```
//**********************************************************************************
*********************

**
// University of Illinois at Chicago, Dept. of Electrical and Computer Engineering
// Programmer: Mitchell Hedditch
// UIN: 677318273
// ECE 367 - Microprocessor-Based Design
// Semester: Spring 2013
// Lab Session: Tuesday 8AM-10:50AM
// Final Exam Project: Microwave Oven Panel Control
// Purpose of Code:  This code is for a standard microwave oven control system
panel.  It has the

following
//        features:
//             1) Countdown Timer set control for minutes and seconds.
//             2) Start/Continue key.  Press to Start.  Press again after Pause to
Continue.
//             3) Pause/Clear key.  Press once for pause.  Press again for
Stop/Clear if if the
//                  Start/Continue has not been pressed.
//             4) LCD Display to show the current timer value (show countdown in
seconds),
//                  PAUSE if in puase state, and done when the timer runs out.  The
system
//                  then return to the initial state by pressing the Pause/Clear
key.
//             5) Power setting for microwave
//             6) EasyCook for quick 30 second cooking at max power
//             7) Quick Cook for different food types and fast cooking
//             8) Temperature Set cook
//             9) Real time clock with fast speed capability

// Date Created: 4/25/2013
// Due Date: 5/10/2013
// Updated: 5/1/2013
// Version: 1
//===============================================
// Keypad Control Functions
//   XIRQ': START/CONTINUE
//   IRQ' : PAUSE/CLEAR
//   A    : SET TIMER (TOGGLE CLOCK SPEED)
//   B    : SET POWER
//   C    : EASY COOK KEY
//   D    : MENU EASY COOK
//   E    : SET TEMPERATURE
//   F    : SET CLOCK
//===============================================
// NANOCore Pin Usage:
//   1 :  USB2NCT 1
//   2 :  USB2NCT 2
//   3 :  USB2NCT 3
//   4 :  USB2NCT 4
//   5 :  A/D CONVERTER FOR THERMISTOR
//   6 :  NOT CONNECTED
//   7 :  NOT CONNECTED
//   8 :  NOT CONNECTED
//   9 :  NOT CONNECTED
//   10:  NOT CONNECTED
//   11:  NOT CONNECTED
//   12:  MICROWAVE ELEMENT LED
//   13:  SERIAL OUT SCK TO SIPO SCK
//   14:  SERIAL OUT MOSI TO SIPO SERIAL DATAIN
//   15:  SERIAL OUT SS TO SIPO RCK
//   16:  NOT CONNECTED
//   17:  PM1 TO LCD ENABLE
//   18:  PM0 TO LCD RS (COMMAND OR CHARACTER)
//   19:  PT0 TO MATRIX KEYPAD P0 (READ COLUMN 1)
//   20:  PT1 TO MATRIX KEYPAD P1 (READ COLUMN 2)
//   21:  PT2 TO MATRIX KEYPAD P2 (READ COLUMN 3)
//   22:  PT3 TO MATRIX KEYPAD P3 (READ COLUMN 4)
//   23:  PT4 TO MATRIX KEYPAD P4 (SET ROW 1 HIGH)
//   24:  PT5 TO MATRIX KEYPAD P5 (SET ROW 2 HIGH)
//   25:  PT6 TO MATRIX KEYPAD P6 (SET ROW 3 HIGH)
//   26:  PT7 TO MATRIX KEYPAD P7 (SET ROW 4 HIGH)
//   27:  IRQ' PAUSE/CLEAR
//   28:  XIRQ' START/CONTINUE
//   29:  VCC POWER
//   30:  RESET
//   31:  GROUND
//   32:  NOT CONNECTED
//**********************************************************************************
*********************

**
/* Some include (header) files needed by Codewarrior with machine info for the
NanoCore12

*/

#include <hidef.h>                           //common defines and macros
#include "derivative.h"                      //derivative-specific definitions

/* Here we give function prototypes before we start on the main code */
extern void near tch2ISR(void);              //tch2ISR() prototype
extern void near tch1ISR(void);              //tch1ISR() prototype
extern void near tch0ISR(void);              //tch0ISR() prototype
extern void near irqISR(void);               //pause_clear prototype
extern void near xirqISR(void);              //start_continue prototype
extern void near UnimplementedISR(void);     //UnimplementedISR prototype

#pragma CODE_SEG __NEAR_SEG NON_BANKED       //REQUIRED PRAGMA

interrupt void  UnimplementedISR(void)       //UNIMPLEMENTED INTERRUPT SUB
{
      for(;;);    // do nothing. simply return
}

#pragma CODE_SEG DEFAULT                     //REQUIRED PRAGMA


typedef void (*near tIsrFunc) (void);        //REQUIRED PRAGMA


const tIsrFunc _vect[] @0xFFEA = {           // VECTOR ARRAY SETUP FOR INTERRUPTS

     tch2ISR,                                // 0xFFEA TIMER CH2
     tch1ISR,                                // 0xFFEC TIMER CH1
     tch0ISR,                                // 0xFFEE TIMER CH0
     UnimplementedISR,                       // 0xFFF0 REAL TIME INTERRUPT
     irqISR,                                 // 0xFFF2 IRQ
     xirqISR,                                // 0xFFF4 XIRQ
};
```

```
/* We need to define some constants. Similar to EQU's in assembly


*/
#define     IOREGS_BASE  0x0000

#define     _IO8(off)    *(unsigned char  volatile *)(IOREGS_BASE + off) //define
form prototype 8-bit
#define     _IO16(off)   *(unsigned short volatile *)(IOREGS_BASE + off) //define
form prototype 16-bit


//#define   PORTT     _IO8(0x240)
/* portT data register is unsigned 8-bit at address $0240


*/
/* because of the form prototype defines above this is the same as


*/
/* #define PORTT *(unsigned char  volatile *) (0x240);    Means PORTT points to
address $0240

*/
/* the statement PORTT = 0x34; means to store $34 at location $0240


*/
/* if the contents of PORTT is 0xd3 then the assignment x = PORTT; means x is now
equal to 0xd3

*/
/***********************************************************************************
*********************

*/
/* The commented out defines already exist in one of the above header files. The
compiler

*/
/* does not like the redundancy. So, they are commented out with the // symbols


*/
//#define     TSCR1   _IO8(0x46)             // timer system control register
//#define     PTT     _IO8(0x240)            // portt data register
//#define     DDRT    _IO8(0x242)            // portt direction register
//#define     CRGFLG  _IO8(0x37)             // pll flags register
//#define     SYNR    _IO8(0x34)             // synthesizer / multiplier register
//#define     REFDV   _IO8(0x35)             // reference divider register
//#define     CLKSEL  _IO8(0x39)             // clock select register
//#define     PLLCTL  _IO8(0x3a)             // pll control register
#define   PORTT    _IO8(0x240)               // PortT data register
#define   PORTTi   _IO8(0x241)               // portT data register
#define   PORTM    _IO8(0x250)               // portM data register
#define   MCCTL    _IO8(0x66)                // modulus down conunter control
#define   MCFLG    _IO8(0x67)                // down counter flags
//******************************************************
//NOTE: TIMER INTERRUPTS INCLUDED IN HEADER FILE ALREADY
//******************************************************
#define   SPCR1    _IO8(0xD8)                //SPI SPCR1 REGISTER LOCATION
#define   SPCR2    _IO8(0xD9)                //SPI SPCR2 REGISTER LOCATION
```

```
#define   SPIB     _IO8(0xDA)               //SPI SPIB REGISTER LOCATION
#define   MCCNT    _IO16(0x76)              //modulus down counter register
#define   keypad   PORTT                    //SET KEYPAD TO PORTT VALUE
// DEFINE BIT FLAG AND CONFIG VALUES HERE
#define   PLLSEL  0x80                      //PLL SELECT REGISTER
#define   LOCK    0x08                      //PLL LOCK REGISTER
#define   TFFCA   0x10                      //PLL REGISTER
#define   MCZF    0x80                      //PLL REGISTER
#define   BIT0    0x01                      //BIT0 VALUE FOR TESTING/SETTING
PURPOSES
#define   BIT1    0x02                      //BIT1 VALUE FOR TESTING/SETTING
PURPOSES
#define   BIT2    0x04                      //BIT2 VALUE FOR TESTING/SETTING
PURPOSES
#define   BIT3    0x08                      //BIT3 VALUE FOR TESTING/SETTING
PURPOSES
#define   BIT4    0x10                      //BIT4 VALUE FOR TESTING/SETTING
PURPOSES
#define   BIT5    0x20                      //BIT5 VALUE FOR TESTING/SETTING
PURPOSES
#define   BIT6    0x40                      //BIT6 VALUE FOR TESTING/SETTING
PURPOSES
#define   BIT7    0x80                      //BIT7 VALUE FOR TESTING/SETTING
PURPOSES
#define   ENABLE  0x02                      //LCD ENABLE USED FOR PM1
#define   RCK     0x08                      //FOR RCK CONNECTED TO PM3
#define   RS      0x01                      //REGISTER SELECT (RS) AT PM0
(0=COMMAND,1=DATA)


//SYSTEM VARIABLES
//****************
volatile unsigned char SYSTEM_MODE;              // SYSTEM MODE INDICATOR VARIABLE
volatile unsigned char PAUSE;                    // SYSTEM PAUSE MODE INDICATOR (0:
RUNNING, 1:PAUSED)
volatile unsigned char BLINK;                    // BLINK FLAG FOR DISPLAY
volatile unsigned char COMPLETE;                 // COUNTDOWN COMPLETE TIMER

//MICROWAVE TIMER
//***************
volatile unsigned char TIMER_FLAG;               // FLAG FOR TIMER INTERRUPT
volatile unsigned char TIMER_SET;                // SET MODE FOR TIMER
volatile unsigned int  TIMER_INTERRUPT_COUNT;    // INTERRUPT COUNTER FOR TIMER
volatile unsigned int  TIMER_VALUE;              // TIMER COUNT IN SECONDS
//REAL TIME CLOCK
//***************
volatile unsigned char CLOCK_FLAG;               // FLAG FOR CLOCK INTERRUPT
volatile unsigned char CLOCK_SET;                // SET MODE FOR CLOCK
volatile unsigned char CLOCK_SPEED;              // SET SPEED FOR CLOCK
volatile unsigned int  CLOCK_INTERRUPT_COUNT;    // INTERRUPT COUNTER FOR CLOCK
volatile unsigned long CLOCK_TIME;               // CLOCK COUNT IN SECONDS
//POWER VARS
//***************
volatile unsigned char POWER_FLAG;               // FLAG FOR POWER INTERRUPT
volatile unsigned char POWER_SET;                // SET MODE FOR POWER
volatile unsigned int  POWER_INTERRUPT_COUNT;    // INTERRUPT COUNTER FOR POWER LED
volatile unsigned char POWER_VAL;                // POWER VALUE (1-10)
volatile unsigned char POWER_COUNT;              // POWER COUNT TIME
//TEMP VARS
//***************
volatile unsigned char SET_TEMP;                 //VALUE OF USER SET TEMPERATURE
volatile unsigned char CUR_TEMP;                 //VALUE OF THERMISTOR TEMPERATURE
volatile unsigned char TEMP_MODE;                //IS SYSTEM IN TEMP MODE?
volatile unsigned int AD_VAL;                     //A/D TEMPERATURE VAL

//*************************************
//Hardware driver prototype subroutines
char getkey(void);                           // PROTOTYPE FOR MATRIX KEYPDAD
void SetClk8(void);                          // PLL PROTOTYPE
void delayby1ms(int k);                      // 1 MILLISECOND DELAY PROTYPE
void keyrelease(void);                       // WAIT FOR KEY RELEASE PROTOTYPE
void Command(char a);                        // SEND COMMAND TO LCD PROTOTYPE
void Print(char a);                          // PRINT ASCII CHAR TO LCD PROTOTYPE
void Clear(void);                            // CLEAR LCD PROTOTYPE
void delay(void);                            // SHORT SYSTEM DELAY PROTOTYPE
void delay3(void);                           // MEDIUM DELAY PROTOTYPE
void systemInitialize(void);                 // INITIALIZE SYSTEM PROTOTYPE
void SPIInitialize(void);                    // INITIALIZE SPI PROTOTYPE
void InitLCD(void);                          // LCD INITIALIZATION PROTOTYPE
void initInterrupts(void);                   // SET UP INTERRUPTS PROTOTYPE
void startInterrupts(void);                  // START INTERRUPTS PROTOTYPE
//*************************************
//Software prototype subroutines
void intro(void);                            // INTRO PROTOTYPE
void changeSystemMode(char mode);            // CHANGE SYSTEM MODE PROTOTYPE
void updateDisplay(void);                    // UPDATE LCD DISPLAY PROTOTYPE
void printString(char *string);              // PRINT A STRING PROTOTYPE
void printParseDecimalChar(char val);        // PARSE A CHAR INTO ASCII DECIMAL AND
PRINT PROTOTYPE
void printSpaces(char spaces);               // PRINT SPACES PROTOTYPE
void cookingComplete(void);                  // COOK COMPLETE PROTOTYPE
//**************************************
//Countdown Timer Subroutines Prototypes
void setTimer(char key);                     // SET COOK TIME PROTOTYPE
void printTimer(void);                       // PRINT TIMER VALUE PROTOTYPE
//**************************************
//Real Time Clock Subroutines Prototypes
void setClock(void);                         // SET THE CLOCK PROTOTYPE
void printClockTime(void);                   // PRINT CLOCK TIME TO LCD
//**************************************
//Power Setting Subroutine Prototypes
void setPower();                             // SET THE POWER PROTOTYPE
void printPower(void);                       // PRINT POWER TO LCD PROTOTYPE
void displayPowerLED(void);                  // LED DISPLAY PROTOTYPE
//**************************************
//Cook Key Subroutines Prototypes
void quickSet(char power, long timer);       // SET SYSTEM VARS (QUICKSET) PROTOTYPE
void easyCook(void);                         // EASY COOK PROTOTYPE
void quickCook(void);                        // SPECIAL COOK MENU PROTOTYPE
//**************************************
//Temperature Subroutine Prototypes
void setTemp(void);                          // SET TEMPERATURE PROTOTYPE
void getTemp(void);                          // GET TEMP PROTOTYPE
void printTemp(void);                        // PRINT TEMPERATURE PROTOTYPE




/******************************************************************/
/*  This is the main code where everything should go,             */
/*  All program function is directed by this code                */
/******************************************************************/
void main(void)
{
     char key1;                              //DEFINE VARIABLE FOR KEY INPUT
```

```
        systemInitialize();                     //RUN SYSTEM INITIALIZATION

     updateDisplay();                        //PRINT DISPLAY FOR THE FIRST TIME

     intro();                                //DISPLAY SYSTEM INTRODUCTION

     while(1)                                //INFINITE LOOP TO ALWAYS RUN
     {
          /* OK. 1ms gone by. Let's see if user pressed the A key or the B key. */
                key1  = getkey();                  //CHECK FOR KEYPRESS
                if(key1 < 0x1f)                    //IF WE HAVE A KEYPRESS
                {
                     keyrelease();                 //WAIT FOR KEY RELEASE
                     delay();                      //RUN A SHORT DELAY
                        while((PORTTi & 0x08));       //IF COLUMN 3 IS HIGH WAIT
HERE UNTIL LOW
               changeSystemMode(key1);       //CHANGE SYSTEM MODE BY USER INPUT
               if(SYSTEM_MODE == 99) {
                    setTimer(key1);          //SEE IF THE USER WANTS TO SET THE
TIMER
               }
          }

          if(COMPLETE == 0x01)               //IS COOK COMPLETE??
          {
               TIMER_VALUE = 0;              //RESET TIMER VALUE
               PAUSE = 0x01;                 //PAUSE THE SYSTEM
               TEMP_MODE = 0;                //GET OUT OF TEMP MODE
               PTAD = PTAD & ~BIT7;          //TURN ELEMENT (LED) OFF
               cookingComplete();            //INDICATE COMPLETE TO USER
               COMPLETE = 0;                 //CLEAR COMPLETE FLAG
          }

             //IF TIMER IS COUNTING DOWN...
             if(TIMER_FLAG == 0x01 && PAUSE == 0x00) {
                  updateDisplay();              //UPDATE THE DISPLAY
                  TIMER_FLAG = 0x00;            //RESET THE TIMER FLAG
             }

             //IF TEMPERATURE MODE IS RUNNING...
             if(TIMER_FLAG == 0x01 && TEMP_MODE == 1 && PAUSE == 0x00) {
                  updateDisplay();              //UPDATE THE DISPLAY
                  TIMER_FLAG = 0x00;            //CLEAR THE TIMER FLAG
             }

             if(PAUSE == 0x00)                  //IS SYSTEM RUNNING?
             {
                  displayPowerLED();            //RUN ELEMENT LED
             }

             //IF HOME SCREEN AND CLOCK FLAG GOES UP...
             if(CLOCK_FLAG == 0x01 && PAUSE == 0x01 && SYSTEM_MODE == 99 &&
TIMER_VALUE == 0)
             {
                  updateDisplay();              //UPDATE THE DISPLAY
                  CLOCK_FLAG = 0x00;            //CLEAR CLOCK FLAG
             }

             //CHECK FOR SYSTEM MODE
             if(SYSTEM_MODE == 0)
             {
                  setTimer(0);                  //SET TIMER MODE
             }
             else if(SYSTEM_MODE == 1)
             {
                  setPower();                   //SET POWER MODE
             }
             else if(SYSTEM_MODE == 2)
             {
                  easyCook();                   //EASY COOK MODE
             }
             else if(SYSTEM_MODE == 3)
             {
                  quickCook();                  //QUICK COOK MODE
             }
             else if(SYSTEM_MODE == 4)
             {
                  setTemp();                    //SET TEMPERATURE MODE
             }
             else if(SYSTEM_MODE == 5)
             {
                  setClock();                   //SET THE CLOCK MODE
             }

             //WAS "A" PRESSED TWICE?
             if(SYSTEM_MODE == 6)
             {
                  if(CLOCK_SPEED == 0x00)       //IF SPEED IS SLOW
                  {
                       CLOCK_SPEED = 0x01;      //SET SPEED FAST
                  } else {
                       CLOCK_SPEED = 0x00;      //SET SPEED SLOW
                  }
             }

             SYSTEM_MODE = 99;                  //CLEAR SYSTEM MODE
     }
}



/********************************************************************/
/*   PRINT START-UP AND INSTRUCTIONS TO LCD SCREEN                  */
/********************************************************************/
void intro(void)
{
     Clear();                                     //CLEAR THE SCREEN
     Command(0x02);                               //MOVE CURSOR TO HOME POSITION
     printString("    HEDDITCH    ");             //PRINT "HEDDITCH" TO LCD
        Command(0xC0);                                            //MOVE CURSOR TO 
NEW LINE
        printString(" MICROWAVE V1.0 ");             //PRINT " MICROWAVE V1.0 " TO 
LCD
        delayby1ms(3000);                            //DELAY 3 SECONDS

        Clear();                                     //CLEAR THE SCREEN
     Command(0x02);                               //MOVE CURSOR TO HOME POSITION
     printString("SEE INSTRUCTIONS");             //PRINT "SEE INSTRUCTIONS" TO LCD
        Command(0xC0);                                            //MOVE CURSOR TO 
NEW LINE
        printString(" FOR OPERATION  ");             //PRINT "  FOR OPERATION " TO 
LCD
        delayby1ms(3000);                            //DELAY 3 SECONDS
}
```

```
/********************************************************************/
/*  This command changes system modes for the microwave             */
/********************************************************************/
void changeSystemMode(char key)
{
     if(key >= 10)                           //DID USER PRESS KEY GREATER THAN 9?
        {
          if(key==10)                        //IF USER PRESSED A...
                {
                     SYSTEM_MODE = 0;              //SYSTEM MODE IS TIMER ENTRY (OR
TOGGLE SPEED)
                }
          else if(key==11)                   //IF USER PRESSED B...
                {
                     SYSTEM_MODE = 1;              //SYSTEM MODE IS POWER ENTRY
                }
          else if(key == 12)                 //IF USER PRESSED C...
          {
               SYSTEM_MODE = 2;              //RUN EASY COOK PROGRAM
                }
             else if(key == 13)                 //IF USER PRESSED D...
          {
               SYSTEM_MODE = 3;              //RUN QUICK COOK
                }
             else if(key == 14)                 //IF USER PRESSED E...
          {
               SYSTEM_MODE = 4;              //SYSTEM MODE IS TEMP ENTRY
                }
             else if(key == 15)                 //IF USER PRESSED F....
          {
               SYSTEM_MODE = 5;              //SYSTEM MODE IS SET CLOCK
                }

             updateDisplay();          // UPDATE THE DISPLAY
     }

     //NOTE: AFTER MODE EXECUTION SYSTEM MODE RETURNS TO 99 BY DEFAULT
}
/********************************************************************/
/*  This function updates the display by printing characters and    */
/*  Commands to it                                                  */
/********************************************************************/
void updateDisplay(void)
{
     char i;                                      //DECLARE I AS CHAR

     Clear();                                     //CLEAR DISPLAY

     Command(0x02);                               //MOVE CURSOR TO HOME POSITION

        if(PAUSE == 0x01 && TEMP_MODE == 0 && TIMER_VALUE == 0)     //IF PAUSED &
TIMER=0
        {
             printString("    HEDDITCH    ");   //PRINT "HEDDITCH" TO LCD
             Command(0xC0);                                  //MOVE CURSOR TO NEW
LINE
             printClockTime();                  //PRINT CLOCK TIME
        }
        else if (TEMP_MODE == 1)                //IF TEMP MODE
        {
          if(PAUSE == 0x01) {
               printString("     PAUSED     ");   //PRINT "PAUSED" TO LCD
          } else {
               printPower();                      //PRINT POWER ON LINE 1
          }
          Command(0xC0);                                        //MOVE CURSOR TO NEW 
LINE
             printTemp();                       //DISPLAY CURRENT TEMPERATURE
        }
        else
        {
          if(PAUSE == 0x01) {
               printString("     PAUSED     ");   //PRINT "PAUSED" TO LCD
          } else {
               printPower();                      //PRINT POWER ON LINE 1
          }
          Command(0xC0);                                        //MOVE CURSOR TO NEW 
LINE
             printTimer();                      //DISPLAY TIMER
        }
}
/************************************************************************************
*****/
/* printStringPrint determines the number of characters in the string so that we can 
   */
/* send the correct number of characters to the LCD print command.                    
   */
/* Then, the characters are printed. There are built in functions in the string.h     
   */
/* library but we are not using that library. So, we will use this home made 
function.  */
/* Not pretty but it works.                                                           
   */
/************************************************************************************
*****/
void printString(char *string)
{
    int i, n;                      //DEFINE VARS
    const char *tmp = string;      //GET FIRST CHAR POINTER

    // NOTE: does NOT check for string == NULL

    while(*tmp != '\0')            //LOOP UNTIL END OF LINE
    {                              // C strings end with \0
        tmp++;                     //INCREMENT CHAR COUNT
    }
    n = tmp - string;              // OK. Now we know how many characters to print

    for(i=0; i<n; i++)             //COUNT THROUGH ALL CHARS
    {
        Print(string[i]);          // Call LCD print command
    }
}
/********************************************************************/
/*  THIS COMMAND TAKES A VALU (MAX 99) AND PARSES IT TO DECIMAL     */
/*  TO HAVE IT PRINTED                                              */
/********************************************************************/
void printParseDecimalChar(char val)
{
    Print((val/10)+0x30);          //PRINT TENS DIGIT

    Print((val % 10)+0x30);        //PRINT ONES DIGIT
}
/********************************************************************/
/*   This subroutine sets the amount of time to run the microwave   */
/*   element for.  It runs continuously                             */
/********************************************************************/
void setTimer(char key)
{
     unsigned char e;                                        //DECLARE VAR e
     volatile unsigned int tens, ones, tenths, hundreths;   //DECLARE VARS
     long time_val;                                          //TEMP VAR FOR
TIMER_VALUE
     time_val = TIMER_VALUE;                                 //GRAB THE CURRENT TIMER
VALUE
     Clear();                                                //CLEAR THE SCREEN
     e = 0;                                                  //SET e TO 0
     SYSTEM_MODE = 0;                                        //PUT US IN TIMER MODE

     tens = (time_val / 600);                                //CALCULATE TENS VALUE
     ones = ((time_val /60) % 10);                           //CALCULATE ONES VALUE
     tenths = ((time_val % 60) / 10);                        //CALCULATE TENTHS VALUE
     hundreths = (time_val % 60) % 10;                       //CALCULATE HUNDRETHS

        while(PAUSE == 0x01 && SYSTEM_MODE == 0)               //LOOP UNTIL USER
PRESSES A AGAIN
     {
          if(e > 0)                                          //IF e GREATER THAN 0
          {
               key  = getkey();                              //GO CHECK FOR KEYPRESS
                keyrelease();                                 //WAIT FOR USER TO
RELEASE KEY
             delay();                                     //DELAY
                while((PORTTi & 0x08));                       //IF COLUMN 3 IS HIGH
WAIT HERE UNTIL LOW
          }

          if(key == 0x0a)                                    //IF A PRESSED AGAIN
          {
               SYSTEM_MODE = 6;                              //CHANGE TO SPEED MODE
          }

                if(key < 0x0a)                                    //DID WE GET A
KEY? THEN PERFORM ACTION
                {
                     Clear();                                     //CLEAR THE SCREEN
             Command(0x02);                               //MOVE CURSOR TO HOME
POSITION
               printString("SET TIMER");                     //PRINT "SET TIMER"
                Command(0xC0);                                            //MOVE
CURSOR TO NEW LINE
                printSpaces(11);                              //PRINT 11 SPACES

               Print(ones + 0x30);                           //PRINT NEW TENS
CHARACTER
                     Print(tenths + 0x30);                        //PRINT NEW ONES
CHARACTER
                     Print(0x3A);                                 //PRINT COLON
               Print(hundreths + 0x30);                      //PRINT NEW TENTHS
CHARACTER
                     Print(key + 0x30);                           //PRINT NEW
HUNDRETHS CHARACTER

                     tens = ones;                                 //SET TENS TO OLD
ONES
               ones = tenths;                                //SET ONES TO OLD TENTHS
               tenths = hundreths;                           //SET TENTHS TO OLD
HUNDRETHS
```

```
               hundreths = key;                             //SET HUNDRETHS TO KEY

               //CALCULATE NEW TIMER VALUE
               TIMER_VALUE = (tens*600) + (ones*60) + (tenths*10) + (hundreths);

          }

             e++;                                              //INCREMENT E
        }

     updateDisplay();                                       //UPDATE OUR DISPLAY
AGAIN
}
/********************************************************************/
/*  THIS SUB DISPLAYS THE NUMBER THAT THE TIMER IS CURRENTLY AT.    */
/********************************************************************/
void printTimer(void)
{
     unsigned char minute, j;                          //DECLARE j, minute VAR

     printSpaces(5);                                   //PRINT 5 SPACES
     minute = TIMER_VALUE / 60;                        //CALCULATE MINUTES
     if(minute < 10)                                   //IF MINUTES IS LESS THAN 10
     {
          Print(0x20);                                 //PRINT A SPACE " "
          Print(minute+0x30);                          //PRINT CURRENT MINUTE
     }
     else
     {
          printParseDecimalChar(minute);               //PRINT MINUTE (GREATER THAN
9)
     }
     Print(0x3A);                                      //PRINT COLON
     printParseDecimalChar(TIMER_VALUE % 60);          //PRINT ONES DIGIT
}
/********************************************************************/
/*             THIS SUB SETS THE TIME CLOCK                         */
/********************************************************************/
void setClock(void)
{
     volatile unsigned char meridian, key, disp;            //DEFINE meridian var
     volatile unsigned char tens, ones, tenths, hundreths;  //DEFINE VARS
     long clock_val;                                        //TEMP VAR FOR
CLOCK_TIME

     clock_val = CLOCK_TIME;                                //GRAB THE CURRENT CLOCK
VALUE
     meridian = 0x00;                                       //MERIDIAN IS AM
     disp = 1;                                              //DISPLAY TIME FLAG SET

     if(clock_val > 43199)                                  //IF CLOCK IS GREATER
THAN NOON....PM
     {
          meridian = 0x01;                                  //THEN IT'S POST
MERIDIAN
          clock_val = clock_val - 43200;                    //SUBTRACT THE FIRST
TWELVE HOURS
     }

     tens = (clock_val / 36000);                            //CALCULATE TENS VALUE
     ones = ((clock_val / 3600) % 10);                      //CALCULATE ONES VALUE
     tenths = ((clock_val % 3600) / 600);                   //CALCULATE TENTHS VALUE
     hundreths = ((clock_val % 3600) / 60) % 10;            //CALCULATE HUNDRETHS
```

```
VALUE

     while(SYSTEM_MODE == 5)                                    //LOOP UNTIL CHANGES
MODE OUT OF 5
     {
          key = getkey();                                       //GO CHECK FOR KEYPRESS
                keyrelease();                                         //WAIT FOR USER TO
RELEASE KEY
             delay();                                              //DELAY
                while((PORTTi & 0x08));                               //IF COLUMN 3 IS
HIGH WAIT HERE UNTIL

LOW

                if(key < 0x0a)                                        //DID WE GET A
KEY? THE PERFORM ACTION
                {
                  tens = ones;                                     //SET TENS TO OLD
ONES
               ones = tenths;                                   //SET ONES TO OLD TENTHS
               tenths = hundreths;                              //SET TENTHS TO OLD
HUNDRETHS
               hundreths = key;                                 //SET HUNDRETHS TO KEY

               disp = 1;                                        //UPDATE DISPLAY FLAG
SET
          }
          else if (key == 0x0e)                                 //IF "E" KEY IS PRESSED
          {
               if(meridian == 0x00)                             //IF MERIDIAN IS AM
               {
                    meridian = 0x01;                            //SET TO PM
               }
               else meridian = 0x00;                            //SET TO AM

               disp = 1;                                        //UPDATE DISPLAY FLAG
SET
          }
          else if (key == 0x0f)                                 //IF KEY IS "F"
          {
               SYSTEM_MODE = 99;                                //CHANGE MODE OUT TO
HOME SCREEN
          }

             if(disp == 1)                                         //IF DISPLAY FLAG SET

             {
                  Clear();                                         //CLEAR THE SCREEN
             Command(0x02);                                   //MOVE CURSOR TO HOME
POSITION
               printString("SET CLOCK");                        //PRINT "SET CLOCK"
                Command(0xC0);                                                //MOVE
CURSOR TO NEW LINE
                printSpaces(9);                                   //PRINT 9 SPACES

               Print(tens + 0x30);                              //PRINT TENS CHARACTER
             Print(ones + 0x30);                              //PRINT ONES CHARACTER
             Print(0x3A);                                     //PRINT COLON
               Print(tenths + 0x30);                            //PRINT TENTHS CHARACTER
             Print(hundreths + 0x30);                         //PRINT HUNDRETHS
CHARACTER
             if(meridian == 0x00) {                           //IS IT AM?
                    Print(0x41);                                //PRINT A
                    Print(0x4d);                        //PRINT M
               } else {
                    Print(0x50);                        //PRINT P
                    Print(0x4d);                        //PRINT M
               }
        }
             disp = 0;                                     //RESET DISPLAY FLAG
     }

     if(meridian == 0x00 && tens == 1 && ones == 2)
     {
          tens = 0;
          ones = 0;
     }

     if(tens > 1 || ones > 2 || tenths > 5 || hundreths > 9)
     {
          Clear();                                 //CLEAR THE SCREEN
        Command(0x02);                           //MOVE CURSOR TO HOME POSITION
          printString("INVALID SETTING");          //PRINT "SET CLOCK"
          delayby1ms(2000);
//delay 2 seconds
     } else {
          //CALCULATE NEW CLOCK_TIME
          CLOCK_TIME = (tens*36000) + (ones*3600) + (tenths*600) + (hundreths*60);

          //IF MERIDIAN IS PM UPDATE CLOCK_TIME
          if(meridian == 0x01 && CLOCK_TIME < 43200) CLOCK_TIME += 43200;

     }

     updateDisplay();                                   //UPDATE OUR DISPLAY
AGAIN
     CLOCK_INTERRUPT_COUNT = 0;                         //RESET THE COUNT
}
/*********************************************************************/
/*        DISPLAY THE CURRENT CLOCK TIME.                            */
/*********************************************************************/
void printClockTime(void)
{
     char hour, meridian;                               //DECLARE hour, meridian
VAR
     int m;                                             //DECLARE i VAR

     printSpaces(4);
//PRINT 4 SPACES TO LCD


     hour = (CLOCK_TIME/3600);                          //CALCULATE CURRENT HOUR

     if(hour < 12) {                                    //IF IT'S LESS THAN 12
          meridian = 0x00;                              //THEN IT'S AM
     } else {
          meridian = 0x01;                              //ELSE IT'S PM
     }

     if(hour == 0) {                                    //IF HOUR IS ZERO
          hour = 12;                                    //THEN HOUR IS TWELVE
     } else if (hour > 12) {                            //IF HOUR IS GREATER
THAN TWELVE
          hour = hour - 12;                             //THEN SUBTRACT TWELVE
FROM IT
     }
     if(hour < 10)                                          //IF HOUR IS LESS THAN
10
     {
          Print(0x20);                                      //PRINT A SPACE
          Print(hour+0x30);                                 //PRINT HOUR
     }
     else
     {
          printParseDecimalChar(hour);                      //ELSE JUST PRINT THE
HOUR
     }

     if(BLINK == 0x01) {                                    //CHECK TO SEE IF COLON
SHOULD BLINK?
          Print(0x3A);                                      //IF IT'S OFF, TURN IT
ON
          BLINK = 0x00;                                     //SET BLINK TO ZERO
     }
     else
     {
          Print(0x20);                                      //IF IT'S ON, TURN IT
OFF
          BLINK = 0x01;                                     //SET BLINK TO ONE
     }

     printParseDecimalChar((CLOCK_TIME % 3600)/60);         //PRINT MINUTES


     printSpaces(1);                                        //PRINT A SPACE

     if(meridian == 0x00) {                                 //IS IT AM?
          Print(0x41);                                      //PRINT A
          Print(0x4d);                                      //PRINT M
     } else {
          Print(0x50);                                      //PRINT P
          Print(0x4d);                                      //PRINT M
     }
}
/*********************************************************************/
/*        THIS SUBROUTINE SETS THE POWER FOR THE MICROWAVE           */
/*********************************************************************/
void setPower(void)
{
     unsigned char key;                          //DEFINE KEY VAR
     key = 0x1f;                                 //INITIALIZE KEY VAR

     Clear();                                    //CLEAR THE SCREEN
     Command(0x02);                              //MOVE CURSOR TO HOME POSITION
     printString("SET POWER");                   //PRINT "SET POWER"
        Command(0xC0);                                     //MOVE CURSOR TO NEW LINE
        printSpaces(15);                            //PRINT 15 SPACES

     while(key > 0x09)                           //LOOP UNTIL USER PRESSES NUMERIC VALUE
     {
          key  = getkey();                       //GO CHECK FOR KEYPRESS
                keyrelease();                          //WAIT FOR USER TO RELEASE KEY
             delay();                               //DELAY
                while((PORTTi & 0x08));                //IF COLUMN 3 IS HIGH WAIT HERE
UNTIL LOW
     }

     POWER_VAL = key;                            //SET THE POWER VALUE TO KEY PRESSED
     Print(key+0x30);                        //PRINT KEY VALUE
     delayby1ms(1000);                                  //DELAY FOR 1 SECOND;
}
/*********************************************************************/
/*        PRINT THE POWER VALUE TO THE LCD                           */
/*********************************************************************/
void printPower(void)
{
     printString("POWER          ");         //PRINT "POWER"
     Print(POWER_VAL+0x30);                   //PRINT POWER VALUE
}
/*********************************************************************/
/*   SETS POWER & TIME FOR QUICKSETS IN MICROWAVE IN POWER AND TIME */
/*   IN TOTAL SECONDS.                                               */
/*********************************************************************/
void quickSet(char power, long timer)
{
     POWER_VAL = power;                       //SET THE POWER

     TIMER_VALUE = timer;                     //SET TIME IN SECONDS

     TC1 = TCNT + 15000;                      //MAKE SURE TO INCREMENT THE CLOCK
TIMER
     PAUSE = 0x00;                            //CHANGE SYSTEM TO RUN MODE



     updateDisplay();                         //UPDATE THE DISPLAY
}
/*********************************************************************/
/*        SETS THE EASYCOOK VALUES AND STARTS THE MICROWAVEe         */
/*********************************************************************/
void easyCook(void)
{
     quickSet(9, 30);                         //CALL QUICKSET FOR 30SEC PWR 9

}
/*********************************************************************/
/*   DISPLAYS A MENU FOR QUICK COOK AND THEN WAITS FOR USER TO       */
/*   MAKE A CHOICE AND STARTS THE QUICK COOK PROGRAM                 */
/*********************************************************************/
void quickCook(void)
{
     char key;                                //DECLARE key VAR
     key = 0x1f;                              //INITIALIZE key
     Clear();                                 //CLEAR THE SCREEN
     Command(0x02);                           //MOVE CURSOR TO HOME POSITION
     printString("1: POPCORN");               //PRINT "1: POPCORN" TO LCD
        Command(0xC0);                                       //MOVE CURSOR TO NEW
LINE
        printString("TIME: 1:50 PWR 5");        //PRINT "TIME: 1:50 PWR 5" TO LCD
        delayby1ms(3000);                       //3 SECOND DELAY

        Clear();                                //CLEAR THE SCREEN
     Command(0x02);                           //MOVE CURSOR TO HOME POSITION
     printString("2: FROZEN PIZZA");          //PRINT "2: FROZEN PIZZA" TO LCD
        Command(0xC0);                                       //MOVE CURSOR TO NEW
LINE
        printString("TIME: 1:40 PWR 2");        //PRINT "TIME: 1:40 PWR 2" TO LCD
        delayby1ms(3000);                       //3 SECOND DELAY

        Clear();                                //CLEAR THE SCREEN
     Command(0x02);                          //MOVE CURSOR TO HOME POSITION
     printString("3: POTATOES");             //PRINT "3: POTATOES" TO LCD
        Command(0xC0);                                       //MOVE CURSOR TO NEW
LINE
        printString("TIME: 5:00 PWR 9");        //PRINT "TIME: 5:00 PWR 9" TO LCD
        delayby1ms(3000);                       //3 SECOND DELAY

        Clear();                                //CLEAR THE SCREEN
     Command(0x02);                          //MOVE CURSOR TO HOME POSITION
     printString("CHOICE:");                 //PRINT "CHOICE: " TO LCD
        Command(0xC0);                                 //MOVE CURSOR TO NEW LINE
        printSpaces(15);                        //PRINT 15 SPACES

     while(key > 0x03)                       //LOOP UNTIL USER PRESSES NUMERIC VALUE
     {
          key  = getkey();                   //GO CHECK FOR KEYPRESS
                keyrelease();                      //WAIT FOR USER TO RELEASE KEY
             delay();                          //DELAY
                while((PORTTi & 0x08));            //IF COLUMN 3 IS HIGH WAIT HERE
UNTIL LOW
     }

     Print(key+0x30);                        //PRINT KEY PRESSED
     delayby1ms(1000);                                //DELAY FOR 1 SECOND;

     if(key == 1) {                          //IF KEY IS 1
          quickSet(5, 110);                  //RUN POPCORN QUICKCOOK
     } else if (key == 2) {                  //IF KEY IS 2
          quickSet(2, 100);                  //RUN FROZEN PIZZA QUICKCOOK
     } else if (key == 3) {                  //IF KEY IS 3
          quickSet(9, 300);                  //RUN POTATOES QUICKCOOK
     }
}
/**********************************************************************/
/* CONTROL THE POWER LED, TURN ON AND OFF BASED ON POWER SETTING    */
/**********************************************************************/
void displayPowerLED(void)
{
     char pwr;                                    //DEFINE pwr VAR
     pwr = (POWER_VAL+1);                         //ADD 1 TO POWER

     if(POWER_FLAG == 1)                          //IF POWER FLAG IS 1
     {
          POWER_COUNT++;                          //INCREMENT POWER_COUNT
          if(POWER_COUNT > 10) POWER_COUNT = 1;   //SET POWER TO 1 IF POWER > 10

          if(POWER_COUNT <= pwr) {                //IF CURRENT POWER <= POWER SET
               PTAD = PTAD | BIT7;                //TURN LED ON
          } else {
               PTAD = PTAD & ~BIT7;               //TURN LED OFF
          }

          POWER_FLAG = 0;                         //RESET POWER FLAG
     }
}
/**********************************************************************/
/*      THIS SETS THE USERS DESIRED TEMPERATURE                      */
/**********************************************************************/
void setTemp(void)
{
     volatile unsigned int tens, ones;       //DEFINE VARS
     volatile unsigned char key;             //DEFINE key VAR
     tens = (SET_TEMP / 10);                 //CALCULATE TENS VALUE
     ones = (SET_TEMP % 10);                 //CALCULATE ONES VALUE
     Clear();                                //CLEAR THE SCREEN
     Command(0x02);                          //MOVE CURSOR TO HOME POSITION
     printString("ENTER TEMP");              //PRINT "ENTER TEMP" ON LCD
        Command(0xC0);                                       //MOVE CURSOR TO NEW
LINE
        printSpaces(13);                        //PRINT 13 SPACES
     Print(tens + 0x30);                     //PRINT TENS DIGIT
     Print(ones + 0x30);                     //PRINT ONES DIGIT
     printString("C");                                  //PRINT LETTER C TO LCD

     while(SYSTEM_MODE == 4)                 //LOOP UNTIL USER PRESSES A AGAIN
     {
          key  = getkey();                   //GO CHECK FOR KEYPRESS
                keyrelease();                      //WAIT FOR USER TO RELEASE KEY
             delay();                           //DELAY
                while((PORTTi & 0x08));            //IF COLUMN 3 IS HIGH WAIT HERE
UNTIL LOW

                if(key < 0x0a)                     //DID WE GET A KEY? THE PERFORM
ACTION
                {
                     tens = ones;                  //SHIFT ONES TO TENS
               ones = key;                   //MAKE ONES USER PRESSED VALUE
               Clear();                      //CLEAR THE SCREEN
             Command(0x02);                //MOVE CURSOR TO HOME POSITION
               printString("ENTER TEMP");    //PRINT "ENTER TEMP" ON LCD
                Command(0xC0);                          //MOVE CURSOR TO NEW LINE
                printSpaces(13);              //PRINT 13 SPACES
               Print(tens + 0x30);           //PRINT TENS DIGIT
             Print(ones + 0x30);           //PRINT ONES DIGIT
             printString("C");                  //PRINT LETTER C TO LCD
                  TEMP_MODE = 1;                //SET TEMP MODE
             COMPLETE = 0;                 //CLEAR COMPLETE FLAG
             PAUSE = 0x00;                 //START/RUN
             SET_TEMP = (tens*10) + ones;  //SET THE TEMP
          }
     }

     updateDisplay();                        //UPDATE OUR DISPLAY AGAIN
}
/********************************************************************/
/*        GETS THE TEMPERATURE FROM THE A/D PORT THERMISTOR         */
/********************************************************************/
void getTemp(void)
{
     ATDCTL2 = 0x80;                                   //TURN ON ATD POWER.  NO FAST
FLAGS
     delay();                                          //WAIT FOR POWER TO COME ON
     ATDCTL3 = 0x20;                                   //DO 4 CONVERSIONS
     ATDCTL4 = 0x05;                                   //10-BIT AT 2MHz
     ATDCTL5 = 0x80;                                   //START CONVESION:
RIGHT-JUSTIFIED,
                                                       //UNSIGNED, SCAN OF CHANNEL 0

     while ((ATDSTAT0 & 0x80) == 0x00)                 //POLL SCF FLAG
                                                       //SEQUENCE COMPLETE FLAG
     AD_VAL = (ATDDR0 + ATDDR1 + ATDDR2 + ATDDR3)/4;   //AVERAGE TEMP READINGS

     ATDCTL2 = 0x00;                                   //TURN OFF ATD POWER
```

```
}
/********************************************************************/
/*        PRINTS THE CURRENT TEMPERATURE AS READ TO THE LCD         */
/********************************************************************/
void printTemp(void)
{
     getTemp();                         //GET THE CURRENT TEMP
     printString("TEMP:");              //PRINT "TEMP" TO LCD
     printSpaces(8);                    //PRINT 9 SPACES

     if(AD_VAL < 230) {                 //ARE WE AT OR LESS THAN 0C?
          CUR_TEMP = 0;                 //SET THE CUR_TEMP VAR
     } else if (AD_VAL < 336) {         //ARE WE AT OR LESS THAN 10C?
          CUR_TEMP = 10;                //SET THE CUR_TEMP VAR
     } else if (AD_VAL < 453) {         //ARE WE AT OR LESS THAN 20C?
          CUR_TEMP = 20;                //SET THE CUR_TEMP VAR
     } else if (AD_VAL < 465) {         //ARE WE AT OR LESS THAN 21C?
          CUR_TEMP = 21;                //SET THE CUR_TEMP VAR
     } else if (AD_VAL < 477) {         //ARE WE AT OR LESS THAN 22C?
          CUR_TEMP = 22;                //SET THE CUR_TEMP VAR
     } else if (AD_VAL < 489) {         //ARE WE AT OR LESS THAN 23C?
          CUR_TEMP = 23;                //SET THE CUR_TEMP VAR
     } else if (AD_VAL < 500) {         //ARE WE AT OR LESS THAN 24C?
          CUR_TEMP = 24;                //SET THE CUR_TEMP VAR
     } else if (AD_VAL < 512) {         //ARE WE AT OR LESS THAN 25C?
          CUR_TEMP = 25;                //SET THE CUR_TEMP VAR
     } else if (AD_VAL < 524) {         //ARE WE AT OR LESS THAN 26C?
          CUR_TEMP = 26;                //SET THE CUR_TEMP VAR
     } else if (AD_VAL < 535) {         //ARE WE AT OR LESS THAN 27C?
          CUR_TEMP = 27;                //SET THE CUR_TEMP VAR
     } else if (AD_VAL < 546) {         //ARE WE AT OR LESS THAN 28C?
          CUR_TEMP = 28;                //SET THE CUR_TEMP VAR
     } else if (AD_VAL < 558) {         //ARE WE AT OR LESS THAN 29C?
          CUR_TEMP = 29;                //SET THE CUR_TEMP VAR
     } else if (AD_VAL < 569) {         //ARE WE AT OR LESS THAN 30C?
          CUR_TEMP = 30;                //SET THE CUR_TEMP VAR
     } else if (AD_VAL < 672) {         //ARE WE AT OR LESS THAN 40C?
          CUR_TEMP = 40;                //SET THE CUR_TEMP VAR
     } else if (AD_VAL < 758) {         //ARE WE AT OR LESS THAN 50C?
          CUR_TEMP = 50;                //SET THE CUR_TEMP VAR
     } else if (AD_VAL < 825) {         //ARE WE AT OR LESS THAN 60C?
          CUR_TEMP = 60;                //SET THE CUR_TEMP VAR
     } else if (AD_VAL < 876) {         //ARE WE AT OR LESS THAN 70C?
          CUR_TEMP = 70;                //SET THE CUR_TEMP VAR
     } else if (AD_VAL < 913) {         //ARE WE AT OR LESS THAN 80C?
          CUR_TEMP = 80;                //SET THE CUR_TEMP VAR
     } else if (AD_VAL < 941) {         //ARE WE AT OR LESS THAN 90C?
          CUR_TEMP = 90;                //SET THE CUR_TEMP VAR
     } else if (AD_VAL < 961) {         //ARE WE AT OR LESS THAN 0C?
          CUR_TEMP = 212;               //SET THE CUR_TEMP VAR
     }

     printParseDecimalChar(CUR_TEMP);   //PRINT THE CURRENT TEMP
     printString("C");                  //PRINT LETTER "C"
}
/********************************************************************/
/*        PRINTS THE CURRENT TEMPERATURE AS READ TO THE LCD         */
/********************************************************************/
void cookingComplete(void)
{
     char t;                                      //DECLARE t VAR
     t = 0;                                       //INITIALIZE t
     for(t; t<3;t++)                              //LOOP 3 TIMES
     {
          Clear();                                //CLEAR THE SCREEN
          Command(0x02);                          //MOVE CURSOR TO HOME POSITION
          printString("    COOKING     ");        //PRINT "ENTER TEMP" ON LCD
             Command(0xC0);                                       //MOVE CURSOR
TO NEW LINE
             printString("    COMPLETE    ");        //PRINT "ENTER TEMP" ON LCD
             delayby1ms(1000);                       //DELAY FOR 1 SECOND
          Command(0x08);                          //TURN THE LCD Off
          delayby1ms(1000);                       //DELAY FOR 1 SECOND
          Command(0x0C);                          //TURN THE LCD ON
     }

     COMPLETE = 0;                                //RESET COMPLETE FLAG
}



/*###################################################################################
#####################

#*/
/*###################################################################################
#####################

#*/
/*                          BEGIN HARDWARE DRIVERS SOFTWARE INTERFACE


*/
/*###################################################################################
#####################

#*/
/*###################################################################################
#####################

#*/




/*********************************************************************/
/*  This is the initialization controller, it initializes program   */
/*  vars & calls other systems to initialize in the hardware        */
/*********************************************************************/
void systemInitialize(void)
{
     SetClk8();                    // go setup the PLL

     // setup the data direction registers
     DDRM = 0xfc;                  // set data direction register for PortM
     DDRT = 0xf0;                  // set data direction register for PortT
     PORTM = 0;                    // clear PortM
     DDRAD = DDRAD | BIT7;         // SET DATA DIRECTION REGISTER FOR PORT AD
     PTAD = PTAD | 0x00;           // CLEAR PORT AD
     SPIInitialize();              // INITIALIZE THE SPI SYSTEM

     InitLCD();                    // INITIALIZE THE LCD

     EnableInterrupts;             // SAME AS asm(cli)
     initInterrupts();             // INITIALIZE THE INTERNAL TIMER INTERRUPTS
     startInterrupts();            // START THE INTERRUPTS

     //SYSTEM VARIABLES
     //****************
     SYSTEM_MODE = 99;             // SYSTEM MODE INDICATOR VARIABLE
     PAUSE = 0x01;                 // SYSTEM PAUSE MODE INDICATOR (0: RUNNING,
1:PAUSED)
     BLINK = 0x01;                 // BLINK FLAG FOR DISPLAY
     COMPLETE = 0x00;              // INITIALIZE COMPLETE TO ZERO

     //MICROWAVE TIMER
     //****************
     TIMER_FLAG = 0x00;            // FLAG FOR TIMER INTERRUPT
     TIMER_SET = 0x00;             // SET MODE FOR TIMER
     TIMER_INTERRUPT_COUNT = 0;    // INTERRUPT COUNTER FOR TIMER
     TIMER_VALUE = 0;              // TIMER COUNT IN SECONDS

     //REAL TIME CLOCK
     //****************
     CLOCK_FLAG = 0x00;            // FLAG FOR CLOCK INTERRUPT
     CLOCK_SET = 0x00;             // SET MODE FOR CLOCK
     CLOCK_SPEED = 0x00;           // INITIALIZE IN NORMAL SPEED
     CLOCK_INTERRUPT_COUNT = 0;    // INTERRUPT COUNTER FOR CLOCK
     CLOCK_TIME = 43200;           // CLOCK COUNT IN SECONDS

     //POWER VARS
     //****************
     POWER_FLAG = 0x00;            // FLAG FOR POWER INTERRUPT
     POWER_SET = 0x00;             // SET MODE FOR POWER
     POWER_INTERRUPT_COUNT = 0;    // INTERRUPT COUNTER FOR POWER LED
     POWER_VAL = 5;                // POWER VALUE (1-10)
     POWER_COUNT = 1;              // POWER ON TIME INITIALIZE

     //TEMP VARS
     //****************
     SET_TEMP = 0;                 // DEFAULT TO 0 DEGREES C
     CUR_TEMP = 0;                 // CURRENT TEMP DEFAULT 0
     TEMP_MODE = 0;                // TEMP MODE = 0
     AD_VAL = 0;                   // A/D VAL = 0
}


/********************************************************************/
/* This function enables PLL and use an 8-MHz crystal oscillator to */
/* generate 24-MHz E clock. Same as done in assembler.              */
/********************************************************************/
void SetClk8(void)
{
     asm(sei);                     // turn of interrupts
     CLKSEL  &= PLLSEL;            // disengage PLL from system
     SYNR    = 0x02;               // set SYSCLK to 24 MHz from a 4-MHz oscillator
     REFDV   = 0;                  //
     PLLCTL  = 0x40;               // turn on PLL, set automatic
     while(!(CRGFLG & LOCK));      // wait for HIGN on LOCK bit at address CRGFLG
     asm(nop);                     // very short delays
     asm(nop);                     // very short delays
     CLKSEL  |= PLLSEL;            // clock derived from PLL
     asm(cli);                     // turn on interrups
}
/********************************************************************/
/*  This subroutine initializes the SPI system on the HC12S         */
```

```
/********************************************************************/
void SPIInitialize(void)
{
     SPIB = 0x22;                  //SPI CLOCKS A 1/24 OF E-CLOCK
     DDRM = 0x3B;                  //SETUP PORTM DATA DIRECTION
     SPCR1 = 0x50;                 //ENABLE SPI AND SET MODE AS MASTER
     SPCR2 = 0x00;                 //RESETS SPCR2 TO $00 (ALSO DOES AT RESET)
     PORTM = PORTM | RCK;          //SET RCK TO IDLE HIGH
     PORTM = PORTM & ~ENABLE;      //ENABLE TO IDLE LOW
}
/********************************************************************/
/*  This subroutine initializes the LCD screen                      */
/********************************************************************/
void InitLCD(void)
{
        Command(0x30);                //Call command method with 0x30
        delay3();                     //Allow the command to take place
        Command(0x30);                //Call command method with 0x30
        delay3();                     //Allow the command to take place
        Command(0x30);                //Call command method with 0x30
        delay3();                     //Allow the command to take place
        Command(0x38);                //Call command method with 0x38
        delay3();                     //Allow the command to take place
        Command(0x0C);                //Call command method with 0x0C
        delay3();                     //Allow the command to take place
        Clear();                      //Clear the homescreen
}
/********************************************************************/
/*  This subroutine initializes the timer interrupt                 */
/********************************************************************/
void initInterrupts(void)
{
     TSCR2 = 0x04;                 //CONFIGURE PRESCALE FACTOR 16 (2/3 usec) 1500 =
1msec
     TIOS = 0x07;                  //ENABLE OC0,OC1,OC2 FOR OUTPUT COMPARE
     TSCR1 = 0x90;                 //ENABLE TCNT & FAST FLAGS CLEAR
     TIE = 0x07;                   //ENABLE INTERRUPTS TC1,TC2,TC3

     asm(ANDCC #$BF);              //ENABLE XIRQ'
}
/********************************************************************/
/*  This subroutine starts the timer interrupts up                  */
/********************************************************************/
void startInterrupts(void)
{
     TC0 = TCNT + 15000;           //INCREMENT TC0 BY 3750 (.01SECOND)
     TC1 = TCNT + 15000;           //INCREMENT TC1 BY 3750 (.01SECOND)
     TC2 = TCNT + 15000;           //INCREMENT TC2 BY 3750 (.01SECOND)
     TFLG1 = 0x07;                 //SET FLAGS FOR TIMER INTERRUPTS
}
/*********************************************************************************/
/*                                                                               */
/* The getkey functions gets the key value from a 4X4 matrix keypad connected    */
/* PortT. Rows (0,1,2,3) = P4,P5,P6,P7                                           */
/*     Columns (0,1,2,3) = P0,P1,P2,P3                                           */
/* The strategy used here is nessted if -else statements and is similar to what  */
/* we did in assembly language. There are more efficient and elegant strategies. */
/*                                                                               */
/*********************************************************************************/
char getkey(void)
{                                   // We test the keys in sequence - row 0 columns
0,1,2,3
                                    // row 2 columns 0,1,2,3 etc. until we have
```

```
checked
   char keyX;                       // all of the keys. If a key is pressed then we
save the
                                    // value in keyX and jump down to return without
                                    // checking any more keys.  Note that there many
                                    // more ways to do this.

   PORTT = 0x00;                    // clear portT
   asm(NOP);                        // short wait times with assembler NOP
   PORTT |= 0x10;                   // PORTT = PORTT | 0x10; OR PORTT with $10. ie.
set row 0 (PT4) High
   asm(nop);                        // ASSEMBLY
   asm(nop);                        // ASSEMBLY
   asm(nop);                        // ASSEMBLY

   if(PORTT & BIT0)                 // AND PORT with 0x01 and check if ans is 1
(TRUE). ie. Check column

0 for HIGH. If High
           keyX = 1;                // then set keyX to 1 and jump to return.
   else if(PORTT & BIT1)            // Check column 1
           keyX = 2;
   else if(PORTT & BIT2)            // Check column 2
           keyX = 3;
   else if(PORTT & BIT3)            // Check column 3
           keyX = 10;
   else {
          PORTT = 0x00;             // Clear PortT and start on row 1
          PORTT |= 0x20;            //  Set row 1 High
          asm(nop);
          asm(nop);
          asm(nop);

          if(PORTT & BIT0)          // Check column 0 etc., etc.
           keyX = 4;
          else if(PORTT & BIT1)
           keyX = 5;
          else if(PORTT & BIT2)
           keyX = 6;
          else if(PORTT & BIT3)
           keyX = 11;
          else {
              PORTT = 0x00;
              PORTT |= 0x40;        // row 2 High
              asm(nop);
              asm(nop);
              asm(nop);

              if(PORTT & BIT0)
                keyX = 7;
              else if(PORTT & BIT1)
                keyX = 8;
              else if(PORTT & BIT2)
                keyX = 9;
              else if(PORTT & BIT3)
                keyX = 12;
              else {
                PORTT = 0x00;
                PORTT |= 0x80;     // row 3 High
                asm(nop);
                asm(nop);
                asm(nop);
                if(PORTT & BIT0)
                  keyX = 0 ;
                else if(PORTT & BIT1)
                  keyX = 15;
                else if(PORTT & BIT2)
                  keyX = 14;
                else if(PORTT & BIT3)
                  keyX = 13;
                else               // if we get to here ==> no key pressed
                  keyX = 0x1f;     // nokey signal
              }
          }
   }
   return (keyX);                  // return the key value
}
/************************************************************/
/* Key release routine. Check each coulmn bit. If HIGH wait */
/* until it goes LOW to break out of the while statement.   */
/* Note that we are reading the input register of PortT     */
/* which is at address $0241 and is called (here) PORTTi    */
/************************************************************/
void keyrelease(void)
{
     //PORTT = 0xf0                // Set all rows high (not needed here. Why?)
     while((PORTTi & 0x01));       // if column 0 is HIGH wait here until LOW
     while((PORTTi & 0x02));       // if column 1 is HIGH wait here until LOW
     while((PORTTi & 0x04));       // if column 2 is HIGH wait here until LOW
     while((PORTTi & 0x08));       // if column 3 is HIGH wait here until LOW
}
/********************************************************************/
/*  This subroutine creates a small delay which counts clock cycles */
/********************************************************************/
void delay(void)
{
     int y = 8000;                 //Initialize Y as 8000
        int i = 0;                    //Initialize i as 0
        for(i; i<=y;i++);             //Do the delay 8000 times
}
/********************************************************************/
/*  This is a slightly larger delay than delay(), it uses a nested  */
/*  loop to increase the time spent here                            */
/********************************************************************/
void delay3(void)                  //This delay has nested while loops - count clock
cycles
{
        int y = 0x0F;                 //Iniialize Y as $0F
        while (y!=0){                 //Loop while Y!=0
                int x = 0xFFFF;          //Initialize X as $FFFF
                while(x!=0){             //Loop while X!=0
                        x--;                //Decrement X
                }
                y--;                     //Decrement Y
        }
}
/*********************************************************************/
/* The following function creates a time delay which is equal to the */
/* multiple of 1ms. The value passed in k specifies the number of    */
/* milliseconds to be delayed.                                       */
/*********************************************************************/
void delayby1ms(int k)
{
     /* Standard Timer Setup */
        int ix;                       //DECLARE ix VAR
        TIOS  |= BIT4;                /* enable OC4 */
        TFLG1 &= BIT4;                /* clear timer flag OC4F*/
        TC4 = TCNT + 1500;            /* add 375 to the tcount*/

        for(ix = 0; ix < k; ix++)     // Do this loop k times. Where k*1ms is the
~time wait we need. Not

necessarily 1 second.
        {
             while(!(TFLG1 & BIT4));  // ASM==> Here BRCLR TFLAG1, $01, Here
                TC4 += 1500;             // If we get here TFLAG1's BIT0 became HIGH
     }
        TIOS  &= (~BIT4);             /* disable OC0  and exit.  note no return
statement required*/
}
/*********************************************************************/
/*  This function clears the LCD screen                              */
/*********************************************************************/
void Clear(void)
{
        Command(0x01);                //Sends the clear command to LCD
}
/*********************************************************************/
/*  This function prints spaces to the LCD                           */
/*********************************************************************/
void printSpaces(char spaces)
{
     unsigned char n;              //DECLARE n VAR
        for(n=0; n < spaces;n++)      //LOOP spaces TIMES
     {
          Print(0x20);             //PRINT A SPACE " "
     }
}
/*********************************************************************/
/*  This subroutine sends a command to the LCD, for example to move */
/*  the cursor to the beginning of the screen                       */
/*********************************************************************/
void Command(char a)
{
        while(!(SPISR & 0x20));       //Wait for register empty flag (SPIEF)
        SPIDR = a;                    //Output command via SPI to SIPO
        while(!(SPISR & 0x80));       //Wait for SPI Flag
        a = SPIDR;                    //Equate a with SPIDR
        asm(nop);                     //Wait for 1 cycle
        PORTM &= ~RCK;                //Pulse RCK
        asm(nop);                     //Wait for 1 cycle
        asm(nop);                     //Wait for 1 cycle
        PORTM |= RCK;                 //Command now available for LCD
        PORTM &= ~RS;                 //RS = 0 for commands
        asm(nop);                     //Wait for 1 cycle
        asm(nop);                     //Wait for 1 cycle
        asm(nop);                     //Wait for 1 cycle
        PORTM |= ENABLE;              //Fire ENABLE
        asm(nop);                     //Wait for 1 cycle
        asm(nop);                     //Wait for 1 cycle
        PORTM &= ~ENABLE;             //ENABLE off
        delay();                      //Delay
}
/*********************************************************************/
/*  This subroutine prints an ASCII character to the screen         */
/*********************************************************************/
void Print(char a)
```

```
{
        while(!(SPISR & 0x20));       //Wait for register empty flag (SPIEF)
        SPIDR = a;                    //Output command via SPI to SIPO
        while(!(SPISR & 0x80));       //Wait for SPI Flag
        a = SPIDR;                    //Equate a with SPIDR
        asm(nop);                     //Wait for 1 cycle
        PORTM &= ~RCK;                //Pulse RCK
        asm(nop);                     //Wait for 1 cycle
        asm(nop);                     //Wait for 1 cycle
        PORTM |= RCK;                 //Command now available for LCD
        PORTM |= RS;                  //RS = 1 for data
        asm(nop);                     //Wait for 1 cycle
        asm(nop);                     //Wait for 1 cycle
        asm(nop);                     //Wait for 1 cycle
        PORTM |= ENABLE;              //Fire ENABLE
        asm(nop);                     //Wait for 1 cycle
        asm(nop);                     //Wait for 1 cycle
        PORTM &= ~ENABLE;             //ENABLE off
        delay();                      //Delay
}
/********************************************************************/
/*             MICROWAVE ELEMENT (LED) TIMER INTERRUPT              */
/********************************************************************/
interrupt void tch2ISR(void)
{

     TC2 = TC2 + 15000;                                 //INCREMENT COUNT BY 15000

     if(PAUSE == 0x00)                                  //IF TIMER IS ON AND GREATER 
THAN 0
     {
          POWER_INTERRUPT_COUNT++;                      //INCREMENT COUNT FOR 
INTERRUPT
          if(POWER_INTERRUPT_COUNT == 50)               //IF WE'RE AT .5 SECONDS
          {
               POWER_FLAG = 0x01;                       //SET POWER FLAG
               POWER_INTERRUPT_COUNT = 0;               //RESET INTERRUPT COUNT
          }
     }
}
/********************************************************************/
/*  Timer Interrupt Service Routine for Time 1, which is to be      */
/*  used for the stop-watch counter                                 */
/********************************************************************/
interrupt void tch1ISR(void)
{

     TC1 = TC1 + 15000;                                 //INCREMENT COUNT BY 15000
     //IF TIMER IS ON AND GREATER THAN 0 OR WE'RE IN TEMP MODE
     if(PAUSE == 0x00)
     {
          TIMER_INTERRUPT_COUNT++;                      //INCREMENT THE COUNT
          if(TIMER_INTERRUPT_COUNT == 100)              //IF WE'VE COUNTED 100
          {
               if(TIMER_VALUE > 0)
               {
                    TIMER_VALUE--;                      //DECREASE VALUE TIME
               }
               TIMER_FLAG = 0x01;                       //SET TIMER FLAG
               TIMER_INTERRUPT_COUNT = 0;               //RESET INTRPT COUNT
          }

          if(TIMER_VALUE == 0 && TEMP_MODE == 0)        //IF TIMER LESS THAN 0
          {
               COMPLETE = 0x01;                        //SET COMPLETE FLAG
          }

          //IF TEMP REACHED
          if((CUR_TEMP >= SET_TEMP) && (TEMP_MODE == 1))
          {
               COMPLETE = 0x01;                        //SET COMPLETE FLAG
          }

     }
}
/********************************************************************/
/*   Real time clock interrupt service routine.  The interrupt      */
/*   is used to run the real time clock                             */
/********************************************************************/
interrupt void tch0ISR(void)
{

     TC0 = TC0 + 15000;                      //INCREMENT COUNT BY 15000 (.01SECOND)

     if(CLOCK_SPEED == 0x00)                 //ARE WE AT NORMAL SPEED?
     {
          CLOCK_INTERRUPT_COUNT++;           //INCREMENT CLOCK INTERRUPT COUNTER
          if(CLOCK_INTERRUPT_COUNT == 100)   //IF IT'S AT 100 (1 SECOND)
          {
               CLOCK_TIME++;                 //INCREMENT CLOCK TIME
               CLOCK_FLAG = 0x01;            //SET THE FLAG
               CLOCK_INTERRUPT_COUNT = 0;    //RESET THE COUNT
          }

     }
     else
     {
          CLOCK_TIME++;                      //INCREMENT CLOCK TIME
          if(CLOCK_TIME % 60 == 0) {         //IF COUNT IS AN EVEN MINUTE
               CLOCK_FLAG = 0x01;            //SET THE FLAG (100X FASTER)
          }
     }
     if(CLOCK_TIME >= 86399) CLOCK_TIME=0;   //IF CLOCK TIME > 11:59PM THEN RESET

}
/********************************************************************/
/*   This is the IRQ' subroutine.  It's purpose is to pause or      */
/*   clear the operation of the microwave element.  It operates     */
/*   by changing the PAUSE flag to 0x01 if the flag is currently    */
/*   0x00, or if the flag is already 0x01, it will clear the        */
/*   timer value to zero.                                           */
/********************************************************************/
interrupt void irqISR(void)
{
     delayby1ms(200);              //DELAY 200 MSECS

     if(PAUSE == 0x01)             //IF WE'RE ALREADY IN PAUSE MODE
     {
          TIMER_VALUE = 0;         //RESET THE TIMER VALUE
          SYSTEM_MODE = 99;        //RESET THE SYSTEM MODE
          TEMP_MODE = 0;           //RESET THE TEMP MODE
          updateDisplay();         //REFRESH THE DISPLAY
     }
     else if(PAUSE == 0x00)        //IF WE'RE IN RUN MODE
     {
          PAUSE = 0x01;            //CHANGE TO PAUSE MODE
          PTAD = PTAD & ~BIT7;     //MAKE SURE THE LED IS OFF
     }
     updateDisplay();
}
/********************************************************************/
/*   This is the XIRQ' subroutine.  It's purpose is to start or     */
/*   continue the operation of the microwave operation.  It         */
/*   operates by changing the PAUSE flag to value 0x00.             */
/********************************************************************/
interrupt void xirqISR(void)
{
     delayby1ms(200);                   //DELAY 200 MSECS

     if(TIMER_VALUE > 0 || TEMP_MODE == 1)   //IF TIMER_VALUE GREATER THAN ZERO
                                             //OR TEMP_MODE IS SET
     {
          TC1 = TCNT + 15000;           //MAKE SURE INCREMENT THE TIMER
          TC0 = TC1;                    //MAKE SURE INCREMENT THE TIMER
          PAUSE = 0x00;                 //CHANGE SYSTEM TO RUN MODE



          TIMER_FLAG = 0x01;            //SET THE TIMER FLAG TO UPDATE DISPLAY
          SYSTEM_MODE = 99;             //CLEAR SYSTEM MODE TO 99
          COMPLETE = 0x00;                 //COMPLETE SET TO 0 (NOT COMPLETE)
     }
     updateDisplay();                   //UPDATE THE DISPLAY
}
```