```
/** ###################################################################
**     Filename  : ProcessorExpert.c
**     Project   : ProcessorExpert
**     Processor : MC13213
**     Version   : Driver 01.11
**     Compiler  : CodeWarrior HCS08 C Compiler
**     Date/Time : 11/27/2010, 5:45 AM
**     Abstract  :
**         Main module.
**         This module contains user's application code.
**     Settings  :
**     Contents  :
**         No public methods
**
** ###################################################################*/
/* MODULE ProcessorExpert */


/* Including needed modules to compile this module/procedure */
#include "Cpu.h"
#include "Events.h"
#include "RS232.h"
#include "I2C.h"
#include "Inhr1.h"
#include "Inhr2.h"
#include "ADC.h"
#include "EXT_IN1.h"
#include "EXT_OUT1_EN.h"
#include "EXT_OUT2_EN.h"
#include "RL_120.h"
#include "RL_AUX.h"
#include "RL_ENV.h"
#include "TEMP_INT.h"
#include "EXT_IN2.h"
#include "EXT_OUT1.h"
#include "EXT_OUT2.h"
#include "ADC_interval.h"
#include "TI1.h"
/* Include shared modules, which are used for whole project */
#include "PE_Types.h"
#include "PE_Error.h"
#include "PE_Const.h"
#include "IO_Map.h"
#include "shared.h"


void process_comms(void);
//word RS232_GetCharsInRxBuf(void);
void pot_init(void);
void temp_init(byte config);
//bool TEMP_INT_GetVal(void);
void manage_thermostat(void);
void manage_power_monitor(void);
byte ADC_interval_Enable(void);
byte TI1_Disable(void);


void main(void)
{
```

```
  /* Write your local variable definition here */

  /*** Processor Expert internal initialization. DON'T REMOVE THIS CODE!!! ***/
  PE_low_level_init();
  /*** End of Processor Expert internal initialization.                    ***/

  error = TI1_Disable();
  pot_init();
  temp_init(temp_config_val_run);

  //if (module_config==module_config_powr_ctl)
       //{
            //error = ADC_interval_Enable();
       //}

  while (1)
  {

            //if (module_config==module_config_zig_base)
           //    {
                 manage_thermostat();
                 if (RS232_GetCharsInRxBuf()>0)
                    {
                       process_comms();
                    }
                 //}


            //if (module_config==module_config_powr_ctl)
           //    {
                 //manage_power_monitor();
           //    }

  }









  /*** Don't write any code pass this line, or it will be deleted during code
generation. ***/
  /*** Processor Expert end of main routine. DON'T MODIFY THIS CODE!!! ***/
  for(;;){}
  /*** Processor Expert end of main routine. DON'T WRITE CODE BELOW!!! ***/
} /*** End of main routine. DO NOT MODIFY THIS TEXT!!! ***/

/* END ProcessorExpert */
/*
** ###################################################################
**
**     This file was created by Processor Expert 4.00 Beta [04.40]
**     for the Freescale HCS08 series of microcontrollers.
**
** ###################################################################
*/
```

```
/*************************************************************************8
 * Comms.c
 *
 *
 *
 */


/* Including needed modules to compile this module/procedure */
#include "Cpu.h"
#include "Events.h"
#include "RS232.h"
#include "I2C.h"
#include "Inhr1.h"
#include "Inhr2.h"
#include "ADC.h"
#include "EXT_IN1.h"
#include "EXT_OUT1_EN.h"
#include "EXT_OUT2_EN.h"
#include "RL_120.h"
#include "RL_AUX.h"
#include "RL_ENV.h"
#include "TEMP_INT.h"
#include "EXT_IN2.h"
#include "EXT_OUT1.h"
#include "EXT_OUT2.h"
/* Include shared modules, which are used for whole project */
#include "PE_Types.h"
#include "PE_Error.h"
#include "PE_Const.h"
#include "IO_Map.h"
#include "shared.h"




void process_comms(void);
byte ADC_MeasureChan(bool WaitForResult,byte Channel);
byte ADC_GetChanValue16(byte Channel,word *Value);
byte RS232_SendChar(RS232_TComData Chr);
byte RS232_RecvChar(RS232_TComData *Chr);
byte RS232_SendBlock(RS232_TComData * Ptr, word Size, word *Snd);
void RL_120_PutVal(bool Val);
void RL_AUX_PutVal(bool Val);
void RL_ENV_PutVal(bool Val);
byte EXT_OUT1_SetRatio16(word Ratio);
byte EXT_OUT2_SetRatio16(word Ratio);
void pot_set_value(byte value);
byte pot_get_value(void);
byte get_local_temp(void);
void set_local_temp_limit_high(byte temp);
void set_local_temp_limit_low(byte temp);
byte get_local_temp_limit_high(void);
byte get_local_temp_limit_low(void);

byte zigbee_power_switch_get_status(byte addr);
void zigbee_power_switch_set_status(byte addr, byte status);
```

```
word zigbee_power_switch_get_pwr(byte addr);
void zigbee_garage_door_set_act(byte addr);
byte zigbee_get_new_device(void);
void zigbee_remove_device(byte addr);
//byte rx_complete_Enable(void);
//byte rx_complete_Disable(void);
byte RS232_ClearRxBuf(void);
byte RS232_ClearTxBuf(void);



byte error = 0;
word adc_result = 0;
byte comm_string[16];
byte in_length = 0;
word out_length = 0;
byte param_h = 0;
byte param_l = 0;
byte addr = 1;
byte cmd = 0;
word sent = 0;
word math = 0;



/*    TEST MODE (ADDR = 0) FUNCTIONS:
 *    MEASURE POWER SOURCE
 *  CONTROL 120AC RELAY
 *  CONTROL AUX RELAY
 *  CONTROL ENV RELAY
 *  SET CURRENT SENSE GAIN
 *  GET CURRENT MEASUREMENT
 *  GET TEMPERATURE
 *  SET EXT OUT1
 *  SET EXT OUT2
 *  GET EXT IN1
 *  GET EXT IN2
 *
 *

 ZIGBEE BASE SERIAL PROTOCOL: ADDRESS, CMD, PARAM/RESP H, PARAM/RESP L


COMMAND TYPE                        ADDR  CMD         PARAM/RESP H
      PARAM/RESP L
THERMO GET MODE   C/H/F       1-X         0           0
      0/1/2
THERMO SET MODE C/H/F         1-X         1           0
      0/1/2
THERMO GET TEMP LIMIT L       1-X         2           0
      TEMP L (SIGNED BYTE)
THERMO SET TEMP LIMIT L       1-X         3           0
      TEMP L (SIGNED BYTE)
THERMO GET TEMP LIMIT H       1-X         4           0
      TEMP H (SIGNED BYTE)
THERMO SET TEMP LIMIT H       1-X         5           0
      TEMP H (SIGNED BYTE)
THERMO GET CURRENT TEMP       1-X         6           0
```

```
      CURRENT TEMP (SIGNED BYTE)

POWER SWITCH GET STATUS       1-X         7           0
      OFF/ON (0/1)
POWER SWITCH SET STATUS       1-X         8           0
      OFF/ON (0/1)
POWER SWITCH GET PWR          1-X         9           PWR H
      PWR L (UNSIGNED SHORT FLOAT WITH 2 DEC PT)

GARAGE DOOR SET ACT                 1-X         10          0
      1
DOOR SENSOR GET STATUS        1-X         11          0
      OPEN/CLOSED (0/1)
POLL FOR NEW DEVICE           (BASE)      12          0
      0 || NEW DEVICE ADDR (1-X) OR 0 IF ERROR
REMOVE DEVICE FROM NETWORK    (BASE)      13          0
      REMOVED DEVICE ADDR (1-X)

** ADDRESS = 0 SETS DEVICE INTO HARDWARE TEST MODE


 */


void process_comms(void)
       {


       switch (in_length)
                   {
                   case 0:
                               //error = rx_complete_Enable();
                               error = RS232_RecvChar(&addr);
                               break;

                   case 1:
                               error = RS232_RecvChar(&cmd);
                               break;

                   case 2:
                               error = RS232_RecvChar(&param_h);
                               break;

                   case 3:
                               error = RS232_RecvChar(&param_l);
                               break;
                   }
       in_length++;

       if (in_length==4)
             {//error = rx_complete_Disable();
              in_length = 0;
             }
       else
             {return;}

       //error = RS232_ClearRxBuf();
       //comm_string[0] = 0x0D;
       //comm_string[1] = 0x0A;
```

```
comm_string[0] = 1;          //address
comm_string[1] = cmd;
comm_string[2] = 0;


if (addr==0)           // local hardware test mode
     {
     switch (cmd)
            {
                 case 0:     // MEASURE POWER SOURCE
                        ADC_MeasureChan(TRUE, 2);
                        error = ADC_GetChanValue16(2, &adc_result);
                        if (adc_result<15000)
                             {comm_string[0] = 'B';
                              comm_string[1] = 'A';
                              comm_string[2] = 'T';
                              comm_string[3] = 'T';
                              out_length = 4;
                             }
                        else
                              {comm_string[0] = 'A';
                               comm_string[1] = 'C';
                               out_length = 2;
                              }
                        break;

                 case 1:     // CONTROL 120AC RELAY
                        RL_120_PutVal((bool)param_l);
                        break;

                 case 2:     // CONTROL AUX RELAY
                        RL_AUX_PutVal((bool)param_l);
                        break;

                 case 3:     // CONTROL ENV RELAY
                        RL_ENV_PutVal((bool)param_l);
                        break;

                 case 4:     // SET CURRENT SENSE GAIN
                        pot_set_value(param_l);
                        comm_string[0] = pot_get_value();
                        out_length = 1;
                        break;

                 case 5:     // GET CURRENT MEASUREMENT
                        ADC_MeasureChan(TRUE, 3);
                        error = ADC_GetChanValue16(3, &adc_result);
                        comm_string[0] = (byte)(adc_result >> 8);
                        comm_string[1] = (byte)(0x00FF & adc_result);
                        out_length = 2;
                        break;

                 case 6:     // GET TEMPERATURE
                        param_l = get_local_temp();
                        if (param_l & 0x80)
                           {
                           comm_string[0] = '-';
                           comm_string[1] = param_l & 0x7F;
                           out_length = 1;
```

```
                        }
                    else
                        {
                        comm_string[0] = param_l;
                        out_length = 1;
                        }
                    break;

                case 7:     // SET EXT OUT1
                    EXT_OUT1_SetRatio16(((word)param_l) * 655);
                    out_length = 0;
                    break;

                case 8:     // SET EXT OUT2
                    EXT_OUT2_SetRatio16(((word)param_l) * 655);
                    out_length = 0;
                    break;

                case 9:     // GET EXT IN1
                    ADC_MeasureChan(TRUE, 0);
                    error = ADC_GetChanValue16(0, &adc_result);
                    comm_string[0] = (byte)(adc_result >> 8);
                    comm_string[1] = (byte)(0x00FF & adc_result);
                    out_length = 2;
                    break;

                case 10:    // GET EXT IN2
                    ADC_MeasureChan(TRUE, 1);
                    error = ADC_GetChanValue16(1, &adc_result);
                    comm_string[0] = (byte)(adc_result >> 8);
                    comm_string[1] = (byte)(0x00FF & adc_result);
                    out_length = 2;
                    break;

                case 11:    // set temp high threshold
                    set_local_temp_limit_high(param_l);
                    comm_string[0] = get_local_temp_limit_high();
                    out_length = 1;
                    break;

                case 12:    // set temp low threshold
                    set_local_temp_limit_low(param_l);
                    comm_string[0] = get_local_temp_limit_low();
                    out_length = 1;
                    break;

                default:
                    break;
            }
    error = RS232_SendBlock(comm_string, out_length, &sent);
    }

else        // networked command mode
    {
    switch (cmd)
            {
            case 0:           // THERMO GET MODE OFF/C/H/F
                comm_string[3] = thermo_mode;
                break;
```

```
                        case 1:           // THERMO SET MODE OFF/C/H/F
                              thermo_mode = param_l;
                              comm_string[3] = param_l;
                              break;
                        /*
                        case 2:           // THERMO GET TEMP LIMIT L
                              comm_string[3] = get_local_temp_limit_low();
                              break;

                        case 3:           // THERMO SET TEMP LIMIT L
                              set_local_temp_limit_high(param_l);
                              comm_string[3] = get_local_temp_limit_low();
                              break;
                        */
                        case 4:     // THERMO GET TEMP LIMIT H
                              //math = (word)(get_local_temp_limit_high());
                              //math = ((math+32)*9)/5;                         //
convert temp from celcius to farenheit
                              //comm_string[3] = (byte)math;
                              if (thermo_mode==cool)
                                    {comm_string[3] = get_local_temp_limit_high();}
                              else if (thermo_mode==heat)
                                    {comm_string[3] = get_local_temp_limit_low();}
                              else {}
                              break;

                        case 5:           // THERMO SET TEMP LIMIT H
                              //math = param_l;
                              //math = ((math-32)*5)/9;                         //
convert temp from farenheit to celcius
                              //local_temp_limit = (byte)math;
                              //set_local_temp_limit_high(local_temp_limit);
                              if (thermo_mode==cool)
                                    {set_local_temp_limit_high(param_l);
                                     comm_string[3] = get_local_temp_limit_high();}
                              else if (thermo_mode==heat)
                                    {set_local_temp_limit_low(param_l);
                                    comm_string[3] = get_local_temp_limit_low();}
                              else {}
                              //comm_string[3] = param_l;
                              break;

                        case 6:           // THERMO GET CURRENT TEMP
                              //math = (word)get_local_temp();          // convert
temp from celcius to farenheit
                              //math = ((math+32)*9)/5;
                              //comm_string[3] = (byte)math;
                              comm_string[3] = get_local_temp();
                              break;

                        case 7:           // POWER SWITCH GET STATUS
                              //comm_string[3] =
zigbee_power_switch_get_status(addr);
                              if (RL_120_GetVal())
                                    {comm_string[3] = 1;}
                              else
                                    {comm_string[3] = 0;}
                              //comm_string[3] = (byte)(RL_120_GetVal());
```

```
                        break;

                  case 8:           // POWER SWITCH SET STATUS
                        //zigbee_power_switch_set_status(addr, param_l);
                        RL_120_PutVal((bool)param_l);
                        comm_string[3] = param_l;
                        break;

                  case 9:           // POWER SWITCH GET PWR
                        math = zigbee_power_switch_get_pwr(addr);
                        comm_string[2] = (byte)(math>>8);
                        comm_string[3] = (byte)(math&0x00ff);
                        break;

                  case 10:    // GARAGE DOOR SET ACT
                        zigbee_garage_door_set_act(addr);
                        comm_string[3] = param_l;
                        break;

                  case 11:    // DOOR SENSOR GET STATUS
                        ADC_MeasureChan(TRUE, 0);
                        error = ADC_GetChanValue16(0, &adc_result);
                        if (adc_result<0x8000)
                              {comm_string[3] = 1;}
                        else
                              {comm_string[3] = 0;}


                        break;

                  case 12:    // POLL FOR NEW DEVICE
                        comm_string[0] = 1;
                        comm_string[1] = 12;
                        comm_string[2] = 0;
                        comm_string[3] = 1;

                        //comm_string[3] = zigbee_get_new_device();
                        break;

                  case 13:    // REMOVE DEVICE FROM NETWORK
                        zigbee_remove_device(param_l);
                        comm_string[3] = param_l;
                        break;

                  default:
                        break;


                  }
      error = RS232_SendBlock(comm_string, 4, &sent);
      //error = RS232_ClearTxBuf();

      }
}
```

```
/*************************************************************
 * Applications.c
 *
 *
 *
 *
 */

/* Including needed modules to compile this module/procedure */
#include "Cpu.h"
#include "Events.h"
#include "RS232.h"
#include "I2C.h"
#include "Inhr1.h"
#include "Inhr2.h"
#include "ADC.h"
#include "EXT_IN1.h"
#include "EXT_OUT1_EN.h"
#include "EXT_OUT2_EN.h"
#include "RL_120.h"
#include "RL_AUX.h"
#include "RL_ENV.h"
#include "TEMP_INT.h"
#include "EXT_IN2.h"
#include "EXT_OUT1.h"
#include "EXT_OUT2.h"
/* Include shared modules, which are used for whole project */
#include "PE_Types.h"
#include "PE_Error.h"
#include "PE_Const.h"
#include "IO_Map.h"
#include "shared.h"




void pot_init(void);
void pot_set_value(byte value);
byte pot_get_value(void);
void temp_init(byte config);
byte I2C_SelectSlave(byte Slv);
byte I2C_SendBlock(void * Ptr,word Siz,word *Snt);
byte I2C_RecvChar(byte *Chr);
byte I2C_RecvBlock(void* Ptr,word Siz,word *Rcv);
byte I2C_SendStop(void);
byte I2C_SendChar(byte Chr);
byte get_local_temp(void);
void set_local_temp_limit_high(byte temp);
void set_local_temp_limit_low(byte temp);
byte get_local_temp_limit_high(void);
byte get_local_temp_limit_low(void);
void manage_thermostat(void);
void manage_power_monitor(void);
//bool RL_120_GetVal(void);
//bool RL_AUX_GetVal(void);
//bool RL_ENV_GetVal(void);
void RL_120_PutVal(bool Val);
void RL_AUX_PutVal(bool Val);
```

```
void RL_ENV_PutVal(bool Val);
byte ADC_MeasureChan(bool WaitForResult,byte Channel);
byte ADC_GetChanValue16(byte Channel,word *Value);
//byte Garage_door_timer_GetCounterValue(Garage_door_timer_TTimerValue *Value)
byte ADC_interval_Enable(void);
byte timer_count = 0;
byte TI1_Enable(void);
byte TI1_Disable(void);



byte I2C_data[8];
word received = 0;
byte thermo_mode = off;
byte local_temp_limit = 0;
word pot_val_left = 0;
word pot_val_right = 0xFFFF;
word mid_point = 0;
word curr_adc_max = 0;
byte curr_pot_val = 0;
bool adc_waiting = FALSE;
unsigned long power_avg = 0;
unsigned long power_math = 0;
word num_avgs = 0;
word curr_adc_val = 0;
word temp_val = 0;


////////////////////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////////////////

void pot_init(void)
      {
      // set up TCON reg (disable A terminal, enable B and W), set initial pot
value
      error = I2C_SelectSlave(pot_addr);

      I2C_data[0] = pot_cmd_TCON_write;        // cmd byte -> TCON
      I2C_data[1] = pot_TCON_init;                              // data byte ->
set TCON (disable A, enable B and W)
      I2C_data[2] = pot_cmd_val_write;         // cmd byte -> set pot value
      I2C_data[3] = pot_val_init;                               // data byte ->
pot value
      curr_pot_val = pot_val_init;

      error = I2C_SendBlock(I2C_data , 4, &sent);
      error = I2C_SendStop();

      }

void pot_set_value(byte value)
      {
      error = I2C_SelectSlave(pot_addr);
      I2C_data[0] = pot_cmd_val_write;         // cmd byte -> set pot value
      I2C_data[1] = value;                                  // data byte -> pot
value

      error = I2C_SendBlock(I2C_data , 2, &sent);
      error = I2C_SendStop();
```

```
	}


byte pot_get_value(void)
	{
	error = I2C_SelectSlave(pot_addr);
	I2C_data[0] = pot_cmd_val_read;                 // cmd byte -> set pot value
	error = I2C_SendBlock(I2C_data , 2, &sent);

	I2C_data[0] = 0;
	I2C_data[1] = 0;

	error = I2C_RecvBlock(I2C_data, 2, &received);

	return I2C_data[1];
	}


void temp_init(byte config)
	{
	error = I2C_SelectSlave(temp_addr);

	// set the configuration reg
	I2C_data[0] = temp_config_reg_w;
	I2C_data[1] = config;
	error = I2C_SendBlock(I2C_data , 2, &sent);
	error = I2C_SendStop();

	// set the conversion reg
	I2C_data[0] = temp_conv_rate_reg_w;
	I2C_data[1] = temp_conv_rate_val;
	error = I2C_SendBlock(I2C_data , 2, &sent);
	error = I2C_SendStop();

	// set the initial high temp limit
	I2C_data[0] = temp_limit_reg_local_high_w;
	I2C_data[1] = temp_limit_val_local_high;
	error = I2C_SendBlock(I2C_data , 2, &sent);
	error = I2C_SendStop();

	// set the initial low temp limit
	I2C_data[0] = temp_limit_reg_local_low_w;
	I2C_data[1] = temp_limit_val_local_low;
	error = I2C_SendBlock(I2C_data , 2, &sent);
	error = I2C_SendStop();

	}


byte get_local_temp(void)
	{
	//float temp_f = 0;
	//signed short temp_w = 0;
	byte temp_b = 0xFF;
	//signed char temp_s = 0;

	error = I2C_SelectSlave(temp_addr);
	error = I2C_SendChar(temp_temp_reg_high);
	error = I2C_RecvChar(&temp_b);
```

```
	error = I2C_SendStop();

	//temp_w = ((signed char)temp_b);
	return temp_b;

	/*
	error = I2C_SelectSlave(temp_addr);
	error = I2C_SendChar(temp_temp_reg_low);
	error = I2C_RecvChar(&temp_b);
	error = I2C_SendStop();

	temp_w |= ((signed short)temp_b);
	*/
	}


void set_local_temp_limit_high(byte temp)
	{
	I2C_data[0] = temp_limit_reg_local_high_w;
	I2C_data[1] = temp;
	error = I2C_SendBlock(I2C_data , 2, &sent);
	error = I2C_SendStop();
	}

void set_local_temp_limit_low(byte temp)
	{
	I2C_data[0] = temp_limit_reg_local_low_w;
	I2C_data[1] = temp;
	error = I2C_SendBlock(I2C_data , 2, &sent);
	error = I2C_SendStop();
	}

byte get_local_temp_limit_high(void)
	{
	byte temp = 0xFF;

	error = I2C_SelectSlave(temp_addr);
	error = I2C_SendChar(temp_limit_reg_local_high_r);
	error = I2C_RecvChar(&temp);
	error = I2C_SendStop();

	return temp;
	}

byte get_local_temp_limit_low(void)
	{
	byte temp = 0xFF;

	error = I2C_SelectSlave(temp_addr);
	error = I2C_SendChar(temp_limit_reg_local_low_r);
	error = I2C_RecvChar(&temp);
	error = I2C_SendStop();

	return temp;
	}

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

```
void manage_thermostat(void)
       {
       // FAN = RL_120
       // COOL = RL_ENV
       // HEAT = RL_AUX

       byte temp = 0;
       byte current_temp = 0;

       if (!TEMP_INT_GetVal() )     // check /ALERT
            {
            error = I2C_SelectSlave(temp_addr);
            error = I2C_SendChar(temp_stat_reg);
            error = I2C_RecvChar(&temp);
            error = I2C_SendStop();


            if ((temp & temp_stat_local_high_bit)&&(thermo_mode==cool)&&(!
RL_ENV_GetVal() ))            // temp is too hot -> turn on cool
               {
               RL_ENV_PutVal(1);
               RL_120_PutVal(1);
               }

            else if ((temp & temp_stat_local_low_bit)&&(thermo_mode==heat)&&(!
RL_AUX_GetVal() ))      // temp is too cold -> turn on heat
                     {
                     RL_AUX_PutVal(1);
                  RL_120_PutVal(1);
                        }

            else {}     // thermostat is in off mode or don't care about alert
            }

       else if ((thermo_mode==cool) && RL_ENV_GetVal() )          // COOLER IS ON
BUT ALERT IS NO LONGER ACTIVE -> turn off cooler
                  {
                  current_temp = get_local_temp();
                  if (current_temp < (local_temp_limit-1) )           // add
hysteresis to temp control
                       {RL_ENV_PutVal(0);
                        RL_120_PutVal(0);
                       }
                  }

       else if ((thermo_mode==heat) && RL_AUX_GetVal() )          // HEATER IS ON
BUT ALERT IS NO LONGER ACTIVE -> turn off heater
                  {
                  if (current_temp > (local_temp_limit+1) )           // add
hysteresis to temp control
                       {RL_AUX_PutVal(0);
                        RL_120_PutVal(0);
                       }
                  }

       if ((thermo_mode != cool) && RL_ENV_GetVal() )             // cooler is on
but no longer in cool mode -> turn off cooler
            {
```

```
             RL_ENV_PutVal(0);
             RL_120_PutVal(0);
            }
       if ((thermo_mode != heat) && RL_AUX_GetVal() )            // heater is on
but no longer in heat mode -> turn off heater
            {
             RL_AUX_PutVal(0);
             RL_120_PutVal(0);
            }

       if ((thermo_mode==fan)&&(!RL_120_GetVal() ))              // in fan mode but
fan isn't on -> turn on fan
            {
            RL_120_PutVal(1);
            }

       if ((RL_120_GetVal() )&&(!RL_ENV_GetVal() )&&(RL_AUX_GetVal() ))      // fan
is on but no longer in fan mode -> turn off fan
            {
             RL_120_PutVal(0);
            }

       }



void manage_power_monitor(void)
      {
      // aim for peaks to be at 90% of rail


      //while (1)
      //      {

              ////////////////////////////////
              // get a peak
              curr_adc_max = 0;
              adc_waiting = TRUE;
              error = ADC_interval_Enable();
              while (adc_waiting)
                      {
                        error = ADC_MeasureChan(TRUE, 3);
                        error = ADC_GetChanValue16(3, &curr_adc_val);
                        if (curr_adc_val>curr_adc_max)           // find max val_
sample for this wave period
                            {curr_adc_max = curr_adc_val;}
                      }

              ////////////////////////////////
              // see if the peak is in range
              if (!((curr_adc_max>adc_window_min)&&(curr_adc_max<adc_window_max)))
      // need to tune pot
                   {pot_val_left = 0;
                    pot_val_right = 0xFFFF;
                    num_avgs = 0;

                    while (!
((curr_adc_max>adc_window_min)&&(curr_adc_max<adc_window_max)) ) // tune pot value
```

```
                                {
                                mid_point = (pot_val_left + pot_val_right)/2;
                                pot_set_value(curr_pot_val);

                                ////////////////////////////////
                                // get a new peak value
                                curr_adc_max = 0;
                                adc_waiting = TRUE;
                                error = ADC_interval_Enable();
                                while (adc_waiting)
                                      {
                                        error = ADC_MeasureChan(TRUE, 3);
                                        error = ADC_GetChanValue16(3, &curr_adc_val);
                                        if (curr_adc_val>curr_adc_max)          //
find max val sample for this wave period
                                           {curr_adc_max = curr_adc_val;}
                                        }

                                if (curr_adc_max>adc_window_max)          // too high,
throw away right half
                                   {pot_val_right = mid_point;}

                                else if (curr_adc_max<adc_window_min)     // too low,
throw away left half
                                           {pot_val_left = mid_point;}

                                else  {}                                  // value
within window

                                }
                          }
                     else      // peak is in range -> keep it
                          {power_math = (1515*(((unsigned long)curr_adc_max)-32768))/
(curr_pot_val);
                           power_avg = ((power_avg*num_avgs) + curr_adc_max);
                           num_avgs++;
                           power_avg = power_avg/num_avgs;
                          }
                     //}
            }

word report_pwr(void)
      {
      word temp_pwr = 0;
      temp_pwr = (word)power_avg;
      power_avg = 0;
      num_avgs = 0;
      return temp_pwr;
      }



void garage_door_activate(void)
      {


      RL_AUX_PutVal(TRUE);
      garage_wait = TRUE;
```

```
//error = TI1_EnableEvent();
error = TI1_Enable();

while (garage_wait==TRUE)
        {;}

error = TI1_Disable();
//error = TI1_DisableEvent();

RL_AUX_PutVal(FALSE);
}
```

```
/*****************************************************************************
 * zigbee_app_interface.c
 *
 *
 *
 *
 *
 *
 */

#include "Cpu.h"
#include "Events.h"
#include "RS232.h"
#include "I2C.h"
#include "Inhr1.h"
#include "Inhr2.h"
#include "ADC.h"
#include "EXT_IN1.h"
#include "EXT_OUT1_EN.h"
#include "EXT_OUT2_EN.h"
#include "RL_120.h"
#include "RL_AUX.h"
#include "RL_ENV.h"
#include "TEMP_INT.h"
#include "EXT_IN2.h"
#include "EXT_OUT1.h"
#include "EXT_OUT2.h"
#include "ADC_interval.h"
/* Include shared modules, which are used for whole project */
#include "PE_Types.h"
#include "PE_Error.h"
#include "PE_Const.h"
#include "IO_Map.h"
#include "shared.h"



byte zigbee_power_switch_get_status(byte addr);
void zigbee_power_switch_set_status(byte addr, byte status);
word zigbee_power_switch_get_pwr(byte addr);
void zigbee_garage_door_set_act(byte addr);
byte zigbee_get_new_device(void);
void zigbee_remove_device(byte addr);
void RL_120_PutVal(bool Val);
word report_pwr(void);
void garage_door_activate(void);




byte zigbee_power_switch_get_status(byte addr)
      {
      return RL_120_GetVal();
      }


void zigbee_power_switch_set_status(byte addr, byte status)
      {
      RL_120_PutVal(status);
```

```
	}


word zigbee_power_switch_get_pwr(byte addr)
	{
	return report_pwr();
	}


void zigbee_garage_door_set_act(byte addr)
	{
	garage_door_activate();
	}


byte zigbee_get_new_device(void)
	{
	return 0;
	}

void zigbee_remove_device(byte addr)
	{

	}
```

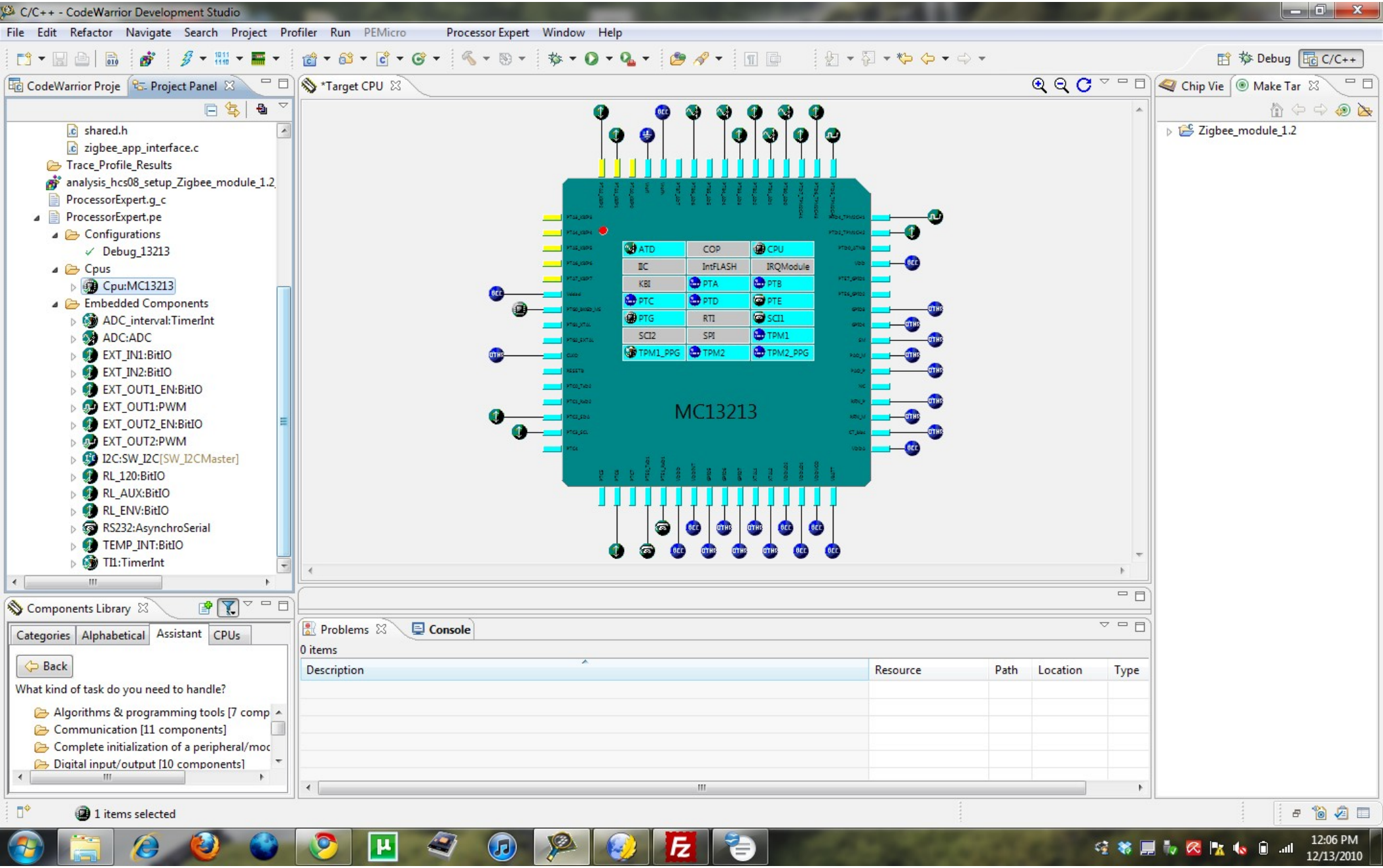
```
/** ###################################################################
**     Filename  : Events.c
**     Project   : ProcessorExpert
**     Processor : MC13213
**     Component : Events
**     Version   : Driver 01.02
**     Compiler  : CodeWarrior HCS08 C Compiler
**     Date/Time : 11/27/2010, 5:45 AM
**     Abstract  :
**         This is user's event module.
**         Put your event handler code here.
**     Settings  :
**     Contents  :
**         No public methods
**
** ###################################################################*/
/* MODULE Events */


#include "Cpu.h"
#include "Events.h"
#include "shared.h"


byte ADC_interval_Disable(void);
//byte rx_complete_Disable(void);
byte RS232_ClearRxBuf(void);
void process_comms(void);



bool garage_wait = TRUE;




/*
** ===================================================================
**     Event       :  ADC_interval_OnInterrupt (module Events)
**
**     Component   :  ADC_interval [TimerInt]
**     Description :
**         When a timer interrupt occurs this event is called (only
**         when the component is enabled - <Enable> and the events are
**         enabled - <EnableEvent>). This event is enabled only if a
**         <interrupt service/event> is enabled.
**     Parameters  : None
**     Returns     : Nothing
** ===================================================================
*/
void ADC_interval_OnInterrupt(void)
{
      error = ADC_interval_Disable();
      adc_waiting = FALSE;
/*
  if ((curr_adc_max>adc_window_min)&&(curr_adc_max<adc_window_max))
       {power_math = (1515*(((unsigned long)curr_adc_max)-32768))/(curr_pot_val);
        power_avg = ((power_avg*num_avgs) + curr_adc_max);
        num_avgs++;
```

```
        power_avg = power_avg/num_avgs;
       }

  else

        curr_adc_max = 0;

*/


}



/*
** ===================================================================
**     Event       :  rx_complete_OnInterrupt (module Events)
**
**     Component   :  rx_complete [TimerInt]
**     Description :
**         When a timer interrupt occurs this event is called (only
**         when the component is enabled - <Enable> and the events are
**         enabled - <EnableEvent>). This event is enabled only if a
**         <interrupt service/event> is enabled.
**     Parameters  : None
**     Returns     : Nothing
** ===================================================================
*/

/*
void rx_complete_OnInterrupt(void)
{
      error = rx_complete_Disable();
      error = RS232_ClearRxBuf();
      in_length = 0;
}
*/


/*
** ===================================================================
**     Event       :  TI1_OnInterrupt (module Events)
**
**     Component   :  TI1 [TimerInt]
**     Description :
**         When a timer interrupt occurs this event is called (only
**         when the component is enabled - <Enable> and the events are
**         enabled - <EnableEvent>). This event is enabled only if a
**         <interrupt service/event> is enabled.
**     Parameters  : None
**     Returns     : Nothing
** ===================================================================
*/
void TI1_OnInterrupt(void)
{
  garage_wait = FALSE;

}
```

```
/*************************************************************
 * shared.h
 *
 *
 *
 *
 *
 *
 */

#include "Cpu.h"
#include "Events.h"
#include "RS232.h"
#include "I2C.h"
#include "Inhr1.h"
#include "Inhr2.h"
#include "ADC.h"
#include "EXT_IN1.h"
#include "EXT_OUT1_EN.h"
#include "EXT_OUT2_EN.h"
#include "RL_120.h"
#include "RL_AUX.h"
#include "RL_ENV.h"
#include "TEMP_INT.h"
#include "EXT_IN2.h"
#include "EXT_OUT1.h"
#include "EXT_OUT2.h"
/* Include shared modules, which are used for whole project */
#include "PE_Types.h"
#include "PE_Error.h"
#include "PE_Const.h"
#include "IO_Map.h"


#define pot_addr              0x2E
#define pot_cmd_val_read      0x0C
#define pot_cmd_val_write     0x00
#define pot_cmd_TCON_write    0x40
#define pot_cmd_TCON_read     0x4C
#define pot_val_init          0x00
#define pot_TCON_init         0x0B

#define temp_addr                          0x48
#define temp_conv_rate_reg_r       0x04
#define temp_conv_rate_reg_w       0x0A
#define temp_conv_rate_val                 0x04  // 1Hz
#define temp_config_val_run                0x15
#define temp_config_val_stop       0x35
#define temp_config_reg_r                  0x03
#define temp_config_reg_w                  0x09
#define temp_limit_reg_local_high_r 0x05
#define temp_limit_reg_local_high_w 0x0B
#define temp_limit_reg_local_low_r  0x06
#define temp_limit_reg_local_low_w  0x0C
#define temp_limit_val_local_high   35
#define temp_limit_val_local_low    10
#define temp_alert_mode_reg                0xBF
#define temp_temp_reg_high                 0x00
#define temp_temp_reg_low                  0x22
```

```
#define temp_stat_reg                       0x02
#define temp_stat_local_high_bit    0x40
#define temp_stat_local_low_bit             0x20
#define temp_stat_remote_high_bit   0x10
#define temp_stat_remote_low_bit    0x08
#define temp_stat_remote_crit_bit   0x02
#define temp_stat_local_crit_bit    0x01

#define off                                         0x00
#define cool                                        0x01
#define heat                                        0x02
#define fan                                         0x03

#define module_config_zig_base              0x00
#define module_config_powr_ctl              0x01
#define module_config_gdoor                 0x02
#define module_config_aux                   0x03


#define adc_window_max                      62270
#define adc_window_min                      55700

//////////////
// DEFINES CURRENT DEVICE CONFIGURATION!!
#define module_config                       0x00
```

```
extern byte error;
extern word sent;
extern byte thermo_mode;
extern byte local_temp_limit;
extern unsigned long power_avg;
extern word num_avgs;
extern word curr_adc_max;
extern byte curr_pot_val;
extern bool adc_waiting;
extern bool garage_wait;
extern byte in_length;
extern byte addr;
extern byte cmd;
extern byte param_l;
extern byte param_h;
```


C/C++ - CodeWarrior Development Studio
File Edit Refactor Navigate Search Project Profiler Run PEMicro Processor Expert Window Help
Debug C/C++
CodeWarrior Proje Project Panel
shared.h
zigbee_app_interface.c
Trace_Profile_Results
analysis_hcs08_setup_Zigbee_module_1.2
ProcessorExpert.g_c
ProcessorExpert.pe
Configurations
Debug_13213
Cpus
Cpu:MC13213
Embedded Components
ADC_interval:TimerInt
ADC:ADC
EXT_IN1:BitIO
EXT_IN2:BitIO
EXT_OUT1_EN:BitIO
EXT_OUT1:PWM
EXT_OUT2_EN:BitIO
EXT_OUT2:PWM
I2C:SW_I2C[SW_I2CMaster]
RL_120:BitIO
RL_AUX:BitIO
RL_ENV:BitIO
RS232:AsynchroSerial
TEMP_INT:BitIO
TI1:TimerInt
*Target CPU
ATD COP CPU
IIC IntFLASH IRQModule
KBI PTA PTB
PTC PTD PTE
PTG RTI SCI1
SCI2 SPI TPM1
TPM1_PPG TPM2 TPM2_PPG
MC13213
Chip Vie Make Tar
Zigbee_module_1.2
Components Library
Categories Alphabetical Assistant CPUs
Back
What kind of task do you need to handle?
Algorithms & programming tools [7 comp
Communication [11 components]
Complete initialization of a peripheral/moc
Digital input/output [10 components]
Problems Console
0 items
Description Resource Path Location Type
1 items selected
12:06 PM
12/13/2010