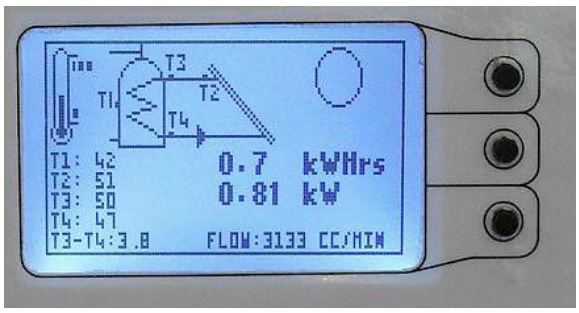
APPLICATION NOTE

LCD6402B Solar Heated Water Controller

1 INTRODUCTION

This application note describes the implementation of an LCD6402 128x64 graphic display module and PIC16F876A to form a solar heated water controller. The solar controller features a user interface, watt meter, kilowatt hour meter and 12 hour solar activity history graph.



2 IMPLEMENTATION

Figure 1 below shows a typical indirect vented solar heated domestic water system. Water is heated by the solar panel and then circulated by pump through a heat exchanger located in the solar tank. In this case, cold water in the tank is preheated before being past to a normal vented tank (not drawn) where it is topped up with heat from a boiler or immersion heater as required.

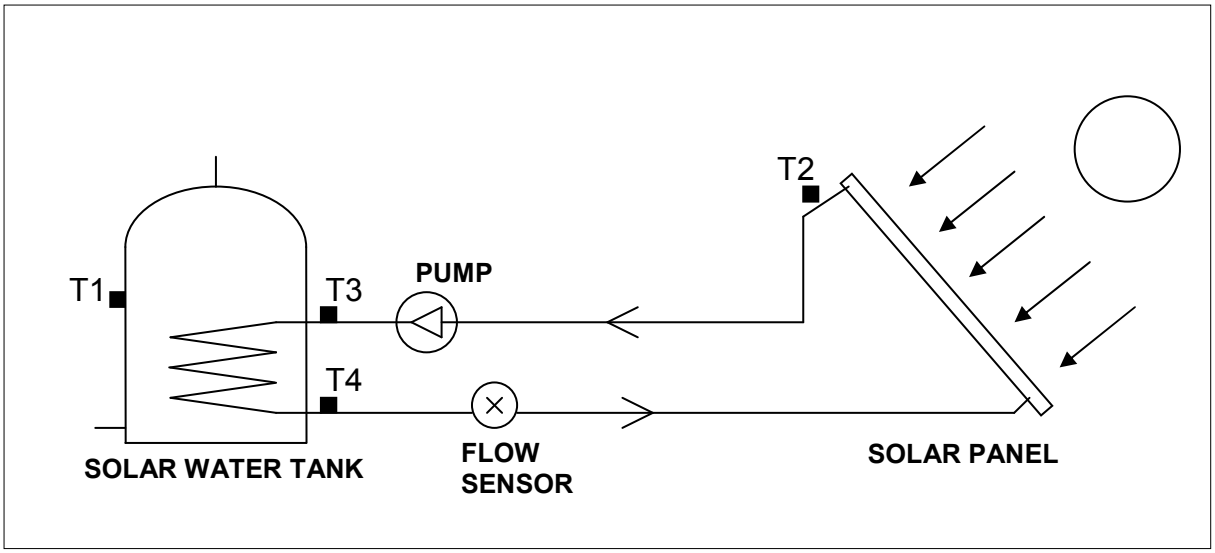


Figure 1 Solar heated water schematic

In a normal solar controller system there are two temperature sensors, T1 and T2. When the solar panel temperature (T2) exceeds the solar water tank temperature (T1) by a preset amount (usually around 3 °C) the circulation pump is switched on and likewise switches the pump off when T2 falls below T1.

In this implementation there is the addition of a power monitor. To measure power, both the flow rate of the directly heated water (measured by the flow sensor) and the difference in temperature between water entering and leaving the heat exchanger coil (measured by T3 and T4), need to be known as well as the elapsed time. It's then a matter of computation to obtain the power. Once power is known, watt hours can also be found by integrating power using a simple accumulator.

2.1 CIRCUIT DESCRIPTION

The full circuit schematic is show in Appendix A. Here U1, a PIC16F876A[2] (PIC) Microchip microcontroller operating from a 5V supply measures the sensor outputs, controls the pump and drives the user interface. LM335 temperature sensors are used throughout which may be located many meters from the controller Screening is provided for these by two core screened cable terminated at the controller end (see schematic).

The LM335[1] temperature sensors, which have a sensitivity of 10mV/K, are directly connected to U1's analogue to digital converter (ADC) inputs. However in order to measure the temperature difference between T3 andT4

with greater resolution a differential amplifier formed by U3 provides gain of approximately five before being presented to the ADC input on U1. Preset VR1 provides a means of zeroing the output of U3:A when initially setting up. Relay RL1 operates the pump via Q1 and flow sensor Y1 pulse output is fed to an interrupt input on U1. The LCD6402 128x64 graphic LCD module is interfaced via a level shifted 3V3 I2C bus.

2.2 USER INTERFACE

The user interface consists of the LCD6402[3] 128x64 graphic LCD module incorporating three user keys positioned vertically down the right side of the display.

Information is presented to the user in as two simple displays, the status display and the history graph display.

2.2.1 Status Display

The status display in Figure 2 below shows a tank thermometer, system schematic, pump activity icon, bath icon and various meter readouts.

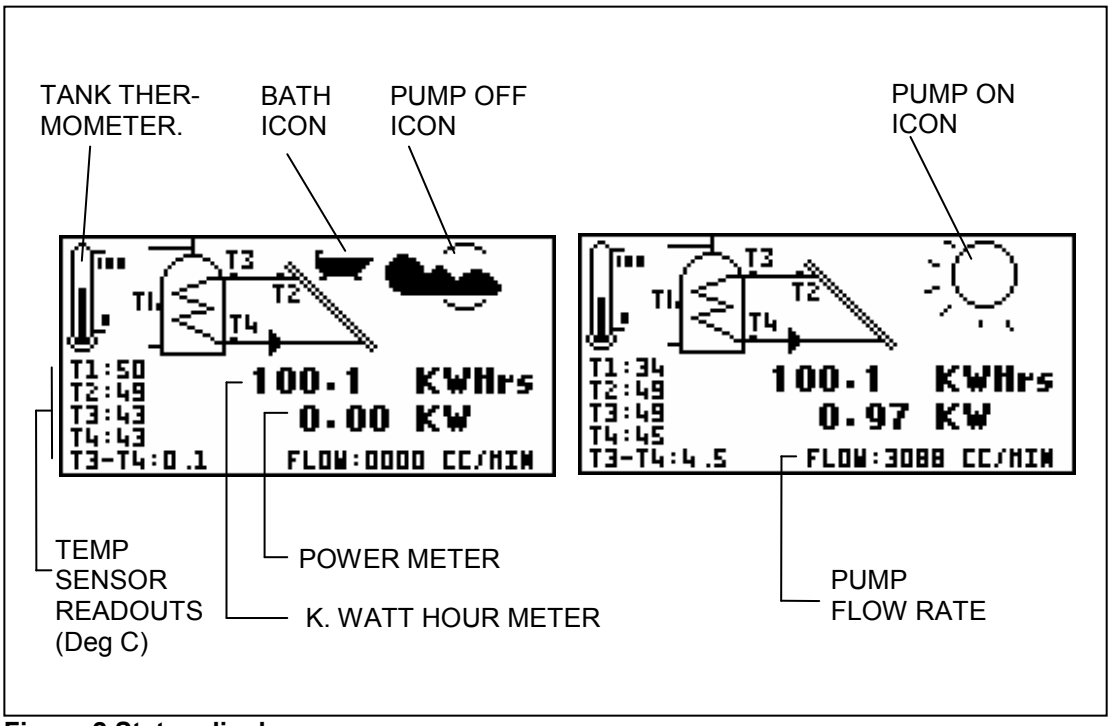


Figure 2 Status display

In more detail these are:

TANK THERMOMETER - Shows the temperature of the solar tank as an analogue thermometer (0^{o}C to 100^{o}C).
BATH ICON - This is present when the tank temperature is 45 ^{o}C or above and may be used to indicate an adequate amount of hot water for a bath

PUMP ICON - Shows a clouded sun icon when the solar panel is colder than the tank temperature (pump off), or an animated sun icon when the panel is hotter (by 3 ^{o}C) than the tank.
TEMP SENSOR READOUTS - Readouts of all temperature sensors and differential temperature T3-T4
POWER METER - Shows the real time power in KW of heat being transferred to the tank.
K. WATT HOUR METER - Shows the total accumulated kilo watt hours since the unit began operating.
PUMP FLOW RATE - The flow rate of the water in the solar circuit in cubic centimeters per minute.

The status display is made up of static and dynamic graphics. Static graphics and custom fonts are created using LCDLAB[4] and stored in the LCD6402 module and are recalled by firmware via the I2C bus. Dynamic graphics, such annotations are generated by firmware in the PIC. This approach both frees up code space in the PIC and allows changes to static graphics and fonts to be easily made using LCDLAB.

2.2.2 History Graph Display

The history graph display in Figure 3 below shows the power due to solar activity over the last 12 hours and the total accumulated watt hours for that period.

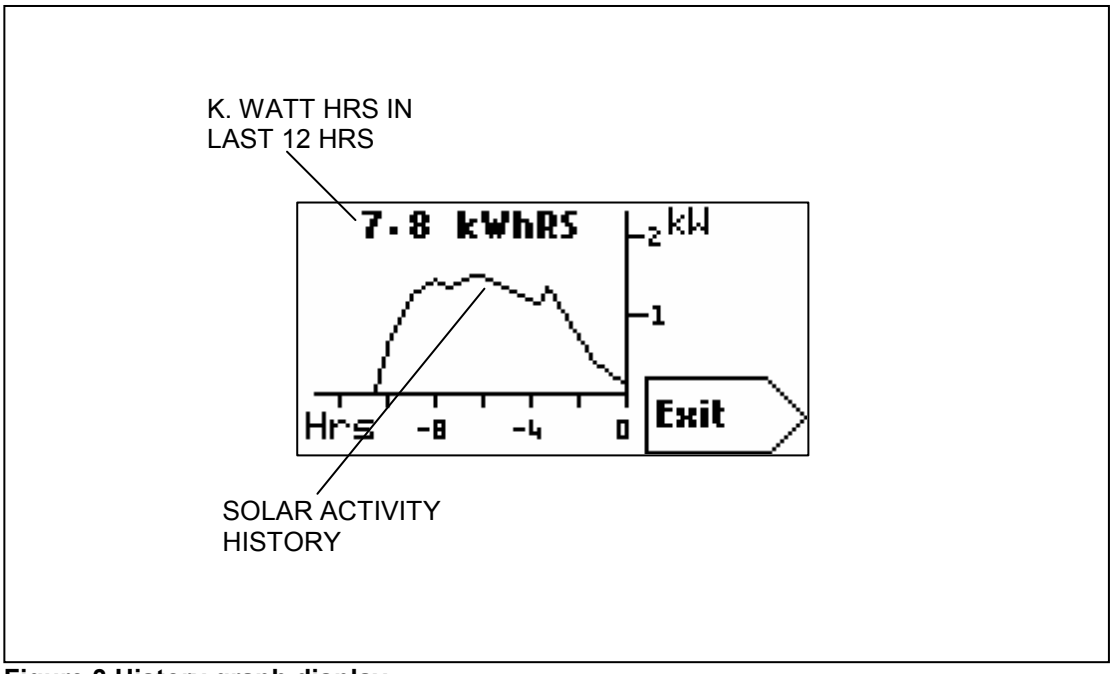


Figure 3 History graph display

The history graph display is accessed by pressing any of the user keys (not shown) and can be exited by pressing the bottom key or automatically after ten seconds in the absence of any key press.

The history graph display is created in a similar way to the status display. There are static graphics for the graph and text and the line and K. WATT HRS are drawn by firmware.

2.3 FIRMWARE DESCRIPTION

The firmware (program) is written in C and has been compiled using a CCS[5] C PCM compiler. The software listing is shown in APPENDIX B.

The program operates as a real time event driven single thread servicing user interface and I/O. Event code requires the use of function pointers, however the PIC cannot support these in the normal way so pseudo function pointers are used instead (see DoEvents() and TFuncID in solarctrl.c).

Execution begins at main() which initialize the program and hardware, shows the status display and enters the main loop repeatedly executing DoEvents(). The purpose of DoEvents() is to detect events and call the handler associated with the event.

Executed every second are:

SolarStateMachine() - This state machine controls the circulation pump according to the difference in temperature between the panel and tank. The various states are described in the source listing.
ReadTempSensors() - All temperature sensors are read and scaled to degrees Celsius, except for T3-T4 which is Celsius x10[1].
_clock.OnSecsTick() - When the status display is shown, updates the display animation and annotations.

Executed every 10 seconds are:

UpdateFlowmeter() - The flow meter produces 4600 pulses per litre. The count accumulated by INTB0_ISR() interrupt is converted to a flow rate in cubic centimetres per minute.
UpdatePowermeter() - Computes the power for the10 second period using flow rate and T3-T4. Power is accumulated in _meter.WHrs360 (watt hours x 360).

Executed every minute:

EveryMinuet() - In every 20 minutes stores the current power in the log and increments the log to the next position. There is sufficient space in the log for a 12 hour history period.

Executed every hour:

SaveWattmeter() - Saves the current watt meter value to non volatile memory.

The user interface is controlled by three keys. When a key is pressed the event is detected and the associated handler is executed. The handlers are assigned by the current user display (i.e. Status() or Graph()). When for example the key press handler for the status display, Status_KeyDown(), is

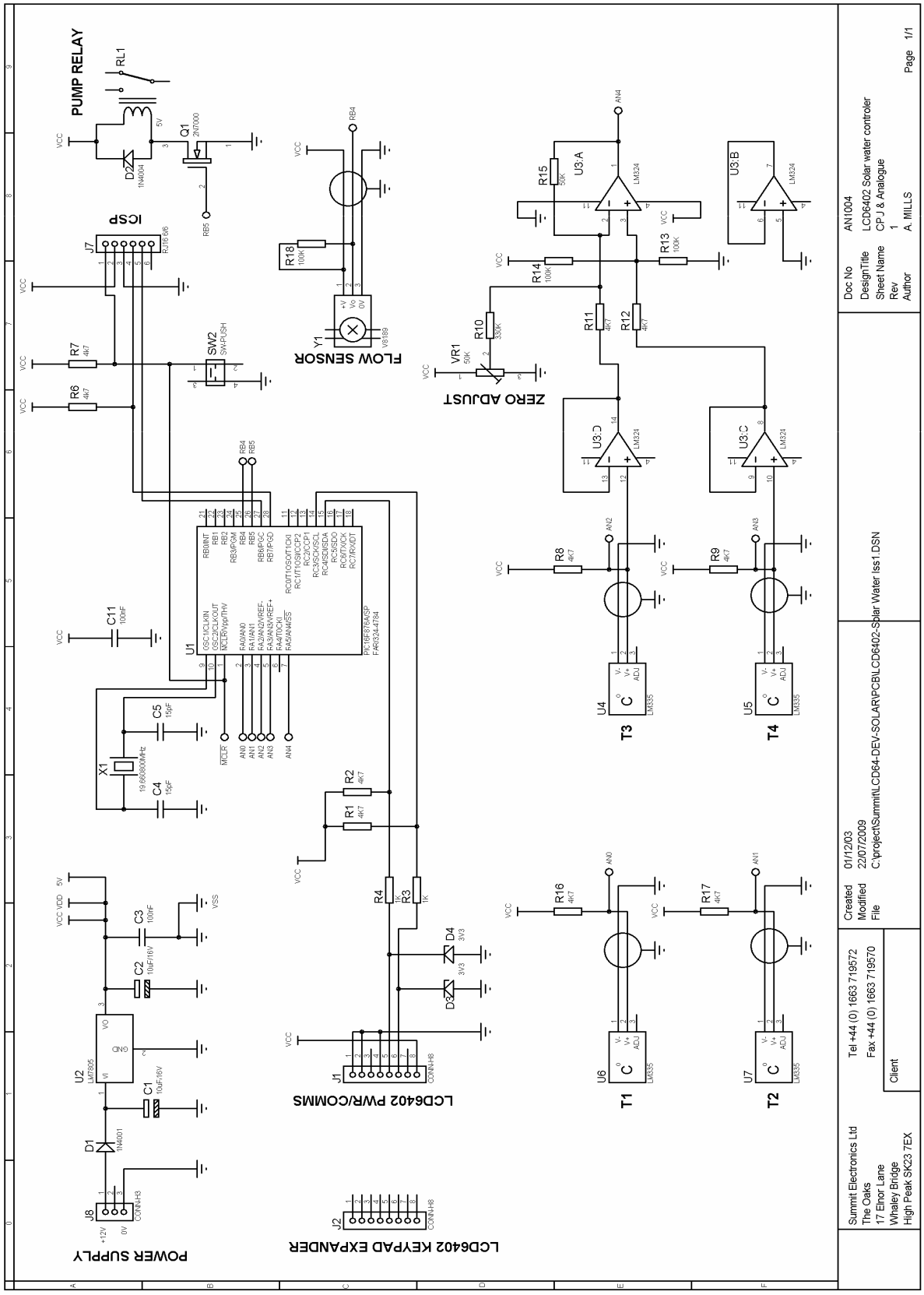
executed, the history graph display is shown and likewise when Graph_KeyDown() is executed the status display is shown.

The static graphics are drawn by Status() or Graph() which simple call a single routine to recall the required graphic from the LCD6402. Dynamic graphics are drawn by a few lines of code. For example the thermometer bar is drawn by two rectangles, one to erase the top portion of the thermometer bar and one to draw the visible lower portion (see Status_Update()).

3 REFERENCES

1. ST Microelectronics (www.st.com). LM335Z data sheet.
2. Microchip (www.microchip.com). PIC16F876A data sheet.
3. Summit Electronics Ltd (www.summitelectronics.co.uk). LCD6402B data sheet (http://www.summitelectronics.co.uk/products/PDF/LCD6402b.pdf)
4. Summit Electronics Ltd (www.summitelectronics.co.uk). LCDLAB software tool.
5. Custom Computer Services (www.ccsinfo.com).

APPENDIX A. CIRCUIT SCHEMATIC



APPENDIX B. SOURCE CODE

lcd64xx.h

```
////////////////////////////////////////////////////////////////////
////                                                            ////
//// File   : lcd64xx.h                                       ////
////                                                            ////
//// Created: 11-05-06                                        ////
////                                                            ////
//// Author : A.Mills. Summit Electronics Ltd.                ////
////                                                            ////
//// Description                                              ////
//// -----------                                              ////
//// Header file for lcd64xx.c                                ////
////                                                            ////
////////////////////////////////////////////////////////////////////


#ifndef lcd64xx_h
#define lcd64xx_h


// LCD64 registers
#define rCMD                    0x20
#define rSTATUS                 0x21
#define rNEWKEY                 0x22
#define rKEY                    0x23
#define rCUX                    0x24
#define rCUY                    0x25
#define rORX                    0x26
#define rORY                    0x27
#define rNVML                   0x28
#define rNVMH                   0x29
#define rOPTION                 0x2A
#define rINS                    0x2B
#define rCONTRAST               0x2C
#define rFGCLR                  0x2D
#define rBGCLR                  0x2E
#define rMODE                   0x2F
#define rFONT                   0x30

// LCD64 rCMD command defines
#define cmdRESET_INTERPRETER    0x91
#define cmdBACKLIGHT_ON         0x92
#define cmdBACKLIGHT_OFF        0x93
#define cmdCLEAR_SCREEN         0x94
#define cmdSAVE_CONTEXT         0xa8
#define cmdRESTOR_CONTEXT       0xa9
#define cmdRESET                0xff

// LCD64 rINS graphic instruction defs  // Expected parameters
#define gRESET                  0x08    //
#define gCLS                    0x21    //
#define gORIGIN                 0x2A    // x, y
#define gCURSOR                 0x3A    // x, y
#define gFGCLR                  0x41    // clr
#define gBGCLR                  0x42    // clr
#define gMODE                   0x43    // mode
#define gPIX                    0x4A    // x, y, n
#define gLINETO                 0x52    // x, y
#define gRECT                   0x5C    // x0, y0, x1, y1
#define gRECTF                  0x64    // x0, y0, x1, y1
#define gFONT                   0x69    // f
#define gPUTC                   0x71    // c
```



```
#define gHRAST                  0x79    // rast
#define gPUTS                   0x80    // c0..cn, 0
#define gHBMP                   0x8A    // w/8, H, rdata,
#define gCIRCLE                 0x99    // r
#define gCIRCLEF                0x9A    // r
#define gVRAST                  0xA1    // r
#define gVBMP                   0xAA    // w, h/8, rdata,
#define gCALLI                  0xF1    // n

// LCD64 rFGCLR & rBGCLR register colour defines
#define clrBLACK                0
#define clrWHITE                0xFF

// LCD64 rMODE register defines
#define mdFG_OR                 0x00    // OR foreground colour.
#define mdFG_XOR                0x01    // XOR foreground colour.
#define mdFG_ON                 0x00    // Foreground colour ON.
#define mdFG_OFF                0x02    // Foreground colour OFF.
#define mdBG_OR                 0x00    // OR backgound colour.
#define mdBG_XOR                0x10    // XOR backgound colour.
#define mdBG_ON                 0x00    // Backgound colour ON.
#define mdBG_OFF                0x20    // Backgound colour OFF.



void LCD64_INSWrite( BYTE data);
void LCD64_CMDWrite( int cmd);
void LCD_Cursor(int x,y);
void LCD_Origin(int x,y);
void LCD_Putc(int c );
void LCD_Font( int f );
void LCD_LineTo( int x, y);
void LCD_Circle( int r, fill);
void LCD64_RegWrite(int reg,int data);

#endif
```

lcd64xx.c

```
////////////////////////////////////////////////////////////////////
////                                                            ////
//// File   : LCD64XX.C                                         ////
////                                                            ////
//// Created: 11-05-06                                          ////
////                                                            ////
//// Author : A.Mills. Summit Electronics Ltd.                  ////
////                                                            ////
//// Description                                                ////
//// -----------                                                ////
//// LCD64XX driver functions                                   ////
////                                                            ////
////////////////////////////////////////////////////////////////////
#include "lcd64xx.h"

//////////////////////////////////////////////////////
//                                                  //
//          LCD64 DEFINES                           //
//                                                  //
//////////////////////////////////////////////////////

#use i2c(sda=PIN_C4, scl=PIN_C3, FORCE_HW)


//////////////////////////////////////////////////////
//                                                  //
```

```
//          LCD64 INTERFACE FUNCTIONS                   //
//                                                      //
//////////////////////////////////////////////////////////



/*********************************************
*** LCD64_Read
***
*** Read value from LCD64 address.
***
*** Parameters:
***     addr    : Address to read from.
***
*** Returns: value read from address.
*********************************************/
#inline
int LCD64_Read(int addr)
{
    int r;

    i2c_start();
    i2c_write(0xa0);
    i2c_write(addr);
    i2c_start();
    i2c_write(0xa0 | 1);
    r=i2c_read(0);
    i2c_stop();
    return(r);
}

/*********************************************
*** LCD64_Write
***
*** Write value to LCD64 address.
***
*** Parameters:
***     addr    : Addresss to write to.
***     data    : Value to write.
*********************************************/
#inline
void LCD64_Write(int addr,int data)
{
    i2c_start();
    i2c_write(0xa0);
    i2c_write(addr);
    i2c_write(data);
    i2c_stop();
}


/*********************************************
*** LCD64_BusyWait
***
*** Wait while LCD64 is busy.
*********************************************/
#inline
void LCD64_BusyWait( void)
{
    int status;

    do {
        delay_ms(1);
        status = LCD64_Read(0x21);
        status &= 0x80;
    } while ( status != 0);
}
```

```
/*********************************************
*** LCD64_CMDWrite
***
*** Wait for LCD64 busy flag to clear and
*** write value to command register (20H).
****
*** Parameters:
***     cmd     : Command to write.
*********************************************/
void LCD64_CMDWrite( int cmd)
{
    LCD64_BusyWait();
    LCD64_Write( 0x20, cmd);
}

/*********************************************
*** LCD64_INSWrite
***
*** Wait for LCD64 busy flag to clear and
*** write value to instruction register (2BH).
****
*** Parameters:
***     data    : Instruction to write.
*********************************************/
void LCD64_INSWrite( BYTE data)
{
    LCD64_BusyWait();
    LCD64_Write( 0x2b, data);
}

/*********************************************
*** LCD64_RegWrite
***
*** Wait for LCD64 busy flag to clear and
*** write value to register.
****
*** Parameters:
***     reg     : Register to write to.
***     data    : Value to write.
*********************************************/
void LCD64_RegWrite(int reg,int data)
{
    LCD64_BusyWait();
    LCD64_Write( reg, data);
}



//////////////////////////////////////////////////////////
//                                                      //
//          LCD WRAPPER FUNCTIONS                       //
//                                                      //
//////////////////////////////////////////////////////////


/*********************************************
*** LCD_Cursor
***
*** Set graphic cursor.
***
*** Parameters:
***     x   : x cursor position.
***     y   : y cursor position.
*********************************************/
void LCD_Cursor(int x,y)
{
    LCD64_BusyWait();
```

```
    LCD64_Write( 0x2b, gCURSOR);
    LCD64_Write( 0x2b, x);
    LCD64_Write( 0x2b, y);
}


/*********************************************
*** LCD_Origin
***
*** Set graphic origin.
***
*** Parameters:
***     x   : x origin.
***     y   : y origin.
*********************************************/
void LCD_Origin(int x,y)
{
    LCD64_BusyWait();
    LCD64_Write( 0x2b, gORIGIN);
    LCD64_Write( 0x2b, x);
    LCD64_Write( 0x2b, y);
}


/*********************************************
*** LCD_Putc
***
*** Print character to display.
***
*** Parameters:
***     c   : Character to print.
*********************************************/
void LCD_Putc(int c )
{
    LCD64_BusyWait();
    LCD64_Write(rINS, 0x71);
    LCD64_Write(rINS, c);
}

/*********************************************
*** LCD_Font
***
*** Set LCD font.
***
*** Parameters:
***     f   : Font type.
*********************************************/
void LCD_Font( int f )
{
    LCD64_BusyWait();
    LCD64_Write(rINS, 0x69);
    LCD64_Write(rINS, f);
}

/*********************************************
*** LCD_LineTo
***
*** Draw line to position.
***
*** Parameters:
***     x   : x position.
***     y   : y position.
*********************************************/
void LCD_LineTo( int x, y)
{
    LCD64_BusyWait();
    LCD64_Write( rINS, gLINETO);
    LCD64_Write( rINS, x);
```

```
    LCD64_Write( rINS, y);
}

/*********************************************
*** LCD_Circle
***
*** Draw filled or unfilled circle.
***
*** Parameters:
***     r   : Radius.
***     fill: 0=unfilled, 1=filled.
*********************************************/
void LCD_Circle( int r, fill)
{
    LCD64_BusyWait();
    if (fill)
        LCD64_Write( rINS, gCIRCLEF);
    else
        LCD64_Write( rINS, gCIRCLE);

    LCD64_Write( 0x2b, r);
}

/*********************************************
*** LCD_Rectangle
***
*** Draw filled or unfilled rectangle.
***
*** Parameters:
***     x0  : Rectangle left.
***     y0  : Rectangle top.
***     x1  : Rectangle right.
***     y1  : Rectangle bottom.
***     fill: 0=unfilled, 1=filled.
*********************************************/
void LCD_Rectangle( int x0, int y0, int x1, int y1, int fill)
{
    LCD64_BusyWait();
    if ( fill)
        LCD64_Write( rINS, gRECTF);
    else
        LCD64_Write( rINS, gRECT);

    LCD64_Write( rINS, x0);
    LCD64_Write( rINS, y0);
    LCD64_Write( rINS, x1);
    LCD64_Write( rINS, y1);
}
```

solarctrl.c

```
////////////////////////////////////////////////////////////////
////                                                        ////
//// File   : solarctrl.c                                   ////
////                                                        ////
//// Created: 11-05-06                                      ////
////                                                        ////
//// Author : A.Mills. Summit Electronics Ltd.              ////
////                                                        ////
//// Description                                            ////
//// -----------                                            ////
//// Solar control application.                             ////
////                                                        ////
//////////////////////////////////////////////////////////// ////////

#include <16F876A.h>
```

```
#device adc=16
#FUSES WDT,HS,NOPUT,NOPROTECT,NODEBUG,BROWNOUT,NOLVP,NOCPD,NOWRT
#use delay(clock=19660800,RESTART_WDT)
#use rs232(baud=9600,parity=N,xmit=PIN_C6,rcv=PIN_C7,bits=8)

#case
#zero_ram

////////////////////////////////////////////////////////
//  HEADER FILES                                      //
////////////////////////////////////////////////////////

#include "lcd64xx.h"


////////////////////////////////////////////////////////
//  SOURCE FILES                                      //
////////////////////////////////////////////////////////

#include "lcd64xx.c"


////////////////////////////////////////////////////////
//  PROTOTYPES                                        //
////////////////////////////////////////////////////////
void Status( void );
void PumpMenu( void );


////////////////////////////////////////////////////////
//  DEFINES                                           //
////////////////////////////////////////////////////////

// Pump operating parameters.
#define PUMP_TMIN       12          // Min pump on time.
#define DELTA_CMIN      3           // Min temp diff (T2-T1) before pump
operate.

// LCD64 stored images (symbols produced by LCDLAB.
#define sun1_bmp 0
#define sun2_bmp 1
#define sun3_bmp 2
#define sun4_bmp 3
#define cloud_bmp 4
#define bath_bmp 5
#define graph_lfm 6
#define status_lfm 7


// Keys along right side of LCD64
#define KEY_A 13    // Top
#define KEY_B 14    // Middle
#define KEY_C 15    // Bottom


// Clock.
typedef struct {
    int1        Show;
    int1        SecTickEvent;
    int         OnSecsTick;
} TClock;


// Virtual timer.
typedef struct {
    long    Counter;
    int1    Event;
```



```
    int1    Enabled;
    int     OnTimerExpire;
} TVTimer;


// Log.
#define LOG_SIZE        36          // Size of log in bytes.
typedef struct {
    unsigned int Data[LOG_SIZE];    // Storage for data.
    unsigned int Pos;               // Log position.
} TLog;

// Solar controler states.
enum TSolarState {
    ssOff,
    ssAutoOff,
    ssAutoOnMinRun,
    ssAutoOn,
    ssMinRun
};


// Sensor data structure.
typedef struct {
    signed long       DeltaC;       // T2-T1
    signed long       TankC;        // C[0]
    signed int        FlowDeltaC;
    signed long       SolarFlowC;
    unsigned long     FlowCounter;
    unsigned long     FlowRate;

    // Scaled temperature (DegC) data by index.
    // 0 (T1) Tank temp.
    // 1 (T2) Solar flow panel temp.
    // 2 (T3) Solar flow tank temp.
    // 3 (T4) Solar return tank temp.
    // 4 (T3-T4) Diff temp * 10.
    signed long      C[5];
} TSensor;


// Meters.
typedef struct {
    signed long     Watts;
    unsigned long   WHrs;
    signed int32    WHrs360;
} TMeter;


// Dispatcher function ID's.
enum TFuncID
    {
    fnNull,
    fnStatus_KeyDown,
    fnStatus_Update,
    fnStatus,
    fnGraph_KeyDown
    };

// Key handler.
typedef struct {
    TFuncID OnKeyDown;
    int Value;
} TKey;


////////////////////////////////////////////////////
//  VARIABLES                                     //
```

```
////////////////////////////////////////////////////

static TKey         _key;           // Key handler.
static TClock       _clock;         // Realtime clock.
static TVTimer      _vSecsTimer;    // Virtual 1 seccond timer.
static TVTimer      _vSecsMenuTimer;// Virtual 1 seccond timer
static TLog         _log;           // Log.
static TSolarState  _solarState;    // Solar controler state.
static TMeter       _meter;         // Meter data.
static TSensor      _sensor;        // Sensor data.
static int1         _pumpRunning;   // Pump run flag.
static int          _updateRobin;   // Used by status display.


/***********************************************
* PumpOn
*
* Switch the pump on.
***********************************************/
void PumpOn(void)
{
    _pumpRunning = 1;
    output_high(PIN_B5);
}


/***********************************************
* PumpOff
*
* Switch the pump off.
***********************************************/
void PumpOff(void)
{
    _pumpRunning = 0;
    output_low(PIN_B5);
}


/***********************************************
* INTB0_ISR
*
* Flow meter pulse interrupt.
***********************************************/
#int_RB
void INTB0_ISR(void)
{
    // Inc flow counter on rising edge.
    if (!input(PIN_B4) )
        _sensor.FlowCounter++;
}


/***********************************************
* RTCC_ISR
*
*  Timer interrupt service routine for
*  driving clock and virtual timers
*  of LCD.
***********************************************/
#int_rtcc
void RTCC_ISR(void) {
    static BYTE ticks=0;

    // CLOCK //
    // 75 ticks per second
    if (++ticks == 75) {
        _clock.SecTickEvent = 1;
```

```
        ticks = 0;
    }

    // Virtual timers //

    // 1 sec timers.
    if (ticks==0) {
        if (_vSecsTimer.Counter != 0 ) {
            _vSecsTimer.Counter--;
            if (_vSecsTimer.Counter == 0)
                _vSecsTimer.Event = 1;
        }

        if (_vSecsMenuTimer.Counter != 0 ) {
            _vSecsMenuTimer.Counter--;
            if (_vSecsMenuTimer.Counter == 0)
                _vSecsMenuTimer.Event = 1;
        }
    }
}



/*********************************************
* EEPROM_Read
*
* Read byte from EEPROM at addr.
*********************************************/
int EEPROM_Read(int addr)
{
    return read_eeprom(addr);
}

/*********************************************
* EEPROM_Write
*
* Write byte to EEPROM at addr.
*********************************************/
void EEPROM_Write(int addr, value)
{
    write_eeprom(addr, value);
}


/*********************************************
* Graph_Update
* Graph_KeyDown
* Graph
*
* Graph display functions.
*********************************************/

void Graph_Update(void)
{
    int i, y, x;
    unsigned long   wHrsIn12Hrs;

    //--------------------//
    // Draw graph statics //
    //--------------------//
    LCD64_CMDWrite(cmdCLEAR_SCREEN);
    LCD64_CMDWrite(graph_lfm);


    //------------------------//
    // Render graph with line //
    //------------------------//
    for (i=0; i<LOG_SIZE; i++) {
```

```
        y = 48 - (_log.Data[_log.Pos] / 5);
        x = i*2+10;
        if (i)
            LCD_LineTo(x, y);
        else
            LCD_Cursor(x, y);

        // Inc log pos to next.
        _log.Pos++;
        if (_log.Pos >= LOG_SIZE)
            _log.Pos = 0;
    }

    //----------------//
    // Print watt hrs //
    //----------------//

    // Compute WHrs over last 12 hours.
    // There are 3 log readings per hour
    wHrsIn12Hrs = 0;
    for (i=0; i<LOG_SIZE; i++) {
        wHrsIn12Hrs += _log.Data[i];
    }
    wHrsIn12Hrs /= 3;

    // Print result.
    LCD_Cursor(2,2);
    LCD_Font(8);
    printf(LCD_Putc, "%5.2Lw kWHrs", wHrsIn12Hrs );

}


//
// Graph display key down hander.
//
void Graph_KeyDown(void)
{
    // Any key down goto status display.
    Status();
}

//
// Load the graph display.
//
void Graph(void)
{
    _key.OnKeyDown = fnGraph_KeyDown;   // Assign key handler.
    Graph_Update();
}


/*********************************************
* Status_Update
* Status_KeyDown
* Status
*
* Status display functions.
*********************************************/


//
// Updates the status display.
//
void Status_Update(void)
{
    int i, x, y;
```

```
signed int t1;


//---------------------------------//
// Draw 'sun' or 'cloud' graphics  //
//---------------------------------//
if (++_updateRobin > 3 )
    _updateRobin = 0;

LCD_Cursor(83,1);
if (_pumpRunning) {
    // Draw animated sun.
    LCD64_CMDWrite(sun1_bmp + _updateRobin);
} else {
    // Draw clouds.
    LCD64_CMDWrite(cloud_bmp);
}
restart_wdt();


// Update ramainder of display every four seconds.
if (_updateRobin != 1)
    return;


//-------------------//
// Update themometer //
//-------------------//
t1 = _sensor.C[0];
if (t1 > 99)
    t1 = 100;
else if (t1 < 0)
    t1 = 0;
y = 24 - (t1 / 5);              // Compute bar position.

// Draw themometer top in white.
LCD64_RegWrite(rFGCLR, clrWHITE);
LCD_Rectangle(  5, 4, // x0, y0,
                6, y, // x1, y1,
                1     // fill
                );

// Draw thermometer bottom in black.
LCD64_RegWrite(rFGCLR, clrBLACK);
LCD_Rectangle(  5, y, // x0, y0,
                6, 24,// x1, y1,
                1     // fill
                );
restart_wdt();


//-------------------------------//
// Print temperatures T1 to T4  //
//-------------------------------//

// T1 to T4
LCD_Font(0);
for (i=0; i<4; i++) {
    LCD_Cursor(2, 32+(i*6));
    printf(LCD_Putc, "T%d:%3Ld ", i+1, _sensor.C[i] );
    restart_wdt();
}

// Print temperature difference (T3-T4).
LCD_Cursor(2, 32+(i*6));
printf(LCD_Putc, "T3-T4:%2.1Lw", _sensor.C[i] );
restart_wdt();
```



```
    //------------------//
    // Print flow rate  //
    //------------------//

    LCD_Cursor(50, 56);
    printf(LCD_Putc, "  FLOW:%04Lu cc/min", _sensor.FlowRate );
    restart_wdt();


    //----------------//
    // Draw bath icon //
    //----------------//

    // If tank temperature 45 DegC or more, draw bath icon
    // else undraw bath icon.
    if ( _sensor.C[0] >= 45)
        i = clrBLACK;
    else
        i = clrWHITE;

    LCD_Cursor(65, 4);          // Position of icon.
    LCD64_RegWrite(rFGCLR, i);  // Set the colour.
    LCD64_CMDWrite(bath_bmp);   // Draw the icon.
    LCD64_RegWrite(rFGCLR, clrBLACK);   // Restore colour.
    restart_wdt();


    //--------------//
    // Print meters //
    //--------------//

    //-- Print watt hour meter.
    LCD_Font(8);
    LCD_Cursor(32, 33);
    printf(LCD_Putc, "%8.1Lw  kWHrs", _meter.WHrs);

    //-- Print watt _meter.
    LCD_Cursor(32, 42);
    printf(LCD_Putc, "%9.2Lw kW", _meter.Watts/10 );
    restart_wdt();

}

//
// Status keydown handler.
//
void Status_KeyDown(void)
{
    // Any key pressed goto graph display.
    _clock.OnSecsTick = fnNull;     // Turn updates off.
    Graph();
}

//
// Load the status display.
//
void Status(void)
{
    _key.OnKeyDown = fnStatus_KeyDown;  // Assign key handler.
    LCD64_CMDWrite(cmdCLEAR_SCREEN);    // Show the status graphic.
    LCD64_CMDWrite(status_lfm);
    _updateRobin = 0;                   // Zero the robin.

    _clock.OnSecsTick = fnStatus_Update;      // Assign 1/sec update rate.

    _vSecsMenuTimer.OnTimerExpire = fnStatus;// Set up auto complete timer.
    _vSecsMenuTimer.Enabled = 0;
```

```
}


/*********************************************
* UpdateFlowmeter
*
* Calculate flowrate every 10 secs.
*********************************************/

void UpdateFlowmeter(void)
{
    int32     capture;

    // Capture the current flow counter and reset the
    // flow rate counter for the next period.
    disable_interrupts(INT_RB);
    capture = _sensor.FlowCounter;
    _sensor.FlowCounter = 0;
    enable_interrupts(INT_RB);

    // Calculate flow rate.
    // Calculate flow rate in cc/min given
    // flow meter sensitivity = 4600 pulses per ltr.
    //
    //  Flow rate:
    //  cc/min = capture * ((min/10) * 1000 / 4600)
    //  where   capture is 4600 pulses / ltr / 10sec,
    //          min is 16 secconds.
    // Also ((min/10) * 1000 / 4600)) approximates to (256 / 197)
    capture *= 256;
    capture /= 197;
    _sensor.FlowRate = capture;    // Flow rate (cc/min).

}

/*********************************************
* UpdatePowermeter
*
* Do every 10 secs.
*********************************************/
void UpdatePowermeter(void)
{
    signed int32 power;

    // General formulae for power in flow of heated water, given
    // 1W = 1 joule/sec and 1W = 60 joules(j)/min.
    //
    // Power for a flow rate in units of ltr/min.
    // P = Q * T * c / 60
    //
    //  where:  P = power in watts
    //          Q = flowrate (ltr/min)
    //          T = change in temperature (DegC)
    //          c = specific heat capacity (4200/kg/DegC for water)
    //
    // Measured q (flowrate cc/min) and t (temp. *10) are scalled so that:
    //  Q = q/1000
    //  T = t/10
    //
    // Therefore:
    //  P = (q/1000) * (t/10) * (4200 / 60)
    //  and
    //  P = q * t * (4200 / 60) / 10000
    //  and
    //  P = q * t * 70 / 10000

    power = (signed int32)_sensor.FlowRate *
        (signed int32)_sensor.C[4] *
```

```
        (signed int32)70;
    power /= 10000;
    _meter.Watts = power;

    // Accumulate watt hrs
    _meter.WHrs360 += power;

    if (_meter.WHrs360 > 9999999*360)
        _meter.WHrs360 = 0;

    // Scale WHrs360 to Kilo watt hours X 10
    _meter.WHrs = _meter.WHrs360 / (int32)36000;
}

/*********************************************
* FrostWarning
*
* Returs true if any temp sensor reads
* below 3 DegC.
*********************************************/
int FrostWarning(void)
{
    int i;

    for(i=0; i<4; i++) {
        if ( _sensor.C[i] < 3) {
            return 1;
        }
    }
    return 0;
}


/*********************************************
* SolarStateMachine
*
* Run every second.
*********************************************/
void SolarStateMachine(void)
{

    switch( _solarState ) {

        // Off //
        case ssOff :
            // Stay in this state until panel temperature exceeds 100 DegC.
            // This is a test state and not normaly entered into.
            if ( _sensor.C[1] > 100 )
                _solarState = ssAutoOff;
            break;

        // AutoOff: Pump is off //
        case ssAutoOff :
            // Change state to AutoOnMinRun and turn pump on when the
            // difference in temperature between the panel and
            // tank (DeletaC) is greater than the preset
            // min (DELTA_CMIN).
            if ( _sensor.DeltaC > DELTA_CMIN) {
                _solarState = ssAutoOnMinRun;
                _vSecsTimer.Counter = PUMP_TMIN * 60;
                PumpOn();

            } else if ( FrostWarning() ) {
                // Frost protection
                PumpOn();

            } else {
                // Anything else, switch pump off.
```

```
                PumpOff();
            }
            break;

        // AutoOnMinRun: Pump is on //
        case ssAutoOnMinRun :
            // Stay in this state untill the min run timer counter
            // (_vSecsTimer.Counter) has expired.
            if (_vSecsTimer.Counter == 0)
                // Next state AutoOn.
                _solarState = ssAutoOn;
            break;

        // AutoOn: Pump is on //
        case ssAutoOn :
            // Stay in this state until temperature difference between the
            // panel and tank is less than the preset minimum.
            if (_sensor.DeltaC < DELTA_CMIN) {
                // Next state AutoOff.
                _solarState = ssAutoOff;

                // Swutch pump off.
                PumpOff();
            }
            break;

        // Illegal state //
        default:
            _solarState = ssAutoOff;
    }
}


/*********************************************
* ReadTempSensors
*
* Read all temperature _sensors and apply
* scaling.
*********************************************/
void ReadTempSensors(void)
{
    int i;
    long adc;

    // Read and scale channels 0 to 3. (T1 to T4).
    // Temp sensor sensitivity: 10mV/K.
    for (i=0; i<4; i++) {
        set_adc_channel(i);
        delay_us(100);
        adc = read_adc();
        adc = adc / 131;
        _sensor.C[i] = (adc - 273);
    }

    // Read and scale channel 4 (T3-T4).
    // Amplifier gain: 50/4.7
    // Temp diff = 108mV/DegC
    set_adc_channel(4);
    delay_us(100);
    adc = read_adc();
    adc -= 32767;           // Substract 2.5V (mid rail) worth.
    (signed long)adc = (signed long)adc / 141; // Scale to temp X 10.
    _sensor.C[4] = adc;


    _sensor.TankC = _sensor.C[0];
    _sensor.DeltaC = _sensor.C[1] - _sensor.C[0];
    _sensor.FlowDeltaC = _sensor.C[4];
```

```
}


/*********************************************
* SaveWattmeter
*
* Save watt _meter to non-volatile memory.
*********************************************/
void SaveWattmeter(void)
{
    int *pdata, i;

    pdata = &_meter.WHrs360;

    for (i=0; i<sizeof(_meter.WHrs360); i++) {
        EEPROM_Write(i, *pdata);
        pdata++;
        restart_wdt();
    }
}

/*********************************************
* LoadWattmeter
*
* Load watt meter from non-volatile memory.
*********************************************/
void LoadWattmeter(void)
{
    int *pdata, i;

    // Read saved watt meter from EEPROM.
    pdata = &_meter.WHrs360;
    for (i=0; i<sizeof(_meter.WHrs360); i++) {
        *pdata = EEPROM_Read(i);
        pdata++;
    }

    // Zero if EEPROM blank.
    if (_meter.WHrs360 < 0)
        _meter.WHrs360 = 0;
}


/*********************************************
* EveryHour
*
* Occurs every hour.
*********************************************/
void EveryHour(void)
{
    SaveWattmeter();
}


/*********************************************
* EveryMinuet
*
* Occurs every minuet.
*********************************************/
void EveryMinuet(void)
{
    static int minCount = 60;
    static int minCount20 = 20;
    signed long watts;

    // Every 20 mins.
    if (minCount20-- == 0) {
        minCount20 = 20;
```

```
        // Log watts rescalled to deci Watts
        // limit range.
        watts = _meter.Watts / 10;
        if (watts > 255)
            watts = 255;
        else  if (watts < 0)
            watts = 0;

        _log.Data[_log.Pos] = watts;
        _log.Pos++;
        if (_log.Pos >= LOG_SIZE)
            _log.Pos = 0;
    }

    // Every hour.
    if (minCount-- == 0) {
        minCount = 60;
        EveryHour();
    }
}


/*********************************************
* EverySecond
*
* Occurs every second.
*********************************************/
void EverySecond(void)
{
    static int secCount10 = 10;
    static int secCount = 60;

    // Every seccond functions.
    ReadTempSensors();
    SolarStateMachine();


    // Every 10 seconds.
    if (secCount10-- == 0) {
        secCount10 = 10;
        UpdateFlowmeter();
        UpdatePowermeter();
    }

    // Every minuet.
    if (secCount-- == 0 ) {
        secCount = 60;
        EveryMinuet();
    }
}


/*********************************************
* DoEvents
*
* Detects events and calls event handlers.
*********************************************/
void DoEvents(void)
{
    TFuncID dispatch;

    dispatch = fnNull;

    //-- Service anu pending events.
    _key.Value = LCD64_Read(rNEWKEY);
    delay_ms(50);
    if ((_key.Value !=0) && (_key.Value !=0xff)) {
```

```
        // Key event.
        dispatch = _key.OnKeyDown;
        _vSecsMenuTimer.Counter = 10;

    } else if (_clock.SecTickEvent) {
        // Timer clock event (1/sec tick).
        _clock.SecTickEvent = 0;
        dispatch = _clock.OnSecsTick;
        EverySecond();

    } else if (_vSecsMenuTimer.Event) {
        // vSecsMenuTimer event (Secconds one shot timer).
        _vSecsMenuTimer.Event = 0;
        dispatch = _vSecsMenuTimer.OnTimerExpire;

    } else if (_vSecsTimer.Event) {
        // vSecsTimer event (Secconds one shot timer).
        _vSecsMenuTimer.Event = 0;
        dispatch = _vSecsTimer.OnTimerExpire;
    }

    //-- Dispatcher.
    switch (dispatch) {
        case fnNull : break;
        case fnStatus_KeyDown:   Status_KeyDown(); break;
        case fnStatus_Update:    Status_Update(); break;
        case fnStatus:           Status(); break;
        case fnGraph_KeyDown:   Graph_KeyDown(); break;
        default: dispatch = fnNull;
    }
    dispatch = fnNull;
}


/*********************************************
* initialise
*
* Initialise hardware etc.
*********************************************/
void initialise(void)
{
    int8 *pData, i;

    LCD64_CMDWrite(cmdCLEAR_SCREEN);
    LCD64_CMDWrite(cmdBACKLIGHT_ON);

    output_float(PIN_A0);
    setup_adc_ports(ALL_ANALOG);
    setup_adc(ADC_CLOCK_INTERNAL);
    setup_counters(RTCC_INTERNAL, RTCC_DIV_256);    // 75 interrupts / sec
    enable_interrupts(INT_TIMER0);                  // (19.6608MHz Xtal)

    output_low(PIN_B7);
    output_low(PIN_B6);
    output_low(PIN_B5);
    input(PIN_B4);

    enable_interrupts(INT_RB);
    enable_interrupts(GLOBAL);

    _solarState = ssAutoOff;
    _meter.WHrs360 = 0;

    // Clear _sensor data
    pData = &_sensor;
    for (i=0; i<sizeof(TSensor); i++) {
        *pData++ = 0;
    }
```



```
    // Clear log.
    _log.Pos = 0;
    for (i=0; i< LOG_SIZE; i++) {
        _log.Data[i] = 0;
    }


    _vSecsTimer.OnTimerExpire = fnNull;

    LoadWattmeter();
}

/*********************************************
* main
*
* Program entry point.
*********************************************/
void main(void)
{
    initialise();               // Initialise.
    Status();                   // Start menu system.

    // Main program loop.
    while(1) {
        DoEvents();
        restart_wdt();
    }
}
```

