

```

1
2 #define UCPHA0 1
3 #define F_CPU 8000000UL
4 #define BAUD_RATE 9600UL
5 #define UBRR_VALUE ((F_CPU)/(2UL*BAUD_RATE))-1
6
7 #include "nrf24.h"
8 #include <avr/io.h>
9
10 uint8_t payload_len;
11
12 void nrf24_init()
13 {
14     nrf24_setupPins();
15     nrf24_ce_digitalWrite(LOW);
16     nrf24_csn_digitalWrite(HIGH);
17 }
18
19 void nrf24_config(uint8_t channel, uint8_t pay_length)
20 {
21     /* Use static payload length ... */
22     payload_len = pay_length;
23
24     // Set RF channel
25     nrf24_configRegister(RF_CH,channel);
26
27     // Set length of incoming payload
28     nrf24_configRegister(RX_PW_P0, 0x00); // Auto-ACK pipe ...
29     nrf24_configRegister(RX_PW_P1, payload_len); // Data payload pipe
30     nrf24_configRegister(RX_PW_P2, 0x00); // Pipe not used
31     nrf24_configRegister(RX_PW_P3, 0x00); // Pipe not used
32     nrf24_configRegister(RX_PW_P4, 0x00); // Pipe not used
33     nrf24_configRegister(RX_PW_P5, 0x00); // Pipe not used
34
35     // 1 Mbps, TX gain: 0dbm
36     nrf24_configRegister(RF_SETUP, (0<<RF_DR)|((0x03)<<RF_PWR));
37
38     // CRC enable, 1 byte CRC length
39     nrf24_configRegister(CONFIG,nrf24_CONFIG);
40
41     // Auto Acknowledgment
42     nrf24_configRegister(EN_AA,(1<<ENAA_P0)|(1<<ENAA_P1)|(0<<ENAA_P2)| 7
43         (0<<ENAA_P3)|(0<<ENAA_P4)|(0<<ENAA_P5));
44
45     // Enable RX addresses
46     nrf24_configRegister(EN_RXADDR,(1<<ERX_P0)|(1<<ERX_P1)|(0<<ERX_P2)| 7
47         (0<<ERX_P3)|(0<<ERX_P4)|(0<<ERX_P5));
48
49     // Auto retransmit delay: 1000 us and Up to 15 retransmit trials
50     nrf24_configRegister(SETUP_RETR,(0x04<<ARD)|(0x0F<<ARC));
51
52     // Dynamic length configurations: No dynamic length

```

```
51     nrf24_configRegister(DYNPD, (0<<DPL_P0)|(0<<DPL_P1)|(0<<DPL_P2)|(0<<DPL_P3)|  
        (0<<DPL_P4)|(0<<DPL_P5));  
52  
53     // Start listening  
54     nrf24_powerUpRx();  
55 }  
56  
57 /* Set the RX address */  
58 void nrf24_rx_address(uint8_t * adr)  
59 {  
60     nrf24_ce_digitalWrite(LOW);  
61     nrf24_writeRegister(RX_ADDR_P1, adr, nrf24_ADDR_LEN);  
62     nrf24_ce_digitalWrite(HIGH);  
63 }  
64  
65 /* Returns the payload length */  
66 uint8_t nrf24_payload_length()  
67 {  
68     return payload_len;  
69 }  
70  
71 /* Set the TX address */  
72 void nrf24_tx_address(uint8_t* adr)  
73 {  
74     /* RX_ADDR_P0 must be set to the sending addr for auto ack to work. */  
75     nrf24_writeRegister(RX_ADDR_P0, adr, nrf24_ADDR_LEN);  
76     nrf24_writeRegister(TX_ADDR, adr, nrf24_ADDR_LEN);  
77 }  
78  
79 /* Checks if data is available for reading */  
80 /* Returns 1 if data is ready ... */  
81 uint8_t nrf24_dataReady()  
82 {  
83     // See note in getData() function - just checking RX_DR isn't good enough  
84     uint8_t status = nrf24_getStatus();  
85  
86     // We can short circuit on RX_DR, but if it's not set, we still need  
87     // to check the FIFO for any pending packets  
88     if ( status & (1 << RX_DR) )  
89     {  
90         return 1;  
91     }  
92  
93     return !nrf24_rxFifoEmpty();  
94 }  
95  
96 /* Checks if receive FIFO is empty or not */  
97 uint8_t nrf24_rxFifoEmpty()  
98 {  
99     uint8_t fifoStatus;  
100  
101     nrf24_readRegister(FIFO_STATUS, &fifoStatus, 1);
```

```
102
103     return (fifoStatus & (1 << RX_EMPTY));
104 }
105
106 /* Returns the length of data waiting in the RX fifo */
107 uint8_t nrf24_payloadLength()
108 {
109     uint8_t status;
110     nrf24_csn_digitalWrite(LOW);
111     spi_transfer(R_RX_PL_WID);
112     status = spi_transfer(0x00);
113     nrf24_csn_digitalWrite(HIGH);
114     return status;
115 }
116
117 /* Reads payload bytes into data array */
118 void nrf24_getData(uint8_t* data)
119 {
120     /* Pull down chip select */
121     nrf24_csn_digitalWrite(LOW);
122
123     /* Send cmd to read rx payload */
124     spi_transfer( R_RX_PAYLOAD );
125
126     /* Read payload */
127     nrf24_transferSync(data,data,payload_len);
128
129     /* Pull up chip select */
130     nrf24_csn_digitalWrite(HIGH);
131
132     /* Reset status register */
133     nrf24_configRegister(STATUS,(1<<RX_DR));
134 }
135
136 /* Returns the number of retransmissions occurred for the last message */
137 uint8_t nrf24_retransmissionCount()
138 {
139     uint8_t rv;
140     nrf24_readRegister(OBSERVE_TX,&rv,1);
141     rv = rv & 0x0F;
142     return rv;
143 }
144
145 // Sends a data package to the default address. Be sure to send the correct
146 // amount of bytes as configured as payload on the receiver.
147 void nrf24_send(uint8_t* value)
148 {
149     /* Go to Standby-I first */
150     nrf24_ce_digitalWrite(LOW);
151
152     /* Set to transmitter mode , Power up if needed */
153     nrf24_powerUpTx();
```

```
154
155     /* Do we really need to flush TX fifo each time ? */
156     #if 1
157         /* Pull down chip select */
158         nrf24_csn_digitalWrite(LOW);
159
160         /* Write cmd to flush transmit FIFO */
161         spi_transfer(FLUSH_TX);
162
163         /* Pull up chip select */
164         nrf24_csn_digitalWrite(HIGH);
165     #endif
166
167     /* Pull down chip select */
168     nrf24_csn_digitalWrite(LOW);
169
170     /* Write cmd to write payload */
171     spi_transfer(W_TX_PAYLOAD);
172
173     /* Write payload */
174     nrf24_transmitSync(value,payload_len);
175
176     /* Pull up chip select */
177     nrf24_csn_digitalWrite(HIGH);
178
179     /* Start the transmission */
180     nrf24_ce_digitalWrite(HIGH);
181 }
182
183 uint8_t nrf24_isSending()
184 {
185     uint8_t status;
186
187     /* read the current status */
188     status = nrf24_getStatus();
189
190     /* if sending successful (TX_DS) or max retries exceded (MAX_RT). */
191     if((status & ((1 << TX_DS) | (1 << MAX_RT))))
192     {
193         return 0; /* false */
194     }
195
196     return 1; /* true */
197 }
198
199
200 uint8_t nrf24_getStatus()
201 {
202     uint8_t rv;
203     nrf24_csn_digitalWrite(LOW);
204     rv = spi_transfer(NOP);
205     nrf24_csn_digitalWrite(HIGH);
```

```
206     return rv;
207 }
208
209 uint8_t nrf24_lastMessageStatus()
210 {
211     uint8_t rv;
212
213     rv = nrf24_getStatus();
214
215     /* Transmission went OK */
216     if((rv & ((1 << TX_DS))))
217     {
218         return NRF24_TRANSMISSION_OK;
219     }
220     /* Maximum retransmission count is reached */
221     /* Last message probably went missing ... */
222     else if((rv & ((1 << MAX_RT))))
223     {
224         return NRF24_MESSAGE_LOST;
225     }
226     /* Probably still sending ... */
227     else
228     {
229         return 0xFF;
230     }
231 }
232
233 void nrf24_powerUpRx()
234 {
235     nrf24_csn_digitalWrite(LOW);
236     spi_transfer(FLUSH_RX);
237     nrf24_csn_digitalWrite(HIGH);
238
239     nrf24_configRegister(STATUS, (1<<RX_DR)|(1<<TX_DS)|(1<<MAX_RT));
240
241     nrf24_ce_digitalWrite(LOW);
242     nrf24_configRegister(CONFIG, nrf24_CONFIG | ((1<<PWR_UP)|(1<<PRIM_RX)));
243     nrf24_ce_digitalWrite(HIGH);
244 }
245
246 void nrf24_powerUpTx()
247 {
248     nrf24_configRegister(STATUS, (1<<RX_DR)|(1<<TX_DS)|(1<<MAX_RT));
249
250     nrf24_configRegister(CONFIG, nrf24_CONFIG | ((1<<PWR_UP)|(0<<PRIM_RX)));
251 }
252
253 void nrf24_powerDown()
254 {
255     nrf24_ce_digitalWrite(LOW);
256     nrf24_configRegister(CONFIG, nrf24_CONFIG);
257 }
```

```
258
259 uint8_t spi_transfer(uint8_t tx)
260 {
261     uint8_t i = 0;
262     uint8_t rx = 0;
263
264     nrf24_sck_digitalWrite(LOW);
265
266     for(i=0;i<8;i++)
267     {
268
269         if(tx & (1<<(7-i)))
270         {
271             nrf24_mosi_digitalWrite(HIGH);
272         }
273         else
274         {
275             nrf24_mosi_digitalWrite(LOW);
276         }
277
278         nrf24_sck_digitalWrite(HIGH);
279
280         rx = rx << 1;
281         if(nrf24_miso_digitalRead())
282         {
283             rx |= 0x01;
284         }
285
286         nrf24_sck_digitalWrite(LOW);
287     }
288
289     return rx;
290 }
291
292
293 /* send and receive multiple bytes over SPI */
294 void nrf24_transferSync(uint8_t* dataout,uint8_t* datain,uint8_t len)
295 {
296     uint8_t i;
297
298     for(i=0;i<len;i++)
299     {
300         datain[i] = spi_transfer(dataout[i]);
301     }
302 }
303
304
305 /* send multiple bytes over SPI */
306 void nrf24_transmitSync(uint8_t* dataout,uint8_t len)
307 {
308     uint8_t i;
309
```

```
310     for(i=0;i<len;i++)
311     {
312         spi_transfer(dataout[i]);
313     }
314
315 }
316
317 /* Clocks only one byte into the given nrf24 register */
318 void nrf24_configRegister(uint8_t reg, uint8_t value)
319 {
320     nrf24_csn_digitalWrite(LOW);
321     spi_transfer(W_REGISTER | (REGISTER_MASK & reg));
322     spi_transfer(value);
323     nrf24_csn_digitalWrite(HIGH);
324 }
325
326 /* Read single register from nrf24 */
327 void nrf24_readRegister(uint8_t reg, uint8_t* value, uint8_t len)
328 {
329     nrf24_csn_digitalWrite(LOW);
330     spi_transfer(R_REGISTER | (REGISTER_MASK & reg));
331     nrf24_transferSync(value,value,len);
332     nrf24_csn_digitalWrite(HIGH);
333 }
334
335 /* Write to a single register of nrf24 */
336 void nrf24_writeRegister(uint8_t reg, uint8_t* value, uint8_t len)
337 {
338     nrf24_csn_digitalWrite(LOW);
339     spi_transfer(W_REGISTER | (REGISTER_MASK & reg));
340     nrf24_transmitSync(value,len);
341     nrf24_csn_digitalWrite(HIGH);
342 }
343
344 #define RF_DDR  DDRC
345 #define RF_PORT PORTC
346 #define RF_PIN  PINC
347
348 #define set_bit(reg,bit) reg |= (1<<bit)
349 #define clr_bit(reg,bit) reg &= ~(1<<bit)
350 #define check_bit(reg,bit) (reg&(1<<bit))
351
352 /* ----- */
353
354 void nrf24_setupPins()
355 {
356     set_bit(RF_DDR,0); // CE output
357     set_bit(RF_DDR,1); // CSN output
358     set_bit(RF_DDR,2); // SCK output
359     set_bit(RF_DDR,3); // MOSI output
360     clr_bit(RF_DDR,4); // MISO input
361 }
```

```
362 /* ----- */
363 void nrf24_ce_digitalWrite(uint8_t state)
364 {
365     if(state)
366     {
367         set_bit(RF_PORT,0);
368     }
369     else
370     {
371         clr_bit(RF_PORT,0);
372     }
373 }
374 /* ----- */
375 void nrf24_csn_digitalWrite(uint8_t state)
376 {
377     if(state)
378     {
379         set_bit(RF_PORT,1);
380     }
381     else
382     {
383         clr_bit(RF_PORT,1);
384     }
385 }
386 /* ----- */
387 void nrf24_sck_digitalWrite(uint8_t state)
388 {
389     if(state)
390     {
391         set_bit(RF_PORT,2);
392     }
393     else
394     {
395         clr_bit(RF_PORT,2);
396     }
397 }
398 /* ----- */
399 void nrf24_mosi_digitalWrite(uint8_t state)
400 {
401     if(state)
402     {
403         set_bit(RF_PORT,3);
404     }
405     else
406     {
407         clr_bit(RF_PORT,3);
408     }
409 }
410 /* ----- */
411 uint8_t nrf24_miso_digitalRead()
412 {
413     return check_bit(RF_PIN,4);
```


414 }

415 /* ----- */

416