Микроконтроллеры и CAN-интерфейс

Олег Вальпа (Челябинская обл.)

В статье приведено описание интерфейса CAN, схемы его подключения и библиотечных функций среды разработки mikroC для поддержки данного интерфейса микроконтроллером с примером программы на языке Си для организации связи между устройствами.

Интерфейс CAN (Controller Area Network, локальная сеть контроллеров) был разработан компанией Robert Bosch GmbH в 1980-х годах. В настоящее время он широко распространён в автомобильной промышленности, автоматизации производства, энергетике, технологиях «умного дома» и других областях.

Описание интерфейса CAN

Интерфейс CAN предназначен для организации последовательных, высоконадёжных и недорогих каналов связи в распределённых системах управления. Он позволяет организовывать как мультиплексные каналы, так и высокоскоростные сети. Данный интерфейс имеет протокол, поддерживающий возможность нахождения на магистрали нескольких ведущих устройств и обеспечивает передачу данных в реальном масштабе времени.

Интерфейс обладает высокой помехоустойчивостью благодаря коррекции ошибок. Передача данных осуществляется кадрами, которые принимаются всеми устройствами сети. Кадр состоит из идентификатора длиной 11 бит для стандартного формата или 29 бит для расширенного формата и блока данных, содержащего от 0 до 8 байт. Идентификатор описывает содержимое пакета данных и служит для определения приоритета при попытке одновременной передачи несколькими устройствами.

Скорость передачи данных выбирается исходя из расстояния, числа абонентов в сети и ёмкости линии связи. Она задаётся программно и может составлять от десятка Кбод до единиц Мбод.

Стандарт интерфейса CAN определяет передачу данных независимо от физического уровня, т.е. канал связи может быть каким угодно, например, радиоканалом или оптоволокном. Однако на практике под CAN-интерфейсом обычно подразумевается сеть с физическим каналом связи в виде дифференциальной пары проводов, определённым в стандарте ISO 11898. На рисунке 1 приведена схема подключения устройств к интерфейсу CAN. Стандартное назначение выводов для широко распространённого 9-контактного разъёма интерфейса CAN приведено на рисунке 2.

Интерфейс CAN относится к типу сетей CR (Collision Resolution, разрешение коллизий), в отличие от сетей типа CD (Collision Detect, обнаружение коллизий), например Ethernet. Тип сетей CR обеспечивает приоритетный доступ к передаче сообщения, что необходимо для промышленных устройств.

Приоритетный доступ к передаче сообщений в стандарте ISO-11898 реализуют так называемые рецессивные и доминантные биты. В зависимости от типа шины, объединяющей устройства CAN, эти биты могут принимать значение лог. 0 либо лог. 1. В таблице 1 приведены два примера организации шин, поясняющие образование управляющих битов.

Спецификация интерфейса CAN избегает описания двоичных значений сигналов, как лог. 0 либо лог. 1, с целью абстрагирования от среды передачи. Именно поэтому вместо них применяются термины «рецессивный» и «доминантный» биты. При этом подразумевается, что при передаче одним устройством сети рецессивного бита, а другим доминантного, принят будет доминантный бит. Например, при реализации физического уровня с помощью радиоканала отсутствие сигнала означает рецессивный бит, а наличие сигнала – доминантный.

Стандарт сети требует от физического уровня передачи, чтобы доминантный бит мог подавить рецессивный, но не наоборот. Например, в оптическом волокне доминантному биту должен соответствовать свет, а рецессивному – его отсутствие. В электрическом проводе рецессивное состояние соответствует высокому потенциалу на линии связи, а доминантное – низкому потенциалу, т.е. когда устройство подключает линию связи к нулевому потенциалу. Если линия находится в рецессивном состоянии, перевести её в доминантное состояние может любое устройство сети, например, включив свет в оптоволокне.

Кадры и их форматы

Кроме рецессивного и доминантного битов, стандарт интерфейса CAN оперирует таким понятием, как кадр, который используется в протоколе обмена информацией между устройствами. Каждый кадр представляет собой единый набор нескольких служебных битов и байтов. Существуют следующие виды кадров:

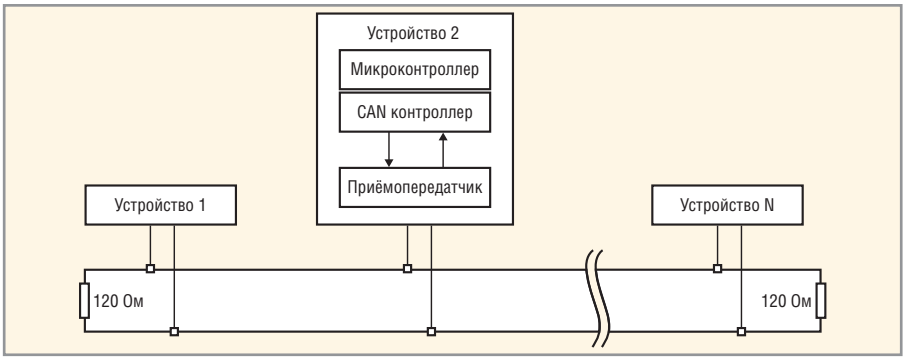

Рис. 1. Схема подключения устройств к интерфейсу CAN

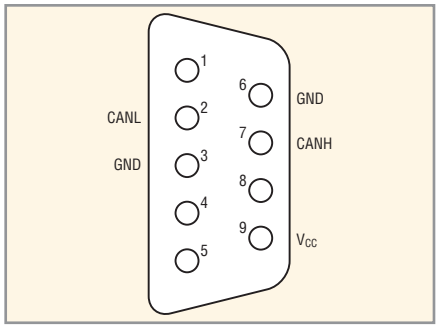

Рис. 2. Стандартное назначение выводов для 9-контактного разъёма CAN-интерфейса

- кадр данных (data frame) служит для передачи данных;
- кадр запроса передачи (remote frame) служит для запроса на передачу кадра данных с тем же идентификатором;
- кадр перегрузки (overload frame) обеспечивает промежуток между кадрами данных или кадрами запросов;
- кадр ошибки (error frame) передаётся узлом, обнаружившим ошибку.

В таблице 2 представлен базовый формат кадра данных, а в таблице 3 – расширенный формат кадра данных. Графическое представление данных кадров, с пояснениями и порядком следования бит, приведено на рисунках 3 и 4 соответственно. Межкадровый интервал служит для разделения кадров между собой и состоит как минимум из трёх рецессивных бит. Он необходим передающему устройству для подготовки очередного кадра.

Таблица 1. Примеры организации шин

Тип шины	Доминантный бит	Рецессивный бит	Результирующий бит
Монтажное «ИЛИ»	1	0	1⊕0 = 1
Монтажное «И»	0	1	0&1 = 0

Таблица 2. Базовый формат кадра данных

Поле	Длина, бит	Описание
Начало кадра (SOF-Start Off Frame)	1	Сигнализирует начало передачи кадра. Доминантный бит
Идентификатор (ID-Identifier)	11	Уникальный идентификатор
Запрос на передачу (RTR, Remote Transmission Request)	1	Должен быть доминантным
Бит расширения идентификатора (IDE- Identifier Extension)	1	Должен быть доминантным
Нулевой резервный бит (RB0-Reserved Bit Zero)	1	Резервный
Длина данных (DLC- Data Length Code)	4	Длина поля данных в байтах (0–8)
Поле данных, байт	0–8	Передаваемые данные (длина в поле DLC)
Контрольная сумма (CRC-Cyclic Redundancy Code)	15	Контрольная сумма всего кадра
Разграничитель контрольной суммы	1	Должен быть рецессивным
Бит подтверждения (ACK)	1	Передатчик передаёт рецессивный бит, а приёмник вставляет доминантный бит
Разграничитель подтверждения	1	Должен быть рецессивным
Конец кадра (EOF)	7	Должен быть рецессивным

Таблица 3. Расширенный формат кадра данных

Поле	Длина, бит	Описание
Начало кадра (SOF-Start Off Frame)	1	Сигнализирует начало передачи кадра. Доминантный бит
Идентификатор A	11	Первая часть идентификатора
Подмена запроса на передачу (SRR)	1	Должен быть рецессивным
Бит расширения идентификатора (IDE- Identifier Extension)	1	Должен быть рецессивным
Идентификатор B	18	Вторая часть идентификатора
Запрос на передачу (RTR)	1	Должен быть доминантным
Резервные биты (RB1 и RB0)	2	Резерв
Длина данных (DLC)	4	Длина поля данных в байтах (0–8)
Поле данных, байт	0–8	Передаваемые данные (длина в поле DLC)
Контрольная сумма (CRC)	15	Контрольная сумма всего кадра
Разграничитель контрольной суммы	1	Должен быть рецессивным
Промежуток подтверждения (ACK)	1	Передатчик передаёт рецессивный бит, а приёмник вставляет доминантный бит
Разграничитель подтверждения	1	Должен быть рецессивным
Конец кадра (EOF)	7	Должен быть рецессивным

Арбитраж доступа к шине

При наличии в сети нескольких устройств, которые могут формировать запросы, в ней могут возникать конфликтные ситуации, когда эти запросы выставляются одновременно. С целью устранения подобных коллизий в интерфейсе CAN предусмотрен механизм арбитража доступа к шине. Когда шина свободна, любое устройство может начинать передачу в любой момент. В случае одновременной передачи кадров двумя и более устройствами производится арбитраж: передавая биты идентификатора, устройство одновременно проверяет состояние шины. Если при передаче очередного бита на шине присутствует противоположный бит, считается, что другое устройство передаёт сообщение с более высоким приоритетом, и передача откладывается до освобождения шины. Таким образом, в отличие, например, от сети Ethernet, в сети CAN не происходит непроизводительной потери пропускной способности канала при коллизиях. Однако при этом не исключена вероятность того, что сообщения с низким приоритетом никогда не будут переданы.

Защита информации от ошибок

Интерфейс CAN имеет несколько механизмов контроля и предотвращения ошибок. Благодаря арбитражу доступа к шине, производится контроль передачи информации, поскольку при передаче битовые уровни в сети сравниваются с передаваемыми битами.

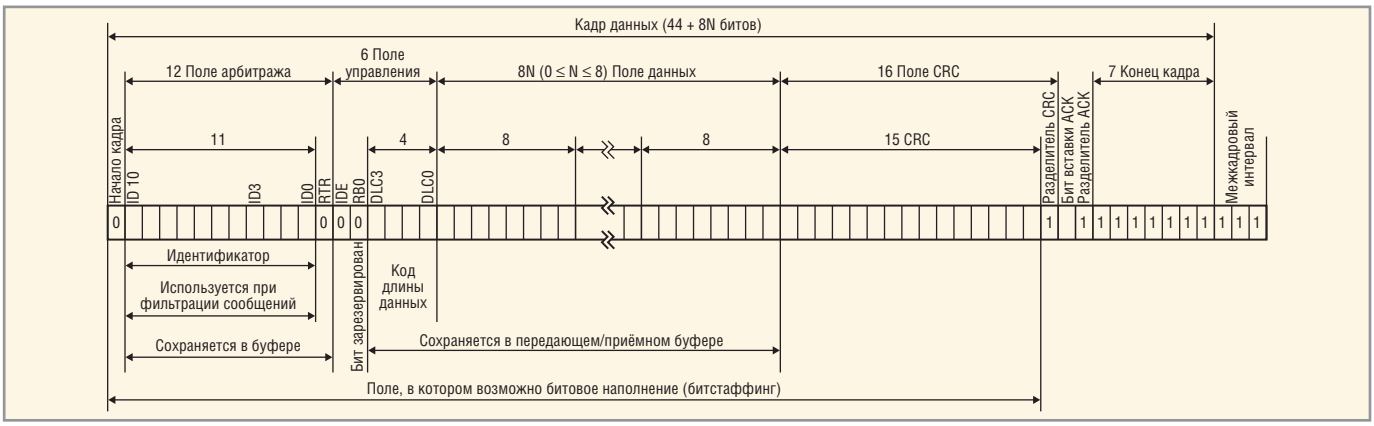

Рис. 3. Стандартный кадр данных

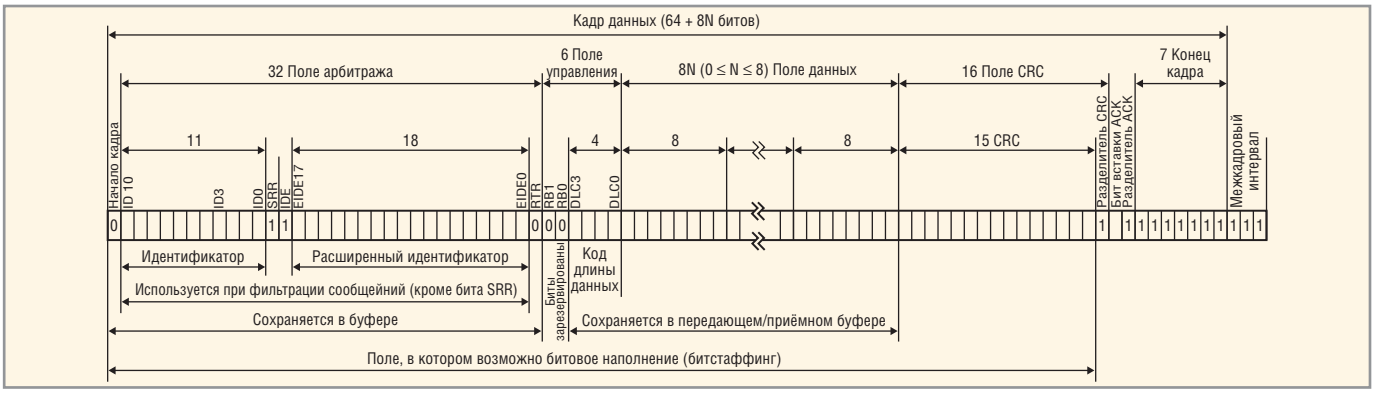

Рис. 4. Расширенный кадр данных

Кроме того, интерфейс CAN использует для защиты информации от ошибок контрольную сумму. Каждый раз при передаче информации, передающее устройство вычисляет контрольную сумму пакета данных и добавляет её в передаваемый кадр. Устройство, принимающее информацию, вычисляет контрольную сумму принимаемого кадра и сравнивает её с контрольной суммой принятого кадра. В случае совпадения контрольных сумм устройство передаёт доминантный бит в промежутке подтверждения.

После передачи пяти одинаковых битов подряд передаётся дополнительный бит противоположного значения – так называемый бит-вставка (bit stuffing), который позволяет защититься от сбоя синхронизации устройств при передаче данных, состоящих из одного и того же двоичного кода.

Упомянутые выше механизмы позволяют обеспечить передачу информации с вероятностью пропуска ошибки $4{,}7 \times 10^{-11}$.

Таблица 4. Соотношение скорости передачи и длины шины

Скорость передачи, Кбит/с	Длина шины, м
1000	40
500	100
125	500
10	5000

Технические характеристики шины

Все устройства в сети CAN должны работать с одинаковой скоростью. Стандарт CAN не определяет скорость работы, но большинство как самостоятельных, так и встроенных в МК контроллеров CAN позволяют плавно изменять скорость в диапазоне от 20 Кбит/с до 1 Мбит/с. Существуют также решения, выходящие далеко за рамки данного диапазона.

Приведённые выше методы защиты информации от ошибок требуют, чтобы изменение бита при передаче успело распространиться по всей сети к моменту контроля его состояния передатчиком. Это приводит к обратной зависимости длины шины от скорости передачи: чем больше скорость передачи информации, тем меньше должна быть длина шины. Соотношения скорости передачи и длины шины для сетей стандарта ISO 11898 приведены в таблице 4.

На скорость передачи информации и длину шины также оказывает влияние использование оптических пар для защиты устройств от высоковольтных помех. Снижают скорость передачи информации и сильно разветвлённые сети в виде паутины, из-за множества отражений сигнала и большой электрической ёмкости шины.

Программное обеспечение

Среда разработки mikroC [1] предоставляет набор готовых функций для создания программ, обеспечивающих совместную работу устройств по CAN-интерфейсу на базе МК различных семейств. Рассмотрим их на примере функций для микроконтроллеров семейства PIC компании microchip [2].

Библиотека для работы с интерфейсом CAN

Библиотека среды разработки mikroC для работы с CAN-интерфейсом включает в себя следующие функции:

- CANSetOperationMode,
- CANGetOperationMode,
- CANInitialize,
- CANSetBaudRate,
- CANSetMask,
- CANSetFilter,
- CANRead,
- CANWrite.

Подробное описание этих функций с требованиями и примером вызова приведено в таблицах 5 – 12. Функции используют множество специальных констант, так называемых констант CAN. Описание этих констант приводится ниже. Данные функции CAN поддерживают не все

Таблица 5. Описание функции CANSetOperationMode

Прототип	void CANSetOperationMode(unsigned short mode, unsigned short wait_flag);
Возвращаемое значение	Нет
Описание	Устанавливает требуемый режим работы CAN, т.е. копирует аргумент mode в CANSTAT. Аргумент mode должен быть одной из констант CAN_OP_MODE (см. константы CAN) Аргумент wait_flag должен имеет значение 0 или 0xFF: • задание значения 0xFF означает блокирующий вызов функции, т.е. функция не завершает работу до тех пор, пока требуемый режим не будет установлен; • задание значения 0 означает деблокирующий вызов функции, т.е. функция завершается без проверки, переключился ли модуль в необходимый режим или нет. Вызывающая программа перед тем, как выполнить определённую операцию, должна использовать вызов CANGetOperationMode, чтобы убедиться, что модуль работает в необходимом режиме
Требования	Микроконтроллер должен быть подключен к CAN-трансиверу (например, MCP2551 или подобному), который соединён с шиной CAN
Пример	CANSetOperationMode(CAN_MODE_CONFIG, 0xFF);

Таблица 6. Описание функции CANGetOperationMode

Прототип	unsigned short CANGetOperationMode(void);
Возвращаемое значение	Текущий режим работы
Описание	Функция возвращает текущий режим работы модуля CAN
Требования	Микроконтроллер должен быть подключен к CAN-трансиверу (MCP2551 или подобному), который соединён с шиной CAN
Пример	if (CANGetOperationMode() == CAN_MODE_NORMAL) { ... };

Таблица 7. Описание функции CANInitialize

Прототип	void CANInitialize(unsigned short SJW, unsigned short BRP, unsigned short PHSEG1, unsigned short PHSEG2, unsigned short PROPSEG, unsigned short CAN_CONFIG_FLAGS);
Возвращаемое значение	Нет

Описание	Инициализирует интерфейс CAN. Все незавершённые передачи отменяются. Все регистры масок устанавливаются в 0 для разрешения всех сообщений. При выполнении функции устанавливается режим конфигурирования. По окончании выполнения данной функции устанавливается нормальный режим
	Регистры фильтров устанавливаются в соответствии со значениями флагов:
	if (CAN_CONFIG_FLAGS & CAN_CONFIG_VALID_XTD_MSG != 0)
	// Установить все фильтры в XTD_MSG
	else if (config & CONFIG_VALID_STD_MSG != 0)
	// Установить все фильтры в STD_MSG
	else
	// Установить половину фильтров в STD, а остальные в XTD_MSG
	Аргументы: SJW, BRP, PHSEG1, PHSEG2 и PROPSEG, как определено в описании PIC-микроконтроллеров семейства P18XXX8
	CAN_CONFIG_FLAGS, формируются из предопределённых констант (см. константы CAN)

Требования	Микроконтроллер должен быть подключен к CAN-трансиверу (MCP2551 или подобному), который соединён с шиной CAN

Пример	init = CAN_CONFIG_SAMPLE_THRICE &
	CAN_CONFIG_PHSEG2_PRG_ON &
	CAN_CONFIG_STD_MSG &
	CAN_CONFIG_DBL_BUFFER_ON &
	CAN_CONFIG_VALID_XTD_MSG &
	CAN_CONFIG_LINE_FILTER_OFF;
	// инициализация CAN7
	CANInitialize(1, 1, 3, 3, 1, init);

Таблица 8. Описание функции CANSetBaudRate

Прототип	void CANSetBaudRate(unsigned short SJW, unsigned short BRP, unsigned short PHSEG1, unsigned short PHSEG2, unsigned short PROPSEG, unsigned short CAN_CONFIG_FLAGS);
Возвращаемое значение	Нет
Описание	Устанавливает скорость обмена в сети CAN. Ввиду сложности протокола CAN, нельзя принудительно установить значение скорости. Вместо этого следует использовать данную функцию, когда CAN находится в режиме конфигурирования. Подробнее см. описание Аргументы: SJW, BRP, PHSEG1, PHSEG2 и PROPSEG, как определено в описании PIC-микроконтроллеров семейства P18XXX8 CAN_CONFIG_FLAGS, формируются из предопределённых констант (см. константы CAN)
Требования	Интерфейс CAN должен быть в режиме конфигурирования, в противном случае функция будет игнорироваться Микроконтроллер должен быть подключен к CAN-трансиверу (MCP2551 или подобному), который соединён с шиной CAN
Пример	init = CAN_CONFIG_SAMPLE_THRICE & CAN_CONFIG_PHSEG2_PRG_ON & CAN_CONFIG_STD_MSG & CAN_CONFIG_DBL_BUFFER_ON & CAN_CONFIG_VALID_XTD_MSG & CAN_CONFIG_LINE_FILTER_OFF; ... CANSetBaudRate(1, 1, 3, 3, 1, init);

Таблица 9. Описание функции CANSetMask

Прототип	void CANSetMask(unsigned short CAN_MASK, long value, unsigned short CAN_CONFIG_FLAGS);
Возвращаемое значение	Нет
Описание	Функция устанавливает маску расширенной фильтрации сообщений. Заданное значение value по битам соответствует буферным регистрам маски Аргументы: • CAN_MASK – одна из предопределённых констант (см. константы CAN). value – значение маски регистра; • CAN_CONFIG_FLAGS задаёт тип сообщения для фильтрации CAN_CONFIG_XTD_MSG или CAN_CONFIG_STD_MSG
Требования	Интерфейс CAN должен быть в режиме конфигурирования, в противном случае функция будет игнорироваться Микроконтроллер должен быть подключен к CAN-трансиверу (MCP2551 или подобному), который соединён с шиной CAN
Пример	/* Установить все биты маски в 1, т.е. все фильтруемые биты имеют значение: */ CANSetMask(CAN_MASK_B1, -1, CAN_CONFIG_XTD_MSG); /* Замечание: –1 – это просто более удобный способ записи 0xFFFFFFFF. Дополнительный код заполнит всё единицами */

Таблица 10. Описание функции CANSetFilter

Прототип	void CANSetFilter(unsigned short CAN_FILTER, long value, unsigned short CAN_CONFIG_FLAGS);
Возвращаемое значение	Нет
Описание	Функция устанавливает фильтр сообщений. Заданное значение value по битам соответствует буферным регистрам маски Аргументы: • CAN_FILTER – одна из предопределённых констант (см. константы CAN) • value – значение регистра фильтрации • CAN_CONFIG_FLAGS задаёт тип сообщения для фильтрации CAN_CONFIG_XTD_MSG или CAN_CONFIG_STD_MSG
Требования	Интерфейс CAN должен быть в режиме конфигурирования, в противном случае функция будет игнорироваться Микроконтроллер должен быть подключен к CAN-трансиверу (MCP2551 или подобному), который соединён с шиной CAN
Пример	/* Установить идентификатор фильтра B1_F1 = 3: */ CANSetFilter(CAN_FILTER_B1_F1, 3, CAN_CONFIG_XTD_MSG);

Таблица 11. Описание функции CANRead

Прототип	unsigned short CANRead(long *id, unsigned short *data, unsigned short *datalen, unsigned short *CAN_RX_MSG_FLAGS);
Возвращаемое значение	Сообщение из буфера приёма или 0, если сообщение не найдено
Описание	Функция читает сообщение из буфера приёма. Если найден хотя бы один заполненный буфер, оно извлекается и возвращается. Если таковых нет, функция возвращает 0 Аргументы: • id – идентификатор сообщения; • data – массив байтов длиной до 8 байтов; • datalen – размер данных, от 1 до 8 CAN_RX_MSG_FLAGS – значение, сформированное из констант (см. константы CAN)
Требования	Интефейс CAN должен быть в режиме, в котором возможен приём Микроконтроллер должен быть подключен к CAN-трансиверу (MCP2551 или подобному), который соединён с шиной CAN

Пример	unsigned short rcv, rx, len, data[8];
	long id;
	// ...
	rx = 0;
	// ...
	rcv = CANRead(id, data, len, rx);

Таблица 12. Описание функции CANWrite

Прототип	unsigned short CANWrite(long id, unsigned short *data, unsigned short datalen, unsigned short CAN_TX_MSG_FLAGS);
Возвращаемое значение	Возвращает 0, если сообщение не может быть поставлено в очередь на передачу (буфер переполнен)
Описание	Если есть хотя бы один пустой буфер передачи, функция пересылает в него сообщение для передачи. Если буфер переполнен, функция возвращает 0 Аргументы: • id – идентификатор сообщения CAN. В зависимости от типа сообщения (стандартное или расширенное) могут быть использованы только 11 или 29 битов; • data – массив байтов длиной до 8 байтов; • datalen – размер данных от 1 до 8 CAN_TX_MSG_FLAGS – значение, сформированное из констант (см. константы CAN)
Требования	Интерфейс CAN должен быть в нормальном режиме Микроконтроллер должен быть подключен к CAN-трансиверу (MCP2551 или подобному), который соединён с шиной CAN
Пример	unsigned short tx, data; long id; // ... tx = CAN_TX_PRIORITY_0 & CAN_TX_XTD_FRAME; // ... CANWrite(id, data, 2, tx);

типы микроконтроллеров. К числу поддерживающих относятся, например, PIC-микроконтроллеры семейства P18XXX8.

Константы CAN

В библиотеке CAN существует ряд предопределённых констант. Чтобы эффективно использовать библиотеку, необходимо разбираться в этих константах.

Константы CAN_OP_MODE определяют режим работы CAN. Функция CANSetOperationMode предполагает использование их в качестве своих аргументов:

```
// Использовать для доступа к битам режима
#define CAN_MODE_BITS 0xE0
#define CAN_MODE_NORMAL 0
#define CAN_MODE_SLEEP 0x20
#define CAN_MODE_LOOP 0x40
#define CAN_MODE_LISTEN 0x60
#define CAN_MODE_CONFIG 0x80
```

Константы CAN_CONFIG_FLAGS определяют флаги, относящиеся к конфигурированию модуля CAN. Функции CANInitialize и CANSetBaudRate предполагают использование одной из них (или побитовой комбинации нескольких) в качестве аргумента:

```
#define CAN_CONFIG_DEFAULT 0xFF
// 11111111
#define CAN_CONFIG_PHSEG2_PRG_BIT
0x01
#define CAN_CONFIG_PHSEG2_PRG_ON
0xFF // XXXXXXX1
#define CAN_CONFIG_PHSEG2_PRG_OFF
0xFE // XXXXXXX0
#define
CAN_CONFIG_LINE_FILTER_BIT 0x02
#define CAN_CONFIG_LINE_FILTER_ON
0xFF // XXXXXX1X
#define
CAN_CONFIG_LINE_FILTER_OFF 0xFD
// XXXXXX0X
#define CAN_CONFIG_SAMPLE_BIT
0x04
#define CAN_CONFIG_SAMPLE_ONCE
0xFF // XXXXX1XX
#define CAN_CONFIG_SAMPLE_THRICE
0xFB // XXXXX0XX
#define CAN_CONFIG_MSG_TYPE_BIT
0x08
#define CAN_CONFIG_STD_MSG 0xFF
// XXXX1XXX
#define CAN_CONFIG_XTD_MSG 0xF7
// XXXX0XXX
#define CAN_CONFIG_DBL_BUFFER_BIT
0x10
#define CAN_CONFIG_DBL_BUFFER_ON
0xFF // XXX1XXXX
#define CAN_CONFIG_DBL_BUFFER_OFF
0xEF // XXX0XXXX
#define CAN_CONFIG_MSG_BITS 0x60
#define CAN_CONFIG_ALL_MSG 0xFF
// X11XXXXX
#define CAN_CONFIG_VALID_XTD_MSG
0xDF // X10XXXXX
#define CAN_CONFIG_VALID_STD_MSG
0xBF // X01XXXXX
#define CAN_CONFIG_ALL_VALID_MSG
0x9F // X00XXXXX
```

Можно использовать побитовое «И» (&) для формирования байта конфигурации из приведённых значений, например:

```
init = CAN_CONFIG_SAMPLE_THRICE &
CAN_CONFIG_PHSEG2_PRG_ON &
CAN_CONFIG_STD_MSG &
CAN_CONFIG_DBL_BUFFER_ON &
CAN_CONFIG_VALID_XTD_MSG &
```

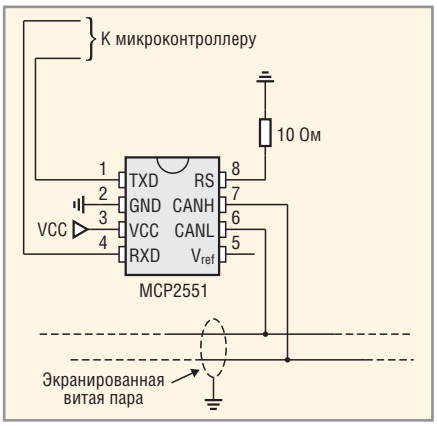

Рис. 5. Схема подключения микроконтроллера к шине CAN с помощью приёмопередатчика MCP2551

```
CAN_CONFIG_LINE_FILTER_OFF;
...
// инициализация CAN
CANInitialize(1, 1, 3, 3, 1, init);
```

Константы CAN_TX_MSG_FLAGS – это флаги, относящиеся к передачам сообщения CAN:

```
#define CAN_TX_PRIORITY_BITS 0x03
#define CAN_TX_PRIORITY_0 0xFC //
XXXXXX00
#define CAN_TX_PRIORITY_1 0xFD //
XXXXXX01
#define CAN_TX_PRIORITY_2 0xFE //
XXXXXX10
#define CAN_TX_PRIORITY_3 0xFF //
XXXXXX11
#define CAN_TX_FRAME_BIT 0x08
#define CAN_TX_STD_FRAME 0xFF //
XXXXX1XX
#define CAN_TX_XTD_FRAME 0xF7 //
XXXXX0XX
#define CAN_TX_RTR_BIT 0x40
#define CAN_TX_NO_RTR_FRAME 0xFF
// X1XXXXXX
#define CAN_TX_RTR_FRAME 0xBF //
X0XXXXXX
```

Здесь также можно использовать побитовое «И» для настройки на требуемую комбинацию флагов, например:

```
/* формирование значения, используемого в CANSendMessage: */
send_config = CAN_TX_PRIORITY_0 &
CAN_TX_XTD_FRAME &
CAN_TX_NO_RTR_FRAME;
...
CANSendMessage(id, data, 1,
send_config);
```

Флаги CAN_RX_MSG_FLAGS относятся к приёму сообщения CAN. Если отдельный бит установлен, соответствующее значение является истинным (TRUE), а в противном случае – ложным (FALSE), например:

```
// Использовать для доступа к битам фильтра
#define CAN_RX_FILTER_BITS 0x07
#define CAN_RX_FILTER_1 0x00
#define CAN_RX_FILTER_2 0x01
#define CAN_RX_FILTER_3 0x02
#define CAN_RX_FILTER_4 0x03
#define CAN_RX_FILTER_5 0x04
#define CAN_RX_FILTER_6 0x05
#define CAN_RX_OVERFLOW 0x08 //
Установлен, если переполнение,
иначе сброшен
#define CAN_RX_INVALID_MSG 0x10
// Установлен, если сообщение недействительное, иначе сброшен
#define CAN_RX_XTD_FRAME 0x20 //
Установлен, если сообщение XTD,
иначе сброшен
#define CAN_RX_RTR_FRAME 0x40 //
Установлен, если сообщение RTR,
иначе сброшен
#define CAN_RX_DBL_BUFFERED 0x80
// Установлен, если сообщение с
двойной аппаратной буферизацией
```

Можно использовать побитовое «И» для настройки на требуемую комбинацию флагов, например:

```
if (MsgFlag & CAN_RX_OVERFLOW !=
0) {
...
// Обнаружено переполнение при
приёме
// Предыдущее сообщение потеряно
}
```

Константы CAN_MASK определяют коды масок. Функция CANSetMask предполагает использование одной из них в качестве аргумента:

```
#define CAN_MASK_B1 0
#define CAN_MASK_B2 1
```

Константы CAN_FILTER определяют коды фильтрации. Функция CANSetFilter предполагает использование одной из них в качестве своего аргумента:

```
#define CAN_FILTER_B1_F1 0
#define CAN_FILTER_B1_F2 1
#define CAN_FILTER_B2_F1 2
#define CAN_FILTER_B2_F2 3
#define CAN_FILTER_B2_F3 4
#define CAN_FILTER_B2_F4 5
```

В листинге на сайте журнала приведён пример простой программы, которая демонстрирует поддержку протокола интерфейса CAN.

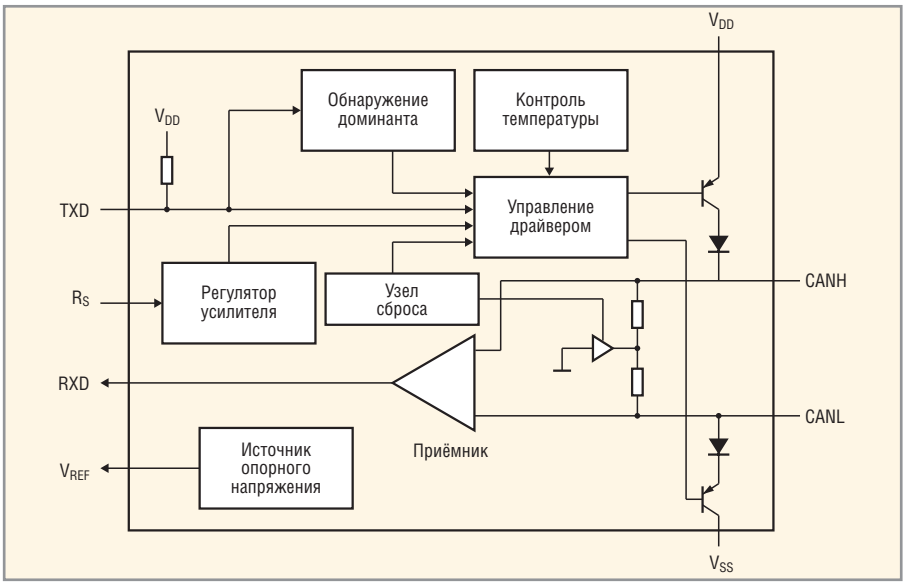

Рис. 6. Блок-схема приёмопередатчика MCP2551

Подключение аппаратуры

Схема подключения микроконтроллера к шине CAN с помощью популярного приёмопередатчика CAN типа MCP2551 приведена на рисунке 5. Данный приёмопередатчик представляет собой восьмивыводную ИС, которая преобразует уровни стандартной логики передающего и приёмного сигналов в физические уровни сигналов сети CAN. Блок-схема приёмопередатчика MCP2551 показана на рисунке 6.

Интерфейс CAN можно подключить к микроконтроллеру и через интерфейс SPI, например, при помощи микросхемы контроллера CAN типа

MCP2510. Пример такой схемы подключения приведён на рисунке 7. Для данного варианта подключения также существует библиотека готовых функций поддержки CAN-интерфейса, которые идентичны по назначению описанным выше и имеют префикс CANSPI. Перед инициализацией функций CANSPI следует вызвать функцию SPI_init().

Протоколы высокого уровня

В базовой спецификации CAN отсутствуют многие возможности, необходимые в реальных системах, например, передача данных длиннее 8 байт, автоматическое распределение идентификаторов между устройствами, единообразное управление устройствами различных типов и производителей. Поэтому вскоре после появления интерфейса CAN для него начали разрабатывать протоколы высокого уровня. К числу самых распространённых на данный момент протоколов относятся CANopen, DeviceNet, CAN Kingdom, J1939 и SDS.

Заключение

Оценивая CAN-интерфейс, можно убедиться в том, что он имеет множество преимуществ, к которым можно смело отнести:

- возможность работы в режиме реального времени;
- простоту реализации и минимальные затраты на использование;
- высокую устойчивость к помехам;
- арбитраж доступа к сети без потери пропускной способности;
- надёжный контроль ошибок передачи и приёма;
- широкий диапазон скоростей работы;
- доступность технологии, наличие широкого ассортимента продуктов от различных поставщиков.

В то же время интерфейс имеет ряд недостатков:

- обратно пропорциональную зависимость длины сети от скорости передачи;
- большой объём служебных данных в пакете по отношению к полезным данным;
- отсутствие единого общепринятого стандарта на протокол высокого уровня.

Стандарт сети CAN предоставляет возможности для практически безошибочной передачи данных между

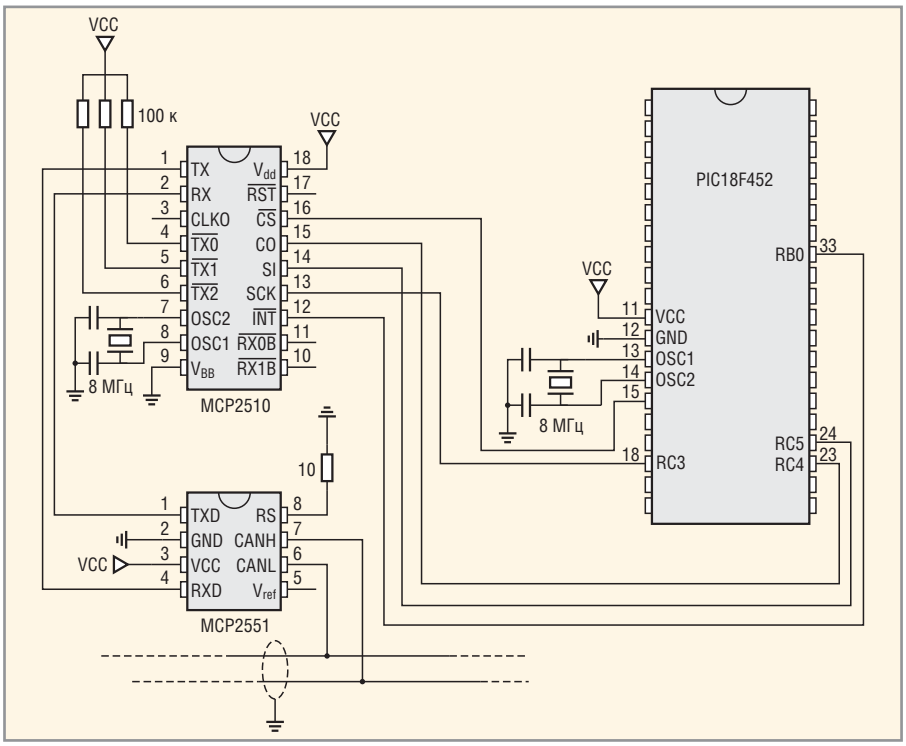

Рис. 7. Схема подключения шины CAN к микроконтроллеру через интерфейс SPI

устройствами. Интерфейс CAN позволяет передавать любой поток информации, который не превышает пропускную способность шины.

Аналогично рассмотренным вариантам использования PIC-микроконтроллеров, можно применять для работы с интерфейсом CAN и микроконтроллеры других семейств. В частности, существует среда разработки mikroC for 8051 для поддержки микроконтроллеров семейства I8051 и mikroC for AVR для микроконтроллеров семейства AVR [3].

Литература

1. www. mikroe.com.
2. www.microchip.com.
3. *Вальпа О.* Современная среда разработки mikroC для программирования микроконтроллеров на языке высокого уровня Си. Современная электроника. 2010. № 6. С. 64.