

```

//code by Brendand Halstead

#include <project.h>

void WREN();
void transmit_address(uint8_t hi, uint8_t med, uint8_t low);

int main()
{
    CyGlobalIntEnable; /* Enable global interrupts. */

    SPIM_1_Start();

    LCD_Start();
    LCD_DisplayOn();

    uint8_t addr_hi = 0x00;
    uint8_t addr_med = 0x00;
    uint8_t addr_low = 0x00;
    //our starting address is going to be at the bottom of memory, 0x000000

    uint8_t num_strings = 0; // We'll use this to keep track of which strings ↗
we've written to the chip
    char * strings[] = {"6.115", " is", " cool!!!"}; // These are the example ↗
strings we'll be writing to the chip
    char read_string[16] = {0}; // This will hold the string we read back ↗
from the chip

    // We need to disable block protection by writing to the status register.

    SS_Write(0); // pull SS active-low to start the Write Enable command
    SPIM_1_WriteTxData(0x06); // opcode for WREN command so we can write to ↗
the status register
    while(!(SPIM_1_ReadTxStatus() & SPIM_1_STS_SPI_DONE)); // wait for ↗
tranceive cycle to finish before pulling SS high
    SS_Write(1); // pull SS high to latch the Write Enable command

    SS_Write(0); // pull SS active-low to start the Write Status Register ↗
command
    SPIM_1_WriteTxData(0x01); // WRSR command
    SPIM_1_WriteTxData(0x00); // Writes all bits in status register to 0, ↗
disabling block protection.
    while(!(SPIM_1_ReadTxStatus() & SPIM_1_STS_SPI_DONE)); // wait for ↗
tranceive cycle to finish before pulling SS high
    SS_Write(1); // pull SS high to latch the Write Status Register command

    for(;;)
    {

```

```

main.c
    if(!SW2_Read()){ // when we press SW2, we will write another string ↗
to the flash memory

        if (num_strings < 3){ // write the next string to the flash chip

            uint8_t length = strlen(strings[num_strings]);

            int i;
            for(i=0; i<length; i++) {
                WREN(); // Must execute WREN before every write command
                SS_Write(0); // pull SS active-low to start the Byte ↗
Program command
                SPIM_1_WriteTxData(0x02); // transmit the opcode for Byte ↗
Program
                transmit_address(addr_hi, addr_med, addr_low + i); // ↗
transmit the three-byte address
                SPIM_1_WriteTxData(strings[num_strings][i]); // transmit ↗
the byte encoding the character
                while(!(SPIM_1_ReadTxStatus() & SPIM_1_STS_SPI_DONE)); // ↗
wait for tranceive cycle to finish before pulling SS high
                SS_Write(1); // pull SS high to latch the Byte Program ↗
command
            }

            addr_low += length; // set the writing address for the next ↗
string
            num_strings += 1; // move on to the next string
        }
    else { //if we have already written all three parts, erase on ↗
next SW2 press
        WREN(); //First, send a WREN

        SS_Write(0); // pull SS active-low to start the Sector Erase ↗
command
        SPIM_1_WriteTxData(0x20); // transmit the opcode for a 4kB ↗
Sector Erase
        transmit_address(0,0,0); // transmit the address of the ↗
sector to be erased
        while(!(SPIM_1_ReadTxStatus() & SPIM_1_STS_SPI_DONE)); // ↗
wait for tranceive cycle to finish before pulling SS high
        SS_Write(1); // pull SS high to latch the Sector Erase command

        CyDelay(25); //Per the datasheet, the 4 kB erase takes at ↗
most 25 ms.
        //This is a simple way to ensure that the erase operation is ↗
complete before sending more commands
    }
    CyDelay(250); // simple way to count button presses (limit ↗
polling frequency to at most 4 Hz)
}
    else if(!SW3_Read()){ // when we press SW3, we will read from flash ↗
memory and display on the LCD

```

main.c

```
        SS_Write(0); // pull SS active-low to start the Read command
        SPIM_1_WriteTxData(0x03); //transmit the opcode for Read
        transmit_address(0x00, 0x00, 0x00); // start the read back at the ↵
base of the flash chip
        while(!(SPIM_1_ReadTxStatus() & SPIM_1_STS_SPI_DONE)); // wait ↵
for tranceive cycle to finish before clearing the RX buffer
        SPIM_1_ClearRxBuffer(); //get junk data out of the RX buffer ↵
before the chip fills it with actual data

        int i;
        for (i = 0; i < 16; i ++){ //sequentially read the data from the ↵
flash chip
                SPIM_1_WriteTxData(0xAA); // write dummy data to run the clock
                while(!(SPIM_1_ReadTxStatus() & SPIM_1_STS_SPI_DONE)); // ↵
make sure each tranceive cycle is finished before we read from buffer
                read_string[i] = SPIM_1_ReadRxData(); // put each ↵
received byte into the string buffer
        }
        SS_Write(1); // pull SS high to terminate the Read command

        LCD_ClearDisplay();
        LCD_PrintString(read_string);
        // Display the string on the LCD.
        CyDelay(250); // simple way to count button presses (limit ↵
polling frequency to at most 4 Hz)
    }
}

void WREN (){
    SS_Write(0);
    SPIM_1_WriteTxData(0x06);
    while(!(SPIM_1_ReadTxStatus() & SPIM_1_STS_SPI_DONE));
    SS_Write(1);
}

void transmit_address (uint8_t hi, uint8_t med, uint8_t low){
    SPIM_1_WriteTxData(hi);
    SPIM_1_WriteTxData(med);
    SPIM_1_WriteTxData(low);
}

/* [] END OF FILE */
```