RĪGAS TEHNISKĀ UNIVERSITĀTE

ELEKTRONIKAS UN TELEKOMUNIKĀCIJU FAKULTĀTE

Datu pārraide bezvadu sensoru tīklos

Laboratorijas darbs Nr.3

I REGV0

Ruslans Babajans 171REB152

Rīga, 2021

Darba uzdevums:

Laboratorijas darbā tiek modificēts kods no 2. laboratorijas darba, lai iekārta izsūtītu ziņu tikai tad, kad tiek nospiesta poga. Kods nosūta pogas nospiešanas reižu skaitu. Uztvērējs parāda ziņas saturu Termite programmā.

Izmantotā aparatūra: DISCO-L072CZ-LRWAN1 izstrādes platforma.

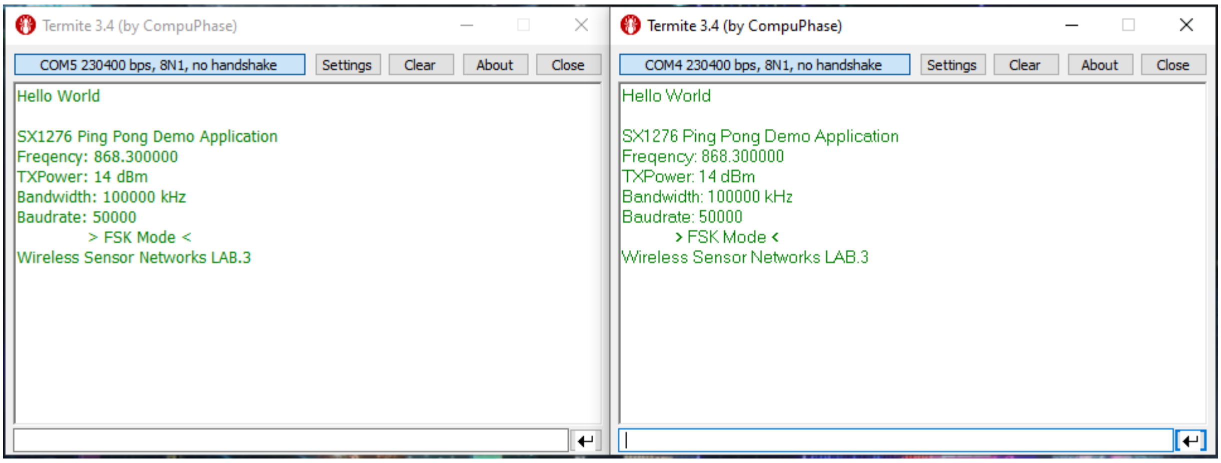
Izstrādātais c++ kods:

```
#include "mbed.h"
#include "PinMap.h"
#include "GenericPingPong.h"
#include "sx1276-mbed-hal.h"
#include "main.h"
#ifdef FEATURE_LORA
/* Set this flag to '1' to display debug messages on the console */
#define DEBUG_MESSAGE   1
/* Set this flag to '1' to use the LoRa modulation or to '0' to use FSK modulation */
#define USE_MODEM_LORA  0
#define USE_MODEM_FSK   !USE_MODEM_LORA
#define RF_FREQUENCY            868300000 // Hz
#define TX_OUTPUT_POWER         14                // 14 dBm
#if USE_MODEM_LORA == 1
#define LORA_BANDWIDTH          125000  // LoRa default, details in SX1276::BandwidthMap
#define LORA_SPREADING_FACTOR   LORA_SF7
#define LORA_CODINGRATE         LORA_ERROR_CODING_RATE_4_5
#define LORA_PREAMBLE_LENGTH    8       // Same for Tx and Rx
#define LORA_SYMBOL_TIMEOUT     5       // Symbols
#define LORA_FIX_LENGTH_PAYLOAD_ON  false
#define LORA_FHSS_ENABLED       false
#define LORA_NB_SYMB_HOP        4
#define LORA_IQ_INVERSION_ON    false
#define LORA_CRC_ENABLED        true
#elif USE_MODEM_FSK == 1
#define FSK_FDEV                25000     // Hz                     H=1
#define FSK_DATARATE            50000     // bps
#define FSK_BANDWIDTH           100000    // Hz                      TX_BW
#define FSK_AFC_BANDWIDTH       103473    // Hz                      RX_BW
#define FSK_PREAMBLE_LENGTH     5         // Same for Tx and Rx
#define FSK_FIX_LENGTH_PAYLOAD_ON   false
#define FSK_CRC_ENABLED         true
#else
   #error "Please define a modem in the compiler options."
#endif

#define RX_TIMEOUT_VALUE    0   // in ms
```

```
//#define BUFFER_SIZE       32        // Define the payload size here
#define BUFFER_SIZE         64        // Define the payload size here
/*
 *  Global variables declarations
 */
typedef enum
{
    IDLE,

    RX_COMPLETE,
    RX_TIMEOUT,
    RX_ERROR,

    TX_START,
    TX_WAITING_COMPLETE,
    TX_COMPLETE,
    TX_TIMEOUT,

} AppStates_t;
volatile AppStates_t State = IDLE;
/*!
 * Radio events function pointer
 */
static RadioEvents_t RadioEvents;
/*
 *  Global variables declarations
 */
SX1276Generic *Radio;
uint16_t rxBufferSize; // Stoer las tRx message length in this global variable
uint8_t rxBuffer[BUFFER_SIZE+1];
uint16_t txBufferSize;
uint8_t txBuffer[BUFFER_SIZE+1];
uint32_t buttonPressCounter;
InterruptIn mybutton(USER_BUTTON);
void myButtonInterruptFunction(){
                if(State == IDLE){
                                State = TX_START;
                                buttonPressCounter++;
                }
}
int SX1276PingPong()
{
                mybutton.fall(&myButtonInterruptFunction);
    Radio = new SX1276Generic(NULL, MURATA_SX1276,
                                LORA_SPI_MOSI, LORA_SPI_MISO, LORA_SPI_SCLK, LORA_CS, LORA_RESET,
            LORA_DIO0, LORA_DIO1, LORA_DIO2, LORA_DIO3, LORA_DIO4, LORA_DIO5,
            LORA_ANT_RX, LORA_ANT_TX, LORA_ANT_BOOST, LORA_TCXO);
    uint8_t i;
    dprintf("SX1276 Ping Pong Demo Application" );
    dprintf("Freqency: %.6f", (double)RF_FREQUENCY/1000000.0);
    dprintf("TXPower: %d dBm",  TX_OUTPUT_POWER);
#if USE_MODEM_LORA == 1
    dprintf("Bandwidth: %d Hz", LORA_BANDWIDTH);
    dprintf("Spreading factor: SF%d", LORA_SPREADING_FACTOR);
#elif USE_MODEM_FSK == 1
    dprintf("Bandwidth: %d kHz",  FSK_BANDWIDTH);
    dprintf("Baudrate: %d", FSK_DATARATE);
#endif
    // Initialize Radio driver
    RadioEvents.TxDone = OnTxDone;
    RadioEvents.RxDone = OnRxDone;
    RadioEvents.RxError = OnRxError;
    RadioEvents.TxTimeout = OnTxTimeout;
    RadioEvents.RxTimeout = OnRxTimeout;
    if (Radio->Init( &RadioEvents ) == false) {
        while(1) {
            dprintf("Radio could not be detected!");
            wait( 1 );
        }
    }
    Radio->SetChannel(RF_FREQUENCY );
#if USE_MODEM_LORA == 1
    if (LORA_FHSS_ENABLED)
        dprintf("             > LORA FHSS Mode <");
    if (!LORA_FHSS_ENABLED)
        dprintf("             > LORA Mode <");
    Radio->SetTxConfig( MODEM_LORA, TX_OUTPUT_POWER, 0, LORA_BANDWIDTH,
                         LORA_SPREADING_FACTOR, LORA_CODINGRATE,
                         LORA_PREAMBLE_LENGTH, LORA_FIX_LENGTH_PAYLOAD_ON,
                         LORA_CRC_ENABLED, LORA_FHSS_ENABLED, LORA_NB_SYMB_HOP,
                         LORA_IQ_INVERSION_ON, 2000 );
    Radio->SetRxConfig( MODEM_LORA, LORA_BANDWIDTH, LORA_SPREADING_FACTOR,
                         LORA_CODINGRATE, 0, LORA_PREAMBLE_LENGTH,
                         LORA_SYMBOL_TIMEOUT, LORA_FIX_LENGTH_PAYLOAD_ON, 0,
                         LORA_CRC_ENABLED, LORA_FHSS_ENABLED, LORA_NB_SYMB_HOP,
                         LORA_IQ_INVERSION_ON, true );

#elif USE_MODEM_FSK == 1
    dprintf("             > FSK Mode <");
    Radio->SetTxConfig( MODEM_FSK, TX_OUTPUT_POWER, FSK_FDEV, 0,
                        FSK_DATARATE, 0,
                        FSK_PREAMBLE_LENGTH, FSK_FIX_LENGTH_PAYLOAD_ON,
                        FSK_CRC_ENABLED, 0, 0, 0, 2000 );
    Radio->SetRxConfig( MODEM_FSK, FSK_BANDWIDTH, FSK_DATARATE,
                        0, FSK_AFC_BANDWIDTH, FSK_PREAMBLE_LENGTH,
                        0, FSK_FIX_LENGTH_PAYLOAD_ON, 0, FSK_CRC_ENABLED,
                        0, 0, false, true );

#else
#error "Please define a modem in the compiler options."
#endif
        dprintf("Wireless Sensor Networks LAB.3");
    Radio->Rx( RX_TIMEOUT_VALUE );
    while( 1 )
    {
#ifdef TARGET_STM32L4
        WatchDogUpdate();
#endif

        switch( State )
        {
        case IDLE:
                // Do nothing - wait for button interrupt
                sleep();
                break;
        case RX_COMPLETE:
                Radio->Rx( RX_TIMEOUT_VALUE ); // Put transceiver back to Rx
                rxBuffer[rxBufferSize]=0;     // For safety add Null Terminating character
                dprintf("Rx complete, data is %s\n",rxBuffer);
            State = IDLE;
            break;
        case RX_TIMEOUT:
                Radio->Rx( RX_TIMEOUT_VALUE ); // Put transceiver back to Rx
                dprintf("Rx Timeout happened\n");
                State = IDLE;
                break;
        case RX_ERROR:
                Radio->Rx( RX_TIMEOUT_VALUE ); // Put transceiver back to Rx
                dprintf("Rx CRC Error happened\n");
                State = IDLE;
                break;
        case TX_START:
                txBufferSize=sprintf((char*)txBuffer, "Hello! buttonPressCounter=%d", buttonPressCounter);
                txBufferSize++; // Add null terminated character to length
                Radio->Sleep( ); // First we need to put chip from Rx to Sleep state
                Radio->Send( txBuffer, txBufferSize );
                dprintf("Message sent! buttonPressCounter was %d", buttonPressCounter);
                State = TX_WAITING_COMPLETE;
                break;
        case TX_WAITING_COMPLETE:
                                                sleep();
            break;
        case TX_COMPLETE:
                dprintf("Message sending complete!");
                Radio->Rx( RX_TIMEOUT_VALUE ); // Put transceiver back to Rx
                State = IDLE;
                break;
        case TX_TIMEOUT:
            dprintf("TX_TIMEOUT happened!");
                Radio->Rx( RX_TIMEOUT_VALUE ); // Put transceiver back to Rx
                State = IDLE;
                break;
        default:
            State = IDLE;
            break;
        }
    }
}
void OnTxDone(void *radio, void *userThisPtr, void *userData)
{
    Radio->Sleep( );
    State = TX_COMPLETE;
    if (DEBUG_MESSAGE)
        dprintf("> OnTxDone");
}
void OnRxDone(void *radio, void *userThisPtr, void *userData, uint8_t *payload, uint16_t size, int16_t rssi, int8_t snr)
{
    Radio->Sleep( );
    rxBufferSize = size;
    memcpy( rxBuffer, payload, rxBufferSize );
    State = RX_COMPLETE;
    if (DEBUG_MESSAGE)
        dprintf("> OnRxDone: RssiValue=%d dBm, SnrValue=%d", rssi, snr);
    dump("Data:", payload, size);
}
void OnTxTimeout(void *radio, void *userThisPtr, void *userData)
{
    Radio->Sleep( );
    State = TX_TIMEOUT;
    if(DEBUG_MESSAGE)
```

```
        dprintf("> OnTxTimeout");
}
void OnRxTimeout(void *radio, void *userThisPtr, void *userData)
{
    Radio->Sleep( );
    State = RX_TIMEOUT;
    if (DEBUG_MESSAGE)
        dprintf("> OnRxTimeout");
}
void OnRxError(void *radio, void *userThisPtr, void *userData)
{
    Radio->Sleep( );
    State = RX_ERROR;
    if (DEBUG_MESSAGE)
        dprintf("> OnRxError");
}
#endif
```

Rezultātā Termite terminālis parada sekojošas ziņas kad kods ir nokompilēts, ielādēts, un iekārtam ir nospiesta "reset" poga.



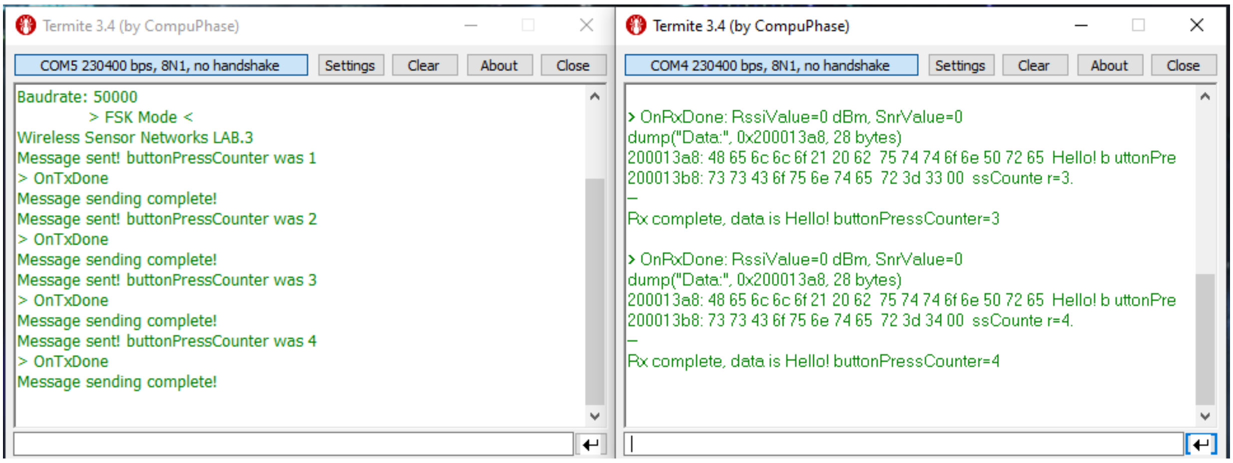
COM5 230400 bps, 8N1, no handshake

```
Hello World

SX1276 Ping Pong Demo Application
Freqency: 868.300000
TXPower: 14 dBm
Bandwidth: 100000 kHz
Baudrate: 50000
          > FSK Mode <
Wireless Sensor Networks LAB.3
```

COM4 230400 bps, 8N1, no handshake

```
Hello World

SX1276 Ping Pong Demo Application
Freqency: 868.300000
TXPower: 14 dBm
Bandwidth: 100000 kHz
Baudrate: 50000
          > FSK Mode <
Wireless Sensor Networks LAB.3
```

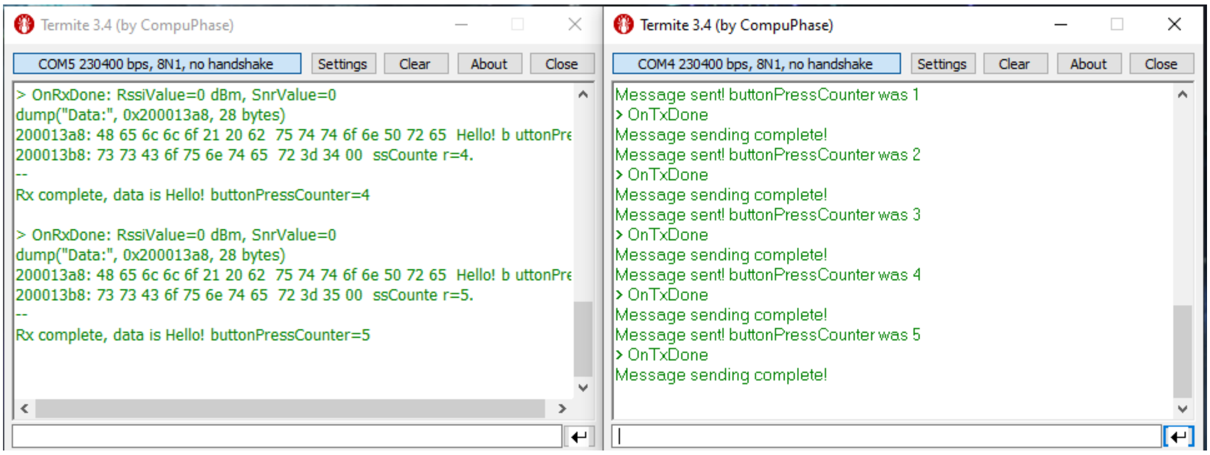
1. att. Ziņas no divām iekārtam


Termite 3.4 (by CompuPhase)
COM5 230400 bps, 8N1, no handshake
Settings
Clear
About
Close
Termite 3.4 (by CompuPhase)
COM4 230400 bps, 8N1, no handshake
Settings
Clear
About
Close

```
Baudrate: 50000
          > FSK Mode <
Wireless Sensor Networks LAB.3
Message sent! buttonPressCounter was 1
> OnTxDone
Message sending complete!
Message sent! buttonPressCounter was 2
> OnTxDone
Message sending complete!
Message sent! buttonPressCounter was 3
> OnTxDone
Message sending complete!
Message sent! buttonPressCounter was 4
> OnTxDone
Message sending complete!
```

```
> OnRxDone: RssiValue=0 dBm, SnrValue=0
dump("Data:", 0x200013a8, 28 bytes)
200013a8: 48 65 6c 6c 6f 21 20 62  75 74 74 6f 6e 50 72 65  Hello! b uttonPre
200013b8: 73 73 43 6f 75 6e 74 65  72 3d 33 00  ssCounte r=3.
--
Rx complete, data is Hello! buttonPressCounter=3

> OnRxDone: RssiValue=0 dBm, SnrValue=0
dump("Data:", 0x200013a8, 28 bytes)
200013a8: 48 65 6c 6c 6f 21 20 62  75 74 74 6f 6e 50 72 65  Hello! b uttonPre
200013b8: 73 73 43 6f 75 6e 74 65  72 3d 34 00  ssCounte r=4.
--
Rx complete, data is Hello! buttonPressCounter=4
```

2. att. Ziņas no divām iekārtam kad tiek nospiesta poga 4 reizes uz vienas izstrādes plates


Termite 3.4 (by CompuPhase)
COM5 230400 bps, 8N1, no handshake
Settings
Clear
About
Close
Termite 3.4 (by CompuPhase)
COM4 230400 bps, 8N1, no handshake
Settings
Clear
About
Close

```
> OnRxDone: RssiValue=0 dBm, SnrValue=0
dump("Data:", 0x200013a8, 28 bytes)
200013a8: 48 65 6c 6c 6f 21 20 62  75 74 74 6f 6e 50 72 65  Hello! b uttonPre
200013b8: 73 73 43 6f 75 6e 74 65  72 3d 34 00  ssCounte r=4.
--
Rx complete, data is Hello! buttonPressCounter=4

> OnRxDone: RssiValue=0 dBm, SnrValue=0
dump("Data:", 0x200013a8, 28 bytes)
200013a8: 48 65 6c 6c 6f 21 20 62  75 74 74 6f 6e 50 72 65  Hello! b uttonPre
200013b8: 73 73 43 6f 75 6e 74 65  72 3d 35 00  ssCounte r=5.
--
Rx complete, data is Hello! buttonPressCounter=5
```

```
Message sent! buttonPressCounter was 1
> OnTxDone
Message sending complete!
Message sent! buttonPressCounter was 2
> OnTxDone
Message sending complete!
Message sent! buttonPressCounter was 3
> OnTxDone
Message sending complete!
Message sent! buttonPressCounter was 4
> OnTxDone
Message sending complete!
Message sent! buttonPressCounter was 5
> OnTxDone
Message sending complete!
```

3. att. Ziņas no divām iekārtam kad tiek nospiesta poga 5 reizes uz otras izstrādes plates

Secinājumi:

Šajā darbā tika izveidots kods kas ļauj divām iekārtam izsūtīt ziņu tikai kad tiek nospiesta poga. Kods nosūta pogas nospiešanas reižu skaitu. Uztvērējs un raidītājs parāda ziņas saturu Termite programmā.