#include "rfid.h"

static inline void EnableRFField (void);

static inline void DisableRFField (void);

static void \_REQA(void);

static void SendAndReceive (void);

static void EraseBuffers (void);

uint8\_t GetByteFromDataBuffer (uint8\_t byte)

{

if (byte < DATABUFFERSIZE)

return data\_buffer[byte];

return 0;

}

uint8\_t Nibble2Hex (uint8\_t nibble)

{

if (nibble > 15)

return 0;

if (nibble < 10)

return (nibble+48);

else

return (nibble+55);

}

static void EraseBuffers(void)

{

uint8\_t loop;

for( loop=0; loop < DATABUFFERSIZE; loop++)

{

data\_buffer[loop] = 0x00;

send\_and\_receive\_buffer[loop] = 0x00;

}

for(;loop < SENDRECEIVEBUFFERSIZE; loop++)

{

send\_and\_receive\_buffer[loop] = 0x00;

}

}

void EncodePCDRequest (uint8\_t length)

{

uint8\_t last\_bit;

uint8\_t next\_bit;

uint8\_t nibble\_counter;

uint8\_t current\_byte;

uint8\_t parity\_bit;

uint8\_t counter;

uint8\_t bit;

uint8\_t byte;

uint8\_t sequence;

uint8\_t byte\_count;

byte\_count=length;

// compute CRC and add to databuffer

// crc is only needed for read and write commands

uint16\_t CRC\_checksum = CRC16 ((uint8\_t\*)data\_buffer, byte\_count);

data\_buffer[byte\_count] = (uint8\_t)(CRC\_checksum & 0x00FF);

data\_buffer[byte\_count+1] = (uint8\_t)(CRC\_checksum >> 8);

byte\_count += 2;

// add start sequence and parity bit

// reverses bits

// one bit is represented in four time units

//X = 0xD Y = 0xF and Z = 0x7

counter = 0;

send\_and\_receive\_buffer[counter] = Z; // add start bit

last\_bit = 0;

nibble\_counter = 1;

for(byte=0; byte < byte\_count; byte++)

{

current\_byte = data\_buffer[byte];

parity\_bit = 1;

for(bit =0; bit < 8; bit++)

{

next\_bit = current\_byte & 0x01; //LSB

current\_byte = current\_byte >> 1;

parity\_bit = parity\_bit ^ next\_bit;

if(next\_bit == 1)

sequence = X;

else if(last\_bit == 0)

sequence = Z;

else

sequence = Y;

send\_and\_receive\_buffer[counter] = send\_and\_receive\_buffer[counter] | (sequence>>(4\*nibble\_counter));

if (nibble\_counter == 1)

{

nibble\_counter = 0;

counter++;

}

else

nibble\_counter = 1;

last\_bit = next\_bit;

}

if(parity\_bit == 1)

sequence = X;

else if (last\_bit == 0)

sequence =Z;

else

sequence = Y;

send\_and\_receive\_buffer[counter] = send\_and\_receive\_buffer[counter] | (sequence>>(4\*nibble\_counter));

if (nibble\_counter == 1)

{

nibble\_counter = 0;

counter++;

}

else

nibble\_counter = 1;

last\_bit = parity\_bit;

}

// end sequence

if(last\_bit == 1)

send\_and\_receive\_buffer[counter] = send\_and\_receive\_buffer[counter] | (Y>>(4\*nibble\_counter));

else

send\_and\_receive\_buffer[counter] = send\_and\_receive\_buffer[counter] | (Z>>(4\*nibble\_counter));

if(nibble\_counter == 1)

{

nibble\_counter = 0;

counter++;

}

else

nibble\_counter = 1;

send\_and\_receive\_buffer[counter] = send\_and\_receive\_buffer[counter] | (Y>>(4\*nibble\_counter));

}

eReturnCode REQA(void)

{

EnableRFField ();

\_REQA();

SendAndReceive();

DecodePICCReply();

DisableRFField();

if((data\_buffer[0] == 0x44) && data\_buffer[1] == 0x00)

return REQA\_REPLY\_OK;

else

return REQA\_REPLY\_ERROR;

}

static void \_REQA(void) //Request type A

{

//REQA code: 0x26 (7bits)

//add start- and endbit;no parity bit is added

//Modified Miller encoding : ZZ XX YZ XY ZY

EraseBuffers();

send\_and\_receive\_buffer[0]=ZZ;

send\_and\_receive\_buffer[1]=XX;

send\_and\_receive\_buffer[2]=YZ;

send\_and\_receive\_buffer[3]=XY;

send\_and\_receive\_buffer[4]=ZY;

PCD\_bit\_count\_LSB = 0x28; // 40d

PCD\_bit\_count\_MSB = 0x00;

//response 0x0044

//2bytes with parity + 1 start bit

//38 eru = 0x26

//saving 0x0190 eru

PICC\_bit\_count\_LSB = 0x90;

PICC\_bit\_count\_MSB = 0x01;

}

eReturnCode WUPA (void) //Wake up type A

{

//WUPA code: 0x52 (7bits)

//add start- and end bit; no parity bit is added

//Modified Miller encoding : ZZ XY ZX YX YY

EraseBuffers();

send\_and\_receive\_buffer[0]=ZZ;

send\_and\_receive\_buffer[1]=XY;

send\_and\_receive\_buffer[2]=ZX;

send\_and\_receive\_buffer[3]=YX;

send\_and\_receive\_buffer[4]=YY;

PCD\_bit\_count\_LSB = 0x28; // 40d

PCD\_bit\_count\_MSB = 0x00;

//response 0x0044

//2bytes with parity + 1 start bit

//38 eru = 0x13

PICC\_bit\_count\_LSB = 0x90; //

PICC\_bit\_count\_MSB = 0x01;

EnableRFField();

SendAndReceive();

DisableRFField();

DecodePICCReply();

if((data\_buffer[0] == 0x44) && data\_buffer[1] == 0x00)

return WUPA\_REPLY\_OK;

else

return WUPA\_REPLY\_ERROR;

}

eReturnCode READ (uint8\_t page)

{

uint16\_t CRC\_checksum;

if(page > 0x0F)

return BAD\_PAGE;

EnableRFField();

\_REQA();

SendAndReceive();

DecodePICCReply();

if( (data\_buffer[0] != 0x44) || data\_buffer[1] != 0x00)

return REQA\_REPLY\_ERROR; // exit function

//READ code: 0x30

//Parameter: 0x00-0x0F

//CRC check

EraseBuffers(); //very important

data\_buffer[0] = 0x30;

data\_buffer[1] = page;

EncodePCDRequest(2); //dont change , 2 is length of command

//2 cmd bytes, 2 crc bytes, 1 start bit, 2 end bits, parity bits for each byte => 39 bits

//-> 39 \* 4 -> 19,5 bytes

PCD\_bit\_count\_LSB = 0x9C; // 156d

PCD\_bit\_count\_MSB = 0x00;

PICC\_bit\_count\_LSB = 0x90; //

PICC\_bit\_count\_MSB = 0x01;

SendAndReceive();

DecodePICCReply();

DisableRFField();

//make CRC check

CRC\_checksum = CRC16 ((uint8\_t\*)data\_buffer,18);

if(CRC\_checksum != 0x0000)

return CRC\_ERROR; //exit function

return READ\_REPLY\_OK;

}

inline void WaitCycles (uint16\_t count)

{

asm volatile ("cp %A0, \_\_zero\_reg\_\_\n\t"

"cpc %B0, \_\_zero\_reg\_\_\n\t"

"breq 2f \n\t"

"1: \n\t"

"sbiw %0,1 \n\t"

"brne 1b \n\t"

"2: "

: "=w" (count)

: "0" (count)

);

}

eReturnCode WRITE (uint8\_t page, uint8\_t\* data)

{

uint16\_t CRC\_checksum;

if((page < 0x04) || (page > 0x0F))

return BAD\_PAGE;

EnableRFField();

\_REQA();

SendAndReceive();

DecodePICCReply();

if((data\_buffer[0] != 0x44) || data\_buffer[1] != 0x00)

return REQA\_REPLY\_ERROR; //exitfunction2

EraseBuffers(); //veryimportant

//read (0)

data\_buffer[0] = 0x30;

data\_buffer[1] = 0x00;

EncodePCDRequest(2);

PCD\_bit\_count\_LSB = 0x9C; // 156d

PCD\_bit\_count\_MSB = 0x00;

PICC\_bit\_count\_LSB = 0x90; //

PICC\_bit\_count\_MSB = 0x01;

SendAndReceive();

DecodePICCReply();

CRC\_checksum = CRC16 ((uint8\_t\*)data\_buffer,18);

if (CRC\_checksum != 0x0000)

return CRC\_ERROR; //exitfunction

EraseBuffers(); //veryimportant

//write

data\_buffer[0] = 0xA2;

data\_buffer[1] = page;

data\_buffer[2] = \*data;

data\_buffer[3] = \*(data+1);

data\_buffer[4] = \*(data+2);

data\_buffer[5] = \*(data+3);

EncodePCDRequest(6);

PCD\_bit\_count\_LSB = 0x2C; // 300d

PCD\_bit\_count\_MSB = 0x01;

PICC\_bit\_count\_LSB = 0x90; //

PICC\_bit\_count\_MSB = 0x01;

SendAndReceive();

//have to send REQA to complete write command

\_REQA();

SendAndReceive();

DecodePICCReply();

DisableRFField();

return WRITE\_OK;

}

uint16\_t CRC16 (uint8\_t\* data, uint8\_t length)

{

uint8\_t byte;

uint16\_t CRC16;

CRC16 = CRC16INITVAL;

do{

byte = \*data++;

byte = (byte^(uint8\_t)((CRC16) & 0x00FF));

byte = (byte^(byte<<4));

CRC16 = (CRC16>>8)^((uint16\_t)byte<<8)^((uint16\_t)byte<<3)^((uint16\_t)byte>>4);

}while(--length);

return CRC16;

}

//decodes elementray receiving units in bits

//do esparity check, remove parity bit

//reverses bits: big endian --> little endian

//bytes are still reversed and crc check not yet done

uint8\_t DecodePICCReply(void)

{

uint16\_t i;

for(i=0; i<82; i++)

{

send\_and\_receive\_buffer[i] = ~send\_and\_receive\_buffer[i];

}

uint8\_t return\_code = 0;

send\_and\_receive\_buffer[SENDRECEIVEBUFFERSIZE-1] = 0x00;

///////////////////////////

//last byte of send\_and\_receive\_buffer must be cleared

//ensures that alorithm stops

//search for first bit set in send\_and\_receive\_buffer bit stream

asm volatile (

"push r20 \n\t"

"push r21 \n\t"//bit position counter for r2

"push r22 \n\t"//current byte from databuffer

"push r23 \n\t"//bit position counter for r4

"push r24 \n\t"//current byte from eu\_buffer

"push r25 \n\t"

"push r26 \n\t"

"push r27 \n\t"

"push r28 \n\t"//LSB of Y

"push r29 \n\t"//MSB of Y

"push r30 \n\t"//LSB of Z

"push r31 \n\t"//MSB of Z

///////////////////

//start of init//

///////////////////

"ldi r28,lo8(data\_buffer)\n\t"//load &data\_buffer in Y

"ldi r29,hi8(data\_buffer)\n\t"

"ldi r30,lo8(send\_and\_receive\_buffer)\n\t"//load &eu\_buffer in Z

"ldi r31,hi8(send\_and\_receive\_buffer)\n\t"

"ldi r21,8 \n\t"

"ldi r23,8 \n\t"

"ld r24,Z+ \n\t"

"ldi r25,0xFF \n\t"//time out

//"ldir26, 0 \n\t"

"ldi r27,0x80 \n\t"//= 1 for odd parity

"ldi r20,0 \n\t"

/////////////////

//end of init//

/////////////////

//////////////

///searching first set bit in eu\_buffer

//if no bit is set with in the r25 bits of eu\_buffer -> quit with error

/////////////

"10: \n\t"

"rcall get\_bit \n\t"

"dec r25 \n\t"

"breq 20f \n\t"//error

"brcc 10b \n\t"

//found 1

"rcall get\_bit \n\t"//found 1 in bit stream !; next bit has to be a 0 for correct start sequence

"brcs 20f \n\t"//if C ist set => error

//decoding data

"15: \n\t"

"rcall get\_bit \n\t"

"brcc 11f \n\t"

"rcall get\_bit \n\t"//expecting 0

"brcs 20f \n\t"//error if 11 sequence

"sec \n\t"

"rcall put\_bit \n\t"//decoded with success 10 -> 1

"rjmp 15b \n\t"

"11: \n\t"

"rcall get\_bit \n\t"//excpecting 1

"brcc 12f \n\t"//if 00 sequence -> end of stream

"clc \n\t"

"rcall put\_bit \n\t"//decoded with success 01 -> 0

"rjmp 15b \n\t"

//put bit from Carry in 2r2

//if byte is full, stores value of r22 in data buffer

//changes r21, r22

"put\_bit: \n\t"

"sbrc r20,0 \n\t"

"rjmp 2f \n\t" //if r20==1 -> carry bit equals parity bit

"ror r22 \n\t"

"mov r26,r22 \n\t"

"andi r26,0x80 \n\t"

"eor r27,r26 \n\t"

"dec r21 \n\t"

"brne 1f \n\t"

"ldi r20,1 \n\t"//next decode bit must be parity bit

"rjmp 1f \n\t"

"2: \n\t"

"eor r26,r26 \n\t"

"ror r26 \n\t"

"cp r26,r27 \n\t"

"brne 20f \n\t"//branch if parity check fails

"st Y+,r22 \n\t"

"ldi r21,0x08 \n\t"

"ldi r20,0 \n\t"

"ldi r27,0x80 \n\t"

"1: \n\t"

"ret \n\t"

//get\_bit routine; puts MSB of r4 into Carry

//if r4 is empty loads next byte from eu\_buffer

//changes C,r3 ,r4

"get\_bit: \n\t"

"lsl r24 \n\t"

"dec r23 \n\t"//does not affect C

"brne 1f \n\t"

"ld r24,Z+ \n\t"

"ldi r23, 0x08 \n\t"

"1: \n\t"

"ret \n\t"

"20: \n\t"

"rjmp 30f \n\t"

"12: \n\t"

"30: \n\t"

"pop r31 \n\t"

"pop r30 \n\t"

"pop r29 \n\t"

"pop r28 \n\t"

"pop r27 \n\t"

"pop r26 \n\t"

"pop r25 \n\t"

"pop r24 \n\t"

"pop r23 \n\t"

"pop r22 \n\t"

"pop r21 \n\t"

"pop r20 \n\t"

::);

return return\_code;

}

void DisplayDataBuffer(uint8\_t length)

{

uint16\_t loop;

uint8\_t temp;

for(loop=0; loop < length; loop++)

{

temp =((data\_buffer[loop] & 0xF0) >> 4);

if (temp < 10)

temp += 48;

else

temp += 55;

lcd\_data(temp);

temp = (data\_buffer[loop] & 0x0F);

if (temp < 10)

temp += 48;

else

temp += 55;

lcd\_data(temp);

}

}

static inline void EnableRFField(void)

{

PORTB = PORTB | 0x01;

WaitCycles(0xFFFF);

}

static inline void DisableRFField(void)

{

PORTB = PORTB & 0xFE;

WaitCycles(0xFFFF);

}

static void SendAndReceive (void)

{

asm volatile ( "push r30 \n\t"//LSB of Z

"push r31 \n\t"//MSB of Z

"push r16 \n\t"//aktuelles Byte aus dem buffer

"push r17 \n\t"//delay counter

"push r18 \n\t"//8 bit PCD\_bit\_count

"push r19 \n\t"//shift\_counter

"push r24 \n\t"//PCD\_bit\_count\_LSB

"push r25 \n\t"//etu counter MSB

"ldi r30,lo8(send\_and\_receive\_buffer)\n\t"//lade die Adresse von buffer in Z

"ldi r31,hi8(send\_and\_receive\_buffer)\n\t"

"ld r16,Z+ \n\t"//lade erstes Byte von buffer in r16

"lds r24,PCD\_bit\_count\_LSB \n\t"

"lds r25,PCD\_bit\_count\_MSB \n\t"

"ldi r19,0x08 \n\t"//one byte, 8 bits

"9: \n\t"//bit count loop

"lsl r16 \n\t"//1 cycle

"brcs 1f \n\t"//jump to 1: if bit was set

"nop \n\t"//to equalize delay;1 cycle

"cbi 0x18,0x00 \n\t"//clears B0 in output port;2 cycles + 1 cycle from branch

"rjmp 0f \n\t"//2 cycles

"1: \n\t"//2 cycles from branch

"sbi 0x18,0x00 \n\t"//sets B0;2 cycles

"nop \n\t"//to equalize delay

"nop \n\t"//to equalize delay

"0: \n\t"//

//2 cycles passed after sbi or cbi execution,

//output must stay constant for further 32-2=30 cycles

"dec r19 \n\t"//shift\_counter--1 cycle

"brne 2f \n\t"//branches if shift\_counter <> 0

"ld r16,Z+ \n\t"//load new byte from buffer;2 cycles + 1 cycle from branch

"ldi r19,0x08 \n\t"//1 cycle

"rjmp 3f \n\t"//2 cycles

"2: \n\t"//2 cycles from branch

"nop \n\t"//to equalize delay

"nop \n\t"//to equalize delay

"nop \n\t"//to equalize delay

"nop \n\t"//to equalize delay

"3: \n\t"

//7 cycles later, still 23 cycles left

//wait 12 cycles (n\*3)

"ldi r17,4 \n\t"//1 cycle

"4: \n\t"

"dec r17 \n\t"//1 cycle

"brne 4b \n\t"//if true 2 cycles if false 1 cycle

"nop \n\t"//to equalize delay

"nop \n\t"//to equalize delay

// "dec r18 \n\t"//1 cycle

"sbiw r24,0x01 \n\t" //2 cacles

"brne 9b \n\t"//branch if r18 <> 0

//26 cycles after last outout change

//response will come in 948 - 26 = 922 cycles

//wait 765 cycles

"ldi r17,0xFF \n\t"// 1cycle

"5: \n\t"

"dec r17 \n\t"//1 cycle

"brne 5b \n\t"//if true 2 cycles if false 1 cycle5

//rest: 157 cycles

//waits 150 cycles

//77 < r17 < 87

"ldi r30,lo8(send\_and\_receive\_buffer)\n\t"//lade die Adresse von buffer in Z; 1 cycle

"ldi r31,hi8(send\_and\_receive\_buffer)\n\t"//1 cycle

"lds r24,PICC\_bit\_count\_LSB \n\t"//2 cycles

"lds r25,PICC\_bit\_count\_MSB \n\t"//2 cycles

//"ldi r18,80 \n\t"//PCD\_bit\_count; 1 cycle

"ldi r19,0x08 \n\t"//one byte, 8 bits; 1 cycle

"eor r16,r16 \n\t"//clear r16; important if not a whole byte is written; 1 cycle

//lastblock 5 cycles

"10: \n\t"

"in r17,0x16 \n\t"//get value of Port B; 1 cycle

"lsr r17 \n\t"//put pin b1 into carry flag; 1 cycle

"lsr r17 \n\t"//put pin b2 into carry flag; 1 cycle

"lsr r17 \n\t"//put pin b3 into carry flag; 1 cycle

"lsr r17 \n\t"//put pin b4 into carry flag; 1 cycle

"lsr r17 \n\t"

"rol r16 \n\t"//put LSB of r17 into r16; 1 cycle

"dec r19 \n\t"//1 cycle

"brne 7f \n\t"

"st Z+,r16 \n\t"//1 cycle from branch + 2 cycles

"ldi r19,0x08 \n\t"//1 cycle

"eor r16,r16 \n\t"//1 cycle

"rjmp 8f \n\t"//2 cycles

"7: \n\t"//+ 2 cycles from branch

"nop \n\t"

"nop \n\t"

"nop \n\t"

"nop \n\t"

"nop \n\t"

"8: \n\t"

//14 cycles since last in instrucion

"ldi r17,15 \n\t"//1 cycle

"5: \n\t"

"dec r17 \n\t"//1 cycle

"brne 5b \n\t"//if true 2 cycles if false 1 cycle

//"dec r18 \n\t" //1 cycle

//"nop \n\t"

"sbiw r24,0x01 \n\t" //2 cycles

"brne 10b \n\t"

"pop r25 \n\t"//PCD bit count LSB

"pop r24 n\t"//etu counter MSB

"pop r19 n\t"

"pop r18 n\t"

"pop r17 n\t"

"pop r16 n\t"

"pop r31 n\t"

"pop r30 n\t"

::);

}

void InitUSART(void)

{

#define F\_CPU 13560000L //Systemtakt in Hz, das L am Ende ist wichtig, NICHT UL verwenden !

#define BAUD 9600L //Baudrate, das L am Ende ist wichtig , NICHT UL verwenden !

#define UBRR\_VAL ((F\_CPU+BAUD\*8)/(BAUD\*16)-1) //clever runden

#define BAUD\_REAL (F\_CPU/(16\*(UBRR\_VAL+1))) // Reale Baudrate

#define BAUD\_ERROR ((BAUD\_REAL\*1000)/BAUD-1000) //Fehler in Promille

#if ((BAUD\_ERROR>10) || (BAUD\_ERROR<-10))

#error Systematischer Fehler der Baudrate grösser 1% und damit zu hoch !

#endif

UCSRB = (1<<RXEN)|(1<<TXEN);

UCSRC |= (1<<URSEL)|(3<<UCSZ0); //Asynchron 8N1

UBRRH = UBRR\_VAL >> 8;

UBRRL = UBRR\_VAL & 0xFF;

}

uint8\_t RX()

{

while (!(UCSRA & (1<<RXC))) {}

return UDR;

}

void TX(uint8\_t byte)

{

while (!(UCSRA & (1<<UDRE))) {}

UDR = byte;

}