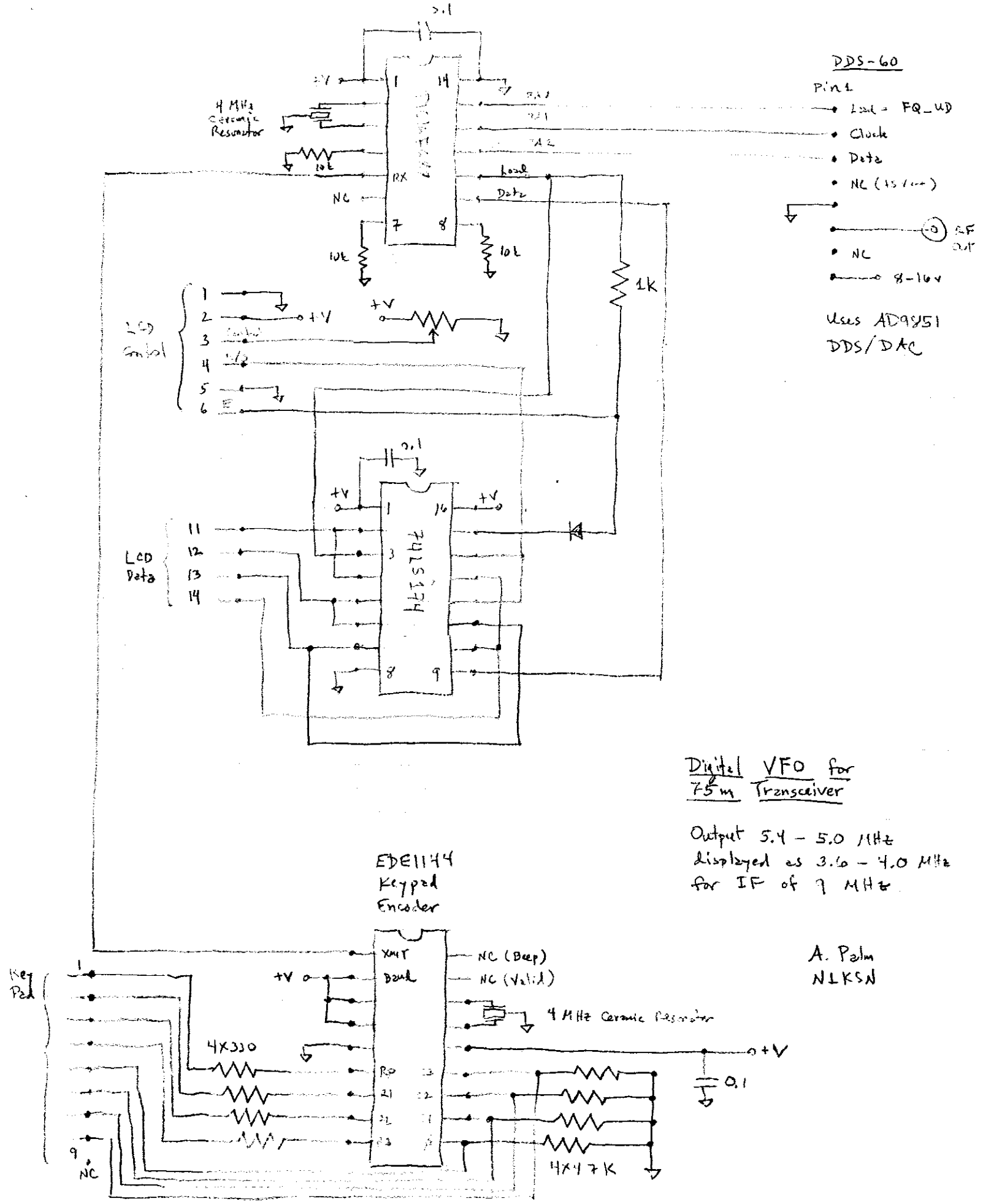
DDS-60
Pin 1
Load = FQ_UD
Clock
Data
NC (15 V out)
RF out
NC
8-16 V
Uses AD9851 DDS/DAC
4 MHz Ceramic Resonator
10k
1k
LCD Control
LCD Data
74LS374
EDE1144 Keypad Encoder
XMIT
Baud
NC (Beep)
NC (Valid)
4 MHz Ceramic Resonator
4X330
4X4.7K
0.1
Key Pad
NC
Digital VFO for 75m Transceiver
Output 5.4 - 5.0 MHz
displayed as 3.6 - 4.0 MHz
for IF of 9 MHz
A. Palm
N1KSN

```
/*****************************************************************
bcDDSVFO4.c - Control program for 5.0-5.4 MHz VFO using DDS-60.
              Displayed frequency is 3.6 to 4.0 MHz for 75m use.

Revision History:
V2 - The keypad is read using polling in the main loop.
V3 - Expanded keypad functions using 2ND key.
V4 - Added calibration entry operation

The DDS-60 is a PCB kit available from the American QRP Club,
except for the Analog Devices AD9851 chip, which must be
obtained separately.

The frequency in the AD9851 is controlled by sending a 40-bit
serial word.  The connection requires 3 wires and ground.

Keypad is 2x8, All Electronics CAT# KP-24.  KP-23 could be
substituted (4x4 format).  Pin 9 is not used.

The keypad is read via the PIC's USART connected to the RS-232
pin of an EDE1144 keypad encoder IC by e-lab/Digital Engineering,
Inc., All Electronics CAT# EDE-1144.  This encoder sents ASCII
characters '0', ..., '9', 'A', ...'F' for the 16 keys.

This program puts a Hitachi 44780-based LCD in 4 bit mode
using a two wire interface via a 74LS174 configured as a shift
register.  See Experiment 48 in "123 PIC Microcontroller
Experiments for the Evil Genius" by Myke Predko.

PIC16F688 pin connections:
RA0      - DDS-60 Load (Pin 1)
RA1      - DDS-60 Clock (Pin 2)
RA2      - DDS-60 Serial Data (Pin 3)
RA3      - Grounded thru 10k resistor (not used)
RA4:RA5  - Ceramic resonator at 4 MHz
RC0      - LCD data out
RC1      - LCD clock out
RC2:3    - Grounded thru 10k resistor (not used)
RC4      - USART TX (NC)
RC5      - USART RX

Keypad functions:

ENTER xxxxxx - Direct frequency entry.  Displays 3.xxx.xxx MHz.
               Min 3.600.000, Max 3.699.999.
CLEAR - Cancel function in progress
HELP n - Retrieve frequency in Memory n, n = 0 to 9
2ND HELP n - Store current frequency in Memory n
UP - Increment frequency digit at cursor position
DOWN - Decrement frequency digit at cursor position
2ND UP - Move cursor to left one digit
2ND DOWN - Move cursor to right one digit
2ND ENTER +xxxx - Enter freq calibration offset.  Must be
                  exactly 5 characters long, first a + or -
                  and with leading zeros present if needed.

Startup frequency is value stored in Memory 0.

Andrew Palm
2007.01.28

*****************************************************************/
/********************* includes *********************************/
#include <system.h>
#include <stdlib.h>

/********************* defines and pragmas **********************/

#pragma DATA _CONFIG, _FCMEN_OFF & _IESO_OFF & _BOD_OFF & \
    _CPD_OFF & _CP_OFF & _MCLRE_OFF & _PWRTE_ON & _WDT_OFF & \
    _HS_OSC
```

```
#pragma CLOCK_FREQ 4000000

// Calibration adjustment
#pragma DATA 0x2100, '-', '0', '1', '2', '6'
// Frequency memory locations
#pragma DATA 0x2105, '3', '9', '7', '7', '5', '0', '0'
#pragma DATA 0x210C, '3', '6', '0', '0', '0', '0', '0'
#pragma DATA 0x2113, '3', '6', '5', '0', '0', '0', '0'
#pragma DATA 0x211A, '3', '7', '0', '0', '0', '0', '0'
#pragma DATA 0x2121, '3', '7', '5', '0', '0', '0', '0'
#pragma DATA 0x2128, '3', '8', '0', '0', '0', '0', '0'
#pragma DATA 0x212F, '3', '8', '5', '0', '0', '0', '0'
#pragma DATA 0x2136, '3', '9', '0', '0', '0', '0', '0'
#pragma DATA 0x213D, '3', '9', '5', '0', '0', '0', '0'
#pragma DATA 0x2144, '3', '9', '9', '9', '9', '9', '0'

#define dds_load_pin porta.0   // Connections to DDS-60
#define dds_clock_pin porta.1
#define dds_data_pin porta.2

#define lcd_clock portc.1      // LCD data clock to shift reg
#define lcd_data_line portc.0  // LCD data line to shift reg

#define DENOM 0xABA95      // (180 x 10^6)/256 where DDS-60 ref
                           // freq is 180 MHz
#define IF_FREQ 0x895440   // IF freq of 9 MHz
#define MIN_100_KHZ 6      // Minimum for 100 KHz digit
#define FREQ_MAX 3999999   // Maximum frequency
#define FREQ_MIN 3600000   // Minimum frequency
#define CUR_MAX_POS 5      // Maximum cursor freq digit pos
#define CUR_MIN_POS 0      // Minimum cursor freq digit pos

/******************* declarations *****************************/
unsigned char dds_word[5];  // 40-bit word to control AD9851
unsigned long vfo_freq;     // Frequency in hex/binary
unsigned long freq_incr;    // Freq incr/decr for UP/DOWN keys

char current_freq[] = "3600000"; // Current frequency (ASCII)
char calib_freq[] = "+0000";     // Frequency calibration term
char new_freq[] = "0000000";     // Freq entry buffer
char new_cal[] = "+0000";        // Calibrate entry buffer
rom char *enter_freq = "3******";  // Freq entry template
rom char *put_mem_msg = "  Put Memory ";  // Store mem msg
rom char *get_mem_msg = "  Get Memory ";  // Retrieve mem msg
rom char *set_sign_msg = " UP=+, DOWN=-";  // Enter cal msg
char *start;  // Used only to allow use of function ltoa

unsigned char i;                // Utility index
unsigned char cntl_state;       // Main operation state index
unsigned char digit_count;      // Count digits entered when
                                //  using direct freq entry
unsigned char new_digit;        // Used to pass memory digit
unsigned char rx_char;          // Character from keypad IC
unsigned char kp_state;         // Keypad state index
unsigned char flag_2nd_key;     // Indicates if 2ND key pushed
unsigned char cursor_pos;       // Cursor digit position 0 to 5

/************************ tables ********************************/
//                      Returned    ASCII    Key
//                        Value      Sent    Face
rom char *KEYPAD_TABLE = { 0x01,   //  '0'     1
                           0x02,   //  '1'     2
                           0x03,   //  '2'     3
                           0xA0,   //  '3'  UP ARROW
                           0x04,   //  '4'     4
                           0x05,   //  '5'     5
                           0x06,   //  '6'     6
                           0xA1,   //  '7' DOWN ARROW
                           0x07,   //  '8'     7
                           0x08,   //  '9'     8
```

```
                                0x09,   //  'A'     9
                                0xA2,   //  'B'    2ND
                                0xA3,   //  'C'   CLEAR
                                0x00,   //  'D'     0
                                0xA4,   //  'E'    HELP
                                0xA5 }; //  'F'   ENTER

/************* functions and subroutines ***********************/
load_dds_word(void)
// Load 40-bit frequency control word into AD9851 DDS chip
{
  unsigned char i, j;

  dds_clock_pin = 0;
  dds_load_pin = 0;

  for (i=0; i<5; i++)
  {
    for (j=0; j<8; j++)
    {
      dds_data_pin = 0;
      if (test_bit(dds_word[i], j))
        dds_data_pin = 1;
      dds_clock_pin = 1;     // Toggle in bit on rising edge
      dds_clock_pin = 0;
    }
  }

  dds_load_pin = 1;         // Toggle in word on rising edge
  dds_load_pin = 0;

}  // end load_dds_word

calc_dds_word(void)
// Calculate DDS freq control word from ASCII rep of frequency.
// Control word is contained in 4 bytes dds_word[0-3] plus a
// fifth control byte dds_word[4] which is always 1.
// Formula for AD9851 is
//
//    Control word = f(MHz) x (2^32) / 180
//
// where 180 is the reference frequency, 6 x ref clock.
// This routine is will work for current_freq between 3.6 and
// 4 MHz.  current_freq is a string of the form "3xxxxxx".
//
// The calib_freq string is used to adjust the freq word for
// calibration.  It is of the form "-0000" where the leading
// character can be minus (-), plus (+) or blank.
//
// Makes use of BoostC function atol which converts an ascii
// numerical string to a long integer, plus 32-bit long integer
// arithmetic.
//
{
  unsigned long r;

  vfo_freq = (unsigned long) ( IF_FREQ - atol(current_freq) \
                               - atol(calib_freq) );

  dds_word[3] = (unsigned char) (vfo_freq / DENOM)  ;
  r = vfo_freq % DENOM;
  r = r * 0x100;

  dds_word[2] = (unsigned char) (r / DENOM);
  r = r % DENOM;
  r = r * 0x100;

  dds_word[1] = (unsigned char) (r / DENOM);
  r = r % DENOM;
  r = r * 0x100;
```

```
  dds_word[0] = (unsigned char) (r / DENOM);
  r = r % DENOM;
  r = r * 0x10;

  r = r / DENOM;  // Check to see if rounding up needed
  if (((unsigned char) r) > 0x7)
  {
    dds_word[0]++;
    if (dds_word[0] == 0)
    {
      dds_word[1]++;
      if (dds_word[1] == 0)
      {
        dds_word[2]++;
        if (dds_word[2]++ == 0)
          dds_word[3]++;
      }
    }
  }

  dds_word[4] = 0x01;

}  // end calc_dds_word

/*****************************************************************/
char eeprom_read(char addr)
{
  eeadr = addr;                    // Address to read
  clear_bit(eecon1, EEPGD);        // Point to data memory
  set_bit(eecon1, RD);             // EE read
  return eedata;                   // Return EE data
}  // end eeprom_read

eeprom_write(char addr, char data)
{
  bit gie_temp;
  clear_bit(pir1, EEIF);
  eeadr = addr;
  eedata = data;
  clear_bit(eecon1, EEPGD);
  set_bit(eecon1, WREN);
  gie_temp = intcon.7;             // Save GIE value
  clear_bit(intcon, GIE);
  eecon2 = 0x55;
  eecon2 = 0xAA;
  set_bit(eecon1, WR);
  while(!test_bit(pir1, EEIF));  // Wait for write completion
  clear_bit(pir1, EEIF);
  clear_bit(eecon1, WREN);
  intcon.7 = gie_temp;             // Restore original GIE value
}  // end eeprom_write

/*****************************************************************/
nybble_shift(char lcd_out, char rs)
// Shift Out a nybble
// lcd_out = 0bzzzzxxxx, 4 high bits zzzz are ignored,
//           xxxx = is nybble to send
// RS = value for RS pin, all but y = Bit 0 ignored
{
  unsigned char i;
  lcd_data_line = 0;
  for (i = 0; i < 6; i++)       // Clear the shift register
  {
    lcd_clock = 1;              // Toggle clock
    lcd_clock = 0;
  }

  lcd_out = lcd_out | (1 << 5) | ((rs & 1) * (1 << 4));
// Transform 0bzzzzxxxx to 0bzzlyxxxx where xxxx = nybble,
// y = RS bit
```

```
  for (i = 0; i < 6; i++)      // Shift 6 lowest bits lyxxxx out
  {
    if ((lcd_out & (1 << 5)) != 0)
      lcd_data_line = 1;       // Shift out Bit 5
    else
      lcd_data_line = 0;
    lcd_out = lcd_out << 1;    // Shift left for next bit out
    lcd_clock = 1;             // Clock bit into the S/R
    lcd_clock = 0;
  }

  nop();
  lcd_data_line = 1; // Clock nybble into LCD.  With S/R Q6 high due
  nop();             // to 1 bit, circuitry allows Data state to go to
  lcd_data_line = 0; // E pin of LCD.  If Q6 low, E held low thru diode.

}  // end nybble_shift

lcd_write(char lcd_data, char rs_value)  // Send Byte to LCD
{
  nybble_shift((lcd_data >> 4) & 0x0F, rs_value);
  nybble_shift(lcd_data & 0x0F, rs_value);  //  Shift out Byte
  if (((lcd_data & 0xFC) == 0) && (rs_value == 0))
    delay_ms(5);           //  Set delay interval longer for
  else                     // clear display and return home
    delay_100us(1);
}  // end lcd_write

lcd_init_2w(void)
// Initialize LCD with two wire setup.  First 4 commands
// don't bother going through lcd_write because 4 highest
// bits are all zeros.
{
  delay_ms(20);    //  Wait for LCD to power up

  nybble_shift(3, 0);       // Start initialization with reset
  delay_ms(5);
  nybble_shift(3, 0);       // Repeat reset command
  delay_100us(2);
  nybble_shift(3, 0);       // Repeat reset again
  delay_100us(2);

  nybble_shift(2, 0);       // Initialize LCD to 4 Bit Mode
  delay_100us(2);

  lcd_write(0b00101000, 0);  // LCD is 4 Bit I/F, 2 Line
  lcd_write(0b00000001, 0);  // Clear LCD
  lcd_write(0b00000110, 0);  // Move cursor after each character
  lcd_write(0b00001110, 0);  // Turn on LCD and enable cursor
//lcd_write(0b00001100, 0);  // Turn on LCD with disabled cursor

} // end init_lcd_2w

disp_freq(unsigned char* lcd_buffer)
// Display frequency in lcd_buffer on LCD.  lcd_buffer
// contains a string in the form "xxxxxxx" which is
// displayed as "  x.xxx.xxx MHz" on the LCD
{
  unsigned char k;

  lcd_write(0b10000000, 0);           // Start Line 1
  lcd_write(32, 1);                   // 2 leading spaces
  lcd_write(32, 1);

  for (k=0; k<7; k++)
  {
    lcd_write(lcd_buffer[k], 1);
    if (k == 0)
      lcd_write('.',1);
    if (k == 3)
      lcd_write('.',1);
```

```
    if (k == 6)
    {
      lcd_write(32, 1);
      lcd_write('M', 1);
      lcd_write('H', 1);
      lcd_write('z', 1);
    }
  }

} // end disp_freq

disp_cal(unsigned char* cal_buffer)
// Display calibration freq adjustment in cal_buffer
// on LCD.  cal_buffer contains a string in the form
// "+xxxx" which is displayed as "Cal: +xxxx" on the LCD
{
  unsigned char k;

  lcd_write(0b10000000, 0);           // Start Line 1
  lcd_write(32, 1);                   // 2 leading spaces
  lcd_write(32, 1);
  lcd_write('C', 1);
  lcd_write('a', 1);
  lcd_write('l', 1);
  lcd_write(':', 1);
  lcd_write(32, 1);
  for (k=0; k<5; k++)
    lcd_write(cal_buffer[k], 1);
  lcd_write(32, 1);
  lcd_write('H', 1);
  lcd_write('z', 1);
  lcd_write(32, 1);

} // end disp_cal

place_cursor(unsigned char cp)
// Put cursor below digit where UP and DOWN keys are active
//   cursor_pos = 0 is 1 Hz digit
//   cursor_pos = 5 is 100 KHz digit
// Address adjusted for display format "  3.xxx.xxx MHz"
{
  cp = (CUR_MAX_POS - cp) + 4;    // Cursor address ones digit
  if (cp > 6) cp++;       // Shift up 1 if right of dec. pt.
  cp = cp + 0x80;         // Add high bit for LCD command
  lcd_write(cp, 0);

} // end place_cursor

/****************** main procedure *****************************/
main()
{
  porta = 0;                  //  Initialize ports
  portc = 0;
  cmcon0 = 7;                 // Turn off comparators
  ansel = 0;                  // Turn off ADC
  trisa = 0b111000;
  trisc = 0b110000;

  lcd_init_2w();    // Initialize LCD with two-wire setup

// Initialize USART
  spbrgh = 0;               // Set up 9600 bps with 4 MHz clock
  spbrg = 25;
  clear_bit(baudctl, BRG16);
  set_bit(txsta, BRGH);
  clear_bit(txsta, SYNC);   // Async. mode
  set_bit(rcsta, SPEN);     // Enable serial ports
  set_bit(rcsta, CREN);     // Enable rx
  clear_bit(pir1, RCIF);    // Clear rx interrupt flag

// Retrieve startup freq from Memory 0
```

```
  for (i=0; i<7; i++)
    current_freq[i] = eeprom_read(i + 5);

// Retrieve freq calibration adjustment from EEPROM
  for (i=0; i<5; i++)
    calib_freq[i] = eeprom_read(i);

  calc_dds_word();      // Calculate freq word for AD9851

  load_dds_word();      // Load freq word
  load_dds_word();      // Load freq word again (needed on startup)

  disp_freq(current_freq);  // Display freq on Line 1 of LCD

  cursor_pos = 1;  // Initial cursor position at 10 Hz
  freq_incr = 10;
  place_cursor(cursor_pos);  // Put cursor under 10 Hz digit

  cntl_state = 0;      // Initialize main state machine
  flag_2nd_key = 0;    // Flag indicates 2ND key pushed

  for(;;)  // Main loop
  {
    if (test_bit(pir1, RCIF))  // Check for rx from keypad
    {
      if (!test_bit(rcsta, FERR) && !test_bit(rcsta, OERR))
      {
        rx_char = rcreg;    // Get character from USART
        // Convert it to a hex value for table lookup
        if (rx_char > 0x40) rx_char = rx_char - 0x41 + 10;
        else rx_char = rx_char - 0x30;

        if (rx_char <= 0x0F)  // Filter out invalid values
        {
          kp_state = KEYPAD_TABLE[rx_char];  // Get code

          switch (kp_state)  // Keypad state machine
          {
            case 0xA0:  // UP ARROW
              if (cntl_state == 7)
              {
                new_digit = '+';
                digit_count = 1;
              }
              else
              {
                if (flag_2nd_key)  // Shift cursor pos. to left
                {
                  flag_2nd_key = 0;
                  if (cursor_pos < CUR_MAX_POS)
                  {
                    cursor_pos++;
                    freq_incr = 10 * freq_incr;
                  }
                  lcd_write(0b11001111, 0);  // Last pos., Line 2
                  lcd_write(32, 1);          // Clear 2ND symbol
                  place_cursor(cursor_pos);
                }
                else              // Increment freq digit
                  cntl_state = 5;
              }
              break;
            case 0xA1:  // DOWN ARROW
              if (cntl_state == 7)
              {
                new_digit = '-';
                digit_count = 1;
              }
              else
              {
                if (flag_2nd_key)  // Shift cursor pos. to right
```

```
    {
      flag_2nd_key = 0;
      if (cursor_pos > CUR_MIN_POS)
      {
        cursor_pos--;
        freq_incr = freq_incr / 10;
      }
      lcd_write(0b11001111, 0);  // Last pos., Line 2
      lcd_write(32, 1);          // Clear 2ND symbol
      place_cursor(cursor_pos);
    }
    else              // Decrement freq digit
      cntl_state = 6;
  }
  break;
case 0xA2:  // 2ND key
  flag_2nd_key = 1;
  lcd_write(0b11001111, 0);  // Last pos., Line 2
  lcd_write('*', 1);         // Write 2ND active symbol
  place_cursor(cursor_pos);
  break;
case 0xA3:  // CLEAR - Cancel operation
  cntl_state = 2;
  digit_count = 0;
  flag_2nd_key = 0;
  break;
case 0xA4:  // HELP - Get/put freq in Memory n
  if (flag_2nd_key)  // Store in memory
  {
    cntl_state = 3;
    flag_2nd_key = 0;
    lcd_write(0b11001111, 0);  // Last pos., Line 2
    lcd_write(32, 1);          // Clear 2ND symbol
    place_cursor(cursor_pos);
  }
  else               // Retrieve from memory
  {
    cntl_state = 4;
  }
  digit_count = 0;
  break;
case 0xA5:  // ENTER - Initiate direct freq entry
  if (flag_2nd_key)
  {
    cntl_state = 7;
    flag_2nd_key = 0;
    lcd_write(0b11001111, 0);  // Last pos., Line 2
    lcd_write(32, 1);          // Clear 2ND symbol
  }
  else
  {
    cntl_state = 1;
  }
  digit_count = 0;
  break;
default:    // Digits 0 to 9
  if (cntl_state == 1)
  {
    new_digit = kp_state;
    digit_count++;
  }
  if ((cntl_state == 3) || (cntl_state == 4))
  {
    new_digit = kp_state;
    digit_count = 1;
  }
  if (cntl_state == 7)
    if (digit_count > 0)
    {
      new_digit = kp_state;
      digit_count++;
```

```
            }
          break;
        }
      }
    }
    else  // rx error
    {
      if (test_bit(rcsta, OERR))  // Overrun error
      {
        clear_bit(rcsta, CREN);  // Reset rx logic
        set_bit(rcsta, CREN);    // Enable rx again
      }
      if (test_bit(rcsta, FERR))  // Framing error
        rx_char = rcreg;          // Read rx buffer, discard
    }
  }

  switch (cntl_state)  // Main operations state machine
  {
    case 0:           // Do nothing
      break;
    case 1:           // Read in new freq digits
      if (digit_count == 0)
        for (i=0; i<7; i++)
          new_freq[i] = enter_freq[i];
      if (digit_count == 1)
        if (new_digit < MIN_100_KHZ)
          new_digit = MIN_100_KHZ;
      if ((digit_count > 0) && (digit_count < 7))
        new_freq[digit_count] = new_digit + '0';
      disp_freq(new_freq);
      if (digit_count == 6)  // Complete freq change
      {
        cntl_state = 0;
        for (i=0; i<7; i++)
          current_freq[i] = new_freq[i];
        calc_dds_word();
        load_dds_word();
        disp_freq(current_freq);   // Normal freq display
        lcd_write(0b11000000, 0);  // Clear 2nd line
        for (i=0; i<16; i++)
          lcd_write(32, 1);
        place_cursor(cursor_pos);
      }
      break;
    case 2:           // Restore normal freq display
      disp_freq(current_freq);
      lcd_write(0b11000000, 0);  // Clear 2nd line
      for (i=0; i<16; i++)
        lcd_write(32, 1);
      place_cursor(cursor_pos);
      cntl_state = 0;
      break;
    case 3:           // Store frequency
      lcd_write(0b11000000, 0);            //  Start line 2
      for (i=0; put_mem_msg[i]!=0; i++)  // Display store msg
        lcd_write(put_mem_msg[i], 1);
      if (digit_count == 1)
      {
        cntl_state = 0;
        digit_count = 0;
        lcd_write(new_digit + '0', 1);   // Display memory number
        delay_s(1);
        for (i=0; i<7; i++)              // Read memory
          eeprom_write(i + 5 + (7 * new_digit), current_freq[i]);
        lcd_write(0b11000000, 0);  // Clear 2nd line
        for (i=0; i<16; i++)
          lcd_write(32, 1);
        place_cursor(cursor_pos);
      }
      break;
```

```
case 4:          // Retrieve frequency
  lcd_write(0b11000000, 0);            //  Start line 2
  for (i=0; put_mem_msg[i]!=0; i++)  // Display retrieve msg
    lcd_write(get_mem_msg[i], 1);
  if (digit_count == 1)
  {
    cntl_state = 0;
    digit_count = 0;
    lcd_write(new_digit + '0', 1);   // Display memory number
    delay_s(1);
    for (i=0; i<7; i++)              // Write memory
      current_freq[i] = eeprom_read(i + 5 + (7 * new_digit));
    calc_dds_word();
    load_dds_word();
    disp_freq(current_freq);
    lcd_write(0b11000000, 0);  // Clear 2nd line
    for (i=0; i<16; i++)
      lcd_write(32, 1);
    place_cursor(cursor_pos);
  }
  break;
case 5:          // Increment freq
  vfo_freq = atol(current_freq);
  if (vfo_freq <= (FREQ_MAX - freq_incr))
    vfo_freq = vfo_freq + freq_incr;
  start = ltoa(vfo_freq, current_freq, 10);
  calc_dds_word();
  load_dds_word();
  disp_freq(current_freq);
  place_cursor(cursor_pos);
  cntl_state = 0;
  break;
case 6:          // Decrement freq
  vfo_freq = atol(current_freq);
  if (vfo_freq >= (FREQ_MIN + freq_incr))
    vfo_freq = vfo_freq - freq_incr;
  start = ltoa(vfo_freq, current_freq, 10);
  calc_dds_word();
  load_dds_word();
  disp_freq(current_freq);
  place_cursor(cursor_pos);
  cntl_state = 0;
  break;
case 7:          // Enter calibrate value
  if (digit_count == 0)
  {
    for (i=0; i<5; i++)
      new_cal[i] = calib_freq[i];
    lcd_write(0b11000000, 0);  // Msg for sign set
    for (i=0; set_sign_msg[i]!=0; i++)
      lcd_write(set_sign_msg[i], 1);
  }
  if (digit_count == 1)
  {
    if ((new_digit == '+') || (new_digit == '-'))
    {
      new_cal[digit_count - 1] = new_digit;
      lcd_write(0b11000000, 0);  // Clear 2nd line msg
      for (i=0; i<16; i++)
        lcd_write(32, 1);
    }
    else                          // Try again
      digit_count = 0;
  }
  if ((digit_count > 1) && (digit_count < 6))
    new_cal[digit_count - 1] = new_digit + '0';
  disp_cal(new_cal);
  if (digit_count == 5)  // Complete cal change
  {
    cntl_state = 0;
    for (i=0; i<5; i++)
```

```
          {
            calib_freq[i] = new_cal[i];
            eeprom_write(i, calib_freq[i]);
          }
          calc_dds_word();
          load_dds_word();
          disp_freq(current_freq);   // Normal freq display
          lcd_write(0b11000000, 0);  // Clear 2nd line
          for (i=0; i<16; i++)
            lcd_write(32, 1);
          place_cursor(cursor_pos);
        }
        break;
      default:        // Do nothing
        break;
    }

  }

}  //  end main
```