```
1
2  #define UCPHA0 1
3
4  #include "nrf24.h"
5  #include "UART_Bluetooth.h"
6
7  volatile uint8_t payload_len;
8  volatile uint8_t selectedChannel;
9
10 uint8_t MOTORIZED_BOARD_ADDR[5] =   {0xF0,0xF0,0xF0,0xF0,0xC9};
11 uint8_t MAIN_BOARD_ADDR[5] =           {0xA4,0xA4,0xA4,0xA4,0xA4};
12 uint8_t POWER_BOARD_ADDR[5] =       {0xF0,0xF0,0xF0,0xF0,0xF0};
13
14 uint8_t NULL_ADDR[5] =      {0x00,0x00,0x00,0x00,0x00};
15
16 uint8_t* BOARD_ADDRESS[3] = {&MAIN_BOARD_ADDR[0], &POWER_BOARD_ADDR[0],
     &MOTORIZED_BOARD_ADDR[0]};
17 uint8_t* CURRENT_BOARD_ADDRESS = &MAIN_BOARD_ADDR[0];
18
19 const uint8_t GENERAL_RF_CHANNEL = 112;
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
35     // Set RF channel
36     nrf24_configRegister(RF_CH,channel);
37
38     // Set length of incoming payload
39     nrf24_configRegister(RX_PW_P0, 0x00); // Auto-ACK pipe ...
40     nrf24_configRegister(RX_PW_P1, payload_len); // Data payload pipe
41     nrf24_configRegister(RX_PW_P2, 0x00); // Pipe not used
42     nrf24_configRegister(RX_PW_P3, 0x00); // Pipe not used
43     nrf24_configRegister(RX_PW_P4, 0x00); // Pipe not used
44     nrf24_configRegister(RX_PW_P5, 0x00); // Pipe not used
45
46     // 1 Mbps, TX gain: 0dbm
47     nrf24_configRegister(RF_SETUP, (0<<RF_DR)|((0x03)<<RF_PWR));
48
49     // CRC enable, 1 byte CRC length
50     nrf24_configRegister(CONFIG,nrf24_CONFIG);
51
```

```
52      // Auto Acknowledgment
53      nrf24_configRegister(EN_AA,(1<<ENAA_P0)|(1<<ENAA_P1)|(0<<ENAA_P2)|
          (0<<ENAA_P3)|(0<<ENAA_P4)|(0<<ENAA_P5));
54
55      // Enable RX addresses
56      nrf24_configRegister(EN_RXADDR,(1<<ERX_P0)|(1<<ERX_P1)|(0<<ERX_P2)|
          (0<<ERX_P3)|(0<<ERX_P4)|(0<<ERX_P5));
57
58      // Auto retransmit delay: 1000 us and Up to 15 retransmit trials
59      nrf24_configRegister(SETUP_RETR,(0x04<<ARD)|(0x0F<<ARC));
60
61      // Dynamic length configurations: No dynamic length
62      nrf24_configRegister(DYNPD,(0<<DPL_P0)|(0<<DPL_P1)|(0<<DPL_P2)|(0<<DPL_P3)|
          (0<<DPL_P4)|(0<<DPL_P5));
63
64  }
65
66
67
68  bool nrf24_checkConfig(){
69      // Check all registers
70      if (nrf24_checkRegister(RF_CH, selectedChannel,1)==false) return false;
71      if (nrf24_checkRegister(RF_SETUP, (0<<RF_DR)|((0x03)<<RF_PWR),1)==false)
          return false;
72      if (nrf24_checkRegister(CONFIG,nrf24_CONFIG,1)==false) return false;
73      if (nrf24_checkRegister(SETUP_RETR,(0x04<<ARD)|(0x0F<<ARC),1)==false) return
          false;
74      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;
75
76      return true;
77  }
78
79  bool nrf24_checkAvailability(){
80      if (nrf24_checkRegister(RF_CH, selectedChannel,1)==true) { return true; }
          else { return false;}
81  }
82
83
84
85
86  void faultyRF_Alarm(){
87      CLEAR_FAULTY_RF_LED;
88      for (uint8_t x = 0; x < 6; x++)
89      {
90          FLIP_FAULTY_RF_LED;
91          _delay_ms(125);
92      }
93      _delay_ms(250);
94  }
95
96
```

```
 97
 98 /* Set the RX address */
 99 void nrf24_rx_address(uint8_t * adr)
100 {
101     nrf24_ce_digitalWrite(LOW);
102     nrf24_writeRegister(RX_ADDR_P1,adr,nrf24_ADDR_LEN);
103     nrf24_ce_digitalWrite(HIGH);
104 }
105
106 /* Set the secondary RX address */
107 void nrf24_secondary_rx_address(uint8_t * adr)
108 {
109     nrf24_ce_digitalWrite(LOW);
110     nrf24_writeRegister(RX_ADDR_P2,adr,1);  // One byte long
111     nrf24_ce_digitalWrite(HIGH);
112 }
113
114
115 /* Returns the payload length */
116 uint8_t nrf24_payload_length()
117 {
118     return payload_len;
119 }
120
121 /* Set the TX address */
122 void nrf24_tx_address(uint8_t* adr)
123 {
124     /* RX_ADDR_P0 must be set to the sending addr for auto ack to work. */
125     nrf24_writeRegister(RX_ADDR_P0,adr,nrf24_ADDR_LEN);
126     nrf24_writeRegister(TX_ADDR,adr,nrf24_ADDR_LEN);
127 }
128
129 /* Checks if data is available for reading */
130 /* Returns 1 if data is ready ... */
131 uint8_t nrf24_dataReady()
132 {
133     // See note in getData() function - just checking RX_DR isn't good enough
134     uint8_t status = nrf24_getStatus();
135
136     // We can short circuit on RX_DR, but if it's not set, we still need
137     // to check the FIFO for any pending packets
138     if ( status & (1 << RX_DR) )
139     {
140         return 1;
141     }
142
143     return !nrf24_rxFifoEmpty();;
144 }
145
146 /* Checks if receive FIFO is empty or not */
147 uint8_t nrf24_rxFifoEmpty()
148 {
```

```
149     uint8_t fifoStatus;
150
151     nrf24_readRegister(FIFO_STATUS,&fifoStatus,1);
152
153     return (fifoStatus & (1 << RX_EMPTY));
154 }
155
156 /* Returns the length of data waiting in the RX fifo */
157 uint8_t nrf24_payloadLength()
158 {
159     uint8_t status;
160     nrf24_csn_digitalWrite(LOW);
161     spi_transfer(R_RX_PL_WID);
162     status = spi_transfer(0x00);
163     nrf24_csn_digitalWrite(HIGH);
164     return status;
165 }
166
167 /* Reads payload bytes into data array */
168 void nrf24_getData(uint8_t* data)
169 {
170     /* Pull down chip select */
171     nrf24_csn_digitalWrite(LOW);
172
173     /* Send cmd to read rx payload */
174     spi_transfer( R_RX_PAYLOAD );
175
176     /* Read payload */
177     nrf24_transferSync(data,data,payload_len);
178
179     /* Pull up chip select */
180     nrf24_csn_digitalWrite(HIGH);
181
182     /* Reset status register */
183     nrf24_configRegister(STATUS,(1<<RX_DR));
184 }
185
186 /* Returns the number of retransmissions occured for the last message */
187 uint8_t nrf24_retransmissionCount()
188 {
189     uint8_t rv;
190     nrf24_readRegister(OBSERVE_TX,&rv,1);
191     rv = rv & 0x0F;
192     return rv;
193 }
194
195 // Sends a data package to the default address. Be sure to send the correct
196 // amount of bytes as configured as payload on the receiver.
197 void nrf24_send(uint8_t* value)
198 {
199     /* Go to Standby-I first */
200     nrf24_ce_digitalWrite(LOW);
```

```
201
202     /* Set to transmitter mode , Power up if needed */
203     nrf24_powerUpTx();
204
205     /* Do we really need to flush TX fifo each time ? */
206     #if 1
207     /* Pull down chip select */
208     nrf24_csn_digitalWrite(LOW);
209
210     /* Write cmd to flush transmit FIFO */
211     spi_transfer(FLUSH_TX);
212
213     /* Pull up chip select */
214     nrf24_csn_digitalWrite(HIGH);
215     #endif
216
217     /* Pull down chip select */
218     nrf24_csn_digitalWrite(LOW);
219
220     /* Write cmd to write payload */
221     spi_transfer(W_TX_PAYLOAD);
222
223     /* Write payload */
224     nrf24_transmitSync(value,payload_len);
225
226     /* Pull up chip select */
227     nrf24_csn_digitalWrite(HIGH);
228
229     /* Start the transmission */
230     nrf24_ce_digitalWrite(HIGH);
231 }
232
233 uint8_t nrf24_isSending()
234 {
235     uint8_t status;
236
237     /* read the current status */
238     status = nrf24_getStatus();
239
240     /* if sending successful (TX_DS) or max retries exceded (MAX_RT). */
241     if((status & ((1 << TX_DS)  | (1 << MAX_RT))))
242     {
243         return 0; /* false */
244     }
245
246     return 1; /* true */
247
248 }
249
250 uint8_t nrf24_getStatus()
251 {
252     uint8_t rv;
```

```
253     nrf24_csn_digitalWrite(LOW);
254     rv = spi_transfer(NOP);
255     nrf24_csn_digitalWrite(HIGH);
256     return rv;
257 }
258 
259 uint8_t nrf24_lastMessageStatus()
260 {
261     uint8_t rv;
262 
263     rv = nrf24_getStatus();
264 
265     /* Transmission went OK */
266     if((rv & ((1 << TX_DS))))
267     {
268         return NRF24_TRANSMISSON_OK;
269     }
270     /* Maximum retransmission count is reached */
271     /* Last message probably went missing ... */
272     else if((rv & ((1 << MAX_RT))))
273     {
274         return NRF24_MESSAGE_LOST;
275     }
276     /* Probably still sending ... */
277     else
278     {
279         return 0xFF;
280     }
281 }
282 
283 void nrf24_powerUpRx()
284 {
285     nrf24_csn_digitalWrite(LOW);
286     spi_transfer(FLUSH_RX);
287     nrf24_csn_digitalWrite(HIGH);
288 
289     nrf24_configRegister(STATUS,(1<<RX_DR)|(1<<TX_DS)|(1<<MAX_RT));
290 
291     nrf24_ce_digitalWrite(LOW);
292     nrf24_configRegister(CONFIG,nrf24_CONFIG|((1<<PWR_UP)|(1<<PRIM_RX)));
293     nrf24_ce_digitalWrite(HIGH);
294 }
295 
296 void nrf24_powerUpTx()
297 {
298     nrf24_configRegister(STATUS,(1<<RX_DR)|(1<<TX_DS)|(1<<MAX_RT));
299 
300     nrf24_configRegister(CONFIG,nrf24_CONFIG|((1<<PWR_UP)|(0<<PRIM_RX)));
301 }
302 
303 void nrf24_powerDown()
304 {
```

```
305     nrf24_ce_digitalWrite(LOW);
306     nrf24_configRegister(CONFIG,nrf24_CONFIG);
307 }
308 
309 uint8_t spi_transfer(uint8_t tx)
310 {
311     uint8_t i = 0;
312     uint8_t rx = 0;
313 
314     nrf24_sck_digitalWrite(LOW);
315 
316     for(i=0;i<8;i++)
317     {
318 
319         if(tx & (1<<(7-i)))
320         {
321             nrf24_mosi_digitalWrite(HIGH);
322         }
323         else
324         {
325             nrf24_mosi_digitalWrite(LOW);
326         }
327 
328         nrf24_sck_digitalWrite(HIGH);
329 
330         rx = rx << 1;
331         if(nrf24_miso_digitalRead())
332         {
333             rx |= 0x01;
334         }
335 
336         nrf24_sck_digitalWrite(LOW);
337 
338     }
339 
340     return rx;
341 }
342 
343 /* send and receive multiple bytes over SPI */
344 void nrf24_transferSync(uint8_t* dataout,uint8_t* datain,uint8_t len)
345 {
346     uint8_t i;
347 
348     for(i=0;i<len;i++)
349     {
350         datain[i] = spi_transfer(dataout[i]);
351     }
352 
353 }
354 
355 /* send multiple bytes over SPI */
356 void nrf24_transmitSync(uint8_t* dataout,uint8_t len)
```

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

```
409
410 /* ---------------------------------------------------------------------- */
411
412 void nrf24_setupPins()
413 {
414     set_bit(RF_DDR,0); // CE output
415     set_bit(RF_DDR,1); // CSN output
416     set_bit(RF_DDR,2); // SCK output
417     set_bit(RF_DDR,3); // MOSI output
418     clr_bit(RF_DDR,4); // MISO input
419 }
420 /* ---------------------------------------------------------------------- */
421 void nrf24_ce_digitalWrite(uint8_t state)
422 {
423     if(state)
424     {
425         set_bit(RF_PORT,0);
426     }
427     else
428     {
429         clr_bit(RF_PORT,0);
430     }
431 }
432 /* ---------------------------------------------------------------------- */
433 void nrf24_csn_digitalWrite(uint8_t state)
434 {
435     if(state)
436     {
437         set_bit(RF_PORT,1);
438     }
439     else
440     {
441         clr_bit(RF_PORT,1);
442     }
443 }
444 /* ---------------------------------------------------------------------- */
445 void nrf24_sck_digitalWrite(uint8_t state)
446 {
447     if(state)
448     {
449         set_bit(RF_PORT,2);
450     }
451     else
452     {
453         clr_bit(RF_PORT,2);
454     }
455 }
456 /* ---------------------------------------------------------------------- */
457 void nrf24_mosi_digitalWrite(uint8_t state)
458 {
459     if(state)
460     {
```

```
461         set_bit(RF_PORT,3);
462     }
463     else
464     {
465         clr_bit(RF_PORT,3);
466     }
467 }
468 /* ------------------------------------------------------------------------ */
469 uint8_t nrf24_miso_digitalRead()
470 {
471     return check_bit(RF_PIN,4);
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
488
489     if (initMode==RECEIVE){
490         nrf24_tx_address(CURRENT_BOARD_ADDRESS);
491         nrf24_rx_address(BOARD_ADDRESS[boardIndex]);
492         }else{
493         nrf24_tx_address(BOARD_ADDRESS[boardIndex]);
494         nrf24_rx_address(CURRENT_BOARD_ADDRESS);
495     }
496
497
498     nrf24_powerUpRx();
499 }
```