```c
/*****************************************************************************
 * EE344 Lab5Main.c
 *     Cooperative multitasking security system. Starts in an armed state and
 *     several triggers cause an alarm with accompanying speaker output.
 *     Utilizes electrode touch sensors, LEDs, accelerometer, LCD screen,
 *     and keypad to control all aspects of alarm system.
 *
 * Several modules based off designs by Todd Morton.
 * Brian Willis, 12/8/2017
 *****************************************************************************/
#include "MCUType.h"
#include "K65TWR_GPIO.h"
#include "key.h"
#include "LCD.h"
#include "SysTickDelay.h"
#include "AlarmWave.h"
#include "Sensor.h"
#include "Temp.h"
#include "Accel.h"
#include "BasicIO.h"
#include "TermInterface.h"
#include "WDOG.h"

#define TIMESLICE 100                                                   //Kernel timeslice in units of 100us
#define CNTRLDISPTASK_PER 30                                            //Max execution time of tasks in ms
#define LEDTASK_PER 50
#define TEMPALARMCELL 0                                                 //Temperature alarm thresholds
#define TEMPALARMCELH 40
#define TEMPALARMFAHRL ((((TEMPALARMCELL*9000000)/5)+32000000)/1000000)
#define TEMPALARMFAHRH ((((TEMPALARMCELH*9000000)/5)+32000000)/1000000)
#define TEMPCONVCEL(T) (INT32S)(((T*2582)-20510000)/1000000)            //Convert ADC output to temperature
#define TEMPCONVFAHR(T) (INT32S)((((((T*2582)-20510000)*9)/5)+32000000)/1000000)

typedef enum{F, C} TEMPDISPLAY;
typedef enum{ON, OFF} TRIGGER;

extern ALARM_STATES AlState;
extern TRIGGER WatchdogTrigger;

static void ControlDisplayTask(void);
static void LEDTask(void);

void main(void){
    //Initialization
    LcdInit();
    (void)SysTickDlyInit();
    KeyInit();
    GpioDBugBitsInit();
    AlarmWaveInit();
    GpioLED8Init();
    GpioLED9Init();
    SensorInit();
```

```
53      TempInit();
54      AccelInit();
55      BIOOpen();
56
57      BIOOutCRLF();                           //Clear terminal and LCD
58      LcdClrDisp();
59
60      LcdMoveCursor(1, 1);                    //Alarm resets to ARMED state
61      LcdDispStrg("ARMED   ");
62
63      WDOGInit();                             //Initialize Watchdog immediately before main loop
64
65      if(WatchdogTrigger == ON){
66          LcdMoveCursor(2, 15);
67          LcdDispStrg("W!");
68      } else{}
69
70      //Main loop
71      while(1){                               //Cooperative multitasking kernel
72          SysTickWaitEvent(TIMESLICE);        //Wait x units of 100us before continuing
73          WDOGTask();                         //Reset Watchdog timer every timeslice
74          KeyTask();                          //Place user key press into buffer
75          ControlDisplayTask();               //Control LCD display, set alarm state
76          LEDTask();                          //Control LEDs
77          TermInterfaceTask();                //Display CPU load and task execution time to terminal for debugging
78          SensorTask();                       //Scan Electrodes
79      }
80  }
81
82
83  /*************************************************************************
84  * ControlDisplayTask() - Controls state of alarm and displays to LCD
85  *
86  * Brian Willis, 12/8/2017
87  *************************************************************************/
88  static void ControlDisplayTask(void){
89      INT8C key;
90      INT32S new_temp;
91      static INT32S Temp;
92      static TRIGGER TempTrigger = OFF;
93      static TRIGGER TempTriggerPrev = ON;        //Default to ON because of temperature defaults
94      static TEMPDISPLAY TempDisplayState = C;
95      static TRIGGER TamperTrigger = OFF;
96      static TRIGGER TamperTriggerPrev = ON;      //Default to ON because of accelerometer register defaults
97      static ALARM_STATES AlStatePrev = ARMED;
98      static INT8U ControlDisplayTaskCntr = 0;
99
100     DB1_TURN_ON();
101
102     if(ControlDisplayTaskCntr == ((10*CNTRLDISPTASK_PER)/TIMESLICE)){       //Call once every period of task
103         ControlDisplayTaskCntr = 0;                                         //Reset counter
104         key = GetKey();                                                     //Place key from buffer into variable
```

```
105
106         TamperTrigger = (AccelPoll() == 1) ? ON : OFF;
107
108         //Calculate temperature, display when change (Max 2Hz)
109         new_temp = (TempDisplayState == C) ? TEMPCONVCEL(TempGet()) : TEMPCONVFAHR(TempGet());
110         if(Temp != new_temp){
111             //Display temperature in Celsius or Fahrenheit
112             if(TempDisplayState == C){
113                 //Check if temperature is over alarm thresholds
114                 if((new_temp < TEMPALARMCELL) || (new_temp > TEMPALARMCELH)){
115                     TempTrigger = ON;
116                 } else{
117                     TempTrigger = OFF;
118                 }
119             } else{
120                 //Check if temperature is over alarm thresholds
121                 if((new_temp < TEMPALARMFAHRL) || (new_temp > TEMPALARMFAHRH)){
122                     TempTrigger = ON;
123                 } else{
124                     TempTrigger = OFF;
125                 }
126             }
127         } else{}
128
129         //If temperature was within alarm thresholds or in disarmed state, display to LCD
130         if(((Temp != new_temp) && (TempTrigger == OFF)) || (AlState == DISARMED)){
131             LcdMoveCursor(2, 1);
132             if(new_temp < 0){                                   //If negative temperature, perform 2's complement
133                 new_temp = ~new_temp + 1;
134                 LcdDispChar('-');
135             } else{}
136
137             LcdDispDecWord((INT32U)new_temp, 2);                //Display converted temperature to LCD
138             LcdDispChar((INT8C)0xDF);
139             if(TempDisplayState == C){
140                 LcdDispStrg("C      ");
141             } else{
142                 LcdDispStrg("F      ");
143             }
144         } else{}
145
146         //Update Fahrenheit vs Celsius display
147         if(key == DC2){
148             TempDisplayState = (TempDisplayState == C) ? F : C; //Toggle between Celsius and Fahrenheit
149         } else{}
150
151         //Update alarm states
152         if((AlState == ARMED) && (tsiSensorFlags != 0)){        //Electrode alarm
153             AlState = ALARM;
154         } else{}
155
156         if((AlState == DISARMED) && (key == DC1)){              //User armed alarm
```

```
157             AlState = ARMED;
158         } else{}
159 
160         if(key == DC4){                                         //User disarmed alarm
161             AlState = DISARMED;
162         } else{}
163 
164         //Only write to LCD once per alarm trigger and update to alarm state
165         if((AlState != DISARMED) && (TempTrigger == ON)
166         && ((TempTrigger != TempTriggerPrev) || (AlStatePrev == DISARMED))){
167             AlState = ALARM;
168             LcdMoveCursor(2, 1);
169             LcdDispStrg("TEMP ALARM ");
170         } else{}
171 
172         //Display tampering trigger to LCD on state change
173         if((TamperTrigger == ON) && (TamperTrigger != TamperTriggerPrev)){
174             LcdMoveCursor(2, 12);           //Always display tampering trigger warning
175             LcdDispStrg("T!");
176         } else{}
177 
178         //Display alarm states to LCD on state change
179         if(AlStatePrev != AlState){
180             switch(AlState){
181                 case(ALARM):
182                     LcdMoveCursor(1, 1);
183                     LcdDispStrg("ALARM   ");
184                     break;
185                 case(ARMED):
186                     LcdMoveCursor(1, 1);
187                     LcdDispStrg("ARMED   ");
188                     break;
189                 case(DISARMED):
190                     LcdMoveCursor(1, 1);
191                     LcdDispStrg("DISARMED");
192                     break;
193                 default:
194                     break;
195             }
196         } else{}
197 
198         //Clear tampering display from LCD
199         if(key == DC3){
200             LcdMoveCursor(2, 12);
201             LcdDispStrg("  ");
202         } else{}
203         Temp = new_temp;
204         TempTriggerPrev = TempTrigger;
205         TamperTriggerPrev = TamperTrigger;
206         AlStatePrev = AlState;
207     } else{}
208     ControlDisplayTaskCntr++;
```

```
209         DB1_TURN_OFF();
210     }
211
212
213     /*************************************************************************
214     * LEDTask() - Controls LEDs
215     *
216     * Brian Willis, 12/8/2017
217     *************************************************************************/
218     static void LEDTask(void){
219         static INT8U LEDTaskCntr = 0;
220         static INT8U LEDActCntr = 0;                                //LED activation counter
221         static TRIGGER Elec1Trigger = OFF;
222         static TRIGGER Elec2Trigger = OFF;
223
224         DB4_TURN_ON();
225
226         if(LEDTaskCntr == ((10*LEDTASK_PER)/TIMESLICE)){            //Call once every period of task
227             LEDTaskCntr = 0;                                        //Reset counter
228
229             switch(AlState){
230                 //Flash LEDs with period of 500ms
231                 case(ARMED):
232                     if(LEDActCntr == 5){te
233                         LED9_TURN_ON();
234                         LED8_TURN_OFF();
235                     } else if(LEDActCntr >= 10){
236                         LED8_TURN_ON();
237                         LED9_TURN_OFF();
238                         LEDActCntr = 0;                             //Reset counter
239                     } else{}
240                     break;
241                 //Flash LEDs corresponding to respective electrodes with period of 100ms
242                 case(ALARM):
243                     if((tsiSensorFlags & (1 << ELECTRODE1)) != 0){  //Electrode is pressed, trigger LED flashing
244                         Elec1Trigger = ON;
245                     } else{}
246                     if((tsiSensorFlags & (1 << ELECTRODE2)) != 0){
247                         Elec2Trigger = ON;
248                     } else{}
249
250                     if(Elec1Trigger == ON){                         //If Electrode was triggered, continuously flash LED
251                         if(LEDActCntr == 1){
252                             LED8_TURN_ON();
253                         } else if(LEDActCntr >= 2){
254                             LED8_TURN_OFF();
255                         } else{}
256                     } else{
257                         LED8_TURN_OFF();
258                     }
259
260                     if(Elec2Trigger == ON){
```

```
                    if(LEDActCntr == 1){
                        LED9_TURN_ON();
                    } else if(LEDActCntr >= 2){
                        LED9_TURN_OFF();
                    } else{}
                } else{
                    LED9_TURN_OFF();
                }

                if(LEDActCntr >= 2){                            //Reset counter
                    LEDActCntr = 0;
                } else{}
                break;
            //Flash LEDs corresponding to respective electrodes with period of 500ms
            case(DISARMED):
                Elec1Trigger = OFF;                             //Reset triggers for ALARM state
                Elec2Trigger = OFF;

                if((tsiSensorFlags & (1 << ELECTRODE1)) != 0){  //Flash LEDs only while electrode is pressed
                    if(LEDActCntr == 5){
                        LED8_TURN_ON();
                    } else if(LEDActCntr >= 10){
                        LED8_TURN_OFF();
                    } else{}
                } else{
                    LED8_TURN_OFF();
                }

                if((tsiSensorFlags & (1 << ELECTRODE2)) != 0){
                    if(LEDActCntr == 5){
                        LED9_TURN_ON();
                    } else if(LEDActCntr >= 10){
                        LED9_TURN_OFF();
                    } else{}
                } else{
                    LED9_TURN_OFF();
                }

                if(LEDActCntr >= 10){                            //Reset counter
                    LEDActCntr = 0;
                } else{}
                break;
            default:
                break;
        }
        LEDActCntr++;
    } else{}
    LEDTaskCntr++;
    DB4_TURN_OFF();
}
```

```c
/**************************************************************************
 * Temp.c - Controls temperature scanning
 *
 * Brian Willis, 12/8/2017
 **************************************************************************/
#include "MCUType.h"
#include "K65TWR_GPIO.h"
#include "Temp.h"

/**************************************************************************
 * TempInit() - Initializes temperature scanner
 *
 * Brian Willis, 12/8/2017
 **************************************************************************/
void TempInit(void){
    //Enable PIT1
    SIM_SCGC6 |= SIM_SCGC6_PIT_MASK;         //Start SCGC6 clock for PIT
    PIT_MCR &= ~PIT_MCR_MDIS_MASK;           //Enable PIT via MCR
    PIT_LDVAL1 |= 0x1C9C37F;                 //Set PIT timer to 29999999 counts
    PIT_TCTRL1 |= PIT_TCTRL_TEN_MASK;        //Enable timer
    PIT_TCTRL1 |= PIT_TCTRL_TIE_MASK;        //Enable timer interrupt

    //Enable ADC0
    SIM_SCGC6 |= SIM_SCGC6_ADC0(1);          //Start SCGC6 clock for ADC0
    ADC0_CFG1 |= ADC_CFG1_ADIV(3);           //Divide bus clock by 8
    ADC0_CFG1 |= ADC_CFG1_MODE(3);           //Set to 16 bit samples
    ADC0_CFG1 |= ADC_CFG1_ADLSMP(1);         //Set to long samples
    ADC0_SC2 |= ADC_SC2_ADTRG(1);            //Set ADC0 for hardware trigger
    ADC0_SC3 |= ADC_SC3_AVGE(1);             //Enable hardware averager
    ADC0_SC3 |= ADC_SC3_AVGS(3);             //Set hardware averager to 32 samples
    SIM_SOPT7 |= SIM_SOPT7_ADC0TRGSEL(5);    //Set PIT1 as hardware trigger
    SIM_SOPT7 |= SIM_SOPT7_ADC0ALTTRGEN(1); //Set ADC0 to have alternate trigger

    PIT_TFLG1 |= PIT_TFLG_TIF_MASK;          //Reset PIT1 interrupt flag
    NVIC_ClearPendingIRQ(PIT1_IRQn);         //Clear PIT1 pending interrupts

    ADC0_SC1A = ADC_SC1_ADCH(3);             //Set input to DADP3

    do{
        ADC0_SC3 = ADC_SC3_CAL(1);                  //Begin calibration
        while((ADC0_SC3 & ADC_SC3_CAL(1)) == 1){}   //Wait for calibration
    } while((ADC0_SC3 & ADC_SC3_CALF(1)) == 1);     //Repeat if failed
}

/**************************************************************************
 * TempGet() - Scans temperature
 *
 * Brian Willis, 12/8/2017
 **************************************************************************/
INT32S TempGet(){
    static INT32S Temp;

```

```
53      if((ADC0_SC1A & ADC_SC1_COCO_MASK) != 0){       //Wait until conversion is complete
54          Temp = (INT32S)ADC0_RA;                     //Get result of ADC
55      } else{}
56
57      return Temp;
58  }
59
```

```c
/*****************************************************************************
 * Sensor.c - Handles TSI scanning
 *
 * Brian Willis, 12/8/2017
 *****************************************************************************/
#include "MCUType.h"
#include "Sensor.h"
#include "K65TWR_GPIO.h"

#define TIMESLICE 100           //Kernel timeslice (100us units)
#define TSISCANTASK_PER 10      //Max execution time of task (ms)

INT32U TSITouchLevels[2] = {0, 0};

typedef enum{E1, E2} SCAN_STAGGER;

/*****************************************************************************
* SensorInit() - Initializes electrodes
*
* Brian Willis, 12/8/2017
*****************************************************************************/
void SensorInit(){
    INT16U tsiBaselineLevels[2] = {0, 0};

    SIM_SCGC5 |= SIM_SCGC5_TSI_MASK;                //Start clock for TSI
    PORTB_PCR18 &= PORT_PCR_MUX(0);                 //Disable pin MUX control
    PORTB_PCR19 &= PORT_PCR_MUX(0);
    TSI0_GENCS |= TSI_GENCS_TSIEN_MASK;             //Enable TSI

    TSI0_GENCS |= TSI_GENCS_REFCHRG(5);             //Set reference oscillator charge
    TSI0_GENCS |= TSI_GENCS_EXTCHRG(5);             //Set electrode oscillator charge
    TSI0_GENCS |= TSI_GENCS_DVOLT(1);               //Set oscillator's voltage rails
    TSI0_GENCS |= TSI_GENCS_NSCN(15);               //Set electrode scans per second
    TSI0_GENCS |= TSI_GENCS_PS(5);                  //Set prescaler

    //Calibrate Electrode 1
    TSI0_DATA = TSI_DATA_TSICH(12);                 //Set channels to be scanned
    TSI0_DATA |= TSI_DATA_SWTS(1);                  //Start a scan sequence

    while(!(TSI0_GENCS & TSI_GENCS_EOSF_MASK)){}    //Wait for scan to finish
    TSI0_GENCS |= TSI_GENCS_EOSF(1);                //Clear flag

    tsiBaselineLevels[ELECTRODE1] = (INT16U)(TSI0_DATA & TSI_DATA_TSICNT_MASK);            //Set baseline level
    TSITouchLevels[ELECTRODE1] = (INT32U)tsiBaselineLevels[ELECTRODE1] + E1_TOUCH_OFFSET;  //Set keypress threshold

    //Calibrate Electrode 2
    TSI0_DATA = TSI_DATA_TSICH(11);                 //Set channels to be scanned
    TSI0_DATA |= TSI_DATA_SWTS(1);                  //Start a scan sequence

    while(!(TSI0_GENCS & TSI_GENCS_EOSF_MASK)){}    //Wait for scan to finish
    TSI0_GENCS |= TSI_GENCS_EOSF(1);                //Clear flag

```

```
    tsiBaselineLevels[ELECTRODE2] = (INT16U)(TSI0_DATA & TSI_DATA_TSICNT_MASK);             //Set baseline level
    TSITouchLevels[ELECTRODE2] =  (INT32U)tsiBaselineLevels[ELECTRODE2] + E2_TOUCH_OFFSET;  //Set keypress threshold
}

/*****************************************************************************
* SensorTask() - Scans electrodes
*
* Brian Willis, 12/8/2017
*****************************************************************************/
void SensorTask(){                              //Scan electrodes
    static SCAN_STAGGER ScanStagger = E1;
    static INT8U TSIScanTaskCntr = 0;

    DB3_TURN_ON();

    if(TSIScanTaskCntr == ((10*TSISCANTASK_PER)/TIMESLICE)){  //Call once every period of task
        TSIScanTaskCntr = 0;                                  //Reset counter
        switch(ScanStagger){                                  //Scans staggered to allow enough time inbetween scans
            case(E1):                                         //Read Electrode 2 and scan Electrode 1
                if((INT16U)(TSI0_DATA & TSI_DATA_TSICNT_MASK) > TSITouchLevels[ELECTRODE2]){    //Read
                    tsiSensorFlags |= (1 << ELECTRODE2);
                } else{
                    tsiSensorFlags &= (INT8U)(~(1 << ELECTRODE2));
                }
                TSI0_DATA = TSI_DATA_TSICH(12);
                TSI0_DATA |= TSI_DATA_SWTS(1);                //Scan

                ScanStagger = E2;
                break;
            case(E2):                                         //Read Electrode 1 and scan Electrode 2
                if((INT16U)(TSI0_DATA & TSI_DATA_TSICNT_MASK) > TSITouchLevels[ELECTRODE1]){    //Read
                    tsiSensorFlags |= (1 << ELECTRODE1);
                } else{
                    tsiSensorFlags &= (INT8U)(~(1 << ELECTRODE1));
                }
                TSI0_DATA = TSI_DATA_TSICH(11);
                TSI0_DATA |= TSI_DATA_SWTS(1);                //Scan

                ScanStagger = E1;
                break;
            default:
                break;
        }
    } else{}
    TSIScanTaskCntr++;
    DB3_TURN_OFF();
}

```

```c
/*****************************************************************************
 * AlarmWave.c - Controls alarm speaker output
 *
 * Brian Willis, 12/8/2017
 *****************************************************************************/
#include "MCUType.h"
#include "K65TWR_GPIO.h"
#include "AlarmWave.h"

//Private variables
//Table representing one period of alarm sinewave
//Fundamental frequency of 300Hz
//600Hz, 900Hz, 1200Hz, and 1500Hz harmonics
static const INT16U SineTable[64] = {0x4000, 0x56FF, 0x6AC0, 0x789B, 0x7F00, 0x7DB3, 0x75CC, 0x696E,
                                     0x5B50, 0x4E2B, 0x4433, 0x3EB2, 0x3DD9, 0x40D6, 0x4616, 0x4BB8,
                                     0x5000, 0x51B9, 0x5078, 0x4C9F, 0x4738, 0x41A7, 0x3D43, 0x3B07,
                                     0x3B50, 0x3DCD, 0x419B, 0x4587, 0x485F, 0x4943, 0x47E0, 0x4482,
                                     0x4000, 0x3B7D, 0x381F, 0x36BC, 0x37A0, 0x3A78, 0x3E64, 0x4232,
                                     0x44AF, 0x44F8, 0x42BC, 0x3E58, 0x38C7, 0x3360, 0x2F87, 0x2E46,
                                     0x2FFF, 0x3447, 0x39E9, 0x3F29, 0x4226, 0x414D, 0x3BCC, 0x31D4,
                                     0x24AF, 0x1691, 0xA33, 0x24C, 0xFF, 0x764, 0x153F, 0x2900};

ALARM_STATES AlState = ARMED;               //Initialize alarm as armed

/*****************************************************************************
* AlarmWaveInit() - Initializes PIT0 and DAC0 for 300Hz sinewave output.
*
* Brian Willis, 11/17/2017
*****************************************************************************/
void AlarmWaveInit(void){
    //Initialize DAC0
    SIM_SCGC2 |= SIM_SCGC2_DAC0_MASK;       //Start SCGC2 clock for DAC0
    DAC0_C0 |= DAC_C0_DACSWTRG_MASK;        //Set DAC0 for software trigger
    DAC0_C0 |= DAC_C0_DACRFS_MASK;          //Set DAC0 for VDDR reference
    DAC0_C0 |= DAC_C0_DACEN_MASK;           //Enable DAC0

    //Initialize PIT0
    SIM_SCGC6 |= SIM_SCGC6_PIT_MASK;        //Start SCGC6 clock for PIT0
    PIT_MCR &= ~PIT_MCR_MDIS_MASK;          //Enable PIT via MCR
    PIT_LDVAL0 |= 0xC34U;                   //Set PIT timer to 3124 counts
    PIT_TCTRL0 |= PIT_TCTRL_TEN_MASK;       //Enable timer
    PIT_TCTRL0 |= PIT_TCTRL_TIE_MASK;       //Enable timer interrupt

    PIT_TFLG0 |= PIT_TFLG_TIF_MASK;         //Reset PIT0 interrupt flag
    NVIC_ClearPendingIRQ(PIT0_IRQn);        //Clear PIT0 pending interrupts
    NVIC_EnableIRQ(PIT0_IRQn);              //Enable PIT0 IRQ
}

/*****************************************************************************
 * PIT0_IRQHandler() - Handler that sends sinewave data to DAC
 *
 * Brian Willis, 12/8/2017
```

```
 ****************************************************************************/
void PIT0_IRQHandler(void){                 //timer interrupt handler to control sine table steps
    DB7_TURN_ON();
    PIT_TFLG0 |= PIT_TFLG_TIF_MASK;         //Reset PIT0 interrupt flag at end of countdown

    static volatile INT8U SineCtr = 0;

    if(SineCtr < 63){
        SineCtr++;
    }else{
        SineCtr = 0;
    }

    //With this implementation, alarm does not reset beeping on alarm state change
    if(AlState == ALARM){
        DAC0_DAT0L = (INT8U)((SineTable[SineCtr]>>4) & 0xFF);       //Send SineTable data to DAC0
        DAC0_DAT0H = (INT8U)(SineTable[SineCtr]>>12);
    } else{}
    DB7_TURN_OFF();
}

```

```c
/*****************************************************************************
 * WDOG.c - Controls Watchdog timer
 *
 * Brian Willis, 12/8/2017
 *****************************************************************************/
#include "MCUType.h"
#include "WDOG.h"
#include "K65TWR_GPIO.h"

#define TIMESLICE 100

typedef enum{ON, OFF} TRIGGER;
TRIGGER WatchdogTrigger = OFF;

/*****************************************************************************
 * WDOGInit() - Initalizes Watchdog
 *
 * Brian Willis, 12/8/2017
 *****************************************************************************/
void WDOGInit(){
    INT32U WDOGTimeoutPer;

    WDOG_UNLOCK = 0xC520;                               //Unlock Watchdog via codes
    WDOG_UNLOCK = 0xD928;

    //Set Watchdog to trigger after 1 period slightly longer than timeslice
    WDOGTimeoutPer = ((TIMESLICE*6000000)/1000)+1;

    WDOG_TOVALH = WDOG_TOVALH_TOVALHIGH((INT16U)(WDOGTimeoutPer >> 16));
    WDOG_TOVALL = WDOG_TOVALL_TOVALLOW((WDOGTimeoutPer & 0xFFFF));

    WDOG_STCTRLH |= WDOG_STCTRLH_WDOGEN(1);

    if((RCM_SRS0 & RCM_SRS0_WDOG(1)) != 0){             //If WDOG bit is flipped, Watchdog caused reset
        WatchdogTrigger = ON;
    } else{}
}

/*****************************************************************************
 * WDOGTask() - Resets Watchdog timer every timeslice
 *
 * Brian Willis, 12/8/2017
 *****************************************************************************/
void WDOGTask(){
    DB6_TURN_ON();
    WDOG_REFRESH = WDOG_REFRESH_WDOGREFRESH(0xA602);    //Reset Watchdog timer every timeslice
    WDOG_REFRESH = WDOG_REFRESH_WDOGREFRESH(0xB480);
    DB6_TURN_OFF();
}

```

```c
/*****************************************************************************
* SysTickDelay.c - A delay module based on the system timer
* v1.1
* 10/30/14 Todd Morton
* v2.1 Modify for k65 at 180MHz
* 11/04/2015 Todd Morton
* 11/05/2017 Todd Morton Modify for new header file structure.
* 12/6/2017 Brian Willis Modify for 100us counter and CPU load calculation.
*****************************************************************************
* Project master header file
*****************************************************************************/
#include "MCUType.h"
#include "SysTickDelay.h"
#include "K65TWR_GPIO.h"

/*****************************************************************************
* Private Resources
*****************************************************************************/
static volatile INT32U stmsCount;   /* 100us counter variable */
static INT8U stInitFlag;
static INT32U stLastEvent;

/*****************************************************************************
* Module Defines
*****************************************************************************/
#define CLK_PER_100US 18000U        /* Clock cycles per 100us        */

/*****************************************************************************
* SysTickDelay Function
*    - Public
*    - Delays 'us100' 100's of microseconds
*    - Accuracy +0/-1 100us
*****************************************************************************/
void SysTickDelay(const INT32U us100){
    INT32U start_cnt;
    start_cnt = stmsCount;
    while((stmsCount - start_cnt) < us100){} /* wait for 100us to pass*/
}

/*****************************************************************************
* Period Delay Function
*    - Public - NOT reentrant...in fact only one instance.
*    - To next event every '100us' 100 microseconds
*    - Accuracy +0/-1 100us
*****************************************************************************/
void SysTickWaitEvent(const INT32U period){
    DB0_TURN_ON();
    if(stInitFlag == 1){
        //Load = percentage*10 (28.5% = 285)
        Load = (INT16U)(((stmsCount - stLastEvent)*1000)/period);
        Load = (Load > LoadPrev) ? Load : LoadPrev;
        //TaskTime is peak cumulative task completion time in us
```

```
53          TaskTime = (INT16U)((stmsCount - stLastEvent)*100);
54          TaskTime = (TaskTime > TaskTimePrev) ? TaskTime : TaskTimePrev;
55
56          while((stmsCount - stLastEvent) < period){}
57      }else{
58          stInitFlag = 1;
59      }
60      stLastEvent = stmsCount;
61      LoadPrev = Load;
62      TaskTimePrev = TaskTime;
63      DB0_TURN_OFF();
64  }
65
66  /*****************************************************************************
67  * SysTickDlyInit() - Initialization routine for SysTickDelay()
68  *****************************************************************************/
69  INT32U SysTickDlyInit(void){
70      INT32U sterr;
71      stInitFlag = 0;
72      stmsCount = 0;
73      stLastEvent = 0;
74      sterr = SysTick_Config(CLK_PER_100US);
75      return sterr;
76  }
77
78  /*****************************************************************************
79  * SysTick_Handler() - System Tick Interrupt Handler.
80  *    - setup for a 100us periodic interrupt.
81  *****************************************************************************/
82  void SysTick_Handler(void){
83      stmsCount++;                    /* Increment 100us counter    */
84  }
85  /*****************************************************************************/
86
```

```c
/*
 * BasicIO.c - is a module with public functions used to send and receive
 * information from a serial port. In this case UART2 configured for the
 * Segger debug USB serial port. K65TWR board.
 * v1.1
 *  Created by: Todd Morton, 10/09/2014
 *  With Contributions by: Jacob Gilbert And Adam Slater
 * v2.1
 *  Created by Todd Morton
 *  Contributions to BIOGetStrg() by Chance Eldridge
 * v3.1
 *  Created by Todd Morton
 *  Deprecated includes.h. Replaced with MCUType.h
 *
 * 12/8/2017 Brian Willis Modify for nonblocking terminal input.
 */
/*****************************************************************
* Project master header file
*****************************************************************/
#include "MCUType.h"
#include "BasicIO.h"

/*****************************************************************
* Private Resources
*****************************************************************/
static INT8C bioHtoA(INT8U hnib);   //Convert nibble to ascii
static INT8U bioIsHex(INT8C c);
static INT8U bioHtoB(INT8C c);
/*****************************************************************
* BIOOpen() - Initialization routine for BasicIO()
*    MCU: K65, UART2 configured for debugger USB.
*****************************************************************/
void BIOOpen(void){
    SIM_SCGC5 |= SIM_SCGC5_PORTE(1);     /* Enable clock gate for PORTE */
    SIM_SCGC4 |= SIM_SCGC4_UART2(1);     //enables UART2 clock (60MHz)

    PORTE_PCR16=PORT_PCR_MUX(3);         //ties peripherals to mux address
    PORTE_PCR17=PORT_PCR_MUX(3);
    UART2_BDH = 0x00U;                   //sets clock divisor for 115200Hz Baud Rate
    UART2_BDL = 0x20U;                   //60M / (16*32.5625) = ~115200
    UART2_C2 |= UART_C2_TE_MASK;         //enables transmission
    UART2_C2 |= UART_C2_RE_MASK;         //enables receive
    UART2_C4 = 0x12U;                    //sets the .5625 of divisor
}

/*****************************************************************
* BIORead() - Checks for a character received
*    MCU: K65, UART2
*    return: ASCII character received or 0 if no character received
*****************************************************************/
INT8C BIORead(void){
    INT8C c;
```

```
53      if ((UART2_S1 & UART_S1_RDRF_MASK) != 0){   //check if char received
54          c = UART2_D;
55      }else{
56          c = '\0';                               //If not return 0
57      }
58      return (c);
59  }
60  /*****************************************************************
61  * BIOGetChar() - Blocks until character is received
62  *    return: INT8C ASCII character
63  *****************************************************************/
64  INT8C BIOGetChar(void){
65      INT8C c;
66      c = BIORead();
67      return c;
68  }
69
70  /*****************************************************************
71  * BIOWrite() - Sends an ASCII character
72  *              Blocks as much as one character time
73  *    MCU: K65, UART2
74  *    parameter: c is the ASCII character to be sent
75  *****************************************************************/
76  void BIOWrite(INT8C c){
77      while ((UART2_S1 & UART_S1_TDRE_MASK)==0){} //waits until transmission
78      UART2_D = (INT8U)c;                         //is ready
79  }
80
81  /*****************************************************************
82  * BIOPutStrg() - Writes a string to monitor
83  *    parameter: strg is a pointer to the ASCII string
84  *****************************************************************/
85  void BIOPutStrg(const INT8C *const strg){
86      const INT8C *strgptr = strg;
87      while (*strgptr != '\0'){               //until a null is reached
88          BIOWrite(*strgptr);
89          strgptr++;
90      }
91  }
92
93  /*****************************************************************
94  * BIOOutDecByte() - Outputs the decimal value of a byte.
95  *    Parameters: bin is the byte to be sent,
96  *                lz is true if leading zeros are sent
97  *****************************************************************/
98  void BIOOutDecByte (INT8U bin, INT8U lz){
99      INT8C digits[3];
100     INT8U lbin = bin;
101     INT8U zon = lz;
102     digits[0]=(INT8C)((lbin%10) + '0');
103     lbin = lbin/10;
104     digits[1]=(INT8C)((lbin%10)+'0');
```

```
105         digits[2]=(INT8C)((lbin/10) +'0');
106
107         if((digits[2] != '0') || (zon != 0)){
108             BIOWrite(digits[2]);
109             zon = TRUE;
110         }else{
111         }
112         if((digits[1] != '0') || (zon != 0)){
113             BIOWrite(digits[1]);
114         }else{
115         }
116         BIOWrite(digits[0]);
117     }
118
119     /*****************************************************************
120     * BIOOutDecHWord() - Outputs a decimal value of two bytes.
121     *    Parameters: bin is the half word to be sent,
122     *                lz is true if leading zeros are sent
123     *****************************************************************/
124     void BIOOutDecHWord (INT16U bin, INT8U lz){
125         INT8C digits[5];
126         INT16U lbin = bin;
127         INT8U zon = lz;
128         digits[0]=(INT8C)((lbin%10) +'0');
129         lbin = lbin/10;
130         digits[1]=(INT8C)((lbin%10)+'0');
131         lbin = lbin/10;
132         digits[2]=(INT8C)((lbin%10)+'0');
133         lbin = lbin/10;
134         digits[3]=(INT8C)((lbin%10)+'0');
135         digits[4]=(INT8C)((lbin/10) +'0');
136
137         if((digits[4] != '0') || (zon != 0)){
138             BIOWrite(digits[4]);
139             zon = TRUE;
140         }else{
141         }
142         if((digits[3] != '0') || (zon != 0)){
143             BIOWrite(digits[3]);
144             zon = TRUE;
145         }else{
146         }
147         if((digits[2] != '0') || (zon != 0)){
148             BIOWrite(digits[2]);
149             zon = TRUE;
150         }else{
151         }
152         if((digits[1] != '0') || (zon != 0)){
153             BIOWrite(digits[1]);
154         }else{
155         }
156         BIOWrite(digits[0]);
```

```
157     }
158
159 /*****************************************************************
160 * BIOGetStrg() - Inputs a string and stores it into an array.
161 *
162 * Descritpion: A routine that inputs a character string to an array
163 *              until a carraige return is received or strglen is exceeded.
164 *              Only printable characters are recognized except carriage
165 *              return and backspace.
166 *              Backspace erases displayed character and array character.
167 *              A NULL is always placed at the end of the string.
168 *              All printable characters are echoed.
169 * Return value: 0 -> if ended with CR
170 *               1 -> if strglen exceeded.
171 * Arguments: *strg is a pointer to the string array
172 *            strglen is the max string length, includes CR/NULL.
173 *****************************************************************/
174 INT8U BIOGetStrg(INT8U strglen, INT8C *const strg){
175     INT8C c;
176     INT8C *strgp = strg;
177     INT8U rvalue = 0;
178     static INT8U strg_char = 0;
179     static INT8U charnum = 0;
180
181     c = BIOGetChar();                                   //Non-blocking
182     if((' ' <= c) && ('~' >= c)){
183        BIOWrite(c);
184        strgp[strg_char] = c;
185        strg_char++;
186        charnum++;
187     } else if(c == '\b'){                               //Backspace
188         BIOWrite('\b');
189         BIOWrite(' ');
190         BIOWrite('\b');
191         strg_char--;
192         charnum--;
193     } else{}
194
195     if((charnum > (strglen-1)) || (c == '\r')){     //Signal if string is completed
196         strg_char = 0;
197         charnum = 0;
198         BIOOutCRLF();
199         rvalue = 1;
200     } else{}
201
202     return rvalue;
203 }
204 /*****************************************************************
205 * BIOOutCRLF() - Outputs a carriage return and line feed.
206 *
207 *****************************************************************/
208 void BIOOutCRLF(void){
```

```
209         BIOPutStrg("\r\n");
210     }
211
212     /*****************************************************************
213     * BIOHexStrgtoWord() - Converts a string of hex characters to a 32-bit
214     *                      word until NULL is reached.
215     * Return value: 0 -> if no error.
216     *               1 -> if string is too long for word.
217     *               2 -> if a non-hex character is in the string.
218     *               3 -> No characters in string. Started with NULL.
219     * Arguments: *strg is a pointer to the string array
220     *            *bin is the word that will hold the converted string.
221     *****************************************************************/
222     INT8U BIOHexStrgtoWord(INT8C *const strg,INT32U *bin){
223         INT8U cnt = 0;
224         INT32U lbin = 0;
225         INT8C *strgptr = strg;
226         INT8U rval = 0;
227         if(*strgptr == '\0'){
228             rval = 3;
229         }else{
230             while(*strgptr != '\0'){
231                 if(bioIsHex(*strgptr) != 0){
232                     lbin = (lbin << 4) | (INT32U)(bioHtoB(*strgptr));
233                 }else{
234                     rval = 2;
235                 }
236                 strgptr++;
237                 cnt++;
238                 if(cnt > 8){
239                     rval = 1;
240                 }else{
241                 }
242             }
243             *bin = lbin;
244         }
245         return rval;
246     }
247
248     /*****************************************************************
249     * BIOOutHexByte() - Output one byte in hex.
250     * bin is the byte to be sent
251     *****************************************************************/
252     void BIOOutHexByte(INT8U bin){
253         BIOWrite(bioHtoA(bin>>4));
254         BIOWrite(bioHtoA(bin & 0x0fu));
255     }
256
257     /*****************************************************************
258     * BIOOutHexHWord() - Output 16-bit word in hex.
259     * bin is the word to be sent
260     *****************************************************************/
```

```
261 void BIOOutHexHWord(INT16U bin){
262     BIOOutHexByte((INT8U)(bin>>8));
263     BIOOutHexByte((INT8U)(bin & 0x00ffu));
264 }
265 /*****************************************************************
266 * BIOOutHexWord() - Output 32-bit word in hex.
267 * bin is the word to be sent
268 * Todd Morton, 10/14/2014
269 *****************************************************************/
270 void BIOOutHexWord(INT32U bin){
271     BIOOutHexByte((INT8U)(bin>>24));
272     BIOOutHexByte((INT8U)(bin>>16));
273     BIOOutHexByte((INT8U)(bin>>8));
274     BIOOutHexByte((INT8U)(bin & 0x000000ff));
275 }
276 /*****************************************************************
277 * bioIsHex() - Checks for hex ascii character - private
278 * returns 1 if hex and 0 if not hex.
279 * Todd Morton, 10/14/2014
280 *****************************************************************/
281 static INT8U bioIsHex(INT8C c){
282     INT8U rval;
283     if(('0' <= c) && ('9' >= c)) || (('a' <= c) && ('f' >= c)) || (('A' <= c) && ('F' >= c))){
284         rval = 1;
285     }else{
286         rval = 0;
287     }
288     return rval;
289 }
290 
291 /*****************************************************************
292 * bioHtoB() - Converts a hex ascii character to a binary byte - private
293 * c is the ascii character to be converted.
294 * returns the binary value.
295 * Note: it returns a 0 if it is not a hex character - this should be fixed.
296 * Todd Morton, 10/14/2014
297 *****************************************************************/
298 static INT8U bioHtoB(INT8C c){
299     INT8U bin;
300     if(('0' <= c) && ('9' >= c)){
301         bin = (INT8U)(c - '0');
302     }else if(('a' <= c) && ('f' >= c)){
303         bin = (INT8U)(c - 'a' + 0xa);
304     }else if(('A' <= c) && ('F' >= c)){
305         bin = (INT8U)(c - 'A' + 0xa);
306     }else{
307         bin = 0;
308     }
309     return bin;
310 }
311 /*****************************************************************
312 * bioHtoA() - Converts a hex nibble to ASCII - private
```

```
313   * hnib is the byte with the LSN to be sent
314   * Todd Morton, 10/14/2014
315   ************************************************************************/
316   static INT8C bioHtoA(INT8U hnib){
317       INT8C asciic;
318       INT8U hnmask = hnib & 0x0fu; /* Take care of any upper nibbles */
319       if(hnmask <= 9U){
320           asciic = (INT8C)(hnmask + 0x30U);
321       }else{
322           asciic = (INT8C)(hnmask + 0x37U);
323       }
324       return asciic;
325   }
326
```

```c
/*****************************************************************************
 * TermInterface.c - Controls updating terminal with CPU load information
 *
 * Brian Willis, 12/8/2017
 *****************************************************************************/
#include "MCUType.h"
#include "TermInterface.h"
#include "K65TWR_GPIO.h"
#include "SysTickDelay.h"
#include "BasicIO.h"

#define TIMESLICE 100                                                   //Kernel timeslice in ms
#define TERMINTERFACETASK_PER 30                                        //Max execution time of tasks in ms

typedef enum{ON, OFF} TRIGGER;
typedef enum{S1, S2, S3, S4, S5, S6, S7, S8, S9, S10, S11} TERMDISPSTATE;

/*****************************************************************************
 * TermInterfaceTask() - Scans and updates terminal
 *
 * Brian Willis, 12/8/2017
 *****************************************************************************/
void TermInterfaceTask(){
    INT8U strg_ready;
    static TRIGGER TermDisplay = OFF;
    static TERMDISPSTATE TermDispState = S1;
    static INT8C InputStrg[6] = {'\0'};
    static INT8U TermInterfaceTaskCntr = 0;

    DB5_TURN_ON();

    if(TermInterfaceTaskCntr == ((10*TERMINTERFACETASK_PER)/TIMESLICE)){    //Call once every period of task
        TermInterfaceTaskCntr = 0;                                          //Reset counter

        strg_ready = BIOGetStrg(6, InputStrg);

        if((strg_ready)
            && (InputStrg[0] == 'l')
            && (InputStrg[1] == 'o')
            && (InputStrg[2] == 'a')
            && (InputStrg[3] == 'd')){
            TermDisplay = ON;
        } else if((strg_ready)
            && (InputStrg[0] == 'r')
            && (InputStrg[1] == 'e')
            && (InputStrg[2] == 's')
            && (InputStrg[3] == 'e')
            && (InputStrg[4] == 't')){
            LoadPrev = 0;
            TaskTimePrev = 0;
            for(INT8U i = 0; i < 6; i++){
                InputStrg[i] = '\0';
```

```
53              }
54          } else{}
55
56          if(TermDisplay == ON){
57              switch(TermDispState){
58                  case(S1):
59                      BIOWrite('\r');
60                      BIOPutStrg("CP");
61                      TermDispState = S2;
62                      break;
63                  case(S2):
64                      BIOPutStrg("U: ");
65                      TermDispState = S3;
66                      break;
67                  case(S3):
68                      BIOOutDecHWord((Load/10), 0);                          //Display tens and ones place of load
69                      TermDispState = S4;
70                      break;
71                  case(S4):
72                      BIOWrite('.');
73                      BIOOutDecHWord((INT16U)(Load-((Load/10)*10)), 0);      //Display tenths place of load
74                      TermDispState = S5;
75                      break;
76                  case(S5):
77                      BIOPutStrg("% |");
78                      TermDispState = S6;
79                      break;
80                  case(S6):
81                      BIOPutStrg(" Ta");
82                      TermDispState = S7;
83                      break;
84                  case(S7):
85                      BIOPutStrg("sk: ");
86                      TermDispState = S8;
87                      break;
88                  case(S8):
89                      BIOOutDecHWord(TaskTime, 0);
90                      TermDispState = S9;
91                      break;
92                  case(S9):
93                      BIOPutStrg("us");
94                      TermDispState = S10;
95                      break;
96                  case(S10):
97                      BIOPutStrg("    ");
98                      TermDispState = S11;
99                      break;
100                 case(S11):
101                     BIOPutStrg("  ");
102                     TermDispState = S1;
103                     TermDisplay = OFF;
104                     for(INT8U i = 0; i < 6; i++){
```

```
                        InputStrg[i] = '\0';
                    }
                    BIOOutCRLF();
                    break;
                default:
                    break;
            }
        } else{}
    }
    TermInterfaceTaskCntr++;
    DB5_TURN_OFF();
}

```