```
/***********************************
*main.c- The main code that includes
*checksum and key scanning.

*Mark Moerdyk 1/23/13
Last Modification:
1/31/13
*************************
*Project Master Header File
*************************/
#include "includes.h"

#define START_ADDR (INT8U *)0xC000
#define END_ADDR (INT8U *)0xFFFF
#define SLICE_PER 10
#define DC1 (INT8U)0x11
#define DC2 (INT8U)0x12
#define DC3 (INT8U)0x13
#define DC4 (INT8U)0x14
#define DB_PORT PTP
#define DB_DDR DDRP
#define PP7 128
#define PP6 64
#define PP5 32
#define DB_OUTS 224




/*************************************
Global variable and function definitions
*************************************/
void WaitForSlice(void);
void DisplayCheckSum(INT16U TotalCheckSum);
void LCDDemoTask(void);
INT16U CalcChkSum( INT8U *startaddr, INT8U *endaddr);
INT8U *startaddr;
INT8U *endaddr;

/***********************************/
void main(void)
{
    INT8U Resultprint[] = "CS: ";
    INT16U CalcChkSumResult;
    DB_PORT = 0x00;      //Initialize Debug bits
    DB_DDR = DB_OUTS;

    OCDlyInit();
    ENABLE_INT();         /* Enable interrupts to use OCDelay()   */
    KeyInit();
    LcdInit();
    LcdClrDisp();
    LcdMoveCursor(2,1);

    startaddr = START_ADDR;
    endaddr = END_ADDR;

    CalcChkSumResult = CalcChkSum( startaddr, endaddr);

    LcdDispStrg(Resultprint); //Prints array based string

   DisplayCheckSum(CalcChkSumResult);

   FOREVER()
   {
        WaitForSlice();
        LCDDemoTask();
        KeyTask();
   }




}

/***********************************
*CalcChkSum Function - A function that takes
the contents of the start and end address and adds the
contents from the start address to the end address

Passes in: startaddr, endaddr
Returns: TotalSum

************************************/
INT16U CalcChkSum( INT8U *startaddr, INT8U *endaddr)
{
    INT16U TotalSum = 0x0000;

    while(startaddr < endaddr)//includes endaddress content
    {
        TotalSum = (INT16U)*startaddr + TotalSum; // adds the 16bit content to totalsum
        startaddr++;

    }
    TotalSum = (INT16U)*endaddr + TotalSum;
    return TotalSum;


}
/***************************
*DisplayCheckSum - Function that takes the 16 bit
sum and displays it into two 8 bit bytes on the LCD

*Modules: LCD
*Member: LcdDispByte()
***************************/
void DisplayCheckSum(INT16U TotalCheckSum)
{
    INT8U HighBit;
    INT8U LowBit;
    INT8U *pointer;

    HighBit = ((INT8U)(TotalCheckSum >> 8)); //high 8 bits of the 16 bit interger
    pointer = &HighBit;
    LcdDispByte(pointer);
    LowBit =(INT8U) TotalCheckSum; //default to the low 8 bit of the 16 bits
    pointer = &LowBit;
    LcdDispByte(pointer);



}
/*****************************************
  WaitForSlice() - Time slicer. Uses OCDelay module for a time slice
*                  period of SLICE_PER.
```

```
* Modules: OCDelay
* Member: GetmSCnt()
*********************************/
void WaitForSlice(void){

    static INT16U LastTime;
    static INT8U TSInit = TRUE;

    DB_PORT |= PP7;

    if(TSInit){   /* Initialize LastTime first time through */
        LastTime = GetmSCnt();
        TSInit = FALSE;
    }else{        /* wait for next time slice */
        while((GetmSCnt() - LastTime) < SLICE_PER){}
        LastTime += SLICE_PER;           /* set up for next time slice */
    }
    DB_PORT &= ~PP7;
}
/***************************
*LCDDemoTask - Goes through all the LCD tasks and
waits inbetween tasks so that the user can visually see each task

*Modules: LCD
*Member: LcdClrDisp(), LcdDispDecByte(), LcdMoveCursor(), LcdDispTime()
LcdCursor(), LcdBSpace(), LcdFSpace(),LcdDispChar()
**************************/
void LCDDemoTask(void)
{
    typedef enum { LCDDISPLAYSTATE0, LCDDISPLAYSTATE1,LCDDISPLAYSTATE2, LCDDISPLAYSTATE3,
                   LCDDISPLAYSTATE4 ,LCDDISPLAYSTATE5, LCDDISPLAYSTATE6, LCDDISPLAYSTATE7, L
                   LCDDISPLAYSTATE9, LCDWAITPERIOD1, LCDWAITPERIOD2,LCDWAITPERIOD3, LCDWAITP
                   LCDWAITPERIOD5, LCDWAITPERIOD6, LCDWAITPERIOD7, LCDWAITPERIOD8} LCDSTATES
    INT8U key;
    INT8U DispByteNum = 0x09;
    INT8U DelayNumber = 150;
    static LCDSTATES CurState = LCDDISPLAYSTATE0;
    static INT8U DelayCount = 1;
    static INT8U NumDelayCount = 3;
    static INT8U ButtonStartDemo = 0;

    DB_PORT |= PP6;

    key = GetKey();

    if (key == DC3 && ButtonStartDemo == 0 )//Cbutton is pressed
    {
        LcdClrDisp();
        LcdMoveCursor(1,1);
        CurState = LCDDISPLAYSTATE1 ;
        ButtonStartDemo = 1; // makes it so C can't restart loop, only B

    }
    else if (key == DC2 && ButtonStartDemo == 1)//B button is pressed after C button starts
    {
        LcdClrDisp();
        LcdMoveCursor(1,1);
        CurState = LCDDISPLAYSTATE1;//resets everything to original state

    }
    else
    {
    }//nothing

    switch (CurState)
    {
    case LCDDISPLAYSTATE0:
        break;

    case LCDDISPLAYSTATE1: //prints char 'B' and changes cursor
       LcdDispChar('B');
       LcdCursor(FALSE,FALSE);
       CurState = LCDWAITPERIOD1;
       break;

    case LCDWAITPERIOD1:    //waits until count = number; results in pause on lcd
        if(DelayCount < DelayNumber)
        {
            DelayCount++;
        }
        else
        {
            DelayCount = 1;
            CurState = LCDDISPLAYSTATE2;
        }
        break;

    case LCDDISPLAYSTATE2: //prints clock layout and changes cursor
        LcdDispTime ( 4,45,8);
        LcdCursor(FALSE, TRUE);
        CurState = LCDWAITPERIOD2;
        break;

    case LCDWAITPERIOD2:  //waits until count = number; results in pause on lcd
        if(DelayCount < DelayNumber)
        {
            DelayCount++;
        }
        else
        {
            DelayCount = 1;
            CurState = LCDDISPLAYSTATE3;
        }
        break;

    case LCDDISPLAYSTATE3: //clears line 1 and moves cursor to start of second line
        LcdClrLine(1);
        LcdMoveCursor(2,1);
        CurState = LCDWAITPERIOD3;
        break;

    case LCDWAITPERIOD3:    //waits until count = number; results in pause on lcd
        if(DelayCount < DelayNumber)
        {
            DelayCount++;
        }
        else
        {
            DelayCount = 1;
            CurState = LCDDISPLAYSTATE4;
        }
        break;

     case LCDDISPLAYSTATE4: //Displays Dec byte with zeros, and changes cursor
       LcdDispDecByte(&DispByteNum,TRUE);
       LcdCursor(TRUE, FALSE);
```

```
        CurState = LCDWAITPERIOD4;

        break;
     case LCDWAITPERIOD4:   //waits until count = number; results in pause on lcd
        if(DelayCount < DelayNumber)
        {
            DelayCount++;
        }
        else
        {
            DelayCount = 1;
            CurState = LCDDISPLAYSTATE5;
        }
        break;

     case LCDDISPLAYSTATE5: //clears second row, shifts cursor right one space
        LcdClrLine(2);
        LcdFSpace();
        CurState = LCDWAITPERIOD5;
        break;

    case LCDWAITPERIOD5:    //waits until count = number; results in pause on lcd
        if(DelayCount < DelayNumber)
        {
            DelayCount++;
        }
        else
        {
            DelayCount = 1;
            CurState = LCDDISPLAYSTATE6;
        }
        break;

    case LCDDISPLAYSTATE6: //prints dec byte without zeros
        LcdDispDecByte(&DispByteNum, FALSE);
        CurState = LCDWAITPERIOD6;
        break;

    case LCDWAITPERIOD6:   //waits until count = number; results in pause on lcd
        if(DelayCount < DelayNumber)
        {
            DelayCount++;
        }
        else
        {
            DelayCount = 1;
            CurState = LCDDISPLAYSTATE7;
        }
        break;

    case LCDDISPLAYSTATE7: //changes cursor and shifts cursor left one
        LcdCursor(TRUE, TRUE);
        LcdBSpace();
        CurState = LCDWAITPERIOD7;
        break;

    case LCDWAITPERIOD7:  //waits until count = number; results in pause on lcd
        if(DelayCount < DelayNumber)
        {
            DelayCount++;
        }
        else
        {
            DelayCount = 1;
            CurState = LCDDISPLAYSTATE8;
        }
        break;

    case LCDDISPLAYSTATE8:  //shifts cursor back 1left and prints 'F'

        LcdBSpace();
        LcdDispChar('F');
        CurState = LCDWAITPERIOD8;
        break;

    case LCDWAITPERIOD8:   //waits until count = number; results in pause on lcd
        if(DelayCount < DelayNumber)
        {
            DelayCount++;
        }
        else
        {
            DelayCount = 1;
            CurState = LCDDISPLAYSTATE9;
        }
        break;

    case LCDDISPLAYSTATE9: //clears display so it can repeat again

        LcdClrDisp();
        CurState = LCDDISPLAYSTATE1;

        break;

    default:
        break;

    }
   DB_PORT &= ~PP6;
}
```