Time interval controller (TIC)

Таблица активности временных слотов. (timeSlotStateTable)

Состояние временного слота 0 timeSlotState	**Состояние временного слота 1 timeSlotState**	**Состояние временного слота N timeSlotState**	**Состояние временного слота 49 timeSlotState**
Флаг приема	Флаг приема	Флаг приема	Флаг приема
Флаг передачи	Флаг передачи	Флаг передачи	Флаг передачи

Аппаратный таймер 32.768K

setTimer(uint16_t time)
setInterruptTime(uint16_t time, timeslotNum)
setInterruptHandler((*handlerIRQ)(*timeslot))
adjustTimer (int adjustTime)

Обработчик аппаратного таймера hwTimerHandler(timeSlot)

Проверяем поле "Флаг передачи" таблицы timeSlotStateTable
Если поле TRUE
    Вызываем обработчик передачи пакета(timeSlot)
    Вызываем TDMAScheduler(timeSlot)
    Вызываем обработчик slotEnd(timeSlot)
    ВЫХОД
Проверяем поле "Флаг приема" таблицы timeSlotStateTable
Если поле TRUE
    Вызываем обработчик приема пакета(timeSlot)
    Вызываем TDMAScheduler(timeSlot)
    Вызываем обработчик slotEnd(timeSlot)
    ВЫХОД
Вызываем TDMAScheduler(timeSlot)
Вызываем обработчик slotEnd(timeSlot)
ВЫХОД

Планировщик TDMA. TDMASheduler(timeSlot)

Перебираем следующие timeSlot из timeSlotStateTabe
Если "Обработчик прием пакета" не равен NULL
ИЛИ "Обработчик передачи пакета" не равен NULL
    Устанавливаем аппаратный таймер на этот таймслот
    ВЫХОД
Тут мы окажемся если обработчиков нет.
Устанавливаем аппаратный таймер на таймслот 0

Time interval controller (TIC)

init()	Инициализация

setReceiveCallback(ticCallback_f handler)	Устанавливает обработчик интервала приема пакета
setSendCallback(ticCallback_f handler)	Устанавливает обработчик интервала передачи пакета
setSlotEndCallback(ticCallback_f handler)	Устанавливает обработчик по завершению слота
clearCallbacks()	Удаляет все обработчики
setSlotINTCallback(ticCallback_f handler)	Устанавливает обработчик по завершению прерывания

bool setReceptionState(timeSlot_t n, bool state)	Установить таймслот в режим приема пакетов
bool setTransmissionState(timeSlot_t n, bool state)	Установить таймслот в режим передачи пакета
bool getReceptionState(timeSlot_t n)	Прочитать активность состояния приема пакета
bool getTransmissionState(timeSlot_t n)	Прочитать активность состояния передачи пакета

bool adjustTimer(usec_t usec)	Подстроить значение таймера
bool setTimer(usec_t usec)	Установить значение таймера
usec_t getTimer()	Прочитать состояние таймера

ticCallback:Type
typedef void (*ticCallback_f)(timeSlot_t n)

Media access control (MAC)

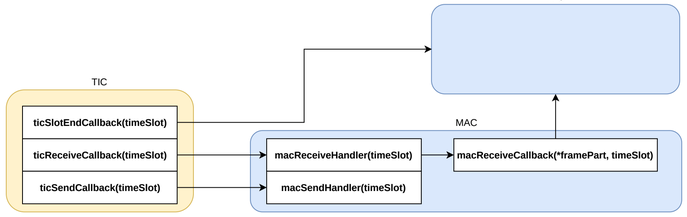
Состояние слотов. (slotStateTable)

Состояние слота 0 slotState	**Состояние слота N slotState**	**Состояние слота 49 slotState**
Состояние передачи	*Состояние передачи*	*Состояние передачи*
Флаг "Есть данные"	Флаг "Есть данные"	Флаг "Есть данные"
Количество попыток передачи	Количество попыток передачи	Количество попыток передачи
Канал радиопередатчика	Канал радиопередатчика	Канал радиопередатчика
Указатель на пакет framePart	Указатель на пакет framePart	Указатель на пакет framePart
Состояние приема	*Состояние приема*	*Состояние приема*
Флаг "Прием разрешен"	Флаг "Прием разрешен"	Флаг "Прием разрешен"
Канал радиопередатчика	Канал радиопередатчика	Канал радиопередатчика

Принятый пакет receivedPacket
Указатель на пакет framePart
timeSlot принятого пакета

Пакет framePart создаеться malloc.

LLC



macSendHandler должен еще заполнять пакеты sync.
Нужно подумать кто будет заполнять. MAC или протокол SYNC.

Media access control (MAC)

init()	Инициализация
mac_setReceiveCallback(ticCallback_f handler)	Устанавливает обработчик интервала приема пакета
mac_setSendCallback(ticCallback_f handler)	Устанавливает обработчик интервала передачи пакета
mac_clearCallbacks()	Удаляет все обработчики
mac_openReceiveSlot(timeSlot n, uint8_t ch)	Разрешает прием пакетов в заданный слот и заданном канале.
mac_closeReceiveSlot(timeSlot n)	Запрещает прием пакетов в заданный слот
bool mac_send(framePacket* framePacket, timeSlot n, uint8_t ch, uint8_t attempts)	Устанавлевает пакет для передачи в слот n, канале ch, количеством попыток attempts.Если тайм слот еще занят возвращает FALSE.
bool mac_getReceivedPacket(framePart* framePart)	Возвращает указатель на последний принятый пакет. FALSE если пакетов нет

Обработчик ticCallback_f sendHandler(timeSlot)

Процедура обработки события передачи пакета TIC
Если флаг "Есть данные" = FALSE
  Запрещаем TIC обработку. setTransmissionState(timeSlot, FALSE)
  ВЫХОД
Включить радиопередатчик, выставить частоту канала передачи
Если PPDU TYPE = WP
  Начать передачу
  Если передача не удалась (контроль CCA)
    Количество попыток - 1
    Если количество попыток = 0
      Уничтожаем пакет, флаг "Есть данные" = FALSE
      Запрещаем TIC обработку. setTransmissionState(timeSlot, FALSE)
    Выключаем радио, ВЫХОД
  Прием пакета в течении 1 мс(?)
  Если приняли ACK
    Уничтожаем пакет, флаг "Есть данные" = FALSE
    Запрещаем TIC обработку. setTransmissionState(timeSlot, FALSE)
    Выключаем радио, ВЫХОД
  Если не приняли ACK
    Количество попыток - 1
      Если количество попыток = 0
        Уничтожаем пакет, флаг "Есть данные" = FALSE
        Запрещаем TIC обработку. setTransmissionState(timeSlot, FALSE)
      Выключаем радио, ВЫХОД
ЕСЛИ PPDU TYPE = SYNC
  Заполнить поля HOUR, MIN, SEC, USEC = 0x80000000
  Начать передачу специальной функцией
  Если передача не удалась
    Количество попыток - 1
      Если количество попыток = 0
        Уничтожаем пакет, флаг "Есть данные" = FALSE
        Запрещаем TIC обработку. setTransmissionState(timeSlot, FALSE)
  Если передача удалась
    Уничтожаем пакет, флаг "Есть данные" = FALSE
    Запрещаем TIC обработку. setTransmissionState(timeSlot, FALSE)
    Выключаем радио, ВЫХОД
Если PPDU TYPE = WP BRADCAST
  Начать передачу
  Если передача не удалась (контроль CCA)
    Количество попыток - 1
    Если количество попыток = 0
      Уничтожаем пакет, флаг "Есть данные" = FALSE
      Запрещаем TIC обработку. setTransmissionState(timeSlot, FALSE)
    Выключаем радио, ВЫХОД

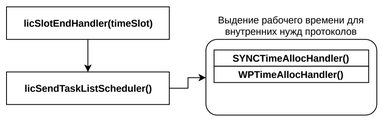
Обработчик ticCallback_f receiveHandler(timeSlot)

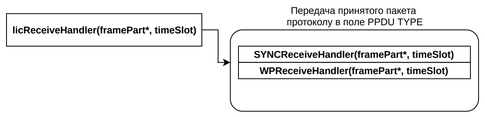
Процедура обработки события приема пакета TIC
Если флаг "Прием разрешен" равен FALSE
  Запрещаем TIC обработку. setTransmissionState(timeSlot,FALSE)
  ВЫХОД
Включить радиопередатчик, выставить частоту канала приема
Ждать приема не более 2-3 мс(?)
Если пакета не пришло
  Выключить радио, ВЫХОД
Если проверка NETID разрешена
  Если NETID не равен нашей сети
    Уничтожаем, выключаем радио, ВЫХОД
Если версия протокола не равна нашей версии
  Уничтожаем, выключаем радио, ВЫХОД
Если PPDU TYPE = WP
  Если размер пакета меньше 26 байт
    Уничтожаем, выключаем радио, ВЫХОД
  Если поле DST не соответсвует адресу узла
    Уничтожаем, выключаем радио, ВЫХОД
  Если поле DLEN не равно размер PPDU - 26 (размер DATA неверен)
    Уничтожаем, выключаем радио, ВЫХОД
  Передаем пакет ACK
  Выключаем радио
  Вызов обработчика приема пакета. macReceiveCallBack(*PPDU, timeSlot)
  ВЫХОД

Link logic control(LLC)

Список задач отправки.
sendTaskList

Задача отправки sendTask
Указатель sendTaskLast
Указатель sendTaskNext
Указатель на пакет framePart
Номер временного слота
Канал радиопередатчика





init()	Инициализация
LLC_addTask(framePart* framePart, timeSlot_t n, uint8_t ch)	Добавить задачу передачи пакета

licSendTaskListScheduler()

Составляет расписание на базе списка задач отправки сообщений
Перебор всех элементов списка sendTaskList
  Если временной слот MAC содержит данных для передачи
  (getTransmissionState(timeSlot_t n) = TRUE)
  Выбор следующего элемента списка
  Добавляем данные для передачи во временной слот
  Удаляем элемент списка
ВЫХОД

Элементарный элемент данных пакета FItem

FItem_s	
FItem_s*	next
FItem_s*	last
FItem_t	type
uint8_t	length
uint8_t*	data

FItem_t	
PPDU_HEADER	1
PPDU_FOOTER	2
MPDU_WP	3
MPDU_MDATA	4
MPDU_MIC	5
RAW	6
SYNC	7
METADATA	8

Global params		
uint8_t	quantity	Кол-во созданных элементов

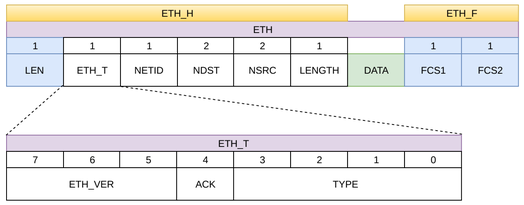
Методы	**Описание**	
bool FI_create(FItem_s* fi, FItem_t type, uint8_t* data, uint8_t length)	Создает FItem указанного типа	Увеличивает или уменьшает глобальный quantity
void FI_delete(FItem_s* fi)	Удаляет указаный FItem. Соседей не связывает	
void FI_setNext(FItem_s* fi, FItem_s* fi_next)	Связывает со следующим соседом	
void FI_setLast(FItem_s* fi, FItem_s* fi_last)	Связывает с предыдущем соседом	
void FI_getNext(FItem_s* fi, FItem_s* fi_next)	Возвращает следующего соседа	
void FI_getLast(FItem_s* fi, FItem_s* fi_last)	Возвращает предыдущего соседа	
FItem_t FI_getType(FItem_s* fi)	Возвращает предыдущего соседа	
uint8_t FI_getLength(FItem_s* fi)	Возвращает длинну данных	
uint8_t* FI_getData(FItem_s* fi)	Возвращает указатель на данные	
uint8_t FI_getObjectQuantity()	Возвращает количество существующих обьектов	

METADATA	
RX	
uint16_t	TIMESTAMP
int8_t	RSSI
TX	
txMethod_e	TX METHOD
void* (framePart)	SFD_CALLBACK
uint8_t	ETH_H.ETH_T.TYPE
bool	ACK

Медуль реализован в виде отдельных функция по работе с структурой FItems_s. Элементы создаются на куче, данные data копируются при создании элемента. Предполагается, что размер данных никогда меняться не будет вплоть до уничтожения обьекта. Контроль количества существующих обьектов в памяти производится с помощью uint8_t FI_getObjectQuantity()

txMethod_e	
BROADCAST	1
UNICAST	2
CALLBACK	3

Структура ethernet пакета



LEN	Общая длина посылки. Поле не контролируется в CRC.
FCS1,2	Поля контролируются радиопередатчиком. Заменяются на RSSI, CRC.
ETH_T	Кодирует тип содержимого и версию протокола.
NETID	Идентификатор принадлжености к сети. Значения от 0 до 0xFF
NDST	Адрес узла получателя пакета.
NSRC	Адрес узла отправителя пакета.
LENGTH	Размер содержимого MPDU
DATA	Данные
ETH_VER	Версия протокола. Значения от 0 до 7.
TYPE	Номер низкоуровневого протокола. Значения от 0 до 15
ACK	1 - Пакет требует подтверждения.

Порядок фильтрации пакетов

№	Поле	Значение	Описание	Действие
1	FCS1,2	CRC	Неверная контрольная сумма	Удаление
2	LEN	RXBUF.LEN	Поле LEN не равно фактической длинне пакета	Удаление
3	NETID	Сеть	Пакет не пренадлежит сети	Удаление
4	ETH_VER	Версия узла	Узел не поддерживает версию протокола	Удаление
5	TYPE	Весия	Узел не имеет обработчика протокола	Удаление
6	LENGTH	Вычисляем	Поле размера данных некоректно	Удаление
7	NDST	0xFFFF	Пакет для всех узлов	Обработка
8	NDST	Адрес узла	Пакет для этого узла	Обработка

*NSRC = 0xFFFF Обозначает пакет от узла без адреса.

При начальной синхронизации фильтры пакетов отключаются, или настриваются на прием пакета SYNC. Применик не разбирает каким образом был передан пакет: либо в ШВС тайм слот 0, или как unicast сообщение, главное что поле NDST = 0xFFFF. При ШВС ответа ACK не должно быть, а при unicast должно быть. Этот момент под ответсвенностью отправителя пакета.

Фильтрация пакетов

Каждый уровень имеет свою логику по фильтрации пакетов. Уровень MAC применяет фильтры непосредственно ethernet frame. Фильтры MAC включают в себя проверку своевременности прихода пакетов, к примеру NDST = 0xFFFF могутприходить только в слот 0. NDST = 0xFFFF, должны иметь ACK = 0.

Структура SYNC пакета

SYNC				
1	1	1	1	2
ETX	HOUR	MIN	SEC	TIMER32K

ETX	Expected transmissions. Ожидаемое количество передач до точки доступа.
HOUR	Часы. Значения от 0 до 23
MIN	Минуты. Значения от 0 до 59
SEC	Секунды. Значения от 0 до 59
TIMER32K	Значения таймера от 0 до 32767 (32768 значений)

Узлы могут не иметь RTC и не ретранслировать часы, минуты, секунды. В этом случаии поля заполняются значениями 0xFF.

Структура ACK пакета

MPDU: ACK
0

Для подтвеждения принятого пакета отправляем пакет с типом TYPE = ACK. LENGTH = 0.
NDST = Адрес узла отправителя данных, NSRC = свой адрес.

Иерархия протоколов

Цепочка данных FChain

FChain_s	
FItem_s*	head
FItem_s*	tail
uint8_t	quantity
FItem_s*	iterator

Global params		
uint8_t	quantity	Кол-во созданных элементов

FChain унифицирует процес работы с структорой пакета данных в стеке протокола.
Интерфейс радио передатчика использует FChain, приемник возвращает FItem типа RAW

Методы	Описание
void FC_create(FChain_s* fc)	Создает цепочку
void FC_delete(FChain_s* fc)	Удаляет всю цепочку включая ее элементы
void FC_iteratorToHead(FChain_s* fc)	Устанавливает итератор на начало списка
void FC_iteratorToTail(FChain_s* fc)	Устанавливает итератор на конец списка
void FC_iteratorToType(FChain_s* fc)	Перемещает итератор с текущего положения на следующий элемент типа type
void FC_iteratorToTypeHead(FChain_s* fc)	Перемещает итератор с начала цепочки на следующий элемент типа type
void FC_next(FChain_s* fc)	Перемещаем итератор на следующий элемент
void FC_last(FChain_s* fc)	Перемещаем итератор на предыдущий элемент
void FC_insertAfter(FChain_s* fc, FItem_s* fi)	Вставляет элемент после итератора
void FC_insertBefore(FChain_s* fc, FItem_s* fi)	Вставляет элемент до итератора
uint8_t FC_getQuantity(FChain_s* fc)	Возвращает количество элементов
bool FC_getIterator(FChain_s* fc, FItem_s* fi)	Возвращает FItem итератора. true - есть элемент
void FC_getObjectQuantity()	Возвращает количество существующих обьектов

Структура IP пакета

IP_HEADER					IP_DATA	IP_MIC
1	2	2	1	1	LENGHT	4
ETX	FDST	FSRC	PROTOCOL	LENGTH	DATA	MIC
	Authentification					
					Encryption	

ETX	Expected transmissions. Ожидаемое количество передач до точки доступа.
FDST	Адрес конечного узла назначения пакета.
FSRC	Адрес узла создателя данного пакета.
PROTOCOL	Номер вышестоящего протокола обработчика пакета
LENGTH	Размер данных.
DATA	Данные.
MIC	Message integrity code. Код целостности сообщения.

№	Поле	Значение	Описание	Действие
1	LENGTH	Вычисляем	Размер данных не соответсвует длине пакета	Удаляем
2	MIC	Авторизация	Авторизация не пройдена	Удаляем
3	FDST	0xFFFF	Пакет для всех узлов	Обработка
4	FDST	Адрес узла	Пакет для этого узла	Обработка
5	FDST	0x0000	Маршрутизация до ТД алгоритмом СТР. Только пакеты с FSRC!=0xFFFF, 0x0000	Маршрутизируем
6	FDST	ANY	Маршрутизация до узла алгоритм таблиц. Только пакеты с FSRC = 0x0000	Маршрутизируем
7	PROTOCOL		Если нет обработчика. При маршрутизации не имеет значения есть ли у нас обработчик.	Удаляем

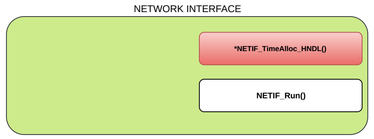
ETX не включен в авторизацию сообщения. Это поле изменяется при маршрутизации пакета.

Radio interface (RI)

Методы	Описание
void RI_on()	Включить радиопередатчик
void RI_off()	Выключить радиопередатчик
bool RI_setChannel(n)	Установить радиоканал.
bool RI_send(*framePart)	Отправить пакет. Возращает TRUE или FALSE
bool RI_sendSFD(*framePart default, callback (*framePart))	Отправка пакета с отложенным заполнением данных. После начала передачи SFD происходит обратный вызов, который возвращает данные пакета. default - данные по умолчанию
bool RI_receive(*framePart, timeout)	Примем пакета с таймаутом.
uint32_t RI_getCRCErrorCount()	Прочитать количество ошибок CRC
uint32_t RI_getCCARejectCount()	Прочитать количество отказов канала
uint64_t RI_getRadioUptime()	Время работы радио в мкс
void RI_init()	Инициализация

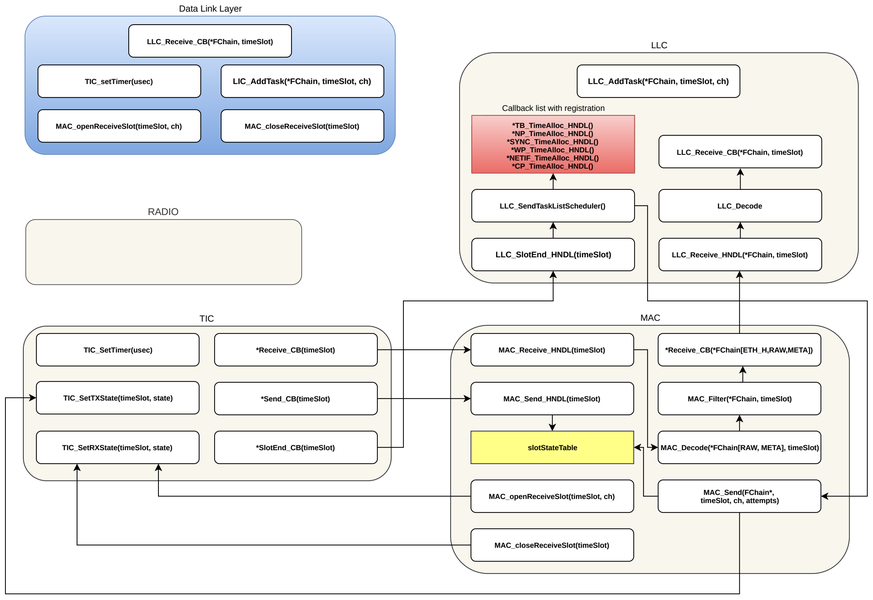
Радиоинтерфейс для передачи данных использует пакет framePart типа PPDU_HEADER. Если пакет другого типа, функция завершит свою работу.
Для передачи пакета framePart, происходит обход всей цепочки framePart и загрузка в микросхему. PPDU_FOOTER не загружается и формируется самой микросхемой. Существует специальная функция радиопередачи - RI_sendSFD. Ее особенность в том, что перед ее вызовом пакет данных не передается. Запрос данных для передачи будет произведен после начала передачи заголовка SFD с помощью обратного вызова. Очень важно, что обратный вызов должен выполниться, до того как радиопередатчик начнет передавать первый байт данных, в противном случаии будут переданы данные default.
При приеме создается framePart типа RAW и добавляется информация METADATA. При передаче METADATA не нужна

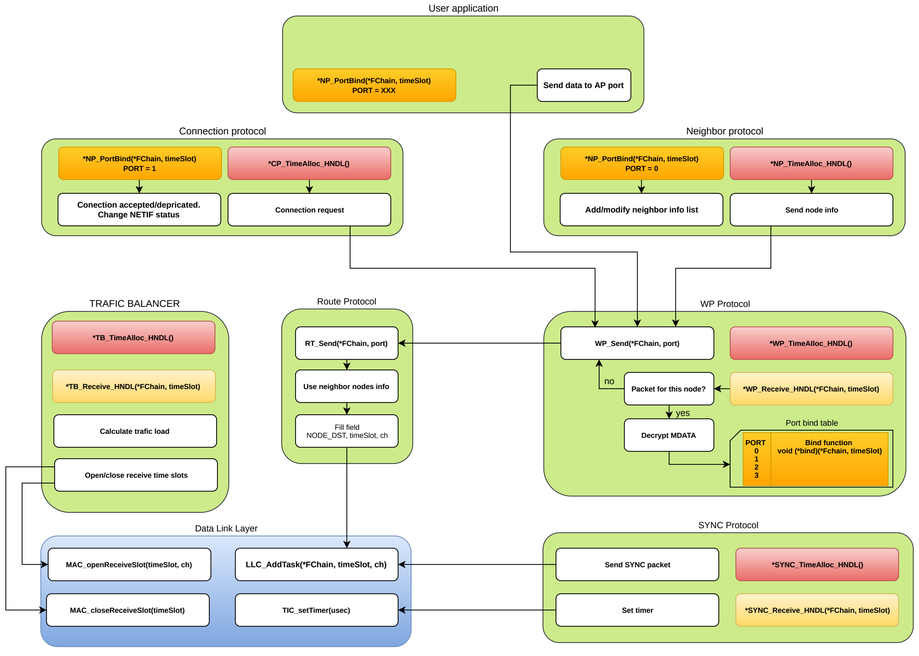
User application
Обрабатывает входящие пакеты, отправляет сообщения точке доступа. Управление получает либо из главного цикла при выходе из прерывания таймера, или по прерываниям настроенных пользователем.

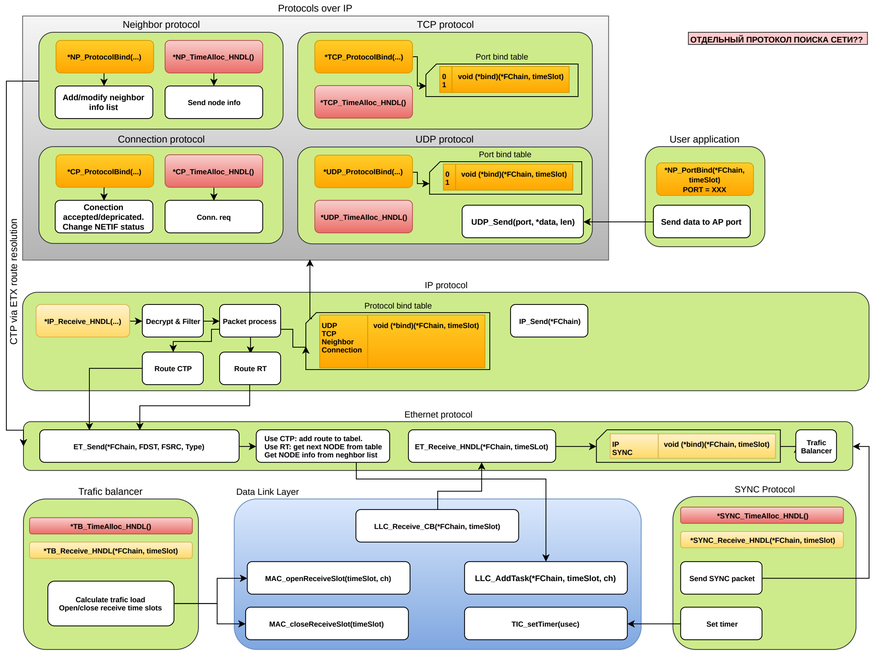


Протоколы взаимодействуют между собоай при помощи чтения состояния NETIF.
Состояние НЕ СИНХРОНЕЗИРОВАН
  РаботаетSYNC протокол в режиме прямого доступа к радио (или всеже MAC).
  Перебирает разрешенные каналы (список в NETIF)
  Ищет синхропакет и устанавлевает состояние NETIF СИНХРОНЕЗИРОВАН
  У NETIF проверяет что к сети с NETID можно подключиться
Состояние СИНХРОНИЗИРОВАН
  Считает что каждый протокол очистил свои данные пока был в режиме НЕ СИНХР.
  Начинает свою работу протокол Соседей.
  При наличии соседей установит NETIF что есть соседи.
  Если есть соседи активируется протокол Подключения.
  Он отсылает запрос шлюзу о подключении
  Обратный ответ приходит через UBC.
  Если подключение разрешено, то устнавливает NETIF В состоянииее подключенно
  Если подключение невозможно, то устанавливает флаг Подключение отказано
NETIF увидит что в подключении отказано и сбросит синхронизацию пометив сеть
на некоторое время недоступной.
В случаи успешного подключения от пользователя принимаем пакеты на передачу

Data Link Layer




User application
*NP_PortBind(*FChain, timeSlot)
PORT = XXX
Send data to AP port
Connection protocol
*NP_PortBind(*FChain, timeSlot)
PORT = 1
*CP_TimeAlloc_HNDL()
Conection accepted/depricated.
Change NETIF status
Connection request
Neighbor protocol
*NP_PortBind(*FChain, timeSlot)
PORT = 0
*NP_TimeAlloc_HNDL()
Add/modify neighbor info list
Send node info
TRAFIC BALANCER
*TB_TimeAlloc_HNDL()
*TB_Receive_HNDL(*FChain, timeSlot)
Calculate trafic load
Open/close receive time slots
Route Protocol
RT_Send(*FChain, port)
Use neighbor nodes info
Fill field
NODE_DST, timeSlot, ch
WP Protocol
WP_Send(*FChain, port)
*WP_TimeAlloc_HNDL()
no
Packet for this node?
*WP_Receive_HNDL(*FChain, timeSlot)
yes
Decrypt MDATA
Port bind table
PORT
0
1
2
3
Bind function
void (*bind)(*FChain, timeSlot)
Data Link Layer
MAC_openReceiveSlot(timeSlot, ch)
LLC_AddTask(*FChain, timeSlot, ch)
MAC_closeReceiveSlot(timeSlot)
TIC_setTimer(usec)
SYNC Protocol
Send SYNC packet
*SYNC_TimeAlloc_HNDL()
Set timer
*SYNC_Receive_HNDL(*FChain, timeSlot)



ОТДЕЛЬНЫЙ ПРОТОКОЛ ПОИСКА СЕТИ??

TIC

TIC - time interval controller, Контроллер управления интервалами времени. Реализует основной алгоритм TDMA: разделяет отрезок времени на временые слоты. Каждому слоту соответсвует одно из действий: примем или передача пакета данных. TIC самостоятельно не занимается передачей данных, а производит обратные вызовы *Receive_CB(timeSlot), *Send_CB(timeSlot). Управлять работой контроллера можно с помощью TIC_SetTXState(timeSlot, state), TIC_SetRXState(timeSlot, state), где state = true/false. Планировщик времени TIC, на основе таблицы состояний временых слотов, управляет работой аппаратного таймер. После завершения обратных вызовов, произойдет вызов *SlotEnd_CB(timeSlot), его цель уведомить о завершении временого слота и предоставить другим частям протокола время для своих действий. В случаи если нет активных действий в слотах, то планировщик будет вызывать SlotEnd_CB в конце слота 0.
Для подстройки значений таймера служит метод TIC_SetTimer(usec), он записывает новое значение времени в аппаратный таймер.
Согласно протоколу длительность временного интервала 1 сек.

MAC

MAC - media access control, контроллер доступа к среде передачи. Взаимодействует с радиоинтерфейсом и осуществляет прием или передачу пакетов. С точки зрения реализации передачи данных, MAC редставляет собой 50 буферов (по количеству временых слотов), и MAC_Send(FChain*, timeSlot, ch, attempts) устанавливает пакет для передачи в буфер timeSlot. MAC открывает слот передачи TIC. Передача пакетов произойдет в соответсвии с метододом txMethod_e указанным в METADATA пакета. Примем и передача происходят в соответсвии с протоколом обмена сообщениями.
Для подтверждения приема пакета, MAC уровень разбирает пакет на ETH_H и RAW. Тут же производится фильтрация пакетов. Данные о NDST, NSRC заносим в METADATA.
Обработка принятых пакетов происходит сразу после завершения приема данных посредством обратного вызова *Receive_CB(*IFrame=[ETH_H, RAW])

LLC

LLC - link logic controller, контроллер логики обмена данными. LLC разбирает очередь пакетов на передачу и по мере готовности MAC ставит на отправку. В отличии от MAC, когда буфер содержит только один пакет для передачи, очередь LLC содержит множество пакетов для передачи. LLC никоем образом не уведомляет верхние уровни о результатах передачи. Нельзя узнать был ли отправлен пакет данных.
Обработка завершения временого слота LLC_SlotEnd_HNDL(timeSlot). LLC запускает свой планировщик, который передаст пакеты MAC. После чего пройдется по списку зарегистрированных обратных вызовов для раздачи рабочего времени остальным участникам стека.
Асинхронный прием пакетов LLC_Receive_HNDL(*FChain[RAW,META], timeSlot). Задача LLC преобразовать формат FChain RAW в FChain с цепочкой FItem. После возврата из callback, LLC сам удаляет пакет, подписчикам этого делать не нужно. Важную информацию помещаем в METADATA.

SYNC protocol

Протокол обрабатывает входящии сигналы синхронизации и генерирует свои собственные. Использует тип передаваемого пакета CALLBACK. После начала передачи заголовка SFD, происходит обратный вызов, который возвращает данные для передачи.

WP protocol

Метод WP_Send(*FChain, port) ожидает на входе FChain([MDATA, METADATA]). Согласно структуре пакета WP протокол добавляет поля WP, MIC и шифрует данные, а также важное поле ETX, значение которого получает из NETIF.
Получается FChain([WP, MDATA, MIC, METADATA]), он передается протоколу маршрутизации.
Обрабатывает входящие пакеты с полем MPDU TYPE равным WP. Обработка входящих пакетов происходит через функцию WP_Receive_HNDL(*FChain, timeSlot). FChain содержит элементы [PPDU_HEADER, WP, MDATA, MIC, METADATA]. Если поле NODE DST совпадает с адресом узла или 0xFF..FF (ШВС), то пакет расшифровывам. Расшифрованый пакет передаем зарегистрированному обработчику через список port bind table. Для экономии памяти список не фиксированной длинны, он создается динамически при регистрации функций. Все остальные пакеты подлежат пересылке, но пересылаются только те пакеты поле ETX, которых больше чем ETX узла. К пример ETX узла 4(4 узла до точки доступа), а мы приняли пакет с ETX = 2. Такой пакет уничтожается, потому что на каждом шаге передачи поле пакета ETX должно уменьшатся для достижения точки доступа.
Получателем данных узла всегда является точка доступа сети. Данные от точки доступа к узлу приходят в виде ШВС.

Trafic balancer

Анализирует поток трафик с помощью вызова TB_Receive_HNDL, производит открытие или закрытие слотов приема данных. Всегда открыт слот номер 0, так как он системный. Информацию об открытых слотах передает Neighbor protocol, который распространяет информацию об узле соседям. Балансер нужен для динамического управления пропускной способностью узла. При открытии слота, он использует информацию о соседях, что бы случайно не выбрать уже используемый канал или временой слот.

Route protocol

Протокол маршрутизации добавляет Fitem PPDU_HEADER. Поле LEN и MPDU_LEN вычисляется из других частей FChain, PVERSION и NETID получат из NETIF. FCS1,2 заполняет нулями эти значения вычислит радиопередатчик налету.
MPDU_TYPE определяется по заголовку первого элемента переданого FChain RT_Send(...).
Далее используя протокол соседей, ищет соседа с минимальным ETX. Из таблицы извлекает MAC адрес соседа, его рабочию частоту и номер тайм слота.
Все эти параметры и FChain передает LLC_AddTask(...)
Если соседей нет, то пакет уничтожается без уведомлений.
Возможна ситуация, что у нас будет несколько соседей с одинаковым ETX, к тому же у соседа может быть открыто несколько входящих подключений. В этом случаи выбор передачи осуществляется случаным образом.

Neighbor protocol

Принимает ШВС сообщения-уведомления от соседей. Ведет список соседних узлов и их параметров. По мимо уведомлений может быть запрос соседей, тогда узел отзывается и посылает ШВС с информацией о себе. Рассылка информации производится через неравные интервалы по таймеру(раз 2-5 минуты). Узел считается устаревшим и его удаляем, если от него не поступало информации в течении 10 минут. Протокол управяет выбором ETX узла, основываясь на таблице. Запрос соседей происходит, при пустом списке, к примеру после обнаружения сети. Точка доступа таким же образом передает ETX=0 и свой ETX не вычисляет на основе соседей.

Connection protocol

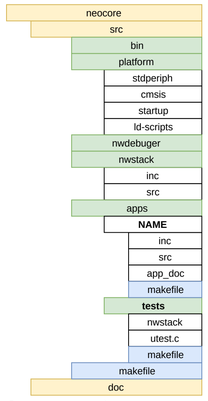
Протокол отправляет точке доступа запрос на подключение с информацией об узле. Отправка запросов происходит периодически, пока нет подтверждения или отказа в подключении. Если было отправленно более 10 запросов и они все безответны, то начинаем поиск сети заново. Больная тема это получения ответа от точки доступа, как передавать пакеты узлам пока окончательно неясно.

IP protocol

Заголовок IP пакета содержит номер протокола, кому предназначен пакет. Передача управления происходит через вызов функции из таблицы Protocol bind table.
IP протокол совмещает в себе функции маршрутизатора. Пакеты проходящии от ТД к узлам передаются на табличную маршрутизация. Пакеты проходящии от узлов к ТД передаются на маршрутизацию СТР.

Route protocol

Нужно сделать две функции отправки сообщения, Первая для маршрутизации до ТД, вторая от ТД к зллам. Непонятно как протокол будет собирать информацию для таблиции маршрутизации,
для этого нужно сохранять информацию о узле, от которого данные были получены.
R3_Send_TR(*FChain, DST ADDR) -



Код расчитан на один тип платформы.
Юнит тесты используют свою функцию main.c (в utest.c).
Юнит тест описывается стандартной структурой данных и содержит вызываемую функцию. Структура данных подготавливается каждым тест-модулем.
Тестирование расчитано на использование отладчика gdb.
Основная функция main распологается в пользовательском приложении, пользователь сам запускает сетевой стек.
Перевод МК в спящий режим так же проблема пользовательского приложения.
Проект имеет один несколько сборочных makefile.
Сборочный файл имеет ключ сбори utest.

Цели сборки makefile:
build
clean
gdb (автоостанов в utest.c или main(), assert)
load (загрузка программы в МК)
utest

Опции сборки:
RELEASE - O2 (по умолчанию)
DEBUG - O0
LOG - разрешает логирование
SET_MAC 0x..... (Использовать предустановленый адрес)
ASSERT

USER_APP - имя приложения для сборки

Организация модулей программы:
Модули используют концепцию ООП. Объект представлен структурой данных и методами работы с структурой. Методы класса реализованны в виде отдельных функций, использующие в качестве аргумента указатель на объект. Такой подход позволяет получить более чистый код и упростит отладку. Обьекты самостоятельно не инициализируют внутри себя вспомогательные обьекты, а получают их от конструктора(к примеру клас mac не создает класс radio. MAC получает указатель на обьект radio и использует его методы). Отсутсвие либо минимальное количество внешних связей обьекта позволят сделать хорошее покрытие кода тестами.

Исключения:
макрос stack_failure(char* msg)
Пишет в LOG сообщение.
Если DEBUG активен, код зацикливаеться, в противном слуаии МК перезап

Отладка/логгер:
Для отладки проекта предназначен макспос ASSERT(condition, message) ес
проект блокирует сове исполнение и производит печать отладочной информ
Вывод данных в serialDebug с использованием printf. LOG_GROUP(message
При активном дефайне LOG поток перенаправляется в serialDebug. Приме
значения переменных. Макрос LOG добавляет имя файла и номер строки. С
примеру LOG_MAC это сообщения уровня стека mac. Каждую группу логов м

Юнит тесты:
Тесты по группам распологаются в файлах с соответсвующими именами и предоставляют метод NAME_testSuit(testSuit* TS, uint8_t size). testSuit содержит перечень указателей на функции-тесты, общее количество тестов size, имена тестов. typedef bool (*utestFN)(char*) указатель на тест-функцию, результатом работы является true/false (наличие ошибки теста) и сообщение с причиной ошибки.
Главная программа включает заголовки всех тестовых модулей и собирет массив, utestRunner используя метод NAME_testSuit(..) извлекает тесты и прогоняет их. Если тест завалился, управление переходит в процедуру testFailed(char* msg). Вывод ошибок осуществляется с помощью serialDebug.

Аппаратная отладка(SIGNAL TRACKER):

ускается.

ли условие истинно, то
ации.
).
ение printf позволяет выводить
GROUP имя группы лога. К
можно включить или выключить.

Основа аппаратной отладки - переключение сигнальных гпио и анализ осци
макрос SIG_SET_NAME, SIG_PULSE_NAME и SIG_CLR_NAME устанавлива
Макрос обеспечивает переключение порта подстановкой команды в код, жел
помощью прямого обращения к регистру. NAME имя контролируемого проце
включать. Пример SIG_SET_SFD_DETECTED , SIG_PULSE_NAME - создает
В файле с макросами SIG_SET_SFD_DETECTED определен
как макрос SIG1_HIGH, который переводит вывод в ВУ. ifdef SIG_SET_SFD_
Для активации отслеживания сигналов достаточно определить SIG_SET_SF

SIGNAL TRACKER, DEBUGER(LOGGER, ASSERT) являются
частью стека протоколов. Все макросы используют слабо связанную
функцию void serialDebugPutChar(char sym).
Тела макросов SIGx_HIGH, SIGx_LOW, SIGx_PULSE, определяются
пользователем в зависимости от платформы
исполнения. Желательно ассемблерная вставка.

лограффом.
ют или сбрасываю линию SIG.
ательно это делать с
сса. Тесты возможно отключать и
короткий импульс

DETECTED SIG1_HIGH elseif ;.
D_DETECTED.

LOG(level,...)	Макрос вывода отладочных сообщений. level - флаги и уровень логирования. Второй аргумент - стандартные аргументы printf
ASSERT_HALT(condition, ...)	Макрос проверки условий. Если условие ложно, производится вывод сообщения ... и остановка работы программы.
ASSERT(condition, ...)	Макрос проверки условий. Если условие ложно, производится вывод сообщения ... , работа программы не прерывается

__attribute__((weak)) void STACK_FAILURE(char* msg) {while(1);}	Функция вызывается при ASSERT_HALT и LOG с флагом M
void nwDebugerInit(void);	Настраивает аппаратуру системы логирования.

Уровни логирования
MSG_INFO MSG_WARNING MSG_ALARM

Флаги логирования
MSG_TRACE MSG_STATE MSG_FRESH MSG_HALT MSG_ON MSG_OFF

SG_HALT