```
                        ;ME218C LiFKIM Test, Matt Hoffman, 05/11/2014
                        ;Added SPI functinality on 05/12/2014
                        ;Modified for vehicle integration on 5/21/2014

                        ;initial file set-up:
list P=PIC16F690
#include "p16F690.inc"
__config(_CP_OFF & _WDT_OFF & _PWRTE_ON & _HS_OSC & _MCLRE_OFF);

                        ;variable definitions
last_send_switch        equ     0x20
current_send_switch     equ     0x21
TX_data_LSB             equ     0x22
TX_data_MSB             equ     0x23
RX_data_LSB             equ     0x24
RX_data_MSB             equ     0x25
TX_UART_Message         equ     0x26
RX_UART_Message         equ 0x27
TX_SPI_Message          equ     0x28
RX_SPI_Message          equ     0x29
last_active_switch      equ     0x30    ;for debugging, sends LiFKIM Active command
current_active_switch   equ     0x31
WREG_TEMP               equ     0x70    ;make variables associated with ISR accessible from any bank
STATUS_TEMP             equ     0x71
PCLATH_TEMP             equ     0x72

        org     0

        pagesel main
        goto    main

        org     4       ;vector for interrupt service routine

ISR:

PUSH:
        movwf   WREG_TEMP       ;save WREG
        movf    STATUS,W        ;store STATUS in WREG
        clrf    STATUS          ;change to file register bank0
        movwf   STATUS_TEMP     ;save STATUS value
        movf    PCLATH,W        ;store PCLATH in WREG
        movwf   PCLATH_TEMP     ;save PCLATH value
        clrf    PCLATH          ;change to program memory page0

ISR_BODY:                       ;ISR should configure Bank as required
                        ;identify interrupt source
                        ;check for UART transmit register empty ONLY if we have enabled the interrupt
                        ;ISR might have a different source, in which case we don't want to transmit just
because TXIF is set
TXIE_test:
        pagesel TXIF_test
        banksel PIE1
        btfsc   PIE1, TXIE
        goto    TXIF_test
        pagesel SPIF_test
        goto    SPIF_test

TXIF_test:
        clrf    STATUS;return to bank 0
        pagesel TXIF_RESPONSE
```

```
        btfsc   PIR1, TXIF
        goto    TXIF_RESPONSE
        pagesel SPIF_test
        goto    SPIF_test

SPIF_test:
        clrf    STATUS;return to bank 0
        pagesel SPIF_RESPONSE
        btfsc   PIR1, SSPIF
        goto    SPIF_RESPONSE
        pagesel POP
        goto    POP

TXIF_RESPONSE:
        clrf    STATUS          ;ensure that we're in bank 0
        movf    TX_UART_Message, w   ;write message to TXREG to transmit
        movwf   TXREG
        banksel PIE1            ;disable TX interrupt since we've just sent 1 of 1 bytes
        bcf     PIE1, TXIE
        clrf    STATUS          ;return to bank 0
        pagesel POP
        goto    POP

SPIF_RESPONSE:
        banksel PIR1
        bcf             PIR1, SSPIF     ;clear SPI interrupt flag
        clrf    STATUS          ;return to bank 0
        movf    SSPBUF, w       ;check to see if message is a command to be forwarded (value <= 0x0f)
        movwf   RX_SPI_Message
        movf    RX_data_LSB, w ;see if this will maintain the data we want to send out in the register
        movwf   SSPBUF
        movf    RX_SPI_Message, w
        pagesel Forward_Message
        sublw   0x0f
        btfsc   STATUS, C
        goto    Forward_Message
        pagesel POP                     ;if message was > 0x0f, it was either a ping or errant
        goto    POP

Forward_Message:
                                        ;;complement message for data format required by LiFKIM
        clrf    STATUS;ensure that we're in bank 0
        swapf   RX_SPI_Message, w
        movwf   TX_SPI_Message
        comf    TX_SPI_Message, w
        andlw   0xf0
        addwf   RX_SPI_Message, w
        movwf   TX_SPI_Message
        movwf   TX_UART_Message         ;prepare message for transmission to LiFKIM
        banksel PIE1
        bsf             PIE1, TXIE      ;enable UART TX interrupt, let ISR handle sending the TX_Message
        ;movwf TXREG
        clrf    STATUS;return to bank 0
        pagesel POP
        goto    POP

POP:
        clrf    STATUS          ;select bank 0
        movf    PCLATH_TEMP,W           ;store saved PCLATH value in WREG
        movwf   PCLATH          ;restore PCLATH
```

```
        movf    STATUS_TEMP,W       ;store saved STATUS value in WREG
        movwf   STATUS          ;restore STATUS
        swapf   WREG_TEMP,F     ;prepare WREG to be restored
        swapf   WREG_TEMP,W         ;restore WREG keeping STATUS bits
        retfie              ;return from interrupt



main:
                    ;initialize PIC microcontroller
        banksel         ANSEL   ;configure ports to be digital
        clrf    ANSEL
        clrf    ANSELH
                    ;leave most of Port C initialized as set (input)
        bsf     STATUS, RP0     ;configure TRIS registers
        bcf     STATUS, RP1
        bcf     TRISC, TRISC5   ;set Port C <3:5> as outputs
        bcf     TRISC, TRISC4
        bcf     TRISC, TRISC3
        bcf     TRISB, TRISB7   ;set TX pin as output
        bcf     TRISC, TRISC7   ;set SDO pin as output

        bcf     OSCCON, SCS     ;configure clock to use external oscillator

                    ;Configure UART for async. comm., 9600 baud, 8 data bits, 1 stop bit (parity N/A on
PIC)
        bcf     BAUDCTL, SCKP;non-inverted polarity
        bcf     BAUDCTL, BRG16 ;use 8-bit baud divisor
        bcf     BAUDCTL, ABDEN      ;disable auto-baud detection
        bcf     BAUDCTL, WUE ;start with UART system active
        clrf    SPBRGH          ;clear upper byte of baud divisor, shouldn't be necessary since we're using
8-bit divisor
        movlw   0x81    ;configure baud rate for 9600 BPS
        movwf   SPBRG
        bcf     TXSTA, TX9      ;TX 8-bit data
        bsf     TXSTA, TXEN     ;enable transmitter
        bcf     TXSTA, SYNC     ;set UART for asynchronous communication
        bsf     TXSTA, BRGH     ;set MSB for baud rate
        bcf     TXSTA, SENDB    ;don't send a break character now
        bsf     INTCON, PEIE    ;enable peripheral interrupts
        ;bsf    PIE1, RCIE      ;enable receiver interrupt
        clrf    STATUS;return to bank 0 for editing RCSTA
        bsf     RCSTA, SPEN     ;enable UART
        bcf     RCSTA, RX9      ;RX 8-bit data
        bsf     RCSTA, CREN     ;enable receiver
        bcf     RCSTA, ADDEN    ;disables address mode detection, not applicable here since we're set to use
8-bit data

                    ;Configure SPI for Mode 3 operation, slave, slave-select enabled
        banksel SSPSTAT
        bcf             SSPSTAT, SMP
        bcf             SSPSTAT, CKE    ;SPI Mode 3
        bcf             SSPSTAT, BF             ;Clear SPI receive buffer full flag
        movf    SSPSTAT, w              ;Clear SPI receive buffer full flag
        clrf    STATUS                  ;return to bank 0
        bsf             SSPCON, SSPEN;enable SPI system
        bsf             SSPCON, CKP             ;SPI Mode 3
        bcf             SSPCON, SSPM3           ;SSPM<3:0> = 0100 for slave mode, slave select enabled
        bsf     SSPCON, SSPM2
        bcf             SSPCON, SSPM1
```

```
        bcf     SSPCON, SSPM0
        bcf     PIR1, SSPIF             ;clear SPI interrupt flag
        banksel PIE1
        bsf     PIE1, SSPIE             ;enable SPI interrupts
        clrf    STATUS                  ;return to bank 0
        bsf     INTCON, PEIE    ;enable peripheral interrupts


                ;initialize variables
        clrf    TX_UART_Message         ;initialize TX and RX to 0 for debugging purposes
        clrf    RX_UART_Message
        clrf    TX_SPI_Message
        clrf    RX_SPI_Message
        clrf    SSPBUF                  ;initialize with 0x00 in SPI transmit buffer
        clrf    PORTC   ;initialize with all LEDs turned off
        clrf    current_send_switch
        clrf    last_send_switch
        clrf    current_active_switch
        clrf    last_active_switch
        btfsc   PORTA, RA0
        bsf     last_send_switch, 0
        btfsc   PORTC, RC0
        bsf     last_active_switch, 0

                ;clear RX interrupt flags (should be done by reading RCREG, I don't think this can
be software-cleared)
        bcf     PIR1, RCIF
        movf    RCREG, w

        bsf     INTCON, GIE     ;enable interrupts


;; Start main loop

event_checker:                  ;start event checking loop
        clrf    STATUS;ensure that we are in Bank 0

event_check_0:                  ;read state on current activate switch, compare to previous value
        clrf    current_active_switch
        btfsc   PORTC, RC0
        bsf     current_active_switch, 0
        movf    current_active_switch, w
        pagesel active_switch_toggled
        subwf   last_active_switch, w
        btfss   STATUS, Z
        goto    active_switch_toggled
        movf    current_active_switch, w        ;store last switch read
        movwf   last_active_switch
        pagesel event_check_1
        goto    event_check_1

active_switch_toggled:          ;check to see if current active switch is 1
        pagesel Activate_LiFKIM
        btfsc   current_active_switch, 0
        call    Activate_LiFKIM
        movf    current_active_switch, w        ;store last switch read
        movwf   last_active_switch
        pagesel event_check_1
        goto    event_check_1
```

```
event_check_1:                  ;read state on current message send switch, compare to previous value
        clrf    current_send_switch
        btfsc   PORTA, RA0
        bsf             current_send_switch, 0
        movf    current_send_switch, w
        pagesel send_switch_toggled
        subwf   last_send_switch, w
        btfss   STATUS, Z
        goto    send_switch_toggled
        movf    current_send_switch, w ;store last switch read
        movwf   last_send_switch
        pagesel event_check_2
        goto    event_check_2

send_switch_toggled:            ;check to see if current send switch is 1
        pagesel Send_UART_Message_Switches
        btfsc   current_send_switch, 0
        call    Send_UART_Message_Switches
        movf    current_send_switch, w ;store last switch read
        movwf   last_send_switch
        pagesel event_check_2
        goto    event_check_2

event_check_2                   ;check to see if new message is available
        pagesel Receive_UART_Message
        btfsc   PIR1, RCIF
        call    Receive_UART_Message
        pagesel event_checker
        goto    event_checker

;; end main loop

Activate_LiFKIM:                ;Transmit command for LiFKIM to enter active state
        movlw   0x0e            ;Command for LiFKIM to enter active state
        movwf   TX_data_LSB
        swapf   TX_data_LSB, w;complement input message in MSB of TX message for LiFKIM protocol
        movwf   TX_UART_Message     ;move MSB of TX message into TX variable
        bcf             TX_UART_Message, 0   ;zero out the LSBs in TX message
        bcf             TX_UART_Message, 1
        bcf             TX_UART_Message, 2
        bcf             TX_UART_Message, 3
        movf    TX_data_LSB, w;combine MSB and LSB into single TX message
        addwf   TX_UART_Message, f
        bsf             STATUS, RP0    ;switch to bank with PIE1 register
        bsf             PIE1, TXIE     ;enable UART TX interrupt, let ISR handle sending the TX_Message
        clrf    STATUS;return to bank 0
        return

Send_UART_Message_Switches:     ;read state of input switches and transmit to LiFKIM
        clrf    TX_data_LSB
        btfsc   PORTA, RA1
        bsf             TX_data_LSB, 0
        btfss   PORTA, RA1
        bcf             TX_data_LSB, 0
        btfsc   PORTA, RA2
        bsf             TX_data_LSB, 1
        btfss   PORTC, RC2
        bcf             TX_data_LSB, 1
        swapf   TX_data_LSB, w;complement input message in MSB of TX message for LiFKIM protocol
        movwf   TX_UART_Message     ;move MSB of TX message into TX variable
```

```
        bcf             TX_UART_Message, 0    ;zero out the LSBs in TX message
        bcf             TX_UART_Message, 1
        bcf             TX_UART_Message, 2
        bcf             TX_UART_Message, 3
        movf    TX_data_LSB, w;combine MSB and LSB into single TX message
        addwf   TX_UART_Message, f
        bsf             STATUS, RP0     ;switch to bank with PIE1 register
        bsf             PIE1, TXIE      ;enable UART TX interrupt, let ISR handle sending the TX_Message
        clrf    STATUS;return to bank 0
        return

Receive_UART_Message:                   ;copy RCREG into a memory variable, decode the message, and
output appropriate LED pattern
        movf    RCREG, w
        movwf   RX_UART_Message
                        ;verify that message is valid by checking to see if 4 MSBs complement 4 LSBs
        movf    RX_UART_Message, w
        ANDLW 0x0f      ;zero out MSBs
        movwf   RX_data_LSB     ;store LSBs
        swapf   RX_data_LSB, w
        movwf   RX_data_MSB
        comf    RX_data_MSB, w
        ANDLW 0xf0      ;zero out LSBs
        movwf   RX_data_MSB     ;store EXPECTED values for MSBs
        addwf   RX_data_LSB, w ;check to see if expected reading matches the actual message reading
        pagesel valid_UART_message
        subwf   RX_UART_Message, w
        btfsc   STATUS, Z
        goto    valid_UART_message
        return          ;return if a format of LiFKIM message is invalid

valid_UART_message:
        movf    RX_data_LSB, w
        movwf   SSPBUF                  ; write valid message to SPI buffer for transmitting during next
reply
                                                ; output message to visual LEDs if message
received is an energy status update
        sublw   0x07
        btfss   STATUS, C
        return
        btfsc   RX_data_LSB, 2
        bsf             PORTC, RC5
        btfss   RX_data_LSB, 2
        bcf             PORTC, RC5
        btfsc   RX_data_LSB, 1
        bsf             PORTC, RC4
        btfss   RX_data_LSB, 1
        bcf             PORTC, RC4
        btfsc   RX_data_LSB, 0
        bsf             PORTC, RC3
        btfss   RX_data_LSB, 0
        bcf             PORTC, RC3
        return

END
```