```
//*************************************************************
// ****** FUNCTIONS FOR SD RAW DATA TRANSFER *******
//*************************************************************
//Controller: ATmega8 (Clock: 8 Mhz-internal)
//Compiler: AVR-GCC
//Version : 2.0
//Author: CC Dharmani, Chennai (India)
// www.dharmanitech.com
//Date: 26 Feb 2009
//*************************************************************
//Link to the Post:
http://www.dharmanitech.com/2009/01/sd-card-interfacing-with-atmega8-fat32.html

//***********************************************
// ***** HEADER FILE : SD_routines.h ******
//***********************************************
#ifndef _SD_ROUTINES_H_
#define _SD_ROUTINES_H_

#define FAT_TESTING_ONLY

#define SD_CS_ASSERT     PORTB &= ~0x02
#define SD_CS_DEASSERT   PORTB |= 0x02

#define GO_IDLE_STATE            0
#define SEND_OP_COND              1
#define SEND_CSD                 9
#define STOP_TRANSMISSION        12
#define SEND_STATUS              13
#define SET_BLOCK_LEN            16
#define READ_SINGLE_BLOCK        17
#define READ_MULTIPLE_BLOCKS     18
#define WRITE_SINGLE_BLOCK       24
#define WRITE_MULTIPLE_BLOCKS    25
#define ERASE_BLOCK_START_ADDR   32
#define ERASE_BLOCK_END_ADDR     33
#define ERASE_SELECTED_BLOCKS    38
#define CRC_ON_OFF               59

#define ON     1
#define OFF    0

extern volatile unsigned long startBlock;
extern volatile unsigned long totalBlocks;
extern volatile unsigned char buffer[512];

unsigned char SD_init(void);
unsigned char SD_sendCommand(unsigned char cmd, unsigned long arg);
unsigned char SD_readSingleBlock(unsigned long startBlock);
unsigned char SD_writeSingleBlock(unsigned long startBlock);
unsigned char SD_readMultipleBlock (unsigned long startBlock, unsigned long
totalBlocks);
unsigned char SD_writeMultipleBlock(unsigned long startBlock, unsigned long
totalBlocks);
```

```
unsigned char SD_erase (unsigned long startBlock, unsigned long totalBlocks);

#endif
//*************************************************************
// ****** FUNCTIONS FOR SD RAW DATA TRANSFER *******
//*************************************************************
//Controller: ATmega8 (Clock: 8 Mhz-internal)
//Compiler: AVR-GCC
//Version : 2.0
//Author: CC Dharmani, Chennai (India)
// www.dharmanitech.com
//Date: 26 Feb 2009
//*************************************************************
//Link to the Post:
http://www.dharmanitech.com/2009/01/sd-card-interfacing-with-atmega8-fat32.html

//***********************************************
// ***** SOURCE FILE : SD_routines.c ******
//***********************************************
#include <avr/io.h>
#include <avr/pgmspace.h>
#include "SPI_routines.h"
#include "SD_routines.h"
#include "UART_routines.h"

//*****************************************************************
//Function: to initialize the SD card in SPI mode
//Arguments: none
//return: unsigned char; will be 0 if no error,
// otherwise the response byte will be sent
//*****************************************************************
unsigned char SD_init(void)
{
unsigned char i, response, retry=0 ;

SD_CS_ASSERT;
do
{
   for(i=0;i<10;i++)
      SPI_transmit(0xff);
   response = SD_sendCommand(GO_IDLE_STATE, 0);//send 'reset & go idle' command
   retry++;
   if(retry>0xfe) {transmitString_F(PSTR("SD init fail.."));  return 1; }//time
out
} while(response != 0x01);

SD_CS_DEASSERT;

SPI_transmit (0xff);
SPI_transmit (0xff);

retry = 0;

do
```

```
{
    response = SD_sendCommand(SEND_OP_COND, 0); //activate card's
initialization process
  response = SD_sendCommand(SEND_OP_COND, 0); //resend command (for
compatibility with some cards)
    retry++;
    if(retry>0xfe) return 1; //time out
}while(response);

SD_sendCommand(CRC_ON_OFF, OFF); //disable CRC; deafault - CRC disabled in SPI
mode
SD_sendCommand(SET_BLOCK_LEN, 512); //set block size to 512

return 0; //normal return
}

//*****************************************************************
//Function: to send a command to SD card
//Arguments: unsigned char (8-bit command value)
// & unsigned long (32-bit command argument)
//return: unsigned char; response byte
//*****************************************************************
unsigned char SD_sendCommand(unsigned char cmd, unsigned long arg)
{
unsigned char response, retry=0;

SD_CS_ASSERT;

SPI_transmit(cmd | 0x40); //send command, first two bits always '01'
SPI_transmit(arg>>24);
SPI_transmit(arg>>16);
SPI_transmit(arg>>8);
SPI_transmit(arg);
SPI_transmit(0x95);

while((response = SPI_receive()) == 0xff) //wait response
   if(retry++ > 0xfe) break; //time out error

SPI_receive(); //extra 8 CLK
SD_CS_DEASSERT;

return response; //return state
}

//*****************************************************************
//Function: to erase specified no. of blocks of SD card
//Arguments: none
//return: unsigned char; will be 0 if no error,
// otherwise the response byte will be sent
//*****************************************************************
unsigned char SD_erase (unsigned long startBlock, unsigned long totalBlocks)
{
unsigned char response;
```

```
response = SD_sendCommand(ERASE_BLOCK_START_ADDR, startBlock<<9); //send
starting block address
if(response != 0x00) //check for SD status: 0x00 - OK (No flags set)
  return response;

response = SD_sendCommand(ERASE_BLOCK_END_ADDR,(startBlock + totalBlocks -
1)<<9); //send end block address
if(response != 0x00)
  return response;

response = SD_sendCommand(ERASE_SELECTED_BLOCKS, 0); //erase all selected
blocks
if(response != 0x00)
  return response;

return 0; //normal return
}

//*****************************************************************
//Function: to read a single block from SD card
//Arguments: none
//return: unsigned char; will be 0 if no error,
// otherwise the response byte will be sent
//*****************************************************************
unsigned char SD_readSingleBlock(unsigned long startBlock)
{
unsigned char response;
unsigned int i, retry=0;

response = SD_sendCommand(READ_SINGLE_BLOCK, startBlock<<9); //read a Block
command
//block address converted to starting address of 512 byte Block
if(response != 0x00) //check for SD status: 0x00 - OK (No flags set)
  return response;

SD_CS_ASSERT;

while(SPI_receive() != 0xfe) //wait for start block token 0xfe (0x11111110)
  if(retry++ > 0xfffe){SD_CS_DEASSERT; return 1;} //return if time-out

for(i=0; i<512; i++) //read 512 bytes
  buffer[i] = SPI_receive();

SPI_receive(); //receive incoming CRC (16-bit), CRC is ignored here
SPI_receive();

SPI_receive(); //extra 8 clock pulses
SD_CS_DEASSERT;

return 0;
}

//*****************************************************************
//Function: to write to a single block of SD card
```

```
//Arguments: none
//return: unsigned char; will be 0 if no error,
// otherwise the response byte will be sent
//*****************************************************************
unsigned char SD_writeSingleBlock(unsigned long startBlock)
{
unsigned char response;
unsigned int i, retry=0;

response = SD_sendCommand(WRITE_SINGLE_BLOCK, startBlock<<9); //write a Block
command
if(response != 0x00) //check for SD status: 0x00 - OK (No flags set)
return response;

SD_CS_ASSERT;

SPI_transmit(0xfe);     //Send start block token 0xfe (0x11111110)

for(i=0; i<512; i++)    //send 512 bytes data
  SPI_transmit(buffer[i]);

SPI_transmit(0xff);     //transmit dummy CRC (16-bit), CRC is ignored here
SPI_transmit(0xff);

response = SPI_receive();

if( (response & 0x1f) != 0x05) //response= 0xXXX0AAA1 ; AAA='010' - data
accepted
{                              //AAA='101'-data rejected due to CRC error
  SD_CS_DEASSERT;              //AAA='110'-data rejected due to write error
  return response;
}

while(!SPI_receive()) //wait for SD card to complete writing and get idle
if(retry++ > 0xfffe){SD_CS_DEASSERT; return 1;}

SD_CS_DEASSERT;
SPI_transmit(0xff);   //just spend 8 clock cycle delay before reasserting the
CS line
SD_CS_ASSERT;         //re-asserting the CS line to verify if card is still
busy

while(!SPI_receive()) //wait for SD card to complete writing and get idle
   if(retry++ > 0xfffe){SD_CS_DEASSERT; return 1;}
SD_CS_DEASSERT;

return 0;
}

#ifndef FAT_TESTING_ONLY
//*****************************************************************************
//Function: to read multiple blocks from SD card & send every block to UART
//Arguments: none
//return: unsigned char; will be 0 if no error,
```

```
// otherwise the response byte will be sent
//*****************************************************************************
unsigned char SD_readMultipleBlock (unsigned long startBlock, unsigned long
totalBlocks)
{
unsigned char response;
unsigned int i, retry=0;

retry = 0;

response = SD_sendCommand(READ_MULTIPLE_BLOCKS, startBlock <<9); //read a Block
command
//block address converted to starting address of 512 byte Block
if(response != 0x00) //check for SD status: 0x00 - OK (No flags set)
return response;

SD_CS_ASSERT;

while( totalBlocks )
{
  retry = 0;
  while(SPI_receive() != 0xfe) //wait for start block token 0xfe (0x11111110)
  if(retry++ > 0xfffe){SD_CS_DEASSERT; return 1;} //return if time-out

  for(i=0; i<512; i++) //read 512 bytes
    buffer[i] = SPI_receive();

  SPI_receive(); //receive incoming CRC (16-bit), CRC is ignored here
  SPI_receive();

  SPI_receive(); //extra 8 cycles
  TX_NEWLINE;
  transmitString_F(PSTR(" --------- "));
  TX_NEWLINE;

  for(i=0; i<512; i++) //send the block to UART
  {
    if(buffer[i] == '~') break;
    transmitByte ( buffer[i] );
  }

  TX_NEWLINE;
  transmitString_F(PSTR(" --------- "));
  TX_NEWLINE;
  totalBlocks--;
}

SD_sendCommand(STOP_TRANSMISSION, 0); //command to stop transmission
SD_CS_DEASSERT;
SPI_receive(); //extra 8 clock pulses

return 0;
}
```

```
//*****************************************************************************
//Function: to receive data from UART and write to multiple blocks of SD card
//Arguments: none
//return: unsigned char; will be 0 if no error,
// otherwise the response byte will be sent
//*****************************************************************************
unsigned char SD_writeMultipleBlock(unsigned long startBlock, unsigned long
totalBlocks)
{
unsigned char response, data;
unsigned int i, retry=0;
unsigned long blockCounter=0, size;

response = SD_sendCommand(WRITE_MULTIPLE_BLOCKS, startBlock<<9); //write a
Block command
if(response != 0x00) //check for SD status: 0x00 - OK (No flags set)
  return response;

SD_CS_ASSERT;

TX_NEWLINE;
transmitString_F(PSTR(" Enter text (End with ~): "));
TX_NEWLINE;

while( blockCounter < totalBlocks )
{
   i=0;
   do
   {
     data = receiveByte();
     if(data == 0x08)   //'Back Space' key pressed
      {
        if(i != 0)
        {
          transmitByte(data);
          transmitByte(' ');
          transmitByte(data);
          i--;
            size--;
        }
        continue;
      }
     transmitByte(data);
     buffer[i++] = data;
     if(data == 0x0d)
     {
        transmitByte(0x0a);
        buffer[i++] = 0x0a;
     }
      if(i == 512) break;
   }while (data != '~');

   TX_NEWLINE;
   transmitString_F(PSTR(" ---- "));
```

```
   TX_NEWLINE;

   SPI_transmit(0xfc); //Send start block token 0xfc (0x11111100)

   for(i=0; i<512; i++) //send 512 bytes data
      SPI_transmit( buffer[i] );

   SPI_transmit(0xff); //transmit dummy CRC (16-bit), CRC is ignored here
   SPI_transmit(0xff);

   response = SPI_receive();
   if( (response & 0x1f) != 0x05) //response= 0xXXX0AAA1 ; AAA='010' - data
accepted
   {                          //AAA='101'-data rejected due to CRC error
      SD_CS_DEASSERT;            //AAA='110'-data rejected due to write error
      return response;
   }

   while(!SPI_receive()) //wait for SD card to complete writing and get idle
     if(retry++ > 0xfffe){SD_CS_DEASSERT; return 1;}

   SPI_receive(); //extra 8 bits
   blockCounter++;
}

SPI_transmit(0xfd); //send 'stop transmission token'

retry = 0;

while(!SPI_receive()) //wait for SD card to complete writing and get idle
   if(retry++ > 0xfffe){SD_CS_DEASSERT; return 1;}

SD_CS_DEASSERT;
SPI_transmit(0xff); //just spend 8 clock cycle delay before reasserting the CS
signal
SD_CS_ASSERT; //re assertion of the CS signal is required to verify if card is
still busy

while(!SPI_receive()) //wait for SD card to complete writing and get idle
   if(retry++ > 0xfffe){SD_CS_DEASSERT; return 1;}
SD_CS_DEASSERT;

return 0;
}
//*****************************************
#endif

//******** END ****** www.dharmanitech.com *****
```