Вступление.

Данный документ был создан по мотивам борьбы с микросхемой CC2500 производства Chipcon AS, Gaustadalléen 21, NO-0349 Oslo, NORWAY http://www.chipcon.com/.

Все нижеприведенное далеко не бесспорно и является по сути отражением личного опыта автора в процессе освоения данной микросхемы. Все данные представлены в виде «как есть», вы можете их использовать бесплатно, на свой страх и риск, автор не несет ответственности за возможный ущерб.

Встречающиеся листинги программ созданы с использованием AVR Studio 4.11 Atmel Corporation http://www.atmel.com/ и предназначены для микроконтроллеров архитектуры AVR.

Микросхема трансивера CC2500 предназначена для работы в диапазоне 2400-2483,5 МГц. Поставляется упакованной в корпус QLP-20 и занимает при монтаже на плате всего 4x4 мм, при высоте 0.8 мм. Заявленная чувствительность приемного тракта -101 dbm, при скорости передачи данных 10 кБит/с и 1% ошибок. Максимальная выходная мощность +1 dbm (0 dbm – 1mW). Средний потребляемый ток в режиме приема: 15 мА, в режиме передачи при максимальной мощности: 22 мА. Микросхема имеет отличный аппаратный модем, два буфера FIFO по 64 байта (прием и передача), позволяет работать в пакетном режиме и в режиме прозрачной передачи данных. Поддерживаются следующие виды модуляции: амплитудная (on-off key - OOK), частотная (2-FSK), фазовая (QPSK она же MSK по терминологии Chipcon). Для связи с контроллером имеется интерфейс SPI. Кроме того в состав микросхемы входит: низкочастотный RC генератор, необходимый для реализации функции WOR (Wake-on-radio) – выход из режима сна при обнаружении передачи; температурный сенсор, который можно использовать для компенсации температурного дрейфа частоты. Поддерживаются технологии обнаружения несущей, определения занятости канала и другие. Синтезатор микросхемы требует для своей работы кварцевый резонатор 26-27 МГц. На мой взгляд данная микросхема имеет весьма привлекательное соотношение параметров цена/эффективность.

С точки зрения программирования CC2500 представляет собой набор регистров, доступных для чтения и записи через интерфейс SPI. Для связи с контроллером необходимы всего 4 линии: MOSI, MISO, SCK, CS. Однако для комфортной работы желательно использовать еще 2 линии – для анализа GDO0, GDO2. Впрочем состояние этих линий можно читать программно в регистре PKTSTATUS. Всего в трансивере содержится 47 конфигурационных регистра, 14 строб-регистров, 12 регистров состояния, 1 регистр доступа к таблице мощности, 1 регистр доступа к буферам приема-передачи. Для расчета параметров конфигурации можно воспользоваться даташитом, но лучше это сделать с помощью программы SmartRF® Studio, которую бесплатно можно загрузить с сайта компании Chipcon. При работе по SPI можно использовать аппаратный модуль контроллеров AVR (или других), однако с точки зрения облегчения разводки печатной платы более эффективно использовать программную реализацию протокола. При этом надо учитывать, что передача ведется старшим битом вперед. Ниже приведена подпрограмма чтения-записи байта в реализации для контроллера ATmega48 работающего от внутреннего генератора на частоте 1МГц. Не претендую на оптимальность, возможно у вас получится лучше, однако это точно работает, проверено.

```
.def    spi_out=r0      ;байт SPI выходной
.def    spi_out1=r1     ;байт SPI выходной второй
.def    spi_in  =r2     ;байт SPI входной
.def    status  =r7     ;status from CC2500

.def    dly1    =r16    ;для формирования задержек
.def    dly2    =r17    ; для формирования задержек
.def    temp    =r18    ;временный регистр

.equ    gd0=0           ;вывод GDO0
.equ    gd2=1           ; вывод GDO2
.equ    cs=2
.equ    mosi=3
.equ    miso=4
```

```
.equ    sck=5           ;биты порта B ATmega48
.dseg
buf_out: .byte 64       ;резервируем буфер для принятого пакета
.cseg
```
Макроопределения:

```
.macro cs_active
        cbi     PORTB,cs
.endm
;---------------------------------------------------------------------
.macro cs_inactive
        sbi     PORTB,cs
.endm
;---------------------------------------------------------------------
.macro sck_hi
        sbi     PORTB,sck
.endm
;---------------------------------------------------------------------
.macro sck_lo
        cbi     PORTB,sck
.endm
;---------------------------------------------------------------------
.macro mosi_hi
        sbi     PORTB,mosi
.endm
;---------------------------------------------------------------------
.macro mosi_lo
        cbi     PORTB,mosi
.endm

;Подпрограмма инициализации SPI
init_spi:
        cs_inactive             ;CS=1
        sbi     DDRB,cs         ;make it an output
        sck_lo                  ;SCK=0
        sbi     DDRB,sck        ;make it an output
        mosi_lo                 ;MOSI=0
        sbi     DDRB,mosi       ;make it an output
        cbi     DDRB,miso       ;MISO-input
        ret
```

Задержки:

```
;---------------------------------------------------------------------
;Подпрограмма задержки на 1ms при Fкварца=1МГц
delay_1ms:
        rcall delay_250us
        rcall delay_250us
        rcall delay_250us
        rcall delay_250us
        ret
;---------------------------------------------------------------------
;подрограмма задержки на 250 мкс Fкварца=1МГц
delay_250us:
        ldi dly1,81
lp250us:        dec dly1
        brne lp250us
```

```
        ret
;-----------------------------------------------------------------------
;подрограмма задержки на 40 мкс Fкварца=1МГц
delay_40us:
        ldi dly1,11
lp40us:dec dly1
        brne lp40us
        ret
```

Подпрограмма приема-передачи байта через SPI

```
;Передаем из spi_out, принимаем в spi_in
rw_spi:
        push temp
        in temp,SREG
        push temp
        ;
        ldi     temp,8          ;счетчик бит
        clr spi_in
spi_loop:
        lsl     spi_out         ;задвинем старший бит в перенос
        brcc put_0              ;если в переносе 0 - перейти
        mosi_hi
        nop
        rjmp r_bit
put_0:
        mosi_lo
        nop
r_bit:
        sck_hi                  ;выдали строб и подождали
        nop
        sbic PINB,miso          ;читаем бит с miso
        rjmp r1_bit
        clc                     ;если 0 - сбросим перенос
        rjmp rend_bit
r1_bit:
        sec                     ;если 1 - установим перенос
rend_bit:
        rol spi_in              ;задвинем перенос во входной байт
        sck_lo                  ;выдали строб и подождали
        nop
        dec     temp
        brne    spi_loop
        ;
        pop temp
        out SREG,temp
        pop temp
        ret
```

Все пересылки между контроллером и CC2500 должны начинаться после того как трансивер будет готов. Сначала необходимо выставить низкий уровень на CS, затем дождаться низкого уровня на выводе MISO и только затем начинать обмен. Невыполнение этого требования приводит к труднообнаружимым глюкам. Все пересылки начинаются с заголовочного (адресного) байта, который содержит бит чтении/записи, бит burst-доступа и 6-ти битный адрес. Во время передачи заголовочного (адресного) байта контроллер принимает байт статуса CC2500. Вслед за адресным могут следовать

байты данных. Если адресный байт это строб – то байт данных отсутствует. Если установлен бит burst-доступа, то байтов данных может быть несколько. Если вам необходимо прочесть данные из CC2500, вслед за адресным байтом должен следовать какой-то байт данных (например 0). Ниже приведены несколько подпрограмм для обмена данными.

Запись строб-команды, команда в spi_out, статус в spi_in

```
write_strob:
        cs_active
ws_loop:
        sbic PINB,miso      ;ждем готовность
        rjmp ws_loop
        rcall rw_spi        ;пишем строб, читаем статус
        mov status,spi_in
        cs_inactive
        ret
```

```
;Запись регистра: адрес в spi_out, данные в spi_out1, статус в spi_in
write_reg:
        cs_active
wr_loop:
        sbic PINB,miso      ;ждем готовность
        rjmp wr_loop
        rcall rw_spi        ;пишем адрес
        mov spi_out,spi_out1
        rcall rw_spi        ;пишем значение, читаем статус
        mov status,spi_in
        cs_inactive
        ret
```

```
;Чтение регистра, адрес в spi_out, результат в spi_in
read_reg:
        ldi temp,0b10000000
        or spi_out,temp     ;выставляем бит чтения
        cs_active
rr_loop:
        sbic PINB,miso
        rjmp rr_loop
        rcall rw_spi        ;пишем адрес с битом чтения
        mov status,spi_in
        clr spi_out
        rcall rw_spi        ;читаем регистр
        cs_inactive
        ret
```

Ниже приведены определения регистров CC2500:

```
;CC2500 STROBE, CONTROL AND STATUS REGSITER
.equ    IOCFG2      =0x00 ;GDO2 output pin configuration
.equ    IOCFG1      =0x01 ;GDO1 output pin configuration
.equ    IOCFG0      =0x02 ;GDO0 output pin configuration
.equ    FIFOTHR     =0x03 ;RX FIFO and TX FIFO thresholds
.equ    SYNC1       =0x04 ;Sync word, high byte
.equ    SYNC0       =0x05 ;Sync word, low byte
.equ    PKTLEN      =0x06 ;Packet length
.equ    PKTCTRL1=0x07     ;Packet automation control
```

```
.equ    PKTCTRL0=0x08     ;Packet automation control
.equ    ADDR=0x09 ;Device address
.equ    CHANNR    =0x0A ;Channel number
.equ    FSCTRL1   =0x0B ;Frequency synthesizer control
.equ    FSCTRL0   =0x0C ;Frequency synthesizer control
.equ    FREQ2     =0x0D ;Frequency control word, high byte
.equ    FREQ1     =0x0E ;Frequency control word, middle byte
.equ    FREQ0     =0x0F ;Frequency control word, low byte
.equ    MDMCFG4   =0x10 ;Modem configuration
.equ    MDMCFG3   =0x11 ;Modem configuration
.equ    MDMCFG2   =0x12 ;Modem configuration
.equ    MDMCFG1   =0x13 ;Modem configuration
.equ    MDMCFG0   =0x14 ;Modem configuration
.equ    DEVIATN   =0x15 ;Modem deviation setting
.equ    MCSM2     =0x16 ;Main Radio Control State Machine configuration
.equ    MCSM1     =0x17 ;Main Radio Control State Machine configuration
.equ    MCSM0     =0x18 ;Main Radio Control State Machine configuration
.equ    FOCCFG    =0x19 ;Frequency Offset Compensation configuration
.equ    BSCFG     =0x1A ;Bit Synchronization configuration
.equ    AGCCTRL2=0x1B     ;AGC control
.equ    AGCCTRL1=0x1C     ;AGC control
.equ    AGCCTRL0=0x1D     ;AGC control
.equ    WOREVT1   =0x1E ;High byte Event 0 timeout
.equ    WOREVT0   =0x1F ;Low byte Event 0 timeout
.equ    WORCTRL   =0x20 ;Wake On Radio control
.equ    FREND1    =0x21 ;Front end RX configuration
.equ    FREND0    =0x22 ;Front end TX configuration
.equ    FSCAL3    =0x23 ;Frequency synthesizer calibration
.equ    FSCAL2    =0x24 ;Frequency synthesizer calibration
.equ    FSCAL1    =0x25 ;Frequency synthesizer calibration
.equ    FSCAL0    =0x26 ;Frequency synthesizer calibration
.equ    RCCTRL1   =0x27 ;RC oscillator configuration
.equ    RCCTRL0   =0x28 ;RC oscillator configuration
.equ    FSTEST    =0x29 ;Frequency synthesizer calibration control
.equ    PTEST=0x2A ;Production test
.equ    AGCTEST   =0x2B ;AGC test
.equ    TEST2=0x2C ;Various test settings
.equ    TEST1=0x2D ;Various test settings
.equ    TEST0=0x2E ;Various test settings

;Strobe commands
.equ    SRES=0x30         ;Reset chip.
.equ    SFSTXON=0x31      ;Enable and calibrate frequency synthesizer (if MCSM0.FS_AUTOCAL=1).
                          ;If in RX/TX: Go to a wait state where only the synthesizer is
                          ;running (for quick RX / TX turnaround).
.equ    SXOFF=0x32        ;Turn off crystal oscillator.
.equ    SCAL=0x33         ;Calibrate frequency synthesizer and turn it off (enables quick start).
.equ    SRX=0x34          ;Enable RX. Perform calibration first if coming from IDLE and
                          ;MCSM0.FS_AUTOCAL=1.
.equ    STX=0x35          ;In IDLE state: Enable TX. Perform calibration first if
                          ;MCSM0.FS_AUTOCAL=1. If in RX state and CCA is enabled:
                          ;Only go to TX if channel is clear.
.equ    SIDLE=0x36        ;Exit RX / TX, turn off frequency synthesizer and exit
                          ;Wake-On-Radio mode if applicable.
.equ    SAFC  =0x37       ;Perform AFC adjustment of the frequency synthesizer
.equ    SWOR=0x38         ;Start automatic RX polling sequence (Wake-on-Radio)
```

```
.equ    SPWD=0x39           ;Enter power down mode when CSn goes high.
.equ    SFRX =0x3A          ;Flush the RX FIFO buffer.
.equ    SFTX =0x3B          ;Flush the TX FIFO buffer.
.equ    SWORRST=0x3C        ;Reset real time clock.
.equ    SNOP =0x3D          ;No operation. May be used to pad strobe commands to two
                            ;bytes for simpler software.
;Status Registers
.equ    PARTNUM=0xF0        ;Part number for CC2500
.equ    VERSION=0xF1        ;Current version number
.equ    FREQEST=0xF2        ;Frequency Offset Estimate
.equ    LQI=0xF3            ;Demodulator estimate for Link Quality
.equ    RSSI=0xF4           ;Received signal strength indication
.equ    MARCSTATE=0xF5      ;Control state machine state
.equ    WORTIME1=0xF6       ;High byte of WOR timer
.equ    WORTIME0=0xF7       ;Low byte of WOR timer
.equ    PKTSTATUS=0xF8      ;Current GDOx status and packet status
.equ    VCO_VC_DAC=0xF9         ;Current setting from PLL calibration module
.equ    TXBYTES=0xFA        ;Underflow and number of bytes in the TX FIFO
.equ    RXBYTES=0xFB        ;Overflow and number of bytes in the RX FIFO
;
.equ    PATABLE=0x3E        ;single byte
.equ    PATABLE_B=0xFE      ;burst
.equ    TXFIFO=0x3F         ;single byte
.equ    TXFIFO_B=0x7F       ;burst write FIFO
.equ    RXFIFO=0xBF         ;single byte
.equ    RXFIFO_B=0xFF       ;burst read FIFO
```

Отдельно следует остановиться на процедуре сброса трансивера при включении питания. Согласно даташита сброс по включении происходит не позднее 5 мс после достижения напряжения питания уровня 1.8 В. Однако для гарантированного сброса рекомендуется выполнить процедуру сброса. Она заключается в следующем:

```
;Сброс по включении питания (согласно datasheet)
por_cc2500:
        sck_hi              ;установим в 1 SCK
        mosi_lo             ;установим в 0 MOSI
        cs_active           ;установим в 0 CS
        nop                 ;пауза ~1мкс
        cs_inactive         ; установим в 1 CS
        rcall delay_40us    ;подождем 40 мкс
        cs_active           ; установим в 0 CS
por_loop:
        sbic PINB,miso      ;ждем 0 на MISO
        rjmp por_loop
        ldi temp,SRES       ;шлем строб SRES
        mov spi_out,temp
        rcall rw_spi
        cs_inactive
        ret
```

Согласно документации Chipcon при сбросе все регистры устанавливаются в значения по умолчанию. Однако есть подозрение, что внутреннее состояние CC2500, недоступное через регистры, отличается при сбросе программном и сбросе при включении питания. Выражается это в том, что иногда CC2500 виснет и сбросить ее программно не удается. Возможно виноваты мои кривые руки ☺.

Процедура инициализации регистров CC2500 должна следовать за сбросом. Здесь тоже не все чисто. Имеет значение порядок инициализации регистров. В начале работы с трансивером я наивно поверил даташиту, в плане того, что возможен burst доступ на запись в конфигурационные регистры, то есть можно коротким циклом без лишних движений залить все конфигурационные регистры инкрементируя адрес от 00 до 2E. Однако нарвавшись на непонятные глюки внимательно посмотрел на примеры использования трансивера в архиве CC1100_CC2500_Examples_Libraries_1_3.zip и обнаружил, что порядок следования регистров при инициализации все же важен. Более того, включать, например, режим WOR путем записи соответствующих регистров надо не во время инициализации, а после неё. В даташите об этом ни слова (или я плохо читал?...).

Ниже приведены значения регистров для инициализации CC2500 с кварцем 26.601712 МГц. В слове первый байт – адрес регистра, второй – значение.

```
Rfsettings:
;следующие 2 слова определяют параметры синтезатора, получены с помощью Smart RF Studio.
_FSCTRL1:   .dw $0B06     ;Frequency synthesizer control
_FSCTRL0:   .dw $0C00     ;Frequency synthesizer control
;следующие три слова определяют базовую несущую частоту. Получено с помощью Smart RF Studio.
_FREQ2:     .dw $0D5B     ;Frequency control word, high byte
_FREQ1:     .dw $0E75     ;Frequency control word, middle byte
_FREQ0:     .dw $0FD4     ;Frequency control word, low byte
;следующие 5 слов определяют параметры встроенного модема CC2500: скорость передачи 100 кБит/с,
;ширина канала 199.707677 кГц, несущая частота 2432.999659 МГц, ширина полосы пропускания
;фильтра при приеме 207.825875 кГц, модуляция MSK (QPSK), манчестерское кодирование выключено,
;FEC включено (должно быть одинаково на приеме и передаче), 8 байт в преамбуле, остальное по
;умолчанию.
_MDMCFG4: .dw $108B     ;Modem configuration
_MDMCFG3: .dw $11ED     ;Modem configuration
_MDMCFG2: .dw $1273     ;Modem configuration
_MDMCFG1: .dw $13C2     ;Modem configuration
_MDMCFG0: .dw $14EC     ;Modem configuration
;номер канала.
_CHANNR:    .dw $0A00     ;Channel number
;девиация, для MSK не имеет смысла.
_DEVIATN:   .dw $1500     ;Modem deviation setting
;следующие два слова получены  с помощью Smart RF Studio.
_FREND1:    .dw $2156     ;Front end RX configuration
_FREND0:    .dw $2210     ;Front end TX configuration
;следующие 3 слова определяют конфигурацию автомата контроля радио в CC2500: RX_TIME_RSSI,
;RX_TIME_QUAL – выключены, RX_TIME – не переходить в IDLE до конца пакета, CCA_MODE –
;канал чист, если сигнал по уровню ниже порога RSSI, после режимов приема и передачи переходить в
;IDLE, MCSM0 – по умолчанию.
_MCSM2:     .dw $1607     ;Main Radio Control State Machine configuration
_MCSM1:     .dw $1730     ;Main Radio Control State Machine configuration
_MCSM0:     .dw $1818     ;Main Radio Control State Machine configuration
;компенсация сдвига частоты.
_FOCCFG:    .dw $1916     ;Frequency Offset Compensation configuration
;конфигурация битовой синхронизации. Нормально описано в даташите версии 1_1.
_BSCFG:     .dw $1A6C     ;Bit Synchronization configuration
;следующие 3 слова получены с помощью Smart RF Studio и определяют параметры малошумящих
;усилителей и порог чувствительности компаратора при приеме.
_AGCCTRL2:.dw $1B43     ;AGC control
_AGCCTRL1:.dw $1C40     ;AGC control
_AGCCTRL0:.dw $1D91     ;AGC control
;следующие 4 слова определяют параметры калибровки синтезатора, получены с помощью Smart RF
;Studio
```

```
_FSCAL3:    .dw $23A9    ;Frequency synthesizer calibration
_FSCAL2:    .dw $240A    ;Frequency synthesizer calibration
_FSCAL1:    .dw $2500    ;Frequency synthesizer calibration
_FSCAL0:    .dw $2611    ;Frequency synthesizer calibration
;следующие 4 слова Chipcon не рекомендует изменять, получены с помощью Smart RF Studio
_FSTEST:    .dw $2959    ;Frequency synthesizer calibration control
_TEST2:     .dw $2C81    ;Various test settings
_TEST1:     .dw $2D35    ;Various test settings
_TEST0:     .dw $2E0B    ;Various test settings
;функция вывода GDO2, в данном случае обнаружение несущей. Если уровень несущей выше порога
RSSI, то 1 иначе 0.
_IOCFG2:    .dw $000E    ;GDO2 conf - Carrier sense
; функция вывода GDO0, в данном случае при приеме устанавливается в 1 при приеме синхрослова,
;затем сбрасывается, когда прием пакета завершен; при передаче 1 в начале передачи, 0 – по окончании
;передачи.
_IOCFG0:    .dw $0206    ;GDO0 conf - packet received with CRC ok
;следующие 2 слова определяют параметры пакетов, в данном случае: включен режим автоматического
добавления к пакету при приеме RSSI и LQI, адрес не проверяется, данные очищаются (Whitening on),
CRC считается, пакет переменной длины, режим использования ФИФО включен.
_PKTCTRL1: .dw $0704    ;Packet automation control
_PKTCTRL0: .dw $0845    ;Packet automation control
; Адрес устройства. На передающей стороне пишем в буфер ФИФО самостоятельно. То, что записано в
;регистр ADDR используется только для проверки при приеме пакета.
_ADDR:      .dw $0900    ;Device address
;длина пакета
_PKTLEN:    .dw $06FF    ;Packet length default (255)
;граница сигнализации переполнения ФИФО, оставлено по умолчанию.
_FIFOTHR:   .dw $0307    ;RX FIFO and TX FIFO thresholds default
```

Всего 37 слов, или 74 байта. Кое что можно наверное опустить, если совпадает с умолчанием, но я не пробовал.

Остальные конфигурационные регистры оставлены в значениях по умолчанию: _SYNC1,_SYNC0,_WOREVT1,_WOREVT0,_WORCTRL,_RCCTRL1,_RCCTRL0,_PTEST,_AGCTEST.

Ниже приведена подпрограмма инициализации конфигурации CC2500.

```
;Запись в CC2500 начальных установок
;значения установок - в массиве Rfsettings
write_settings:
        push temp
        in temp, SREG
        push temp
        ;
        ldi flash,74                ;длина блока установок в байтах
        ldi ZL,low(Rfsettings*2)    ;начало блока во флэше
        ldi ZH,high(Rfsettings*2)
wrs_loop:
        lpm temp,Z+                 ;читаем адрес
        mov spi_out1,temp
        dec flash
        lpm temp,Z+                 ;читаем значение
        mov spi_out,temp
        rcall write_reg             ;пишем в регистр
        dec flash
        brne wrs_loop
```

```
;
pop temp
out SREG,temp
pop temp
ret
```

Трансивер CC2500 имеет 8-ми байтную таблицу значений мощности передатчика PATABLE. Я использовал только одно значение. Подпрограмма ниже:

```
;Пишем мощность передатчика в таблицу мощностей
write_patable:
	ldi temp,PATABLE
	mov spi_out,temp
	ldi temp,$BB		;-2 дБ см. даташит для других мощностей
	mov spi_out1,temp
	rcall write_reg
	ret
```

Таким образом общий процесс инициализации CC2500 при включении питания будет таким:

```
	rcall init_spi		;инициализируем программный SPI
	rcall por_cc2500	;сброс CC2500
	rcall write_settings	;пишем конфигурацию в регистры
	rcall write_patable	;заполняем таблицу мощностей
	;
	ldi temp,SIDLE
	mov spi_out,temp
	rcall write_strob	;перешли в режим IDLE
	;
	ldi temp,SFRX		;обнулить фифо, сделать это можно только из режима IDLE
	mov spi_out,temp
	rcall write_strob
	;
	ldi temp,SFTX		;обнулить фифо, сделать это можно только из режима IDLE
	mov spi_out,temp
	rcall write_strob
```

Я не привожу здесь код программы начальной инициализации ATmega48, для которой написаны все подпрограммы, так как процесс этот не должен вызвать затруднений и к тому же зависит от конкретной схемы включения. По окончании инициализации CC2500 будет находиться в состоянии IDLE. Включение режимов приема и передачи производится посылкой соответствующих стробов:

```
	ldi temp,SRX
	mov spi_out,temp
	rcall write_strob	;включили RX
	rcall delay_1ms
	;
	ldi temp,STX
	mov spi_out,temp
	rcall write_strob	;включили TX
	rcall delay_1ms
```

Обратите внимание на задержку в 1мс после посылки стробов. В данном случае переход CC2500 из состояния IDLE в состояние RX или TX происходит с последующей калибровкой синтезатора и занимает этот процесс 809 мкс при кварце 26 МГц (см. даташит версии 1_1, стр. 37). Возможно сконфигурировать CC2500 для другого варианта, когда переход в режим приема или передачи будет происходить без калибровки. В таком случае время перехода 88.4 мкс (см. даташит версии 1_1, стр. 37),

однако это потребует изменения конфигурационных регистров CC2500 (MCSM0, стр. 64 даташит версии 1_1). Время нахождения в режиме приема или передачи тоже может быть ограничено. В приведенной выше конфигурации указано перейти в IDLE по завершении приема или передачи пакета (MCSM1, стр. 63 даташит версии 1_1). Таким образом, если мы хотим что-то непрерывно принимать (передавать), то надо в цикле крутить установку в режим приема (передачи). С точки зрения простоты можно было бы сконфигурировать MCSM1 так, что бы оставаться в режиме приема после окончания приема пакета, однако в этом случае возрастает энергопотребление (15 мА – прием, 1.5 мА - IDLE), кроме того для принудительной очистки буфера (в случае ошибки при приеме, например) все равно придется переходить в IDLE, так как строб SFRX или SFTX можно выдать только находясь в IDLE. Если нет желания тратить процессорное время, можно вместо задержки покрутить в цикле чтение статуса CC2500 до получения состояния IDLE. Вообще для понимания функционирования CC2500 полезно внимательно рассмотреть диаграммы состояния CC2500 (стр. 18, 34 даташита 1_1). При использовании сигналов на ногах GDO0, GDO2 можно организовать работу по прерываниям:

```
        ;**** Interrupt Vectors ****
                rjmp    RESET           ;=$000 ;Reset Handler
                reti                    ;=$001 ;External Interrupt0
                reti                    ;=$002 ;External Interrupt1
                rjmp    PCI0            ;=$003 ;Pin Change Interrupt0 используем для приема
                ;
                ….
                ;
;настройка прерываний
        ldi temp,(1<<PCINT0)        ;по стробу gdo0
        sts PCMSK0,temp
        ldi temp,(1<<PCIE0)
        sts PCICR,temp

;/////////////////ВЕКТОР ПРЕРЫВАНИЙ//////////////////////////////
PCI0:
        push temp
        in temp,SREG                ;сохраним состояние и используемый регистр
        push temp
        ;
        sbic PINB,gd0
        rjmp PCI0_end
        rcall read_fifo             ;читаем ФИФО
        ;
        ldi temp,SIDLE
        mov spi_out,temp
        rcall write_strob
        ;
        ldi temp,SFRX               ;обнулить фифо
        mov spi_out,temp
        rcall write_strob
PCI0_end:
        ;
        pop temp
        out SREG,temp
        pop temp
        reti
```

Использование прерывания по изменению уровня сигнала на ноге порта вызвано удобством разводки, с точки зрения упрощения обработчика прерывания лучше использовать вход INT0 или INT1. Еще одно замечание – если вы строите трансивер (прием и передача), данная подпрограмма потребует модификации для случая прерывания при передаче. Подпрограмма чтения ФИФО приведена ниже:

```
read_fifo:
        push YL
        push YH
        push flash
        ;
        ;читаем RXBYTES  ****************************************
        ldi temp,RXBYTES                                        ;*
        mov spi_out,temp                                        ;*
        rcall read_reg                                          ;*
        ldi temp,$80            ;маскируем бит переполнения     ;*
        and temp,spi_in                                         ;*
        cpi temp,0              ;если нет переполнения читаем пакет ;*
        breq rf1_packet                                         ;*
        rjmp rf1_end            ;завершаемся по ошибке переполнения ;*
        ;****************************************************************
rf1_packet:
        ldi temp,RXFIFO_B  ;burst read
        mov spi_out,temp
        cs_active
rf1_loop:
        sbic PINB,miso
        rjmp rf1_loop
        rcall rw_spi            ;прочитали статус
        nop
        ;
        ldi temp,0
        mov spi_out,temp
        rcall rw_spi            ;читаем длину пакета
        mov flash,spi_in
        inc flash
        inc flash               ;пакет и 2 байта RSSI and LQI
        ;
        ldi YL,low(buf_out)
        ldi YH,high(buf_out)
        ;
rf1_loop1:
        ldi temp,0
        mov spi_out,temp
        ;
        rcall rw_spi
        st Y+,spi_in
        dec flash
        brne rf1_loop1
        cs_inactive
rf1_end:
        pop flash
        pop YH
        pop YL
        ret
```

В приведенной подпрограмме переменная flash используется для хранения длины принятого пакета в ФИФО, к ней надо добавить 2, чтобы прочесть байт RSSI и байт LQI. Принятый пакет записывается в ОЗУ для дальнейшего анализа. Первым байтом в ФИФО будет адрес передатчика, затем собственно пакет и два байта RSSI и LQI.

С приемом пока все, посмотрим на передачу. В общем виде у меня процесс передачи пакета организован так: пишем пакет в буфер ФИФО, выдаем строб передачи, ждем 1 на GDO0, что говорит о передаче синхрослова, затем ждем 0 на GDO0, что говорит о завершении передачи пакета. Вот кусочек из листинга:

```
        rcall write_txfifo
        ldi temp,STX
        mov spi_out,temp
        rcall write_strob
mn1_loop:
        sbis PINB,gd0
        rjmp mn1_loop
mn2_loop:
        sbic PINB,gd0
        ;
```

Ниже привожу подпрограмму записи пакета в ФИФО:

```
Paket:  .db  'Q','W','E','R','T' ;5 байт, пакет для передачи

;Пишем в буфер передачи пакет
write_txfifo:
        ldi flash,5             ;длина блока в байтах
        ldi ZL,low(Paket*2)     ;начало блока во флэше
        ldi ZH,high(Paket*2)
        ldi temp,TXFIFO
        mov spi_out,temp
        ldi temp,0x40           ;burst
        add spi_out,temp
        cs_active
wtxt1_loop:
        sbic PINB,miso
        rjmp wtxt1_loop
        rcall rw_spi
        ;
        ldi temp,6              ;длина пакета, вместе с байтом длины
        mov spi_out,temp
        rcall rw_spi
        ldi temp,0              ;адрес, заполняем ручками, какой надо
        mov spi_out,temp
        rcall rw_spi
        ;
wtxt1_lp1:
        lpm temp,Z
        adiw r30,1
        mov spi_out,temp
        rcall rw_spi
        dec flash
        brne wtxt1_lp1
        cs_inactive
        ret
```

В принципе приведенной информации достаточно для организации приема и передачи. Теперь коснемся вопроса подстройки частоты. Как известно, точного совпадения частоты кварцев не бывает, всегда есть погрешность. Чем ниже точность кварца, тем сложнее может оказаться подгонка. Задачу усложняет возможность заузить полосу приема вплоть до 58 кГц. Для подстройки достаточно изменять значение двух регистров CC2500: FREQ0 и FREQ1. Я провожу эту процедуру следующим образом. При

указанной выше конфигурации ставлю одно из устройств на прием. На этом устройстве смотрю на уровень сигнала на ноге GDO2, сконфигурированной для индикации обнаружения несущей и на уровень на ноге GDO0, сконфигурированной для индикации приема пакета (см. описание регистров конфигурации выше). Второе устройство в цикле непрерывно передает пакет известного мне содержания с периодом в одну - две секунды. Частоту несущей можно менять или на приемном или на передающем конце (но не одновременно). Подпрограммы для изменения частоты приведена ниже:

```
Up_freq:
        ;
        cli
        ldi temp,FREQ0
        mov spi_out,temp
        rcall read_reg
        mov XL,spi_in
        ldi temp,FREQ1
        mov spi_out,temp
        rcall read_reg
        mov XH,spi_in
        adiw XL,25              ;увеличим частоту на 10 кГц
        mov spi_out1,XL
        ldi temp,FREQ0
        mov spi_out,temp
        rcall write_reg         ;запишем в регистр
        mov spi_out1,XH
        ldi temp,FREQ1
        mov spi_out,temp
        rcall write_reg         ;запишем в регистр
        ;
        ldi temp,SCAL
        mov spi_out,temp
        rcall write_strob       ;калибруем синтезатор
        ;                       ;вновь вернемся в режим приема
        ldi temp,SRX
        mov spi_out,temp
        rcall write_strob
        sei
        ;
        ret
Down_freq:
        ;
        cli
        ldi temp,FREQ0
        mov spi_out,temp
        rcall read_reg
        mov XL,spi_in
        ldi temp,FREQ1
        mov spi_out,temp
        rcall read_reg
        mov XH,spi_in
        sbiw XL,25              ;уменьшим частоту на 10 кГц
        mov spi_out1,XL
        ldi temp,FREQ0
        mov spi_out,temp
        rcall write_reg         ;запишем в регистр
        mov spi_out1,XH
        ldi temp,FREQ1
        mov spi_out,temp
```

```
rcall write_reg          ;запишем в регистр
;
ldi temp,SCAL
mov spi_out,temp
rcall write_strob        ;калибруем синтезатор
;                        ;вновь вернемся в режим приема
ldi temp,SRX
mov spi_out,temp
rcall write_strob
sei
;
ret
```

Собственно подпрограммы не отличаются друг от друга кроме одной строки. Принцип такой: читаем два регистра FREQ0 и FREQ1, сохраняем их в регистровой паре X, увеличиваем (уменьшаем) величину 16 разрядного числа в регистровой паре X, пишем регистры обратно, калибруем синтезатор. Число 25 выбрано для удобства отсчета смещения частоты. Изменение регистра FREQ0 на единицу вызывает изменение частоты приблизительно на 400 Гц (можно посчитать точно, это зависит от частоты кварца), соответственно 25*400=10000. Можно сделать шаг перестройки больше или меньше, кому как надо. Мне оказалось достаточно 10 кГц при полосе пропускания 200 кГц. При минимальной полосе 58 кГц (при этом правда и скорость будет не больше ~20 кБит/с) вероятно имеет смысл изменить шаг на 1 кГц. Дальше дело техники. Подпрограммы можно навесить на обработчик нажатий кнопки (например повесить пару кнопок на свободные ноги Меги) и повышая или понижая частоту (с одной стороны) добиваемся сначала появления обнаружения несущей (1 на GDO2), затем приема пакета (1 на GDO0). Посчитав количество нажатий кнопки, можно пересчитать значение для инициализации регистров CC2500. Например, если устойчивый прием пакетов начался после второго шага увеличения частоты, а прекратился после 18-го то начальное значение регистров FREQ0 и FREQ1 надо увеличить на: (2+(18-2)/2)*25=250. Заметьте, что работать с регистровой парой гораздо удобнее, не надо следить за переполнением из младшего байта.

Из личного опыта могу сказать, что даже при использовании кварцев 26.601712 МГц приходилось иногда в пределах 20-30 кГц подстраивать, а кварцы 27.000 МГц все требовали подстройки в пределах 100-120 кГц, причем иногда в разные стороны. Еще одно замечание: при точной настройке приемной и передающей частей чувствительность устройств максимальна, что позволяет иметь устойчивую связь на заданном расстоянии при меньшей выходной мощности.

Вот пожалуй и все, для начала. Во второй части опуса планирую описать варианты примененных антенн, сопоставить по мере сил плюсы и минусы.

В заключении – по моему мнению CC2500 весьма неплохая реализация трансивера диапазона 2.4 ГГц относительно несложная для освоения и имеющая достаточно богатые возможности. Я описал только те ньюансы, с которыми разобрался. Сейчас на повестке дня режим WOR, здесь пока не все понятно. По мере сил буду стараться информировать сообщество о результатах.

P.S. Критика дополнения и исправления приветствуются.

Коломыцев С.В. Ставрополь 2006 г.