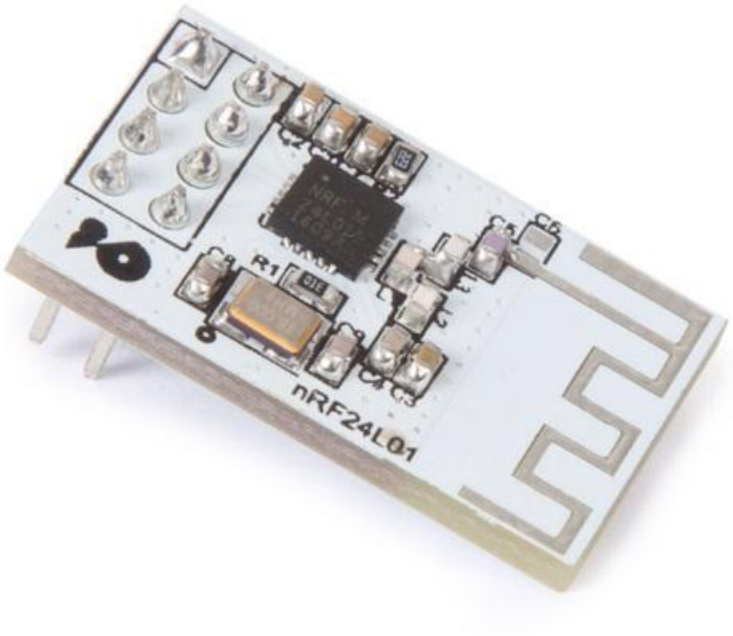
VMA322

2.4 GHZ NRF24L01 WIRELESS TRANCEIVER MODULE (2 PCS)



USER MANUAL

USER MANUAL

1. Introduction

To all residents of the European Union

Important environmental information about this product

This symbol on the device or the package indicates that disposal of the device after its lifecycle could harm the environment. Do not dispose of the unit (or batteries) as unsorted municipal waste; it should be taken to a specialized company for recycling. This device should be returned to your distributor or to a local recycling service. Respect the local environmental rules.

If in doubt, contact your local waste disposal authorities.

Thank you for choosing Velleman®! Please read the manual thoroughly before bringing this device into service. If the device was damaged in transit, do not install or use it and contact your dealer.

2. Safety Instructions

- This device can be used by children aged from 8 years and above, and persons with reduced physical, sensory or mental capabilities or lack of experience and knowledge if they have been given supervision or instruction concerning the use of the device in a safe way and understand the hazards involved. Children shall not play with the device. Cleaning and user maintenance shall not be made by children without supervision.

- Indoor use only.
  Keep away from rain, moisture, splashing and dripping liquids.

3. General Guidelines

- Refer to the Velleman® Service and Quality Warranty on the last pages of this manual.
- Familiarise yourself with the functions of the device before actually using it.
- All modifications of the device are forbidden for safety reasons. Damage caused by user modifications to the device is not covered by the warranty.
- Only use the device for its intended purpose. Using the device in an unauthorised way will void the warranty.
- Damage caused by disregard of certain guidelines in this manual is not covered by the warranty and the dealer will not accept responsibility for any ensuing defects or problems.
- Nor Velleman nv nor its dealers can be held responsible for any damage (extraordinary, incidental or indirect) – of any nature (financial, physical…) arising from the possession, use or failure of this product.
- Due to constant product improvements, the actual product appearance might differ from the shown images.
- Product images are for illustrative purposes only.
- Do not switch the device on immediately after it has been exposed to changes in temperature. Protect the device against damage by leaving it switched off until it has reached room temperature.
- Keep this manual for future reference.

4. What is Arduino®

Arduino® is an open-source prototyping platform based in easy-to-use hardware and software. Arduino® boards are able to read inputs – light-on sensor, a finger on a button or a Twitter message – and turn it into an output – activating of a motor, turning on an LED, publishing something online. You can tell your board what to do by sending a set of instructions to the microcontroller on the board. To do so, you use the Arduino programming language (based on Wiring) and the Arduino® software IDE (based on Processing).

Surf to www.arduino.cc and www.arduino.org for more information.

5. Overview

VMA322

This module based on Nordic nRF24L01, highly integrated, ultra-low power (ULP) 2 Mbps RF transceiver for the 2.4 GHz ISM (Industrial, Scientific and Medical) band. Nordic nRF24L01+ integrates a complete 2.4 GHz RF transceiver, RF synthesizer, and baseband logic including the Enhanced ShockBurst™ hardware protocol accelerator supporting a high-speed SPI interface for the application controller.

power supply: 1.9 to 3.6 V
IO port working voltage: 0-3.3 V
transmitting rate: +7 dB
receiving sensitivity: < 90 dB
transmission range: 250 m in open area
dimensions: 15 x 29 mm

6. Example

You will need two Arduino® boards and at least two RF modules, one to transmit and the other to receive. If you use a standard Arduino® board, you should use the Arduino® board's 3.3 V pin (VDD3V3) to provide power (the nRF24L01 module has 1.9-3.6 V power voltage level). Please note: do not use 5 V pin (VDD5V) to provide power as this may destroy the module.

VMA322	VMA100
GND	GND
VCC	3.3V
CS	D8
CSN	D9
SCK	D10
MOSI	D11
MISO	D12
IRQ	D13

Download the code below into the TX Arduino® (transmit). This code will drive the nRF24L01 module to send out data form 0×00 to 0xFF.

```
// Code Begin

#include "NRF24L01.h"

//***************************************************
#define TX_ADR_WIDTH    5   // 5 unsigned chars TX(RX) address width
#define TX_PLOAD_WIDTH  32  // 32 unsigned chars TX payload

unsigned char TX_ADDRESS[TX_ADR_WIDTH]  =
{
  0x34,0x43,0x10,0x10,0x01
}; // Define a static TX address

unsigned char rx_buf[TX_PLOAD_WIDTH] = {0}; // initialize value
unsigned char tx_buf[TX_PLOAD_WIDTH] = {0};
//***************************************************
void setup()
{
```

```
  SPI_DIR = ( CE + SCK + CSN + MOSI);
  SPI_DIR &=~ ( IRQ + MISO);
  //  attachInterrupt(1, _ISR, LOW);// interrupt enable
  Serial.begin(9600);
  init_io();                        // Initialize IO port
  unsigned char status=SPI_Read(STATUS);
  Serial.print("status = ");
  Serial.println(status,HEX);     // There is read the mode's status register, the default value should be 'E'
  Serial.println("*********************TX_Mode Start*****************************");
  TX_Mode();                      // set TX mode
}
void loop()
{
  int k = 0;
  for(;;)
  {
    for(int i=0; i<32; i++)
       tx_buf[i] = k++;
    unsigned char status = SPI_Read(STATUS);                // read register STATUS's value
    if(status&TX_DS)                                        // if receive data ready (TX_DS) interrupt
    {
      SPI_RW_Reg(FLUSH_TX,0);
      SPI_Write_Buf(WR_TX_PLOAD,tx_buf,TX_PLOAD_WIDTH);       // write playload to TX_FIFO
    }
    if(status&MAX_RT)                                       // if receive data ready (MAX_RT) interrupt, this is
retransmit than  SETUP_RETR
    {
      SPI_RW_Reg(FLUSH_TX,0);
      SPI_Write_Buf(WR_TX_PLOAD,tx_buf,TX_PLOAD_WIDTH);      // disable standy-mode
    }
    SPI_RW_Reg(WRITE_REG+STATUS,status);                    // clear RX_DR or TX_DS or MAX_RT interrupt
flag
    delay(1000);
  }
}

//****************************************************
// Function: init_io();
// Description:
// flash led one time,chip enable(ready to TX or RX Mode),
// Spi disable,Spi clock line init high
//****************************************************
void init_io(void)
{
  SPI_PORT&=~CE;                    // chip enable
  SPI_PORT|=CSN;                    // Spi disable
  SPI_PORT&=~SCK;                   // Spi clock line init high
}

/****************************************************
 * Function: SPI_RW();
 *
 * Description:
 * Writes one unsigned char to nRF24L01, and return the unsigned char read
```

```
 * from nRF24L01 during write, according to SPI protocol
 ****************************************************/
unsigned char SPI_RW(unsigned char Byte)
{
 unsigned char i;
 for(i=0;i<8;i++)                  // output 8-bit
 {
  if(Byte&0x80)
  {
   SPI_PORT |=MOSI;    // output 'unsigned char', MSB to MOSI
  }
  else
  {
   SPI_PORT &=~MOSI;
  }
  SPI_PORT|=SCK;                    // Set SCK high..
  Byte <<= 1;                      // shift next bit into MSB..
  if(SPI_IN & MISO)
  {
   Byte |= 1;                  // capture current MISO bit
  }
  SPI_PORT&=~SCK;                   // ..then set SCK low again
 }
 return(Byte);                  // return read unsigned char
}
/****************************************************/

/****************************************************
 * Function: SPI_RW_Reg();
 *
 * Description:
 * Writes value 'value' to register 'reg'
/****************************************************/
unsigned char SPI_RW_Reg(unsigned char reg, unsigned char value)
{
 unsigned char status;

 SPI_PORT&=~CSN;                // CSN low, init SPI transaction
 status = SPI_RW(reg);           // select register
 SPI_RW(value);                // ..and write value to it..
 SPI_PORT|=CSN;                  // CSN high again

 return(status);                // return nRF24L01 status unsigned char
}
/****************************************************/

/****************************************************
 * Function: SPI_Read();
 *
 * Description:
 * Read one unsigned char from nRF24L01 register, 'reg'
/****************************************************/
unsigned char SPI_Read(unsigned char reg)
{
```

```
 unsigned char reg_val;

 SPI_PORT&=~CSN;              // CSN low, initialize SPI communication...
 SPI_RW(reg);                // Select register to read from..
 reg_val = SPI_RW(0);         // ..then read register value
 SPI_PORT|=CSN;               // CSN high, terminate SPI communication

 return(reg_val);             // return register value
}
/**************************************************/

/**************************************************
 * Function: SPI_Read_Buf();
 *
 * Description:
 * Reads 'unsigned chars' #of unsigned chars from register 'reg'
 * Typically used to read RX payload, Rx/Tx address
/**************************************************/
unsigned char SPI_Read_Buf(unsigned char reg, unsigned char *pBuf, unsigned char bytes)
{
 unsigned char status,i;

 SPI_PORT&=~CSN;                // Set CSN low, init SPI tranaction
 status = SPI_RW(reg);          // Select register to write to and read status unsigned char

 for(i=0;i<bytes;i++)
 {
  pBuf[i] = SPI_RW(0);    // Perform SPI_RW to read unsigned char from nRF24L01
 }

 SPI_PORT|=CSN;                 // Set CSN high again

 return(status);                // return nRF24L01 status unsigned char
}
/**************************************************/

/**************************************************
 * Function: SPI_Write_Buf();
 *
 * Description:
 * Writes contents of buffer '*pBuf' to nRF24L01
 * Typically used to write TX payload, Rx/Tx address
/**************************************************/
unsigned char SPI_Write_Buf(unsigned char reg, unsigned char *pBuf, unsigned char bytes)
{
 unsigned char status,i;

 SPI_PORT&=~CSN;                // Set CSN low, init SPI tranaction
 status = SPI_RW(reg);          // Select register to write to and read status unsigned char
 for(i=0;i<bytes; i++)          // then write all unsigned char in buffer(*pBuf)
 {
  SPI_RW(*pBuf++);
 }
 SPI_PORT|=CSN;                 // Set CSN high again
```

```
  return(status);               // return nRF24L01 status unsigned char
}
/****************************************************/

/****************************************************
 * Function: TX_Mode();
 *
 * Description:
 * This function initializes one nRF24L01 device to
 * TX mode, set TX address, set RX address for auto.ack,
 * fill TX payload, select RF channel, datarate & TX pwr.
 * PWR_UP is set, CRC(2 unsigned chars) is enabled, & PRIM:TX.
 *
 * ToDo: One high pulse(>10us) on CE will now send this
 * packet and expext an acknowledgment from the RX device.
 ****************************************************/
void TX_Mode(void)
{
  SPI_PORT&=~CE;

  SPI_Write_Buf(WRITE_REG + TX_ADDR, TX_ADDRESS, TX_ADR_WIDTH);    // Writes TX_Address to
nRF24L01
  SPI_Write_Buf(WRITE_REG + RX_ADDR_P0, TX_ADDRESS, TX_ADR_WIDTH); // RX_Addr0 same as
TX_Adr for Auto.Ack

  SPI_RW_Reg(WRITE_REG + EN_AA, 0x01);      // Enable Auto.Ack:Pipe0
  SPI_RW_Reg(WRITE_REG + EN_RXADDR, 0x01);  // Enable Pipe0
  SPI_RW_Reg(WRITE_REG + SETUP_RETR, 0x1a); // 500us + 86us, 10 retrans...
  SPI_RW_Reg(WRITE_REG + RF_CH, 40);        // Select RF channel 40
  SPI_RW_Reg(WRITE_REG + RF_SETUP, 0x07);   // TX_PWR:0dBm, Datarate:2Mbps, LNA:HCURR
  SPI_RW_Reg(WRITE_REG + CONFIG, 0x0e);     // Set PWR_UP bit, enable CRC(2 unsigned chars) &
Prim:TX. MAX_RT & TX_DS enabled..
  SPI_Write_Buf(WR_TX_PLOAD,tx_buf,TX_PLOAD_WIDTH);

  SPI_PORT|=CE;
}

// Code End
```

Download the code below into the RX Arduino® (receive). This code will drive the nFR24L01 module to receive the data that transmit form the TX module and print it to the serial port.

```
// Code Begin
#include "NRF24L01.h"

//****************************************************
#define TX_ADR_WIDTH    5   // 5 unsigned chars TX(RX) address width
#define TX_PLOAD_WIDTH  32  // 32 unsigned chars TX payload

unsigned char TX_ADDRESS[TX_ADR_WIDTH]  =
{
  0x34,0x43,0x10,0x10,0x01
}; // Define a static TX address
```

```
unsigned char rx_buf[TX_PLOAD_WIDTH];
unsigned char tx_buf[TX_PLOAD_WIDTH];
//*************************************************
void setup()
{
 SPI_DIR = ( CE + SCK + CSN + MOSI);
 SPI_DIR &=~ ( IRQ + MISO);
 //  attachInterrupt(1, _ISR, LOW); // interrupt enable
 Serial.begin(9600);
 init_io();                    // Initialize IO port
 unsigned char status=SPI_Read(STATUS);
 Serial.print("status = ");
 Serial.println(status,HEX);      // There is read the mode's status register, the default value should be 'E'
 Serial.println("******************RX_Mode start*******************************R");
 RX_Mode();                    // set RX mode
}
void loop()
{
 for(;;)
 {
   unsigned char status = SPI_Read(STATUS);                    // read register STATUS's value
   if(status&RX_DR)                                          // if receive data ready (TX_DS) interrupt
   {
     SPI_Read_Buf(RD_RX_PLOAD, rx_buf, TX_PLOAD_WIDTH);           // read playload to rx_buf
     SPI_RW_Reg(FLUSH_RX,0);                                   // clear RX_FIFO
     for(int i=0; i<32; i++)
     {
        Serial.print(" ");
        Serial.print(rx_buf[i],HEX);                          // print rx_buf
     }
     Serial.println(" ");
   }
   SPI_RW_Reg(WRITE_REG+STATUS,status);                        // clear RX_DR or TX_DS or MAX_RT
interrupt flag
   delay(1000);
 }
}

//*************************************************
// Function: init_io();
// Description:
// flash led one time,chip enable(ready to TX or RX Mode),
// Spi disable,Spi clock line init high
//*************************************************
void init_io(void)
{
 SPI_PORT&=~CE;                    // chip enable
 SPI_PORT|=CSN;                    // Spi disable
 SPI_PORT&=~SCK;                   // Spi clock line init high
}

/*************************************************
 * Function: SPI_RW();
```

```
 *
 * Description:
 * Writes one unsigned char to nRF24L01, and return the unsigned char read
 * from nRF24L01 during write, according to SPI protocol
 ****************************************************/
unsigned char SPI_RW(unsigned char Byte)
{
 unsigned char i;
 for(i=0;i<8;i++)                  // output 8-bit
 {
  if(Byte&0x80)
  {
   SPI_PORT |=MOSI;    // output 'unsigned char', MSB to MOSI
  }
  else
  {
   SPI_PORT &=~MOSI;
  }
  SPI_PORT|=SCK;                    // Set SCK high..
  Byte <<= 1;                      // shift next bit into MSB..
  if(SPI_IN & MISO)
  {
   Byte |= 1;                  // capture current MISO bit
  }
  SPI_PORT&=~SCK;                   // ..then set SCK low again
 }
 return(Byte);                  // return read unsigned char
}
/****************************************************/

/*****************************************************
 * Function: SPI_RW_Reg();
 *
 * Description:
 * Writes value 'value' to register 'reg'
/*****************************************************/
unsigned char SPI_RW_Reg(unsigned char reg, unsigned char value)
{
 unsigned char status;

 SPI_PORT&=~CSN;                 // CSN low, init SPI transaction
 status = SPI_RW(reg);           // select register
 SPI_RW(value);                 // ..and write value to it..
 SPI_PORT|=CSN;                  // CSN high again

 return(status);                // return nRF24L01 status unsigned char
}
/****************************************************/

/*****************************************************
 * Function: SPI_Read();
 *
 * Description:
 * Read one unsigned char from nRF24L01 register, 'reg'
```

```
/**************************************************/
unsigned char SPI_Read(unsigned char reg)
{
 unsigned char reg_val;

 SPI_PORT&=~CSN;              // CSN low, initialize SPI communication...
 SPI_RW(reg);               // Select register to read from..
 reg_val = SPI_RW(0);         // ..then read register value
 SPI_PORT|=CSN;              // CSN high, terminate SPI communication

 return(reg_val);            // return register value
}
/**************************************************/

/**************************************************
 * Function: SPI_Read_Buf();
 *
 * Description:
 * Reads 'unsigned chars' #of unsigned chars from register 'reg'
 * Typically used to read RX payload, Rx/Tx address
/**************************************************/
unsigned char SPI_Read_Buf(unsigned char reg, unsigned char *pBuf, unsigned char bytes)
{
 unsigned char status,i;

 SPI_PORT&=~CSN;               // Set CSN low, init SPI tranaction
 status = SPI_RW(reg);            // Select register to write to and read status unsigned char

 for(i=0;i<bytes;i++)
 {
  pBuf[i] = SPI_RW(0);    // Perform SPI_RW to read unsigned char from nRF24L01
 }

 SPI_PORT|=CSN;               // Set CSN high again

 return(status);              // return nRF24L01 status unsigned char
}
/**************************************************/

/**************************************************
 * Function: SPI_Write_Buf();
 *
 * Description:
 * Writes contents of buffer '*pBuf' to nRF24L01
 * Typically used to write TX payload, Rx/Tx address
/**************************************************/
unsigned char SPI_Write_Buf(unsigned char reg, unsigned char *pBuf, unsigned char bytes)
{
 unsigned char status,i;

 SPI_PORT&=~CSN;               // Set CSN low, init SPI tranaction
 status = SPI_RW(reg);           // Select register to write to and read status unsigned char
 for(i=0;i<bytes; i++)          // then write all unsigned char in buffer(*pBuf)
 {
```

```
    SPI_RW(*pBuf++);
  }
  SPI_PORT|=CSN;                  // Set CSN high again
  return(status);                 // return nRF24L01 status unsigned char
}
/****************************************************/

/****************************************************
 * Function: RX_Mode();
 *
 * Description:
 * This function initializes one nRF24L01 device to
 * RX Mode, set RX address, writes RX payload width,
 * select RF channel, datarate & LNA HCURR.
 * After init, CE is toggled high, which means that
 * this device is now ready to receive a datapacket.
/****************************************************/
void RX_Mode(void)
{
  SPI_PORT&=~CE;
  SPI_Write_Buf(WRITE_REG + RX_ADDR_P0, TX_ADDRESS, TX_ADR_WIDTH); // Use the same address on
the RX device as the TX device
  SPI_RW_Reg(WRITE_REG + EN_AA, 0x01);      // Enable Auto.Ack:Pipe0
  SPI_RW_Reg(WRITE_REG + EN_RXADDR, 0x01);  // Enable Pipe0
  SPI_RW_Reg(WRITE_REG + RF_CH, 40);        // Select RF channel 40
  SPI_RW_Reg(WRITE_REG + RX_PW_P0, TX_PLOAD_WIDTH); // Select same RX payload width as TX
Payload width
  SPI_RW_Reg(WRITE_REG + RF_SETUP, 0x07);   // TX_PWR:0dBm, Datarate:2Mbps, LNA:HCURR
  SPI_RW_Reg(WRITE_REG + CONFIG, 0x0f);     // Set PWR_UP bit, enable CRC(2 unsigned chars) &
Prim:RX. RX_DR enabled..
  SPI_PORT|=CE;                           // Set CE pin high to enable RX device
  //  This device is now ready to receive one packet of 16 unsigned chars payload from a TX device sending
to address
  //  '3443101001', with auto acknowledgment, retransmit count of 10, RF channel 40 and datarate = 2Mbps.
}
/****************************************************/
// Code End
```

Now, power on both Arduino® and connect the RX to the PC via USB. Open the IDE serial port monitor, change the baud rate to 9600 bps, and you can see the received data.

If you want to change the Arduino® pin connection to the module, just modify the definition on the nFR24L01.

7. More Information

Please refer to the VMA322 product page on www.velleman.eu for more information.

Also, consult http://playground.arduino.cc/InterfacingWithHardware/Nrf24L01.

RED Declaration of Conformity

Hereby, Velleman NV declares that the radio equipment type VMA322 is in compliance with Directive 2014/53/EU.

The full text of the EU declaration of conformity is available at the following internet address: www.velleman.eu.

Use this device with original accessories only. Velleman nv cannot be held responsible in the event of damage or injury resulting from (incorrect) use of this device. For more info concerning this product and the latest version of this manual, please visit our website www.velleman.eu. The information in this manual is subject to change without prior notice.

Velleman® Service and Quality Warranty

Since its foundation in 1972, Velleman® acquired extensive experience in the electronics world and currently distributes its products in over 85 countries.

All our products fulfil strict quality requirements and legal stipulations in the EU. In order to ensure the quality, our products regularly go through an extra quality check, both by an internal quality department and by specialized external organisations. If, all precautionary measures notwithstanding, problems should occur, please make appeal to our warranty (see guarantee conditions).

General Warranty Conditions Concerning Consumer Products (for EU):

• All consumer products are subject to a 24-month warranty on production flaws and defective material as from the original date of purchase.

• Velleman® can decide to replace an article with an equivalent article, or to refund the retail value totally or partially when the complaint is valid and a free repair or replacement of the article is impossible, or if the expenses are out of proportion.

You will be delivered a replacing article or a refund at the value of 100% of the purchase price in case of a flaw occurred in the first year after the date of purchase and delivery, or a replacing article at 50% of the purchase price or a refund at the value of 50% of the retail value in case of a flaw occurred in the second year after the date of purchase and delivery.

• Not covered by warranty:

- all direct or indirect damage caused after delivery to the article (e.g. by oxidation, shocks, falls, dust, dirt, humidity...), and by the article, as well as its contents (e.g. data loss), compensation for loss of profits;

- consumable goods, parts or accessories that are subject to an aging process during normal use, such as batteries (rechargeable, non-rechargeable, built-in or replaceable), lamps, rubber parts, drive belts... (unlimited list);

- flaws resulting from fire, water damage, lightning, accident, natural disaster, etc.…;

- flaws caused deliberately, negligently or resulting from improper handling, negligent maintenance, abusive use or use contrary to the manufacturer's instructions;

- damage caused by a commercial, professional or collective use of the article (the warranty validity will be reduced to six (6) months when the article is used professionally);

- damage resulting from an inappropriate packing and shipping of the article;

- all damage caused by modification, repair or alteration performed by a third party without written permission by Velleman®.

• Articles to be repaired must be delivered to your Velleman® dealer, solidly packed (preferably in the original packaging), and be completed with the original receipt of purchase and a clear flaw description.

• Hint: In order to save on cost and time, please reread the manual and check if the flaw is caused by obvious causes prior to presenting the article for repair. Note that returning a non-defective article can also involve handling costs.

• Repairs occurring after warranty expiration are subject to shipping costs.

• The above conditions are without prejudice to all commercial warranties.

The above enumeration is subject to modification according to the article (see article's manual).

Made in PRC
Imported by Velleman nv
Legen Heirweg 33, 9890 Gavere, Belgium
www.velleman.eu