ECET 491Senior Design Project Title:
Wind Speed Data Logger

Department of Computer, Electrical, and Information Technology
Indiana University-Purdue University Fort Wayne
Date: 02/19/2012

Student: Chuck Craft
Faculty Advisor: Professor Paul I-Hai Lin
Class Instructor: Professor Paul I-Hai Lin

TO: Professor Paul I-Hai Lin

FROM: Chuck Craft

DATE: 02/19/2012

SUBJECT: ECET 491 Design Report #2

Purpose

The purpose of this report is to provide feedback on the overall progress of the Wind Speed Data Logger project.

Problem Review/Discussion

As you may recall from our last discussion my project hardware consists of commercial off the shelf (COTS) devices including an anemometer and three evaluation modules. The anemometer selected for the project has a reed switch which opens and closes once per rotation of the wind cups. This one pulse per second is equivalent to 2.5 miles per hour of wind speed. The pulsed signal is transferred to an eZ430-RF2500 evaluation module using a 25 foot cable that is permanently attached to the anemometer. The eZ430-RF2500 module supplied by Texas Instruments includes a MPS430 microcontroller connected to a RF2500 packet transceiver using a SPI interface. The receive side of the data link is provided by an identical eZ430-RF2500 module which is used to receive anemometer data and pass this information to the main processing module. The main processing module is an AVR32 EVK100 evaluation kit from Atmel and will perform all database and user interface operations. The block diagram is included as figure 1 for reference.

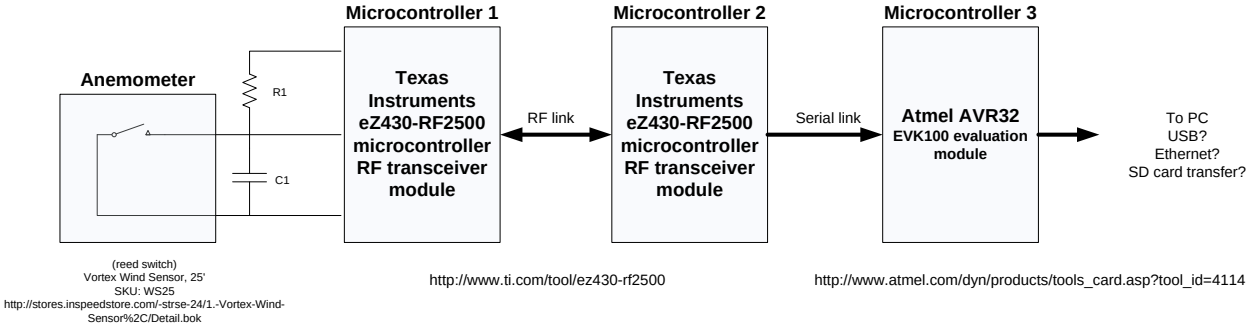

Figure 1

The focus of my effort for this reporting period was to complete the sensor link from anemometer through microcontroller 2. This task involved final testing of the anemometer signal to verify the elimination of false triggers due to transmission line impedance mismatch. The testing then progressed to developing code for the MSP430 microcontroller to measure the period of anemometer rotations. The final task of this reporting period was to transfer data between microcontroller 1 and microcontroller 2 and verify that the wind speed reported matched the measured anemometer rotational speed. Test data are included in the following sections and the main program listings for microcontrollers 1 and 2 are included in appendix A.

Completed/Remaining Tasks

1. Anemometer testing (100% complete)
2. Anemometer and microcontroller 1 module integration (100% complete)
3. Microcontroller 1 to microcontroller 2 integration (100% complete)
4. Wind speed information message verified at microcontroller 2 (100% complete)
5. Microcontroller 2 to microcontroller 3 integration (not started)
6. Microcontroller 3 code development (not started)
    a. User interface
    b. Memory structure
    c. Real time clock
    d. Display
    e. Serial link to microcontroller 2

Anemometer Testing

The testing consisted of constructing the circuit shown in figure 2 and spinning the anemometer at different speeds. The period of the anemometer rotations were measured using an oscilloscope and the readings compared to the timer values captured by microcontroller 1. A time delay of 24us was added to the microcontroller input to eliminate false triggers. The component values for the termination were selected from common values and the time delay was calculated as

$T_d = R_1 * C_1$ 12,000 * 0.000000002 = 0.000024
$R_1 = 12{,}000\ ohms$
$C_1 = 0.002uf$

The circuit in figure 2 was used to measure the anemometer signal integrity. Figure 3 and figure 4 show the input signal before and after termination.

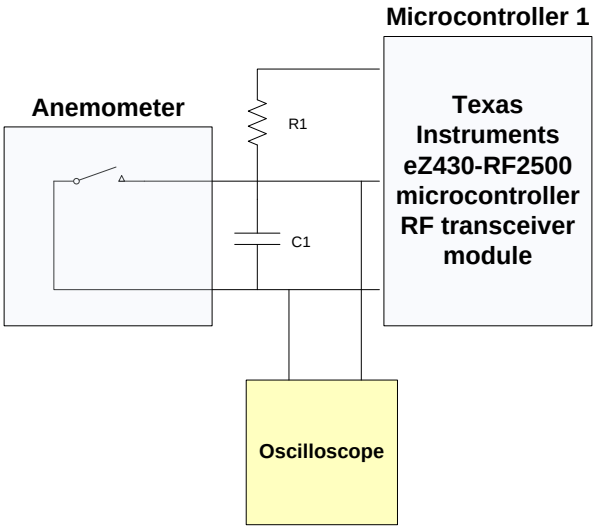

Figure 2

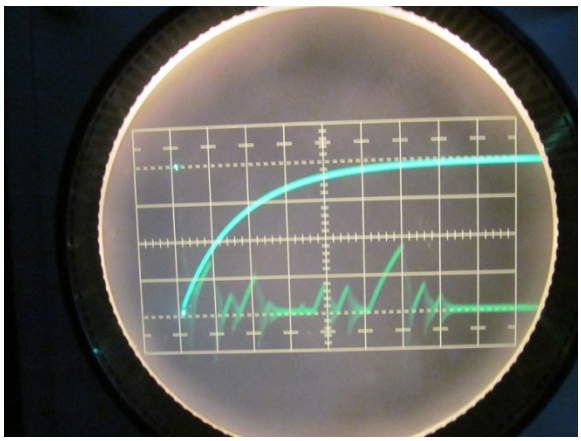

Figure 3 (no termination, 5us/div)

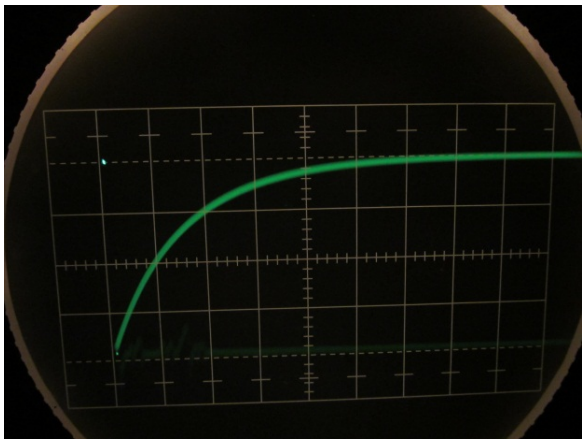

Figure 4 (terminated, 10us/div)

The next step in the testing process was to integrate the anemometer with microcontroller 1 and capture counter sample in local memory.

Microcontroller 1 development/integration

Microcontroller 1 code development began with an example program provided by Texas Instruments. The example program was modified to remove unnecessary code and move all existing delay functions from timer B to a CPU cycle based delay. New functions were created using timer B and a capture pin interrupt was enabled to record the timer B value at the rising edge of the anemometer signal. A Schmitt trigger was enabled on the input pin to prevent false triggers due to the slow signal rise time. The timer input was also set for synchronous operation to prevent timer glitches.

A calibrated 1MHz internal clock divided by 8 was selected for the timer clock, providing a resolution of 8 us ± 1%. The timer resolution required to meet the ±1 mph accuracy requirement is defined as follows:

Anemometer resolution: 1Hz = 2.5 mph => 0.4Hz = 1 mph
Worst case wind speed accuracy requirement (max speed ± error) => 50 mph ± 1 mph
Minimum period: 50 mph (upper limit) = (50 * 0.4) Hz = 20 Hz => 1/20 = 50 ms
Tolerance (±1 mph): 51 mph = 49 ms => 50 ms – 49 ms = 1 ms
Minimum period resolution requirement = 50 ms ± 1 ms

The timer clock period is 125 times faster than the period required for minimum accuracy requirements. The substantial oversampling provides adequate time for the microcontroller to capture and store the timer value without inducing unacceptable error.

$$oversample\ rate = \frac{minimum\ anemometer\ period}{timer\ B\ clock\ period} = \frac{1\ ms}{8\ us} = 125$$

Microcontroller 1 operation begins at power-up with an initialization routine to set default radio address parameters and initial operating settings for the RF link. Figure 5 is a flow diagram detailing microcontroller 1 operation.

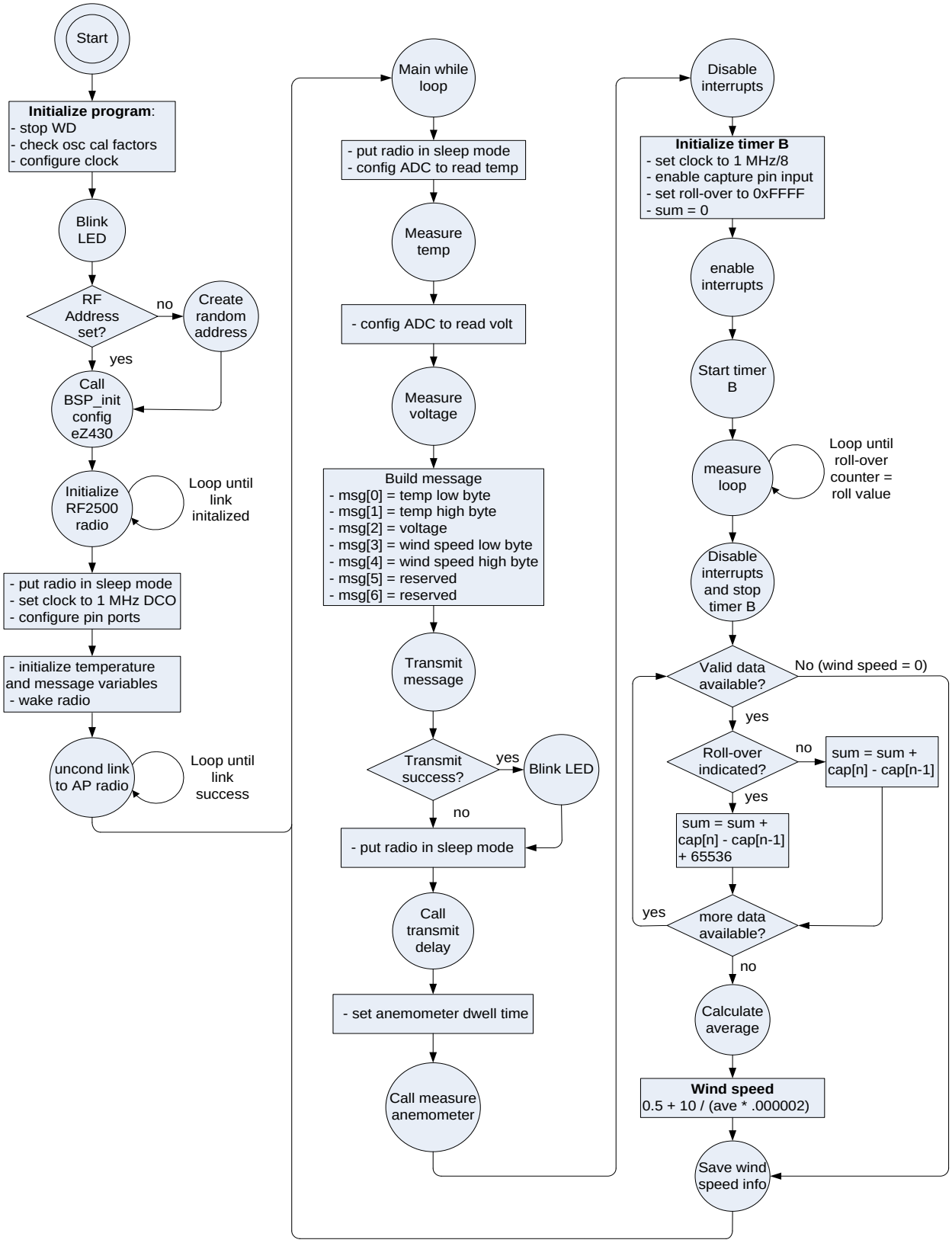
Start
Initialize program:
- stop WD
- check osc cal factors
- configure clock
Blink LED
RF Address set?
no
Create random address
yes
Call BSP_init config eZ430
Initialize RF2500 radio
Loop until link initalized
- put radio in sleep mode
- set clock to 1 MHz DCO
- configure pin ports
- initialize temperature and message variables
- wake radio
uncond link to AP radio
Loop until link success
Main while loop
- put radio in sleep mode
- config ADC to read temp
Measure temp
- config ADC to read volt
Measure voltage
Build message
- msg[0] = temp low byte
- msg[1] = temp high byte
- msg[2] = voltage
- msg[3] = wind speed low byte
- msg[4] = wind speed high byte
- msg[5] = reserved
- msg[6] = reserved
Transmit message
Transmit success?
yes
Blink LED
no
- put radio in sleep mode
Call transmit delay
- set anemometer dwell time
Call measure anemometer
Disable interrupts
Initialize timer B
- set clock to 1 MHz/8
- enable capture pin input
- set roll-over to 0xFFFF
- sum = 0
enable interrupts
Start timer B
measure loop
Loop until roll-over counter = roll value
Disable interrupts and stop timer B
Valid data available?
No (wind speed = 0)
yes
Roll-over indicated?
no
sum = sum + cap[n] - cap[n-1]
yes
sum = sum + cap[n] - cap[n-1] + 65536
yes
more data available?
no
Calculate average
Wind speed
0.5 + 10 / (ave * .000002)
Save wind speed info

Figure 5

Two interrupt service routines (ISR) perform timer value capture and roll-over tracking functions. The capture ISR moves timer B capture register contents to a capture data array, stores roll-over event information, and increments the array index and data available counters. Timer B roll-over ISR increments the roll-over counter and clears the interrupt. Figure 6 is the flow diagram for the capture ISR while figure 7 depicts the roll-over ISR.

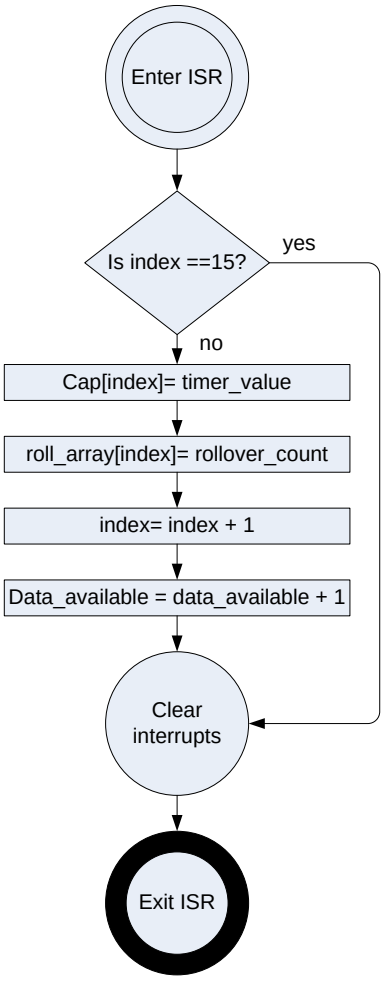

Figure 6

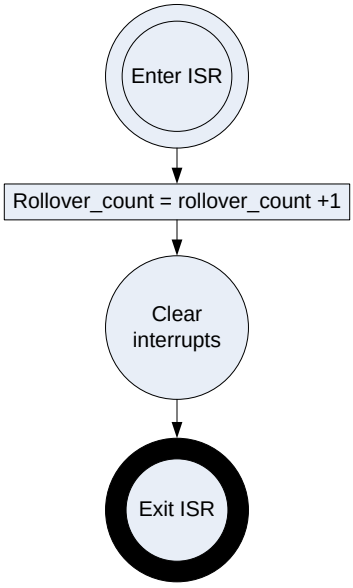

Figure 7

Microcontroller 1 performs board initialization and configures the RF2500 radio on the circuit board. The initialization routine continues with the configuration of input pins and clocks. The main clock is a 1 MHz calibrated internal oscillator that provides a 1 us CPU instruction cycle time. The next phase is to establish communication with the access point on the radio network. The access point listens for new radios to join the network and will respond to the end point when a valid message is received. The two radios exchange information at this point to establish the data link. The network link setup sequence is shown in figure 8.

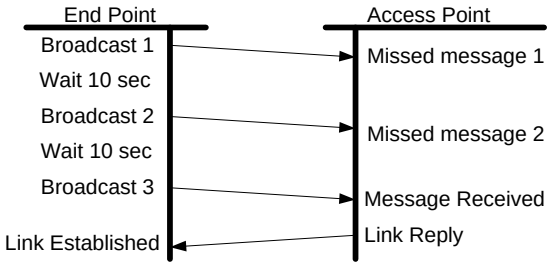

Figure 8

The program enters the main program loop after the radio network is established. The main uses the internal ADC to measure a temperature sensor and the power supply voltage on the circuit board. A second loop uses timer B to measure the delay between rising edges on the anemometer signal. The timer is also used to set the timing between transmissions by counting the number of times the counter rolls-over. The 16-bit timer is clocked at an 8 us rate to deliver a roll-over indication at the rate indicated below.

$$65536 * 8us = 524.288\, ms$$

The code is designed to allow for a dwell time selection at compile time that is based on multiples of dwell time. The current implementation is set for a dwell multiple of 2 which provides a sample window of approximately 1.0485 seconds. The sample window is the amount of time that timer B is monitoring the anemometer input signal. This is considered to be the minimum dwell time allowable to meet the minimum speed measurement requirement of 5 mph. The anemometer will only have two revolutions in one second at 5 mph and the time difference between rising edges is used to determine period, requiring at least 2 revolutions to measure speed. Timer B counter values are recorded in an array at every anemometer rising edge. Information concerning the roll-over status of the timer is also captured at this time. The data collection process completes when the roll-over count is equal to the dwell multiplier setting. Data collection is performed by 2 interrupt routines with one routine monitoring the anemometer pin and the other the timer B roll-over events.

Post processing begins with clearing the sum and average variables. The data available variable is checked to determine the amount of valid data contained in the capture array. There must be a minimum of 2 samples in the array to perform post processing; otherwise the wind speed is reported as zero. A loop based on the data available variable is used to accumulate the valid difference between adjacent timer samples. The samples are qualified to make sure that timer roll-over events are properly compensated for to avoid corrupting the summation process. The valid sample are accumulated (there is an upper limit of 16 samples to prevent memory overflow) and an average is calculated using a divisor variable determined by the loop. Checks are in place throughout the process to prevent any mathematical problems such as memory overflows or divide by zero errors. The average count value is converted to wind speed in mph with 1 decimal point resolution and rounding applied using the following equations.

$$f = \frac{1}{(average\ count\ value * 8\ us)} = 2.5 * mph$$

$$mph = \frac{f}{2.5} = \frac{\frac{1}{(average\ count\ value * 8\ us)}}{2.5} = \frac{1}{(average\ count\ value * 20\ us)}$$

The calculated value is multiplied by 10, 0.5 is added for rounding, and the fractional part is truncated to create an integer value representing mph in xx.x format. The final equation is shown below.

$$mph = integer(0.5 + \frac{10}{(average\ count\ value * 20\ us)})$$

The computed wind speed is placed in a shared variable to allow main loop access. The main loop builds a message array using the temperature, voltage, and wind speed data collected. The message format is listed in table 1.

Message array	**Function**
Msg[0]	Temperature lower byte
Msg[1]	Temperature upper byte
Msg[2]	Voltage
Msg[3]	Wind speed lower byte
Msg[4]	Wind speed upper byte
Msg[5]	Reserved
Msg[6]	Reserved

Table 1

The message array is transferred to the radio interface function where it is transmitted to the access point microcontroller.

Testing was performed to verify the accuracy of anemometer measurements by spinning the anemometer with an external wind source. The anemometer signal was monitored using an oscilloscope to manually measure the period of pulses created as the anemometer spins. Microcontroller 1 was set for a roll-over count of 2 providing a sample time of ~1 second. The captured data from microcontroller 1 was transmitted to microcontroller 2 where it was converted to serial format and transferred to a laptop computer over a USB port for display. The laptop computer used a simple terminal monitor program to display the message traffic as it was received. The circuit detailed in figure 9 was used for testing and message traffic was captured to a log file on the laptop. Final integration test data are included after the next section detailing microcontroller 2.

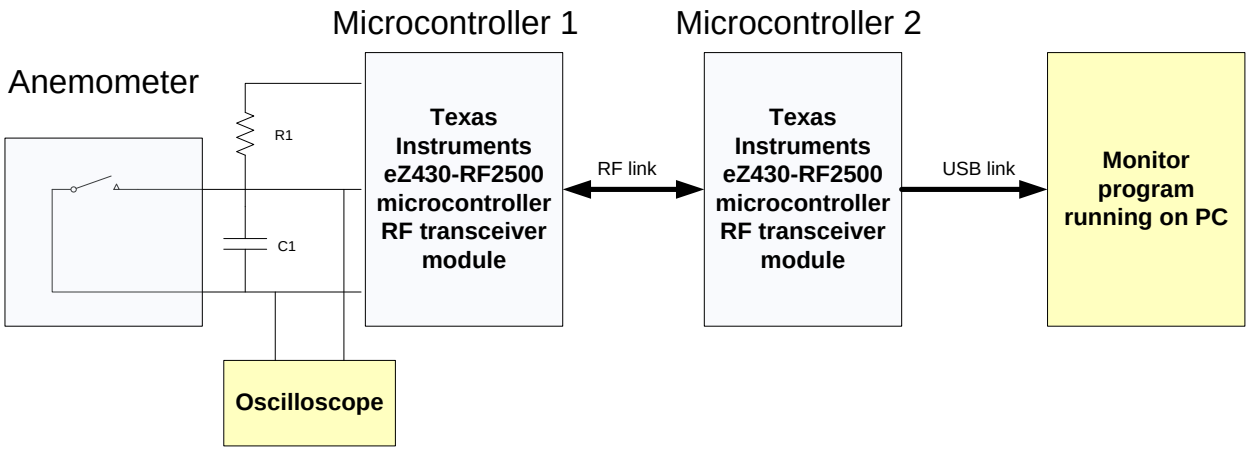
Microcontroller 1
Microcontroller 2
Anemometer
R1
C1
Texas Instruments eZ430-RF2500 microcontroller RF transceiver module
RF link
Texas Instruments eZ430-RF2500 microcontroller RF transceiver module
USB link
Monitor program running on PC
Oscilloscope

Figure 9

Appendix A
Microcontroller 1 program listing

```
//*****************************************************************************
// THIS PROGRAM IS PROVIDED "AS IS". TI MAKES NO WARRANTIES OR
// REPRESENTATIONS, EITHER EXPRESS, IMPLIED OR STATUTORY,
// INCLUDING ANY IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS
// FOR A PARTICULAR PURPOSE, LACK OF VIRUSES, ACCURACY OR
// COMPLETENESS OF RESPONSES, RESULTS AND LACK OF NEGLIGENCE.
// TI DISCLAIMS ANY WARRANTY OF TITLE, QUIET ENJOYMENT, QUIET
// POSSESSION, AND NON-INFRINGEMENT OF ANY THIRD PARTY
// INTELLECTUAL PROPERTY RIGHTS WITH REGARD TO THE PROGRAM OR
// YOUR USE OF THE PROGRAM.
//
// IN NO EVENT SHALL TI BE LIABLE FOR ANY SPECIAL, INCIDENTAL,
// CONSEQUENTIAL OR INDIRECT DAMAGES, HOWEVER CAUSED, ON ANY
// THEORY OF LIABILITY AND WHETHER OR NOT TI HAS BEEN ADVISED
// OF THE POSSIBILITY OF SUCH DAMAGES, ARISING IN ANY WAY OUT
// OF THIS AGREEMENT, THE PROGRAM, OR YOUR USE OF THE PROGRAM.
// EXCLUDED DAMAGES INCLUDE, BUT ARE NOT LIMITED TO, COST OF
// REMOVAL OR REINSTALLATION, COMPUTER TIME, LABOR COSTS, LOSS
// OF GOODWILL, LOSS OF PROFITS, LOSS OF SAVINGS, OR LOSS OF
// USE OR INTERRUPTION OF BUSINESS. IN NO EVENT WILL TI'S
// AGGREGATE LIABILITY UNDER THIS AGREEMENT OR ARISING OUT OF
// YOUR USE OF THE PROGRAM EXCEED FIVE HUNDRED DOLLARS
// (U.S.$500).k
//
// Unless otherwise stated, the Program written and copyrighted
// by Texas Instruments is distributed as "freeware".  You may,
// only under TI's copyright in the Program, use and modify the
// Program without any charge or restriction.  You may
// distribute to third parties, provided that you transfer a
// copy of this license to the third party and the third party
// agrees to these terms by its first use of the Program. You
// must reproduce the copyright notice and any other legend of
// ownership on each copy or partial copy, of the Program.
//
// You acknowledge and agree that the Program contains
// copyrighted material, trade secrets and other TI proprietary
// information and is protected by copyright laws,
// international copyright treaties, and trade secret laws, as
// well as other intellectual property laws.  To protect TI's
// rights in the Program, you agree not to decompile, reverse
// engineer, disassemble or otherwise translate any object code
// versions of the Program to a human-readable form.  You agree
// that in no event will you alter, remove or destroy any
```

```
// copyright notice included in the Program.  TI reserves all
// rights not specifically granted under this license. Except
// as specifically provided herein, nothing in this agreement
// shall be construed as conferring by implication, estoppel,
// or otherwise, upon you, any license or other right under any
// TI patents, copyrights or trade secrets.
//
// You may not use the Program in non-TI devices.
//
//******************************************************************************
//  eZ430-RF2500 Temperature Sensor End Device using Cymbet Solar Energy
//  Harvesting Board
//
//  Description:
//     This is the End Device software for the eZ430-RF2500-SEH Temperature
//     Sensing demo when hooked up to a Cymbet solar Energy Harvester board.
//
//     The Energy Harvester End Device (EHED) will join the traditional
//     Access Point (AP).  The EHED was optimized to reduce active time
//     especially during start up.
//
//   W. Goh
//   Version 1.5
//   Texas Instruments, Inc
//   March 2009
//   Built with IAR Embedded Workbench Version: 4.11B
//   Built with Code Composer Essentials Version: 3.1 build 3.2.3.6.4
//******************************************************************************
// Change Log:
//******************************************************************************
// Version:  1.5 using SimpliciTI ver 1.06
// Comments: Fixed an un-initialized bug inside SimpliciTI
//          Removed unnecessary port initialization in code
// Version:  1.4 using simpliciTI ver 1.06
// Comments: Fixed several bugs.
//          Added blinking LED on power-up
//          Application files now compiles on both IAR and CCE
// Version:  1.3 using Simpliciti ver 1.06
// Comments: Added count battery used count fields
//          Added number_transmits counts up and down
//          Added check if battery charged for 1 hour
// Version:  1.0
// Comments: Initial Release date
//******************************************************************************
```

```
//*****************************************************************************/
/
//*****************************************************************************/
/
// This program has been extensively modified to provide an anemometer
// input and calculate wind speed for the Wind Speed Data Logger (WSDL)
// senior design project.
//
// Chuck Craft
// Senior, IPFW BSEET program
// February 2012
// Version: 1.0
//
//*****************************************************************************/
/
// Change Log:
//*****************************************************************************/
/
// Version:  1.0
// Comments: Initial Release date
//*****************************************************************************/
/

#include "bsp.h"
#include "mrfi.h"
#include "bsp_leds.h"
#include "bsp_buttons.h"
#include "nwk_types.h"
#include "nwk_api.h"
#include "nwk_frame.h"
#include "nwk.h"

#include "msp430x22x4.h"
#include "vlo_rand.h"

#define WakeupPeriod      15000       // ~10 sec (=15000/(12000/8))
#define a_d_wakeup_time   4500      // ~3 sec
#define TXPeriod          7500          // ~5 sec  (=7500/(12000/8))
#define delay_time        500           // led delay time
#define debounce_time     750         // key debounce

//Timer count for time between transmit
#define sec1              1500              // ~1 sec
#define sec2              2610
#define sec5              7500              // ~5 sec  (=7500/(12000/8))
#define sec10             15000             // ~10 sec
```

```
#define sec20           30000          // ~20 swec
#define sec40           60000          // ~40 sec
#define sec30_2          43000         // ~30sec 2 min?
#define sec30_4          50434         // ~30sec 4 min?
#define one_hour         5400000

#define port_delay       10            // 6ms - 1.5 msec

#define status_one       1
#define status_two       2
#define status_three     3
#define status_four      4
#define status_five      5
#define status_six       6

#define timer_state_1sec  0
#define timer_state_1     1
#define timer_state_2     2
#define timer_state_3     3
#define timer_state_4     4
#define timer_state_5     5
#define timer_state_6     6

#define run_voltage      29             // Minimum voltage to execute 2.9V
#define ad_check_voltage  29
#define key_down_count    12            // # times to check if button is
                                        // still button pressed

#define battery_time_test 174           // 3 min count at 10 sec for testing
#define running_on_battery 100          // Tells GUI that it is running on
                                        // battery

#defineON             1
#define OFF            0

unsigned int timer_state;
unsigned char change_mode;
unsigned int battery_ready = 0;
unsigned int in_delay = 0;
char status = 0;
unsigned int battery_full_flag = 0;
unsigned long battery_full_timer = 0;

//*************************************************************
// WSDL specific variables
unsigned int data_avail;        // number of samples captured
```

```
unsigned int rollover_count;       // number of timer B rollovers for total dwell
unsigned int capture_array[16];  // RAM array for captures
unsigned int roll_array[16];       // track timer B roll-overs for each sample
unsigned char index = 0;           // index for sample RAM
unsigned int delay_count;          // number of roll-overs to execute dwell time
unsigned int wind_speed;          // mph = f(Hz)/0.4
                                   // (1/(average* 0.000008))/0.4
//************************************************************

//unsigned int number_transmits;

void linkTo(void);
void StatusBlink_led1(int BlinkCount);
void StatusBlink_led2(int BlinkCount);
void status_indicator(char status, int status_led);
void delay(unsigned int BlinkCount);
void button_still_pressed(void);
unsigned int get_voltage(void);
void transmit_time_delay(void);
void display_mode(void);
//void check_bat_full(void);
void createRandomAddress(void);

//************************************************************
// WSDL specific functions
void sleep(unsigned int count); // simple delay to replace original
                        // delay using timer B
void check_anemometer(unsigned int RollCount); // new function
//************************************************************

void main (void)
{
  addr_t lAddr;
  char *Flash_Addr;

  WDTCTL = WDTPW + WDTHOLD;              // Stop WDT

  P1DIR |= 0x03;                                    // Set P1.0,1 Output
  if( CALBC1_1MHZ == 0xFF && CALDCO_1MHZ == 0xFF &&
     CALBC1_8MHZ == 0xFF && CALDCO_8MHZ == 0xFF )// Do not run if cal values
  {                                                 // are erased and set LEDs ON
    P1OUT |= 0x03;                                 // Set P1.0,1 High
    __bis_SR_register(LPM4_bits);                  // Enter LPM4 if Cal missing
  }
```

```
// Blink LED for startup feedback
  P1OUT |= 0x03;                          // Set P1.0,1 High
  __delay_cycles(10000);
  P1OUT &= ~0x03;                         // Set P1.0,1 Low

  Flash_Addr = (char *)0x10F0;            // RF Address = 0x10F0
  if( Flash_Addr[0] == 0xFF &&            // Check if device Address is missing
      Flash_Addr[1] == 0xFF &&
      Flash_Addr[2] == 0xFF &&
      Flash_Addr[3] == 0xFF )
  {
    createRandomAddress();                // Create Random device address at
  }                                       // initial startup if missing
  lAddr.addr[0] = Flash_Addr[0];
  lAddr.addr[1] = Flash_Addr[1];
  lAddr.addr[2] = Flash_Addr[2];
  lAddr.addr[3] = Flash_Addr[3];
  SMPL_Ioctl(IOCTL_OBJ_ADDR, IOCTL_ACT_SET, &lAddr);

  BSP_Init();                             // Initialize eZ430 hardware

  BCSCTL3 |= LFXT1S_2;                    // LFXT1 = ACLK = VLO

  status = status_four;                   // Set status to 4

  // Initialize SimpliciTI
  while(SMPL_NO_JOIN == SMPL_Init((uint8_t (*)(linkID_t))0))
  {
    delay(sec10);
  }

  // Put radio to sleep
  SMPL_Ioctl( IOCTL_OBJ_RADIO, IOCTL_ACT_RADIO_SLEEP, "" );

  BCSCTL1 = CALBC1_1MHZ;                  // Set DCO = 1MHz
  DCOCTL = CALDCO_1MHZ;

  // SimpliciTI will change port pin settings as well
  P1DIR = 0xFB;                           // P1.2 (button) = input
  P1OUT = 0x04;                           // P1.2 pullup
  P1REN |= 0x04;                          // P1.2 pullup
  P1IE |= 0x04;                           // P1.3 interrupt enabled
  P1IES |= 0x04;                          // P1.3 Hi/lo edge
  P1IFG &= ~0x04;                         // P1.3 IFG cleared
  P2DIR = 0x2E;
  P2REN |= 0x01;
```

```
  P2OUT = 0x01;
  P3DIR |= 0xD0;                        // port 3 set after initilization
  P3OUT &= ~0x30;                       // set up port 3
  P3REN |= 0x20;                        // Enable Pull-Down Res for /Charge
//*************************************************************
// WSDL pin changes
  P4DIR = 0xEF;                         // P4.4 input
  P4REN |= 0x10;                        // enable pullup
  P4SEL |= 0x10;                        // select P4.4
  ADC10AE1 = 0x00;                      // select schmitt trigger
//*************************************************************
  P4OUT = 0x00;

  timer_state = timer_state_1;          // set timer state to 2 ~ 10 sec
  change_mode = 10;                     // Default GUI display mode set to
                                        // 10 sec

  SMPL_Ioctl( IOCTL_OBJ_RADIO, IOCTL_ACT_RADIO_AWAKE, "" );
  // unconditional link to AP which is listening due to successful join.
  linkTo();
}

/*****************************************************************************
 * @fn          linkTo
 *****************************************************************************/
void linkTo(void)
{
  linkID_t linkID1;
  uint8_t  msg[7];
  unsigned int *tempOffset;                     // Initialize temperature offset
  tempOffset = (unsigned int *)0x10F4;      // coefficient

  // keep trying to link
  while (SMPL_SUCCESS != SMPL_Link(&linkID1))
  {
    delay(sec10);
  }

  // put radio to sleep once a successfull connection has been established
  SMPL_Ioctl( IOCTL_OBJ_RADIO, IOCTL_ACT_RADIO_SLEEP, "" );

  while(1)
  {
    volatile long temp;
    int degC, volt;
    int results[2];
```

```
// If battery charging, go back to sleep  (if P3.5 = 1, Sleep)
P3REN &= ~0x20;                     // turn off pulldown resistor
delay(port_delay);

// Measure Temperature
ADC10CTL1 = INCH_10 + ADC10DIV_4;      // Temp Sensor ADC10CLK/5
ADC10CTL0 = SREF_1 + ADC10SHT_3 + REFON + ADC10ON + ADC10IE + ADC10SR;
__delay_cycles(350);                                // delay to allow reference to settle
ADC10CTL0 |= ENC + ADC10SC;                 // Sampling and conversion start
__bis_SR_register(LPM0_bits + GIE);            // LPM0 with interrupts enabled
results[0] = ADC10MEM;
ADC10CTL0 &= ~ENC;

// Measure Battery Voltage
ADC10CTL1 = INCH_11;                                // AVcc/2
ADC10CTL0 = SREF_1 + ADC10SHT_3 + REFON + ADC10ON + ADC10IE + REF2_5V;
__delay_cycles(350);                               // delay to allow reference to settle
ADC10CTL0 |= ENC + ADC10SC;                 // Sampling and conversion start
__bis_SR_register(LPM0_bits + GIE);            // LPM0 with interrupts enabled
results[1] = ADC10MEM;
ADC10CTL0 &= ~ENC;
ADC10CTL0 &= ~(REFON + ADC10ON);    // turn off A/D to save power

// oC = ((A10/1024)*1500mV)-986mV)*1/3.55mV = A10*423/1024 - 278
// the temperature is transmitted as an integer where 32.1 = 321
// hence 4230 instead of 423
temp = results[0];
degC = ((temp - 673) * 4230) / 1024;
if( *tempOffset != 0xFFFF )
{
  degC += *tempOffset;
}

/* message format,  UB = upper Byte, LB = lower Byte
-------------------------------------------------------------------
|degC LB | degC UB | volt | Wind Speed LB | Wind Speed UB | ? | ? |
-------------|-------------|------|---------------------|---------------------|---|---|
     0              1        2              3                    4           5   6
*/

// Wake radio-up
SMPL_Ioctl( IOCTL_OBJ_RADIO, IOCTL_ACT_RADIO_AWAKE, "" );

temp = results[1];
volt = (temp*25)/512;
```

```
    msg[0] = degC&0xFF;
    msg[1] = (degC>>8)&0xFF;
    msg[2] = volt;
    msg[3] = wind_speed&0xFF;
    msg[4] = (wind_speed>>8)&0xFF;
    msg[5] = 0;
    msg[6] = 0;

    // Send message
    if (SMPL_SUCCESS == SMPL_Send(linkID1, msg, sizeof(msg)))
    {
      delay(port_delay);
      status_indicator(status_one, 1);      // blink LED for successful transfer
    }

    SMPL_Ioctl( IOCTL_OBJ_RADIO, IOCTL_ACT_RADIO_SLEEP, "" );
    transmit_time_delay();                  // sleep time between transmits
  }
}

/*****************************************************************************
**
* BEGHDR
*
* NAME:createRandomAddress()
*
* DESCRIPTION: generate random address
*****************************************************************************
*/
void createRandomAddress()
{
  unsigned int rand, rand2;
  char *Flash_Addr;
  Flash_Addr = (char *)0x10F0;

  do
  {
    rand = TI_getRandomIntegerFromVLO();   // first byte can not be 0x00 of 0xFF
  }
  while( (rand & 0xFF00)==0xFF00 || (rand & 0xFF00)==0x0000 );
  rand2 = TI_getRandomIntegerFromVLO();

  BCSCTL1 = CALBC1_1MHZ;                  // Set DCO to 1MHz
  DCOCTL = CALDCO_1MHZ;
  FCTL2 = FWKEY + FSSEL0 + FN1;           // MCLK/3 for Flash Timing Generator
  FCTL3 = FWKEY + LOCKA;                  // Clear LOCK & LOCKA bits
```

```
  FCTL1 = FWKEY + WRT;                  // Set WRT bit for write operation

  Flash_Addr[0]=(rand>>8) & 0xFF;
  Flash_Addr[1]=rand & 0xFF;
  Flash_Addr[2]=(rand2>>8) & 0xFF;
  Flash_Addr[3]=rand2 & 0xFF;

  FCTL1 = FWKEY;                        // Clear WRT bit
  FCTL3 = FWKEY + LOCKA + LOCK;         // Set LOCK & LOCKA bit
}

/*****************************************************************************
* BEGHDR
* Function:    void StatusBlink_led1(int BlinkCount)
* DESCRIPTION: Blinks LED 1 - Red based on specified delay
* INPUTS:      BlinkCount
* PROCESSING:  Turns on and off the RED LED with specified blink time
* OUTPUTS:     VOID
*****************************************************************************/
void StatusBlink_led1(int BlinkCount)
{
  BSP_TURN_ON_LED1();
  delay(BlinkCount);
  BSP_TURN_OFF_LED1();
}

/*****************************************************************************
* BEGHDR
* Function:    void StatusBlink_led1(int BlinkCount)
* DESCRIPTION: Blinks LED 1 - Green based on specified delay
* INPUTS:      BlinkCount
* PROCESSING:  Turns on and off the Green LED with specified blink time
* OUTPUTS:     VOID
*****************************************************************************/
void StatusBlink_led2(int BlinkCount)
{
  BSP_TURN_ON_LED2();
  delay(BlinkCount);
  BSP_TURN_OFF_LED2();
}
/*****************************************************************************
* CEC
* Function:    void delay(unsigned int BlinkCount)
* DESCRIPTION: Creates simple delay counter
*
* INPUTS:      BlinkCount
```

```
* PROCESSING:  Delay length of time of BlinkCount
* OUTPUTS:     VOID
*******************************************************************************/
void delay(unsigned int BlinkCount)
{
  volatile unsigned int i = 0;

     while(i++ < BlinkCount)
                  {
                           sleep(300);
       }
}

/*******************************************************************************
* BEGHDR
* Function:    void status_indicator(char status , int status_led)
* DESCRIPTION: This can be useful to blink the LED to indicate where the
*              program is executing for debugging purposes. It blinks the red or
*              green LED the number of times in status. For example, status_five
*              blinks the LED 5 times.
* INPUTS:      status, status_led
* PROCESSING:  Blinks the red or green led the number of times in status and the
*              correct led in status_led
* OUTPUTS:     VOID
*******************************************************************************/
void status_indicator(char status , int status_led)
{
  volatile unsigned int i = 0;
  switch (status)
  {
    case status_one:
      if (status_led == 1)
        StatusBlink_led1(15);
      if (status_led == 2)
        StatusBlink_led2(15);
      break;

    case status_two:
      if (status_led == 1)
      {
        StatusBlink_led1(15);
        delay(delay_time);
        StatusBlink_led1(15);
      }
      if (status_led==2)
      {
```

```
    StatusBlink_led2(15);
    delay(delay_time);
    StatusBlink_led2(15);
   }
   break;

 case status_three:
   if(status_led == 1)
   {
    for(i=0 ; i < (status-1) ; i++)
    {
     StatusBlink_led1(15);
     delay(delay_time);
    }
    StatusBlink_led1(15);
   }
   if(status_led == 2)
   {
    for(i=0 ; i < (status-1) ; i++)
    {
     StatusBlink_led2(15);
     delay(delay_time);
    }
    StatusBlink_led2(15);
   }
   break;

 case status_four:
   if(status_led == 1)
   {
    for(i=0 ; i < (status-1) ; i++)
    {
     StatusBlink_led1(15);
     delay(delay_time);
    }
    StatusBlink_led1(15);
   }
   if(status_led == 2)
   {
    for(i=0 ; i < (status-1) ; i++)
    {
     StatusBlink_led2(15);
     delay(delay_time);
    }
    StatusBlink_led2(15);
   }
```

```
    break;

   case status_five:
    if(status_led == 1)
    {
     for(i=0 ; i < (status-1) ; i++)
     {
      StatusBlink_led1(15);
      delay(delay_time);
     }
     StatusBlink_led1(15);
    }
    if(status_led == 2)
    {
     for(i=0 ; i < (status-1) ; i++)
     {
      StatusBlink_led2(15);
      delay(delay_time);
     }
     StatusBlink_led2(15);
    }
    break;
   default:
    break;
  }
}

/*************************************************************************
* BEGHDR
* Function:    unsigned int get_voltage(void)
* DESCRIPTION: Get battery voltage with A/D
* INPUTS:      void
* PROCESSING:  Read battery voltage from ADC10 and returns the value
* OUTPUTS:     Battery voltage from A/D
*************************************************************************/
unsigned int get_voltage(void)
{
  unsigned int rt_volts;

  ADC10CTL1 = INCH_11;                  // AVcc/2
  ADC10CTL0 = SREF_1 + ADC10SHT_2 + REFON + ADC10ON + ADC10IE + REF2_5V;
  __delay_cycles(250);                  // delay to allow reference to settle
  ADC10CTL0 |= ENC + ADC10SC;           // Sampling and conversion start
  __bis_SR_register(LPM0_bits + GIE);     // LPM0 with interrupts enabled
  rt_volts = ADC10MEM;
  ADC10CTL0 &= ~ENC;
```

```
  ADC10CTL0 &= ~(REFON + ADC10ON);        // turn off A/D to save power
  rt_volts = (rt_volts*25)/512;
  return (rt_volts);
}

/*****************************************************************************
* BEGHDR
* NAME:       void transmit_time_delay(void)
* DESCRIPTION: Sets timer to transmit time based on timer_state
* INPUTS:     void
* PROCESSING:  Sets timer to transmit time, for 2 min and 4 min transmit times
*             loop number of 30 sec times to make 2 min and 4 min.
* OUTPUTS:    void
*****************************************************************************/
void transmit_time_delay(void)
{
  volatile unsigned int i = 0;
  in_delay = 1;
  switch (timer_state)
  {
    case timer_state_1sec:                // Timer State == 0; 1 Secs
      delay_count = 2;
      check_anemometer(delay_count);
      in_delay = 0;
      battery_full_timer += sec1;
      break;
    case timer_state_1:                   // Timer State == 1; 5 Secs
      delay_count = 10;
      check_anemometer(delay_count);
      in_delay = 0;
      battery_full_timer += sec5;
      break;
    case timer_state_2:                   // Timer State == 2; 10 Secs
      delay_count = 20;
      check_anemometer(delay_count);
      in_delay = 0;
      battery_full_timer += sec10;
      break;

    case timer_state_3:                   // Timer State == 3; 20 Secs
      delay_count = 40;
      check_anemometer(delay_count);
      in_delay = 0;
      battery_full_timer += sec20;
      break;
```

```
   case timer_state_4:                  // Timer State == 4; 40 Secs
     delay_count = 80;
     check_anemometer(delay_count);
     in_delay = 0;
     battery_full_timer += sec40;
     break;

   default:
     break;
  }
}
/*****************************************************************************
* BEGHDR
* NAME:       void display_mode(void)
* DESCRIPTION: Sets mode time to be displayed on the GUI in the voltage stage
*             for the first display
* INPUTS:      void
* PROCESSING:  Sets change_mode number based on timer_state
* OUTPUTS:     void
*****************************************************************************/
void display_mode(void)
{
  switch(timer_state)
  {
    case timer_state_1:
      change_mode=5;                   //~=5 sec
      break;
    case timer_state_2:
      change_mode=10;                   //~=10 sec
      break;
    case timer_state_3:
      change_mode=20;                   //~=20 sec
      break;
    case timer_state_4:
      change_mode=40;                   //~=40 sec
      break;
    case timer_state_5:
      change_mode=2;                   //~=2 min
      break;
    case timer_state_6:
      change_mode=4;                   //~=4 min
      break;
    default:
      break;
  }
}
```

```
/*****************************************************************************
*BEGHDR
*NAME:       __interrupt void ADC10_ISR(void)
*DESCRIPTION: ADC10 interrupt service routine
*INPUTS:     void
*PROCESSING:  Exit from LPM after interrupt
*OUTPUTS:    void
*****************************************************************************/
#pragma vector=ADC10_VECTOR
__interrupt void ADC10_ISR(void)
{
  __bic_SR_register_on_exit(LPM0_bits);     // Clear CPUOFF bit from 0(SR)
}
/*****************************************************************************
* BEGHDR
* NAME:       __interrupt void Port_1(void)
* DESCRIPTION: Port 1 interrupt service routine function key
* INPUTS:     void
* PROCESSING:  process the push button to switch to the next time mode
* OUTPUTS:    void
*****************************************************************************/
#pragma vector=PORT1_VECTOR
__interrupt void Port_1(void)
{
  if((P3IN & 0x20))                     // /Charge=1; battery, Blink Red
  {
    BSP_TURN_ON_LED1();
    __delay_cycles(10000);
    BSP_TURN_OFF_LED1();
  }
  else                                  // /Charge=0; Solar, blink green
  {
    BSP_TURN_ON_LED2();
    __delay_cycles(10000);
    BSP_TURN_OFF_LED2();
  }

  // If successful link, change timer state.
  if(status == status_six || status == status_five)
  {
    if(timer_state >= timer_state_6)        // If transmit time is == 6,
    {                                   // Set timer_state = 1
      timer_state = timer_state_1;
      display_mode();                   // Change GUI display time
    }
    else
```

```
   {
     timer_state++;                    // Change transmit time state
     display_mode();                   // Change GUI display time
   }
   if(in_delay)                        // If in transmit delay, exit and
   {                                   // send a new packet with new time
     __bic_SR_register_on_exit(LPM4_bits); // Clear LPM3 bit from 0(SR)
   }
  }
  __delay_cycles(150000);              // Debounce software delay
  while(!(P1IN & 0x04));               // Loop if button is still pressed
  P1IFG &= ~0x04;                      // P1.2 IFG cleared key interrupted
}
/*******************************************************************************
* CEC
* Function:    void sleep(unsigned int count)
* DESCRIPTION: A simple delay added to free-up timer B (will use more power
            than original delay)
* INPUTS:      delay count
* PROCESSING:  perform a number of no-ops to create a delay
* OUTPUTS:     VOID
*******************************************************************************/
void sleep(unsigned int count)
{
        int i;

        for (i = 0; i < 10; i++)
        {
                while(count > 0)
                {
                        count--;
                        __no_operation();
                        __no_operation();
                        __no_operation();
                }
        }
}
/*******************************************************************************
* CEC
* Function:    void check_anemometer(unsigned int RollCount)
* DESCRIPTION: Use timer B to measure the time between two pulses and create
*            a transmit delay. The function uses the timer B roll-over
*            interrupt and capture register 1 interrupt.
* INPUTS:      RollCount
* PROCESSING:  count number of timer B roll-over and build array containing
*            capture register data from anemometer interrupt pin. Compute
```

```
*           average of all capture samples then convert to mph.
*
*     Sum = all valid capture samples are adjusted for roll-over and summed
*     Average = sum/number of samples
*     timer clock period = 1/(ACLK/8) => ~ 1/125kHz ~ 8us
*     anemometer resolution = 2.5 mph/Hz or 0.4 Hz/mph
*     f = 1/(count * 8us)
*     mph = f/0.4 or (1/(count * 0.000008))/0.4
*     round = 0.05 (round to 1 decimal point)
*     offset = multiply answer by 10 to create integer value representing
*           1 decimal point of resolution
*
*     mph(rounded & offset) = 10(0.05 + (1/(count * 0.000008))/0.4))
*     mph(rounded & offset) = .5 + 10/(count * 0.00002)
*     wind_speed = integer(mph) Ex. 3.546 mph will give 35 wind_speed
*
* OUTPUTS: void
*******************************************************************************/
void check_anemometer(unsigned int RollCount)
{
  int TimerTemp, i, divisor;
  unsigned long int sum_value; // sum of all samples (roll-over offset applied)
  unsigned long int ave_value; // sum/#valid samples

  ave_value = 0;
  sum_value = 0;
  divisor = 0;

  _disable_interrupts();              // Disable interrupts

  rollover_count = 0;                 // clear counter
  index = 0;                          // clear RAM address
  data_avail = 0;                     // how many words written to RAM
  TimerTemp = TBCCR0;                 // Save current content of TBCCR0
  TBCCR0 = 0xFFFF;                    // Set new TBCCR0 delay (approx 524.3 ms)
  TBCTL |= TBCLR;                     // Clear TBR counter
  TBCCTL0 &= ~CCIFG;                  // Clear CCIFG Flag
  TBCCTL1 &= ~CCIFG;                  // Clear CCIFG Flag
  TBCCTL0 = CCIE;                     // TBCCR0 interrupt enabled

  // pos. edge + CCIXA + sync input + capture mode + Interrupt enable
  TBCCTL1 |= CM_1 + CCIS_1 + SCS + CAP + CCIE;

  __bis_SR_register(GIE);             // Enable interrupts
  TBCTL = TBSSEL_2 + MC_1 + ID_3;     // ACLK (~1MHz/8 = 125kHz) Start Timer B
```

```
  while (rollover_count < RollCount); // stay in loop until roll complete

  _disable_interrupts();                    // Disable interrupts
  TBCTL &= ~(MC_1);                         // Stop Timer B
  TBCCR0 = TimerTemp;

  // process data ****************************************
    // check for at least two samples captured, then calculate difference
    if (data_avail > 1)
     {
       // loop to process all valid capture data
       for(i=1 ; i < index ; i++)
        {
          // Timer did not roll over so no offset required
         if (roll_array[i] == roll_array[(i-1)])
          {
           sum_value = sum_value + capture_array[i] - capture_array[(i-1)];
           divisor++;
          }
         else
           {
                    // Timer rolled-over 1 time, add 65536 to data
                    if (roll_array[i] == (roll_array[(i-1)] + 1))
                             {
                                      sum_value = sum_value + capture_array[i]
                                               - capture_array[(i-1)] + 65536;
               divisor++;
                             }
                    else;  // Timer rolled over too many times, discard data
           }
        }
        // leave ave_value = 0 if no valid data (too many roll-overs)
        if (divisor != 0)
          ave_value = sum_value / divisor; // calculate average
     }
    else
      ave_value = 0; // less than 2 valid captures

          if (ave_value == 0)
      wind_speed = 0;
    else
      wind_speed = (int) (.5 + (10 / (ave_value * .00002)));
}
/*****************************************************************************
* CEC
* NAME:       __interrupt void Timer_B (void)
```

```
* DESCRIPTION: Timer B1 interrupt service routine
* INPUTS:      Void
* PROCESSING:  capture timer value and roll-over value in arrays, keep track
*              of valid data
* OUTPUTS:     Void
*******************************************************************************/
#pragma vector=TIMERB1_VECTOR
__interrupt void TimerB1_ISR (void)
{
        if (index == 15);                       // capture up to 16 samples
        else
                {
        capture_array[index] = TBCCR1;      // capture timer count at rising edge
    roll_array[index++] = rollover_count;     // keep track of rollovers
    data_avail++;                             // number of samples captured
   }
    TBCCTL1 &= ~CCIFG;                        // Clear CCIFG Flag
}
/*******************************************************************************
* BEGHDR
* modified by CEC to add roll-over
* NAME:        __interrupt void Timer_B (void)
* DESCRIPTION: Timer B0 interrupt service routine
* INPUTS:      Void
* PROCESSING:  increment roll counter and exit from LPM after interrupt
* OUTPUTS:     Void
*******************************************************************************/
#pragma vector=TIMERB0_VECTOR
__interrupt void TimerB_ISR (void)
{
 rollover_count++;
 __bic_SR_register_on_exit(LPM3_bits);     // Clear LPM3 bit from 0(SR)
}
```