15-5-2019

Home domotica system

DIY Project

Ian Markies

Index

[Preface 2](#_Toc8821642)

[1. Research 3](#_Toc8821643)

[1.1. Antenna 3](#_Toc8821644)

[1.2. Communication protocol 3](#_Toc8821645)

[1.2.1. Length 3](#_Toc8821646)

[1.2.2. ID-Modules 3](#_Toc8821647)

[1.2.3. Function codes 4](#_Toc8821648)

[1.2.4. Data 4](#_Toc8821649)

[1.2.5. LRC 4](#_Toc8821650)

[1.3. Battery check 4](#_Toc8821651)

[Appendix 5](#_Toc8821652)

[Appendix 1: Connection schematic 5](#_Toc8821653)

# Preface

This project is for a DIY home domotica system for controlling your light, doors, window shutters and many more. This project will have one master and the ability to communicate with up to nine slaves. Later this project will include a Arduino with a WIFI module and an app to control the system.

It is not for meant for large scale use in buildings or for use in products. It is meant for the hobbyists to explore wireless communication or improve this project.

# Research

## Antenna

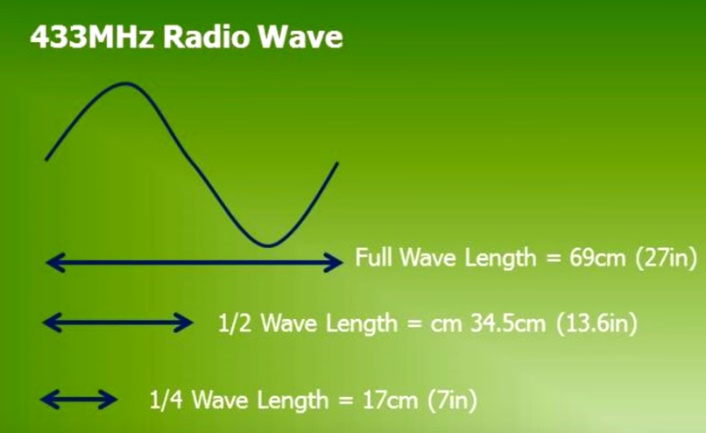
 The transmitters and receivers that are used don't have an antenna this means that the transmitter and receiver can't be more than 5 meters apart from each other. To solve this problem an antenna with the correct length must be added to the transmitters and receiver. The formula that is used is: (300/f[MHz])/2 -> (300/433)/2 = 34.6cm this is the total length of the antenna. The length of the antenna is divided in two sperate antenna's, one signal 17.3cm and one ground 17.3cm. The transmitters and receivers can have the signal and the ground antenna but for practical use only the signal antenna will be mounted to the transmitters and receivers. To not mount the ground antenna will add noise in the signal but this will be terminated by the use of a LRC (Longitudinal Redundancy Check).

Figure 1: antenna length

## Communication protocol

For this project we need to make sure that the different modules only decode the message that is meant for that module.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Start | Length | ID | ID receiver | Function | Stop | LRC |
| 1 Byte | 1 Byte | 1 Byte | 1 Byte | 1 Byte | 1 Byte | 1 Byte |
| 0x01 | LENGTH | ID | DEST | FUNCTION | 0x04 | LRC\_RESULT |

Table 1: message structure without data

Some modules do not only send an acknowledge but also can send data that the control module can receive and process. This data could be the battery level, the light intensity from the sun or if a sensor is activated or not. For these kind of messages is the following structure.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Start | Length | ID | ID receiver | Function | Data | Stop | LRC |
| 1 Byte | 1 Byte | 1 Byte | 1 Byte | 1 Byte | X Byte | 1 Byte | 1 Byte |
| 0x01 | LENGTH | ID | DEST | FUNCTION | DATA | 0x04 | LRC\_RESULT |

Table 2: message structure with data

### Length

The length of a message is the numbers of bytes from start to stop. Only the LRC is not included in the length. By including the length of the message the receiver can easily check if the message came through in its full length.

### ID-Modules

To make sure that each module won’t decode a code that is not for that module, each module gets its own ID. This system will be able to communicate with ten modules with each it’s own ID, in the table beneath the ten addresses will be displayed.

|  |  |  |  |
| --- | --- | --- | --- |
| Module | HEX | DEC | BIN |
| Control | 20 | 32 | 0010 0000 |
| #1 | 21 | 33 | 0010 0001 |
| #2 | 22 | 34 | 0010 0010 |
| #3 | 23 | 35 | 0010 0011 |
| #4 | 24 | 36 | 0010 0100 |
| #5 | 25 | 37 | 0010 0101 |
| #6 | 26 | 38 | 0010 0110 |
| #7 | 27 | 39 | 0010 0111 |
| #8 | 28 | 40 | 0010 1000 |
| #9 | 29 | 41 | 0010 1001 |

Table : ID codes

### Function codes

Each module has its own functions and actions, because the slave modules won’t do anything without the needed message from the control module. But for each action and/or communication message the receiver needs to know what the function of the message is. For this reason each message needs a function code, which are displayed in the table 4.

|  |  |
| --- | --- |
| Code | Description |
| 0x40 | ACK (acknowledgement) |
| 0x41 | Message received wrong, resend last message |
| 0x42 | Battery status |
| 0x43 |  |
|  |  |
|  |  |
|  |  |

Table : Function codes

### Data

Some modules are able to send

One byte can hold a value of 255 and the unprocessed data value could have a maximal value of 1023 (10 bits). When this data is processed the value of the data could have decimal numbers, this could be fixed by round the numbers or by using one byte for the two decimal places numbers.

### LRC

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Data pack 1 | 1 | 0 | 1 | 0 | 1 | 1 | 1 | 1 |
| Data pack 2 | 1 | 0 | 0 | 0 | 1 | 0 | 0 | 1 |
| Data pack 3 | 0 | 0 | 1 | 1 | 0 | 0 | 0 | 1 |
| Data pack 4 | 1 | 0 | 1 | 1 | 0 | 0 | 0 | 1 |
| LRC | 1 | 0 | 1 | 0 | 0 | 1 | 1 | 0 |

When a message is received we need to make sure that the message is received correctly. By sending a LRC with the message we can check is the message is received correctly and that there was no change in the message by any disturbance and/or noise.

With the chosen LRC it is not possible to check which part of the message is wrong, for that is a double LRC needed that also checks each byte.

Table : Example LRC

## Battery check

# Appendix

## Appendix 1: Connection schematic

