Appendix A: Source code listing for Adapter 3: Furnace Adapter

```
//*******************************************************************************
// Furnace Adapter 3
//
// Jed Margolin  3/14/2016.
//  MSP430G2211
//  Built with Code Composer Studio v6
//  Calibrate DCO at 1MHz to the 32.768 KHz crystal
//*******************************************************************************
#include <msp430.h>
//#include <msp430G2211.h>

// main.c
#define DELTA_1MHZ    244                 // 244 x 4096Hz = 999.4Hz
#define TIME 6
#define DCOUNT 8

void Set_DCO(unsigned int Delta);
void wait(unsigned int);

unsigned char caldata1;caldata2;              // Temp. storage for constants

int main(void)
{
int count;
unsigned char switches;

        WDTCTL = WDTPW | WDTHOLD;          // Stop watchdog timer

        wait(1000);                 // give crystal 500ms to stabilize

        P1SEL  |= BIT0;             // ACLK
//      P1SEL2 &= ~(BIT0);          // g2211 doesn't have it
        P1DIR  |= BIT0;

        P1SEL  |= BIT2;             // Out for testing
//      P1SEL2 &= ~(BIT2);          // g2211 doesn't have it
        P1DIR |= BIT2;

        P1DIR  &= ~(BIT3);          // In - Flame Good
        P1OUT |= BIT3;
        P1SEL  &= ~(BIT3);
//      P1SEL2 &= ~(BIT3);          // g2211 doesn't have it
        P1REN |= BIT3;              // pullup/pulldown enabled

        P1SEL  |= BIT4;             // Out for SMCLK so we know it's working
//      P1SEL2 &= ~(BIT4);          // g2211 doesn't have it
```

```
        P1DIR |= BIT4;

        P1DIR &= ~(BIT5);           // In - Diagnostics
        P1OUT |= BIT5;              // pullup
        P1SEL  &= ~(BIT5);
//      P1SEL2 &= ~(BIT5);          // g2211 doesn't have it
        P1REN |= BIT5;              // pullup/pulldown enabled

        P1SEL  &= ~(BIT6);          // Out - Backchannel Link
//      P1SEL2 &= ~(BIT6);          // g2211 doesn't have it
        P1DIR |= BIT6;              // Out   P1OUT |= BIT6;

        P1DIR &= ~(BIT7);           // In - 60Hz phase
        P1OUT |= BIT7;              // pullup
        P1SEL  &= ~(BIT7);
//      P1SEL2 &= ~(BIT7);          // g2211 doesn't have it
        P1REN |= BIT7;              // pullup/pulldown enabled

        switches = 0x00;

// Lock the DCO to the 32.768 KHz crystal - 1 MHz
        Set_DCO(DELTA_1MHZ);  // Set DCO and obtain constants
        caldata1 = DCOCTL;          // for debugging, two bytes at 0x200
        caldata2 = BCSCTL1;

// This is the Main Loop
        for(;;)
        {

// wait for one (negative 24VAC half-wave)
                count = DCOUNT;
                for(;;)
                {
                        switches = P1IN & BIT7;     // Read the pin

                        if(switches == BIT7)  {count--;}
                        else {count = DCOUNT;}      // reset the count

                        if(count == 0)  {break;}        // we have a one
                }

// wait for zero (positive 24VAC half-wave)
                count = DCOUNT;
                for(;;)
                {
                        switches = P1IN & BIT7;     // Read the pin

                        if(switches == 0)       {count--;}
                        else {count = DCOUNT;}      // reset the count
```

```
                        if(count == 0) {break;}         // we have a zero
                }

// This happens about 124us after 24VAC goes positive
// Now do the switches

// First switch input (from Diagnostic LED)  - First time period
                switches = P1IN & BIT5;                 // check the bit

                if(switches == 0)
                {
                        P1OUT &= ~(BIT6);               // First Pulse Out
                }
                else  { P1OUT |= BIT6; }
                wait(TIME);

// Now the second switch (from Flame Good LED) - Second time period
                switches = P1IN & BIT3;

                if(switches == 0)
                {
                        P1OUT &= ~(BIT6);               // Second Pulse Out
                }
                else  { P1OUT |= BIT6; }
                wait(TIME);

// clear BIT6
                P1OUT &= ~(BIT6);
        }  // end of main loop

        return(0);
}  // end main

//=============================================================================
void wait(unsigned int time)            // 500 us
{
   volatile unsigned int i,j;

   for(i=time; i>0; i--)
   {
        for(j=48; j>0; j--);{}
   }
        return;
}
//=============================================================================
void Set_DCO(unsigned int Delta)                // Set DCO to selected frequency
{
  unsigned int Compare, Oldcapture = 0;
```

```
  BCSCTL1 |= DIVA_3;                        // ACLK = LFXT1CLK/8
  TACCTL0 = CM_1 + CCIS_1 + CAP;            // CAP, ACLK
  TACTL = TASSEL_2 + MC_2 + TACLR;    // SMCLK, cont-mode, clear

  while (1)
  {
    while (!(CCIFG & TACCTL0));          // Wait until capture occured
    TACCTL0 &= ~CCIFG;                   // Capture occured, clear flag
    Compare = TACCR0;                    // Get current captured SMCLK
    Compare = Compare - Oldcapture;      // SMCLK difference
    Oldcapture = TACCR0;                 // Save current captured SMCLK

    if (Delta == Compare)
      break;                         // If equal, leave "while(1)"
    else if (Delta < Compare)
    {
      DCOCTL--;                      // DCO is too fast, slow it down
      if (DCOCTL == 0xFF)            // Did DCO roll under?
        if (BCSCTL1 & 0x0f)
          BCSCTL1--;                 // Select lower RSEL
    }
    else
    {
      DCOCTL++;                      // DCO is too slow, speed it up
      if (DCOCTL == 0x00)            // Did DCO roll over?
        if ((BCSCTL1 & 0x0f) != 0x0f)
          BCSCTL1++;                 // Sel higher RSEL
    }
  }
  TACCTL0 = 0;                       // Stop TACCR0
  TACTL = 0;                         // Stop Timer_A
  BCSCTL1 &= ~DIVA_3;                // ACLK is divide by 1
}
//=============================================================================
//=============================================================================
```

Appendix B: Source code listing for Adapter 3: Thermostat Adapter

```
//*******************************************************************************
// Thermostat Adapter 3
//
// Jed Margolin  3/14/2016.
//
//  MSP430G2211
//  Built with Code Composer Studio v6
//  Calibrate DCO at 1MHz to the 32.768 KHz crystal
//*******************************************************************************
#include <msp430.h>
//#include <msp430G2211.h>

// main.c
#define DELTA_1MHZ    244             // 244 x 4096Hz = 999.4Hz
#define DELTA_8MHZ    1953            // 1953 x 4096Hz = 7.99MHz
#define TIME 6
#define DCOUNT 8

void Set_DCO(unsigned int Delta);
void wait(unsigned int);

unsigned char caldata1, caldata2;     // Temp. storage for constants

int main(void)
{
int count;
unsigned char switches;

        WDTCTL = WDTPW | WDTHOLD;       // Stop watchdog timer

        wait(1000);                     // give crystal 500ms to stabilize

        P1SEL  |= BIT0;         // ACLK
//      P1SEL2 &= ~(BIT0);      // g2211 doesn't have it
        P1DIR  |= BIT0;

        P1SEL  &= ~(BIT2);      // Out for testing
//      P1SEL2 &= ~(BIT2);      // g2211 doesn't have it
        P1DIR |= BIT2;

        P1SEL  &= ~(BIT3);      // Out - Flame Good LED
//      P1SEL2 &= ~(BIT3);      // g2211 doesn't have it
        P1DIR |= BIT3;

        P1SEL  |= BIT4;         // Out for SMCLK so we know it's working
//      P1SEL2 &= ~(BIT4);      // g2211 doesn't have it
        P1DIR |= BIT4;
```

```
        P1SEL  &= ~(BIT5);          // Out - Diagnostics LED
//      P1SEL2 &= ~(BIT5);          // g2211 doesn't have it
        P1DIR |= BIT5;

        P1DIR &= ~(BIT6);           // In - Backchannel link
        P1OUT |= BIT6;              // pullup
        P1SEL  &= ~(BIT6);
//      P1SEL2 &= ~(BIT6);          // g2211 doesn't have it
        P1REN |= BIT6;              // pullup/pulldown enabled

        P1DIR &= ~(BIT7);           // In - 60Hz phase
        P1OUT |= BIT7;              // pullup
        P1SEL  &= ~(BIT7);
//      P1SEL2 &= ~(BIT7);          // g2211 doesn't have it
        P1REN |= BIT7;              // pullup/pulldown enabled

        switches = 0x00;

// Lock the DCO to the 32.768 KHz crystal - 1 MHz
        Set_DCO(DELTA_1MHZ);        // Set DCO and obtain constants
        caldata1 = DCOCTL;          // for debugging, two bytes at 0x200
        caldata2 = BCSCTL1;

// This is the Main Loop
        for(;;)
        {

// wait for one (negative 24VAC half-wave)
                count = DCOUNT;
                for(;;)
                {
                        switches = P1IN & BIT7;         // Read the pin

                        if(switches == BIT7)  {count--;}
                        else {count = DCOUNT;}          // reset the count

                        if(count == 0)  {break;}        // we have a one
                }

// wait for zero (positive 24VAC half-wave)
                count = DCOUNT;
                for(;;)
                {
                        switches = P1IN & BIT7;         // Read the pin

                        if(switches == 0)       {count--;}
                        else {count = DCOUNT;}          // reset the count
```

```
                if(count == 0)  {break;}                // we have a zero
            }

// We have sync
// Now read the datastream
// Strobe Test Flag
            P1OUT |= BIT2;                  // turn on strobe
            P1OUT &= ~(BIT2);               // turn off strobe

// wait half-bit time
            wait(3);

            // Read Bit 6 (G')
            switches = P1IN & BIT6;         // check the bit

            if(switches == 0)
            {
                P1OUT &= ~(BIT5);           // turn on LED1
            }
            else  { P1OUT |= BIT5; }        // turn off LED1

            // Strobe Test Flag
            P1OUT |= BIT2;                  // turn on strobe
            P1OUT &= ~(BIT2);               // turn off strobe

// Do second bit
            wait(6);                        // wait for middle of second bit

// Read Bit 6 (G')
            switches = P1IN & BIT6;         // check the bit

            if(switches == 0)
            {
                P1OUT &= ~(BIT3);           // turn on LED2
            }
            else  { P1OUT |= BIT3; }        // turn off LED2

// Strobe Test Flag
            P1OUT |= BIT2;                  // turn on strobe
            P1OUT &= ~(BIT2);               // turn off strobe

        }  // end main for

        return(0);

} // end main
// ===============================================
void wait(unsigned int time)            // 500 uS
{
```

```
  volatile unsigned int i,j;

  for(i=time; i>0; i--)
  {
       for(j=48; j>0; j--);{}
  }
       return;
}
// ===========================================
void Set_DCO(unsigned int Delta)              // Set DCO to selected frequency
{
 unsigned int Compare, Oldcapture = 0;

 BCSCTL1 |= DIVA_3;                           // ACLK = LFXT1CLK/8
 TACCTL0 = CM_1 + CCIS_1 + CAP;               // CAP, ACLK
 TACTL = TASSEL_2 + MC_2 + TACLR;             // SMCLK, cont-mode, clear

 while (1)
 {
  while (!(CCIFG & TACCTL0));                 // Wait until capture occured
  TACCTL0 &= ~CCIFG;                          // Capture occured, clear flag
  Compare = TACCR0;                           // Get current captured SMCLK
  Compare = Compare - Oldcapture;             // SMCLK difference
  Oldcapture = TACCR0;                        // Save current captured SMCLK

  if (Delta == Compare)
    break;                                    // If equal, leave "while(1)"
  else if (Delta < Compare)
  {
    DCOCTL--;                                 // DCO is too fast, slow it down
    if (DCOCTL == 0xFF)                       // Did DCO roll under?
      if (BCSCTL1 & 0x0f)
        BCSCTL1--;                            // Select lower RSEL
  }
  else
  {
    DCOCTL++;                                 // DCO is too slow, speed it up
    if (DCOCTL == 0x00)                       // Did DCO roll over?
      if ((BCSCTL1 & 0x0f) != 0x0f)
        BCSCTL1++;                            // Sel higher RSEL
  }
 }
 TACCTL0 = 0;                                 // Stop TACCR0
 TACTL = 0;                                   // Stop Timer_A
 BCSCTL1 &= ~DIVA_3;                          // ACLK is divide by 1
}
//=============================================================================
```

Appendix C: Source code listing for Adapter 4: Both Furnace Adapter and Thermostat Adapter

```
//*******************************************************************************
// Adapter 4 - Both
//   Calibrates DCO synced to 32.768 KHz crystal (1MHz)
//   Uses UART (UART uses ACLK from crystal)

// MSP430G2553

// Jed Margolin  2/27/2016.
//  Built with Code Composer Studio v6
//*******************************************************************************
********
// main.c

#include <msp430.h>
//#include <msp430G2553.h>

#define DELTA_1MHZ    244          // 244 x 4096Hz = 999.4Hz
#define DELTA_8MHZ    1953         // 1953 x 4096Hz = 7.99MHz
#define DELTA_12MHZ   2930         // 2930 x 4096Hz = 12.00MHz
#define DELTA_16MHZ   3906        // 3906 x 4096Hz = 15.99MHz
#define TIME 6
#define DCOUNT 8

void wait(unsigned int);
void wait2(void);
void uartsetup_4800(void);
void uart_init(void);
void uart_send_byte(unsigned char byte);
void Set_DCO(unsigned int Delta);

static volatile char rx_byte = '\0';
//=================================================
// main - uart - send bytes
int main(void)
{
        unsigned char caldata1, caldata2;               // Temp. storage for constants
        volatile unsigned int k = 0;
        unsigned char mbyte, temp;
        int count,n;
        unsigned char switches;

        WDTCTL = WDTPW + WDTHOLD;         // Stop watchdog timer

        wait(1000);                             // give the crystal time to stabilize

        P1SEL  |= BIT0;                         // ACLK
```

```
        P1SEL2 &= ~(BIT0);
        P1DIR  |= BIT0;

        P1SEL  |= BIT1;                 // RX In
        P1SEL2 |= BIT1;
        P1REN |= BIT1;                  // pullup/pulldown enabled
        P1OUT |= BIT1;
        P1DIR &= ~(BIT1);               // In

        P1SEL  |= BIT2;                 // TX Out
        P1SEL2 |= BIT2;
        P1DIR |= BIT2;

        P1SEL  &= ~(BIT3);              // Out - for testing
        P1SEL2 &= ~(BIT3);
        P1DIR |= BIT3;

        P1SEL  |= BIT4;                 // Out for SMCLK so we know it's working
        P1SEL2 &= ~(BIT4);
        P1DIR |= BIT4;

        P1SEL  &= ~(BIT5);              // In  1 = furnace, 0=thermostat
        P1SEL2 &= ~(BIT5);
        P1REN |= BIT5;                  // pullup/pulldown enabled
        P1OUT |= BIT5;                  // pullup
        P1DIR &= ~(BIT5);               // In

        P1DIR &= ~(BIT6);               // In
        P1OUT |= BIT6;                  // pullup
        P1SEL  &= ~(BIT6);
        P1SEL2 &= ~(BIT6);
        P1REN |= BIT6;                  // pullup/pulldown enabled

        P1DIR &= ~(BIT7);               // In - 60Hz phase
        P1OUT |= BIT7;                  // pullup
        P1SEL  &= ~(BIT7);
        P1SEL2 &= ~(BIT7);
        P1REN |= BIT7;                  // pullup/pulldown enabled

        switches = 0x00;

// Lock the DCO to the 32.768 KHz crystal - 1 MHz
          Set_DCO(DELTA_1MHZ);          // Set DCO and obtain constants
          caldata1 = DCOCTL;            // for debugging
          caldata2 = BCSCTL1;

/*
Port 2
```

On the Furnace Adapter P2.5, P2.4, P2.3 are sent to the Thermostat Adapter and are output on Thermostat P2.2, P2.1, P2.0

On the Thermostat Adapter P2.5, P2.4, P2.3 are sent to the Furnace Adapter and are output on Furnace P2.2, P2.1, P2.0

P2.5	In	Furnace=Diagnostic LED	Thermostat=Call for Heat
P2.4	In	Furnace=Flame Good LED	Thermostat=Call for Cool
P2.3	In	Furnace= not used	Thermostat=Auto/Fan
P2.2	Out	Furnace= Heat Relay	Thermostat=Diagnostic LED
P2.1	Out	Furnace=Cool Relay	Thermostat= Flame Good LED
P2.0	Out	Furnace= Fan Relay	Thermostat= not used

```
P2 In
        P2SEL  &= ~(BIT5|BIT4|BIT3);
        P2SEL2 &= ~(BIT5|BIT4|BIT3);
        P2REN  |= BIT5|BIT4|BIT3;          // pullup/pulldown enabled
        P2OUT  |= BIT5|BIT4|BIT3;          // pullup
        P2DIR  &= ~(BIT5|BIT4|BIT3);               // In

P2 Out
        P2SEL  &= ~(BIT2|BIT1|BIT0);
        P2SEL2 &= ~(BIT2|BIT1|BIT0);
        P2DIR  |=  (BIT2|BIT1|BIT0);               // Out
*/

// Initialize
        if(P1IN & BIT5) {P2OUT &= ~(BIT2|BIT1|BIT0);}       // furnace (relays) - reset to low
        else {P2OUT |= (BIT2|BIT1|BIT0);}          // thermostat (LEDs) - reset to high

        uartsetup_4800();
        uart_init();                               // no interrupts for either RX or TX

// Main Loop
        while (1)                       // main loop
        {
// send a byte
                P1OUT |= BIT3;                     // sync for 'scope
                P1OUT &= ~(BIT3);

                mbyte = 0;

                if(P2IN & BIT5)  {mbyte |= BIT0;}
                if(P2IN & BIT4)  {mbyte |= BIT1;}
                if(P2IN & BIT3)  {mbyte |= BIT2;}

//              mbyte = 0x05;                // for testing
                uart_send_byte(mbyte);
                wait2();
```

```
// wait for our byte to be echoed
// At 4800 Baud it takes 2.1 ms to send a character so we will wait for the character
// to get sent and received.
                rx_byte = 0;
                for(k=17; k>0; k--)
                {
                        if(IFG2 & UCA0RXIFG)
                        {
                                rx_byte = UCA0RXBUF;            // Get the received character
                                break;
                        }
                        wait2();
                }

                if(rx_byte != mbyte)
                {
                // we had a data collision
                        P1OUT |= BIT3;                  // flag the error for 'scope
                        if(P1IN & BIT5) {wait(6);}      // furnace waits for time 1
                        else {wait(10);}                // thermostat waits for time 2
                        P1OUT &= ~(BIT3);
                        wait(2);
                }

                wait(2);
// end of waiting for and checking the echoed byte

// Check for incoming byte but timeout after 19 ms
        for(k=17; k>0; k--)
        {
                if(IFG2 & UCA0RXIFG)
                {
                        rx_byte = UCA0RXBUF;                    // Get the received character
                        break;
                }
                else {wait(2);}
        }       // end looking for received byte

        if(k != 0)                              // we received a byte from the thermostat/furnace
        {
                if(rx_byte & BIT0)  {P2OUT |= BIT2;}    // Call for Heat/Diagnostic LED
                else {P2OUT &= ~BIT2;}

                if(rx_byte & BIT1)  {P2OUT |= BIT1;}    // Call for Cooling/Flame Good LED
                else {P2OUT &= ~BIT1;}

                if(rx_byte & BIT2)  {P2OUT |= BIT0;}    // Auto-Fan/not used
                else {P2OUT &= ~BIT0;}
```

```
        }

        if(P1IN & BIT5) {wait(1);}          // furnace
        else {wait(3);}                     // thermostat

        }  // end main loop
}  // end main
//=================================
void wait(unsigned int time)            // 500 us
{
   volatile unsigned int i,j;

   for(i=time; i>0; i--)
   {
        for(j=48; j>0; j--);
        {
        }
   }

   return;
}
//=============================================================
// Low-Frequency Baud Rate Mode
// Read chapter 15.3.10 of msp430x2xxx user's guide for baud rate generation
void uartsetup_4800(void)
{
        P3SEL = 0x30;                   // P3.4,5 = USCI_A0 TXD/RXD, No need to set direction
        UCA0CTL1 |= UCSSEL_1;   // CLK = ACLK
        UCA0BR0 = 0x06;                 // 6 = INT(6.83), 32.768KHz/4800 = 6.8
        UCA0BR1 = 0x00;                                                 //
        UCA0MCTL = UCBRS2 + UCBRS1 + UCBRS0;  // Modulation UCBRSx = 7 =
round(0.83 * 8)
        UCA0CTL1 &= (~UCSWRST);                                 // ** Initialize USCI state
machine
}

void uart_init(void)
{
        IE2 &= ~UCA0RXIE;       // default: Disable USCI_A0 RX interrupt
        IE2 &= ~UCA0TXIE;           // default: Disable USCI_A0 TX interrupt
}

// Sends a single byte out through UART
void uart_send_byte(unsigned char byte)
{
        while (!(IFG2&UCA0TXIFG));      // USCI_A0 TX buffer ready?
        UCA0TXBUF = byte;               // TX -> RXed character
}
```

```
// USCI A0/B0 Transmit ISR
#pragma vector=USCIAB0TX_VECTOR
__interrupt void USCI0TX_ISR(void)
{
        IE2 &= ~UCA0TXIE;                       // Disable USCI_A0 TX interrupt
}

//=========================================================
void wait2()
{
volatile unsigned int i;

        for(i=0; i<48; i++){}
        return;
}
//=========================================================
// This function came from Texas Instruments program msp430g2xx3_dco_flashcal.c
void Set_DCO(unsigned int Delta)          // Set DCO to selected frequency
{
 unsigned int Compare, Oldcapture = 0;

 BCSCTL1 |= DIVA_3;                     // ACLK = LFXT1CLK/8
 TACCTL0 = CM_1 + CCIS_1 + CAP;          // CAP, ACLK
 TACTL = TASSEL_2 + MC_2 + TACLR;        // SMCLK, cont-mode, clear

 while (1)
 {
  while (!(CCIFG & TACCTL0));        // Wait until capture occured
  TACCTL0 &= ~CCIFG;                 // Capture occured, clear flag
  Compare = TACCR0;                  // Get current captured SMCLK
  Compare = Compare - Oldcapture;    // SMCLK difference
  Oldcapture = TACCR0;               // Save current captured SMCLK

  if (Delta == Compare)
    break;                           // If equal, leave "while(1)"
  else if (Delta < Compare)
  {
    DCOCTL--;                        // DCO is too fast, slow it down
    if (DCOCTL == 0xFF)              // Did DCO roll under?
      if (BCSCTL1 & 0x0f)
        BCSCTL1--;                   // Select lower RSEL
  }
  else
  {
    DCOCTL++;                        // DCO is too slow, speed it up
    if (DCOCTL == 0x00)              // Did DCO roll over?
      if ((BCSCTL1 & 0x0f) != 0x0f)
        BCSCTL1++;                   // Sel higher RSEL
  }
```

```
  }
  TACCTL0 = 0;                        // Stop TACCR0
  TACTL = 0;                          // Stop Timer_A
  BCSCTL1 &= ~DIVA_3;                 // ACLK is divide by 1
}
//=====================================================================
```