1. Блок 4.14*/
    CANIE2 |= (1 << IEMOB1);
    /* Встановити режим прийому та IDE. Блок 4.12 */
    CANCDMOB = (1 << CONMOB1) | (1 << IDE) | (MOB1_DLC << DLC0);
    /* Відновити поточний MOb. */
    CANPAGE = save_canpage;
}
```

В даному випадку встановлюємо перший MOb в якості поточного (нумерація починається з нульового MOb).

Результати виконання функцій set_MOb1_rx() та set_MOb2_rx() продемонстровані на рисунку 11.22. Коли CAN-модуль налаштований на прийом, він очікує поки в CAN-мережі з’явиться кадр даних з відповідним ідентифікатором. Функція set_MOb1_rx() налаштовує Mob1 на прийом кадру даних з ідентифікатором 0x0123. Нагадаємо що маска (поле ID Mask = 1FFFFFFF) встановлюється програмно.

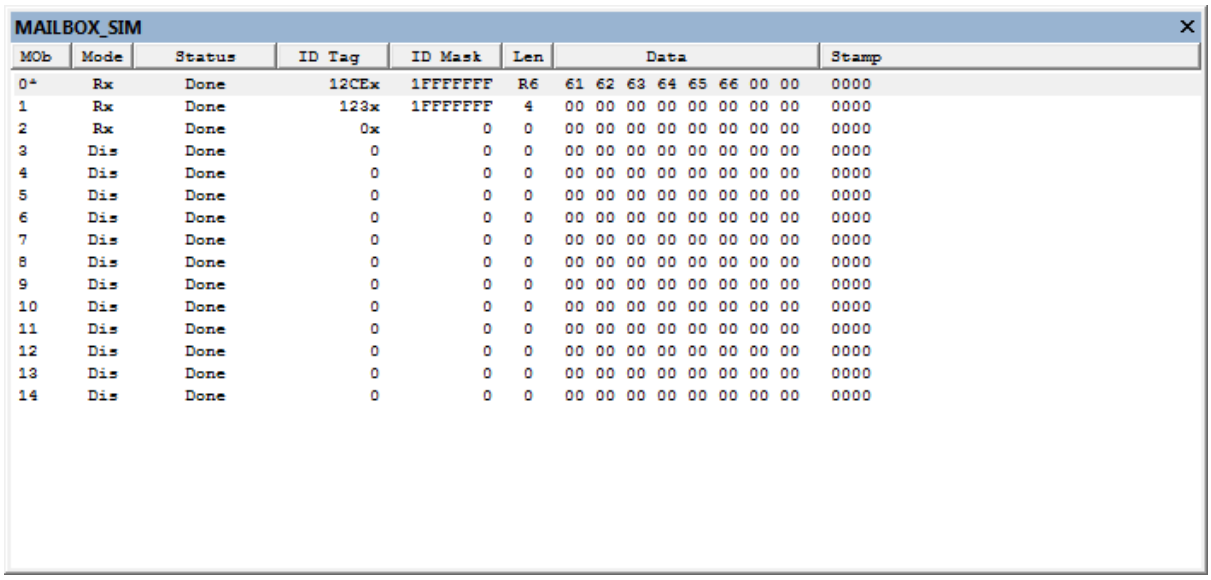


Рисунок 11.22 – Моделювання функцій set_MOb1_rx() та set_MOb2_rx()

Після того, як виконання функцій set_MOb1_rx() та set_MOb2_rx() закінчилось (рисунок 11.22), відкриємо вікно CANIN_SIM та подвійно натиснемо на повідомленні з ідентифікатором 0x0123 (рисунок 11.23). Після цього, щоб побачити прийом повідомлення потрібно натискати F10 або F11. Різниця між двома клавішами описана вище.

Потім відкриємо панель TRAFFIC_SIM, в якій з’явиться це повідомлення (рисунок 11.24). Тобто повідомлення дійсно існує в мережі.

На рисунку 11.25 показано, що Mob1, який налаштований на ідентифікатор 0x0123, після симулювання надсилання повідомлення почав приймати це повідомлення у буфер.

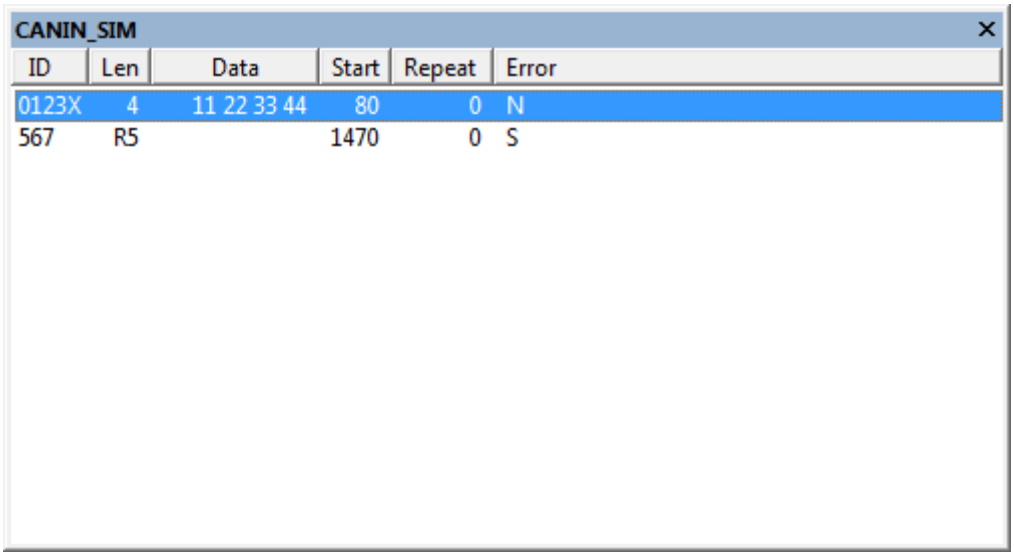


Рисунок 11.23 – Симулювання надсилання повідомлення контролеру

Після того, як повідомлення буде прийнято, вміст панелі MAILBOX_SIM зміниться, як показано на рисунку 11.26. У буфері знаходяться саме ті дані, які ми мали передати.

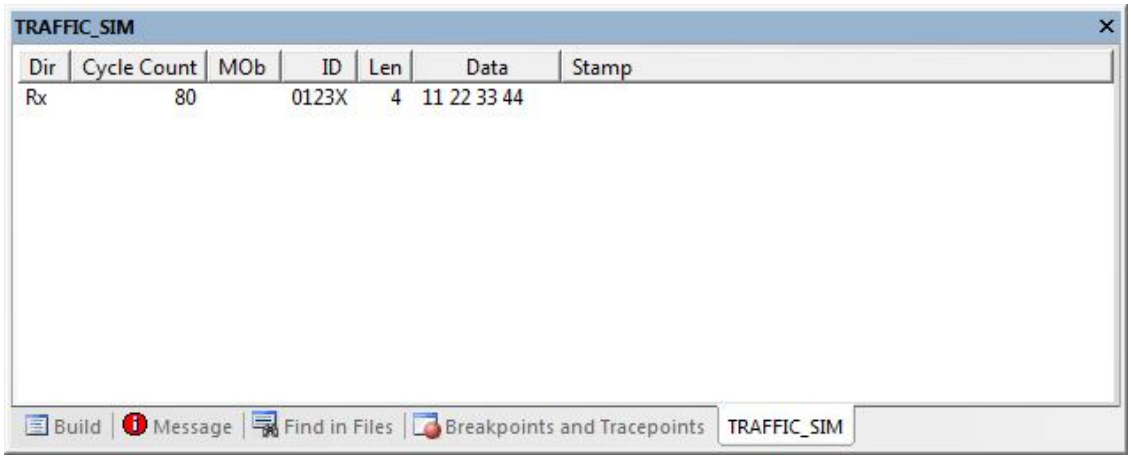


Рисунок 11.24 – Вміст панелі TRAFFIC_SIM після симулювання надсилання повідомлення

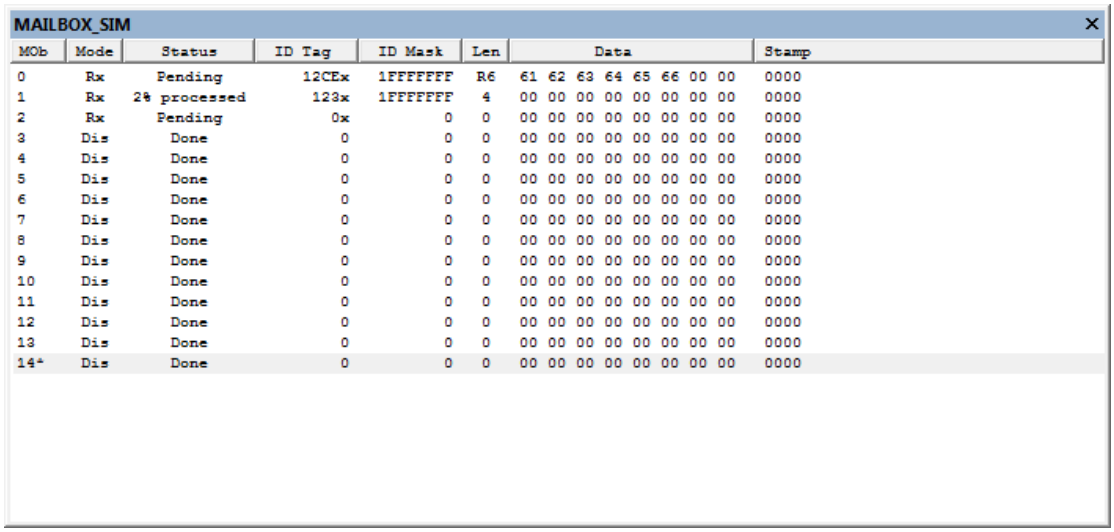


Рисунок 11.25 – Mob1 почав приймати повідомлення у буфер

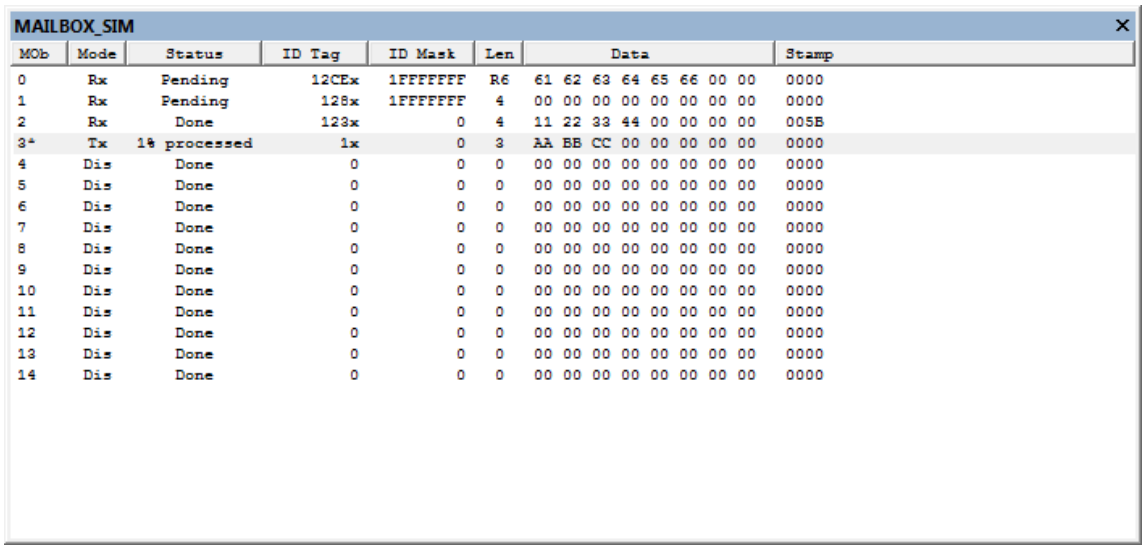


Рисунок 11.26 – Повідомлення прийнято у буфер

В даній роботі передача кадрів даних моделюється за допомогою функції set_mob3_tx.

Перед відправкою цього повідомлення відповідні поля MOb повинні бути ініціалізовані:

```
/*
 * Функція надсилання повідомлення з обраного MOb.
 *    message - вказівник на структуру повідомлення, що містить
 *              ідентифікатор, власне дані та їх довжину.
 *    send_mob - порядкомий номер MOb, що буде підготовлений для
 *              передачі.
 */
void set_mob_tx(CAN_message *message, uint8_t send_mob)
{
   uint8_t mob = send_mob;
```

```
  /* Зберегти поточний MOb. */
  uint8_t save_canpage = CANPAGE;

  /* Встановити бажаний MOb у якості поточного. */
  CANPAGE = mob << 4;
  /* Встановити ідентифікатор повідомлення. */
  CANIDT4 = (uint8_t) (message->id << 3);
  CANIDT3 = (uint8_t) (message->id >> 5);
  CANIDT2 = (uint8_t) (message->id >> 13);
  CANIDT1 = (uint8_t) (message->id >> 21);

  /*
   * Заповнити дані повідомлення.
   * Оскільки біт CANPAGE.AINC скинуто (тобто встановлено
   * автоінкремент номеру байта в буфері MOb) послідовно
   * записати message->length байт у CANMSG.
   */
  for (uint8_t i = 0; i < message->length; i++) {
    CANMSG = message->data[i];
  }

  if (send_mob < 8)
  {
    CANIE2 |= 1 << send_mob;
  }
  else
  {
    CANIE1|= 1 << send_mob;
  }

  /* Зкинути прапорці переривань. */
  CANSTMOB = 0;

  /* Встановити режим передачі та IDE. */
  CANCDMOB = (1 << CONMOB0) | (1 << IDE) | (message->length << DLC0);

  /* Відновити поточний MOb. */
  CANPAGE = save_canpage;
}
```

Функція set_mob3_tx буде надсилати повідомлення message з обраного MOb, який встановлюється параметром send_mob.

```
/* Надіслати повідомлення з MOb3. */
  set_mob_tx(&tx_message, 3);

  /* Надіслати повідомлення з MOb7. */
  set_mob_tx(&tx_message1, 7);

  /* Надіслати повідомлення з MOb4. */
  set_mob_tx(&tx_message2, 4);
```

```
/* Дочекатися завершення передачі MOb7 */
/* Щоб використовувати його далі */
uint8_t save_canpage = CANPAGE;
CANPAGE = 7 << 4;
while (!(CANSTMOB & (1 << TXOK)));
CANPAGE = save_canpage;
```

Спочатку ми налаштовуємо на передачу 3 моб та починаємо передавати повідемлення, одночасно налаштовуючи на передачу спочатку 7 потім 4 моби. Після закінчення передачі повідомлення 3 мобом починає передавати 4 моб оскільки у ньго менше порядковій номер, і тому він має більший пріоритет в порівняннні з MOb7. Передавши повідомлення переходять в дезактивований режим.

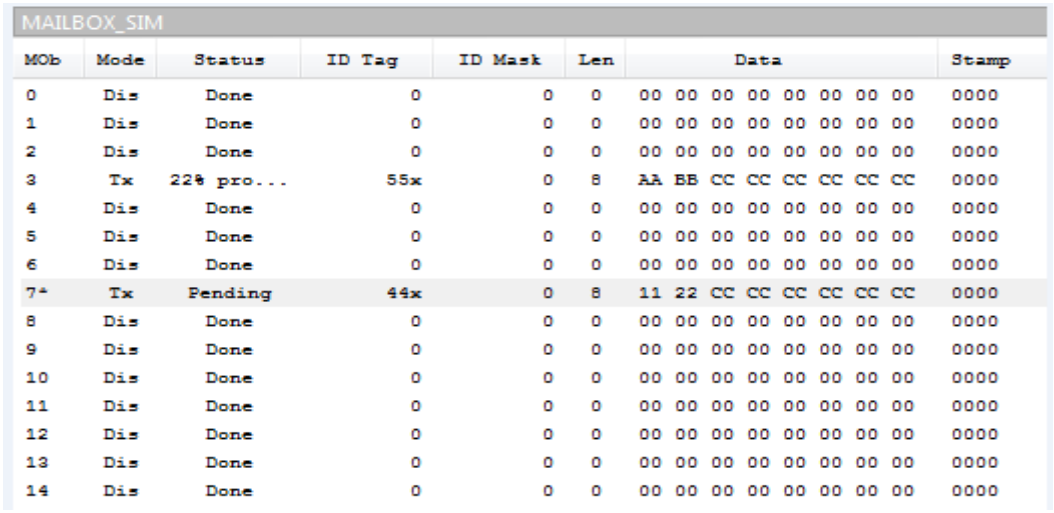
Перед тим як йти далі – дочекаємося завершення передачі MOb7.

Процес моделювання можливо спостерігати за допомогою рисунків 11.27 - 11.31. Поле Status показує який процент повідомлення надісланий.

MAILBOX_SIM

MOb	Mode	Status	ID Tag	ID Mask	Len	Data	Stamp
0	Dis	Done	0	0	0	00 00 00 00 00 00 00 00	0000
1	Dis	Done	0	0	0	00 00 00 00 00 00 00 00	0000
2	Dis	Done	0	0	0	00 00 00 00 00 00 00 00	0000
3	Tx	7% proc...	55x	0	8	AA BB CC CC CC CC CC CC	0000
4	Dis	Done	0	0	0	00 00 00 00 00 00 00 00	0000
5	Dis	Done	0	0	0	00 00 00 00 00 00 00 00	0000
6	Dis	Done	0	0	0	00 00 00 00 00 00 00 00	0000
7*	Dis	Done	0	0	0	00 00 00 00 00 00 00 00	0000
8	Dis	Done	0	0	0	00 00 00 00 00 00 00 00	0000
9	Dis	Done	0	0	0	00 00 00 00 00 00 00 00	0000
10	Dis	Done	0	0	0	00 00 00 00 00 00 00 00	0000
11	Dis	Done	0	0	0	00 00 00 00 00 00 00 00	0000
12	Dis	Done	0	0	0	00 00 00 00 00 00 00 00	0000
13	Dis	Done	0	0	0	00 00 00 00 00 00 00 00	0000
14	Dis	Done	0	0	0	00 00 00 00 00 00 00 00	0000

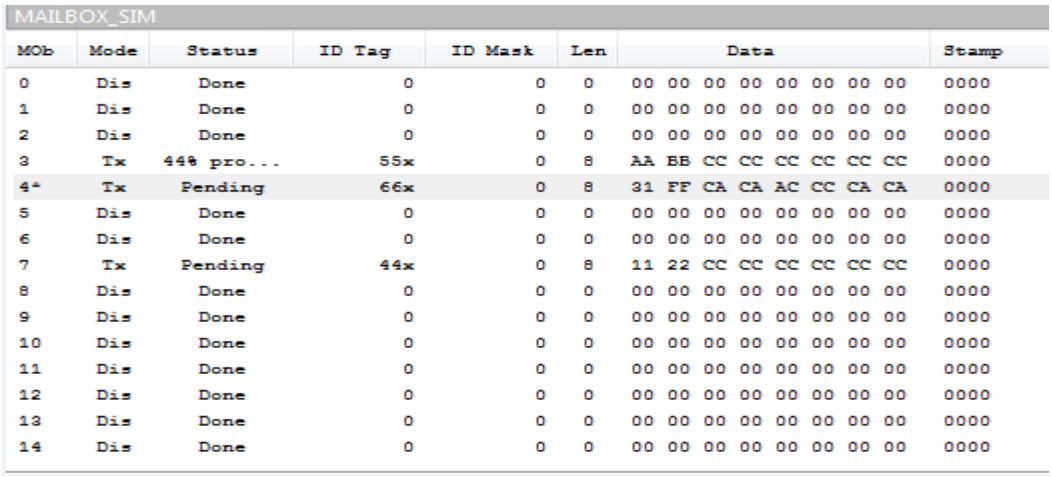
Рисунок 11.27 – Контролер передає повідомлення з ідентифікатором 55 (7% processed)



MAILBOX_SIM

MOb	Mode	Status	ID Tag	ID Mask	Len	Data	Stamp
0	Dis	Done	0	0	0	00 00 00 00 00 00 00 00	0000
1	Dis	Done	0	0	0	00 00 00 00 00 00 00 00	0000
2	Dis	Done	0	0	0	00 00 00 00 00 00 00 00	0000
3	Tx	22% pro...	55x	0	8	AA BB CC CC CC CC CC CC	0000
4	Dis	Done	0	0	0	00 00 00 00 00 00 00 00	0000
5	Dis	Done	0	0	0	00 00 00 00 00 00 00 00	0000
6	Dis	Done	0	0	0	00 00 00 00 00 00 00 00	0000
7*	Tx	Pending	44x	0	8	11 22 CC CC CC CC CC CC	0000
8	Dis	Done	0	0	0	00 00 00 00 00 00 00 00	0000
9	Dis	Done	0	0	0	00 00 00 00 00 00 00 00	0000
10	Dis	Done	0	0	0	00 00 00 00 00 00 00 00	0000
11	Dis	Done	0	0	0	00 00 00 00 00 00 00 00	0000
12	Dis	Done	0	0	0	00 00 00 00 00 00 00 00	0000
13	Dis	Done	0	0	0	00 00 00 00 00 00 00 00	0000
14	Dis	Done	0	0	0	00 00 00 00 00 00 00 00	0000

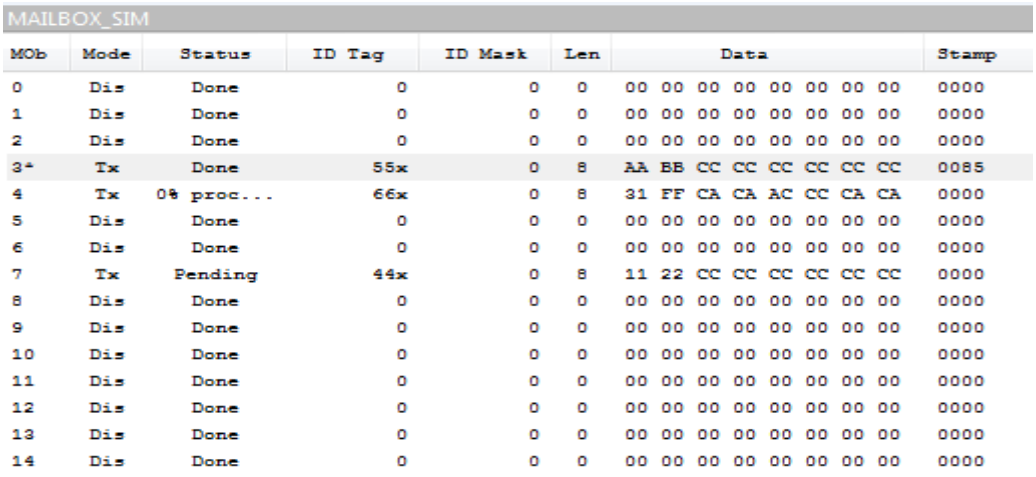
Рисунок 11.28 – Контролер передає повідомлення з ідентифікатором 55 (22% processed), та підготував MOB7 до передачі.



MAILBOX_SIM

MOb	Mode	Status	ID Tag	ID Mask	Len	Data	Stamp
0	Dis	Done	0	0	0	00 00 00 00 00 00 00 00	0000
1	Dis	Done	0	0	0	00 00 00 00 00 00 00 00	0000
2	Dis	Done	0	0	0	00 00 00 00 00 00 00 00	0000
3	Tx	44% pro...	55x	0	8	AA BB CC CC CC CC CC CC	0000
4*	Tx	Pending	66x	0	8	31 FF CA CA AC CC CA CA	0000
5	Dis	Done	0	0	0	00 00 00 00 00 00 00 00	0000
6	Dis	Done	0	0	0	00 00 00 00 00 00 00 00	0000
7	Tx	Pending	44x	0	8	11 22 CC CC CC CC CC CC	0000
8	Dis	Done	0	0	0	00 00 00 00 00 00 00 00	0000
9	Dis	Done	0	0	0	00 00 00 00 00 00 00 00	0000
10	Dis	Done	0	0	0	00 00 00 00 00 00 00 00	0000
11	Dis	Done	0	0	0	00 00 00 00 00 00 00 00	0000
12	Dis	Done	0	0	0	00 00 00 00 00 00 00 00	0000
13	Dis	Done	0	0	0	00 00 00 00 00 00 00 00	0000
14	Dis	Done	0	0	0	00 00 00 00 00 00 00 00	0000

Рисунок 11.29 – Контролер передає повідомлення з ідентифікатором 55 (44% processed), та підготував MOB7 та MOB 4 до передачі.



MAILBOX_SIM

MOb	Mode	Status	ID Tag	ID Mask	Len	Data	Stamp
0	Dis	Done	0	0	0	00 00 00 00 00 00 00 00	0000
1	Dis	Done	0	0	0	00 00 00 00 00 00 00 00	0000
2	Dis	Done	0	0	0	00 00 00 00 00 00 00 00	0000
3*	Tx	Done	55x	0	8	AA BB CC CC CC CC CC CC	0085
4	Tx	0% proc...	66x	0	8	31 FF CA CA AC CC CA CA	0000
5	Dis	Done	0	0	0	00 00 00 00 00 00 00 00	0000
6	Dis	Done	0	0	0	00 00 00 00 00 00 00 00	0000
7	Tx	Pending	44x	0	8	11 22 CC CC CC CC CC CC	0000
8	Dis	Done	0	0	0	00 00 00 00 00 00 00 00	0000
9	Dis	Done	0	0	0	00 00 00 00 00 00 00 00	0000
10	Dis	Done	0	0	0	00 00 00 00 00 00 00 00	0000
11	Dis	Done	0	0	0	00 00 00 00 00 00 00 00	0000
12	Dis	Done	0	0	0	00 00 00 00 00 00 00 00	0000
13	Dis	Done	0	0	0	00 00 00 00 00 00 00 00	0000
14	Dis	Done	0	0	0	00 00 00 00 00 00 00 00	0000

Рисунок 11.30 – Контролер передає повідомлення з ідентифікатором 66 (0% processed)

MAILBOX_SIM

MOb	Mode	Status	ID Tag	ID Mask	Len	Data	Stamp
0	Dis	Done	0	0	0	00 00 00 00 00 00 00 00	0000
1	Dis	Done	0	0	0	00 00 00 00 00 00 00 00	0000
2	Dis	Done	0	0	0	00 00 00 00 00 00 00 00	0000
3	Dis	Done	0	0	0	AA BB CC CC CC CC CC CC	0085
4*	Tx	Done	66x	0	8	31 FF CA CA AC CC CA CA	00E5
5	Dis	Done	0	0	0	00 00 00 00 00 00 00 00	0000
6	Dis	Done	0	0	0	00 00 00 00 00 00 00 00	0000
7	Tx	1% proc...	44x	0	8	11 22 CC CC CC CC CC CC	0000
8	Dis	Done	0	0	0	00 00 00 00 00 00 00 00	0000
9	Dis	Done	0	0	0	00 00 00 00 00 00 00 00	0000
10	Dis	Done	0	0	0	00 00 00 00 00 00 00 00	0000
11	Dis	Done	0	0	0	00 00 00 00 00 00 00 00	0000
12	Dis	Done	0	0	0	00 00 00 00 00 00 00 00	0000
13	Dis	Done	0	0	0	00 00 00 00 00 00 00 00	0000
14	Dis	Done	0	0	0	00 00 00 00 00 00 00 00	0000

Рисунок 11.31 – Контролер передає повідомлення з ідентифікатором 44 (1% processed). MOB3 дезактивовано.

При необхідності передачі або прийому повідомленнь з довжиною більше 8 байт використовується сегментація повідомлення, так звані мультикадри. Детально алгоритми їх прийому та передічі описано в стандарті ISO15765_2.

Моделювання режиму прийому мультикадру виконується за допомогою наступних функцій:

1) void set_frame_buffer(CAN_buffer *buf) – функція для підготовки обраних Mob'ів до прийому мультикадру;
2) void get_frame_buffer(CAN_buffer *buf) – функція зчитування прийнятого мультикадру з обраних MOb у пам'ять даних.

Мультикадр моделюється за допомогою структури даних CAN_buffer, яка включає у себе:

1) uint32_t id – ідентифікатор повідомлень, що входять до складу мультикадру;
2) uint16_t buffered_mobs – обрані для прийому мультикадру Mob'и, що задаються встановленням відповідного біту у цьому полі (слід пам'ятати, що встановлення 16-го біту викликає помилку виконання програми, адже кількість доступних Mob'ів рівна 15)
3) uint8_t data[8*15] – дані, прийняті за допомогою мультикадру.

Так як CAN-модуль мікроконтролерів AT90CAN* підтримує прийом лише одного мультикадру, у наведеній в Додатку К програмі запропоновано для моделювання прийому мультикадру оперувати із глобальною змінною CAN_Buffer buffer. Підготовка до прийому мультикадру наведено у фрагменті програми нижче:

```
        /*
 * Підготовка буфера для прийому мультикадру.
 * Зчитування прийнятого мультикадру відбувається в обробнику
 * переривань від CAN-модуля.
 */
/* Обрання Mob'ів для прийому мультикадра */
buffer.buffered_mobs = (uint16_t) (1 << 6) | (1 << 7)| (1 << 8); //Блок 2
/* Встановлення ідентифікатору повідомлень мультикадра*/
buffer.id = MULTIFRAME_ID;
/* Ініціалізація Mob'ів для прийому мультикадра */
set_frame_buffer(&buffer);

/* Дозволити переривання. */
sei();
```

Обробка прийняття мультикадру здійснюється у функції обробки переривання SIGNAL(SIG_CAN_INTERRUPT1) наступним чином:

```
/*
 * Якщо мультикадр був прийнятий повністю,
 * встановлюється прапорець BXOKBXOK 4 186
 */
if (CANGIT & (1 << BXOK)) {
    get_frame_buffer(&buffer);

    /*
     * Скидаємо прапорець BXOK записуючи у нього "1".
     */
    CANGIT |= 1 << BXOK;
    return;
}
```

Необхідно зазначити, що перевірку прапорця BXOK необхідно виконувати до перевірки регістру CANHPMOB, так як при прийнятті мультикадру значення CANHPMOB буде рівним 0xF0 як і у випадку, коли нічого прийнято не було. Також важливим зауваженням є те, що для підготовлених до прийому мультикадру MOb необхідно обов’язково заборонити переривання у регістрах CANIE1-2, щоб не викликати непотрібних переривань від встановлення прапорця RXOK після прийому повідомлення у кожному з MOb (див. функцію void set_frame_buffer(CAN_buffer *buf) у Додатку К).

Середовище програмування AVRStudio не дозволяє автоматично змоделювати переривання після прийому мультикадру, тому це необхідно виконати вручну. Список необхідний дій:

1) запустити на покрокове виконання програму наведену у Додатку К;
2) покроковим виконанням або за допомогою встановлення точки призупину виконання перейти у нескінченний цикл функції int main (void) (у нескінченному циклі підготовка до прийому мультикадру вже виконана);
3) виконати передачу обох повідомлень із ідентифікатором 0x111 за допомогою панелі CANIN_SIM (на прийом повідомлень із зазначеним ідентифікатором налаштовані Mob'и у буфері мультикадру) і пересвідчитись у успішному прийомі за допомогою панелі MAILBOX_SIM;
4) викликати переривання встановленням прапорця BXOK у регістрі CANGIT у панелі I/O View і виконанням однієї програмної інструкції, щоб було встановлено прапорець CANIT у цьому ж регістрі (рисунок 11.32);

Рисунок 11.32 – Викликання переривання встановленням прапорця BXOK у панелі панелі I/O View

5) покроково виконати процедуру обробки переривання, щоб переконатися в успішній обробці мультикадру.

Примітка: запис даних мультикадру у пам'ять даних можна спостерігати у панелі Memory під час покорокового виконання фрагменту коду на рисунку нижче:

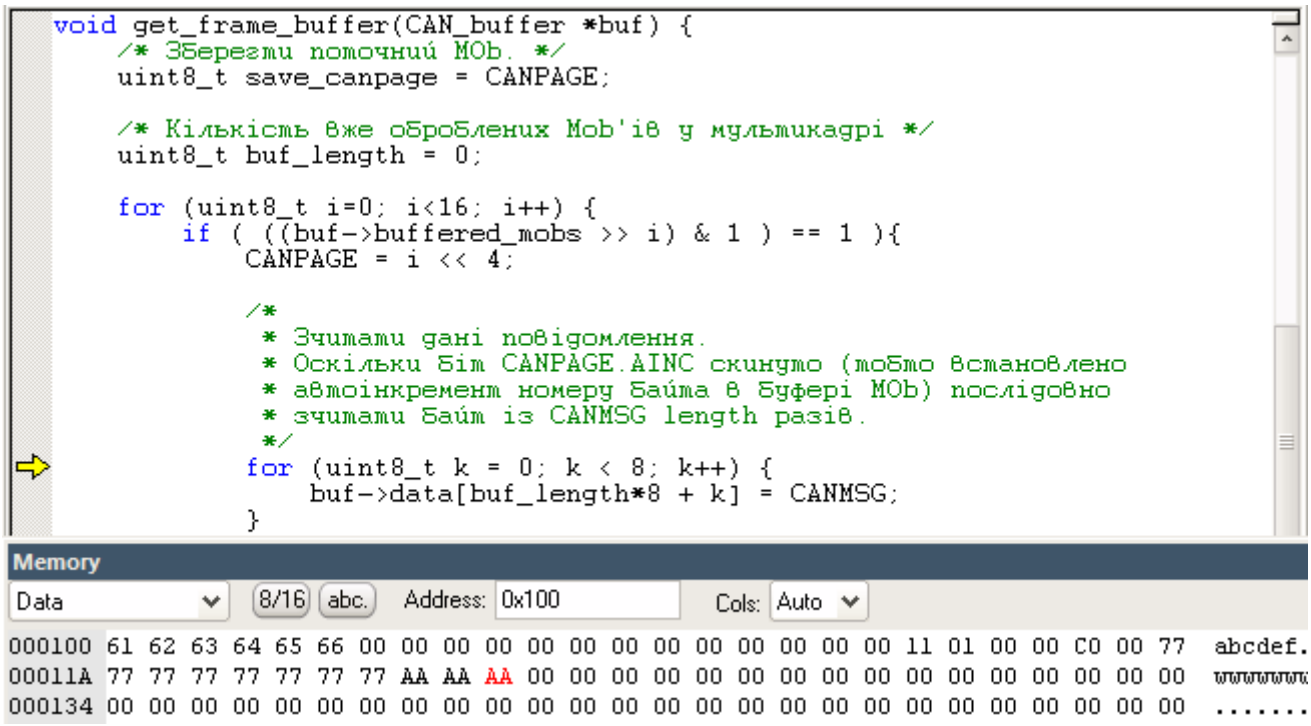


Рисунок 11.33 – Запис даних прийнятого мультикадру у пам’ять даних мікроконтролера із підсвіткою записаного байту у панелі Memory

Підготовка та передача мультикадру моделюється за допомогою функції send_multiframe().

```
/*
 * Функція передачі мультикадру. Три вільних MOb налаштовуються на передачу
 * та заповнюються масивом даних multiframe_data.
 */
void send_multiframe()
{
    uint8_t mobs_to_send = MULTIFRAME_LEN;

    /* Зберегти поточний MOb. */
    uint8_t save_canpage = CANPAGE;

    /* Останній MOb, що буде відправлено у складі мультикадру. */
    uint8_t last_mob = 0;

    /*
     * Перевірка у циклі стану всіх MOb'ів та підготовка до передачі
     * повідомлення перших трьох вільних.
     */
    for(uint8_t scan_mob = 0; scan_mob < 15; scan_mob++)
    {
        /* MOb, що перевіряється */
        CANPAGE = scan_mob << 4;

        /* Перевірка значень бітів CONMOB0-1, 00 - вільний*/
        uint8_t mob_state = CANCDMOB & ((1 << CONMOB0) | (1 << CONMOB1));
        if (mob_state == 0)
        {
            /* Підготовка повідомлення до передачі. */
            CAN_message tx_message;
            tx_message.length = 8;
            tx_message.id = MULTIFRAME_ID;
            /* Заповнення повідомлення даними з масиву multiframe_data */
            for (uint8_t i = 0; i < 8; i++)
            {
                uint8_t byte_to_send = (MULTIFRAME_LEN - mobs_to_send)*8 + i;
                tx_message.data[i] = multiframe_data[byte_to_send];
            }

            /* Налаштування MOb на передачу */
            set_mob_tx(&tx_message, scan_mob);

            last_mob = scan_mob;
            mobs_to_send -= 1;
        }

        /* Завершити цикл, якщо необхідну кількість MOb'ів підготовлено */
        if (mobs_to_send == 0)
        {
            break;
        }
    }

    CANPAGE = last_mob << 4;

    /* Дочекатися закінчення надсилання повідомлення. */
    while (!(CANSTMOB & (1 << TXOK)))
        ;

    /* Відновити поточний MOb. */
    CANPAGE = save_canpage;
```

```
}
```

Функція send_multiframe() налаштовує три вільних MOb на передачу мультикадру. Джерелом даних є буфер multiframe_data. Тобто дані з буферу сегментуються і поміщуються у відповідні структури для подальшої передачі.

Процес моделювання передачі мультикадру можливо спостерігати за допомогою рисунків 11.34 - 11.35.

MAILBOX_SIM

MOb	Mode	Status	ID Tag	ID Mask	Len	Data	Stamp
0	Dis	Done	0	0	0	00 00 00 00 00 00 00 00	0000
1	Dis	Done	0	0	0	00 00 00 00 00 00 00 00	0000
2	Dis	Done	0	0	0	00 00 00 00 00 00 00 00	0000
3	Dis	Done	0	0	0	AA BB CC CC CC CC CC CC	0085
4	Dis	Done	0	0	0	31 FF CA CA AC CC CA CA	00E5
5	Dis	Done	0	0	0	00 00 00 00 00 00 00 00	0000
6	Dis	Done	0	0	0	00 00 00 00 00 00 00 00	0000
7	Dis	Done	0	0	0	11 22 CC CC CC CC CC CC	0145
8	Dis	Done	0	0	0	00 00 00 00 00 00 00 00	0000
9	Dis	Done	0	0	0	00 00 00 00 00 00 00 00	0000
10	Tx	55% pro...	22Bx	0	8	10 0E 21 22 23 24 25 26	0000
11	Tx	Pending	22Bx	0	8	21 27 28 29 30 31 32 33	0000
12*	Tx	Pending	22Bx	0	8	31 34 FF FF FF FF FF FF	0000
13	Dis	Done	0	0	0	00 00 00 00 00 00 00 00	0000
14	Dis	Done	0	0	0	00 00 00 00 00 00 00 00	0000

Рисунок 11.34 – Контролер передає мультикадр з ідентифікатором 22B (55% MOB10 processed).

TRAFFIC_SIM

Dir	Cycle Count	MOb	ID	Len	Data	Stamp
Tx	3754	7	44X	8	11 22 CC C...	0x145
Tx	6754	10	22BX	8	10 0E 21 22 ...	0x2bc
Tx	7523	11	22BX	8	21 27 28 29 ...	0x31c
Tx	8292	12	22BX	8	31 34 FF FF ...	0x37a

Рисунок 11.35 - Передача мультикадру від CAN – контролера

СПИСОК ЛІТЕРАТУРИ

1 Опис протоколу CAN 2.0A – http://www.gaw.ru/html.cgi/txt/interface/can/can20a.htm

2 Евстифеев А.В. Микроконтроллеры AVR семейств Tiny и Mega фирмы ATMEL – М.: Додэка, 2005.

3 Евстифеев А.В. Микроконтроллеры AVR семейства – М.: Додэка-XXI, 2007.

4 Опис протоколу CAN - http://www.itt-ltd.com/reference/ref_can.html

5 Корольов Н.В., Корольов Д.Н. AVR-микроконтроллеры второго поколения: новые аппаратные возможности // Компоненты и Технологии. 2003.

6 CAN-словарь CAN-dictionaryv2_ru PDF-формат – http://www.can-cia.org

7 Схеми та пояснення роботи CAN контролерів та трансиверів - http://atmel.com

8 Опис протоколів CAN-мереж - http://www.chipinfo.ru

9 Інформація про развиток контролерів AVR – http://www.atmel.ru/Articles/Atmel30.html

10 Кривченко И.В., Ламберт Е.В. AVR-микроконтроллеры: семь ярких лет становления. Что дальше? Часть 1. Текущее положение AVR на мировом рынке микроконтроллеров и планы развития семейства в 2004 году // Компоненты и Технологии. 2004.

11 Конрад Этшебергер Controller Area Network. Basics, Protocols, Chips and Application // IXXAT Press, 2001.

12 Microchip MCP2551. High-Speed CAN Transceiver, 2003 Microchip Technology Inc.

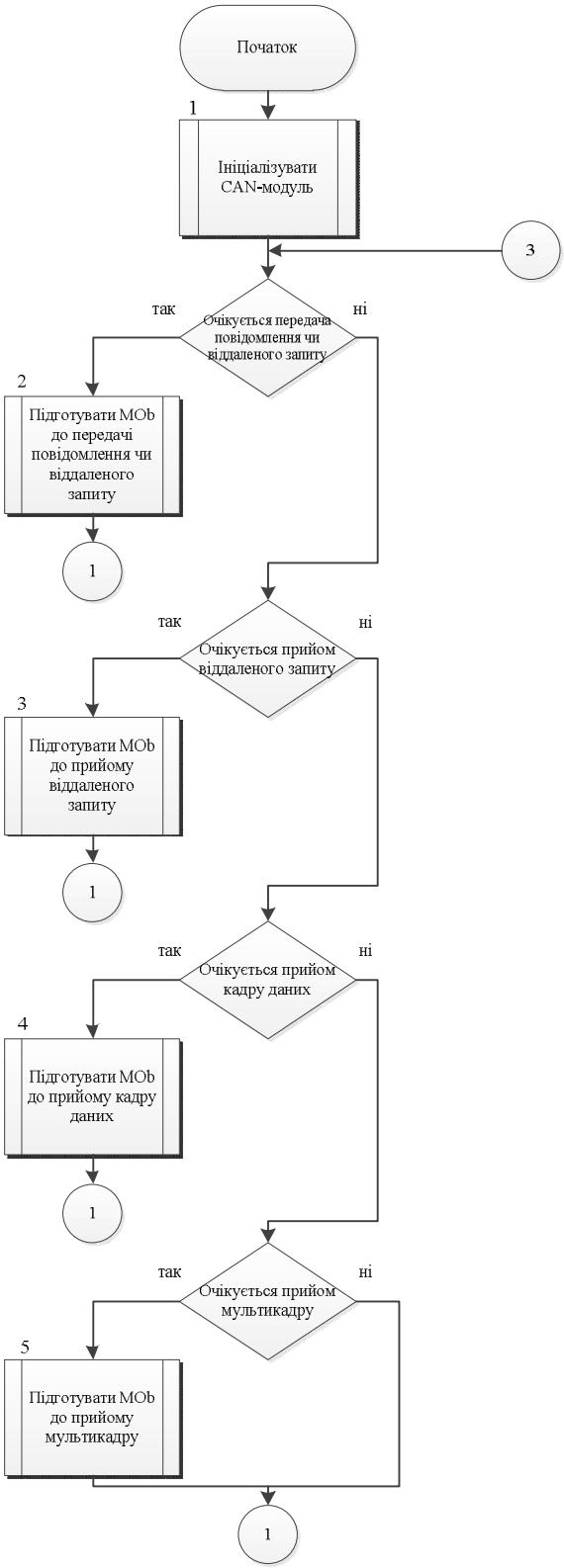
13 CAN on the AVR. Draft 1.2 prerelease – Mike Blenderman, англ. – http://www.avrfreaks.net/

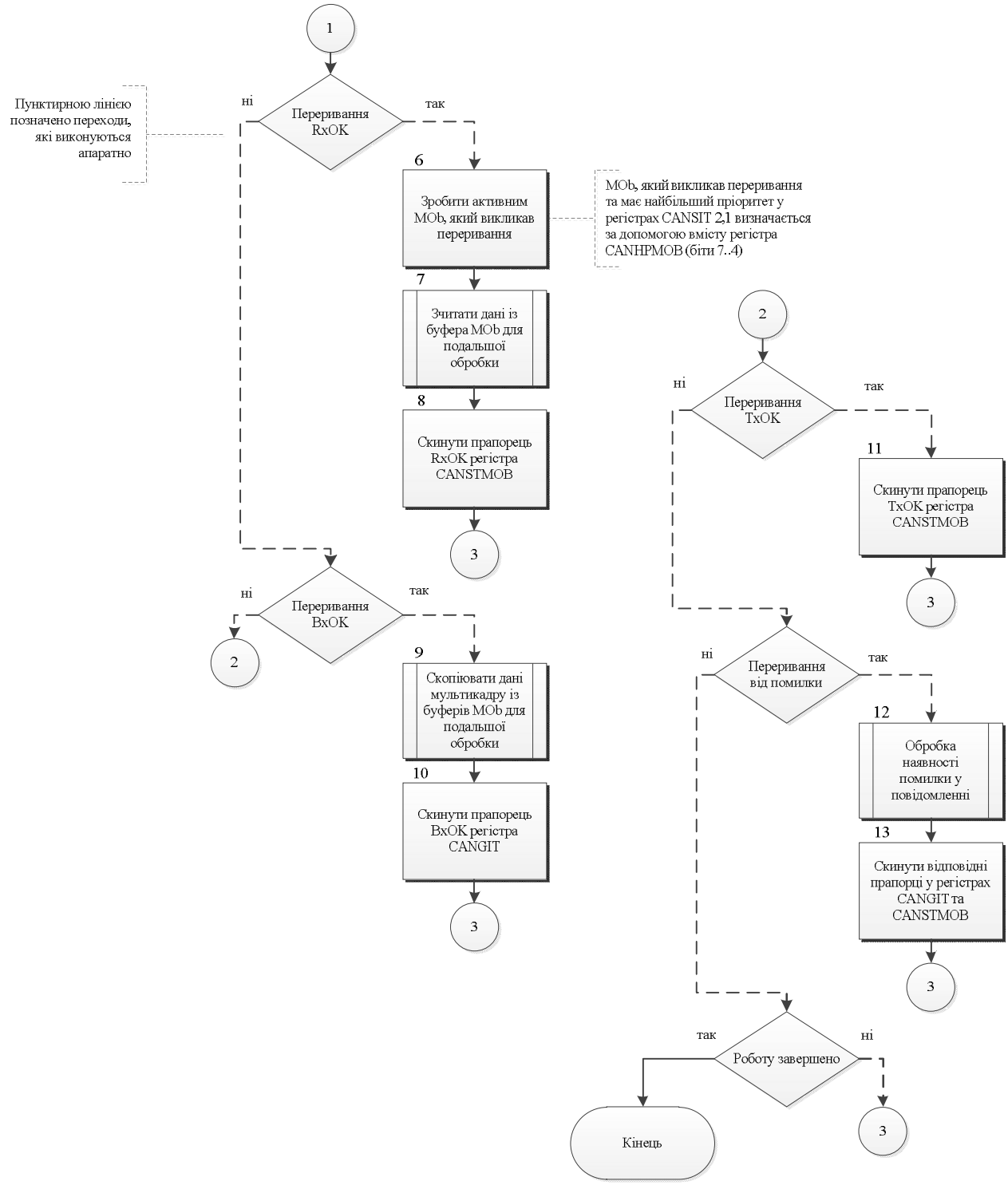
14 AVR-CAN – макетная плата – http://www.terraelectronica.ru/catalog_info.php

15 Микроконтроллер AT90CAN128 PDF-формат.

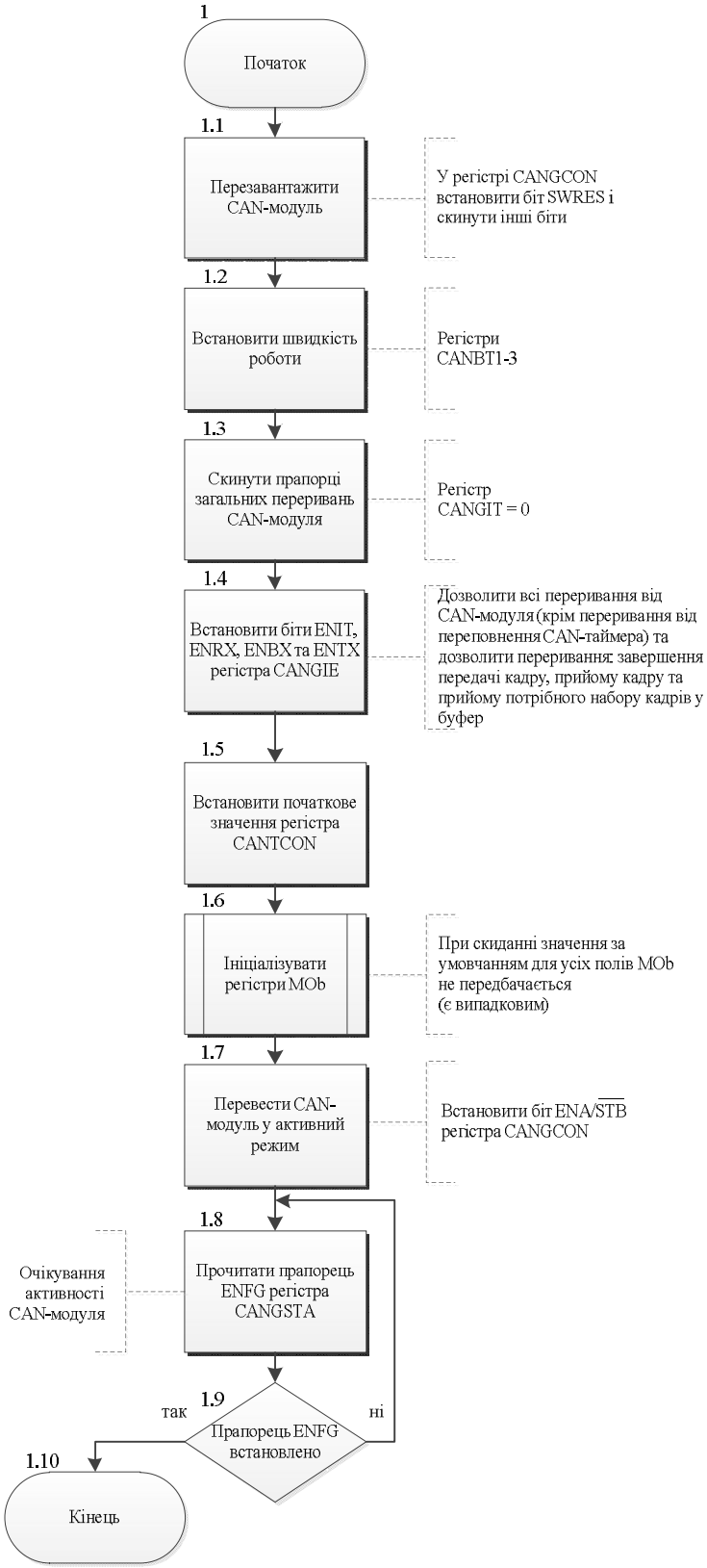
Додаток А

Схема алгоритму роботи CAN-модуля



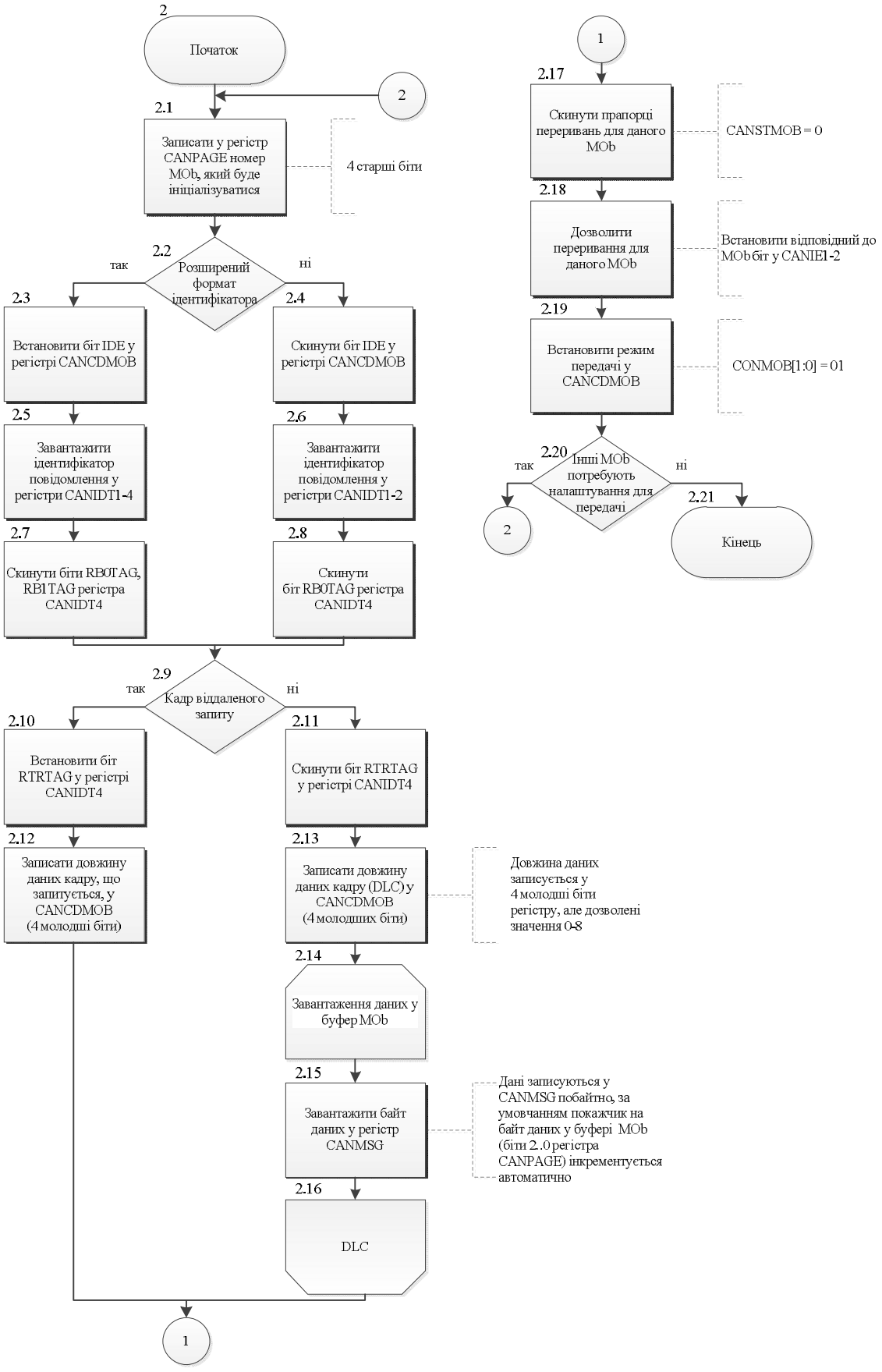

1
Пунктирною лінією позначено переходи, які виконуються апаратно
ні
Переривання RxOK
так
6
Зробити активним MOb, який викликав переривання
MOb, який викликав переривання та має найбільший пріоритет у регістрах CANSIT 2,1 визначається за допомогою вмісту регістра CANHPMOB (біти 7..4)
7
Зчитати дані із буфера MOb для подальшої обробки
8
Скинути прапорець RxOK регістра CANSTMOB
3
ні
Переривання BxOK
так
2
9
Скопіювати дані мультикадру із буферів MOb для подальшої обробки
10
Скинути прапорець BxOK регістра CANGIT
3
2
ні
Переривання TxOK
так
11
Скинути прапорець TxOK регістра CANSTMOB
3
ні
Переривання від помилки
так
12
Обробка наявності помилки у повідомленні
13
Скинути відповідні прапорці у регістрах CANGIT та CANSTMOB
3
так
Роботу завершено
ні
Кінець
3

Додаток Б
Схема алгоритму ініціалізації CAN-модуля



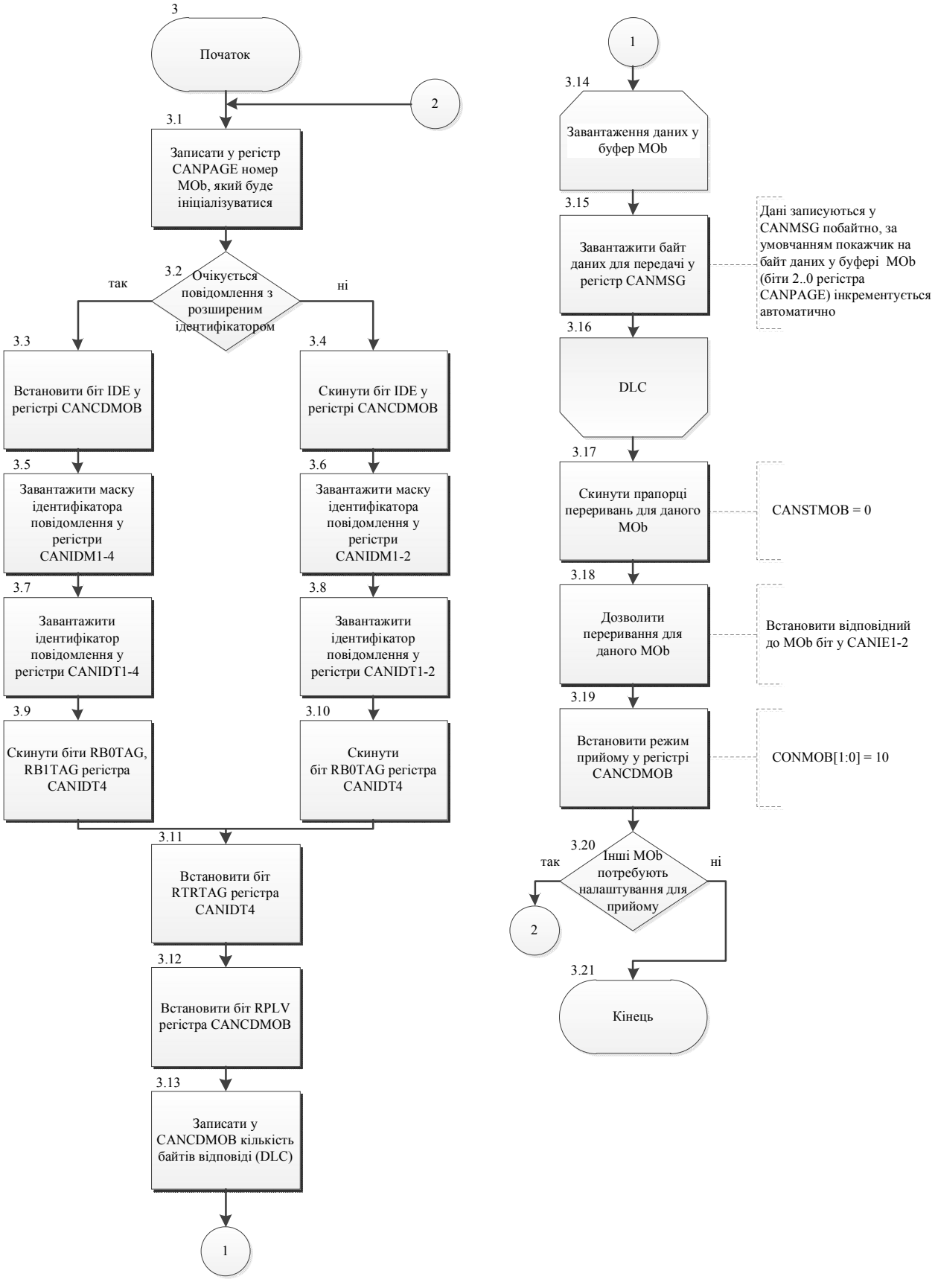
Додаток В

Схема алгоритму підготовки MOb до передачі повідомлення чи віддаленого запиту



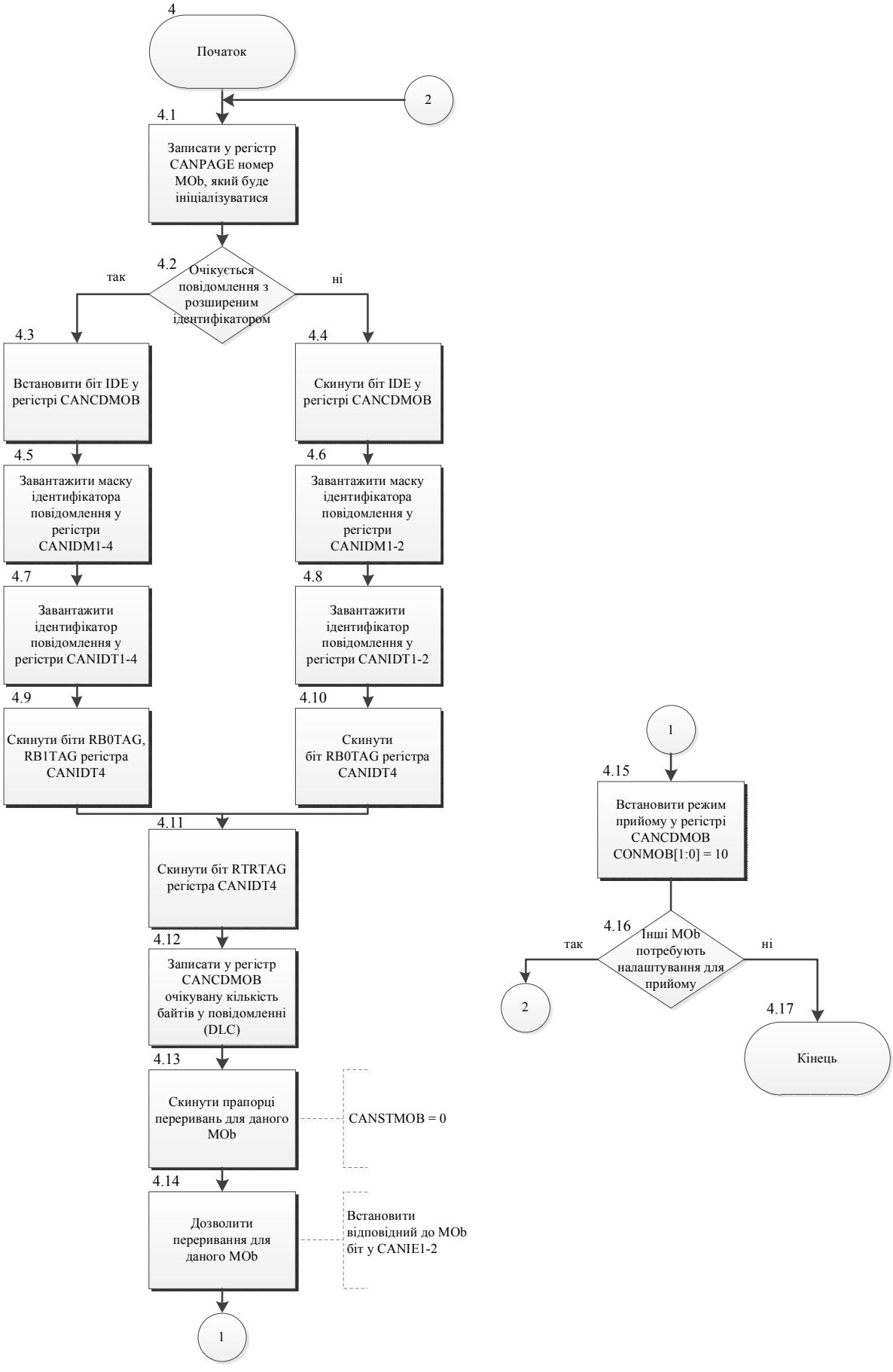
Додаток Г

Схема алгоритму підготовки MOb до прийому віддаленого запиту та автоматичної відповіді



Додаток Д

Схема алгоритму підготовки MOb до прийому кадру даних

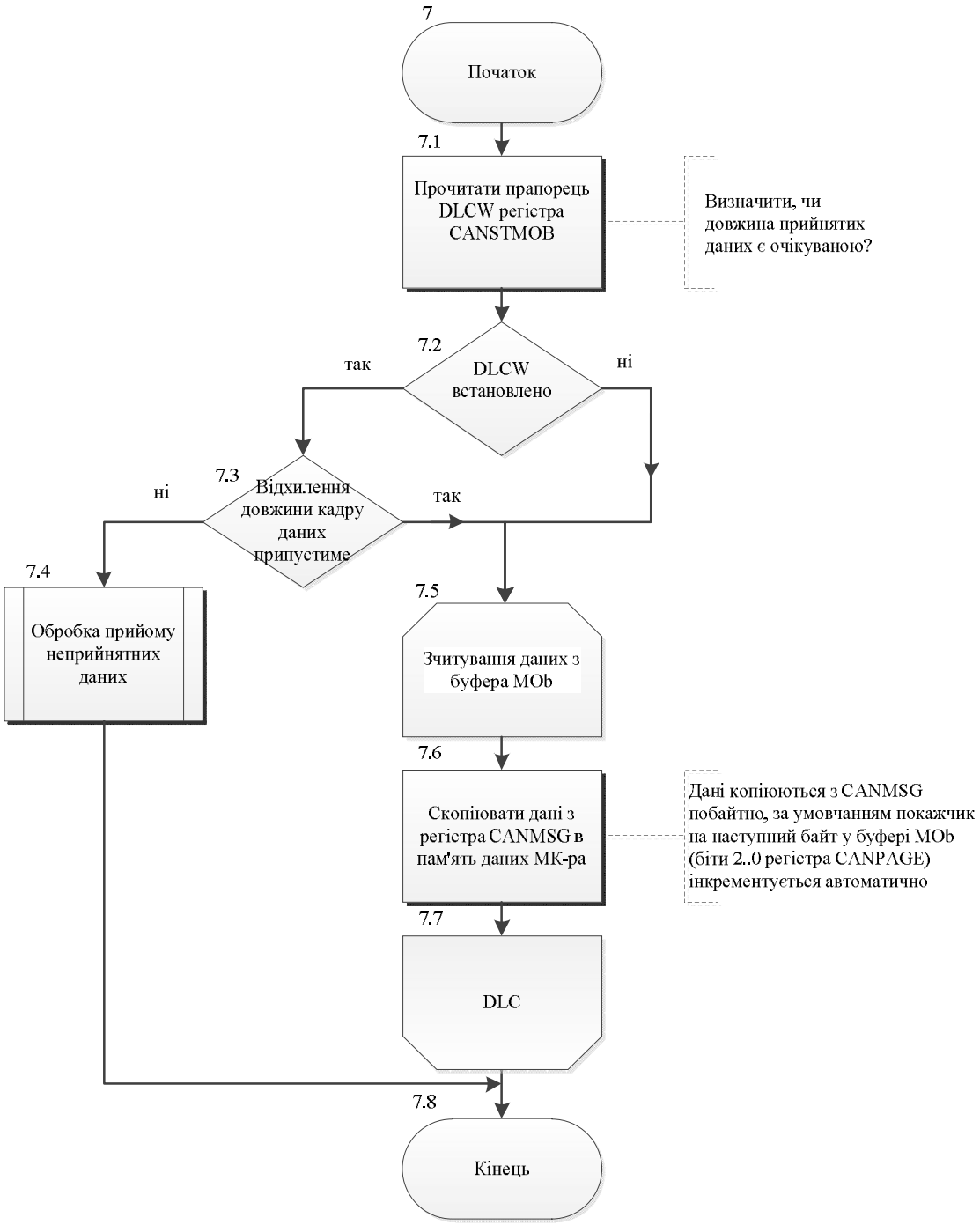


Додаток Е

Схема алгоритму підготовки MOb до прийому мультикадру (відповідного набору кадрів)

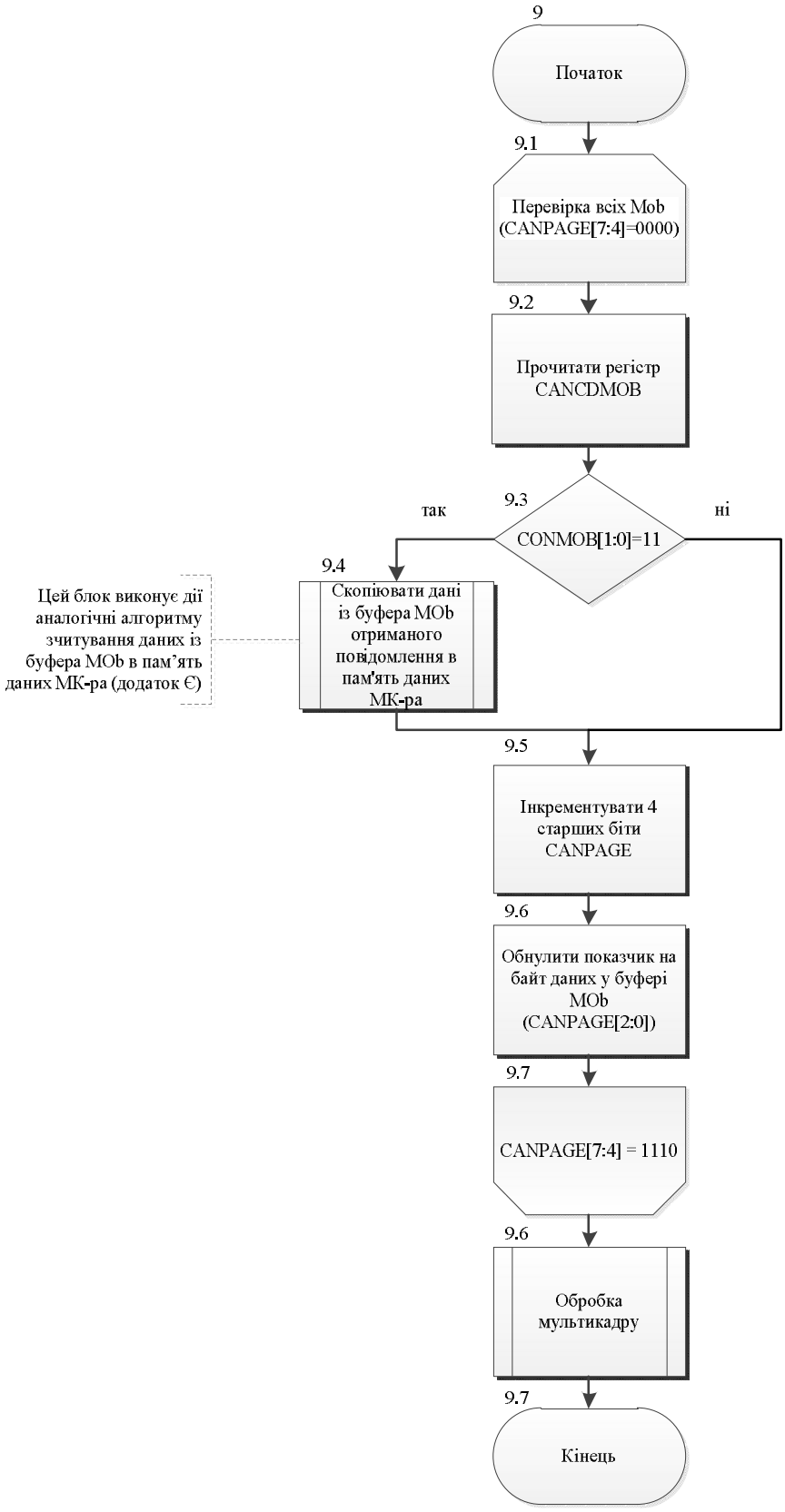
Додаток Є

Схема алгоритму зчитування даних із буфера MOb в пам'ять даних МК-ра



Додаток Ж

Схема алгоритму зчитування даних мультикадру із буферів MOb в пам'ять даних МК-ра



Додаток З

Тлумачний словник основних загальноприйнятих термінів CAN-технологій.

A

acceptance filter ***приймальний фільтр***	Приймальний фільтр CAN-контролера служить для селекції вхідних повідомлень в залежності від їх ідентифікаторів. Більшість мікросхем CAN-контролерів підтримують апаратний приймальний фільтр, який проводить відбір CAN-повідомлень з певним ідентифікатором або групою ідентифікаторів. Фільтрація, що задається користувачем, розвантажує мікроконтроллер від додаткового виконання функцій приймального фільтра.
acknowledge (ACK) delimiter ***роздільник підтвердження***	Другий біт поля підтвердження CAN-кадру. Є рецесивним. Домінантне значення роздільника вважається порушенням формату і викликає передачу кадру помилки.
acknowledge error ***помилка підтвердження***	Виявлення передавальним вузлом рецесивного стану в слоті підтвердження інтерпретується як помилка підтвердження. Помилки підтвердження не викликають стану відключення від шини. Зазвичай, такі помилки трапляються, якщо в мережі є лише один вузол і він починає передачу CAN-повідомлень.
acknowledge (ACK) field ***поле підтвердження***	Поле підтвердження складається з двох біт: слота підтвердження та роздільника підтвердження.
acknowledge (ACK) slot ***слот підтвердження***	Перший біт поля підтвердження CAN-кадру. Встановлюється рецесивним з боку передавального вузла і домінантним з боку всіх одержувачів, які здійснили успішну перевірку CRC (циклічного надлишкового коду). Якщо передавальний вузол виявляє домінантний стан цього біта, він може бути впевнений, що хоча б один вузол прийняв повідомлення без помилок.
active error flag ***активний прапорець помилки***	Активний прапорець помилки є початковою частиною активного кадру помилки і складається з шести послідовних домінантних біт.
application layer ***прикладний рівень***	Прикладний рівень є складовою частиною семирівневої еталонної моделі OSI (інтерфейс відкритих систем). Він забезпечує комунікаційну підтримку прикладних програм.

application objects ***прикладні об'єкти***	Прикладними об'єктами є функції та параметри прикладної програми, які видимі для інтерфейсу програмування (API) прикладного рівня.
application profile ***прикладний профіль***	Прикладний профіль визначає всі комунікаційні та прикладні об'єкти для кожного пристрою CAN-мережі.
arbitration field ***полі арбітражу***	Поле арбітражу включає в себе 11 або 29 бітовий ідентифікатор, біт RTR (віддалений запит), а, в разі розширеного формату кадру, також біти SRR (заміна віддаленого запиту) і IDE (прапорець розширеного формату).
assembly object ***складальний об'єкт***	Об'єкт протоколу DeviceNet, що визначає вміст повідомлення введення-виведення.
asynchronous PDO ***асинхронний PDO [Педео]***	Передача асинхронного PDO проводиться при виникненні певної внутрішньої події. Таким може бути, зокрема, закінчення таймера PDO-події. При отриманні асинхронного PDO прикладна програма повинна негайно оновити в об'єктному словнику відображені цим PDO-об'єкти.
automatic retransmission ***автоматична повторна передача***	Повторна передача спотворених повідомлень (кадрів даних і віддаленого запиту) проводиться автоматично після успішної передачі кадрів помилок.
auto bit rate detection ***авто-детектування швидкості***	У режимі автодетектуваня швидкості кожен CAN-вузол прослуховує мережевий трафік і при виявленні не спотвореного повідомлення підтверджує прийом кадру. Якщо такого не виявлено, CAN-вузол автоматично перемикається на наступну зумовлену швидкість. При виконанні цієї процедури в мережі повинен бути тільки один передавальний вузол. Автодетектування швидкості підтримується рядом CAN-контролерів. Цього можна досягти і за допомогою додаткової зовнішньої схемотехніки.

B

bandwidth ***Смуга пропускання***	Смуга пропускання визначає величину, яка задає обсяг інформації, переданої в одиницю часу.
BasicCAN ***BasicCAN-[Бейсіккан]***	Термін, що застосовувався на початку становлення CAN-технологій. Описує реалізацію, яка використовує тільки два приймальні буфера повідомлень, що заповнюються і зчитуються почергово.

Продовження додатку З

basic frame format ***основний формат кадру***	У кадрах даних і кадрах віддаленого запиту основного формату використовуються 11 бітові ідентифікатори.
basic cycle ***основний цикл***	У системі TTCAN основний цикл завжди починається з опорного повідомлення, за яким слідують декілька вікон: явних, арбітражу або порожніх. Один або кілька основних циклів формують матричний цикл TTCAN.
bit encoding ***кодування біт***	У CAN-технологіях біти кодуються без повернення до нуля (NRZ-код).
bit error ***помилка біта***	Випадок, коли біт передається домінантним рівнем, а приймається рецесивним або навпаки, розглядається як помилка біта і викликає передачу кадру помилки у черговому бітовому інтервалі. Така ситуація, однак, не є помилковою під час передачі поля арбітражу або слота підтвердження.
bit monitoring ***моніторинг біт***	Кожен передавальний CAN-контролер прослуховує мережу і при цьому здійснює моніторинг стану біт.
bit rate ***бітова швидкість***	Визначає число переданих біт за одиницю часу, незалежно від їх подання. Бітова швидкість в CAN-мережах обмежена значенням 1 мегабіт в секунду.
bit resynchronization ***бітова ресинхронізація***	Обмежена точність опорних генераторів може призвести до виходу з синхронізації будь-якого вузла. CAN-контролер здійснює ресинхронізації за кожним переходом сигналу з рецесивного на домінантний рівень.
bit stuffing ***біт-стафінг***	Вставка додаткових біт в потік даних для забезпечення зміни рівня сигналу на шині і можливості періодичної ресинхронізації.
bit time ***бітовий час***	Тривалість одного біта.
bit timing ***бітове хронування***	Бітове хронування CAN-контролера визначається квантом часу, який залежить від частоти опорного генератора і параметрів дільника (pre-scaler) бітової швидкості вузла.
bridge ***міст***	Пристрій, що забезпечує зв'язок двох мереж на канальному рівні.
Broadcast transmission ***широкомовна передача***	Комунікаційний протокол, який виконує одночасну передачу даних від одного вузла до всіх інших.
boot-up message ***повідомлення завантаження***	Комунікаційне повідомлення CANopen, яке передається при переході вузла мережі в передопераційний стан після ініціалізації.

Продовження додатку З

bus ***шина***	Топологія комунікаційної мережі, при якій всі вузли об'єднуються за допомогою пасивних каналів зв'язку, що дозволяють вести передачу в обох напрямках.
bus access ***доступ до шини***	Якщо шина вільна, будь-який вузол мережі може починати передачу кадру. У мережі CAN доступ до шини ініціюється передачею домінантного SOF-біта (біт початку кадру).
bus analyzer ***аналізатор шини***	Інструментарій, що дозволяє відстежувати і відображати передані по мережі дані. Аналізатори шини можуть працювати на фізичному, канальному або різних прикладних рівнях, наприклад, CANopen або DeviceNet.
bus arbitration ***арбітраж шини***	Процедура арбітражу необхідна для вирішення ситуацій, коли кілька вузлів намагаються отримати одночасний доступ до шини. Після її виконання доступ до шини має лише один вузол. У CAN-протоколі використовується алгоритм арбітражу CMSA / CD (Множинний доступ з виявленням несучої / детектування колізій) спільно з AMP (арбітраж за пріоритетом повідомлення). Цей протокол дозволяє проводити арбітраж без руйнування повідомлень.
bus comparator ***компаратор шини***	Елемент, що здійснює перетворення фізичних сигналів, які розповсюджуються у комунікаційному середовищі, в логічну інформацію або дані.
bus driver ***драйвер шини***	Елемент, який здійснює перетворення логічної інформації чи даних в фізичні сигнали, які можуть передаватися у комунікаційному середовищі.
bus idle ***шина вільна***	При вільному стані шини передача CAN-кадру не виконується, а всі вузли мережі виставляють рецесивні біти.
bus latency ***латентність шини***	Являє собою час між виникненням запиту на передачу і передачею SOF-біта (біт початку кадру). У CAN-мережах максимальна латентність становить: час одного повідомлення мінус тривалість одного біта.
bus length ***довжина шини***	Являє собою довжину мережного кабелю між двома термінаторами. Довжина шини для CAN-мереж обмежена використовуваною швидкістю передачі. При швидкості 1 Мбіт / с максимальна довжина шини теоретично складає 40 метрів. При менших швидкостях передачі можлива довжина шини зростає: при 50 Кбіт / с вона може досягати 1 кілометра.

busload ***завантаження шини***	Завантаження шини визначається відношенням числа переданих біт до числа біт, коли шина вільна, за певний проміжок часу.
bus monitoring mode ***режим моніторингу шини***	У режимі моніторингу CAN-контролер відключає передачу повідомлення. Це означає, що прапорець помилки і слот підтвердження (ACK) не можуть передаватися.
bus-off state ***стан bus-off*** ***[Басоф]***	Переключення CAN-контролерів у стан bus-off (відключення від шини) проводиться, коли лічильник помилок передачі (TEC) досягає значення 255. При знаходженні в стані bus-off, CAN-контролери передають лише рецесивні біти.
bus state ***стан шини***	Один з двох станів шини: домінантний або рецесивний.

С

CAN ***CAN*** ***[Кан]***	Мережа контролерів CAN є системою на основі послідовної шини. Спочатку була розроблена фірмою Robert Bosch GmbH (произносится Роберт Бош ГмбХ). Мережа CAN затверджена міжнародним стандартом ISO 11898-1. CAN реалізована багатьма виробниками мікроелектроніки.
CANaerospace ***CANaerospace*** ***[Кан-аеро-Спейс]***	Протокол високого рівня для авіаційних та аерокосмічних додатків.
CAN Application Layer (CAL) ***Прикладний рівень CAN***	Прикладний рівень, розроблений членами CiA (CAN в автоматизації). Підтримує ряд комунікаційних сервісів та відповідних протоколів.
CAN common ground ***спільне заземлення CAN***	Для кожної мережі CAN необхідна наявність спільного заземлення, що дозволяє уникнути проблем придушення синфазного сигналу. Однак, це створює умови для виникнення небажаних контурів струму у спільній землі.
CAN device ***CAN-пристрій***	Пристрій з CAN-інтерфейсом.
CAN_H ***CAN_H*** ***[Кан хай]***	Лінія CAN_H – трансивера, для відповідного стандарту ISO 11898-2, знаходиться в рецесивном стані при напрузі 2.5 В і в домінантному при 3.5 В.

Продовження додатку З

CAN in Automation (CiA) ***CiA*** ***[Ціа]***	CiA (CAN в автоматизації) є міжнародним співтовариством користувачів і виробників рішень на основі CAN. CiA підтримує також ряд протоколів високого рівня. Засновано в 1992 році
CAN Kingdom ***CAN Kingdom*** ***[Кан кіндем]***	Структура високорівневого протоколу, оптимізована для глибоко вбудованих мереж. CAN Kingdom підходить, зокрема, для додатків реального часу.
CAN_L ***CAN_L*** ***[Кан лоу]***	Лінія CAN_L – трансивера для відповідного стандарту ISO 11898-2 знаходиться в рецесивному стані при напрузі 2.5 В і в домінантному при 1.5 В
CAN Message Specification (CMS) ***Специфікація CAN повідомлення***	Частина специфікації прикладного рівня CAN, що визначає комунікаційні сервіси.
CAN-module ***CAN-модуль***	Реалізація контролера CAN-протоколу спільно з апаратним фільтром вхідних кадрів та буферами повідомлень у складі мікроконтролера або спеціалізованої інтегральної схеми (ASIC).
CAN-node ***CAN-вузол***	Синонім CAN-пристрою.
CANopen ***CANopen*** ***[Каноупен]***	Сімейство профілів, використовуваних для побудови вбудованих мереж в машинобудуванні, медичному обладнанні, автоматизації будівель (системи керування ліфтами, електронними дверима, інтегровані системи контролю приміщень), в залізничної та морської електроніці, вантажних і позашляхових автомобілях тощо.
CANopen application layer ***прикладний рівень CANopen***	Прикладний рівень і комунікаційний профіль CANopen визначаються стандартом EN 50325-2. Цей стандарт описує комунікаційні сервіси та об'єкти. Крім того, він дає специфікацію об'єктного словника і протоколу керування мережею (NMT).
CANopen manager ***CANopen менеджер***	CANopen менеджер відповідає за керування мережею. У CANopen пристрої, що підтримує таке керування , знаходиться NMT-майстер (майстер керування), менеджер SDO (сервісний об'єкт даних) та конфігураційний менеджер.

Продовження додатку З

CANopen Safety ***Безпечний CANopen***	Комунікаційний протокол, що дозволяє здійснювати безпечну передачу даних. Для його реалізації достатньо однієї фізичної CAN-мережі. Надлишковість досягається шляхом передачі кожного повідомлення двічі з побітно інвертованими даними і ідентифікаторами, які відрізняються як мінімум двома бітами.
CAN protocol controller ***контролер CAN-протоколу***	Контролер CAN-протоколу є складовою частиною CAN-модуля і здійснює інкапсуляцію / декапсуляцію даних, бітове хронування, підрахунок CRC, біт-стафінг, обробку помилок, локалізацію відмов, тощо.
CAN transceiver ***CAN-трансивер***	CAN-трансивер служить сполученню CAN-контролера і фізичної шини. Він забезпечує прийом і передачу сигналу в лінію. Існують високошвидкісні, відмовостійкі і однопровідні трансивери, а також трансивери для силових ліній і оптоволоконних кабелів.
CAPL ***CAPL [Капл]***	Мова програмування доступу до CAN(CAPL) використовується при розробці інструментарію для аналізу CAN-мереж.
CCP ***CCP [cicini]***	CAN-протокол калібрування (CCP) використовується для обміну калібрувальними даними в системах автомобільних двигунів.
certification ***сертифікація***	Офіційний тест відповідності компонентів і пристроїв будь-якому стандарту. Університет прикладних наук Німеччини проводить сертифікацію мікросхем CAN-контролерів. Організація ODVA офіційно сертифікує вироби DeviceNet, а CiA здійснює офіційну сертифікацію CANopen-пристроїв.
CiA 102 ***CiA 102*** ***[Ціа 102]***	Проектний стандарт для високошвидкісної передачі даних у відповідності до ISO 11898-2, що передбачає використання 9 - вивідних D-sub роз'ємів.
CiA 201 . . 207 ***CiA 201 . . 207*** ***[Ціа 201 . . 207]***	Специфікації прикладного рівня CAN, що визначають CMS, DBT, NMT і LMT сервіси та протоколи.
CiA 301 ***CiA 301*** ***[Ціа 301]***	Специфікація прикладного рівня і комунікаційного профілю CAN, що визначає функціональність CANopen NMT (протокол керування мережею) slave-пристроїв.
CiA 302 ***CiA 302*** ***[Ціа 302]***	План проектного стандарту для програмованих CANopen-пристроїв. Включає в себе функціональність CANopen-менеджера, динамічні SDO-з'єднання, стандартизовану процедуру завантаження NMT slave-пристроїв, а також завантаження програм.

Продовження додатку З

CiA 303 ***CiA 303*** ***[Ціа 303]***	Проектні рекомендації для CANopen, що стосуються типів кабелю і призначення виводів роз'ємів, кодування префіксів та одиниць міжнародної системи СІ, а також використання світлодіодів.
CiA 304 ***CiA 304*** ***[Ціа 304]***	Специфікація безпечного CANopen-протоколу, схвалена повноважними органами Німеччини та відповідає вимогам додатків SIL класу 3 у відповідності до стандарту МЕК 61508.
CiA 305 ***CiA 305*** ***[Ціа 305]***	Сервіс установки мережевого рівня (LSS), який визначає, яким чином призначати номер вузла та швидкість передачі з використанням CANopen-мережі.
CiA 306 ***CiA 306*** ***[Ціа 306]***	План проектного стандарту , який визначає формат і вміст електронних специфікацій (EDS), що використовуються у конфігураційному інструментарії.
CiA 308 ***CiA 308*** ***[Ціа 308]***	Технічний звіт CANopen, який визначає часові виміри, такі як час обернення PDO, джитер об'єкта синхронізації і час відгуку SDO. Крім того, цей звіт визначає стандартне завантаження шини.
CiA 309 ***CiA 309*** ***[Ціа 309]***	Набір специфікацій, який визначає служби і протоколи мережі TCP / IP, поєднані з CANopen-мережею. Також визначено протоколи для мережі ModbusTCP і протоколи ASCII команд.
CiA 401 ***CiA 401*** ***[Ціа 401]***	CANopen-профіль для модулів введення / виведення загального призначення визначає цифрові й аналогові пристрої введення / виведення.
CiA 402 ***CiA 402*** ***[Ціа 402]***	CANopen-профіль контролерів руху і приводів визначає інтерфейс для інверторів частоти, серво-контролерів, а також крокових двигунів.
CiA 404 ***CiA 404*** ***[Ціа 404]***	CANopen-профіль вимірювальних пристроїв і контролерів зі зворотним зв'язком. Підтримує також багатоканальні пристрої.
CiA 405 ***CiA 405*** ***[Ціа 405]***	CANopen-профіль пристроїв і інтерфейсів для IEC 61131-3 сумісних контролерів, заснований на специфікації CiA DSP 302. Використовується відображення мережевих параметрів в PDO, функціональні блоки для SDO-обміну тощо.
CiA 408 ***CiA 408*** ***[Ціа 408]***	Профіль CANopen-пристроїв для гідравлічних контролерів і пропорційних клапанів. Сумісний з незалежним від мережі VDMA профілем.
CiA 410 ***CiA 410*** ***[Ціа 410]***	Профіль CANopen-пристроїв для вимірювання нахилу. Підтримує як 16-і, так і 32-х розрядні датчики.

Продовження додатку З

CiA 412 ***CiA 412*** ***[Ціа 412]***	Профілі CANopen-пристроїв для медичного обладнання Визначають інтерфейси рентгенівських колиматорів, генераторів рентгенівського випромінювання, медичних стійок і столів.
CiA 413 ***CiA 413*** ***[Ціа 413]***	Профіль CANopen-інтерфейсу для шлюзів внутрішніх мереж вантажних автомобілів. Визначає шлюзи до ISO11992, J1939 та інших внутрішніх мереж. Мережа CANopen як правило використовується для керування додатковим обладнанням вантажівок або трейлерів, наприклад в сміттєзбиральних машинах, підйомних кранах, бетономішалках.
CiA 414 ***CiA 414*** ***[Ціа 414]***	Профіль CANopen-пристрою для ткацьких машин. Визначає інтерфейс механізму, що подає.
CiA 415 ***CiA 415*** ***[Ціа 415]***	Прикладний профіль CANopen для асфальтоукладальників. Визначає інтерфейси різних пристроїв, що використовуються в дорожньобудівельній техніці.
CiA 416 ***CiA 416*** ***[Ціа 416]***	Прикладний профіль CANopen для дверей будівель. Визначає інтерфейси замків, сенсорів та інших пристроїв, що використовуються в керованих електронікою дверях.
CiA 417 ***CiA 417*** ***[Ціа 417]***	Прикладний профіль CANopen для ліфтів. Визначає інтерфейси контролерів кабіни, дверей, контролера виклику та інших, а також вузлів кабіни і дверей , кнопкових панелей , дисплеїв.
CiA 418 ***CiA 418*** ***[Ціа 418]***	Профіль CANopen-пристрою для батарейних модулів. Визначає інтерфейс для роботи із зарядними пристроями.
CiA 419 ***CiA 419*** ***[Ціа 419]***	Профіль CANopen-пристрою для зарядних пристроїв. Визначає інтерфейс для роботи з батарейними модулями.
CiA 420 ***CiA 420*** ***[Ціа 420]***	Сімейство профілів CANopen пристрою для екструдерів. Визначає інтерфейси для витягувача, гофрувача і розпилювача.
CiA 422 ***CiA 422*** ***[Ціа 422]***	Прикладний профіль CANopen для муніципальних транспортних засобів. Використовується, зокрема, для керування додатковим обладнанням сміттєзбиральних машин і визначає інтерфейси таких підсистем, як пресувальник, ваги і т.ін.
CiA 425 ***CiA 425*** ***[Ціа 425]***	CANopen-профіль для допоміжних медичних пристроїв. Визначає plug-and-play інтерфейси інжектора контрастної речовини та електрокардіографа.

Продовження додатку З

Client SDO ***SDO клієнта [есдео]***	Ініціює та виконує обмін даними за допомогою SDO-протоколу шляхом читання або запису об'єктного словника серверного пристрою.
client/server communication ***режим клієнт/сервер***	При передачі даних в режимі клієнт / сервер ініціалізація зв'язку здійснюється клієнтом. Цей обмін завжди реалізується методом точка-точка.
COB-ID ***COB-ID [кобід]***	COB-ID є об'єктом, який визначає CAN-ідентифікатор, а також додаткові параметри, такі, як дійсний / недійсний і підтримка віддаленого запиту.
communication object (COB) ***комунікаційний об'єкт***	Комунікаційним об'єктом називається одне або декілька CAN-повідомлень з певним призначенням, наприклад, PDO, SDO, термінове повідомлення, часова мітка або повідомлення контролю помилок.
communication profile ***комунікаційний профіль***	Комунікаційний профіль визначає вміст комунікаційних об'єктів CANopen, таких, як термінове повідомлення, часова мітка, об'єкти синхронізації, серцебиття, NMT-об'єкти тощо.
Configuration Manager ***конфігураційний менеджер***	Конфігураційний Менеджер (CMT) підтримує механізми конфігурування CANopen-пристроїв при завантаженні.
Confirmed communication ***обмін з підтвердженням***	Для обміну з підтвердженням необхідна двунаправлена передача інформації, коли вузол-одержувач відправляє підтвердження успішного прийому повідомлення.
conformance test plan ***план тесту відповідності***	Визначення протоколів тестування, які повинні бути успішно пройдені для того, щоб було досягнуто відповідність комунікаційному стандарту. План тесту відповідності для CAN-стандартизований ISO 16845.
conformance test tool ***інструментарій тесту відповідності***	Інструментарієм тесту відповідності є реалізація плану тесту відповідності.
connector ***роз'єм***	Електромеханічний компонент, який використовується для під'єднання пристрою до шини CAN, або для подовження шини. Призначення виводів CAN- роз'єму для CAN і CANopen визначається CiA, а для DeviceNet - ODVA.
consumer ***споживач***	Споживачем називається одержувач повідомлення в CAN-мережах. Це означає, що для даного повідомлення приймальний фільтр відкритий.

contention-based arbitration ***арбітраж на основі змагання***	Процедура арбітражу на основі множинного доступу з виявленням несучої (CMSA), при якій спроба захоплення шини здійснюється багатьма вузлами одночасно і для отримання доступу проводяться їх змагання.
control field ***контрольне поле***	Контрольне поле в кадрах даних та віддаленого запиту має довжину 6 біт і включає чотири біта довжини поля даних, біт прапорця розширеного ідентифікатора та зарезервований біт.
CRC ***CRC*** ***[цеерце]***	Циклічний надлишковий код (CRC) у CAN-кадрах даних і віддаленого запиту є числом, розрахованим на основі вмісту блоку даних і призначеним для контролю спотворень цих даних. Розрахувавши CRC для блоку прийнятих даних і порівнявши його зі значенням, отриманим від сторони, яка передає, приймаюча сторона може визначити деякі типи помилок передачі даних.
CRC delimiter ***CRC-разділник*** ***[цеерце]***	Біт CRC-роздільника є останнім бітом поля CRC в кадрах даних та віддаленого запиту. Цей біт завжди встановлюється рецесивним рівнем.
CRC error ***помилка CRC*** ***[цеерце]***	Помилкою CRC є ненульовий результат підрахунку циклічного коду на стороні приймального CAN- вузла. При виявленні цієї помилки наступним за полем підтвердження в мережу передається кадр помилки.
CRC field ***поле CRC*** ***[цеерце]***	Поле CRC в кадрах даних та віддаленого запиту включає в себе 15 бітову CRC-послідовність і 1 біт CRC-роздільника. CRC-послідовність дозволяє виявити до п'яти випадково розподілених спотворень окремих біт у полі початку кадру, арбітражу, контрольному полі, полі даних або до 15 біт пакетної помилки. Відстань Хеммінга визначено рівним 6, без урахування біт-стафінгу.
CSMA/CD+AMP ***CSMA/CD+AMP*** ***[сіесемей/сіді +амп]***	У CAN використовується алгоритм арбітражу за пріоритетом повідомлення на основі множинного доступу з виявленням несучої і детектуванням колізій. Цей метод обслуговує і дозволяє одночасні запити доступу до шини.
cyclic redundancy check (CRC) ***циклічний надлишковий код (CRC)*** ***[цеерце]***	Перевірка циклічного надлишкового коду виконується на основі полінома, що розраховується як в передавальному, так і в приймальному вузлах.

D

data consistency ***спроможність даних***	Механізм глобалізації локальної помилки гарантує спроможність даних для всіх CAN-вузлів мережі, що знаходяться в активному до помилки стані.
data field ***поле даних***	Поле даних CAN-кадру містить від 0 до 8 байт інформації. Його довжина вказується кодом DLC.
data frame ***кадр даних***	CAN-кадр даних передає інформацію від постачальника одному або декільком споживачам. Він складається з біта початку кадру, поля арбітражу, контрольного поля, поля даних, поля CRC, поля підтвердження і поля кінця кадру.
data length code (DLC) ***довжина поля даних (DLC) [діелсі]***	Довжина поля даних CAN-кадру вказується 4-х бітовим кодом в контрольному полі. У кадрах віддаленого запиту DLC повинна збігатися з довжиною поля даних запитуваного кадру!
data type ***тип даних***	Атрибут CANopen-об'єкта, що визначає формат даних. Наприклад: unsigned8 (беззнаковое 8 біт), integer16 (ціле зі знаком 16 біт), boolean (булево / логічне).
data link layer ***канальний рівень***	Другий рівень еталонної моделі ISO / OSI, що забезпечує базові комунікаційні сервіси. Канальний рівень CAN визначає кадри даних, віддаленого запиту, помилки і перевантаження.
DBT ***дистрибутор (DBT) [дібіті]***	Дистрибутор є складовою частиною специфікації прикладного рівня CAN-(CAL) і визначає метод автоматичного розподілу ідентифікаторів при завантаженні мережі.
default value ***значення за замовчуванням***	Атрибут CANopen-об'єкта, що визначає наперед встановлені значення для не конфігурованих користувачем об'єктів після включення живлення або перезапуску програми.
DeviceNet ***DeviceNet [девайснет]***	Протокол високого рівня та визначення профілів пристроїв на основі CAN. DeviceNet був розроблений для автоматизації промислових підприємств і детально описує фізичний рівень CAN для забезпечення високої сумісності та взаємозамінності готових пристроїв. Специфікації DeviceNet підтримуються не комерційною організацією ODVA (www. odva. org).

device profile ***профіль пристрою***	Профіль пристрою визначає специфічні для цього пристрою комунікаційні сервіси, в тому числі детально описує процедури конфігурування.
Diagnostics on CAN ***CAN-діагностика***	Стандарт ISO 15765 визначає протоколи і сервіси CAN-діагностики, які застосовуються в діагностичному CAN-інтерфейсі пасажирських автомобілів.
dominant bit ***домінантний біт***	Домінантний стан біта на CAN-шині. Логічне значення домінантного біта - "0".
dominant state ***домінантний стан***	Домінантний стан за визначенням витісняє (перезаписує) рецесивний.
double-reception of message ***дубльований прийом повідомлення***	Спотворення останнього біта поля кінця кадру(EOF) в передавальному вузлі викличе повторну передачу кадру. Так як після передостаннього біта поля кінця кадру одержувачі вже прийняли повідомлення, вони отримають його двічі.
DR (draft recommendation) ***проектні рекомендації***	Проектні рекомендації ще не визначені остаточно, але вже опубліковані. Проектні рекомендації CiA незмінні протягом одного року.
DS (draft standard) ***проектний стандарт***	Проектний стандарт ще не визначений остаточно, але вже опублікований. Проектні стандарти CiA незмінні протягом одного року.
DSP (draft standard proposal) ***план проектного стандарту***	План проектного стандарту є тільки опублікованою пропозицією. Плани проектного стандарту CiA можуть змінюватися в будь-який час без будь-яких повідомлень.
D-sub connector ***роз'єм D-sub***	Стандартні роз'єми. Найбільш поширений роз'єм, який має 9 виводів D-sub (DIN 41652). Призначення його виводів для CAN-мереж описано в CiA DS 102.

E

EDS checker ***засіб перевірки EDS [ідеес]***	Програмний інструментарій, що дозволяє тестувати узгодженість електронних специфікацій. Засіб перевірки EDS для CANopen може бути завантажений з сайту CiA.
EDS generator ***EDS-генератор***	Програмний інструментарій, що дозволяє створювати електронні специфікації. Доступний для CANopen і DeviceNet.
electronic data sheet	Електронна специфікація описує функціональність

Продовження додатку З

(EDS) ***електронна специфікація (EDS) [ідеес]***	пристрою стандартним чином. CANopen і DeviceNet використовують різні формати EDS.
Emergency ***термінове повідомлення***	Визначений комунікаційний сервіс CANopen, який підтримується одним 8-ми байтовим кадром даних. Кадр складається з двох байт стандартного коду помилки, одного байта регістра помилки і п'яти байт інформації виробника пристрою. Термінове повідомлення використовується для інформування про апаратні і програмні відмови пристрою.
EN 50325 ***EN 50325 [ієн]***	Набір стандартів CENELEC [сінелек], що визначає DeviceNet (частина 2), інтелектуальні розподілені системи (частина 3) і CANopen (частина 4) .
entry category ***категорія запису***	Атрибут об'єкта CANopen, що визначає, чи є цей об'єкт обов'язковим або опціонним.
end of frame (EOF) ***кінець кадру(EOF)[іоф]***	EOF складається з семи рецесивних біт в CAN-кадрах даних і віддаленого запиту.
error active state ***активний до помилки стан***	В активному до помилки стані CAN-контролер може передавати активні кадри помилки, що містять активний прапорець помилки. Якщо всі CAN-вузли активні до помилки, то гарантується спроможність даних у всій мережі.
error code ***код помилки***	У CANopen визначені стандартні коди помилок, що передаються в складі термінового повідомлення.
error control message ***повідомлення контролю помилки***	Повідомлення контролю помилки CANopen представляє собою однобайтовий кадр даних, якому призначено фіксований ідентифікатор, що залежить від номера CAN-вузла. Цей кадр передається в якості повідомлення завантаження при переході вузла мережі в передопераційний стан після ініціалізації, при віддаленому запиті від NMT-майстра (протокол охорони вузла) або періодично самим пристроєм (протокол серцебиття).
error counter ***лічильник помилок***	У кожному CAN-контролері є два лічильника помилок: один для прийнятих повідомлень і один для переданих. Значення лічильників збільшуються і зменшуються прозоро для користувача, забезпечуючи виконання правил, викладених у ISO 118981. Лічильники помилок використовуються для визначення поточного стану CAN-модуля (активне до помилки, пасивне до помилки і відключення шини - *bus-off*).

error delimiter ***роздільник помилки***	Кінцевий сегмент кадру помилки, що складається з 8-ми рецесивних біт.
error detection capability ***здатність виявлення помилок***	У CAN-протоколі використовуються п'ять різних механізмів виявлення помилок, які дають можливість зафіксувати практично будь-яке спотворення CAN-повідомлення. Ймовірність не виявлення помилки залежить від інтенсивності помилок, швидкості передачі, завантаження мережі, числа вузлів і коефіцієнта здібності виявлення помилок.
error flag ***прапорець помилки***	Початковий сегмент кадру помилки, що складається з шести біт одного знака. Один прапорець помилки може перекриватися іншим, що передається ще одним вузлом.
error frame ***кадр помилки***	Передається при виявленні помилки. Складається з прапорця помилки і роздільника помилки.
error globalization ***глобалізація помилки***	Передача кадру помилки, викликана локальною відмовою, буде сприйнята як помилка біт-стафінга, що викличе передачу прапорця помилки іншими вузлами. Таким чином, відбувається глобалізація локальної помилки і гарантується спроможність даних для всіх активних до помилки вузлів мережі.
error passive state ***пасивний до помилки стан***	У пасивному до помилки стані CAN-контролер може передавати тільки пасивні кадри помилки, що містять пасивний прапорець помилки.
error signaling ***сигналізація помилки***	Сигналізація помилки забезпечується шляхом передачі кадрів помилки.
event-driven ***керований подією***	Керовані подією повідомлення передаються, коли у пристрої відбуваються певні зміни. Це може бути зміна стану вхідних параметрів, закінчення проміжку часу або будь-яка інша локальна подія.
event-driven PDO ***керований подією PDO***	Кероване подією PDO передається, коли у пристрої відбувається певна внутрішнє подія. Таким може бути переповнення таймера події PDO. При отриманні керованого подією PDO програмне забезпечення негайно оновлює задані в PDO-відображенні запису об'єктного словника.
event timer ***таймер події***	Таймер події може бути призначений у CANopen кожному TPDO. Він задає частоту передачі цього PDO-об'єкта.

expedited SDO ***прискорений SDO [есдео]***	Прискорений SDO є комунікаційним сервісом з підтвердженням у CANopen (рівний до рівного). Включає в себе одне повідомлення, що ініціює SDO з боку вузла клієнта і відповідне повідомлення про підтвердження від вузла сервера. Прискорені SDO можуть використовуватися для передачі до чотирьох байт даних.
Explicit Message ***явне повідомлення***	Явне повідомлення є комунікаційним сервісом з підтвердженням у DeviceNet, який використовується в конфігураційних цілях. Для обміну інформацією обсягом понад 8 байт, явне повідомлення підтримує сегментовану передачу.
extended frame format ***розширений формат кадру***	У кадрах даних та віддаленого запиту розширеного формату використовуються 29 бітові ідентифікатори.

F

fault confinement ***локалізація відмови***	CAN-вузли в змозі відрізнити короткочасні порушення в своїй роботі від постійної відмови. Несправні передавальні вузли відключаються, що означає їх логічне від'єднання від мережі.
fault-tolerant transceiver ***відомовостійкий трансивер***	У відповідності зі специфікаціями ISO 11898-3 і ISO 11992-1 трансивери можуть працювати по одній лінії шини спільно з CAN-землею, у разі якщо в іншій лінії стався обрив або коротке замикання або погано під'єднанні резистори-термінатори.
flying master ***перемикальний майстер***	У критичних з безпеки додатках може виникати потреба автоматичного заміщення NMT-майстра, що відмовив, іншим резервним NMT-майстром. Такий підхід до надлишковості називається перемикальним майстром.
form error ***помилка формату***	Спотворення одного із зумовлених рецесивних біт (CRC-роздільника, роздільника підтвердження, EOF) розглядається як помилка формату, яка викликає передачу кадру помилки в черговому інтервалі бітового часу.

frame ***кадр***	Сутність протоколу канального рівня, що визначає порядок і значення біт або бітових полів в послідовності, яка передається.
frame coding ***кодування кадру***	Послідовність полів в CAN-кадрах, наприклад, для кадру даних: SOF, поле арбітражу, контрольне поле, поле даних, поле CRC, поле підтвердження і EOF. Кодування кадру також включає в себе біт-стафінг.
frame format ***формат кадру***	Стандарт CAN розрізняє основний формат кадру, який використовує 11 бітові ідентифікатори, і розширений формат з 29 бітовими ідентифікаторами.
frame types ***типи кадру***	У CAN використовуються чотири типи кадрів: кадр даних, кадр віддаленого запиту, кадр помилки і кадр перевантаження.
FullCAN ***FullCAN*** ***[фулкан]***	Реалізація CAN, в якій використовуються єдині буфери прийнятих і переданих повідомлень, що мають різні ідентифікатори.
function code ***код функції***	Перші чотири біти CAN-ідентифікатора в зумовленому розподілі ідентифікаторів CANopen, що вказують функціональне призначення комунікаційного об'єкта (наприклад, TPDO_1 або повідомлення контролю помилок).

G

galvanic isolation ***гальванічна ізоляція***	Для гальванічної ізоляції в CAN-мережах використовуються оптрони або трансформатори, встановлені між CAN-контролером і трансивером.
gateway ***шлюз***	Шлюзом називається пристрій, що має мінімум два мережеві інтерфейси і перетворює всі сім рівнів протоколів еталонної моделі ISO / OSI, наприклад, шлюз між CANopen і Ethernet.
global error ***глобальна помилка***	Глобальна мережева помилка впливає на всі під'єднанні CAN-пристрої.
global fail-safe command ***глобальна команда безпечної відмови***	Глобальна команда безпечної відмови (GFC) є високо пріоритетним CAN-повідомленням, визначеним у безпечному CANopen-протоколі. Її використання скорочує час реакції на відмову. Слідом за GFC-командою повинно бути передано відповідне SRDO.

H

Hamming distance ***відстань Хеммінга***	Відстань Хеммінга дорівнює 6 (теоретичне значення для CAN-мереж), дозволяє виявити до п'яти випадково розподілених спотворень окремих біт.
hard synchronization ***апаратна синхронізація***	Всі CAN-вузли апаратно синхронізуються фронтом біта початку кадру при його виявленні на шині. Апаратна синхронізація відбувається коли шина вільна, при відкладеній передачі і в третьому біті міжкадрового проміжку.
heartbeat ***серцебиття***	Для інформування про працездатність вузла CANopen і DeviceNet використовують повідомлення серцебиття. Його передача здійснюється періодично.
heartbeat consumer time ***період серцебиття споживача***	Період серцебиття споживача визначає час, після закінчення якого вважається, що вузол, який не передав повідомлення серцебиття, втратив працездатність.
heartbeat producer time ***період серцебиття постачальника***	Період серцебиття постачальника визначає частоту передачі повідомлення серцебиття.
higher-layer protocol (HLP) ***протокол високого рівня***	Протоколи високого рівня відповідають комунікаційним протоколами транспортного, сеансового, рівня представлення даних або прикладного рівня еталонної моделі ISO / OSI.
high-speed transceiver ***високошвидкісний трансивер***	Трансивери, що підтримують швидкість передачі до 1 Мбіт / с відповідно до ISO 11898-2.

I

identifier ***ідентифікатор***	CAN-ідентифікатор визначає вміст кадру даних. Ідентифікатор кадру віддаленого запиту відповідає ідентифікатору запитуваного кадру . Ідентифікатор неявно задає пріоритет арбітражу повідомлення в CAN-мережі.

Продовження додатку З

identifier extension flag (IDE) ***прапор розширеного формату***	Прапорець розширеного формату (біт IDE) вказує, яким чином інтерпретуються наступні біти: як контрольні або ж задають другу частину 29 бітового ідентифікатора.
identifier field ***поле ідентифікатора***	Поле ідентифікатора містить 11 біт для кадру основного формату і додатково 18 біт для розширеного формату кадру.
index ***індекс***	Індексом є 16 бітова адреса, що використовується для доступу до об'єктного словника CANopen. Для масивів і записів ця адреса розширюється 8 двійковим субіндексом.
inhibit timer ***таймер придушення***	CANopen-об'єкт для PDO і термінового повідомлення, який здійснює заборону передачі протягом певного часу (час придушення) відповідного комунікаційного об'єкта.
initialization state ***стан ініціалізації***	Стан NMT slave-пристрою CANopen, в яке воно переходить автоматично після включення живлення і при скиданні комунікації або програми.
inner priority inversion ***внутрішня інверсія пріоритету***	До внутрішньої інверсії пріоритету призводять випадки, коли низькопріоритетне повідомлення не може бути передано в CAN-мережу внаслідок значного трафіку високопріоритетних повідомлень. При цьому, якщо у пристрої виникає запит на передачу високо-пріоритетного повідомлення, він не може бути задоволений CAN-контролером, оскільки низькопріоритетне повідомлення все ще очікує своєї черги на відправлення.
interface profile ***профіль інтерфейсу***	CANopen-профіль, який визначає тільки інтерфейс, а не прикладну поведінку пристрою, наприклад профіль шлюзів і мостів.
interframe space ***міжкадровий проміжок***	Міжкадровий проміжок формується трьома рецесивними бітами. Він поділяє всі CAN-кадри, в тому числі кадри помилки і перевантаження.
intermission field ***поле паузи***	Синонім міжкадрового проміжку (interframe space).
I/O Message ***Повідомлення введення - виведення (I / O [io])***	Комунікаційний об'єкт DeviceNet (з об'єктів транспортної програми), що представляє введення або виведення. I / O повідомлення відображаються одним або кількома CAN-кадрами даних з підтримкою сегментованої передачі.

Продовження додатку З

ISO 11898-1 ***ISO 11898-1 [ico]***	Міжнародний стандарт, що визначає канальний рівень CAN, в тому числі підрівні LLS (служби налаштування рівня), MAC (контроль доступу до комунікаційного середовища) і PLS (фізичні сигнали).
ISO 11898-2 ***ISO 11898-2 [ico]***	Міжнародний стандарт, що визначає високошвидкісне CAN- MAU (пристрій приєднання до комунікаційного середовища).
ISO 11898-3 ***ISO 11898-3 [ico]***	Міжнародний стандарт, що визначає низькошвидкісне, відмовостійке CAN- MAU (пристрій приєднання до комунікаційного середовища).
ISO 11898-4 ***ISO 11898-4 [ico]***	Міжнародний стандарт, що визначає синхрочасовий комунікаційний протокол на основі CAN.
ISO 11992 ***ISO 11992 [ico]***	Міжнародний стандарт, що визначає прикладні CAN-профілі для вантажівок , трейлерів. Друга частина цього стандарту містить специфікації для гальмівних пристроїв і коробок передач, частина 3 описує профілі інших пристроїв, а четверта частина - діагностичні системи.
ISO 11745-2 ***ISO 11645-2 [ico]***	Міжнародний стандарт, що визначає структуру інтеграції додатків для систем керування на основі ISO11898, таких, як CANopen і DeviceNet.
ISO 11783 ***ISO 11783 [ico]***	Міжнародний стандарт, що визначає прикладний CAN-профіль, який використовується в сільськогосподарських і лісівницьких машинах. Заснований на профілі J1939.
ISO 16844 ***ISO 16844 [ico]***	Міжнародний стандарт, що визначає тахограф на основі CAN, який використовується у вантажівках і автобусах.
ISO 16845 ***ISO 16845 [ico]***	Міжнародний стандарт, що визначає план тесту відповідності для реалізацій ISO 11898-1.

J

J1939 application profile ***прикладний профіль J1939 [джей1939]***	Прикладний профіль, розроблений Співтовариством Інженерів Автомобілебудування (SAE), задає специфікацію внутрішніх інформаційних комунікацій для вантажівок і автобусів. Профіль J1939 визначає комунікаційні сервіси і сигнали, в тому числі відображення даних в CAN-кадри за допомогою PGN (номерів груп параметрів).
J2284 bit timing ***бітовий час [джей2284]***	Визначення бітового часу для внутрішніх мереж легкових автомобілів при швидкостях шини 250 кбіт /с і 500 кбіт / с. Запропоновано SAE.
J2411 single-wire CAN ***однопровідний CAN J2411 [джей2411]***	Специфікація однопровідної передачі для CAN-мереж, розроблена SAE. Бітова швидкість обмежена величиною 40кбіт/с.

L

layer-2 protocol ***протокол рівня 2***	Протокол рівня 2 використовує комунікаційні сервіси CAN безпосередньо, не спираючись на певний високорівневий протокол.
layer-7 protocol ***протокол рівня 7***	Протокол рівня 7 використовує комунікаційні сервіси CAN стандартизованим чином. Це надає можливість повторного використання прикладного програмного забезпечення без переробки комунікаційного програмного забезпечення CAN.
layer setting services (LSS) ***сервіси встановлення рівня (LSS) [елесес]***	CANopen-сервіси встановлення рівня (LSS) визначають комунікаційні сервіси для конфігурування номеру вузла і бітової швидкості з використанням CAN-мережі.
life guarding ***охорона працездатності***	Метод, що дозволяє виявити, що NMT-майстер більше не спостерігає працездатність NMT slave-пристрою. Використовується на прикладному рівні CAN і в CANopen.

line topology ***лінійна топологія***	Топологія мережі, де всі вузли безпосередньо приєднуються до єдиної шині. Теоретично CAN-мережі використовують тільки лінійну топологію без будь-яких відгалужень. Тим не менш, на практиці зустрічаються також деревовидні і зіркоподібні топології.
LMT protocols ***LMT протокол [елемті]***	Протокол, визначений на прикладному рівні CAN(CAL) , що задає номер вузла і бітову швидкість з використанням CAN-мережі.
local bus error ***локальна помилка шини***	Локальна помилка шини впливає на один чи кілька, але не на всі вузли мережі.
logical link control (LLC) ***контроль логічного з'єднаня (LLC) [елелсі]***	Підрівень контролю логічного з'єднання (LLC) описує верхні компоненти канального рівня ISO / OSI. Він стосується тих аспектів протоколу, які не залежать від методу доступу до комунікаційного середовища.
low-power mode ***режим зниженого енергоспоживання***	CAN-контролер і CAN-трансивер можуть підтримувати режим зниженого енергоспоживання (режим stand-by), коли потужність споживання менша, ніж у робочому режимі.
low-speed transceiver ***низькошвидкісной трансивер***	Синонім відмовостійкого трансивера.
LSS protocols ***протоколи LSS [елесес]***	Сервіси установки рівня (LSS) визначають протоколи, що задають номер вузла і бітової швидкості з використанням CAN-мережі.

M

master ***майстер***	Комунікаційна або прикладна сутність, якій дозволено контролювати певні операції. У мережевих програмах такою операцією може бути, наприклад, ініціалізація комунікаційних сервісів.
master/slave communication ***майстер/slave комунікації [/слейв]***	У майстер / slave комунікаціях ініціалізацію і контроль процесу обміну здійснює майстер. Пристрою slave не дозволено виступати ініціатором ніяких комунікаційних операцій.
matrix cycle ***матричний цикл***	У TTCAN матричний цикл складається з одного або декількох основних циклів. Кожен основний цикл починається з реперної посилки, але за нею можуть слідувати різні вікна.

medium access control (MAC) ***контроль доступу до комунікаційного середовища (MAC) [мак]***	Підрівень контролю доступу до комунікаційного середовища (MAC) представляє нижні компоненти канального рівня ISO / OSI. Він обслуговує інтерфейси до LLC-підрівня і фізичного рівня. MAC включає в себе правила і функції, пов'язані з інкапсуляцією / декапсуляцією, виявленням помилок і видачею сигналів.
medium dependent interface (MDI) ***інтерфейс,залежний від середовища (MDI)[емдіай]***	MDI визначає вимоги до роз'ємів, кабелів та термінаторів.
message ***повідомлення***	CAN-повідомленням є кадр даних чи кадр віддаленого запиту.
message buffer ***буфер повідомлень***	Мікросхеми CAN-контролерів підтримують буфери повідомлень для прийнятих і / або переданих кадрів. Реалізація та алгоритми роботи з буферами повідомлень не стандартизовані.
multicast transmission ***мультимовна передача***	Метод адресації, коли один кадр направляється декільком вузлам одночасно.
multi-master communication ***мульти-майстерні комунікації***	У мульти-мастерних комунікаційних системах кожен з вузлів може тимчасово контролювати обмін даними в мережі. Це означає, що кожен вузол теоретично має право доступу до шині в будь-який час, за умови що вона вільна.
multiplexed PDO (MPDO) ***мультиплексований PDO (MPDO) [емпедео]***	Мультиплексований PDO складається з 8 байт, в тому числі одного контрольного байта, трьох байт мультиплексора (містить 16 бітовий індекс і 8 бітовий субіндекс) і чотирьох байт власне даних об'єкта.

N

network length ***довжина мережі***	Див bus length (довжина шини).

network management ***менеджер мережі***	Менеджер мережі несе відповідальність за процедуру завантаження і опціонне конфігурування вузлів. Менеджер може також виконувати функції контролю стану вузлів мережі, наприклад, охорону вузла.
network variables ***мережеві змінні***	Мережеві змінні використовуються в програмованих CANopen-пристроях для відображення в PDO за завершенням програмування пристрою.
NMT ***NMT [енемте]***	Менеджер мережі (мережевий менеджер) у прикладному рівні CAN і CANopen.
NMT master ***NMT-майстер [енемте]***	NMT-майстер здійснює менеджмент мережі шляхом передачі NMT-повідомлень. З їх допомогою майстер керує NMT-діаграмами станів всіх приєднаних NMT slave-пристроїв.
NMT slave ***NMT slave [енемте слейв]***	NMT slave приймає NMT-повідомлення, що містять команди керування для NMT-діаграмами станів, що реалізована на прикладному рівні CAN і CANopen пристроїв.
NMT slave state machine ***машина станів NMT slave [енемте]***	Машина (діаграма) станів NMT slave-пристрою визначена для прикладного рівня CAN і CANopen. NMT-майстер керує переходом пристрою з одного стану в інший шляхом передачі CAN-повідомлення з найвищим пріоритетом.
node ***вузол***	Модуль, приєднаний до CAN-мережі і здатний здійснювати комунікації відповідно до CAN-протоколів.
node guarding ***охорона вузла***	Механізм, який використовується на прикладному рівні CAN і CANopen для виявлення від'єднаних пристроїв, або що знаходяться в стані bus-off. NMT-майстер посилає кадр віддаленого запиту для NMT slave, на який той відповідає відповідним повідомленням контролю помилки.
node-ID ***ідентифікатор вузла***	Унікальний ідентифікатор пристрою, який використовується різними високорівневими CAN-протоколами, наприклад, CANopen або DeviceNet, при призначенні цьому пристрою CAN-ідентифікаторів. Для зумовленого розподілу ідентифікаторів CANopen або DeviceNet ідентифікатор вузла є частиною CAN-ідентифікатора.

nominal bit-rate ***номінальна бітова швидкість***	Номінальна бітова швидкість визначається числом біт у секунду, яке передається ідеальним передавачем у відсутності ресинхронізації.
nominal bit-time ***номінальний бітовий час***	Номінальний бітовий час може бути представлено як розділений на окремі часові сегменти, що не перекриваються.
non-return to zero (NRZ) ***без повернення до нуля (NRZ) [енерзет]***	Метод подання двійкового сигналу, при якому його рівень незмінний протягом одного бітового часу. При кодуванні без повернення до нуля необхідна можливість ресинхронізації, оскільки бітовий потік може мати один і той же логічний знак, а значить в сигналі не буде будь-яких перепадів рівня.

О

object dictionary ***об'єктний словник***	Об'єктний словник є серцевиною кожного CANopen-пристрою і містить описи всіх комунікаційних і прикладних об'єктів
operational state ***операційний стан***	В операційному NMT-стані доступні всі комунікаційні сервіси CANopen.
OSEK/VDX ***OSEK/VDX [осек/видиекс]***	Набір специфікацій, що визначають комунікації (COM), менеджмент мережі (NM), операційну систему реального часу (OS), і мову реалізації (OIL). OSEK / VDX частково реалізований у легкових автомобілях.
OSI (open system interconnection) reference model ***еталонна модель ISO/OSI [ісо/осі]***	Семирівнева еталонна модель протоколів передачі даних. Визначає рівні: фізичний, канальний, мережевий, транспортний, сеансовий, представницький і прикладний рівень. У CAN-мережах звичайно реалізуються тільки фізичний, канальний і прикладний рівні.

outer priority inversion ***зовнішня інверсія пріоритету***	Якщо вузлу необхідно передати два високо пріоритетних CAN-повідомлення, але він не може відіслати друге з них відразу слідом за міжкадровим проміжком, інший вузол мережі може встигнути передати низькопріоритетне повідомлення. Така ситуація називається зовнішньою інверсією пріоритету.
overload condition ***стан перевантаження***	Ситуації, при яких CAN-контролер передає кадр перевантаження, наприклад, домінантний рівень серед перших двох біт міжкадрового проміжку, домінантне значення останнього біта EOF, невірне значення біт в кадрах помилки або перевантаження.
overload delimiter ***роздільник перевантаження***	Кінцевий сегмент кадру перевантаження, що складається з 8-ми рецесивних біт.
overload flag ***прапорець перевантаження***	Початковий сегмент кадру перевантаження, що складається з шести домінантних біт. Один прапорець перевантаження може перекриватися іншим, переданим ще одним вузлом.
overload frame ***кадр перевантаження***	Кадр для позначення стану перевантаження.

Р

parameter group number (PGN) ***номер групи параметрів (PGN) [піджіен]***	У стандартах SAE J1939, ISO 11783 та ISO11992 визначені номери груп параметрів (PGN-и), які визначають вміст відповідного CAN-повідомлення. PGN відображається в 29 бітовому ідентифікаторі.
passive error flag ***пасивний прапорець помилки***	Пасивний прапорець помилки є початковою частиною пасивного кадру помилки і складається з шести послідовних рецесивних біт.
PDO mapping ***PDO-відображення [педео]***	У кожен PDO може бути відображено до 64 об'єктів. PDO-відображення описується параметрами PDO-відображень.
pending transmission request ***запит на передачу,що очікує***	Внаслідок зайнятості шини або з причини програшу вузлом процедури арбітражу в CAN-контролері може зберігатися один і більше повідомлень, які очікують передачі в мережу.

phase error ***фазова помилка***	Фазова помилка фронту сигналу задається його положенням щодо сегмента синхронізації (Sync_Seg). Вимірюється в квантах часу (time quanta).
phase segment 1 *(Phase_Seg 1)* ***фазовий сегмент 1***	Частина бітового часу, що використовується для компенсації фазової помилки фронту сигналу. Може бути подовжений при ресинхронізації.
phase segment 2 *(Phase_Seg 2)* ***фазовий сегмент 2***	Частина бітового часу, що використовується для компенсації фазової помилки фронту сигналу. Може бути укорочений при ресинхронізації.
physical layer ***фізичний рівень***	Нижчий рівень еталонної моделі ISO / OSI, що визначає роз'єми, типи кабелю, електричні або оптичні сигнали,що задають рівень біт, а також синхронізацію і ресинхронізацію.
physical signaling *(PLS)* ***фізичні сигнали*** ***(PLS) [пієлс]***	Підрівень фізичного рівня. Приймає і передає в схему трансивера бітовий потік. Виконує кодування /декодування біт, контролює бітове хронування і синхронізацію.
pin assignment ***призначення виводів***	Визначення призначення виводів роз'ємів.
pre-defined *connection set* ***визначений розподіл*** ***ідентифікаторів***	Набір CAN-ідентифікаторів, що використовуються за замовчуванням для різних комунікаційних протоколів CANopen або DeviceNet.
Pre-operational state ***передопераційний стан***	У передопераційному NMT-стані комунікаційні об'єкти PDO заборонені.
priority ***пріоритет***	Атрибут кадру, що визначає його арбітражний ранг. У CAN-кадрах даних та віддаленого запиту пріоритет задається ідентифікатором. Чим менше його значення, тим вищий пріоритет кадру.
Process Data Object *(PDO)* ***об'єкт даних*** ***процесу (PDO)*** ***[педэо]***	Комунікаційний об'єкт, який визначається комунікаційним PDO-параметром і параметром PDO-відображення. Відноситься до комунікаційних сервісів, які не підтверджуються, і не призводить до надлишковості протоколу.
producer ***постачальник***	У CAN-мережах постачальником називається пристрій, який здійснює передачу повідомлень.
propagation segment *(Prop_Seg)* ***сегмент*** ***розповсюдження***	Частина бітового часу, що використовується для компенсації фізичних затримок сигналу в мережі. Повний час затримки включає в себе час поширення сигналу шиною і час внутрішньої затримки сигналу у вузлах мережі.

protocol ***протокол***	Формальний набір угод і правил, що описує обмін інформацією між вузлами мережі, в тому числі специфікації керування кадром, передачі кадру та фізичного рівня.
priority inversion ***інверсія пріоритету***	Інверсія пріоритетів відбувається, коли об'єкт з низьким пріоритетом обробляється або передається перш об'єкта високого пріоритету. У недостатньо добре спроектованих CAN-пристроях може спостерігатися внутрішня або зовнішня інверсія пріоритету.

R

receive error counter (REC) ***лічильник помилок прийому (REC) [рек]***	Внутрішній лічильник помилок прийому CAN-контролера. У деяких контролерах значення цього лічильника доступне для читання.
receiver ***приймальний вузол***	CAN-вузол називається приймальним вузлом або споживачем, якщо він не є передавальним вузлом і шина не вільна.
reception buffer(s) ***приймальні буфери***	Локальна пам'ять CAN-контролера, що використовується для проміжного зберігання отриманих повідомлень.
recessive bit ***рецесивний біт***	Рецесивний стан біта на CAN-шині. Логічне значення рецесивного біта - "1".
recessive state ***рецесивний стан***	За визначенням, рецесивний стан витісняється (перезаписується) домінантним.
recovery time ***час відновлення***	Проміжок часу між першим бітом прапорця помилки і моментом, коли може бути розпочато автоматична повторна передача. В активних до помилки вузлах максимальний час відновлення становить 23 інтервали бітового часу, в пасивних до помилки вузлах – 31 інтервал бітового часу.
redundant networks ***надлишкові мережі***	У ряді критичних за безпекою програм (наприклад, морські системи) можуть використовуватися надлишкові мережі, які забезпечують можливість перемикання з однієї мережі на іншу при виявленні комунікаційних помилок.

Reference message ***опорне повідомлення***	У системі TTCAN кожен основний цикл починається з опорного повідомлення.
remote frame ***кадр відаленного запиту***	З допомогою кадру віддаленого запиту вузол мережі запитує передачу іншим вузлом відповідного кадру даних з тим же самим значенням ідентифікатора. Код довжини поля даних (DLC) віддаленого запиту повинен мати те ж значення,що і DLC відповідного кадру даних. Довжина поля даних кадру віддаленого запиту - нуль байт.
remote transmit request (RTR) ***віддалений запит {на передачу} (RTR)*** ***[ертеер]***	Біт поля арбітражу, який вказує, чи є даний кадр кадром віддаленого запиту (рецесивний рівень) або кадром даних (домінантний рівень).
repeater ***репі тер***	Пасивний пристрій, що повторює сигнали на CAN-шині. Репітер використовується для збільшення максимально можливого числа вузлів , що під'єднуються до шини, або для збільшення протяжності мереж (понад 1 км) або для реалізації деревоподібної або складної топології.
Reset ***скидання***	Скидання CAN-контролера здійснюється за допомогою команди (можливо, що подається апаратно). До переходу в активний до помилки стан CAN-контролер повинен зареєструвати 128 одинадцяти бітових рецесивних сегментів.
reset application ***скидання програми***	NMT-команда скидання програми встановлює значення за замовчуванням або збережені конфігуровані значення для всіх об'єктів CANopen-пристрою.
reset communication ***скидання комунікацій***	NMT-команда скидання комунікацій встановлює значення за замовчуванням або збережені конфігуровані значення тільки для комунікаційних об'єктів CANopen-пристрою.
re-synchronization jump width (SJW) ***стрибок ресинхронізації***	Число квантів часу, на яке може бути подовжений фазовий сегмент 1 (Phase_Seg 1) або укорочений фазовий сегмент 2 (Phase_Seg 2).
RPDO ***RPDO [ерпедео]***	RPDO (об'єкт даних процесу, що приймається) є комунікаційним об'єктом, який приймається CANopen-пристроєм.

S

sample point ***точка вибірки***	Точкою вибірки є момент часу, коли зчитується сигнал на шині й значення відповідного біта. Вона розташовується між фазовим сегментом 1 і фазовим сегментом 2.
SCT ***SCT [ессіті]***	Тривалість циклу безпеки (Safe-guard Cycle Time) визначає максимальний інтервал часу між двома періодично переданими SRDO (безпечний об'єкт даних).
SDO block transfer ***блокова передача SDO [есдео]***	Блокова передача SDO - це комунікаційний сервіс CANopen, призначений для збільшення продуктивності SDO-обміну. При блоковій передачі підтвердження висилається лише після прийому певної кількості SDO-сегментів.
SDO manager ***SDO-менеджер [есдео]***	SDO-менеджер керує динамічною установкою SDO-з'єднань. Він розміщується на тому ж сайті, що і NMT майстер.
segmented SDO ***сегментований SDO [есдео]***	Сегментований SDO використовується для передачі об'єктів довжиною понад 4 байти. Повне число сегментів в циклі передачі теоретично не обмежена.
Server SDO ***SDO сервера [есдео]***	SDO-сервер приймає SDO-повідомлення від відповідного SDO-клієнта і повертає відповідь на кожне повідомлення або на блок повідомлень (блокова передача SDO).
Service Data Object (SDO) ***сервісний об'єкт даних (SDO) [есдео]***	Сервісні об'єкти даних забезпечують доступ до записів об'єктного словника CANopen. Кожний SDO формується як мінімум двома CAN-повідомленнями з різними ідентифікаторами. SDO – це комунікаційний сервіс типу точка-точка, що підтверджується.
single-shot transmission ***одноразова передача***	Ряд CAN-контролерів підтримує режим одноразової передачі, коли в разі виявлення помилки не здійснюється автоматична повторна посилка повідомлення. Такий режим необхідний для протоколу TTCAN.

single-wire CAN (SWC) ***однопровідний CAN***	Фізичний рівень, де використовується тільки одна лінія шини і CAN-земля. Спільнота Інженерів Автомобілебудування (SAE) розробила специфікацію SWC-трансивера(J2411).
SI unit ***Система СІ***	Міжнародна система одиниць фізичних величин, визначена стандартом ISO 1000:1983.
sleep mode ***сплячий режим***	CAN-контролер та трансивер можуть перебувати у сплячому режимі, або режимі зниженого енергоспоживання, відключаючись при цьому від шини.
SRDO ***SRDO[есердео]***	У безпечному CANopen-протоколі визначено безпечний об'єкт даних (Safety Related Data Object), який складається з двох CAN-повідомлень. Поле даних іншого повідомлення формується з побітно інвертованих даних першого повідомлення.
SRVT ***SRVT[есервити]***	Час достовірності безпечного об'єкта даних (Safety Related object Validation Time) визначає максимальний інтервал часу між двома CAN-повідомленнями, що складають SRDO.
start of frame (SOF) ***початок кадру (SOF) [соф]***	Самий перший біт кожного кадру даних і кадру віддаленого запиту. Стан SOF завжди домінантний.
star topology ***топологія зірка***	У деяких легкових автомобілях використовується зіркоподібна топологія CAN-мереж, а термінатори мережі підключені в центрі цієї зірки.
Stopped state ***стан зупинки***	NMT-стан, в якому відпрацьовуються тільки NMT-повідомлення і при деяких умовах також передаються повідомлення контролю помилок.
stuff-bit ***стаф-біт***	Кожного разу, коли CAN-передавач виявляє в бітовому потоці 5послідовних біт одного знака, він автоматично виконує вставку біта протилежного знака (стаф-біт). CAN-приймач автоматично видаляє стаф-біти, так що отримане повідомлення стає ідентичним переданому. Біт-стафінг використовується для автоматичної ресинхронізації у схемі бітового хронування CAN-модуля.
stuff error ***помилка біт-стафінга***	Помилка біт-стафінга відбувається, коли на шині виявляється шостий послідовний біт одного знака в полях початку кадру, арбітражу, контрольному полі, полі даних і полі CRC.
sub-index ***субіндекс***	8-бітова субадреса в об'єктному словнику для реалізації доступу до елементів масивів і записів.

substitute remote request (SRR) ***заміна віддаленного запиту (SRR) [есерер]***	Біт кадру розширеного формату, замісник RTR-біта (біт віддаленого запиту), розміщується у повідомленні за першою частиною ідентифікатора (11 біт). SRR-біт має рецесивний рівень.
suspend transmission ***відкладена передача***	У пасивному до помилки стані CAN-контролер повинен додатково очікувати 8 одиниць бітового часу до того, як починати передачу чергового кадру даних або віддаленого запиту.
SYNC message ***посилка синхронізації SYNC [сінк]***	Спеціальне CANopen-повідомлення, що дає вказівку приймальним вузлам вважати значення входів, відображених в синхронні TPDO. Отримання цього повідомлення також наказує вузлу встановити значення виходів, які були прийняті попереднім синхронним RPDO.
sync segment (Sync_Seg) ***сегмент синхронізації***	Частина бітового часу, що служить для синхронізації різних вузлів мережі. У межах сегмента синхронізації очікується поява фронту сигналу.

Т

termination resistor ***резистор - термінатор***	У високошвидкісних CAN-мережах з лінійною топологією для зменшення відбиття сигналу до обох кінців шини приєднуються резистори-термінатори.
thick cable ***товстий кабель***	Товстий кабель використовується в DeviceNet-мережах, протяжністю понад 100 метрів.
thin cable ***тонкий кабель***	Тонкий кабель використовується в DeviceNet мережах для відгалужень і в випадку, якщо протяжність мережі не перевищує 100 метрів.
TIME message ***посилка часової мітки TIME [тайм]***	Стандартизоване CANopen-повідомлення, яке містить значення абсолютного часу, представлене шістьма байтами в вигляді числа мілісекунд після опівночі і кількості днів після 1січня 1984 року.
time quanta ***квант часу***	Одиниця атомарного часу для CAN-мережі.

time stamp ***часова мітка***	Деякі CAN-контролери забезпечують можливість присвоєння часової мітки кожному прийнятому повідомленню. У протоколі TTCAN-рівня 2 передавальні вузли також повинні мати здатність захоплювати значення часу і включати часову мітку в поле даних поточного повідомлення.
time-triggered ***синхро-часовий***	Синхро-часові повідомлення передаються в зумовлені інтервали часу (часові слоти). Це призводить до необхідності глобальної синхронізації часу і запобіганню автоматичної повторної передачі спотворених повідомлень. Синхро-часові CAN-комунікації стандартизовані в ISO 11898-4 (TTCAN).
topology ***топологія***	Фізична структура мережевих з'єднань, наприклад, шина, кільце, зірка, деревоподібна топологія.
TPDO ***TPDO [тепедео]***	TPDO (об'єкт даних процесу, що передається) є комунікаційним об'єктом, який передається CANopen-пристроєм.
transmission buffer(s) ***передавальні буфери***	Локальна пам'ять CAN-контролера, що використовується для зберігання повідомлень,що очікують передачу.
transmission request ***запит на передачу***	Внутрішня подія в CAN-контролері, що виникає перед передачею повідомлення.
transmission time capture ***захоплення моменту передачі***	У протоколі TTCAN-рівня 2 повинно забезпечуватися захоплення моменту часу передачі біта початку кадру(SOF) опорного повідомлення.
transmission type ***тип передачі***	Об'єкт CANopen, визначає режими прийому і передачі PDO.
transmit error counter (TEC) ***лічильник помилок передачі(TEC) [тек]***	Внутрішній лічильник помилок передачі CAN-контролера У деяких контролерах значення лічильника TEC доступно для читання.
transmitter ***передавальний вузол***	Вузол, що породжує кадр даних або віддаленого запиту. Він залишається передавальним вузлом, поки шина для передачі знову не стане вільною або цей вузол не програє арбітраж.
tree topology ***деревоподібна топологія***	Мережева топологія з магістральною сполучною шиною і лініями відгалужень. Не забезпечені термінаторами відгалуження можуть викликати відображення сигналів, рівень яких не повинен

	перевищувати критичного значення.
TSEG1 ***TSEG1 [тсег1]***	Значення TSEG1 включає в себе сегмент розповсюдження (Prop_Seg) і фазовий сегмент 1 (Phase_Seg 1) бітового часу.
TSEG2 ***TSEG2 [тсег2]***	Значення TSEG2 дорівнює тривалості фазового сегмента 2 (Phase_Seg 2) бітового часу.
TTCAN-протокол ***протокол TTCAN [тітікан]***	Протокол високого рівня, що визначає синхро-часові комунікаційні режими в мережі CAN. При використанні TTCAN-протоколу CAN-контролери повинні мати можливість відключати автоматичну повторну передачу спотворених повідомлень. Крім того, контролери повинні бути здатні захоплювати 16-бітове значення таймера при передачі біта початку кадру (SOF) для того, щоб включати це значення в поточне повідомлення.

V

value definition ***визначення значень***	Детальний опис діапазону значень в CANopen-профілях.
value range ***діапазон значень***	Атрибут CANopen-об'єкта, що визначає його допустимі значення.

W

wake-up procedure ***процедура пробудження***	Спеціальна процедура, що може бути використана для пробудження CAN-вузлів, що знаходяться в сплячому режимі, або режимі зниженого енергоспоживання.

Додаток К

Робоча програма керування CAN-модулем

```
/* Тактова частота мікроконтролера. */
#define F_CPU 4000000UL

#include <avr/interrupt.h>
#include <util/delay.h>

/*
 * Ідентифікатори повідомлень, довжина повідомлення і маски для
 * використовуваних MOb.
 */
#define MOB0_ID     0x000012CEL
#define MOB0_MASK   0xFFFFFFFFL
#define MOB0_DLC    6
#define MOB1_ID     0x00000123L
#define MOB1_MASK   0xFFFFFFFFL
#define MOB1_DLC    4               // очікувана довжина повідомлення

#define MOB_STRICT_MASK 0xFFFFFFFFL

#define DELAY       5

/* Дані для MOb0. */
uint8_t mob0_data[MOB0_DLC] = {0x61, 0x62, 0x63, 0x64, 0x65, 0x66};

/*
 * Налаштування мультикадру:
 *  - ідентифікатор повідомлень
 *  - кількість блоків по 8 байт, що мають бути передані
 *  - дані, що мають бути передані
 */
#define MULTIFRAME_ID   0x022B
#define MULTIFRAME_LEN  3

uint8_t multiframe_data[MULTIFRAME_LEN*8] = {
             0x10, 0x0E, 0x21, 0x22, 0x23, 0x24, 0x25, 0x26,
             0x21, 0x27, 0x28, 0x29, 0x30, 0x31, 0x32, 0x33,
             0x31, 0x34, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF};

/* Повідомлення. */
typedef struct {
    uint32_t id; // ідентифікатор повідомлення
    uint8_t length; // довжина повідомлення
    uint8_t data[8]; // масив байтів повідомлення
} CAN_message;

/* Мультикадр */
typedef struct {
    uint32_t id;
    uint16_t buffered_mobs;
    uint8_t data[8*15];
} CAN_buffer;

/* Отриманий мультикадр копіюється в глобальну змінну buffer. */
CAN_buffer buffer;

/* Отримане повідомлення копіюється в глобальну змінну rx_message. */
CAN_message rx_message;
```

```
void set_frame_buffer(CAN_buffer *buf) {
    /* Зберегти поточний MOb. */
    uint8_t save_canpage = CANPAGE;

    /*
     * Необхідно заборонити переривання від Mob'ів, що входять у
     * мультикадр, для правильної обробки переривання RXOK. Блок 5.2
     */
    CANIE2 &= (uint8_t) ~buf->buffered_mobs;
    CANIE1 &= (uint8_t) ~(buf->buffered_mobs >> 8);

      /* Блок 5.3 */
    for (uint8_t i=0; i<16; i++) {
            if ((buf->buffered_mobs >> i) & 1){
            CANPAGE = i << 4;

            /* Встановити ідентифікатор та біт RTR. */
            CANIDT4 = (uint8_t) ((buf->id << 3) | (0 << RTRTAG));
            CANIDT3 = (uint8_t) (buf->id >> 5);
            CANIDT2 = (uint8_t) (buf->id >> 13);
            CANIDT1 = (uint8_t) (buf->id >> 21);

            /* Встановити маску для перевірки IDE, біта RTR та ідентифікатора.
*/
            CANIDM4 = (uint8_t) ((MOB_STRICT_MASK << 3) | (1 << RTRMSK) | (1 <<
IDEMSK));
            CANIDM3 = (uint8_t) (MOB_STRICT_MASK >> 5);
            CANIDM2 = (uint8_t) (MOB_STRICT_MASK >> 13);
            CANIDM1 = (uint8_t) (MOB_STRICT_MASK >> 21);

            /* Встановити режим прийому та IDE */
            CANCDMOB = (3 << CONMOB0) | (1 << IDE) | (8 << DLC0);
        }
    }

    /* Відновити поточний MOb. */
    CANPAGE = save_canpage;
}

void get_frame_buffer(CAN_buffer *buf) {
    /* Зберегти поточний MOb. */
    uint8_t save_canpage = CANPAGE;

    /* Кількість вже оброблених Mob'ів у мультикадрі */
    uint8_t buf_length = 0;
    //Блок 9.1
    for (uint8_t i=0; i<16; i++) {
        if ( ((buf->buffered_mobs >> i) & 1 ) == 1 ){
            CANPAGE = i << 4;

            /*
             * Зчитати дані повідомлення.
             * Оскільки біт CANPAGE.AINC скинуто (тобто встановлено
             * автоінкремент номеру байта в буфері MOb) послідовно
             * зчитати байт із CANMSG length разів.
             */
            for (uint8_t k = 0; k < 8; k++) {
                //Блок 9.4
                buf->data[buf_length*8 + k] = CANMSG;
            }
```

```
/*
             * Необхідно перезаписати CONMOB0-1, щоб скинути
             * прапорець BXOK
             */
            CANCDMOB &= (~0xC0);

            /* Інкрементуємо кількість оброблених Mob'ів */
            buf_length += 1;
        }
    }

    /* Відновити поточний MOb. Блок  9.7*/
    CANPAGE = save_canpage;
}

/*
 * Функція налаштування MOb0 на автоматичну відповідь.Блок 2
 */
void set_mob0_rtr_tx(void)
{
    uint8_t mob = 0;
    /* Зберегти поточний MOb. Блок 2.1 */
    uint8_t save_canpage = CANPAGE;

    /* Встановити MOb0 у якості поточного. */
    CANPAGE = mob << 4;
    /* Встановити ідентифікатор та біт RTR. Блоки 2.3-2.11 */
    CANIDT4 = (uint8_t) ((MOB0_ID << 3) | (1 << RTRTAG));
    CANIDT3 = (uint8_t) (MOB0_ID >> 5);
    CANIDT2 = (uint8_t) (MOB0_ID >> 13);
    CANIDT1 = (uint8_t) (MOB0_ID >> 21);
    /* Встановити маску для перевірки IDE, біта RTR та ідентифікатора. */
    CANIDM4 = (uint8_t) ((MOB0_MASK << 3) | (1 << RTRMSK) | (1 << IDEMSK));
    CANIDM3 = (uint8_t) (MOB0_MASK >> 5);
    CANIDM2 = (uint8_t) (MOB0_MASK >> 13);
    CANIDM1 = (uint8_t) (MOB0_MASK >> 21);

    /*
     * Записати дані повідомлення у буфер.
     * Оскільки біт CANPAGE.AINC скинуто
     * (тобто встановлено автоінкремент
     * номеру байта в буфері MOb),
     * послідовно записати у регістр CANMSG
     * до 8-ми байтів даних. Блок 2.13-2.15
     */
    for (uint8_t i = 0; i < MOB0_DLC; i++) {
        CANMSG = mob0_data[i];
    }
    /* Дозволити переривання для MOb0. Блок 2.18*/
    CANIE2 |= (1 << IEMOB0);
    /* Встановити режим прийому, автовідповіді, IDE та довжину повідомлення.
Блоки 2.12, 2.16, 2.19*/
    CANCDMOB = (1 << CONMOB1) | (1 << RPLV) | (1 << IDE) | (MOB0_DLC << DLC0);
    /* Відновити поточний MOb. */
    CANPAGE = save_canpage;
}
```

```
/*
 * Функція налаштування MOb1 на прийом.
 */
void set_mob1_rx(void)
{
    uint8_t mob = 1;
    /* Зберегти поточний MOb. Блок 4.1*/
    uint8_t save_canpage = CANPAGE;

    /* Встановити MOb1 у якості поточного. */
    CANPAGE = mob << 4;
    /* Встановити ідентифікатор та біт RTR. Блок 4.7 */
    CANIDT4 = (uint8_t) (MOB1_ID << 3);
    CANIDT3 = (uint8_t) (MOB1_ID >> 5);
    CANIDT2 = (uint8_t) (MOB1_ID >> 13);
    CANIDT1 = (uint8_t) (MOB1_ID >> 21);
    /* Встановити маску для перевірки IDE, біта RTR та ідентифікатора. Блок
4.5*/
    CANIDM4 = (uint8_t) ((MOB1_MASK << 3) | (1 << IDEMSK));
    CANIDM3 = (uint8_t) (MOB1_MASK >> 5);
    CANIDM2 = (uint8_t) (MOB1_MASK >> 13);
    CANIDM1 = (uint8_t) (MOB1_MASK >> 21);
    /* Дозволити переривання для MOb1. Блок 4.14*/
    CANIE2 |= (1 << IEMOB1);
    /* Встановити режим прийому та IDE. Блок 4.12 */
    CANCDMOB = (1 << CONMOB1) | (1 << IDE) | (MOB1_DLC << DLC0);
    /* Відновити поточний MOb. */
    CANPAGE = save_canpage;
}

/*
 * Функція налаштування MOb2 на прийом.
 */
void set_mob2_rx(void)
{
    uint8_t mob = 2;

    /* Зберегти поточний MOb. */
    uint8_t save_canpage = CANPAGE;

    /* Встановити MOb2 у якості поточного. */
    CANPAGE = mob << 4;
    /*
     * Встановити маску для перевірки IDE та скинути маску для
     * перевірки ідентифікатора. Регістри CANIDTn не встановлюємо,
     * оскільки ми прийматимемо будь-які повідомлення.
     * Перевірка IDE необхідна, оскільки ми працюємо тільки
     * з повідомленнями, що мають 29-бітний ідентифікатор.
     */
    CANIDM4 = (1 << IDEMSK);
    CANIDM3 = 0x00;
    CANIDM2 = 0x00;
    CANIDM1 = 0x00;
    /* Дозволити переривання для MOb2. */
    CANIE2 |= (1 << IEMOB2);
    /*
     * Встановити режим прийому та IDE. Довжина даних нас не цікавить,
     * адже ми прийматимемо будь-яке повідомлення.
     */
    CANCDMOB = (1 << CONMOB1) | (1 << IDE);
    /* Відновити поточний MOb. */
```

```
  CANPAGE = save_canpage;
}

/*
 * Функція надсилання повідомлення з MOb3.
 *    message - вказівник на структуру повідомлення, що містить
 *              ідентифікатор, власне дані та їх довжину.
 */
void set_mob3_tx(CAN_message *message)
{
    uint8_t mob = 3;
    /* Зберегти поточний MOb. */
    uint8_t save_canpage = CANPAGE;

    /* Встановити MOb3 у якості поточного. Блок 2.1 */
    CANPAGE = mob << 4;
    /* Встановити ідентифікатор повідомлення. Блоки 2.5 та 2.7 */
    CANIDT4 = (uint8_t) (message->id << 3);
    CANIDT3 = (uint8_t) (message->id >> 5);
    CANIDT2 = (uint8_t) (message->id >> 13);
    CANIDT1 = (uint8_t) (message->id >> 21);

    /*
     * Заповнити дані повідомлення.
     * Оскільки біт CANPAGE.AINC скинуто (тобто встановлено
     * автоінкремент номеру байта в буфері MOb) послідовно
     * записати message->length байт у CANMSG.
     * Блоки 2.14 - 2.16
     */
    for (uint8_t i = 0; i < message->length; i++) {
        CANMSG = message->data[i];
    }

    /* Наступні кроки алгоритму
     *  - Скинути прапорці переривань для MOb, Блок 2.17
     *  - Дозволити переривання для MOb, Блок 2.18
     * не виконуються, так як обробка завершення передачі
     * відбувається в тілі функції.
     */

    /* Встановити режим передачі та IDE. Блоки 2.3, 2.13 та 2.19 */
    CANCDMOB = (1 << CONMOB0) | (1 << IDE) | (message->length << DLC0);
    /* Дочекатися закінчення надсилання повідомлення. */
    while (!(CANSTMOB & (1 << TXOK)))
        ;

    /* Відновити поточний MOb. */
    CANPAGE = save_canpage;
}

/*
 * Функція надсилання повідомлення з обраного MOb.
 *    message - вказівник на структуру повідомлення, що містить
 *              ідентифікатор, власне дані та їх довжину.
 *    send_mob - порядкомий номер MOb, що буде підготовлений для
 *               передачі.
 */
void set_mob_tx(CAN_message *message, uint8_t send_mob)
{
    uint8_t mob = send_mob;
    /* Зберегти поточний MOb. */
```

```
    uint8_t save_canpage = CANPAGE;

    /* Встановити бажаний MOb у якості поточного. */
    CANPAGE = mob << 4;
    /* Встановити ідентифікатор повідомлення. */
    CANIDT4 = (uint8_t) (message->id << 3);
    CANIDT3 = (uint8_t) (message->id >> 5);
    CANIDT2 = (uint8_t) (message->id >> 13);
    CANIDT1 = (uint8_t) (message->id >> 21);

    /*
     * Заповнити дані повідомлення.
     * Оскільки біт CANPAGE.AINC скинуто (тобто встановлено
     * автоінкремент номеру байта в буфері MOb) послідовно
     * записати message->length байт у CANMSG.
     */
    for (uint8_t i = 0; i < message->length; i++) {
        CANMSG = message->data[i];
    }

    if (send_mob < 8)
    {
        CANIE2 |= 1 << send_mob;
    }
    else
    {
        CANIE1|= 1 << send_mob;
    }


    /* Зкинути прапорці переривань. */
    CANSTMOB = 0;

    /* Встановити режим передачі та IDE. */
    CANCDMOB = (1 << CONMOB0) | (1 << IDE) | (message->length << DLC0);


    /* Відновити поточний MOb. */
    CANPAGE = save_canpage;
}

/*
 * Функція передачі мультикадру. Три вільних MOb налаштовуються на передачу
 * та заповнюються масивом даних multiframe_data.
 */
void send_multiframe()
{
    uint8_t mobs_to_send = MULTIFRAME_LEN;

    /* Зберегти поточний MOb. */
    uint8_t save_canpage = CANPAGE;

    /* Останній MOb, що буде відправлено у складі мультикадру. */
    uint8_t last_mob = 0;

    /*
     * Перевірка у циклі стану всіх MOb'ів та підготовка до передачі
     * повідомлення перших трьох вільних.
     */
```

```
for(uint8_t scan_mob = 0; scan_mob < 15; scan_mob++)
    {
        /* MOb, що перевіряється */
        CANPAGE = scan_mob << 4;

        /* Перевірка значень бітів CONMOB0-1, 00 - вільний*/
        uint8_t mob_state = CANCDMOB & ((1 << CONMOB0) | (1 << CONMOB1));
        if (mob_state == 0)
        {
            /* Підготовка повідомлення до передачі. */
            CAN_message tx_message;


            tx_message.length = 8;
            tx_message.id = MULTIFRAME_ID;
            /* Заповнення повідомлення даними з масиву multiframe_data */
            for (uint8_t i = 0; i < 8; i++)
            {
                uint8_t byte_to_send = (MULTIFRAME_LEN - mobs_to_send)*8 + i;
                tx_message.data[i] = multiframe_data[byte_to_send];
            }

            /* Налаштування MOb на передачу */
            set_mob_tx(&tx_message, scan_mob);

            last_mob = scan_mob;
            mobs_to_send -= 1;
        }

        /* Завершити цикл, якщо необхідну кількість MOb'ів підготовлено */
        if (mobs_to_send == 0)
        {
            break;
        }
    }

    CANPAGE = last_mob << 4;

    /* Дочекатися закінчення надсилання повідомлення. */
    while (!(CANSTMOB & (1 << TXOK)))
        ;

    /* Відновити поточний MOb. */
    CANPAGE = save_canpage;
}

/*
 * Функція підготовки (скидання) поточного MOb після прийому та автовідповіді
 * до прийому нового повідомлення та автовідповіді на нього.
 */
void reset_mobx_rtr_tx(void)
{
    /* Встановити режим прийому та автовідповіді. */
    CANCDMOB = (1 << CONMOB1) | (1 << RPLV) | (1 << IDE);
    /* Встановити біт RTR. */
    CANIDT4 |= (1 << RTRTAG);
}

/*
 * Функція зчитування даних під час прийому в MOb.
 * На момент виклику функції необхідний MOb повинен
```

```
 * бути встановленим у якості поточного.
 */
void get_mobx_rx(CAN_message *message)
{
    /* Зчитати довжину повідомлення. */
    message->length = (CANCDMOB >> DLC0) & 0x0f;
    /* Зчитати ідентифікатор повідомлення. */
    message->id = ((uint32_t) CANIDT1) << 24;
    message->id |= ((uint32_t) CANIDT2) << 16;
    message->id |= ((uint32_t) CANIDT3) << 8;
    message->id |= CANIDT4;
    message->id >>= 3;

    /*
     * Зчитати дані повідомлення.
     * Оскільки біт CANPAGE.AINC скинуто (тобто встановлено
     * автоінкремент номеру байта в буфері MOb) послідовно
     * зчитати байт із CANMSG length разів.
     */
    for (uint8_t i = 0; i < message->length; i++) {
        message->data[i] = CANMSG;
    }
    /*
     * Перевірити маску повідомлення, встановлену у MOb.
     * Якщо маска рівна 0x0001, то відновлювати
     * режим прийому не потрібно.
     */
    uint32_t mask;
    mask  = (uint32_t) (CANIDM1 << 24);
    mask |= (uint32_t) (CANIDM2 << 16);
    mask |= (uint32_t) (CANIDM3 << 8);
    mask |= (uint32_t) CANIDM4;

    if (mask != 1)
    {
        /* Встановити режим прийому та IDE */
        CANCDMOB = (1 << CONMOB1) | (1 << IDE);
    }
}

/*
 * Функція ініціалізації CAN-модуля та скидання всіх MOb.
 */
void init_can(void) //Блок 1
{
    //Блок 1.1
    CANGCON = (1 << SWRES);

    /* Швидкість інтерфейсу та параметри імпульсів. Блок 1.2 */
    CANBT1 = 0x01;
    CANBT2 = 0x02;
    CANBT3 = 0x03;
    //Блок 1.5
    CANTCON = 0x00;
    /*
     * Очистити регістри статусу, керування та DLC для усіх MOb. Це
     * робити обов'язково, оскільки ці біти значень за замовчанням не мають.
Блок 1.6
     */
    for (uint8_t mob = 0; mob < 15; mob++) {
        CANPAGE = mob << 4;
```

```
        CANSTMOB = 0;
        CANCDMOB = 0;
    }
    /* Встановити активний режим роботи CAN-модуля. Блок 1.7*/
    CANGCON |= (1 << ENASTB);
    /*
     * Дозволити переривання завершення прийому та передачі. Помилки
     * оброблювати ми не будемо.
     */
    //Блок 1.4
    CANGIE = (1 << ENIT) | (1 << ENTX) | (1 << ENRX) | (1 << ENBX);
    /*
     * Заборонити переривання від усіх MOb, очистивши регістри
     * дозволу переривань. Блок 1.3
     */
    CANIE2 = 0x00;
    CANIE1 = 0x00;
    /* Дочекатися активності CAN-контролера. Блоки 1.8-1.9*/
    while (!(CANGSTA & (1 << ENFG)))
        ;
}

/*
 * Обробник переривання від CAN-модуля.
 */
SIGNAL(SIG_CAN_INTERRUPT1)
{

 /*
     * Якщо мультикадр був прийнятий повністю,
     * встановлюється прапорець BXOKBXOK 4 186
     */
    if (CANGIT & (1 << BXOK)) {
        get_frame_buffer(&buffer);

        /*
         * Скидаємо прапорець BXOK записуючи у нього "1".
         */
        CANGIT |= 1 << BXOK;
        return;
    }

    uint8_t hp_mob = CANHPMOB;

    /*
     * Якщо номер MOb, що ініціював переривання не визначено,
     * вийти із підпрограми. Див. опис регістра CANHPMOB.
     */
    if ((hp_mob & 0xF0) == 0xF0) {
        return;
    }

    /* Зберегти поточний MOb. */
    uint8_t save_canpage = CANPAGE;

    /* Встановити MOb-ініціатор переривання у якості поточного. */
    CANPAGE = hp_mob;
    /*
     * Якщо переривання по надсиланню повідомлення, значить було
     * прийнято повідомлення та згенеровано автовідповідь із MOb0.
     * Тоді викликати підпрограму підготовки (скидання) MOb0 для
```

```
     * автовідповіді. Зверніть увагу, що для роботи MOb3, який
     * налаштований на передачу, ми переривання не використовуємо.
     */

    if (CANSTMOB & (1 << TXOK)) {
        /*
         * Очистити біт успішної передачі. Це треба робити
         * якомога раніше, щоб випадково не пропустити перивання,
         * що може виникнути під час виконання цього обробника.
         */
        CANCDMOB = 0x00;
        CANSTMOB &= ~(1 << TXOK);
    }

    /*
     * Якщо переривання по прийому повідомлення, то викликати
     * підпрограму зчитування даних під час прийому з MOb-а.
     */
    if (CANSTMOB & (1 << RXOK) && (CANCDMOB >> CONMOB0 != 3)) {
        /*
         * Очистити біт успішного прийому. Це треба робити
         * якомога раніше, щоб випадково не пропустити перивання,
         * що може виникнути під час виконання цього обробника.
         */
        CANSTMOB &= ~(1 << RXOK);
        get_mobx_rx(&rx_message);
    }


  /* Відновити поточний MOb. */
    CANPAGE = save_canpage;
}

/*
 * Головна програма.
 */
int main(void)
{
    /* Ініціалізація CAN. */
    init_can(); //Блок 1

    /* Підготовка MOb до прийому. */
    set_mob0_rtr_tx();
    set_mob1_rx();
    set_mob2_rx();

    /* Повідомлення для передачі. */
    CAN_message tx_message;
    CAN_message tx_message1;
    CAN_message tx_message2;

    /* Сворення нових повідомленнь для mobів */
    tx_message.id = 0x055;
    tx_message.length = 8;
    tx_message.data[0] = 0xaa;
    tx_message.data[1] = 0xbb;
    tx_message.data[2] = 0xcc;
    tx_message.data[3] = 0xcc;
    tx_message.data[4] = 0xcc;
    tx_message.data[5] = 0xcc;
    tx_message.data[6] = 0xcc;
```

```
  tx_message.data[7] = 0xcc;

  tx_message1.id = 0x044;
  tx_message1.length = 8;
  tx_message1.data[0] = 0x11;
  tx_message1.data[1] = 0x22;
  tx_message1.data[2] = 0xcc;
  tx_message1.data[3] = 0xcc;
  tx_message1.data[4] = 0xcc;
  tx_message1.data[5] = 0xcc;
  tx_message1.data[6] = 0xcc;
  tx_message1.data[7] = 0xcc;

  tx_message2.id = 0x066;
  tx_message2.length = 8;
  tx_message2.data[0] = 0x31;
  tx_message2.data[2] = 0xca;
  tx_message2.data[3] = 0xca;
  tx_message2.data[4] = 0xac;
  tx_message2.data[5] = 0xcc;
  tx_message2.data[6] = 0xca;
  tx_message2.data[7] = 0xca;

  /* Надіслати повідомлення з MOb3. */
  set_mob_tx(&tx_message, 3);

  /* Надіслати повідомлення з MOb7. */
  set_mob_tx(&tx_message1, 7);


/* Надіслати повідомлення з MOb4. */
  set_mob_tx(&tx_message2, 4);

  /* Дочекатися завершення передачі MOb7 */
  /* Щоб використовувати його далі */
  uint8_t save_canpage = CANPAGE;
  CANPAGE = 7 << 4;
  while (!(CANSTMOB & (1 << TXOK)));
  CANPAGE = save_canpage;

  /*
   * Підготовка буфера для прийому мультикадру.
   * Зчитування прийнятого мультикадру відбувається в обробнику
   * переривань від CAN-модуля.
   */
  /* Обрання Mob'ів для прийому мультикадра */
  buffer.buffered_mobs = (uint16_t) (1 << 6) | (1 << 7)| (1 << 8); //Блок 2
  /* Встановлення ідентифікатору повідомлень мультикадра*/
  buffer.id = MULTIFRAME_ID;
  /* Ініціалізація Mob'ів для прийому мультикадра */
  set_frame_buffer(&buffer);

  /* Дозволити переривання. */
  sei();

  /*
   * Передача мультикадру - три послідовних повідомлення
   * з ідентифікатором 0x0111, що містять 24 байти даних.
   * Передача мультикадру буде вважатись завершеною, коли
   * буде передано повідомлення з останнього MOb'а.
   */
```

```
    send_multiframe();

     /* Нескінченний цикл. */
    while ((CANGSTA & 4) == 4){
        uint8_t a = 3;
    }
}
```