AN1216

Setback Thermostat Design Using the NEURON® IC

INTRODUCTION

For several years setback thermostats (SBTs) have been microprocessor driven, giving the end–user energy–efficient control over HVAC (Heating, Ventilating, and Air Conditioning) systems. However today's designs still present system limitations such as a single point of failure and lack of flexibility. For example most SBTs have central control over all units (i.e., heater and AC) in a HVAC system, which requires point–to–point wiring and limits the performance capabilities of the system to those of the SBT processor. A viable alternative is to distribute processing to each unit in the system over a network. The MC143150 (NEURON IC) is a multimedia communications and control processor with an embedded standard protocol that lends itself to easy network communication of control information. Specifically, the open architecture design of the LONTALK protocol allows an OEM to independently design interoperable HVAC units as well as network interfaces to non–HVAC systems without writing a cumbersome and costly software protocol. Using the NEURON IC, a setback thermostat can provide control data (e.g., the time of day, time/temperature set points, and current temperature) to a network of "smart" units with NEURON ICs which can each interpret data according to system specifications (see Figure 1).

This application note describes how a NEURON IC can be used as a SBT processor. NEURON IC functions include a 3×4 keypad interface for programmability; a 6–digit, 7–segment LCD display interface for time and temperature indication; a temperature–to–frequency converted input; and a software real–time clock. See Figure 2 for a schematic of the system hardware. Although the NEURON SBT node tends to be more expensive than the traditional SBT, cost savings will result from ease of installation, reduction in wiring, and higher system reliability (lower rate of product maintenance and return). Each of the hardware interfaces and its related software functions will be described in the following sections. Additionally, the NEURON C software for the SBT is included in Print Out 1 at the end of this document. (See the *NEURON C Programmer's Guide* for details on unfamiliar syntax. Request document NEURONCPG/AD.)

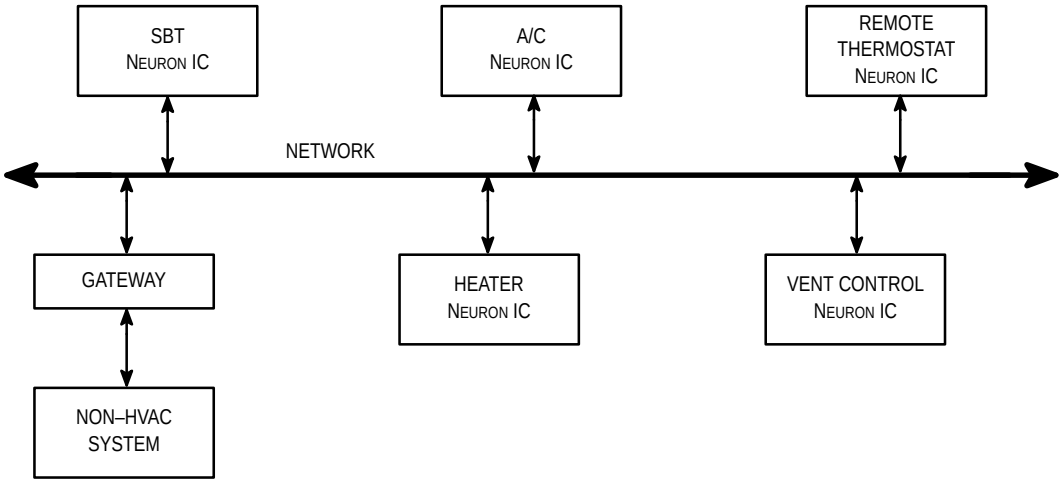

Figure 1. Block Diagram of Distributed HVAC System

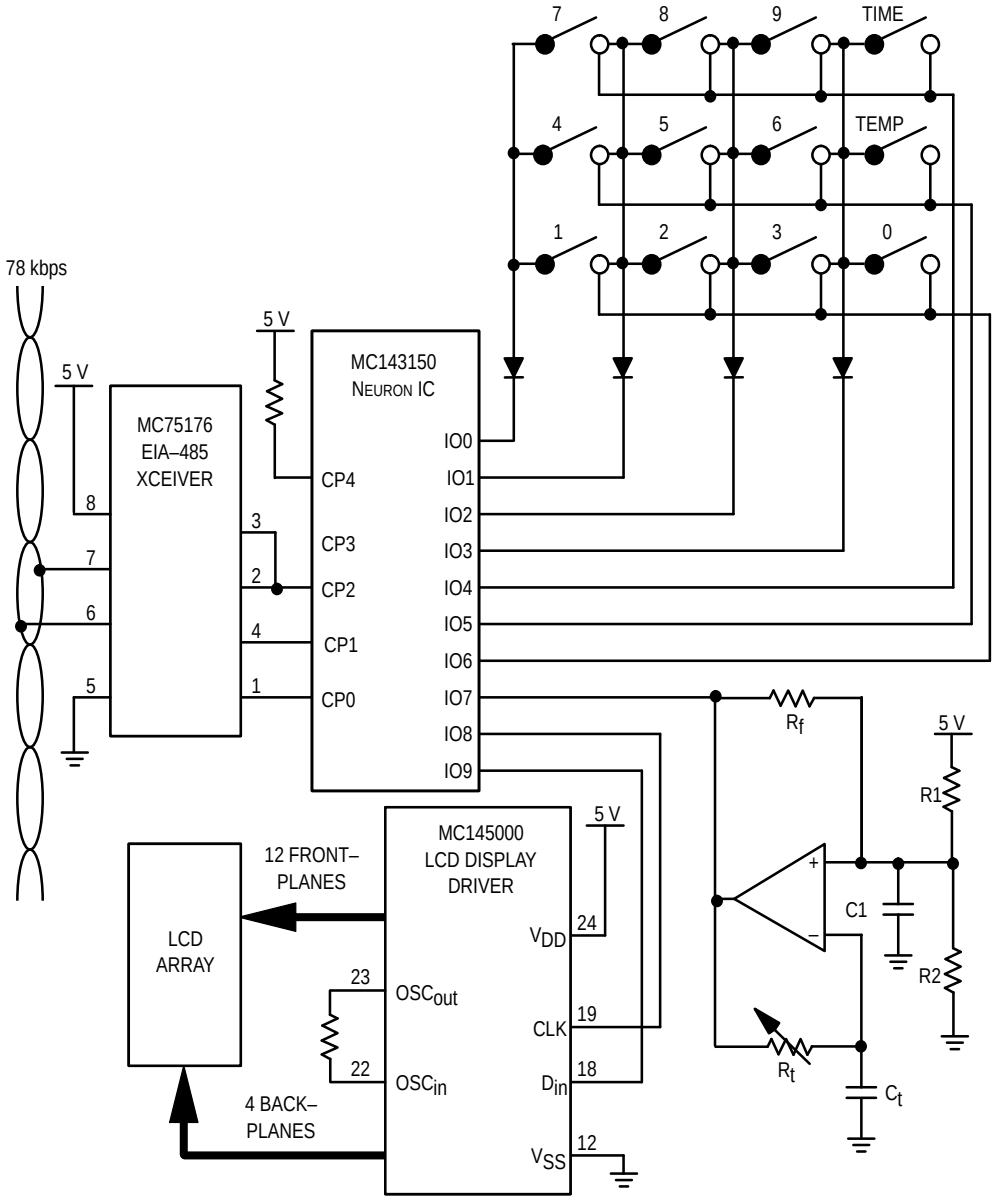

Figure 2. Setback Thermostat System Hardware

KEYPAD INTERFACE

Hardware

This interface is quite similar to the 4 × 4 keypad described in the application note entitled, “Scanning a Keypad With the NEURON Chip” (EB151/D). The major difference is that the rows are monitored by three 1–bit control lines on the NEURON IC as opposed to one 4–bit nibble.

Software

The keypad is read as follows: the software periodically scans the three keypad rows until a key is depressed, at which time each column is driven low until the row of the depressed key is determined. See Figure 3 for an illustration of the SBT keypad.

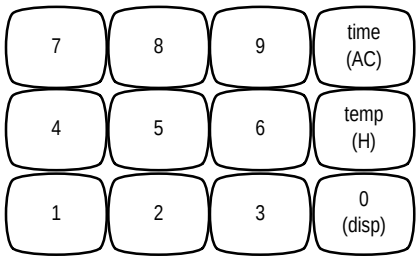

Figure 3. SBT Keypad

The SBT is programmed using the keypad command sequences described in Figure 4. A maximum of three time/ temperature set points can be saved for each unit (heater and air conditioner). Data structures of “temp_time” type store a temperature value as well as an hour and minute value. Seven such structures have been created for each air conditioner set point (ac_1 – ac_3), each heater set point (heat_1 – heat_3), and the current time and temperature (sbt_data).

DISPLAY INTERFACE

Hardware

This interface uses a 6–digit, 7–segment LCD display driven by an MC145000 Serial Input Multiplexed LCD Driver interfaced to the SBT NEURON IC processor. Two I/O lines are required from the NEURON IC: clock and data out (a chip select line and data out are not necessary).

Software

The display driver IC receives data via NEUROWIRE (identical to Motorola’s SPI and National’s Microwire) from the NEURON IC. The display will change when the time changes (once per second), any time the SBT is being programmed, and while the end user is sequencing the SBT through its display modes (described immediately below).

Under normal operating conditions (normal mode) the SBT will provide current military time (hours, minutes) and current temperature on its display. Also the decimal after the hour indicator will alternately blink either on or off each second. When in display mode (entered by pressing the ‘‘0’’ key), the SBT will display the first set point time and temperature for the air conditioner. If no other key is pressed for 5 seconds (at any time during display mode) the SBT will

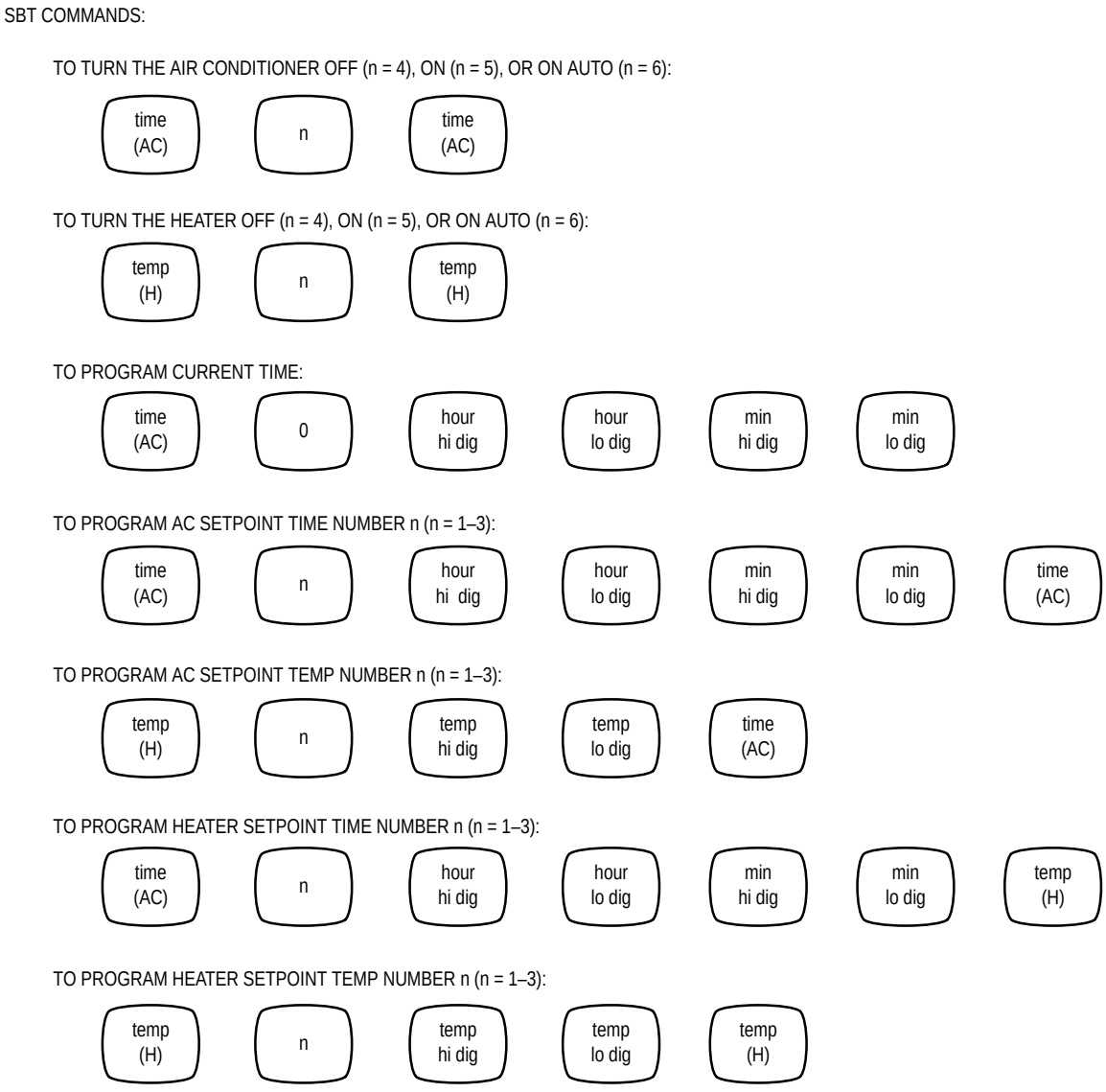

Figure 4. Keypad Command Sequences to Program the NEURON SBT

return to the current time and temperature display. If “0” is pressed again before the 5 second timeout, the SBT will display the second set point time and temperature for the air conditioner. Similarly, as “0” is successively pressed, the SBT will display time and temperature set points for the remaining AC set point followed by each of the three set points for the heater. When in programming mode (entered by pressing either the “time” or “temp” key), the display will actively display numeric keypad sequences (see Figure 4 for programming command sequences).

TEMPERATURE SENSOR INTERFACE

Hardware

The temperature sensor interface is implemented with a comparator using an RC combination in its feedback loop. The R component is a thermistor. When the circuit is initially powered up, Point B (refer to Figure 5) has a “high” voltage value. Point A is “low” so the RC circuit charges up in an attempt to make A equal to B. Eventually the RC circuit forces the voltage at A above the voltage at B, causing point C, and hence B, to go “low”. In an attempt to make point A equal to B again, the RC circuit discharges. Eventually the RC circuit discharges too much, causing point C, and thus point B, to go “high” again. At this time the process repeats itself, resulting in a periodic square wave from the temperature sensor output C, which serves as a “frequency” input to the NEURON IC.

The “high” and “low” points discussed above are determined by the values of R1, R2, R_f, and R_p. The frequency range of the output C is determined by the value of C_t and the characteristic of the thermistor R_t (as the RC time constant changes, the rise and fall times of graph A in Figure 5 change).

Software

The frequency input from the temperature sensor is read using a frequency input object called “temp_signal_in”, which is periodically read (every 2 seconds). The frequency value is then converted to a temperature using a look–up table that was created using the characteristic of the thermistor used. The temperature range of this SBT is 32 to 122 degrees Fahrenheit. The current temperature value is stored in the “temp” field of the “sbt_data” structure.

REAL–TIME CLOCK

The real–time clock interface is implemented in software since the I/O port of the NEURON IC is dedicated to keypad, display, and temperature sensor interfaces. A millisecond timer object called read_timer is programmed to expire every 984 milliseconds, at which time the software checks for changes in minutes, hours, and days. Additionally, the software automatically updates the 1 second counter (984 ms added to the average software delay of 16 ms equals 1 second). The software time delay was experimentally determined by running the real–time clock for long periods of time and comparing its output to an accurate timepiece. The current time in minutes and hours is stored in the “sbt_data” structure.

NETWORK VARIABLES

The SBT has time, temperature, and status data which it may submit to the heater and air conditioner units on the network. These units can receive current time and temperature in addition to up to three time and temperature set points from the SBT. The SBT in this document is designed to output network data every 30 seconds. The system is flexible in that the HVAC units read only the information required. For example, a heater may be designed to receive only two set points, in which case the third set point made available by the SBT would not be bound to the heater. (See Chapter 3 of the *NEURON C Programmers Guide* for details on network variables and binding.) This degree of flexibility allows independent suppliers to manufacture compatible equipment.

NETWORK INTERFACE

The SBT NEURON IC is interfaced to a 78 kbps twisted pair network via an MC75176BP EIA–485 Differential Transceiver IC. According to the EIA–485 standard this allows for up to 32 nodes per bus over a length of up to 1200 m (4000 ft). Note that each node in the HVAC system (e.g., heater, vent control, air conditioner) requires a transceiver with its NEURON IC. Also note that the network is accessed in the same manner regardless of the media interfaced to the NEURON IC’s general purpose communication port.

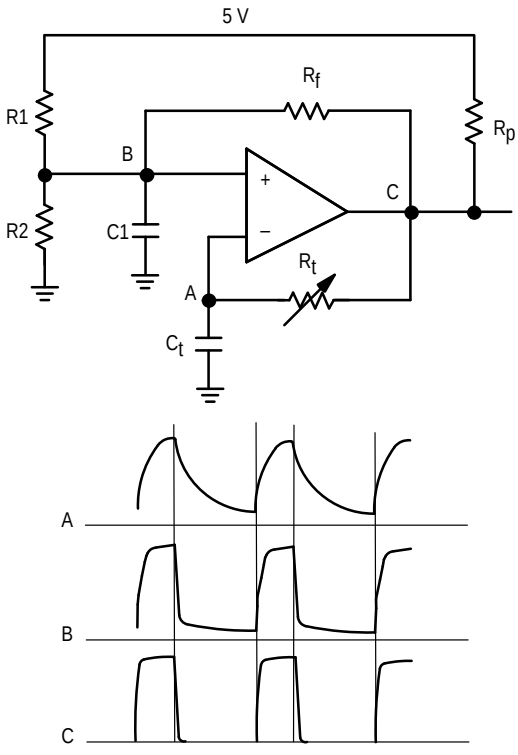

Figure 5. Temperature Circuit and Waveforms for SBT

CONCLUSIONS

In conclusion, the NEURON IC can adequately perform as a setback thermostat processor. Additionally, it provides HVAC systems with characteristics previously unavailable at a reasonable cost. For example, the owner of a two–story house has begrudgingly accepted the fact that one air conditioner with one thermostat will keep the first level about five degrees cooler than the second level, given the exorbitant cost of two separate A/C units. With a LONWORKS HVAC system, the network of communication (not only in the home, but in factories and other buildings as well) is in place for remote thermostats, zoning, vent controls, etc., each at the low cost of an additional node. The number of units communicating with a single NEURON IC SBT is virtually unlimited.

HVAC OEMs are now taking the time to compare their present system technologies to the solution offered by LONWORKS. Traditionally node costs have been the foremost concern of HVAC manufacturers, but today's OEM also realizes that installation, maintenance, and reliability can be key to the long term cost and value of a system. Although the SBT node requires an external display driver and a transceiver, external relays are not needed since all necessary data is passed to intelligent units which individually control themselves. Also, a distributed control network allows independent control of zones or units in the event of a wire break.

Finally, this document presents a rather simple SBT (functionally). Note that a system designer is not limited to EIA–485 twisted pair as a medium; the NEURON IC has a general purpose communication port which will interface to transceivers for virtually any medium (power line and RF are both popular). Also, keeping in mind that the code in Print–Out 1 consumes approximately 1.7K bytes of the NEURON IC's ROM, one can conclude that the available 42K of ROM may be used to give the SBT many more capabilities (i.e., system diagnostics, interfaces to ventilation control, safety features, etc.).

```
/***************** Print–Out 1. NEURON IC as a Setback Thermostat *****************/

#pragma enable_io_pullups
#pragma one_domain
#pragma num_addr_table_entries 8

IO_8 neurowire master select (IO_7) IO_to_LCD;
IO_0 output nibble IO_keypad_column;
IO_4 input bit IO_row0;
IO_5 input bit IO_row1;
IO_6 input bit IO_row2;
IO_7 input period IO_temp_in;

struct temp_time {
       unsigned int temp;
       unsigned int minutes;
       unsigned int hours;
};

struct unit_type {
       struct temp_time data_out;
       unit_states unit_data;
};

network output struct unit_type NV_ac_data;
network output struct unit_type NV_heat_data;
network output struct temp_time NV_sbt_out;
network output struct temp_time NV_ac_set1;
network output struct temp_time NV_ac_set2;
network output struct temp_time NV_ac_set3;
network output struct temp_time NV_heat_set1;
network output struct temp_time NV_heat_set2;
network output struct temp_time NV_heat_set3;

const char table [3] [4]         = {”789A”, ”456B”, ”1230”};
const unsigned int lcd_table[17] = {0,215,6,227,167,54,181.245,7,247,55,119,244,209,
                                    230,241,113};
const unsigned long r_table[50]  = {22592,21678,20973,20207,19469,18757,18072,17412,
                                    16776,16163,15573,15004,14456,13928,13420,
                                    12929,12457,12002,11564,11141,10734,10342,
                                    9965,9601,9250,8912,8587,8273,7971,7680,7399,
                                    7129,6869,6618,6376,6143,5919,5703,5494,5294,
                                    5100,4914,4734,4562,4395,4234,4080,3931,3787,
                                    3549};
```

```
typedef enum {ON,OFF,AUTO} unit_states;
unit_states ac_mode;
unit_states heater_mode;

struct temp_time sbt_data;
struct temp_time ac_1;
struct temp_time ac_2;
struct temp_time ac_3;
struct temp_time heat_1;
struct temp_time heat_2;
struct temp_time heat_3;

struct unit_type heater;
struct unit_type air_conditioner;

struct bcd digits;

unsigned int lcd_update;
unsigned int lcd_data[6];
unsigned int array_index[6];
unsigned int indicator;
unsigned int serial_out[6];
unsigned int row;
unsigned int col;
unsigned int key_input;
unsigned int i;
unsigned int prog_busy;
unsigned int temp_adder;
unsigned int num_bytes;
unsigned int second_byte;
unsigned int point_on;
unsigned int seconds;
unsigned int display_next;

unsigned long period_in;
unsigned long thermistor_value;
unsigned long value;

stimer display_timer;
stimer nv_timer;
mtimer read_timer;
mtimer temp_timer;


/*****************************************************************************/
/* This function will update the 6-digit LCD clock display which includes    */
/* time in hours and seconds and temperature in degrees fahrenheit. This     */
/* function is called every 1.0s.                                            */
/*****************************************************************************/


void update_clock (struct temp_time *disp_in) {
      for (i=0; i<6; i+=2) {
               value = *(unsigned int*)disp_in;   //point to next digit of display
               (unsigned int*)disp_in += 1;       //increment pointer
               bin2bcd(value,&digits);            //convert digit to BCD
               array_index[i+1] = digits.d5 + 1;  //store BCD values
               array_index[i] = digits.d6 + 1;
      }             //end for

      //use look-up table to encode BCD digits for LCD driver

      for (i=0; i<6; i++) {
               lcd_data[i] = lcd_table[array_index[i]];
               serial_out[i] = lcd_data[i];
      }             //end for
```

```
       //determine whether decimal point should be on or off

       if (point_on || (sbt_mode == DISPLAY)) {
                point_on = 0;
                serial_out[4] += 0x08;
       }              //end if
       else point_on = 1;
       io_out (IO_to_LCD,&serial_out,48);          //serial xmit to LCD driver
       read_timer = 984;                //set real time clock for another 1s
}              //end update_clock


/*****************************************************************************/
/* This function converts ASCII clock data to decimal and loads hours and    */
/* minutes into the address of the structure sent.                            */
/*****************************************************************************/


void clock_init (struct temp_time *data_in) {
       if (indicator) indicator = 1;               //use data indicator as index
       (unsigned int*)data_in += 1;                //point to minute field
       *(unsigned int*)data_in = array_index[indicator + 2] * 10 +
                array_index[indicator + 1] – 11;             //store minutes
       (unsigned int*)data_in += 1;                //point to hour field
       *(unsigned int*)data_in = array_index[indicator + 4] * 10 +
                array_index[indicator + 3] – 11;                    //store hours
}              //end clock_init


/*****************************************************************************/
/* This function converts ASCII temperature data to decimal and returns      */
/* the value.                                                                 */
/*****************************************************************************/


unsigned int temp_init () {
       return (array_index[3] * 10 + array_index[2] – 11);
}              //end temp_init


/************************************* reset event ************************************/


when (reset) {
       for (i=0; i<6; i++) serial_out[i] = 0xff;    //turn all segments on
       io_out (IO_to_LCD, &serial_out, 48);         //serial xmit to LCD
       seconds = 0;              //initialize real time clock to midnight
       sbt_mode = NORMAL;                //nromal display mode
       sbt_data.minutes = 0;
       sbt_data.hours = 0;
       nv_timer = 30;                    //30s timer for network xmission
       read_timer = 1000;                //1s real time clock timer
       temp_timer = 2000;                //2s temperature timer
}              //end when


/******************************** check for keypad depression ***************************/


when (io_changes (IO_row1) to 0)
when (io_changes (IO_row2) to 0)
when (io_changes (IO_row3) to 0) {
       delay (400);              //debounce
```

```
	//find row and column of pressed key
	for (col=0; col<4; col++) {
		io_out (keypad_column, ~(1<<col));
		key_input = ~((io_in (IO_row2) * 4) + (io_in (IO_row1) * 2) +
			(io_in (IO_row0)));
		for (row=0; row<3; row++) {
			if (key_input & (1<<row)) {
				array_index[0] = (unsigned int)table[row][col];
				goto jump1;
			}		//end if
		}		//end for
	}		//end for

	array_index[0] = 48;		//if key not found: assign null character
	jump1:

	// if programming temperature or time value
	if ((sbt_mode == TIME) || (sbt_mode == TEMP)) {
		lcd_update = 1;			//display update flag
		if (second_byte) {			//second byte indicates function
			second_byte = 0;			//clear flag
			indicator = array_index[0] – 48;		//ASCII to decimal
			if (indicator == 0) {			//program current time
				read_timer = 0;		//stop realtime clock
				num_bytes = 6;		//this function requires 6 bytes
			}		//end if
			if (indicator >= 4)			//program unit status
				num_bytes = 3;		//this function requires 3 bytes
		}		//end if
	}		//end if

	//if in display mode
	else if (sbt_mode == DISPLAY) {
		if (array_index[0] == 48) lcd_update = 1;		//toggle to next display
	}		//end else if

	// if in normal mode (first key touched)
	else {

		// if time or temperature key touched
		if ((array_index[0] >= 65)) {
			for (i=0; i<6; i++) lcd_data[i] = 0;		//clear out display
			if (array_index[0] == 65) {			//program time
				sbt_mode = TIME;
				num_bytes = 7;	//this function requires 7 bytes
			}		//end if
			else {		//program temperature
				sbt_mode = TEMP;
				num_bytes = 5;	//this function requires 5 bytes
			}		//end else
			lcd_update = 1;	//set display update flag
			prog_busy = 1;	//set busy flag to prevent clock function
			second_byte = 1;	//prepare for second byte
		}		//end if

		// if display key touched
		if (array_index[0] == 48) {
			sbt_mode = DISPLAY;
			lcd_update = 1;		//set display update flag
		}		//end if
	}		//end else
	io_out (keypad_column,0);		//clear all columns to prepare for next read
}		//end when
```

```
/************************** check for the display update flag **************************/

when (lcd_update) {

       //if programming time or temperature
       if ((sbt_mode == TIME) || (sbt_mode == TEMP)) {
                //shift display to the left
                for (i=5; i>0; i--) lcd_data[i] = lcd_data[i-1];
                if (array_index[0] < 58) array_index[0] -= 47;
                else array_index[0] -= 54;
                for (i=5; i>0; i--) array_index[i] = array_index[i-1];
                lcd_data[0] = lcd_table[array_index[0]];
                for (i=0; i<6; i++) serial_out[i] = lcd_data[i];
                io_out (IO_to_LCD,&serial_out,48);                    //update display

                // update sbt data after last programming byte has been entered
                if ((prog_busy++ >= num_bytes)) {
                       prog_busy = 0;                clear byte count
                       if (sbt_mode == TIME) {
                                  //update current time
                                  if (indicator == 0) clock_init (&sbt_data);
                                  //update ac status
                                  else if (indicator == 4) air_conditioner.unit_data = OFF;
                                  else if (indicator == 5) air_conditioner.unit_data = ON;
                                  else if (indicator == 6) air_conditioner.unit_data = AUTO;
                                  //update ac time
                                  else if (array_index[1] == 11) {
                                         if (indicator == 1) clock_init (&ac_1);
                                         else if (indicator == 2) clock_init (&ac_2);
                                         else if (indicator == 3) clock_init (&ac_3);
                                  }                 //end else if
                                  //update heater time
                                  else if (array_index[1] == 12) {
                                         if (indicator == 1) clock_init (&heat_1);
                                         else if (indicator == 2) clock_init (&heat_2);
                                         else if (indicator == 3) clock_init (&heat_3);
                                  }                 //end else if
                       }                end if
                       else {
                                  //update heater status
                                  if (indicator == 4) heater.unit_data = OFF;
                                  else if (indicator == 5) heater.unit_data = ON;
                                  else if (indicator == 6) heater.unit_data = AUTO;
                                  //update ac temperature
                                  else if (array_index[1] == 11) {
                                         if (indicator == 1) ac_1.temp = temp_init();
                                         else if (indicator == 2) ac_2.temp = temp_init();
                                         else if (indicator == 3) ac_3.temp = temp_init();
                                  }                 //end else if
                                  //update heater temperature
                                  else if (array_index[1] == 12) {
                                         if (indicator == 1) heat_1.temp = temp_init();
                                         else if (indicator == 2) heat_2.temp = temp_init();
                                         else if (indicator == 3) heat_3.temp = temp_init();
                                  }                 //end else if
                       }                //end else
                       sbt_mode = NORMAL;                  //return to normal mode
                       update_clock (&sbt_data);                  //display current time/temp
                }                //end if
       }                //end if
```

```
      //if in display mode
      else {
               display_timer = 5;                //5s timeout on any one display
               if (display_next++ > 6) display_next = 1;                    //wrap around
               //display 1st programmed ac value
               if (display_next == 1) update_clock (&ac_1);
               //display second programmed ac value
               else if (display_next == 2) update_clock (&ac_2);
               //display 3rd programmed ac value
               else if (display_next == 3) update_clock (&ac_3);
               //display 1st programmed heater value
               else if (display_next == 4) update_clock (&heat_1);
               //display 2nd programmed heater value
               else if (display_next == 5) update_clock (&heat_2);
               //display 3rd programmed heater value
               else if (display_next == 6) update_clock(&heat_3);
               //display current time/temp
               else {
                     update_clock (&sbt_data);
                     display_timer = 0;
                     sbt_mode = NORMAL;
               }               //end else
      }              //end else
      lcd_update = 0;
}              //end when

/***************************** check the real time clock timer ****************************/

when (timer_expires (read_timer)) {
      if (seconds++ > 58) {           //check for minute expiration
               seconds = 0;
               if (sbt_data.minutes++ > 58) {     //check for hour expiration
                     sbt_data.minutes = 0;
                     if (sbt_data.hours++ > 22)   //check for day expiration
                              sbt_data.hours = 0;
               }               //end if
      }              //end if
      if (sbt_mode == NORMAL)          //update display
               update_clock (&sbt_data);
      else read_timer = 984;           //set clock for one more second
}              //end when

/***************************** check for next temperature update ****************************/

when (timer_expires (temp_timer)) {
      thermistor_value = io_in (IO_temp_in) * 7;            //read frequency value
      temp_adder = 1;

      //look up temperature value in table and convert to fahrenheit
      while (temp_adder) {
               if (thermistor_value >= r_table[temp_adder]) {
                     if ((r_table[temp_adder – 1] – thermistor_value) <
                         (thermistor_value – r_table[temp_adder]))
                              temp_adder––;
                     sbt_data.temp = temp_adder * 3 / 5 * 3 + 32;
                     temp_adder = 0;
               }               //end if
               else temp_adder++;
               if (temp_adder > 50) temp_adder = 0;
      }              //end while
      temp_timer = 2000;               //read temperature every 2s
}              //end when
```

```
/********************** check for 5s timeout while in display mode ***********************/

when (timer_expires (display_timer)) {
       update_clock (&sbt_data);        //display current time/temp
       display_next = 0;
       sbt_mode = NORMAL;
}              //end when


/****************************** send out sbt data every 30s ****************************/

when (timer_expires (nv_timer)) {
       heater.data_out = sbt_data;
       air_conditioner.data_out = sbt_data;
       NV_heat_out = heater;                        //xmit current heater status
       NV_ac_out = air_conditioner;                 //xmit current ac status
       NV_ac_set1 = ac_1;               //xmit 1st ac value
       NV_ac_set2 = ac_2;               //xmit 2nd ac value
       NV_ac_set3 = ac_3;               //xmit 3rd ac value
       NV_heat_set1 = heat_1;           //xmit 1st heater value
       NV_heat_set2 = heat_2;           //xmit 2nd heater value
       NV_heat_set3 = heat_3;           //xmit 3rd heater value
}              //end when
```