```
1
2  #define UCPHA0 1
3  #define BAUD_RATE 38400UL
4  #define UBRR_VALUE ((F_CPU)/(2UL*BAUD_RATE))-1
5
6  #include "nrf24.h"
7
8  uint8_t payload_len;
9  uint8_t selectedChannel;
10
11 uint8_t MOTORIZED_BOARD_ADDR[5] =   {0xF0,0xF0,0xF0,0xF0,0xC9};
12 uint8_t MAIN_BOARD_ADDR[5] =           {0xA4,0xA4,0xA4,0xA4,0xA4};
13 uint8_t POWER_BOARD_ADDR[5] =       {0xF0,0xF0,0xF0,0xF0,0xF0};
14
15 uint8_t* BOARD_ADDRESS[3] = {&MAIN_BOARD_ADDR[0], &POWER_BOARD_ADDR[0],
     &MOTORIZED_BOARD_ADDR[0]};
16 uint8_t* CURRENT_BOARD_ADDRESS = &POWER_BOARD_ADDR[0];
17
18 uint8_t GENERAL_RF_CHANNEL = 112;
19
20
21
22 void nrf24_init()
23 {
24     nrf24_setupPins();
25     nrf24_ce_digitalWrite(LOW);
26     nrf24_csn_digitalWrite(HIGH);
27 }
28
29 void nrf24_config(uint8_t channel, uint8_t pay_length)
30 {
31     /* Use static payload length ... */
32     payload_len = pay_length;
33     selectedChannel = channel;
34     // Set RF channel
35     nrf24_configRegister(RF_CH,channel);
36     // Set length of incoming payload
37     nrf24_configRegister(RX_PW_P0, 0x00); // Auto-ACK pipe ...
38     nrf24_configRegister(RX_PW_P1, payload_len); // Data payload pipe
39     nrf24_configRegister(RX_PW_P2, 0x00); // Pipe not used
40     nrf24_configRegister(RX_PW_P3, 0x00); // Pipe not used
41     nrf24_configRegister(RX_PW_P4, 0x00); // Pipe not used
42     nrf24_configRegister(RX_PW_P5, 0x00); // Pipe not used
43     // 1 Mbps, TX gain: 0dbm
44     nrf24_configRegister(RF_SETUP, (0<<RF_DR)|((0x03)<<RF_PWR));
45     // CRC enable, 1 byte CRC length
46     nrf24_configRegister(CONFIG,nrf24_CONFIG);
47     // Auto Acknowledgment
48     nrf24_configRegister(EN_AA,(1<<ENAA_P0)|(1<<ENAA_P1)|(0<<ENAA_P2)|
         (0<<ENAA_P3)|(0<<ENAA_P4)|(0<<ENAA_P5));
49     // Enable RX addresses
50     nrf24_configRegister(EN_RXADDR,(1<<ERX_P0)|(1<<ERX_P1)|(0<<ERX_P2)|
```

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

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

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

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

```
252     /* Transmission went OK */
253     if((rv & ((1 << TX_DS))))
254     {
255         return NRF24_TRANSMISSON_OK;
256     }
257     /* Maximum retransmission count is reached */
258     /* Last message probably went missing ... */
259     else if((rv & ((1 << MAX_RT))))
260     {
261         return NRF24_MESSAGE_LOST;
262     }
263     /* Probably still sending ... */
264     else
265     {
266         return 0xFF;
267     }
268 }
269
270 void nrf24_powerUpRx()
271 {
272     nrf24_csn_digitalWrite(LOW);
273     spi_transfer(FLUSH_RX);
274     nrf24_csn_digitalWrite(HIGH);
275
276     nrf24_configRegister(STATUS,(1<<RX_DR)|(1<<TX_DS)|(1<<MAX_RT));
277
278     nrf24_ce_digitalWrite(LOW);
279     nrf24_configRegister(CONFIG,nrf24_CONFIG|((1<<PWR_UP)|(1<<PRIM_RX)));
280     nrf24_ce_digitalWrite(HIGH);
281 }
282
283 void nrf24_powerUpTx()
284 {
285     nrf24_configRegister(STATUS,(1<<RX_DR)|(1<<TX_DS)|(1<<MAX_RT));
286
287     nrf24_configRegister(CONFIG,nrf24_CONFIG|((1<<PWR_UP)|(0<<PRIM_RX)));
288 }
289
290 void nrf24_powerDown()
291 {
292     nrf24_ce_digitalWrite(LOW);
293     nrf24_configRegister(CONFIG,nrf24_CONFIG);
294 }
295
296 uint8_t spi_transfer(uint8_t tx)
297 {
298     uint8_t i = 0;
299     uint8_t rx = 0;
300
301     nrf24_sck_digitalWrite(LOW);
302
303     for(i=0;i<8;i++)
```

```
304     {
305 
306         if(tx & (1<<(7-i)))
307         {
308             nrf24_mosi_digitalWrite(HIGH);
309         }
310         else
311         {
312             nrf24_mosi_digitalWrite(LOW);
313         }
314 
315         nrf24_sck_digitalWrite(HIGH);
316 
317         rx = rx << 1;
318         if(nrf24_miso_digitalRead())
319         {
320             rx |= 0x01;
321         }
322 
323         nrf24_sck_digitalWrite(LOW);
324 
325     }
326 
327     return rx;
328 }
329 
330 /* send and receive multiple bytes over SPI */
331 void nrf24_transferSync(uint8_t* dataout,uint8_t* datain,uint8_t len)
332 {
333     uint8_t i;
334 
335     for(i=0;i<len;i++)
336     {
337         datain[i] = spi_transfer(dataout[i]);
338     }
339 
340 }
341 
342 /* send multiple bytes over SPI */
343 void nrf24_transmitSync(uint8_t* dataout,uint8_t len)
344 {
345     uint8_t i;
346 
347     for(i=0;i<len;i++)
348     {
349         spi_transfer(dataout[i]);
350     }
351 
352 }
353 
354 /* Clocks only one byte into the given nrf24 register */
355 void nrf24_configRegister(uint8_t reg, uint8_t value)
```

```
356 {
357     nrf24_csn_digitalWrite(LOW);
358     spi_transfer(W_REGISTER | (REGISTER_MASK & reg));
359     spi_transfer(value);
360     nrf24_csn_digitalWrite(HIGH);
361 }
362
363 /* Read single register from nrf24 */
364 void nrf24_readRegister(uint8_t reg, uint8_t* value, uint8_t len)
365 {
366     nrf24_csn_digitalWrite(LOW);
367     spi_transfer(R_REGISTER | (REGISTER_MASK & reg));
368     nrf24_transferSync(value,value,len);
369     nrf24_csn_digitalWrite(HIGH);
370 }
371
372 /* Write to a single register of nrf24 */
373 void nrf24_writeRegister(uint8_t reg, uint8_t* value, uint8_t len)
374 {
375     nrf24_csn_digitalWrite(LOW);
376     spi_transfer(W_REGISTER | (REGISTER_MASK & reg));
377     nrf24_transmitSync(value,len);
378     nrf24_csn_digitalWrite(HIGH);
379 }
380
381 /* Check single register from nrf24 */
382 bool nrf24_checkRegister(uint8_t reg, uint8_t desiredValue, uint8_t len)
383 {
384     uint8_t registerValue;
385     nrf24_readRegister(reg,&registerValue,len);
386     if (registerValue==desiredValue) { return true; } else { return false; }
387 }
388
389 #define RF_DDR  DDRD
390 #define RF_PORT PORTD
391 #define RF_PIN  PIND
392
393 #define CE_CSN_DDR  DDRC
394 #define CE_CSN_PORT PORTC
395 #define CE_CSN_PIN  PINC
396
397 #define MISO_BIT_POS    0
398 #define MOSI_BIT_POS    1
399 #define SCK_BIT_POS     4
400
401 #define CE_BIT_POS      0
402 #define CSN_BIT_POS     1
403
404 #define set_bit(reg,bit) reg |= (1<<bit)
405 #define clr_bit(reg,bit) reg &= ~(1<<bit)
406 #define check_bit(reg,bit) (reg&(1<<bit))
407
```

```
408 /* ------------------------------------------------------------------------ */
409
410 void nrf24_setupPins()
411 {
412     set_bit(CE_CSN_DDR, CE_BIT_POS); // CE output
413     set_bit(CE_CSN_DDR, CSN_BIT_POS); // CSN output
414
415     clr_bit(RF_DDR, MISO_BIT_POS); // MISO input
416     set_bit(RF_DDR, MOSI_BIT_POS); // MOSI output
417     set_bit(RF_DDR, SCK_BIT_POS); // SCK output
418 }
419 /* ------------------------------------------------------------------------ */
420 void nrf24_ce_digitalWrite(uint8_t state)
421 {
422     if(state)
423     {
424         set_bit(CE_CSN_PORT, CE_BIT_POS);
425     }
426     else
427     {
428         clr_bit(CE_CSN_PORT, CE_BIT_POS);
429     }
430 }
431 /* ------------------------------------------------------------------------ */
432 void nrf24_csn_digitalWrite(uint8_t state)
433 {
434     if(state)
435     {
436         set_bit(CE_CSN_PORT, CSN_BIT_POS);
437     }
438     else
439     {
440         clr_bit(CE_CSN_PORT, CSN_BIT_POS);
441     }
442 }
443 /* ------------------------------------------------------------------------ */
444 void nrf24_sck_digitalWrite(uint8_t state)
445 {
446     if(state)
447     {
448         set_bit(RF_PORT, SCK_BIT_POS);
449     }
450     else
451     {
452         clr_bit(RF_PORT, SCK_BIT_POS);
453     }
454 }
455 /* ------------------------------------------------------------------------ */
456 void nrf24_mosi_digitalWrite(uint8_t state)
457 {
458     if(state)
459     {
```

```
460         set_bit(RF_PORT, MOSI_BIT_POS);
461     }
462     else
463     {
464         clr_bit(RF_PORT, MOSI_BIT_POS);
465     }
466 }
467 /* ------------------------------------------------------------------------ */
468 uint8_t nrf24_miso_digitalRead()
469 {
470     return check_bit(RF_PIN, MISO_BIT_POS);
471 }
472 /* ------------------------------------------------------------------------ */
473
474
475 void nrf24_initRF_SAFE(uint8_t boardIndex,TransmissionMode initMode){
476
477     initliazeMemory();
478     bool successfulRfInit = false;
479
480     while(successfulRfInit==false){
481         nrf24_powerDown();
482         nrf24_init();
483         nrf24_config(GENERAL_RF_CHANNEL,32);
484         if (nrf24_checkConfig()) { successfulRfInit = true; } else
              { faultyRF_Alarm(); }
485     }
486
487     if (initMode==TRANSMIT){
488         nrf24_tx_address(CURRENT_BOARD_ADDRESS);
489         nrf24_rx_address(BOARD_ADDRESS[boardIndex]);
490         }else{
491         nrf24_tx_address(BOARD_ADDRESS[boardIndex]);
492         nrf24_rx_address(CURRENT_BOARD_ADDRESS);
493     }
494     nrf24_powerUpRx();
495 }
```