ZX-ESP WiFi internet card API

Оглавление

1. Ограничения и возможности

1.1 Ограничения ZX-SPECTRUM

С точки зрения нормальной работы по обмену данными, основным ограничением ZX-SPECTRUM является отсутствие прерываний от COM-порта.

В некоторых клонах такое прерывание вроде бы и есть, но в общем случае — его, увы нет. Это приводит к тому, что при разработке асинхронных систем обмена данными, а интернет-соединение - именно такая система в общем случае.

По-простому — асинхронная, значит, что данные после установления соединения могут прилететь в любой момент. И объём этих данных неизвестен.

Выход — постоянный опрос COM-порта. Чем это чревато — объяснять не надо. COM-порт устройство относительно медленное и ваша программа будет постоянно висеть и ждать - «а не пришел ли пакет с той стороны?», вместо того, чтобы заниматься чем-то полезным.

Разумеется, опрос можно повесить на прерывание и впадать в состояние ожидания только когда «на той стороне» будут данные. Это улучшит картину, но не сильно.

Пока смиримся с тем что есть.

1.2 Возможности ESP8266

Возможности ESP8266 весьма широки:

- поддержка режима WiFi-точки доступа или станции (можно и то и другое одновременно, но, говорят, не очень хорошо работает — сам не проверял);
- поддержка до 4х IP-соединений одновременно (для ZX хватит за глаза);
- поддержка слушающих и соединяющихся сокетов (то есть можно как коннектится к серверам, так и самому быть сервером);
- очень неплохой SDK, который позволяет на языках C/C++ реализовать многие сетевые фантазии и запихать их в вашу уникальную прошивку;
- встроенная периферия — в частности UART, он же - COM-порт.

Для нас важно, что мы можем реализовать удобный нам протокол с помощью этого SDK не оглядываясь на ограничения AT-команд стандартной прошивки.

2. Минимально необходимое API

Что нам необходимо для работы с сетью? А ничего выдумывать не надо.

Можно просто поглядеть как это сделано в существующих системах и сделать упрощённую копию API со стороны пользователя. Уникальным будет только API для настройки WiFi.

Сразу оговорюсь, что скорость обмена по COM-порту будет фиксированной — 115200. Может потом её можно будет и изменять, но пока забью на эти заморочки.

Итак, минимальное API для настройки WiFi.

2.1 WiFi API

Функция настройки WiFi:

int8_t wifi_config(uint8_t mode, const char* name, const char* pass, uint8_t auth);

mode - режим (AP/Station)
name - имя точки доступа;
pass - пароль;
auth - режим аутентификации (только для AP).

Возвращает функция 0 если всё хорошо или код ошибки.

Функция проверки состояния WiFi:

int8_t wifi_status(uint8_t* mode, char* name, char* pass, uint8_t* auth);

Возвращает 0 если всё хорошо или код ошибки, так же возвращает:

mode - режим (AP/Station)
name - имя точки доступа;
pass - пароль;
auth - режим аутентификации (только для AP).

Для AP и Station режимов коды ошибок разные.

2.2 Socket API

Функции работы с сокетами. Я честно содрал их из юникса. Чтобы было:)

2.2.1. Общие функции

int8_t socket(int8_t domain, int8_t type, int8_t protocol);

Создаёт сокет.
domain — для IP — всегда AF_INET;
type — SOCK_STREAM для TCP и SOCK_DGRAM для UDP (для начала хватит);
protocol — всегда 0.
Возвращает номер соединения (>=0) или код ошибки (<0).

int8_t close(int8_t fd);

Закрывает сокет.
fd — номер сокета (>=0).
Возвращает 0, если все в порядке или код ошибки (<0).

void esp_poll();

Тот самый мерзкий вызов, который опрашивает состояние ESP8266 по COM-порту и вычитывает данные. Должен вызываться периодически. Может быть — в прерывании.

2.2.2. Функции соединяющегося сокета

int8_t connect(int8_t sockfd, const struct sockaddr *addr, int16_t addrlen);

Установить соединение.
sockfd — номер сокета, созданного функцией **socket()**;
sockaddr — адрес (для IP — это IP-адрес и номер порта);
addrlen — размер структуры sockaddr (он разный для разных сокетов, но у нас пока всегда 6 байт).
Возвращает 0, если соединение установлено или код ошибки, если нет.

int16_t recv(int8_t sockfd, void *buf, int16_t len, int8_t flags);

Читает данные из сокета.
sockfd — номер сокета;
buf — буфер, куда читать данные;
len — размер буфера;
flags — флаги (пока не используются).

Возвращает: количество считанных байт (>0), код ошибки (<0) или, если возвращен 0, то соединение закрыто.

int16_t send(int8_t sockfd, const void *buf, int16_t len, int flags);
Отправляет данные в сокет.
sockfd — номер сокета;
buf — буфер, с данными данные;
len — сколько байт нужно отправить;
flags — флаги (пока не используются).

Возвращает: количество переданных байт (>0), код ошибки (<0) или, если возвращен 0, то соединение закрыто.

2.2.3. Функции слушающего сокета (может быть потом и их реализуем)

int8_t bind(int8_t sockfd, const struct sockaddr *addr, int16_t addrlen);

int8_t listen(int8_t sockfd, int8_t backlog);

int8_t accept(int8_t sockfd, struct sockaddr *addr, int16_t *addrlen);

3. Протокол обмена

По сути, необходимо преобразовать асинхронный обмен данными в синхронный.

Управляющим устройством (master) будет ZX, а ESP8266 должно выполнять его команды.

Все принятые данные ESP8266 должно хранить в своих буферах, насколько позволит размер памяти.

Так как к COM-порту будет привязано не более одного устройства, то команды — безадресные.

В качестве транспорта используем протокол SLIP.

Границей SLIP-кадра является уникальный флаг END (0xC0). Уникальность этого флага поддерживается байт-стаффингом внутри кадра с ESC-последовательностью 0xDB, причём байт END (0xC0) заменяется последовательностью (0xDB, 0xDC), а байт ESC (0xDB) — последовательностью (0xDB, 0xDD).

Такой формат позволит точно найти начало и конец пакета (кадра) при сбое, накладные расходы на анализ байт — минимальны.

Общий формат команд (исходный, до байт-стаффинга):

Резерв, всегда 0 **reserv0**	**Код команды** **cmd**	**ID команды** **id**	**Данные (зависят от cmd)** **data**	**CRC8 данных** **crc8**
1 байт	**1 байт**	**1 байт**	**NNNN байт**	**1 байт**
Заголовок				

Каждая команда предполагает ответ от ESP8266.

Заголовок команды и ответа состоит из:

- резервного байта **reserv0**, всегда равного 0 (потом может поменяться);
- кода команды **cmd**. Коды команды приведены в табл.1;
- идентификатора команды **id**. **id** ответа всегда равен **id** команды, на которую получен ответ;
- данных **data**, размер которых зависит от конкретной команды. Данные могут и отсутствовать;
- контрольной суммы crc8, которая вычисляется от байта **reserv0** до последнего байта данных включительно.

Так как протокол SLIP имеет чёткие границы пакета, то нет необитаемости передавать размер поля **data**; он вычисляется при приёме данных.

Примечание. В описании команд описываются только поля данных

запроса и ответа без байт-стаффинга. То есть пакет имеет вид без преобразования к формату SLIP-протокола. Поля заголовка и контрольной суммы в описании команд опускаются. Коды команд указаны в в табл.1.

3.1 Коды команд

Табл. 1: Коды команд, указываемые в SLIP-пакетах

Команда	Код	Комментарий
Общие команды		
gettxtinfo	**0x01**	**Получить отладочный вывод ESP**
esp_poll	**0x02**	**Проверить состояние сокетов на ESP и вернуть его в ZX**
Команды работы с WiFi		
wifi_config	**0x08**	**Задать конфигурацию WiFi**
wifi_status	**0x09**	**Получить конфигурацию WiFi**
Команды управления сокетом		
socket	**0x10**	**Создать сокет**
close	**0x11**	**Закрыть соединение и уничтожить сокет**
fcntl	**0x12**	**Задать режим чтения-записи (блокирующий или не блокирующий)**
Клиентская часть (ZX соединяется с сервером)		
connect	**0x18**	**Соединиться с заданным IP**
recv	**0x19**	**Принять данные из указанного сокета**
send	**0x1A**	**Оправить данные в указанный сокет**
Серверная часть (ZX является сервером)		
bind	**0x20**	**Привязать сокет к адресу и порту (по факту только к порту)**
listen	**0x21**	**Слушать указанный сокет и ждать запроса на соединение**
accept	**0x22**	**Принять соединение**

3.2 Общие команды

3.2.1. Команда esp_poll

Данная команда осуществляет опрос состояния ESP8266 и возвращает данные по нему.

Запрос. Данные отсутствуют.
Ответ.

WiFiStatus	**Nsock**	**SockStatus**
1 байт	**1 байт**	**3*Nsock байт**

WiFiStatus — состояние соединения с сетью WiFi

Байт состояния соединения с сетью WiFi имеет следующий формат:

Таблица 1: Байт WiFiStatus

Бит	**Значение**
0	**Режим (0-Station 1-AP)**
1	**Для Station — 1 — есть подключение к AP, 0 — нет подключения к AP**
2	**0**
3	**0**
4	**0**
5	**0**
6	**0**
7	**0 — нет информационного вывода. 1 — есть информационный вывод и его надо считать запросом gettxtinfo. Это просто текст и применяется для отладки.**

Nsock — количество открытых сокетов.
SockStatus — массив трёхбайтовых записей о состоянии каждого сокета.

Каждая запись массива **SockStatus** состоит из трёх байт и содержит следующую информацию:

SockStatus, байт 0 — содержит номер сокета;
SockStatus, байт 1 — содержит код ошибки (0-ошибок нет)
SockStatus, байт 2 — содержит биты состояния сокета (см. табл).

Таблица 2: SockStatus, байт 2

Бит	**Значение**

0	**Сокет (1) открыт или закрылся (0).** **0 будет передаваться до тех пор пока не будет выполнено чтение из сокета.**
1	**0 — возможна отправка данных по сокету. 1- буфер полон.**
2	**0 — данные для чтения в сокете отсутствуют. 1 — можно читать данные.**
3	**0 — соединяющийся сокет 1 — слушающий сокет**
4	**0 — для слушающего сокета нет входящих соединений, 1 — есть соединение (принять с помощью accept или закрыть с помощью close)**
5	**0**
6	**0**
7	**0**

3.2.2. Команда gettxtinfo

Запрос. Данные отсутствуют.
Ответ.

text
ASCIIZ

Содержит одну строку, завершающуюся нулём. Используется для отладки.

3.3 Команды работы с WiFi

3.3.1. Команда wifi_config

Запрос.

mode	auth	name	password
1 байт	1 байт	(ASCIIZ)	(ASCIIZ)

mode — режим (0-Station 1-AP)
auth — тип авторизации (опишем позже)
name — имя (завершается \0)
password — пароль (завершается \0)

Ответ.

WiFiStatus	Error
1 байт	1 байт

WiFiStatus — аналогично **poll**
Error — код ошибка (0 — ошибок нет)

3.3.2. Команда wifi_ status

Запрос. Данные отсутствуют.
Ответ.

mode	auth	name	password	WiFiStatus	Error
1 байт	1 байт	(ASCIIZ)	(ASCIIZ)	1 байт	1 байт

Все поля аналогично **wifi_config**.

3.4 Команды управления сокетом

3.4.1. Команда socket

Запрос.

domain	type	protocol
1 байт	1 байт	1 байт

domain — AF_INET (пока только IP-пакеты)
type — SOCK_STREAM для TCP и SOCK_DGRAM для UDP
protocol — 0 (резерв)

Ответ.

ExitCode
1 байт

ExitCode — код ошибки (<0) или номер созданного сокета (>=0)

3.4.2. Команда close

Запрос.

fd_sock
1 байт

fd_sock — номер закрываемого сокета

Ответ.

ExitCode
1 байт

ExitCode — код ошибки (<0) или 0, если сокет успешно закрыт

3.4.3. Команда fcntl

Запрос.

fd_sock	cmd_fcntl	data
1 байт	1 байт	N байт

fd_sock — номер сокета
cmd_fcntl — команда управления сокетом
data — данные, зависит от значения **cmd_fcntl** и могут вовсе отсутствовать

Ответ.

fd_sock	**ExitCode**	**cmd_fcntl**	**data**
1 байт	**1 байт**	**1 байт**	**N байт**

fd_sock — номер сокета
ExitCode — зависит от команды
cmd_fcntl — команда управления сокетом на которую пришёл ответ
data — данные, зависит от значения cmd_fcntl и могут вовсе отсутствовать

3.5 Клиентская часть (ZX соединяется с сервером)

3.5.1. Команда connect

Запрос.

fd_sock	addr
1 байт	1 байт

fd_sock — номер сокета, ранее созданного вызовом **socket()** ;
addr — адрес, с которым нужно соединиться (для IP — это 4 байта IP-адрес и 2 байта порт);

Ответ.

ExitCode
1 байт

ExitCode — код ошибки (<0) или 0 если соединение установлено.

3.5.2. Команда recv

Запрос.

fd_sock	flags	dlen
1 байт	1 байт	2 байта

fd_sock — номер сокета, из которого хотим прочитать;
flags — флаги (пока не используются, всегда 0);
dlen — сколько байт хотим прочитать (максимум);

Ответ.

ExitCode	fd_sock	flags	dlen	data
1 байт	1 байт	1 байт	2 байта	len байт

ExitCode — код ошибки (<0) или 0 если чтение удалось;
fd_sock — номер сокета, из которого читали;
flags — флаги (пока не используются, всегда 0);
dlen — сколько байт реально прочитано (**dlen**==0 — соединение закрыто, если **ExitCode** == 0). Если сокет не закрыт, но данных ещё не поступало, то возвращается ошибка EAGAIN.
data — прочитанные данные. Если нет данных, то **dlen**==0 и поле **data** отсутствует.

3.5.3. Команда send

int16_t send(int8_t sockfd, const void *buf, int16_t len, int flags);

Запрос.

fd_sock	flags	dlen	data
1 байт	1 байт	2 байта	dlen байт

fd_sock — номер сокета, в который хотим писать;
flags — флаги (пока не используются, всегда 0);
dlen — сколько байт хотим записать (максимум);
data — записываемые данные.

Ответ.

ExitCode	fd_sock	flags	dlen
1 байт	1 байт	1 байт	2 байта

ExitCode — код ошибки (<0) или 0 если запись удалась;
fd_sock — номер сокета, из которого читали;
flags — флаги (пока не используются, всегда 0);
dlen — сколько байт реально записано. Если данные не удалось записать, то dlen==0.

3.6 Серверная часть (ZX является сервером)