

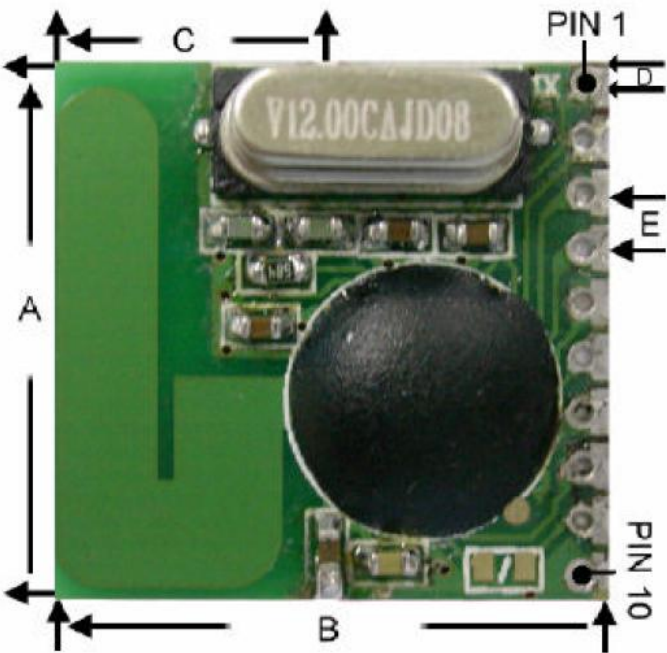
TR24A ISM Band Transceiver

Это не официальная документация.

В этом документе могут содержать ошибки, ошибочные высказывания и предположения. Если вы найдете ошибки то сообщите о них по адресу a9d@mail.ru

Внешний вид трансивера.

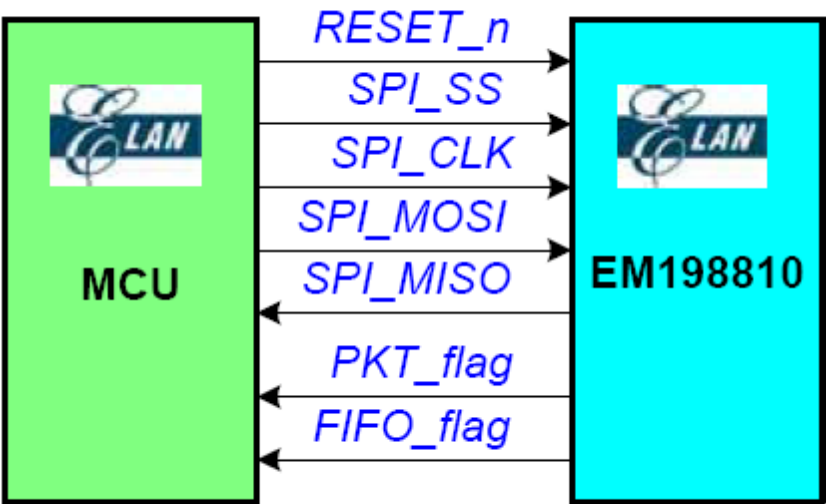
Mechanical Characteristics



TR24A(H) : IC → Chip
TR24A(W) : IC → Package

	PIN	Description
PIN 1	+3.3V	
PIN 2	SPI_MISO	SPI data out
PIN 3	RESET_n	Reset input, active low
PIN 4	SPI_CLK	SPI data in
PIN 5	SPI_MOSI	SPI data in
PIN 6	SPI_SS	SPI slave select input
PIN 7	FIFO_FLAG	FIFO full/empty
PIN 8	PKT_FLAG	Packet TX/RX flag
PIN 9	BRCLK	clk. out. (optional)
PIN 10	GND	

Внешний вид трансивера.



Определение In/Out.



- ↑
1. Automatically set **FIFO write_point=0** when RX received SYNC
 2. Automatically set **FIFO read_point=0** when RX received SYNC or after transmit SYNC when TX

- Figure 1 -

- **Preamble:** 1 ~ 8 bytes programmable
- **SYNC:** 32/48/64 bits programmable as device syncword
- **Trailer:** 4~18 bits programmable
- **Payload:** TX/RX data, there are 4 data types: raw data, 8_10 bits, Manchester, interleave, with FEC option
- **CRC:** 16 bit CRC is option

Note: For transmit, it is required to clear FIFO write point before application writes in data via access reg82[15].

Формат пакета (мак. размер пакета 255 байт, но размер буфера 64 байта).

От себя.

В этой документации я объясню, как работать с этим трансивером. Напишу выводы, к которым я пришел. Также будут написаны мои догадки и предположения.

Для того, что бы освоить этот трансивер мне пришлось провести очень большое количество экспериментов и потратить на это много времени. На его освоение было потрачено около 6-ти месяцев.

Работать с трансивером, руководствуясь только официальной документацией, очень тяжело. В ней присутствует очень много пробелов.

В этом документе под «скоростью» я буду подразумевать пропускную способность. Скорость передачи данных трансивером всегда составляет 1 Мбит/с.

Завершив разработку исходного кода, я смог достичь следующих результатов:

Испытание на скорость.

Условия:

- односторонний пинг;
- расстояние 5 м;
- максимальный разгон трансивера, FEC выкл., Data type –NRZ, скрамблер выкл., CRC – аппаратное ,все остальное по умолчанию;
- размер пакета 64 байта;
- учитывались только полезные данные, без ошибок;

мин.: 190 Кбит/с.

сред.: 412 Кбит/с.

макс.: 508 Кбит/с.

Условия:

- односторонний пинг;
- расстояние 5 м;
- достоверность доставки максимальная, FEC – FEC13, Data type – 8/10 line code, скрамблер вкл., CRC – аппаратное, CRC - программное;
- размер пакета 64 байта;
- учитывались только полезные данные, без ошибок;

мин: 182 Кбит/с.

сред: 186 Кбит/с.

макс: 187 Кбит/с.

Если вы смогли добиться лучшего результата, то напишите как вы этого добились(a9d@mail.ru).

Скорость передачи данных зависит от выбранного канала. На разных каналах разная зашумленность. Уровень постоянно меняется. Поэтому в протоколе имеет смысл реализовать функцию которая будет выбирать наилучший канал для передачи данных.

В теории можно выбирать каналы из диапазона 2400-2482 MHz. По умолчанию выбрана частота 2402+76 MHz. Но в реальности удавалось передавать пакеты на частоте близкой к 2700 MHz. Скорей всего трансивер работает стабильно только на штатных частотах.

Что не было сделано:

- не проводились испытания на дальность;
- оценка зависимости RSSI от расстояния;
- оценка скорости передачи данных на разных дистанциях;
- не были проверены параметры которые определяют мощность сигнала. По ним нет документации;
- не реализована возможность отправки пакетов длина которых превышает 64 байта.

Планирую разработку протокола, а также провести испытания на дальность. НО:

Разработка протокола приостановленна. После двух лет работы с интернет магазином www.kosmodrom.com.ua (г.Харьков) был обманут и не получил от них часть элементов. Эти элементы необходимы для разработки устройства ради которого пишется протокол. Всем кто сможет помочь , буду признателен. 19.08.2010
E-mail: a9d@mail.ru

Для нормальной рабы трансивера необходимо:

- более качественное питание для трансивера. Добился хорошего результата, поставив в цепь питание конденсаторов различной емкости.
- убедиться в том, что шум на линии SPI не мешает передачи данных. Проверяется путем записи в трансивер данных и считывания их. Испытания проводить с отключенным программатором (z-состояние оказывает влияние)!
- скорость передачи данных по SPI не должна превышать 3 МГц.
- во время записи/чтения из регистров трансивера необходимо отключать прерывания.

При разработке кода главным критерием была простота.

Запись в регистр трансивера.

Формат:

1 байт – номер регистра.

2 байта – значение регистра.

Пример:

```
//записать в регистр трансивера значение
void CTr24a::write(unsigned char reg,unsigned int data)
{
    union
    {
        unsigned int buf;
        unsigned char b[2];
    };
    buf=data;

    MySpi.SS_ON(SS);
    MySpi.send(reg);    //регистр
    _delay_us(2);
    MySpi.send(b[1]);   //старшая часть
    _delay_us(2);
    MySpi.send(b[0]);   //младшая часть
    _delay_us(2);
    MySpi.SS_OFF(SS);

} //end writeByte
//=====
```

Чтение из регистра трансивера.

Формат:

1 байт – номер регистра. Старший бит = 1.

2 байта – значение регистра.

Пример:

```
//Чтение из регистра трансивера
unsigned int CTr24a::read(unsigned char reg)
{
    union
    {
        unsigned int buf;
        unsigned char b[2];
    };

    MySpi.SS_ON(SS);
    MySpi.send(reg | 0x80);    //Старший бит определяет операцию
    _delay_us(2);
    b[1]=MySpi.send(0x0FF);
    _delay_us(2);
    b[0]=MySpi.send(0x0FF);
    _delay_us(2);
    MySpi.SS_OFF(SS);

    return buf;
} //end readByte
//=====
```



```

const unsigned char tbl_rfinit[54] = {0x09,0x21,0x01,
                                     0x00,0x35,0x4D,
                                     0x02,0x1F,0x01,
                                     0x04,0xBC,0xF0,
                                     0x05,0x00,0xA1,
                                     0x07,0x12,0x4C,
                                     0x08,0x80,0x00,
                                     0x0C,0x80,0x00,
                                     0x0E,0x16,0x9B,
                                     0x0F,0x90,0xAD,
                                     0x10,0xB0,0x00,
                                     0x13,0xA1,0x14,
                                     0x14,0x81,0x91,
                                     0x16,0x00,0x02,
                                     0x18,0xB1,0x40,
                                     0x19,0xA8,0x0F,
                                     0x1A,0x3F,0x04,
                                     0x1C,0x58,0x00};

union
{
    unsigned int data;      //значение регистра
    unsigned char b[2];
};

channel=76;    //канал по умолчанию
swallow=9;     //делитель частоты по умолчанию
Error.byte=0;  //обнулить все ошибки
ProgCRC=0;     //программное CRC выкл
TrState=0;     //предыдущей режим работы трансивера

reset(); // Если во время первой инициализации произошла ошибка, снижает
         // вероятность ошибки при повторной инициализации

unsigned char i;
for(i=0;i<30;i=i+3) //инициализация кадра
{
    b[1]=tbl_frame[i+1];
    b[0]=tbl_frame[i+2];
    write(tbl_frame[i],data);
}

__delay_ms(5);
for(i=0;i<54;i=i+3) //инициализация передатчика
{
    b[1]=tbl_rfinit[i+1];
    b[0]=tbl_rfinit[i+2];
    write(tbl_rfinit[i],data);
}

//Проверить правильность инициализации трансивера
for(i=0;i<54;i=i+3)
{
    data=read(tbl_rfinit[i]);

    if(b[1]!=tbl_rfinit[i+1])
    {
        Error.bit.Init=1;
    }
    else if(b[0]!=tbl_rfinit[i+2])
    {
        Error.bit.Init=1;
    }
}

for(i=0;i<30;i=i+3)
{

```



```

        data=read(tbl_frame[i]);

        if(b[1]!=tbl_frame[i+1])
        {
            Error.bit.Init=1;
        }
        else if(b[0]!=tbl_frame[i+2])
        {
            Error.bit.Init=1;
        }
    }

} //end init
//=====

```

Флаг FIFO.

Этот флаг сбрасывается перед установкой флага PKT. Поэтому это событие легко пропустить.

В режиме TX. Выставляется в 1 только если буфер пуст. Можно использовать для передачи пакетов длина которых превышает 64 байта. Т.е. сначала записывается кадр длиной 64 байта, далее ожидается выставление флага FIFO. После записываются следующий кадр. И так до конца посылки. Но намного проще записывать в буфер трансивера данные не слишком быстро. Скорость записи данных в буфер зависит от настроек трансивера(FEC, Code type и т.п).

В режиме RX. Выставляется в 1 только если буфер заполнен полностью(если передаются пакеты длина которых не превышает 64 байта, то никогда не выставиться). Имеет значение только если передаются пакеты длина которых превышает 64 байта. При его выставлении необходимо тут же начать процесс считывания данных.

Пример:

```

//получить состояние флага FIFO
// FIFO флаг сбрасывается перед установкой флага PKT!!!!
//RX- 1 буфер полон, выставляется только при передаче пакетов размер которых
превышает 64 байта
//TX- 1 буфер пуст
unsigned char CTr24a::GetFifoFlag()
{
    return !bit_is_clear(PINB,PIN_FIFO);
} //end GetFifoFlag
//=====

```

Флаг РКТ.

Сообщает о том, что пакет получен/передан.

В режиме TX. После входа в режим сбрасывается, выставляется в 1 когда пакет отправлен полностью.

В режиме RX. После входа в режим сбрасывается, выставляется в 1 только если будет получен пакет полностью. Если во время приема произойдет ошибка в Trailer, Preamble, Syncword то пакет не будет принят и флаг не выставится. Этот флаг не стоит опрашивать в бесконечном цикле.

Пример:

```
//получить состояние флага РКТ
//RX- 1 если получен пакет
//TX- 1 если пакет передан
unsigned char CTr24a::GetPktFlag()
{
    return !bit_is_clear(PINB, PIN_PKT);
} //end GetPktFlag
//=====
```

Получить текущий режим работы.

Определить текущий режим работы трансивера можно опросив регистр 64. Биты 15-12 указывают текущий режим работы.

Пример:

```
//получить текущий режим работы трансивера
//на выходе:
// 0 - режим простоя
// 1 - режим передачи
// 2 - режим приема
// 3 - режим сна
// 0xFF - ошибка
unsigned char CTr24a::GetState()
{
    union
    {
        unsigned int buf;
        unsigned char b[2];
    };

    buf= read(0x40); //регистр 64 - состояние трансивера

    switch (b[1]&0xF0)
    {
        case 0xC0: //idle
        {
            return 0;
            break;
        }
        case 0xE0: //TX
        {
            return 1;
            break;
        }
    }
}
```

```

        case 0xD0: //RX
        {
            return 2;
            break;
        }
        case 0x80: //Sleep
        {
            return 3;
            break;
        }
    }

    return 0xFF; //Error
} //end GetState
//=====

```

Получить текущую температуру.

Значение текущей температуры содержится в регистре 3, биты 0-3. Цена деления: 1 бит = 4 градуса Цельсия. Есть подозрение, что некоторые параметры зависят от температуры. Но неизвестно какие и какова зависимость.

Пример:

```

//температура рег.3
//на выходе:
// температура 4ре бита
unsigned char CTr24a::GetTemp()
{
    union
    {
        unsigned int buf;
        unsigned char b[2];
    };

    //получить значение регистра 3
    buf=read(0x03);

    return (b[0]&0x0F);
} //end GetTemp
//=====

```

Получить информацию об ошибках(реализовано программно).

Для получения информации об ошибках было создано битовое поле ERROR.

```

typedef union{
unsigned char byte;
struct{
    unsigned char PKT:1; //пакет еще не принят полностью
    unsigned char State:1; //перед началом приема пакета трансивер
                            не находился в режиме RX
    unsigned char Len:1; //длина пакета не верная
    unsigned char CRC:1; //апаратное CRC не совпало
    unsigned char ProgCRC:1; //программное CRC не совпало
    unsigned char RSSI:1; //значение RSSI не верное
    unsigned char Init:1; //трансивер не инициализирован верно
    unsigned char reserved:1;
} bit;
}ERROR;

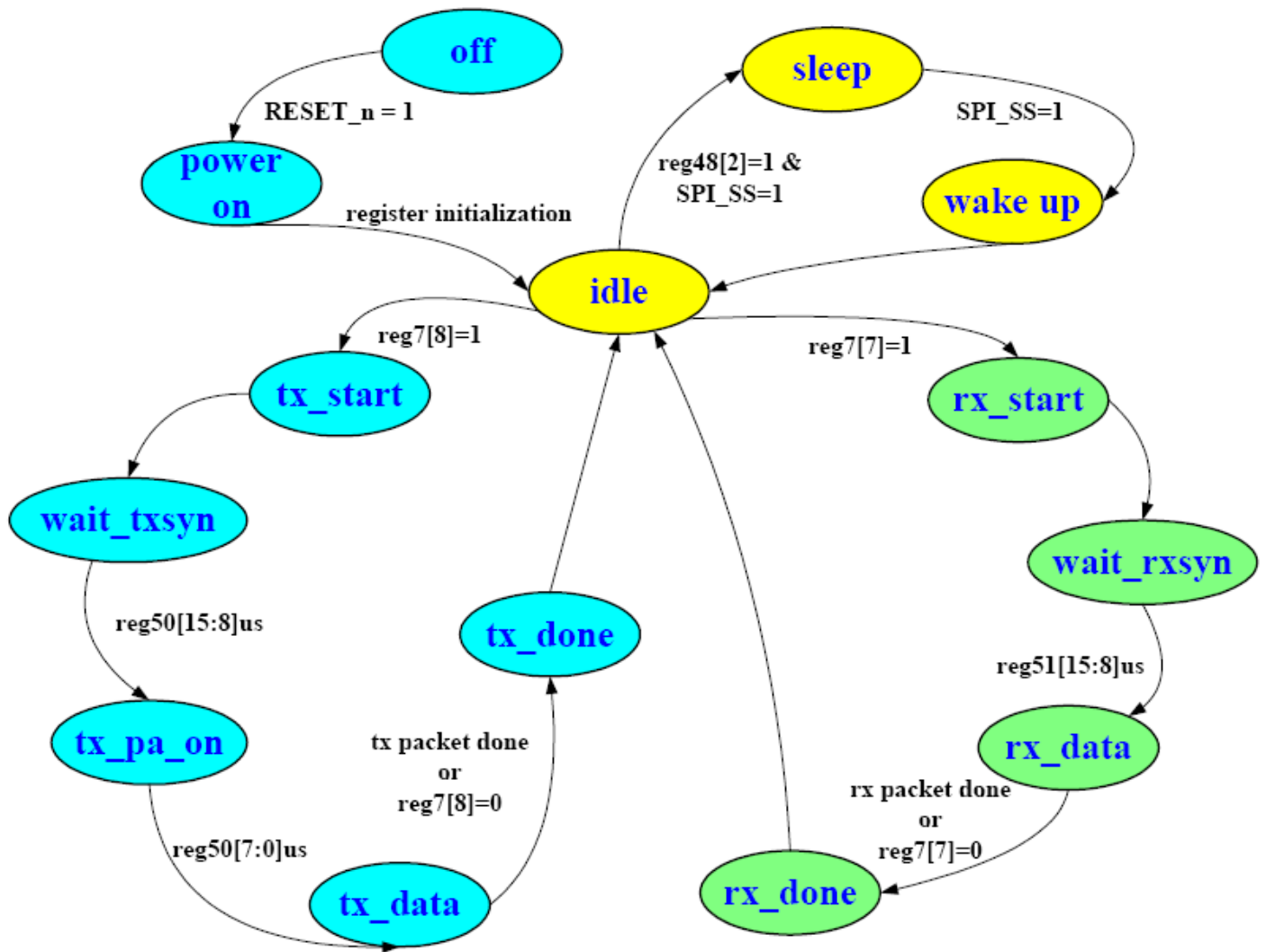
```

Содержимое этого поля можно получить вызвав метод GetLastError().

Пример:

```
//вернуть номер последней ошибки
unsigned char CTr24a::GetLastError()
{
    return Error.byte;
} //end GetLastError
//=====
```

Режимы работы трансивера.



После завершения операции трансивер переходит в режим простоя (Idle). Смена режима работы на Idle сообщает об удачном завершении операции. Энергопотреблением в этом режиме можно управлять через регистр 21, биты 15-13. При минимальном энергопотреблении скорость смены режимов будет более продолжительной.

IDLE

Режим простоя. Находясь в этом режиме можно безопасно менять настройки трансивера.

Пример:

```
//режим простоя
void CTr24a::ModeIdle()
{
    DATA buf;

    buf.data=read(0x07);
    buf.bit.b8=0;
    buf.bit.b7=0;

    write(0x07,buf.data); // переход в режим RX, задаем канал
} //end IdleMode
//=====
```

RX

Режим приема данных. Во время установки этого режима происходит установка делителя частоты кварца и номер используемого канала. Номер канала можно менять динамически, при условии, что делитель частоты выключен. Иначе номер канала будет статическим (В теории, с делителем частоты еще не разобрался полностью).

Находясь в этом режиме можно сбросить указатель на голову. Но это происходит автоматически сразу после перехода в этот режим. Скорей всего эта возможность будет полезной при реализации протокола. Например, в заголовке пакета содержится адрес, после его считывания было установлено, что данные от этого адресата получать нельзя. Тогда чтобы не тратить время на считывания всего пакета можно сбросить указатель.

Сброс осуществляется путем установки 7-го бита регистра 82.

Имеет смысл проверить режим работы после перехода. Иногда(скорей всего в результате шума) смена режима не происходит.

После успешного приема пакета состояние меняется на Idle.

Пример:

```
//перейти в режим приема данных
void CTr24a::ModeRecive()
{
    DATA buf;

    TrState=1; //переходим в режим RX, необходимыми для функции приема пакета

    buf.data=read(0x07);
    buf.byte[1]=(swallow<<1);
    buf.byte[0]=chanel;
```

```

buf.bit.b8=0;
buf.bit.b7=1;

write(0x07,buf.data); // переход в режим RX, задаем канал

__delay_us(10);
} //end ReciveMode
//=====

```

Указание канала. Пример:

```

//установить канал
//на входе:
// ch - канал (7 бит)
void CTr24a::SetChanel(unsigned char ch)
{
    chanel=ch&0x7F;
} //end SetChanel
//=====

```

Получить номер используемого канала. Пример:

```

//вернуть текущий канал
unsigned char CTr24a::GetChanel()
{
    return chanel;
} //end GetChanel
//=====

```

TX

Режим отправки данных. После перехода в этот режим не обходи сразу начать заполнять буфер данными.

Во время установки этого режима происходит установка делителя частоты кварца и номер используемого канала. Номер канала можно менять динамически, при условии, что делитель частоты выключен. Иначе номер канала будет статическим (В теории, с делителем частоты еще не разобрался полностью).

Также после перехода и перед заполнением буфера данными необходимо сбросить указатель на голову буфера. Если этого не сделать, то пакеты не будут верно приниматься. Так и не понял зачем это делать, ведь после передачи пакета указатель сбрасывается сам. Возможно кроме сброса осуществляются еще какие-то действия. Или это предосторожность на тот случай когда процесс отправки пакета был прерван.

Сброс осуществляется путем установки 15-го бита регистра 82.

Пример:

```

//перейти в режим передачи данных
void CTr24a::ModeTransmit()
{
    DATA buf;

```

```

    buf.data=read(0x07);
    buf.byte[1]=(swallow<<1);
    buf.byte[0]=chanel;

    buf.bit.b8=1;
    buf.bit.b7=0;

    write(0x07,buf.data); // переход в режим RX, задаем канал

} //end TransmitMode
//=====

```

SLEEP

Режим сна. Режим наименьшего энергопотребления. Выход из этого режима происходит по активному уровню на выводе SS.

Пример:

```

//управление режимом сна (Регистр 10 бит 15 управляет режимом сна в режиме
тестирования)
//на входе:
//mode
// 0-усыпить
// 1-пробудить
void CTr24a::ModeSleep(unsigned char mode)
{
    DATA buf;

    if(mode==0)
    {
        buf.data=read(48);
        buf.bit.b2=1;
        write(48,buf.data);
    }
    else
    {
        MySpi.SS_ON(SS);
        __delay_ms(2);
        MySpi.SS_OFF(SS);
    }
} //end ModeSleep
//=====

```

Сброс указателя на голову буфера.

Сбросить указатель на голову, соответствующего режима, можно только находясь в этом режиме.

Пример:

```

//Очистить буффер, МЕШАЕТ НОРМАЛЬНОЙ РАБОТЕ ТРАНСИВЕРА
void CTr24a::ClearFIFO()
{
    switch (GetState())
    {

```

```

    case 1:          //TX
    {
        write(0x52,0x8000); //очистить буффер передатчика TX
        break;
    }
    case 2:          //RX
    {
        write(0x52,0x0080); //очистить буффер приемника RX
        break;
    }
}

}

} //end ClearFIFO
//=====

```

Прием пакета.

Это самая сложная процедура. Перед ее началом необходимо перейти в режим RX. После выдержать паузу или по какому либо признаку определить факт получения пакета.

Признаки получения пакета:

- смена режима работы на Idle;
- флаг FIFO активен, если идет процесс передачи больших пакетов;
- флаг РКТ активен;
- получено слово синхронизации (Рег. 64 бит 10), сбрасывается перед выставлением флага РКТ;
- указатель на голову изменился.

Для того чтобы удостовериться точно в факте получения пакета происходит проверка текущего состояния. Если состояние равно Idle, то происходит проверка предыдущего состояния трансивера. В случае не выполнения этих условий процесс получения пакета будет прерван.

На флаг РКТ ориентироваться не стоит. Если после отправки пакета произошел сбой и трансивер не перешел в режим работы RX, то факт получения пакета будет принят ошибочно.

Первый байт в пакете содержит размер пришедшего пакета.

После считывания пакета из буфера происходит аппаратная проверка CRC. Если используется FEC(система исправления ошибок), то аппаратное CRC сообщает о том, что произошла ошибка которая возможно не была исправлена. В этом случае необходимо произвести программную проверку CRC.

После проверки CRC происходит получение значения RSSI.

Пример:

```
//принять пакет, ускоренный
//на входе:
// data- буффер
//на выходе:
// длина пакета
unsigned char CTr24a::PkgRead(unsigned char *data)
{
    unsigned char len;    //длина пакета
    unsigned char i,j;
    unsigned char state;
    unsigned int crc;

    union
    {
        unsigned int crc;    //значение регистра
        unsigned char b[2];
    }InCRC;

    Error.bit.PKT=0;
    Error.bit.State=0;
    Error.bit.Len=0;
    Error.bit.ProgCRC=0;

    //можно использовать другие способы определения приема пакета.(Флаг
    PTR,Register 64[10]). Но они имеют ряд недостатков
    state=GetState();
    if(state>0) //если текущее состояние не Idle, то пакет еще не был принят
                полностью
    {
        Error.bit.PKT=1;
        return 0;
    }

    if(TrState==0) //проверить режим работы трансивера в котором он находился
    {
        Error.bit.State=1;
        return 0;
    }

    TrState=0; //сбрасываем предыдущее состояние

    MySpi.SS_ON(SS);
    MySpi.send(0x50|(1<<7));    //reg80
    _delay_us(1);
    len=MySpi.send(0xFF);

    if(len>63) //длина пакета не корректная
    {
        Error.bit.Len=1;
        return 0;
    }
    else if (len<3)    //если не используется программная проверка CRC то можно
                        поставить значение 1
    {
        Error.bit.Len=1;
        return 0;
    }

    for(i=0;i<len;i++) //получить пакет
    {
        _delay_us(1);
        data[i] = MySpi.send(0xFF);
    }

    MySpi.SS_OFF(SS);
```

```

    _delay_us(1);

    //проверка CRC
    CheckCRC();
    if(Error.bit.CRC==1)
    {
        if(ProgCRC==1) //проверка программного CRC
        {
            InCRC.b[0]=data[len-1];
            InCRC.b[1]=data[len-2];

            crc = 0xFFFF;
            for(int j=0;j<len;j++)
            {
                crc^=data[j]<<8;
                for( i = 0; i < 8; i++ )
                    crc = crc & 0x8000 ? ( crc << 1 ) ^ 0x1021 : crc << 1;
            }

            if(InCRC.crc!=crc)
            {
                Error.bit.ProgCRC=1;
            }
        }
    }

    //получить RSSI
    SetRSSI();

    return len;
} //end ReadPkg
//=====

```

Получить значение RSSI. Пример:

```

//вернуть мощность сигнала
unsigned char CTr24a::GetRSSI()
{
    return RSSI;
} //end GetRSSI
//=====

```

Отправка пакета.

Перед отправкой пакета, если используется программная проверка CRC, то в конце пакета будет размешено значение контрольной суммы(2-ва байта).

После необходимо перевести контроллер в режим TX , сбросить указатель на голову и начать заполнять буфер данными.

Пример:

```

//отправить пакет данных
//на входе:
// len - длина пакета
// data - массив данных
void CTr24a::PkgSend(unsigned char len,unsigned char *data)
{
    if(ProgCRC==1)
    {
        AddCRC2pkg(len,data);
    }
}

```

```

    }

    ModeTransmit();
    write(0x52,0x8000); //сбросить указатель на голову TX.
                        //Необходимо для передачи пакетов размер которых меньше
                        //64 байтов.
                        //Так и непонял зачем. При переходе в режим TX
                        //указатель сбрасывает.
                        //Но если не сделать сброс через рег. 52, то пакеты
                        //будут отправляться битые.

    MySpi.SS_ON(SS);
    __delay_us(1);
    MySpi.send(0x50); //reg80
    __delay_us(1);
    MySpi.send(len); //длина пакета
    unsigned char i;
    for(i=0;i<len;i++) //отправить пакет
    {
        __delay_us(1);
        MySpi.send(data[i]);
    }

    MySpi.SS_OFF(SS);
} //end SendPkg
//=====

```

Управление скремблером.

Трансивер поддерживает аппаратное шифрование. Но ключ шифрования 7-ми битный, скорей всего надежность ключа очень низкая. Включенное шифрование влияет на скорость передачи данных.

Пример:

```

//вкл/выкл шифрование рег. 57
//на входе:
// state:
// 0-отключить шифрование
// 1-включить шифрование
void CTr24a::ControlScramble(unsigned char state)
{
    DATA buf;

    buf.data=read(0x39);

    if(state==0) //отключить шифрование
    {
        buf.bit.b14=0;
    }
    else
    {
        buf.bit.b14=1;
    }

    write(0x39,buf.data);

} //end ControlScramble
//=====

```

Указание ключа. Пример:

```

//установить ключ шифрования рег.51

```

```

//на входе:
// key - 7ми битный ключ
// 0- данные не шифруются
void CTr24a::SetKey(unsigned char key)
{

    union
    {
        unsigned int buf;
        unsigned char b[2];
    };

    //получить значение регистра 51
    buf=read(0x33);

    b[0] &= (1<<7);
    b[0] |=key;

    write(0x33,buf);
} //end SetKey
//=====

```

Система исправления ошибок.

В трансивере реализована аппаратная система FEC. Эта система повышает вероятность доставки пакета, но сильно уменьшает скорость передачи данных. К примеру, использование FEC13 увеличивает количество служебной информации на 66%.

Пример:

```

//вкл/выкл алгоритм исправления ошибок рег. 48
//на входе:
// state
// 0 - No FEC
// 1 - FEC13 (1/3 т.е. 1 часть полезных данных, 2 части служебных данных)
// 2 - FEC23 (2/3 2 части полезных данных, 1 часть служебных данных)
void CTr24a::ControlFEC(unsigned char state)
{
    DATA buf;

    //получить значение регистра 48
    buf.data=read(0x30);

    switch (state)
    {
        case 1: //FEC13
        {
            buf.bit.b5=0;
            buf.bit.b4=1;
            break;
        }
        case 2: //FEC23
        {
            buf.bit.b5=1;
            buf.bit.b4=0;
            break;
        }
        default: //No FEC
        {
            buf.bit.b5=0;
            buf.bit.b4=0;
        }
    }
}

```

```

        break;
    }
}

write(0x30,buf.data);
} //end FECControl
//=====

```

Управление делителем частоты кварца.

Используя делитель частоты можно повысить вероятность доставки пакета. Но мне не удалось полностью определить как с ним работать. По умолчанию выключен.

Пример:

```

// вкл/выкл делитель частоты кварца (влияет на прием/передачу)
// на входе:
// 0- выкл
// 1- вкл
void CTr24a::ControlSwallow(unsigned char state)
{
    DATA buf;

    buf.data=read(21);

    if(state==1)
    {
        buf.bit.b12=1;
    }
    else
    {
        buf.bit.b12=0;
    }

    write(21,buf.data);
} //end ControlSwallow
//=====

```

Установка делителя. Пример:

```

//установить делитель (по умолчанию 9)
//на входе:
// div - делитель (4ре бита)
void CTr24a::SetSwallow(unsigned char div)
{
    swallow=div&0x0F;
} //end SetSwallow
//=====

```

Определение типа кодирования данных.

Тип кодирования данных влияет на вероятность доставки и на скорость передачи данных. Но в отличии от FEC влияние на скорость значительно меньше.

Пример:

```

//определить алгорит кодирования сигнала рег.48
//на входе:
// type
// 0 - NRZ law data
// 1 - Manchester data type
// 2 - 8/10 line code (лучший результат)
// 3 - interleave date type

```

```

void CTr24a::ControlDataType(unsigned char type)
{
    DATA buf;

    //получить значение регистра 48
    buf.data=read(0x30);

    switch (type)
    {
        case 1:      //Manchester data type
        {
            buf.bit.b7=0;
            buf.bit.b6=1;
            break;
        }
        case 2:      //8/10 line code
        {
            buf.bit.b7=1;
            buf.bit.b6=0;

            break;
        }
        case 3:      //interleave date type
        {
            buf.bit.b7=1;
            buf.bit.b6=1;
            break;
        }
        default:      //NRZ law data
        {
            buf.bit.b7=0;
            buf.bit.b6=0;
            break;
        }
    }

    write(0x30,buf.data);

}

} //end ControlDataType
//=====

```

Аппаратная поддержка CRC.

В трансивере реализована аппаратная поддержка CRC. Но если используется FEC, то ошибка аппаратного CRC сообщает о том, что произошла ошибка которая возможно не была исправлена. В этом случае имеет смысл использовать программную проверку CRC.

Пример:

```

//вкл/выкл аппаратную поддержку CRC
//на входе:
// state
// 0- ВЫКЛ CRC
// 1- ВКЛ CRC
void CTr24a::ControlCRC(unsigned char state)
{
    DATA buf;

    buf.data=read(57);

    if(state==1)
    {

```

```

        buf.bit.b15=1;
    }
    else
    {
        buf.bit.b15=0;
    }

    write(57,buf.data);
} //end ControlCRC
//=====

```

Проверка CRC. Пример:

```

//проверить контрольную сумму
//на выходе:
//1- есть ошибка
//0- нет ошибки
//, если используется FEC, 1 сообщает о том, что ВОЗМОЖНО ошибка не была
    исправлена
// необходима программная проверка CRC
void CTr24a::CheckCRC()
{
    DATA buf;
    Error.bit.CRC=0;

    buf.data= read(0x40); //регистр 64 - состояние трансивера

    Error.bit.CRC=buf.bit.b11;
} //end CheckCRC
//=====

```

Задание начального значения CRC. Пример:

```

//начальное значение (в даташите написано, что трансивер использует CRC16.Но
    начальное значение задается почему-то 8-ми битное)
//на входе:
// initData - начальное значение CRC
void CTr24a::SetCRCInitData(unsigned char initData)
{
    DATA buf;

    buf.data=read(57);

    buf.byte[0]=initData;

    write(57,buf.data);
} //end SetCRCInitData
//=====

```

Программная реализация CRC.

В случае если используется FEC, имеет смысл использовать программную проверку CRC. Программное CRC занимает два последних байта в посылке.

Пример:

```

//вкл/выкл программную поддержку CRC (занимает 2 последних байта, размер полезных
    данных уменьшается на 2 байта)
//на входе:
// state
// 0-выкл
// 1-вкл
void CTr24a::ControlProgramCRC(unsigned char state)

```

```

{
    if(state==0)
        ProgCRC=0;
    else
        ProgCRC=1;

} //end ControlProgramCRC
//=====

```

Добавление CRC в конец пакета. Пример:

```

//расчитать и добавить CRC в конец пакета
//на входе:
// len-длина пакета
// data-пакет (последние два байта CRC)
void CTr24a::AddCRC2pkg(unsigned char len,unsigned char *data)
{
    unsigned int crc = 0xFFFF;
    unsigned char i,j;

    for(j=0;j<len-2;j++)
    {
        crc^=data[j]<<8;
        for( i = 0; i < 8; i++ )
            crc = crc & 0x8000 ? ( crc << 1 ) ^ 0x1021 : crc << 1;
    }

    data[len-2]=(crc>>8);
    data[len-1]=(char)crc;
} //end AddCRC2pkg
//=====

```

Установить длину преамбулы.

Преамбула отвечает за побитную и побайтную синхронизацию. Размер влияет на достоверность доставки и скорость.

Пример:

```

//установить длину приамбулы
//на входк:
// len - от 1 до 8 байт
void CTr24a::SetPreambleLen(unsigned char len)
{
    DATA buf;

    buf.data=read(48);

    switch (len)
    {
        case 1:
        {
            buf.bit.b13=0;
            buf.bit.b14=0;
            buf.bit.b15=0;
            break;
        }
        case 2:
        {
            buf.bit.b13=1;
            buf.bit.b14=0;
            buf.bit.b15=0;
            break;
        }
        case 3:
        {
            buf.bit.b13=0;

```



```

buf.data=read(48);

switch (len)
{
    case 1:                                //16
    {
        buf.bit.b11=0;
        buf.bit.b12=0;
        break;
    }
    case 2:                                //32
    {
        buf.bit.b11=1;
        buf.bit.b12=0;
        break;
    }
    case 3:                                //48
    {
        buf.bit.b11=0;
        buf.bit.b12=1;
        break;
    }
    default:                                //64
    {
        buf.bit.b11=1;
        buf.bit.b12=1;
        break;
    }
}

write(48,buf.data);

unsigned char i;
for(i=0;i<len;i++)
{
    buf.byte[1]=tbl_frame[i+1];
    buf.byte[0]=tbl_frame[i+2];
    write(tbl_frame[i],buf.data);
}

} //end SetSyncwordLen
//=====

```

Установка длины трейлера.

Зачем он необходим точно не знаю. Но скорей всего он реализует задержку. Размер влияет на достоверность доставки и на скорость передачи.

Пример:

```

//установить длину Trailer
//на входе:
// len - длина
void CTr24a::SetTrailerLen(unsigned char len)
{
    DATA buf;

    buf.data=read(48);

    switch (len)
    {
        case 1:                                //4 бита
        {
            buf.bit.b8=0;

```

```

        buf.bit.b9=0;
        buf.bit.b10=0;
        break;
    }
    case 2: //6
    {
        buf.bit.b8=1;
        buf.bit.b9=0;
        buf.bit.b10=0;
        break;
    }
    case 3: //8
    {
        buf.bit.b8=0;
        buf.bit.b9=1;
        buf.bit.b10=0;
        break;
    }
    case 4: //10
    {
        buf.bit.b8=1;
        buf.bit.b9=1;
        buf.bit.b10=0;
        break;
    }
    case 5: //12
    {
        buf.bit.b8=0;
        buf.bit.b9=0;
        buf.bit.b10=1;
        break;
    }
    case 6: //14
    {
        buf.bit.b8=1;
        buf.bit.b9=0;
        buf.bit.b10=1;
        break;
    }
    case 7: //16
    {
        buf.bit.b8=0;
        buf.bit.b9=1;
        buf.bit.b10=1;
        break;
    }
    default: //18
    {
        buf.bit.b8=1;
        buf.bit.b9=1;
        buf.bit.b10=1;
        break;
    }
}

    write(48,buf.data);
} //end SetTrailerLen
//=====

```

Получение значения RSSI.

С помощью него можно определить мощность входящего сигнала, а также примерное расстояние до передатчика. Значение сбрасывается в ноль после перехода в режим RX и определяется только после получения пакета.

Пример:

```
//Получить значение RSSI от трансивера
void CTr24a::SetRSSI()
{
    DATA buf;
    Error.bit.RSSI=0;

    buf.data= read(0x06); //регистр 6

    if(buf.bit.b8==1)
    {
        //    MyUart.sendByte('R');
        Error.bit.RSSI=1;
    }

    RSSI=buf.byte[0];
} //end GetRSSI
//=====
```

Конец.

Описал самое основное и необходимое. Трансивер имеет еще множество нюансов, но они не представляют особой важности. Множество функций трансивера мною не были изучены. По ним отсутствует, какая либо, документация.