```
/*


                      ///PROBA DI RECEZIONE. PROGRAMA TEST UNICAST.\\\
                                                 Università degli Studi di Pavia
                                                        + Pablo Meca Calderón +
                                                       UNICAST RECEIVER
                                                                 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, to port 61622 */


sockaddr_t dirsens =
{
        ADDR_802_15_4_PAN_SHORT,
        { 0x00, 0x00, 0x00, 0x00,
        0x00, 0x00, 0x00, 0x00, 0x00, 0x00 },
        61622
};


sockaddr_t sa  =
{
        ADDR_802_15_4_PAN_SHORT,
        { 0x00, 0x00, 0x00, 0x00,
        0x00, 0x00, 0x00, 0x00, 0x00, 0x00 },
        61619
};

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;
        uint8_t s1_count = 0;
        uint8_t s2_count = 0;
        int16_t byte, time;
        uint8_t i =0,pos = 0;
        uint8_t channel;
        buffer_t *buf, *buf_receive;
```

```c
        uint8_t length = 0, tx_power = 100;

        uint16_t date_request = 0;
        uint16_t U1_value = 0,var1 = 0;
        uint16_t U2_value = 0,var2 = 0;
        uint8_t count = 0;

        stack_init_t *stack_rules=0;

        uint8_t ind=0, header=0,sending_request=0,invio_risponse=0, activa_temp
= 0, option = 0,primeravez = 0;
        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("   RECEPTORE 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 a socket send UNICAST*/
        broadcast_socket = socket(MODULE_CUDP, 0);
        if (broadcast_socket) {
                if (socket_bind(broadcast_socket, &dirsens) != pdTRUE)
                {
                        debug("Socket bind Send failed.\r\n");
                }
                else {

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

        /* Open and bind a socket receive Port 61620 */
        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)
                        {

                                case 'h':
                                        debug("********** \r\n");
                                        debug("Shell help:\r\n");
                                        debug("\r\n+ - Power up 10\r\n- - Power 
down 10");

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

                                        break;

                                case '?':
                                        debug("\r\nCurrent power: ")
                                        debug_int(tx_power);
                                        debug("\r\n");
                                        break;

                                case '\r':
                                        debug("\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 
10.\r\n");


                                        }else{
                                                        tx_power -= 25;
                                                        rf_power_set(tx_power);
                                                        debug("Current power ");
                                                        debug_int(tx_power);
                                                        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;

                                default:
                                        debug_put(byte);
                                        break;
                        }
                }


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

                if(invio_risponse == 0){

                        buf_receive = socket_read(app_socket, 100);
                        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++];

                                //Read Messagge Head
                                if(header == REQUEST){

                                        //If "invio_risponse == 1" can not 
receive messages Request
                                        if(invio_risponse == 0){



                                                debug("\r\nREQUEST ARRIVED");
                                                invio_risponse = 1;
                                                debug("\r\nMode Sending 
Measure");


                                        }else{
                                                debug("\r\nRefused REQUEST 
PACKET\r\n");
                                        }
                                }else{
                                        debug("\r\nNo Req mesagge\r\n");

                                }

                                if(header == CONF){
                                        tx_power = buf_receive->buf[ind++];
                                        rf_power_set(tx_power);

                                }


                                stack_buffer_free(buf_receive);
                        }

                }else{
                        debug("\r\nNo entro a recibir\r\n");
                }




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

                if(invio_risponse == 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, perche in 
queste essempio è la massima distanza posibile.

                                //Construction of packet RISPONSE
                                //Read Sensor data.
                                if(get_adc_value(N710_LIGHT, &U1_value) == 0){

                                        if(get_adc_value(N710_TEMP, &U2_value)
== 0){

                                                debug("\r\nCreating Packet
risponse");
                                                buf->buf[buf->buf_end++] =
RISPONSE;


                                                buf->buf[buf->buf_end++] =
(U1_value >> 8);
                                                buf->buf[buf->buf_end++] =
(uint8_t) U1_value;

                                                buf->buf[buf->buf_end++] =
(U2_value >> 8);
                                                buf->buf[buf->buf_end++] =
(uint8_t) U2_value;

                                                invio_risponse = 0;

                                                }else{
                                                        debug("\r\nError: Failed
Read Sensor Measure\r\n");
                                                }


                                        }else{
                                                debug("\r\nError: Failed Read
Sensor Measure\r\n");
                                        }




                                if (socket_sendto(broadcast_socket, &dirsens,
buf) == pdTRUE) {
                                        debug("\r\nSend OK -1sec Sleep-");




                                }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;
}
```