```
/*


                     ///PROBA DI RECEZIONE. PROGRAMA TEST UNICAST.\\\
                                                 Università degli Studi di Pavia
                                                        + Pablo Meca Calderón +
                                                        UNICAST EMISORE
                                                                 Versione 1.0








    NanoStack: MCU software and PC tools for IP-based wireless sensor
networking.

    Copyright (C) 2006-2007 Sensinode Ltd.

    This program is free software; you can redistribute it and/or modify
    it under the terms of the GNU General Public License as published by
    the Free Software Foundation; either version 2 of the License, or
    (at your option) any later version.

    This program is distributed in the hope that it will be useful,
    but WITHOUT ANY WARRANTY; without even the implied warranty of
    MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
    GNU General Public License for more details.

    You should have received a copy of the GNU General Public License along
    with this program; if not, write to the Free Software Foundation, Inc.,
    51 Franklin Street, Fifth Floor, Boston, MA 02110-1301 USA.

                Address:
                Sensinode Ltd.
                Teknologiantie 6
                90570 Oulu, Finland

                E-mail:
                info@sensinode.com
*/

/**
 *
 * \file main.c
 * \brief This program test Reception Misures in a WSN Network and provides code
to read the sensors on remote nodes.
 *
 */

/* Standard includes. */
#include <stdlib.h>
#include <string.h>
#include <sys/inttypes.h>


/* Scheduler includes. */
#include "FreeRTOS.h"
#include "task.h"
#include "queue.h"

/* NanoStack includes */
#include "socket.h"
#include "debug.h"
#include "ssi.h"
```

```
#include "control_message.h"

/* Platform includes */
#include "uart.h"
#include "rf.h"
#include "bus.h"
#include "dma.h"
#include "timer.h"
#include "gpio.h"
#include "adc.h"

#include "neighbor_routing_table.h"


/* Message types */
#define REQUEST 0x50
#define RISPONSE 0x51
#define CONF 0x52

/*Control Measures*/
#define XTIME 1000 //X * 1000 = X000 Ms


static void vAppTask( int8_t *pvParameters );

int8_t get_adc_value(adc_input_t channel, uint16_t *value);


ssi_sensor_t ssi_sensor[] =
{/*  ID             | unit type        |scale|data|status*/
  {1, SSI_DATA_TYPE_INT,  0,   {0},   0},
  {2, SSI_DATA_TYPE_INT,  0,   {0},   0},
  {3, SSI_DATA_TYPE_INT,  0,   {0},   0}
};



const uint8_t *ssi_description[] =
{
  "Light",
  "Temp",
        "LEDs"
};

const uint8_t *ssi_unit[] =
{
        "RAW",
        "RAW",
        "xxxxxx21"
};




const uint8_t ssi_n_sensors = sizeof(ssi_sensor)/sizeof(ssi_sensor_t);





/* Setup a default address structure, short address, broadcast, to port 61619 */
sockaddr_t broadcast =
{
        ADDR_BROADCAST,
        { 0x00, 0x00, 0x00, 0x00,
        0x00, 0x00, 0x00, 0x00, 0x00, 0x00 },
        61619
};
```

```
sockaddr_t sa  = //Local sensor address
{
        ADDR_802_15_4_PAN_SHORT,
        { 0xFF, 0xFF, 0x3F, 0x12,
        0x02, 0x00, 0x00, 0x20, 0x15, 0x00 },
        61622
};



xQueueHandle button_events;

/*LED blink times*/
uint16_t led1_count;
uint16_t led2_count;


socket_t *broadcast_socket=0;
socket_t *app_socket;

/* Main task, initialize hardware and start the FreeRTOS scheduler */
int main( void )
{
        /* Initialize the Nano hardware */
        LED_INIT();
        bus_init();
        N710_SENSOR_INIT();


        /* Setup the application task and start the scheduler */
        xTaskCreate( vAppTask, "App", configMINIMAL_STACK_SIZE+200, NULL,
(tskIDLE_PRIORITY + 1 ), ( xTaskHandle * )NULL );
        vTaskStartScheduler();

        /* Scheduler has taken control, next vAppTask starts executing. */

        return 0;
}



discover_res_t echo_result;
stack_event_t stack_event;
int8_t ping_active=0;
portTickType ping_start = 0;






/**
 * Application task
 */
static void vAppTask( int8_t *pvParameters )
{



        uint8_t event;
        uint8_t buttons = 0;

        int16_t byte, time, exit;


        buffer_t *buf, *buf_receive, *buf_receivetem;
        uint8_t ind = 0, sending_request = 0, activa_temp = 0, length = 0,
header = 0, i= 0,remoto = 0;
        uint16_t rec_start = 0, count_rec = 0,count_env = 0;

        uint32_t sum = 0;

        uint16_t date_request = 0, luce = 0, temp = 0;
        uint16_t U1_value = 0,var1 = 0;
        uint16_t U2_value = 0,var2 = 0;
        uint8_t count = 0, tx_power = 100;

        stack_init_t *stack_rules=0;

        uint8_t  primeravez = 0;

        pvParameters;




        N710_BUTTON_INIT();

        /* Start the debug UART at 115k */
        debug_init(115200);
        button_events = xQueueCreate( 4, 1 /*message size one byte*/ );

        led1_count = 50;
        led2_count = 100;

        vTaskDelay( 50 / portTICK_RATE_MS );
        /* Start the debug UART at 115k */
        vTaskDelay( 200 / portTICK_RATE_MS );

        /* Initialize NanoStack with default parameters, NanoStack task
automatically created. */
        {
                if(stack_start(NULL)==START_SUCCESS)
                {
                        debug("            NanoStack Start Ok\r\n");
                        debug("   EMISORE UNICAST. TX TEST. Versione
1.0\r\n\r\n");
                }
        }

        LED1_ON();
        vTaskDelay( 500 / portTICK_RATE_MS );
        LED1_OFF();

        stack_event             = open_stack_event_bus();               /* Open
socket for stack status message */
        stack_service_init( stack_event,NULL, 0 , NULL );       /* No Gateway
discover */


        //Open and bind Broadcast socket
        broadcast_socket = socket(MODULE_CUDP, 0);
        if (broadcast_socket) {
                if (socket_bind(broadcast_socket, &broadcast) != pdTRUE)
                {
                        debug("Socket bind Send1 failed.\r\n");
                }
                else {

                        debug("Open and bind Send s1 socket\r\n");
                }
        }
        /*Open and bind local socket*/
        app_socket = socket(MODULE_CUDP, 0);
        if (app_socket) {
                if (socket_bind(app_socket, &sa) != pdTRUE)
                {
                        debug("Socket bind receive failed.\r\n");
                }
                else {

                        debug("Open and bind receive socket\r\n");
                }
        }





        /* Start an endless task loop, we must sleep most of the time allowing
execution of other tasks. */
        for (;;)
        {


                /* Sleep for 1000 ms */
                vTaskDelay( 1000 / portTICK_RATE_MS );

/*******************************************************************************
***********************/
        /* Sleep for 10 ms or received from UART */
                byte = debug_read_blocking(10 / portTICK_RATE_MS);
                if (byte != -1)
                {
                        switch(byte)
                        {

                                //Start remote power configuration.
                                case 'x':
                                        if(remoto == 1){

                                                remoto = 0;
                                        }else{

                                                        remoto = 1;
                                                        debug("\r\nSending CONF
packet");
                                                }

                                        break;


                                //Shows console
                                case 'h':
                                        debug("********** \r\n");
                                        debug("Shell help:\r\n1 - Start listen
process\r\n2 - Finish listen process\r\n");
                                        debug("\r\n+ - Power up 10\r\n- - Power
down 10");
                                        debug("\r\n********** \r\n");

                                        break;


                                //Start Request process; Sends Request Packets
to Remote node. Includes a routine time entered by the user.
                                case '1':

                                        if(sending_request == 0){
                                                LED1_ON();
                                                LED2_ON();


                                                time = 0;
                                                debug("\r\nINSERT WAITING TIME
in sec. (0,9]: \r\n");

                                                LED1_ON();
                                                LED2_ON();

                                                time = debug_read_blocking(10000
/ portTICK_RATE_MS);
                                                time = time - 48;  //ASCII to
decimal.

                                                if (time >= 0 && time < 10 ){

                                                        debug_int(time);

                                                        time = time * XTIME;
                                                        time = time - 1000;

                                                        debug_int(time);
                                                        LED1_OFF();
                                                        LED2_OFF();

                                                        debug("\r\nMode Sending
Request.");
                                                        debug("\r\n");

                                                        LED1_ON();
                                                        LED2_ON();

                                                        sending_request = 1;


                                                }else{
                                                        debug("Error introducted
time");
                                                        time = 0;

                                                }
                                        break;


                                //Local Tx power configuration
                                case '+':

                                        if(tx_power==100){
                                                debug("Max Tx power set
up.\r\n");

                                        }else{
                                                        tx_power += 25;
                                                        rf_power_set(tx_power);
                                                        debug("Current power ");
                                                        debug_int(tx_power);
                                                        debug("\r\n");
                                                }
                                                break;

                                case '-':
                                        if(tx_power==25){
                                                debug("Min Tx power set up
25.\r\n");
```

```
                                        }else{
                                                        tx_power -= 25;
                                                        rf_power_set(tx_power);
                                                        debug("Current power ");
                                                        debug_int(tx_power);
                                                        debug("\r\n");
                                                }
                                                break;


                                case '\r':
                                        debug("\r\n");
                                        break;

                                //Prints Local mac
                                case 'm':
                                        {
                                                sockaddr_t mac;

                                                rf_mac_get(&mac);

                                                debug("MAC: ");
                                                debug_address(&mac);
                                                debug("\r\n");
                                        }
                                        break;


                                //Start Ping Process
                                case 'p':
                                        if(ping_active == 0)
                                        {
                                                echo_result.count=0;
                                                if(ping(NULL, &echo_result) ==
pdTRUE) /* Broadcast */
                                                {
                                                        ping_start =

xTaskGetTickCount();
                                                        ping_active = 2;
                                                        debug("Ping\r\n");
                                                }
                                                else
                                                        debug("No buffer.\r\n");
                                        }
                                        break;

                                case 'u':
                                        if(ping_active == 0)
                                        {
                                                echo_result.count=0;
                                                if(udp_echo(NULL, &echo_result)
== pdTRUE)
                                                {
                                                        ping_start =

xTaskGetTickCount();
                                                        ping_active = 1;
                                                        debug("udp

echo_req()\r\n");
                                                }
                                                else
                                                        debug("No buffer.\r\n");
                                        }
                                        break;

                                default:
                                        debug_put(byte);
                                        break;
```

```
                        }
                }

                //Time and recived loop for Sending Recived process
                        if(activa_temp == 1){

                                rec_start = xTaskGetTickCount();

                                debug("\r\n-----------\r\n");

                                while(activa_temp == 1){

                                        //Loop for Reading packets
                                        buf_receivetem = socket_read(app_socket,
500);
                                        if(buf_receivetem){     //Arriva il
messagge.
                                                ind = 0;
                                                ind = buf_receivetem->buf_ptr;
                                                length = buf_receivetem->buf_end
- buf_receivetem->buf_ptr;
                                                header =
buf_receivetem->buf[ind++];

                                                if(header == RISPONSE){
                                                        count_rec ++;
                                                        debug("\r\n");


                                                        for(i=0;i<4;i++){

debug_hex(buf_receivetem->src_sa.address[9-i]);
                                                                if(i!=3){

debug(":");

                                                                }
                                                        }

                                                        //Reconstruction of
Remote Light data
                                                        var1 =
buf_receivetem->buf[ind++];
                                                        var2 =
buf_receivetem->buf[ind++];
                                                        var1 = var1 << 8;
                                                        var1 = var1 + var2;

                                                        luce = var1;

                                                        //Reconstruction of
Remote Temperature data
                                                        var1 =
buf_receivetem->buf[ind++];
                                                        var2 =
buf_receivetem->buf[ind++];
                                                        var1 = var1 << 8;
                                                        var1 = var1 + var2;

                                                        temp = var1;

                                                        var1 = var2 = 0;

                                                        sum = (uint32_t) temp;

                                                        //Conversion of
temperature data in Celsius
                                                        sum *= 122;
                                                        sum /= 10000;
```

```
                                                  sum -= 68;

                                                  temp = (uint16_t) sum;



                                                  debug("\r\nTemperature
ºC: ");

                                                  debug_int(temp);
                                                  debug("Luce: ");
                                                  debug_int(luce);






                                                  debug("\r\n\r\n");

stack_buffer_free(buf_receivetem);

                                          }
                                  }


                                  if (((xTaskGetTickCount() -
rec_start)*portTICK_RATE_MS) >  time ){


                                                  i = 0;
                                                  sending_request = 1;
                                                  activa_temp = 0;
                                                  LED1_OFF();
                                                  LED2_OFF();

debug("\r\n-----------\r\n");


                                  }

                          }

                          debug("Request Send: ");
                          debug_int(count_env);
                          debug(" Recived ok: ");
                          debug_int(count_rec);
                          debug("\r\n");
                  }





                ////////////////////////////////////
                //      Code for received messages    //
                ////////////////////////////////////


                  buf_receive = socket_read(app_socket, 10);

                  if(buf_receive){        //Message Arrived

                          ind = 0;
                          ind = buf_receive->buf_ptr;
                          length = buf_receive->buf_end -
buf_receive->buf_ptr;

                          header = buf_receive->buf[ind++];

                          //      Reading the Head of Message
```

```
                                if(header == RISPONSE){ //Arrived RISPONSE
message.
                                                activa_temp = 1;

                                }else{
                                        debug("\r\nArrived Risponse out of
Time\r\n");
                                }
                                stack_buffer_free(buf_receive);
                        }



                ////////////////////////////////////
                //         Code for Send messages    //
                ////////////////////////////////////

                if((sending_request == 1)||(remoto == 1)){


                        //Construction of messages
                        buf = socket_buffer_get(broadcast_socket);

                        if (buf) {
                                buf->buf_end=0;
                                buf->buf_ptr=0;
                                buf->options.hop_count = 1;// Hop = 1, Star
Topology, only one hop.

                                //Construction of Power configuration messages
                                if(remoto == 1){
                                        buf->buf[buf->buf_end++] = CONF;
                                        buf->buf[buf->buf_end++] = tx_power;
                                        debug("CONF ");
                                        remoto = 0;
                                        //Solo un unico invio di request
                                }

                                //Construction of REQUEST messages
                                if(sending_request == 1){

                                        debug("REQ");
                                        buf->buf[buf->buf_end++] = REQUEST;
                                        activa_temp = 1;

                                        //Solo un unico invio di request

                                }


                                if (socket_sendto(broadcast_socket, &broadcast,
buf) == pdTRUE) {

                                                if(sending_request == 1){
                                                        count_env++;
                                                        sending_request = 0;
                                                }

                                                debug("\r\nSend OK");

                                        }else{
                                                debug("\r\nSEND FAILED\r\n");
                                        }

                        }else{
                                debug("\r\nError: socket_buffer_get(). Any
buffer created\r\n");
                        }
                }



                /* ping response handling */
                if ((xTaskGetTickCount() - ping_start)*portTICK_RATE_MS > 1000
&& ping_active)
                {
                        debug("Ping timeout.\r\n");
                        stop_ping();
                        if(echo_result.count)
                        {
                                debug("Response: \r\n");
                                for(i=0; i<echo_result.count; i++)
                                {

debug_address(&(echo_result.result[i].src));
                                        debug(" ");
                                        debug_int(echo_result.result[i].rssi);
                                        debug(" dbm, ");
                                        debug_int(echo_result.result[i].time);
                                        debug(" ms\r\n");
                                }
                                echo_result.count=0;

                        }
                        else
                        {
                                debug("No response.\r\n");
                        }
                        ping_active = 0;
                }

                /* stack events */
                if(stack_event)
                {
                        buffer_t *buffer = waiting_stack_event(10);
                        if(buffer)
                        {
                                switch (parse_event_message(buffer))
                                {
                                        case BROKEN_LINK:
                                                debug("Route broken to ");
                                                debug("\r\n");

debug_address(&(buffer->dst_sa));
                                                debug("\r\n");
                                                break;

                                        case NO_ROUTE_TO_DESTINATION:
                                                debug("ICMP message back, no
route ");
                                                debug("\r\n");

debug_address(&(buffer->dst_sa));
                                                debug("\r\n");
                                                break;

                                        case TOO_LONG_PACKET:
                                                debug("Payload Too Length\r\n");
                                                break;

                                        case DATA_BACK_NO_ROUTE:
                                                debug("DATA back, No route");
                                                debug("\r\n");
                                                debug("To ");

debug_address(&(buffer->dst_sa));
```

```
                                                debug("\r\n");
                                                break;

                                        default:

                                                break;
                                }
                                if(buffer)
                                {
                                        socket_buffer_free(buffer);
                                        buffer = 0;
                                }
                        }
                } /*end stack events*/
        } /*end main loop*/
}


int8_t get_adc_value(adc_input_t channel, uint16_t *value)
{
        int8_t retval;

        if (adc_convert_single(channel, ADCREF_AVDD, ADCRES_14BIT) == 0)
        {
                retval = 0;
                while (retval != 1)
                {
                        retval = adc_result_single(value);
                }
                retval = 0;
        }
        else
        {
                retval = -1;
        }

        return retval;
}
```