```
/*

                  ///PROBA DI RECEZIONE. PROGRAMA TEST BROADCAST.\\\
                                                        + Pablo Meca Calderón +
                                                      BROADCAST 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 Example bare skeleton for starting a new Nano series application.
 *
 */

/* 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  = //direccion local del sensor
{
        ADDR_802_15_4_PAN_SHORT,
        { 0xFF, 0xFF, 0x7B, 0x10,
        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;






/**
 * Skeleton application task
 */
static void vAppTask( int8_t *pvParameters )
{

        uint8_t event;
        uint8_t buttons = 0;
        uint8_t s1_count = 0;
        uint8_t s2_count = 0;
        int16_t byte, time;
        uint8_t i =0,a =0, pos = 0;
        uint8_t channel;
        buffer_t *buf, *buf_receive, *buf_receivetem;
        uint8_t length = 0;
```

```
        uint16_t date_request = 0;
        uint16_t U1_value = 0,var1 = 0;
        uint16_t U2_value = 0,var2 = 0,tx_power =100,sum = 0;
        uint8_t count1 = 0, count2= 0, count3= 0, count_env= 0, countpar=0;

        stack_init_t *stack_rules=0;

        uint8_t ind=0,
header=0,sending_request=0,waiting_risponse=0,invio_risponse=0, activa_temp = 0,
option = 0, select_socket=0, remoto = 0;
        uint16_t temp;
        uint16_t luce;
        uint16_t rec_start=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 BROADCAST. 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 */

        channel = mac_current_channel();



        //Open and bind a socket1 send Broadcast
        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 a socket app*/
	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( 100 / portTICK_RATE_MS );

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

				case 'x'://configura los nodos remotos con la
misma potencia que el local.

					if(remoto == 1){

						remoto = 0;
					}else{

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

					break;


				case 'h':
					debug("********** \r\n");
					debug("Shell help:\r\n1 - Start listen
process\r\n2 - Finish listen process\r\n");
					debug("\r\np - Start ping process\r\nu -
Start udp echo_req()");
					debug("\r\n********** \r\n");

					break;


				case '1':

					if(waiting_risponse == 0 &&
sending_request == 0 && invio_risponse == 0){
```

```
                                        LED1_ON();
                                        LED2_ON();


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

                                        LED1_ON();
                                        LED2_ON();

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

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

                                                //debug_int(time);

                                                time = time * XTIME;


                                                //debug_int(time);

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

                                                sending_request = 1;


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

                                        }
                                   }else{
                                        debug("\r\nTo finish Receive
information Mode. push '2'.\r\n");
                                   }

                                   break;


                              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;

                case 'm':
                        {
                                sockaddr_t mac;

                                rf_mac_get(&mac);

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

                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;

                case 'C':
                        if (channel < 26) channel++;
                        channel++;
                case 'c':
                        if (channel > 11) channel--;
                        mac_set_channel(channel);
                        debug("Channel: ");
                        debug_int(channel);
```

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

            default:
                debug_put(byte);
                break;
        }
    }


    ////////////////////////////////////
    //Codize di recezione dei messaggi//
    ////////////////////////////////////


    if(activa_temp == 1){
        rec_start = xTaskGetTickCount();
    }
    while(activa_temp == 1){

        buf_receive = socket_read(app_socket, 10);


        if(buf_receive){        //Arriva il messagge.

            debug("Estoy en buf_receive");
            ind = 0;
            ind = buf_receive->buf_ptr;
            length = buf_receive->buf_end -
buf_receive->buf_ptr;
            header = buf_receive->buf[ind++];

            //Leggendo il HEAD del Messagge
            if(header == RISPONSE){ //Arriva RISPONSE,
guardo se arriva nel tempo corretto


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



                    //lettura dello arrivo
di packets


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

                    //Print Mac Address
                    for(i=0;i<4;i++){

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

debug(":");
                        }
                    }



                    //Ricostruzione del dato
di Temperatura.
                    var1 =
buf_receive->buf[ind++];
                    var2 =
buf_receive->buf[ind++];
                    var1 = var1 << 8;
                    var1 = var1 + var2;
```

```
                                                        luce = var1;
                                                        //Ricostruzione del dato
di Luce.
                                                        var1 =
buf_receive->buf[ind++];
                                                        var2 =
buf_receive->buf[ind++];
                                                        var1 = var1 << 8;
                                                        var1 = var1 + var2;
                                                        temp = var1;
                                                        var1 = var2 = 0;
                                                        sum = (uint32_t) temp;
                                                        /*Visualizzazione dei
dati di temperatura in Celsius.*/
                                                        sum *= 122;
                                                        sum /= 10000;
                                                        sum -= 68;
                                                        temp = (uint16_t) sum;


                                                        debug("\r\nTemp C: ");
                                                        debug_int(temp);
                                                        debug("Luce: ");
                                                        debug_int(luce);


                                                        //contador segun recibo
del 1 , 2, o 3
switch(buf_receive->src_sa.address[6]){
                                                                case '0x8E':
count1++;
                                                                        break;
                                                                case '0xE9':
count2++;
                                                                        break;
                                                                case '0x44':
count2++;
                                                                        break;
                                                                default:
                                                                        break;
                                                        }
                                                        debug("\r\n\r\n");
stack_buffer_free(buf_receive);
                                }//Si recivo RISPONSE
                                stack_buffer_free(buf_receive);
                        }//si recivo algo
```

```
                        if(count_env == 50){
                                i = 0;
                                sending_request = 0;
                                activa_temp = 0;
                                count_env=count1=count2=count3=0;
                        }



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

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



                                debug("ReqSend:");
                                debug_int(count_env);
                                debug("\r\nArriv 8E:");
                                debug_int(count1);
                                debug(" Arriv E9:");
                                debug_int(count2);
                                debug(" Arriv 44:");
                                debug_int(count3);
                                debug("\r\n");


                        }


                }//activa_temp




                ////////////////////////////////
                //Codize dell'invio dei messaggi.//
                ////////////////////////////////

                if((sending_request == 1)||(remoto == 1)){ //debo fare l'invio
                        //construzione comune ai due packets
                        debug("\r\n Ento a enviar\r\n");
                        buf = socket_buffer_get(broadcast_socket);
                        debug("\r\n Ento a 2\r\n");

                        if (buf) {
                                buf->buf_end=0;
                                buf->buf_ptr=0;
                                buf->options.hop_count = 2;// Hop = 1, perche in
queste essempio è la massima distanza posibile.

                                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
```

```
                                }


                                if(sending_request == 1){ //construzione del
packet REQUEST

                                        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) {//invio dei packets.

                                                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;
                                /*¿Como fare per liberare la memoria di una
variabile tipo echo_result?*/
                        }
                        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;
}
```