

```

1
2 #define UCPHA0 1
3 #define BAUD_RATE 9600UL
4 #define UBRR_VALUE ((F_CPU)/(2UL*BAUD_RATE))-1
5
6 #include "nrf24.h"
7 #include "Command_Handler.h"
8
9 uint8_t payload_len;
10 uint8_t selectedChannel;
11
12 uint8_t MOTORIZED_BOARD_ADDR[5] = {0xF0,0xF0,0xF0,0xF0,0xC9};
13 uint8_t MAIN_BOARD_ADDR[5] = {0xA4,0xA4,0xA4,0xA4,0xA4};
14 uint8_t POWER_BOARD_ADDR[5] = {0xF0,0xF0,0xF0,0xF0,0xF0};
15
16 uint8_t* BOARD_ADDRESS[3] = {&MAIN_BOARD_ADDR[0], &POWER_BOARD_ADDR[0],
17                               &MOTORIZED_BOARD_ADDR[0]};
18
19 uint8_t* CURRENT_BOARD_ADDRESS = &MOTORIZED_BOARD_ADDR[0];
20
21 uint8_t GENERAL_RF_CHANNEL = 112;
22
23 void nrf24_init()
24 {
25     nrf24_setupPins();
26     nrf24_ce_digitalWrite(LOW);
27     nrf24_csn_digitalWrite(HIGH);
28 }
29
30 void nrf24_config(uint8_t channel, uint8_t pay_length)
31 {
32     /* Use static payload length ... */
33     payload_len = pay_length;
34     selectedChannel = channel;
35     // Set RF channel
36     nrf24_configRegister(RF_CH,channel);
37     // Set length of incoming payload
38     nrf24_configRegister(RX_PW_P0, 0x00); // Auto-ACK pipe ...
39     nrf24_configRegister(RX_PW_P1, payload_len); // Data payload pipe
40     nrf24_configRegister(RX_PW_P2, 0x00); // Pipe not used
41     nrf24_configRegister(RX_PW_P3, 0x00); // Pipe not used
42     nrf24_configRegister(RX_PW_P4, 0x00); // Pipe not used
43     nrf24_configRegister(RX_PW_P5, 0x00); // Pipe not used
44     // 1 Mbps, TX gain: 0dbm
45     nrf24_configRegister(RF_SETUP, (0<<RF_DR)|((0<<03)<<RF_PWR));
46     // CRC enable, 1 byte CRC length
47     nrf24_configRegister(CONFIG,nrf24_CONFIG);
48     // Auto Acknowledgment
49     nrf24_configRegister(EN_AA,(1<<ENAA_P0)|(1<<ENAA_P1)|(0<<ENAA_P2)|
50                          (0<<ENAA_P3)|(0<<ENAA_P4)|(0<<ENAA_P5));
51     // Enable RX addresses

```

```

51     nrf24_configRegister(EN_RXADDR,(1<<ERX_P0)|(1<<ERX_P1)|(0<<ERX_P2)|
    (0<<ERX_P3)|(0<<ERX_P4)|(0<<ERX_P5));
52     // Auto retransmit delay: 1000 us and Up to 15 retransmit trials
53     nrf24_configRegister(SETUP_RETR,(0x04<<ARD)|(0x0F<<ARC));
54     // Dynamic length configurations: No dynamic length
55     nrf24_configRegister(DYNPD,(0<<DPL_P0)|(0<<DPL_P1)|(0<<DPL_P2)|(0<<DPL_P3)|
    (0<<DPL_P4)|(0<<DPL_P5));
56
57 }
58
59 bool nrf24_checkConfig(){
60     // Check all registers
61     if (nrf24_checkRegister(RF_CH, selectedChannel,1)==false) return false;
62     if (nrf24_checkRegister(RX_PW_P0, 0x00,1)==false) return false;
63     if (nrf24_checkRegister(RX_PW_P1, payload_len,1)==false) return false;
64     if (nrf24_checkRegister(RX_PW_P2, 0x00,1)==false) return false;
65     if (nrf24_checkRegister(RX_PW_P3, 0x00,1)==false) return false;
66     if (nrf24_checkRegister(RX_PW_P4, 0x00,1)==false) return false;
67     if (nrf24_checkRegister(RX_PW_P5, 0x00,1)==false) return false;
68     if (nrf24_checkRegister(RF_SETUP, (0<<RF_DR)|((0x03)<<RF_PWR),1)==false)
    return false;
69     if (nrf24_checkRegister(CONFIG,nrf24_CONFIG,1)==false) return false;
70     if (nrf24_checkRegister(EN_AA,(1<<ENAA_P0)|(1<<ENAA_P1)|(0<<ENAA_P2)|
    (0<<ENAA_P3)|(0<<ENAA_P4)|(0<<ENAA_P5),1)==false) return false;
71     if (nrf24_checkRegister(SETUP_RETR,(0x04<<ARD)|(0x0F<<ARC),1)==false) return
    false;
72     if (nrf24_checkRegister(DYNPD,(0<<DPL_P0)|(0<<DPL_P1)|(0<<DPL_P2)|
    (0<<DPL_P3)|(0<<DPL_P4)|(0<<DPL_P5),1)==false) return false;
73
74     return true;
75 }
76
77 bool nrf24_checkAvailability(){
78     if (nrf24_checkRegister(RF_CH, selectedChannel,1)==true) { return true; }
    else { return false;}
79 }
80
81
82
83
84 void faultyRF_Alarm(){
85     CLEAR_FAULTY_RF_LED;
86     for (uint8_t x = 0; x < 6; x++)
87     {
88         FLIP_FAULTY_RF_LED;
89         _delay_ms(125);
90     }
91     _delay_ms(250);
92 }
93
94
95

```

```
96  /* Set the RX address */
97  void nrf24_rx_address(uint8_t * adr)
98  {
99      nrf24_ce_digitalWrite(LOW);
100     nrf24_writeRegister(RX_ADDR_P1,adr,nrf24_ADDR_LEN);
101     nrf24_ce_digitalWrite(HIGH);
102 }
103
104 /* Returns the payload length */
105 uint8_t nrf24_payload_length()
106 {
107     return payload_len;
108 }
109
110 /* Set the TX address */
111 void nrf24_tx_address(uint8_t* adr)
112 {
113     /* RX_ADDR_P0 must be set to the sending addr for auto ack to work. */
114     nrf24_writeRegister(RX_ADDR_P0,adr,nrf24_ADDR_LEN);
115     nrf24_writeRegister(TX_ADDR,adr,nrf24_ADDR_LEN);
116 }
117
118 /* Checks if data is available for reading */
119 /* Returns 1 if data is ready ... */
120 uint8_t nrf24_dataReady()
121 {
122     // See note in getData() function - just checking RX_DR isn't good enough
123     uint8_t status = nrf24_getStatus();
124
125     // We can short circuit on RX_DR, but if it's not set, we still need
126     // to check the FIFO for any pending packets
127     if ( status & (1 << RX_DR) )
128     {
129         return 1;
130     }
131
132     return !nrf24_rxFifoEmpty();;
133 }
134
135 /* Checks if receive FIFO is empty or not */
136 uint8_t nrf24_rxFifoEmpty()
137 {
138     uint8_t fifoStatus;
139
140     nrf24_readRegister(FIFO_STATUS,&fifoStatus,1);
141
142     return (fifoStatus & (1 << RX_EMPTY));
143 }
144
145 /* Returns the length of data waiting in the RX fifo */
146 uint8_t nrf24_payloadLength()
147 {
```

```
148     uint8_t status;
149     nrf24_csn_digitalWrite(LOW);
150     spi_transfer(R_RX_PL_WID);
151     status = spi_transfer(0x00);
152     nrf24_csn_digitalWrite(HIGH);
153     return status;
154 }
155
156 /* Reads payload bytes into data array */
157 void nrf24_getData(uint8_t* data)
158 {
159     /* Pull down chip select */
160     nrf24_csn_digitalWrite(LOW);
161
162     /* Send cmd to read rx payload */
163     spi_transfer( R_RX_PAYLOAD );
164
165     /* Read payload */
166     nrf24_transferSync(data,data,payload_len);
167
168     /* Pull up chip select */
169     nrf24_csn_digitalWrite(HIGH);
170
171     /* Reset status register */
172     nrf24_configRegister(STATUS,(1<<RX_DR));
173 }
174
175 /* Returns the number of retransmissions occurred for the last message */
176 uint8_t nrf24_retransmissionCount()
177 {
178     uint8_t rv;
179     nrf24_readRegister(OBSERVE_TX,&rv,1);
180     rv = rv & 0x0F;
181     return rv;
182 }
183
184 // Sends a data package to the default address. Be sure to send the correct
185 // amount of bytes as configured as payload on the receiver.
186 void nrf24_send(uint8_t* value)
187 {
188     /* Go to Standby-I first */
189     nrf24_ce_digitalWrite(LOW);
190
191     /* Set to transmitter mode , Power up if needed */
192     nrf24_powerUpTx();
193
194     /* Do we really need to flush TX fifo each time ? */
195     #if 1
196         /* Pull down chip select */
197         nrf24_csn_digitalWrite(LOW);
198
199         /* Write cmd to flush transmit FIFO */
```

```
200     spi_transfer(FLUSH_TX);
201
202     /* Pull up chip select */
203     nrf24_csn_digitalWrite(HIGH);
204 #endif
205
206     /* Pull down chip select */
207     nrf24_csn_digitalWrite(LOW);
208
209     /* Write cmd to write payload */
210     spi_transfer(W_TX_PAYLOAD);
211
212     /* Write payload */
213     nrf24_transmitSync(value,payload_len);
214
215     /* Pull up chip select */
216     nrf24_csn_digitalWrite(HIGH);
217
218     /* Start the transmission */
219     nrf24_ce_digitalWrite(HIGH);
220 }
221
222 uint8_t nrf24_isSending()
223 {
224     uint8_t status;
225
226     /* read the current status */
227     status = nrf24_getStatus();
228
229     /* if sending successful (TX_DS) or max retries exceded (MAX_RT). */
230     if((status & ((1 << TX_DS) | (1 << MAX_RT))))
231     {
232         return 0; /* false */
233     }
234
235     return 1; /* true */
236 }
237
238
239 uint8_t nrf24_getStatus()
240 {
241     uint8_t rv;
242     nrf24_csn_digitalWrite(LOW);
243     rv = spi_transfer(NOP);
244     nrf24_csn_digitalWrite(HIGH);
245     return rv;
246 }
247
248 uint8_t nrf24_lastMessageStatus()
249 {
250     uint8_t rv;
251
```

```
252     rv = nrf24_getStatus();
253
254     /* Transmission went OK */
255     if((rv & ((1 << TX_DS))))
256     {
257         return NRF24_TRANSMISSION_OK;
258     }
259     /* Maximum retransmission count is reached */
260     /* Last message probably went missing ... */
261     else if((rv & ((1 << MAX_RT))))
262     {
263         return NRF24_MESSAGE_LOST;
264     }
265     /* Probably still sending ... */
266     else
267     {
268         return 0xFF;
269     }
270 }
271
272 void nrf24_powerUpRx()
273 {
274     nrf24_csn_digitalWrite(LOW);
275     spi_transfer(FLUSH_RX);
276     nrf24_csn_digitalWrite(HIGH);
277
278     nrf24_configRegister(STATUS, (1<<RX_DR)|(1<<TX_DS)|(1<<MAX_RT));
279
280     nrf24_ce_digitalWrite(LOW);
281     nrf24_configRegister(CONFIG, nrf24_CONFIG|((1<<PWR_UP)|(1<<PRIM_RX)));
282     nrf24_ce_digitalWrite(HIGH);
283 }
284
285 void nrf24_powerUpTx()
286 {
287     nrf24_configRegister(STATUS, (1<<RX_DR)|(1<<TX_DS)|(1<<MAX_RT));
288
289     nrf24_configRegister(CONFIG, nrf24_CONFIG|((1<<PWR_UP)|(0<<PRIM_RX)));
290 }
291
292 void nrf24_powerDown()
293 {
294     nrf24_ce_digitalWrite(LOW);
295     nrf24_configRegister(CONFIG, nrf24_CONFIG);
296 }
297
298 uint8_t spi_transfer(uint8_t tx)
299 {
300     uint8_t i = 0;
301     uint8_t rx = 0;
302
303     nrf24_sck_digitalWrite(LOW);
```

```
304
305     for(i=0;i<8;i++)
306     {
307
308         if(tx & (1<<(7-i)))
309         {
310             nrf24_mosi_digitalWrite(HIGH);
311         }
312         else
313         {
314             nrf24_mosi_digitalWrite(LOW);
315         }
316
317         nrf24_sck_digitalWrite(HIGH);
318
319         rx = rx << 1;
320         if(nrf24_miso_digitalRead())
321         {
322             rx |= 0x01;
323         }
324
325         nrf24_sck_digitalWrite(LOW);
326
327     }
328
329     return rx;
330 }
331
332 /* send and receive multiple bytes over SPI */
333 void nrf24_transferSync(uint8_t* dataout,uint8_t* datain,uint8_t len)
334 {
335     uint8_t i;
336
337     for(i=0;i<len;i++)
338     {
339         datain[i] = spi_transfer(dataout[i]);
340     }
341 }
342
343
344 /* send multiple bytes over SPI */
345 void nrf24_transmitSync(uint8_t* dataout,uint8_t len)
346 {
347     uint8_t i;
348
349     for(i=0;i<len;i++)
350     {
351         spi_transfer(dataout[i]);
352     }
353 }
354 }
355
```

```
356 /* Clocks only one byte into the given nrf24 register */
357 void nrf24_configRegister(uint8_t reg, uint8_t value)
358 {
359     nrf24_csn_digitalWrite(LOW);
360     spi_transfer(W_REGISTER | (REGISTER_MASK & reg));
361     spi_transfer(value);
362     nrf24_csn_digitalWrite(HIGH);
363 }
364
365 /* Read single register from nrf24 */
366 void nrf24_readRegister(uint8_t reg, uint8_t* value, uint8_t len)
367 {
368     nrf24_csn_digitalWrite(LOW);
369     spi_transfer(R_REGISTER | (REGISTER_MASK & reg));
370     nrf24_transferSync(value,value,len);
371     nrf24_csn_digitalWrite(HIGH);
372 }
373
374 /* Write to a single register of nrf24 */
375 void nrf24_writeRegister(uint8_t reg, uint8_t* value, uint8_t len)
376 {
377     nrf24_csn_digitalWrite(LOW);
378     spi_transfer(W_REGISTER | (REGISTER_MASK & reg));
379     nrf24_transmitSync(value,len);
380     nrf24_csn_digitalWrite(HIGH);
381 }
382
383 /* Check single register from nrf24 */
384 bool nrf24_checkRegister(uint8_t reg, uint8_t desiredValue, uint8_t len)
385 {
386     uint8_t registerValue;
387     nrf24_readRegister(reg,&registerValue,len);
388     if (registerValue==desiredValue) { return true; } else { return false; }
389 }
390
391 #define RF_DDR  DDRC
392 #define RF_PORT PORTC
393 #define RF_PIN  PINC
394
395 #define set_bit(reg,bit) reg |= (1<<bit)
396 #define clr_bit(reg,bit) reg &= ~(1<<bit)
397 #define check_bit(reg,bit) (reg&(1<<bit))
398
399 /* ----- */
400
401 void nrf24_setupPins()
402 {
403     set_bit(RF_DDR,0); // CE output
404     set_bit(RF_DDR,1); // CSN output
405     set_bit(RF_DDR,2); // SCK output
406     set_bit(RF_DDR,3); // MOSI output
407     clr_bit(RF_DDR,4); // MISO input
```



```
408 }
409 /* ----- */
410 void nrf24_ce_digitalWrite(uint8_t state)
411 {
412     if(state)
413     {
414         set_bit(RF_PORT,0);
415     }
416     else
417     {
418         clr_bit(RF_PORT,0);
419     }
420 }
421 /* ----- */
422 void nrf24_csn_digitalWrite(uint8_t state)
423 {
424     if(state)
425     {
426         set_bit(RF_PORT,1);
427     }
428     else
429     {
430         clr_bit(RF_PORT,1);
431     }
432 }
433 /* ----- */
434 void nrf24_sck_digitalWrite(uint8_t state)
435 {
436     if(state)
437     {
438         set_bit(RF_PORT,2);
439     }
440     else
441     {
442         clr_bit(RF_PORT,2);
443     }
444 }
445 /* ----- */
446 void nrf24_mosi_digitalWrite(uint8_t state)
447 {
448     if(state)
449     {
450         set_bit(RF_PORT,3);
451     }
452     else
453     {
454         clr_bit(RF_PORT,3);
455     }
456 }
457 /* ----- */
458 uint8_t nrf24_miso_digitalRead()
459 {
```

```
460     return check_bit(RF_PIN,4);
461 }
462 /* ----- */
463
464 void nrf24_initRF_SAFE(uint8_t boardIndex,TransmissionMode initMode){
465     initliazeMemory();
466     bool successfulRfInit = false;
467
468     while(successfulRfInit==false){
469         nrf24_powerDown();
470         nrf24_init();
471         nrf24_config(GENERAL_RF_CHANNEL,32);
472         if (nrf24_checkConfig()) { successfulRfInit = true; } else
473             { faultyRF_Alarm(); }
474     }
475
476     if (initMode==TRANSMIT){
477         nrf24_tx_address(CURRENT_BOARD_ADDRESS);
478         nrf24_rx_address(BOARD_ADDRESS[boardIndex]);
479     }else{
480         nrf24_tx_address(BOARD_ADDRESS[boardIndex]);
481         nrf24_rx_address(CURRENT_BOARD_ADDRESS);
482     }
483     nrf24_powerUpRx();
484 }
```