```
//-------------------------------------------------------------------
// Name:            Kyle Hughes, Alex Kim
// Date:            Spring 2021
// Purp:            lab10
//
// Assisted:        The entire class of EENG 383
// Assisted by:     Microchips 18F26K22 Tech Docs
//-
//- Academic Integrity Statement: I certify that, while others may have
//- assisted me in brain storming, debugging and validating this program,
//- the program itself is my own work. I understand that submitting code
//- which is the work of other individuals is a violation of the course
//- Academic Integrity Policy and may result in a zero credit for the
//- assignment, or course failure and a report to the Academic Dishonesty
//- Board. I also understand that if I knowingly give my original work to
//- another individual that it could also result in a zero credit for the
//- assignment, or course failure and a report to the Academic Dishonesty
//- Board.
//-------------------------------------------------------------------------
#include "mcc_generated_files/mcc.h"
#include "sdCard.h"
#pragma warning disable 520     // warning: (520) function "xyz" is never called  3
#pragma warning disable 1498    // fputc.c:16:: warning: (1498) pointer (unknown)


void myTMR0ISR(void);

#define BLOCK_SIZE          512
#define RATE                1600
#define MAX_NUM_BLOCKS      4
#define SINE_WAVE_ARRAY_LENGTH 26
#define RED 0
#define BLUE 1

uint8_t whichBuffer = RED;
uint8_t sampleRate = 100;
uint8_t audioFlag = 0, buffFlag = 0, playFlag = 0;

// Large arrays need to be defined as global even though you may only need to
// use them in main.  This quirk will be important in the next two assignments.

uint8_t redBuffer[BLOCK_SIZE];
uint8_t blueBuffer[BLOCK_SIZE];
uint8_t sdCardBuffer[BLOCK_SIZE];
//-----------------------------------------------
// Main "function"
//-----------------------------------------------

void main(void) {
    const uint8_t   sin[SINE_WAVE_ARRAY_LENGTH] = {128,
      159,187,213,233,248,255,255,
        248,233,213,187,159,128,97, 69,   43,   23,   8,1,1,8,23, 43,   69,   97};

    uint8_t status;
    uint16_t i, j, loops, lastBlock;
    uint32_t sdCardAddress = 0x00000000;
    char cmd, letter;

    letter = '0';
```

```
    SYSTEM_Initialize();
    CS_SetHigh();

    // Provide Baud rate generator time to stabilize before splash screen
    TMR0_WriteTimer(0x0000);
    INTCONbits.TMR0IF = 0;
    while (INTCONbits.TMR0IF == 0);

    TMR0_SetInterruptHandler(myTMR0ISR);

    INTERRUPT_GlobalInterruptEnable();
    INTERRUPT_PeripheralInterruptEnable();

    printf("inLab 09\r\n");
    printf("SD card testing\r\n");
    printf("Dev'21\r\n");
    printf("No configuration of development board\r\n> "); // print a nice command
prompt

    SPI2_Close();
    SPI2_Open(SPI2_DEFAULT);

    for (;;) {

        if (EUSART1_DataReady) { // wait for incoming data on USART
            cmd = EUSART1_Read();
            switch (cmd) { // and do what it tells you to do

                    //-------------------------------------------
                    // Reply with help menu
                    //-------------------------------------------
                case '?':
                    printf("\r\n-------------------------------------------------\
r\n");
                    printf("SD card address:  ");
                    printf("%04x", sdCardAddress >> 16);
                    printf(":");
                    printf("%04x", sdCardAddress & 0X0000FFFF);
                    printf("\r\n");
                    printf("-------------------------------------------------\r\
n");
                    printf("?: help menu\r\n");
                    printf("o: k\r\n");
                    printf("Z: Reset processor\r\n");
                    printf("z: Clear the terminal\r\n");
                    printf("----------------SPI TEST-------------------------\r\
n");
                    printf("t: send a Test character over SPI\r\n");
                    printf("--------------SD CARD TESTS----------------------\r\
n");
                    printf("i: Initialize SD card\r\n");
                    printf("a/A decrease/increase read address\r\n");
                    printf("r: read a block of %d bytes from SD card\r\n",
BLOCK_SIZE);
                    printf("w: write a block of %d bytes to SD card\r\n",
BLOCK_SIZE);
                    printf("-------------------------------------------------\r\
n");
```

```
                    break;

                    //-----------------------------------------
                    // Reply with "k", used for PC to PIC test
                    //-----------------------------------------
                case 'o':
                    printf("o:     ok\r\n");
                    break;

                    //-----------------------------------------
                    // Reset the processor after clearing the terminal
                    //-----------------------------------------

                case 'Z':
                    for (i = 0; i < 40; i++) printf("\n");
                    RESET();
                    break;

                    //-----------------------------------------
                    // Clear the terminal
                    //-----------------------------------------

                case 'z':
                    for (i = 0; i < 40; i++) printf("\n");
                    break;

                    //-----------------------------------------
                    // Clear the terminal
                    //-----------------------------------------

                case 't':
                    printf("    Connect oscilloscope channel 1 to PIC header pin
RB1 (vertical scale 2v/div)\r\n");
                    printf("    Connect oscilloscope channel 2 to PIC header pin
RB3 (vertical scale 2v/div)\r\n");
                    printf("    Trigger on channel 1\r\n");
                    printf("    Set the horizontal scale to 500ns/div\r\n");
                    printf("    Hit any key when ready\r\n");
                    while (!EUSART1_DataReady);
                    (void) EUSART1_Read();

                    printf("sent: %02x  received: %02x\r\n", letter,
SPI2_ExchangeByte(letter));
                    letter += 1;
                    break;

                    //-----------------------------------------
                    // Init SD card to get it read to perform read/write
                    // Will hang in infinite loop on error.
                    //-----------------------------------------
                case 'i':
                    SPI2_Close();
                    SPI2_Open(SPI2_DEFAULT);    // Reset the SPI channel for SD
card communication
                    SDCARD_Initialize(true);
                    break;

                    //-----------------------------------------
                    // Increase or decrease block address
```

```
                    //-------------------------------------------
                case 'A':
                case 'a':
                    if (cmd == 'a') {
                        sdCardAddress -= BLOCK_SIZE;
                        if (sdCardAddress >= 0x04000000) {
                            printf("Underflowed to high address\r\n");
                            sdCardAddress = 0x04000000 - BLOCK_SIZE;
                        } else {
                            printf("Decreased address\r\n");
                        }
                    } else {
                        sdCardAddress += BLOCK_SIZE;
                        if (sdCardAddress >= 0x04000000) {
                            printf("Overflowed to low address\r\n");
                            sdCardAddress = 0x00000000;
                        } else {
                            printf("Increased address\r\n");
                        }
                    }

                    // 32-bit integers need printed as a pair of 16-bit integers
                    printf("SD card address:  ");
                    printf("%04x", sdCardAddress >> 16);
                    printf(":");
                    printf("%04x", sdCardAddress & 0X0000FFFF);
                    printf("\r\n");
                    break;

                    //-------------------------------------------
                    // w: write a block of BLOCK_SIZE bytes to SD card
                    //-------------------------------------------
                case 'w':
                    for (i = 0; i < BLOCK_SIZE; i++) sdCardBuffer[i] = 255 - i;
                    WRITE_TIME_PIN_SetHigh();
                    SDCARD_WriteBlock(sdCardAddress, sdCardBuffer);
                    while ((status = SDCARD_PollWriteComplete()) ==
WRITE_NOT_COMPLETE);
                    WRITE_TIME_PIN_SetLow();

                    printf("Write block of decremented 8-bit values:\r\n");
                    printf("    Address:    ");
                    printf("%04x", sdCardAddress >> 16);
                    printf(":");
                    printf("%04x", sdCardAddress & 0X0000FFFF);
                    printf("\r\n");
                    printf("    Status:     %02x\r\n", status);
                    break;

                    //-------------------------------------------
                    // r: read a block of BLOCK_SIZE bytes from SD card

                    //-------------------------------------------
                case 'r':
                    READ_TIME_PIN_SetHigh();
                    SDCARD_ReadBlock(sdCardAddress, sdCardBuffer);
                    READ_TIME_PIN_SetLow();
                    printf("Read block: \r\n");
                    printf("    Address:    ");
```

```
                    printf("%04x", sdCardAddress >> 16);
                    printf(":");
                    printf("%04x", sdCardAddress & 0X0000FFFF);
                    printf("\r\n");
                    hexDumpBuffer(sdCardBuffer);
                    break;

                case '+':
                    if (sampleRate <= 170) {sampleRate += 10; printf("Sampling rate
set to %u us.\r\n", sampleRate);}
                    else {printf("Sampling rate set to max (180 us).\r\n");}
                    break;
                case '-':
                    if (sampleRate >= 30) {sampleRate -= 10; printf("Sampling rate
set to %u us.\r\n", sampleRate);}
                    else {printf("Sampling rate set to min (20 us).\r\n");}
                    break;
                case '1':
                    j = 0;
                    loops = 0;
                    printf("Writing to SD card. Press any key to quit.\r\n", loops/
2);
                    while (!EUSART1_DataReady) {
                        for (i = 0; i <= BLOCK_SIZE; i++) {
                            blueBuffer[i] = sin[(i + j) % SINE_WAVE_ARRAY_LENGTH];
                        }
                        j = i;
                        WRITE_TIME_PIN_SetHigh();
                        SDCARD_WriteBlock(sdCardAddress + loops * BLOCK_SIZE,
blueBuffer);
                        while ((status = SDCARD_PollWriteComplete()) ==
WRITE_NOT_COMPLETE);
                        WRITE_TIME_PIN_SetLow();
                        loops++;
                        for (i = 0; i <= BLOCK_SIZE; i++) {
                            redBuffer[i] = sin[(i + j) % SINE_WAVE_ARRAY_LENGTH];
                        }
                        j = i;
                        WRITE_TIME_PIN_SetHigh();
                        SDCARD_WriteBlock(sdCardAddress + loops * BLOCK_SIZE,
redBuffer);
                        while ((status = SDCARD_PollWriteComplete()) ==
WRITE_NOT_COMPLETE);
                        WRITE_TIME_PIN_SetLow();
                        loops++;
                    }
                    (void) EUSART1_Read();
                    lastBlock = loops/2;
                    printf("%u blocks recorded.\r\n", lastBlock);
                    break;

                case '/':
                case 'W':
                    printf("Double-Buffer ADC Read/SD Write operation.\r\n");
                    printf("Press any key to start recording audio and press any
key to stop recording.\r\n");
                    while (!EUSART1_DataReady);
                    (void) EUSART1_Read();
                    printf("Recording...\r\n");
```

```
                    audioFlag = 1;
                    loops = 0;
                    while(!EUSART1_DataReady) {
                        while (buffFlag != 1);
                        buffFlag = 0;
                        WRITE_TIME_PIN_SetHigh();
                        if (whichBuffer == RED) {
                            SDCARD_WriteBlock(sdCardAddress + loops * BLOCK_SIZE,
redBuffer);
                        }
                        if (whichBuffer == BLUE) {
                            SDCARD_WriteBlock(sdCardAddress + loops * BLOCK_SIZE,
blueBuffer);
                        }
                        while ((status = SDCARD_PollWriteComplete()) ==
WRITE_NOT_COMPLETE);
                        WRITE_TIME_PIN_SetLow();
                        loops++;
                    }
                    lastBlock = loops;
                    (void) EUSART1_Read();
                    audioFlag = 0;
                    printf("Recording complete.\r\n");
                    printf("%u blocks recorded.\r\n", lastBlock);
                    break;

                    //--------------
                    // Spooling code
                    //--------------
                case's':
                    for (i = 0; i < 40; i++) printf("\n");
                    printf("You may terminate spooling at anytime with a keypress.\
r\n");
                    printf("To spool terminal contents into a file follow these
instructions:\r\n\n");
                    printf("Right mouse click on the upper left of the PuTTY
window\r\n");
                    printf("Select:     Change settings...\r\n");
                    printf("Select:     Logging\r\n");
                    printf("Select:     Session logging: All session output\r\n");
                    printf("Log file name:  Browse and provide a .csv extension to
your file name\r\n");
                    printf("Select:     Apply\r\n\n");
                    printf("Press any key to start\r\n");

                    while (!EUSART1_DataReady);
                    (void) EUSART1_Read();

                    loops = 0;
                    while (true) {
                        bool breakWhile = false;
                        READ_TIME_PIN_SetHigh();
                        SDCARD_ReadBlock(sdCardAddress + (BLOCK_SIZE * loops),
sdCardBuffer);
                        READ_TIME_PIN_SetLow();
                        for (i = 0; i < BLOCK_SIZE; i++) {
                            printf("%u\r\n", sdCardBuffer[i]);
                            if(EUSART1_DataReady) {(void) EUSART1_Read();
breakWhile = true; break;}
```

```
                    }
                    if (breakWhile || loops == lastBlock) {break;}
                    loops++;
                }
                printf("Spooled %u out of the %u blocks.\r\n", loops,
lastBlock);
                printf("To close the file follow these instructions:\r\n\n");
                printf("Right mouse click on the upper left of the PuTTY
window\r\n");
                printf("Select:     Change settings...\r\n");
                printf("Select:     Logging\r\n");
                printf("Select:     Session logging : none\r\n");
                printf("Select:     Apply\r\n");
                break;
            case 'P':
                //Playback
                printf("Starting playback\r\n");
                whichBuffer = RED;
                READ_TIME_PIN_SetHigh();
                SDCARD_ReadBlock(sdCardAddress, redBuffer);
                READ_TIME_PIN_SetLow();
                playFlag = 1;
                i = 0;
                while (i <= lastBlock) {
                    i++;

                    if (whichBuffer == RED) {
                        READ_TIME_PIN_SetHigh();
                        SDCARD_ReadBlock(sdCardAddress + i * BLOCK_SIZE,
blueBuffer);
                        READ_TIME_PIN_SetLow();
                    }
                    else {
                        READ_TIME_PIN_SetHigh();
                        SDCARD_ReadBlock(sdCardAddress + BLOCK_SIZE * i,
redBuffer);
                        READ_TIME_PIN_SetLow();
                    }

                    while (buffFlag == 0);
                    buffFlag = 0;
                    if(EUSART1_DataReady) {
                        (void) EUSART1_Read();
                        printf("Completed %u of %u blocks.\r\n", i, lastBlock);
                        break;
                    }
                }
                playFlag = 0;
                printf("\r\nPlayback complete\r\n");
                break;
            case '2':
                EPWM1_LoadDutyValue(128);
                break;
                //------------------------------------------
                // If something unknown is hit, tell user
                //------------------------------------------
            default:
                printf("Unknown key %c\r\n", cmd);
                break;
```

```
            } // end switch

        } // end if
    } // end while
} // end main


//---------------------------------------------
// As configured, we are hoping to get a toggle
// every 100us - this will require some work.
//
// You will be starting an ADC conversion here and
// storing the results (when you reenter) into a global
// variable and setting a flag, alerting main that
// it can read a new value.
//
// !!!MAKE SURE THAT TMR0 has 0 TIMER PERIOD in MCC!!!!
//---------------------------------------------
#define WASTING_TIME    true
#define BASIC_TIME      false

void myTMR0ISR(void) {

    static uint16_t bufferIndex = 0;
    TMR0_WriteTimer(0);
    //TEST_PIN_SetHigh();
    ADCON0bits.GO_NOT_DONE = 1; // start a new conversion
    if (audioFlag == 1) {
        if (whichBuffer == RED) {blueBuffer[bufferIndex] = ADRESH;}
        else {redBuffer[bufferIndex] = ADRESH;}
        if (bufferIndex == BLOCK_SIZE - 1 && whichBuffer ==  RED) {bufferIndex = -
1; whichBuffer = BLUE; buffFlag = 1;}
        if (bufferIndex == BLOCK_SIZE - 1 && whichBuffer == BLUE) {bufferIndex = -
1; whichBuffer =  RED; buffFlag = 1;}
        bufferIndex++;
    }
    else if(playFlag == 1) {
        if (whichBuffer == RED) {EPWM1_LoadDutyValue(redBuffer[bufferIndex]);}
        else EPWM1_LoadDutyValue(blueBuffer[bufferIndex]);
        if (bufferIndex == BLOCK_SIZE - 1 && whichBuffer ==  RED) {bufferIndex = -
1; whichBuffer = BLUE; buffFlag = 1;}
        if (bufferIndex == BLOCK_SIZE - 1 && whichBuffer == BLUE) {bufferIndex = -
1; whichBuffer =  RED; buffFlag = 1;}
        bufferIndex++;
    }
    //TEST_PIN_SetLow();
    TMR0_WriteTimer( TMR0_ReadTimer() + (0x10000 - (16 * sampleRate)));   // More
accurate
    INTCONbits.TMR0IF = 0;
}

/* end of file */
```