```
/*
 * File:   csFlashOp.c
 * Author: Jared Fowler
 *
 * Created on February 26, 2015
 *
 * --The previous version of this file was scrapped. These functions were written
 * by me, Jared Fowler, tested, and found to be working. A lot of reference help
 * was received from microchip manuals and code examples. I will attempt to make
 * notes of what is going on in this file. For a description of how to use the
 * functions, please referr to the .h file.
 *
 * --March 4, 2015 - Jared Fowler - Modified the name of the write function to indicate
 * that it will only write to the low word of the 24 bit instruciton. Accordingly,
 * I wrote a separate function which will only read the low word. These come in
 * handy for working with the journal located in flash memory.
 *
 * --March 25, 2015 - Jared Fowler - Two of the functions have been manually placed
 * into flash memory in specific locations. This is to accomodate the function
 * 'reflashProgramImage', which will utilize these functions. Conceptially, these
 * need to be placed separately from the rest of the code as to allow the rest of
 * flash memory to be erased and written to.
 *
 *
 * ABOUT:   --Anyone working on this file should read all of this .....
 *
 * Flash memory is ordered by a table page and an address on that page. The table
 * page is a page of flash memoy which is composed of 512 instruction lines. Each
 * instruction line is 24bits, so the total change in address line numbers is 1024
 * for every page. For the operations below, we start by setting our working enviroment
 * to the correct table page, and offset within that table page. (Address) Many
 * times well want to assign variable values to registers. We can do this by
 * preceeding the variable name with an underscore '_'. The variable names need
 * to be in the global section of code.
 *
 * READ:
 *
 * After navigating to the right table page and offset we use the instructions
 * TBLRDL (table read low) and TBLRDH (table read high). These read the high byte
 * and the lower word which make up the 24bit instruciton. In this function I used
 * the #_flashHiWord to assign the registers to point to these variables. This way
 * the results from the table operations are read directly into my variables.
 *
 * ERASE/WRITE COMMON:
 *
 * There are two special registers associated with these commands. The NVMCON and
 * NVMKEY. THE NVMKEY is there to prevent you, or the system, from doing stupid things,
 * like erasing or writing to flash when you don't want to. For this reason, before
 * doing either of these operations you have to perform a series of writes to NVMKEY
 * followed immediately with the start process bit being set in NVMCON. This
 * series of operations can be seen below. The NVMCON register has its bits configured
 * in special ways to determine what operation it should do on flash. Below you will
 * see several hard-coded values.. these have to do with this. Of importance are bits
```

```
 * 14 and 15. Bit 14 enables write/erase mode and bit 15 starts the operation.
 *
 * WRITE:
 *
 * There is something known as a 'latch', which is the idea that we can fill up
 * a buffer which the system knows about and then ask it to write to flash. To fill
 * up this buffer we simply attempt to write to flash where we want the changes to
 * be made to. We use the instructions TBLWTL and TBLWTH (table write low, table write high)
 * After perfoming the series of operations for the NVMKEY we set the 15th bit of
 * NVMCON and the operation takes place. I've implimented the option to write single
 * words at a time or a full row of instructions (64 instructions). I should note that
 * if a bit has been set to 0 in flash memroy, the only way to set it to 1 is by
 * erasing it. Multiple writes on the same flash address can only set 1's to 0.
 *
 * ERASE:
 *
 * This operation needs to be done by pages. The page size of our device is 512
 * instructions. All the 24bit instrucitons will be erased to all 1's, or 0xFFFFFF.
 *
 *
 * OTHER/CAREFUL!!!:
 *
 * -Some values in assembly did not function as seen in manuals provided by microchip.
 * The #WR which represents the 15th bit of NVMCON, for example, is not recognized. I
 * also believe that several of the bit-set and check operations are not working, or
 * I did not use them properly. These were worked around with other commands.
 *
 * -If you step through the assembly in debug mode be careful not to interfere with
 * the series of operations for NVMKEY. In order to set the 15th bit in NVMCON, it
 * needs to be set the cycle right after the series is performed. If your stepping
 * through you will miss that cycle!
 *
 * -Interrupts need to be disabled during the NVMKEY cycle.
 *
 * -These funcitons are written to do some input error/warning checking, but really not that
 * much... A misuse of the Write and Erase functions could potentially destroy the
 * program image!
 *
 *                   JWF - jared.fowler.379@my.csun.edu
 */

#include "csFlashOP.h"


//------------------------------------------------------------------------------
UINT8 readProgramMemory(UINT32* dest, UINT32 addrStart, UINT32 n){

    //Local Variables
    UINT8 errorStatus = 0;
    UINT32 index;

    //Clear variable values...in case of space scrambling.
    flashMemAddr  = addrStart;
```

```
    flashHiAddr   = 0;
    flashLoAddr   = 0;
    flashHiWord   = 0;
    flashLoWord   = 0;

    //Loop!
    for(index = 0; index < n; index++){

        //Divide flashMemAddr into high and low words
        flashHiAddr = (flashMemAddr >> 16);
        flashLoAddr  = (UINT16)flashMemAddr;

        //Enter assembly snippit of code to read values from code memory into my variables
        asm(
            "PUSH       TBLPAG             ;Backup the current table page    \n"
            "MOV        _flashHiAddr, W0   ;Copy Table Page to register      \n"
            "MOV        W0, TBLPAG         ;Navigate to correct table page   \n"
            "MOV        _flashLoAddr, W0   ;Set out location to read in mem  \n"
            "MOV        #_flashHiWord, W1  ;Have W1 point to hiWord out      \n"
            "MOV        #_flashLoWord, W2  ;Have W2 point to loWord out      \n"
            "NOP                           ;                                 \n"
            "TBLRDL     [W0], [W2]         ;Read low instr. into lo word     \n"
            "NOP                           ;                                 \n"
            "TBLRDH     [W0++], [W1]       ;Read hi instr. into hi word      \n"
            "NOP                           ;                                 \n"
            "POP        TBLPAG             ;Restore table page               \n"
        );

        //Load the read values into the destination array
        *(dest + index) = (((UINT32)flashHiWord << 16) | flashLoWord);

        //Prepare the next memroy address to read
        flashMemAddr += FLASH_ADDRESS_INCREASE_PER_LINE;

    }

    return errorStatus;
}
//------------------------------------------------------------------------------
UINT8 eraseProgramMemoryPAGES(UINT32 addrStart, UINT16 n){

    //Local Variables
    BYTE errorStatus = 0;
    UINT16 index;

    //Error Check - Is addrStart an even address of 1024?
    if((addrStart % FLASH_ADDRESSES_DIFF_PER_ERASE) != 0)
        return (errorStatus | FLASH_ERROR_ERASE_NON_PG_MULT);

    //Clear variable values...in case of space scrambling.
    flashMemAddr  = addrStart;
    flashHiAddr   = 0;
    flashLoAddr   = 0;
```

```
    for(index = 0; index < n; index++){

        //Divide flashMemAddr into high and low words
        flashHiAddr = (UINT16)(flashMemAddr >> 16);
        flashLoAddr  = (UINT16)flashMemAddr;

        asm(
            "PUSH       TBLPAG              ;Backup location in table          \n"
            "MOV        _flashHiAddr, W0    ;Load up high address byte         \n"
            "MOV        W0, TBLPAG          ;Change to right table             \n"
            "MOV        _flashLoAddr, W0    ;Load up low address word          \n"
            "tblwtl     W0, [W0]            ;Dummy Write to select row         \n"
            "MOV        #0x4042, W7         ;0x4042 is page erase code         \n"
            "MOV        W7, NVMCON          ;Set Reg. for page erase           \n"
            "MOV        #0xC042, W7         ;Prepare W7 for start phase        \n"
            "MOV        #0x8000, W6         ;Prepare W6 for comparison         \n"

            "DISI       #7                  ;Disable IRQ for 7 inst.           \n"
            "MOV        #0x55, W0           ;Special key seq which             \n"
            "MOV        W0, NVMKEY          ;indicates that we are             \n"
            "MOV        #0xAA, W0           ;going to erase pgm                \n"
            "MOV        W0, NVMKEY          ;memory                            \n"
            "MOV        W7, NVMCON          ;Start!                            \n"
            "NOP                            ;Required NOP                      \n"
            "NOP                            ;Required NOP                      \n"

            "ERASE_WAIT:                                                       \n"
            "MOV        NVMCON, W7          ;Get current value of NVMCON       \n"
            "AND        W5, W6, W7          ;Is operation complete?            \n"
            "BRA        NZ, ERASE_WAIT      ;Jump to label                     \n"

            "POP        TBLPAG              ;Restore location in table         \n"
        );


        //Update to the next address location
        flashMemAddr += FLASH_ADDRESSES_DIFF_PER_ERASE;

    }

    return errorStatus;
}
//------------------------------------------------------------------------------
UINT8 writeProgramMemoryLoWord(UINT16* src, UINT32 addrStart, UINT32 n){

    //Local Variables
    BYTE errorStatus = 0;
    UINT16 index;

    //Clear variable values...in case of space scrambling.
    flashMemAddr  = addrStart;
    flashHiAddr   = 0;
```

```
    flashLoAddr   = 0;
    flashHiWord   = 0;
    flashLoWord   = 0;

    //Link flashPtr to the passed in source
    flashPtr = src;

    //Loop!
    for(index = 0; index < n; index++){

        //Divide flashMemAddr into high and low words
        flashHiAddr = (flashMemAddr >> 16);
        flashLoAddr  = (UINT16)flashMemAddr;

        //Get the value we are to write to flash
        flashLoWord = *flashPtr;

        asm(
            "PUSH       TBLPAG              ;Backup location in table          \n"

            "MOV        _flashHiAddr, W0    ;Load up high address byte         \n"
            "MOV        W0, TBLPAG          ;Change to right table             \n"
            "MOV        _flashLoAddr, W0    ;Load up low address word          \n"

            "MOV        _flashLoWord, W3    ;Load in the lo word to write      \n"
            "MOV        #0, W2              ;Load in the hi byte to write      \n"
            "TBLWTL     W3, [W0]            ;Write to latch                    \n"
            "TBLWTH     W2, [W0++]          ;Write to latch                    \n"

            "MOV        #0x4003, W7         ;Set up NVMCON to write to mem.    \n"
            "MOV        W7, NVMCON          ;                                  \n"
            "MOV        #0xC003, W7         ;Prepare W7 for start phase        \n"
            "MOV        #0x8000, W6         ;Prepare W6 for comparison         \n"

            "DISI       #7                  ;Disable Interrupts                \n"
            "MOV        #0x55, W0           ;Write the key sequence            \n"
            "MOV        W0, NVMKEY          ;                                  \n"
            "MOV        #0xAA, W0           ;                                  \n"
            "MOV        W0, NVMKEY          ;                                  \n"
            "MOV        W7, NVMCON          ;Start!                            \n"
            "NOP                            ;                                  \n"
            "NOP                            ;                                  \n"

            "WRITE_WAIT_S:                                                     \n"
            "MOV        NVMCON, W7          ;Get current value of NVMCON       \n"
            "AND        W5, W6, W7          ;Is operation complete?            \n"
            "BRA        NZ, WRITE_WAIT_S    ;Jump to label                     \n"

            "POP        TBLPAG              ;Restore location in table         \n"
        );

        //Update values for next iteration
        flashPtr += 1;
```

```
        flashMemAddr += sizeof(UINT16);
    }

    return errorStatus;
}
//------------------------------------------------------------------------------
UINT8 writeProgramMemoryROWS(UINT32* src, UINT32 addrStart, UINT32 n){

    //Local Variables
    BYTE errorStatus = 0;
    UINT16 index;

    //Error Check - Valid row start address?
    if((addrStart % FLASH_ADDRESSES_DIFF_PER_WRITE) != 0)
        return (errorStatus | FLASH_ERROR_ERASE_NON_PG_MULT);

    //Clear variable values...in case of space scrambling.
    flashMemAddr  = addrStart;
    flashHiAddr   = 0;
    flashLoAddr   = 0;
    flashHiWord   = 0;
    flashLoWord   = 0;

    //Link flashPtr to the passed in source
    flashPtr = (UINT16*)src;

    //Loop!
    for(index = 0; index < n; index++){

        //Divide flashMemAddr into high and low words
        flashHiAddr = (flashMemAddr >> 16);
        flashLoAddr  = (UINT16)flashMemAddr;

        //We will load up what we want to write, and then write it.
        asm(
            "PUSH       TBLPAG             ;Back up table page               \n"
            "MOV        _flashHiAddr, W0   ;Set the table page               \n"
            "MOV        W0, TBLPAG         ;                                 \n"
            "MOV        _flashLoAddr, W0   ;Set the table offset             \n"
            "MOV        #64, W3            ;Set up counter                   \n"
            "MOV        _flashPtr, W2      ;Have W2 point to values to write \n"

            "WR_PREP_LOOP:                 ;                                 \n"
            "TBLWTL     [W2++], [W0]       ;Write lower word instruction     \n"
            "TBLWTH     [W2++], [W0++]     ;Write high word instruction      \n"

            "DEC        W3, W3             ;Update loop count                \n"
            "BRA        NZ, WR_PREP_LOOP   ;If not done, loop back           \n"

            "MOV        #0x4001, W7        ;Set up NVMCON to write to mem.   \n"
            "MOV        W7, NVMCON         ;                                 \n"
            "MOV        #0xC001, W7        ;Prepare W7 for start phase       \n"
```

```
            "DISI       #7                  ;Disable Interrupts             \n"
            "MOV        #0x55, W0           ;Write the key sequence         \n"
            "MOV        W0, NVMKEY          ;                               \n"
            "MOV        #0xAA, W0           ;                               \n"
            "MOV        W0, NVMKEY          ;                               \n"
            "MOV        W7, NVMCON          ;Start!                         \n"
            "NOP                            ;                               \n"
            "NOP                            ;                               \n"

            "WRITE_WAIT_R:                                                  \n"
            "MOV        NVMCON, W7          ;Get current value of NVMCON    \n"
            "AND        W5, W6, W7          ;Is operation complete?         \n"
            "BRA        NZ, WRITE_WAIT_R    ;Jump to label                  \n"

            "POP        TBLPAG              ;Restore location in table      \n"
        );

        //Prepare the next block of instruction addresses to write to
        flashMemAddr += FLASH_ADDRESSES_DIFF_PER_WRITE;
        flashPtr += FLASH_ADDRESSES_DIFF_PER_WRITE;

    }

    return errorStatus;
}
//------------------------------------------------------------------------------
UINT8 readProgramMemoryLoWord(UINT16* dest, UINT32 addrStart, UINT32 n){

    //Local Variables
    BYTE errorStatus = 0;
    UINT16 index;

    //Clear variable values...in case of space scrambling.
    flashMemAddr  = addrStart;
    flashHiAddr   = 0;
    flashLoAddr   = 0;
    flashHiWord   = 0;
    flashLoWord   = 0;

    //Loop!
    for(index = 0; index < n; index++){

        //Divide flashMemAddr into high and low words
        flashHiAddr = (flashMemAddr >> 16);
        flashLoAddr  = (UINT16)flashMemAddr;

        //Enter assembly snippit of code to read values from code memory into my variables
        asm(
            "PUSH       TBLPAG              ;Backup the current table page  \n"
            "MOV        _flashHiAddr, W0    ;Copy Table Page to register    \n"
            "MOV        W0, TBLPAG          ;Navigate to correct table page \n"
            "MOV        _flashLoAddr, W0    ;Set out location to read in mem \n"
            "MOV        #_flashLoWord, W2   ;Have W2 point to loWord out    \n"
```

```
            "NOP                            ;                                   \n"
            "TBLRDL     [W0++], [W2]        ;Read low instr. into lo word       \n"
            "NOP                            ;                                   \n"
            "POP        TBLPAG              ;Restore table page                 \n"
        );

        //Load the read values into the destination array
        *(dest + index) = flashLoWord;

        //Prepare the next memroy address to read
        flashMemAddr += FLASH_ADDRESS_INCREASE_PER_LINE;

    }

    return errorStatus;
}
```