```
 1
 2  #define F_CPU 16000000UL
 3
 4  #define bit_get(p,m) ((p) & (m))
 5  #define bit_set(p,m) ((p) |= (m))
 6  #define bit_clear(p,m) ((p) &= ~(m))
 7  #define bit_flip(p,m) ((p) ^= (m))
 8  #define bit_write(c,p,m) (c ? bit_set(p,m) : bit_clear(p,m))
 9  #define BIT(x) (0x01 << (x))
10  #define LONGBIT(x) ((unsigned long)0x00000001 << (x))
11
12  #include "nrf24.h"
13  #include "Command_Handler.h"
14
15  #include <avr/io.h>
16  #include <string.h>
17  #include <stdlib.h>
18  #include <util/delay.h>
19
20  bool initRF();
21  void initIO();
22  void faultyRF_Alarm();
23
24  int main(void)
25  {
26      initIO();
27      initRF();
28
29      while (1)
30      {
31          if(nrf24_dataReady())
32          {
33              bit_clear(PORTB, BIT(0));
34
35              nrf24_getData(command_buffer);
36
37              bit_set(PORTD, BIT(7));
38              _delay_ms(500);
39              commandType currentCommand;
40              bool success = decomposeCommand(command_buffer, &currentCommand,
                  parameter);
41              if (success) { currentCommand.handlerFunction(); }
42              bit_clear(PORTD, BIT(7));
43          }
44
45          if (nrf24_checkAvailability()==false) { while(initRF()==false); }
46      }
47  }
48
49  void initIO(){
50      /*
51          Input/Output pin initialization
```

```
52          1 : OUTPUT | 0 : INPUT | 0b76543210 Bit order
53          ATTACHMENTS
54              RELAY 0     : PD3               |   OUTPUT
55              RELAY 1     : PD2               |   OUTPUT
56              RELAY 2     : PD6               |   OUTPUT
57              RELAY 3     : PD5               |   OUTPUT
58              RED LED     : PD7               |   OUTPUT
59              GREEN LED   : PB0               |   OUTPUT
60          nRF24L01
61              CE   : PC0                      |   OUTPUT
62              CSN  : PC1                      |   OUTPUT
63              MISO : PD0 (MSPIM MISO ATMEGA)  |   INPUT
64              MOSI : PD1 (MSPIM MOSI ATMEGA)  |   OUTPUT
65              SCK  : PD4 (MSPIM XCK)          |   OUTPUT
66      */
67      DDRD = 0b11111110;
68      DDRB = 0b00101001;
69      DDRC = 0b11011111;
70
71      PORTD = 0b00000000;
72      PORTC = 0b00000000;
73      PORTB = 0b00000000;
74  }
75
76  bool initRF(){
77      uint8_t tx_address[5] = {0xD7,0xD7,0xD7,0xD7,0xD7};
78      uint8_t rx_address[5] = {0xE7,0xE7,0xE7,0xE7,0xE7};
79
80      initliazeMemory();
81
82      /* Power down module */
83      nrf24_powerDown();
84
85      nrf24_init();
86
87      /* Channel #112 , payload length: 32 */
88      nrf24_config(112,32);
89
90      /* Check module configuration */
91      if (nrf24_checkConfig()==false) { faultyRF_Alarm(); return false; }
92
93      /* Set the device addresses */
94      nrf24_tx_address(tx_address);
95      nrf24_rx_address(rx_address);
96
97      /* Power up in receive mode */
98      nrf24_powerUpRx();
99
100     return true;
101 }
102
103 void faultyRF_Alarm(){
```

```
104     bit_clear(PORTD, BIT(7));
105     for (uint8_t x = 0; x < 6; x++)
106     {
107         bit_flip(PORTD, BIT(7));
108         _delay_ms(125);
109 
110     }
111 }
112 
113 
114 
115 
```

```
1
2
3  #ifndef COMMAND_HANDLER_H_
4  #define COMMAND_HANDLER_H_
5
6  #include <stdbool.h>
7  #include <stdint.h>
8
9  #ifndef nullptr
10 #define nullptr ((void *)0)
11 #endif
12
13 #ifndef F_CPU
14 #define F_CPU               16000000UL
15 #endif
16
17 #define AVAILABLE_COMMANDS 6
18 #define COMMAND_BUFFER_SIZE 32
19 #define PARAMETER_BUFFER_SIZE 28
20
21 #ifndef BIT_MANIPULATION_MACRO
22 #define BIT_MANIPULATION_MACRO 1
23 #define bit_get(p,m) ((p) & (m))
24 #define bit_set(p,m) ((p) |= (m))
25 #define bit_clear(p,m) ((p) &= ~(m))
26 #define bit_flip(p,m) ((p) ^= (m))
27 #define bit_write(c,p,m) (c ? bit_set(p,m) : bit_clear(p,m))
28 #define BIT(x) (0x01 << (x))
29 #define LONGBIT(x) ((unsigned long)0x00000001 << (x))
30 #endif
31
32 typedef struct commandType {
33     const char *commandBase;
34     uint8_t nParameters;
35     void (*handlerFunction)();
36 } commandType;
37
38 void *parameter[3];
39 uint8_t *command_buffer;
40 extern bool initliazeMemory();
41 bool memoryInitialized;
42 extern void TURN_RELAY_ON_HANDLE(), TURN_RELAY_OFF_HANDLE(),
     BUILT_IN_LED_TEST_HANDLER(), TURN_EVERYTHING_ON_HANDLER(),
     TURN_EVERYTHING_OFF_HANDLER(), CALL_NURSE_HANDLE();
43
44 extern void composeCommand(void* output_buffer, commandType* commandT, void**
     inputParameter);
45 extern bool decomposeCommand(void* input_buffer, commandType* commandT, void**
     outputParameter);
46
47
48 #endif /* COMMAND_HANDLER_H_ */
```

```
1
2  #include "Command_Handler.h"
3  #include "nrf24.h"
4  #include <stdbool.h>
5  #include <string.h>
6  #include <stdlib.h>
7  #include <stdint.h>
8  #include <avr/io.h>
9  #include <util/delay.h>
10
11
12 const commandType availableCommand[AVAILABLE_COMMANDS] = {
13     { .commandBase = "TURN_RELAY_ON", .nParameters = 1, .handlerFunction =
       &TURN_RELAY_ON_HANDLE},
14     { .commandBase = "TURN_RELAY_OFF", .nParameters = 1, .handlerFunction =
       &TURN_RELAY_OFF_HANDLE},
15     { .commandBase = "BUILT_IN_LED_TEST", .nParameters = 0, .handlerFunction =
       &BUILT_IN_LED_TEST_HANDLER},
16     { .commandBase = "TURN_EVERYTHING_ON", .nParameters = 0, .handlerFunction =
       &TURN_EVERYTHING_ON_HANDLER},
17     { .commandBase = "TURN_EVERYTHING_OFF", .nParameters = 0, .handlerFunction =
       &TURN_EVERYTHING_OFF_HANDLER},
18     { .commandBase = "CALL_NURSE", .nParameters = 0, .handlerFunction =
       &CALL_NURSE_HANDLE}
19 };
20
21 bool initliazeMemory(){
22     if(memoryInitialized) return false;
23     parameter[0] = (void*)calloc(28,1);
24     parameter[1] = (void*)calloc(28,1);
25     parameter[2] = (void*)calloc(28,1);
26     command_buffer = (uint8_t*)calloc(32,1);
27     if(parameter[0]==nullptr||parameter[1]==nullptr||parameter[2]==nullptr||
       command_buffer==nullptr) return false;
28     memoryInitialized = true;
29     return true;
30 }
31
32
33 void composeCommand(void* output_buffer, commandType* commandT, void**
     inputParameter){
34     strcpy(output_buffer, commandT->commandBase);
35     char* startParamPTR = (char*)(output_buffer+strlen(commandT->commandBase));
36     char* endParamPTR = (char*)(startParamPTR+1+strlen(*inputParameter));
37
38     for (uint8_t index = 0; index < commandT->nParameters; index++){
39         *startParamPTR='[';
40         strcpy(startParamPTR+1, *inputParameter);
41         *endParamPTR=']';
42         startParamPTR=(endParamPTR+1);
43         if (index!=(commandT->nParameters-1)){
44             inputParameter++;
```

```
45              uint8_t len = strlen(*inputParameter);
46              endParamPTR = (char*)(startParamPTR+len+1);
47          }
48      }
49      *startParamPTR='\0';
50  }
51
52  bool decomposeCommand(void* input_buffer, commandType* commandT, void**
      outputParameter){
53
54      for (uint8_t index = 0; index < AVAILABLE_COMMANDS; index++){
55          if (memmem(input_buffer, COMMAND_BUFFER_SIZE, availableCommand
              [index].commandBase, strlen(availableCommand[index].commandBase))!
              =nullptr)
56          {
57              *commandT = availableCommand[index]; break;
58          }
59          else if (index==(AVAILABLE_COMMANDS-1)) { return false;}
60      }
61
62      for (uint8_t x = 0; x < commandT->nParameters; x++){
63          uint8_t* startNumPTR = memchr(input_buffer, '[', COMMAND_BUFFER_SIZE);
64          uint8_t* endNumPTR = memchr(input_buffer, ']', COMMAND_BUFFER_SIZE);
65          if (startNumPTR==nullptr||endNumPTR==nullptr) { if(x==0) return false;
              break; }
66          (*startNumPTR) = 0x20;
67          (*endNumPTR) = 0x20;
68          startNumPTR++;
69          uint32_t bytes = ((endNumPTR)) - ((startNumPTR));
70          if (bytes>PARAMETER_BUFFER_SIZE) return false;
71          memcpy(outputParameter[x], startNumPTR, bytes);
72      }
73
74      return true;
75  }
76
77  void TURN_RELAY_ON_HANDLE() {
78      uint8_t relayIndex =  atoi(parameter[0]);
79      switch (relayIndex) {
80          case 0:
81          bit_set(PORTD, BIT(3));
82          break;
83          case 1:
84          bit_set(PORTD, BIT(2));
85          break;
86          case 2:
87          bit_set(PORTD, BIT(6));
88          break;
89          case 3:
90          bit_set(PORTD, BIT(5));
91          break;
92      }
```

```
 93  }
 94
 95  void TURN_RELAY_OFF_HANDLE() {
 96      uint8_t relayIndex =  atoi(parameter[0]);
 97      switch (relayIndex) {
 98          case 0:
 99          bit_clear(PORTD, BIT(3));
100          break;
101          case 1:
102          bit_clear(PORTD, BIT(2));
103          break;
104          case 2:
105          bit_clear(PORTD, BIT(6));
106          break;
107          case 3:
108          bit_clear(PORTD, BIT(5));
109          break;
110      }
111  }
112
113  void BUILT_IN_LED_TEST_HANDLER(){
114      for (uint8_t x = 0; x < 8; x++) {
115          bit_flip(PORTD, BIT(7));
116          bit_flip(PORTB, BIT(0));
117          _delay_ms(250);
118      }
119      bit_clear(PORTD, BIT(7));
120      bit_clear(PORTB, BIT(0));
121  }
122
123  void TURN_EVERYTHING_ON_HANDLER(){
124      bit_set(PORTD, BIT(3));
125      bit_set(PORTD, BIT(2));
126      bit_set(PORTD, BIT(6));
127      bit_set(PORTD, BIT(5));
128  }
129
130  void TURN_EVERYTHING_OFF_HANDLER(){
131      bit_clear(PORTD, BIT(3));
132      bit_clear(PORTD, BIT(2));
133      bit_clear(PORTD, BIT(6));
134      bit_clear(PORTD, BIT(5));
135  }
136
137  void CALL_NURSE_HANDLE(){
138      bit_set(PORTD, BIT(5));
139      _delay_ms(500);
140      bit_clear(PORTD, BIT(5));
141      _delay_ms(500);
142      bit_set(PORTD, BIT(5));
143      _delay_ms(500);
144      bit_clear(PORTD, BIT(5));
```

```
145     _delay_ms(500);
146     bit_set(PORTD, BIT(5));
147     _delay_ms(500);
148     bit_clear(PORTD, BIT(5));
149 }
150
```

```
1  #ifndef NRF24
2  #define NRF24
3
4  #include "nRF24L01_Definitions.h"
5  #include <stdint.h>
6  #include <stdbool.h>
7  #include <util/delay.h>
8
9  #define LOW 0
10 #define HIGH 1
11
12 #define nrf24_ADDR_LEN 5
13 #define nrf24_CONFIG ((1<<EN_CRC)|(0<<CRCO))
14
15 #define NRF24_TRANSMISSON_OK 0
16 #define NRF24_MESSAGE_LOST   1
17
18 void    nrf24_init();
19 void    nrf24_rx_address(uint8_t* adr);
20 void    nrf24_tx_address(uint8_t* adr);
21 void    nrf24_config(uint8_t channel, uint8_t pay_length);
22 bool    nrf24_checkRegister(uint8_t reg, uint8_t desiredValue, uint8_t len);
23 bool    nrf24_checkConfig();
24 bool    nrf24_checkAvailability();
25
26
27 uint8_t nrf24_dataReady();
28 uint8_t nrf24_isSending();
29 uint8_t nrf24_getStatus();
30 uint8_t nrf24_rxFifoEmpty();
31
32 void    nrf24_send(uint8_t* value);
33 void    nrf24_getData(uint8_t* data);
34
35 uint8_t nrf24_payloadLength();
36
37 uint8_t nrf24_lastMessageStatus();
38 uint8_t nrf24_retransmissionCount();
39
40 uint8_t nrf24_payload_length();
41
42 void    nrf24_powerUpRx();
43 void    nrf24_powerUpTx();
44 void    nrf24_powerDown();
45
46 uint8_t spi_transfer(uint8_t tx);
47 void    nrf24_transmitSync(uint8_t* dataout,uint8_t len);
48 void    nrf24_transferSync(uint8_t* dataout,uint8_t* datain,uint8_t len);
49 void    nrf24_configRegister(uint8_t reg, uint8_t value);
50 void    nrf24_readRegister(uint8_t reg, uint8_t* value, uint8_t len);
51 void    nrf24_writeRegister(uint8_t reg, uint8_t* value, uint8_t len);
52
```

```
53  extern void nrf24_setupPins();
54
55  extern void nrf24_ce_digitalWrite(uint8_t state);
56
57  extern void nrf24_csn_digitalWrite(uint8_t state);
58
59  extern void nrf24_sck_digitalWrite(uint8_t state);
60
61  extern void nrf24_mosi_digitalWrite(uint8_t state);
62
63  extern uint8_t nrf24_miso_digitalRead();
64
65  #endif
66
```

```
 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  uint8_t selectedChannel;
12
13  void nrf24_init()
14  {
15      nrf24_setupPins();
16      nrf24_ce_digitalWrite(LOW);
17      nrf24_csn_digitalWrite(HIGH);
18  }
19
20  void nrf24_config(uint8_t channel, uint8_t pay_length)
21  {
22      /* Use static payload length ... */
23      payload_len = pay_length;
24      selectedChannel = channel;
25      // Set RF channel
26      nrf24_configRegister(RF_CH,channel);
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      // 1 Mbps, TX gain: 0dbm
35      nrf24_configRegister(RF_SETUP, (0<<RF_DR)|((0x03)<<RF_PWR));
36      // CRC enable, 1 byte CRC length
37      nrf24_configRegister(CONFIG,nrf24_CONFIG);
38      // Auto Acknowledgment
39      nrf24_configRegister(EN_AA,(1<<ENAA_P0)|(1<<ENAA_P1)|(0<<ENAA_P2)|
          (0<<ENAA_P3)|(0<<ENAA_P4)|(0<<ENAA_P5));
40      // Enable RX addresses
41      nrf24_configRegister(EN_RXADDR,(1<<ERX_P0)|(1<<ERX_P1)|(0<<ERX_P2)|
          (0<<ERX_P3)|(0<<ERX_P4)|(0<<ERX_P5));
42      // Auto retransmit delay: 1000 us and Up to 15 retransmit trials
43      nrf24_configRegister(SETUP_RETR,(0x04<<ARD)|(0x0F<<ARC));
44      // Dynamic length configurations: No dynamic length
45      nrf24_configRegister(DYNPD,(0<<DPL_P0)|(0<<DPL_P1)|(0<<DPL_P2)|(0<<DPL_P3)|
          (0<<DPL_P4)|(0<<DPL_P5));
46
47  }
48
49  bool nrf24_checkConfig(){
```

```
50      // Check all registers
51      if (nrf24_checkRegister(RF_CH, selectedChannel,1)==false) return false;
52      if (nrf24_checkRegister(RX_PW_P0, 0x00,1)==false) return false;
53      if (nrf24_checkRegister(RX_PW_P1, payload_len,1)==false) return false;
54      if (nrf24_checkRegister(RX_PW_P2, 0x00,1)==false) return false;
55      if (nrf24_checkRegister(RX_PW_P3, 0x00,1)==false) return false;
56      if (nrf24_checkRegister(RX_PW_P4, 0x00,1)==false) return false;
57      if (nrf24_checkRegister(RX_PW_P5, 0x00,1)==false) return false;
58      if (nrf24_checkRegister(RF_SETUP, (0<<RF_DR)|((0x03)<<RF_PWR),1)==false)
          return false;
59      if (nrf24_checkRegister(CONFIG,nrf24_CONFIG,1)==false) return false;
60      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;
61      if (nrf24_checkRegister(SETUP_RETR,(0x04<<ARD)|(0x0F<<ARC),1)==false) return
          false;
62      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;
63
64      return true;
65  }
66
67  bool nrf24_checkAvailability(){
68      if (nrf24_checkRegister(RF_CH, selectedChannel,1)==true) { return true; }
          else { return false;}
69  }
70
71
72
73  /* Set the RX address */
74  void nrf24_rx_address(uint8_t * adr)
75  {
76      nrf24_ce_digitalWrite(LOW);
77      nrf24_writeRegister(RX_ADDR_P1,adr,nrf24_ADDR_LEN);
78      nrf24_ce_digitalWrite(HIGH);
79  }
80
81  /* Returns the payload length */
82  uint8_t nrf24_payload_length()
83  {
84      return payload_len;
85  }
86
87  /* Set the TX address */
88  void nrf24_tx_address(uint8_t* adr)
89  {
90      /* RX_ADDR_P0 must be set to the sending addr for auto ack to work. */
91      nrf24_writeRegister(RX_ADDR_P0,adr,nrf24_ADDR_LEN);
92      nrf24_writeRegister(TX_ADDR,adr,nrf24_ADDR_LEN);
93  }
94
95  /* Checks if data is available for reading */
96  /* Returns 1 if data is ready ... */
```

```
 97  uint8_t nrf24_dataReady()
 98  {
 99      // See note in getData() function - just checking RX_DR isn't good enough
100      uint8_t status = nrf24_getStatus();
101
102      // We can short circuit on RX_DR, but if it's not set, we still need
103      // to check the FIFO for any pending packets
104      if ( status & (1 << RX_DR) )
105      {
106          return 1;
107      }
108
109      return !nrf24_rxFifoEmpty();;
110  }
111
112  /* Checks if receive FIFO is empty or not */
113  uint8_t nrf24_rxFifoEmpty()
114  {
115      uint8_t fifoStatus;
116
117      nrf24_readRegister(FIFO_STATUS,&fifoStatus,1);
118
119      return (fifoStatus & (1 << RX_EMPTY));
120  }
121
122  /* Returns the length of data waiting in the RX fifo */
123  uint8_t nrf24_payloadLength()
124  {
125      uint8_t status;
126      nrf24_csn_digitalWrite(LOW);
127      spi_transfer(R_RX_PL_WID);
128      status = spi_transfer(0x00);
129      nrf24_csn_digitalWrite(HIGH);
130      return status;
131  }
132
133  /* Reads payload bytes into data array */
134  void nrf24_getData(uint8_t* data)
135  {
136      /* Pull down chip select */
137      nrf24_csn_digitalWrite(LOW);
138
139      /* Send cmd to read rx payload */
140      spi_transfer( R_RX_PAYLOAD );
141
142      /* Read payload */
143      nrf24_transferSync(data,data,payload_len);
144
145      /* Pull up chip select */
146      nrf24_csn_digitalWrite(HIGH);
147
148      /* Reset status register */
```

```
149     nrf24_configRegister(STATUS,(1<<RX_DR));
150 }
151
152 /* Returns the number of retransmissions occured for the last message */
153 uint8_t nrf24_retransmissionCount()
154 {
155     uint8_t rv;
156     nrf24_readRegister(OBSERVE_TX,&rv,1);
157     rv = rv & 0x0F;
158     return rv;
159 }
160
161 // Sends a data package to the default address. Be sure to send the correct
162 // amount of bytes as configured as payload on the receiver.
163 void nrf24_send(uint8_t* value)
164 {
165     /* Go to Standby-I first */
166     nrf24_ce_digitalWrite(LOW);
167
168     /* Set to transmitter mode , Power up if needed */
169     nrf24_powerUpTx();
170
171     /* Do we really need to flush TX fifo each time ? */
172     #if 1
173     /* Pull down chip select */
174     nrf24_csn_digitalWrite(LOW);
175
176     /* Write cmd to flush transmit FIFO */
177     spi_transfer(FLUSH_TX);
178
179     /* Pull up chip select */
180     nrf24_csn_digitalWrite(HIGH);
181     #endif
182
183     /* Pull down chip select */
184     nrf24_csn_digitalWrite(LOW);
185
186     /* Write cmd to write payload */
187     spi_transfer(W_TX_PAYLOAD);
188
189     /* Write payload */
190     nrf24_transmitSync(value,payload_len);
191
192     /* Pull up chip select */
193     nrf24_csn_digitalWrite(HIGH);
194
195     /* Start the transmission */
196     nrf24_ce_digitalWrite(HIGH);
197 }
198
199 uint8_t nrf24_isSending()
200 {
```

```
201     uint8_t status;
202
203     /* read the current status */
204     status = nrf24_getStatus();
205
206     /* if sending successful (TX_DS) or max retries exceded (MAX_RT). */
207     if((status & ((1 << TX_DS)  | (1 << MAX_RT))))
208     {
209         return 0; /* false */
210     }
211
212     return 1; /* true */
213
214 }
215
216 uint8_t nrf24_getStatus()
217 {
218     uint8_t rv;
219     nrf24_csn_digitalWrite(LOW);
220     rv = spi_transfer(NOP);
221     nrf24_csn_digitalWrite(HIGH);
222     return rv;
223 }
224
225 uint8_t nrf24_lastMessageStatus()
226 {
227     uint8_t rv;
228
229     rv = nrf24_getStatus();
230
231     /* Transmission went OK */
232     if((rv & ((1 << TX_DS))))
233     {
234         return NRF24_TRANSMISSON_OK;
235     }
236     /* Maximum retransmission count is reached */
237     /* Last message probably went missing ... */
238     else if((rv & ((1 << MAX_RT))))
239     {
240         return NRF24_MESSAGE_LOST;
241     }
242     /* Probably still sending ... */
243     else
244     {
245         return 0xFF;
246     }
247 }
248
249 void nrf24_powerUpRx()
250 {
251     nrf24_csn_digitalWrite(LOW);
252     spi_transfer(FLUSH_RX);
```

```
253     nrf24_csn_digitalWrite(HIGH);
254
255     nrf24_configRegister(STATUS,(1<<RX_DR)|(1<<TX_DS)|(1<<MAX_RT));
256
257     nrf24_ce_digitalWrite(LOW);
258     nrf24_configRegister(CONFIG,nrf24_CONFIG|((1<<PWR_UP)|(1<<PRIM_RX)));
259     nrf24_ce_digitalWrite(HIGH);
260
261     _delay_ms(5);
262 }
263
264 void nrf24_powerUpTx()
265 {
266     nrf24_configRegister(STATUS,(1<<RX_DR)|(1<<TX_DS)|(1<<MAX_RT));
267
268     nrf24_configRegister(CONFIG,nrf24_CONFIG|((1<<PWR_UP)|(0<<PRIM_RX)));
269
270     _delay_ms(5);
271 }
272
273 void nrf24_powerDown()
274 {
275     nrf24_ce_digitalWrite(LOW);
276     nrf24_configRegister(CONFIG,nrf24_CONFIG);
277
278     _delay_ms(5);
279 }
280
281 uint8_t spi_transfer(uint8_t tx)
282 {
283     uint8_t i = 0;
284     uint8_t rx = 0;
285
286     nrf24_sck_digitalWrite(LOW);
287
288     for(i=0;i<8;i++)
289     {
290
291         if(tx & (1<<(7-i)))
292         {
293             nrf24_mosi_digitalWrite(HIGH);
294         }
295         else
296         {
297             nrf24_mosi_digitalWrite(LOW);
298         }
299
300         nrf24_sck_digitalWrite(HIGH);
301
302         rx = rx << 1;
303         if(nrf24_miso_digitalRead())
304         {
```

```
305             rx |= 0x01;
306         }
307
308         nrf24_sck_digitalWrite(LOW);
309
310     }
311
312     return rx;
313 }
314
315 /* send and receive multiple bytes over SPI */
316 void nrf24_transferSync(uint8_t* dataout,uint8_t* datain,uint8_t len)
317 {
318     uint8_t i;
319
320     for(i=0;i<len;i++)
321     {
322         datain[i] = spi_transfer(dataout[i]);
323     }
324
325 }
326
327 /* send multiple bytes over SPI */
328 void nrf24_transmitSync(uint8_t* dataout,uint8_t len)
329 {
330     uint8_t i;
331
332     for(i=0;i<len;i++)
333     {
334         spi_transfer(dataout[i]);
335     }
336
337 }
338
339 /* Clocks only one byte into the given nrf24 register */
340 void nrf24_configRegister(uint8_t reg, uint8_t value)
341 {
342     nrf24_csn_digitalWrite(LOW);
343     spi_transfer(W_REGISTER | (REGISTER_MASK & reg));
344     spi_transfer(value);
345     nrf24_csn_digitalWrite(HIGH);
346 }
347
348 /* Read single register from nrf24 */
349 void nrf24_readRegister(uint8_t reg, uint8_t* value, uint8_t len)
350 {
351     nrf24_csn_digitalWrite(LOW);
352     spi_transfer(R_REGISTER | (REGISTER_MASK & reg));
353     nrf24_transferSync(value,value,len);
354     nrf24_csn_digitalWrite(HIGH);
355 }
356
```

```
357 /* Write to a single register of nrf24 */
358 void nrf24_writeRegister(uint8_t reg, uint8_t* value, uint8_t len)
359 {
360     nrf24_csn_digitalWrite(LOW);
361     spi_transfer(W_REGISTER | (REGISTER_MASK & reg));
362     nrf24_transmitSync(value,len);
363     nrf24_csn_digitalWrite(HIGH);
364 }
365
366 /* Check single register from nrf24 */
367 bool nrf24_checkRegister(uint8_t reg, uint8_t desiredValue, uint8_t len)
368 {
369     uint8_t registerValue;
370     nrf24_readRegister(reg,&registerValue,len);
371     if (registerValue==desiredValue) { return true; } else { return false; }
372 }
373
374 #define RF_DDR  DDRD
375 #define RF_PORT PORTD
376 #define RF_PIN  PIND
377
378 #define CE_CSN_DDR  DDRC
379 #define CE_CSN_PORT PORTC
380 #define CE_CSN_PIN  PINC
381
382 #define MISO_BIT_POS    0
383 #define MOSI_BIT_POS    1
384 #define SCK_BIT_POS     4
385
386 #define CE_BIT_POS      0
387 #define CSN_BIT_POS     1
388
389 #define set_bit(reg,bit) reg |= (1<<bit)
390 #define clr_bit(reg,bit) reg &= ~(1<<bit)
391 #define check_bit(reg,bit) (reg&(1<<bit))
392
393 /* ---------------------------------------------------------------------------- */
394
395 void nrf24_setupPins()
396 {
397     set_bit(CE_CSN_DDR, CE_BIT_POS); // CE output
398     set_bit(CE_CSN_DDR, CSN_BIT_POS); // CSN output
399
400     clr_bit(RF_DDR, MISO_BIT_POS); // MISO input
401     set_bit(RF_DDR, MOSI_BIT_POS); // MOSI output
402     set_bit(RF_DDR, SCK_BIT_POS); // SCK output
403 }
404 /* ---------------------------------------------------------------------------- */
405 void nrf24_ce_digitalWrite(uint8_t state)
406 {
407     if(state)
408     {
```

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