**Teknisk Systembeskrivning**

Projektrapport Inbyggda System Adam Pettersson

**Introduction**

The function of this project is to allow the users to communicate via knocking on a door. It consists of two parts. One part is a module that is to be put on a door. It then senses vibrations and is able to detect when someone is knocking on the door. When somebody knocks on the door, or opens it, the device transmits a message via the Lora network.

The other part of the project is a module that recieves these messages and lets the user know when someone has knocked on the door where the first module is placed.

One application can be for a teacher to carry the receiving module with him whenever he leaves his office. If someone then knocks on the door to his office he will be notified.

**Material**

**Software:**

* The Things Network-LoRa
* MQTT Explorer
* Arduion IDE
* Segger Embedded Studio
* StmCubeMX

**Hardware:**

* Stm32
* ESP32
* LoRa-E5 mini
* ADXL335 accelerometer
* Lipo Battery
* Battery pack with six AA rechargeable batteries
* LM358 OP-amplifier

**Functionality:**

A sensor module is attached to a door or another surface. When someone knocks on the surface, a message is sent via LoRa and MQTT to a receiver module, which notifies that someone is knocking on the door where the sensor is attached. A yellow LED that lights up and a sound signal that beeps for a second signal to the user that someone is knocking on the door. If someone has opened the door, a permanent warning signal is activated, along with a red LED. The receiver must then confirm that it has noted that someone has knocked to stop the sound signal by pressing a button.

**System overlook**

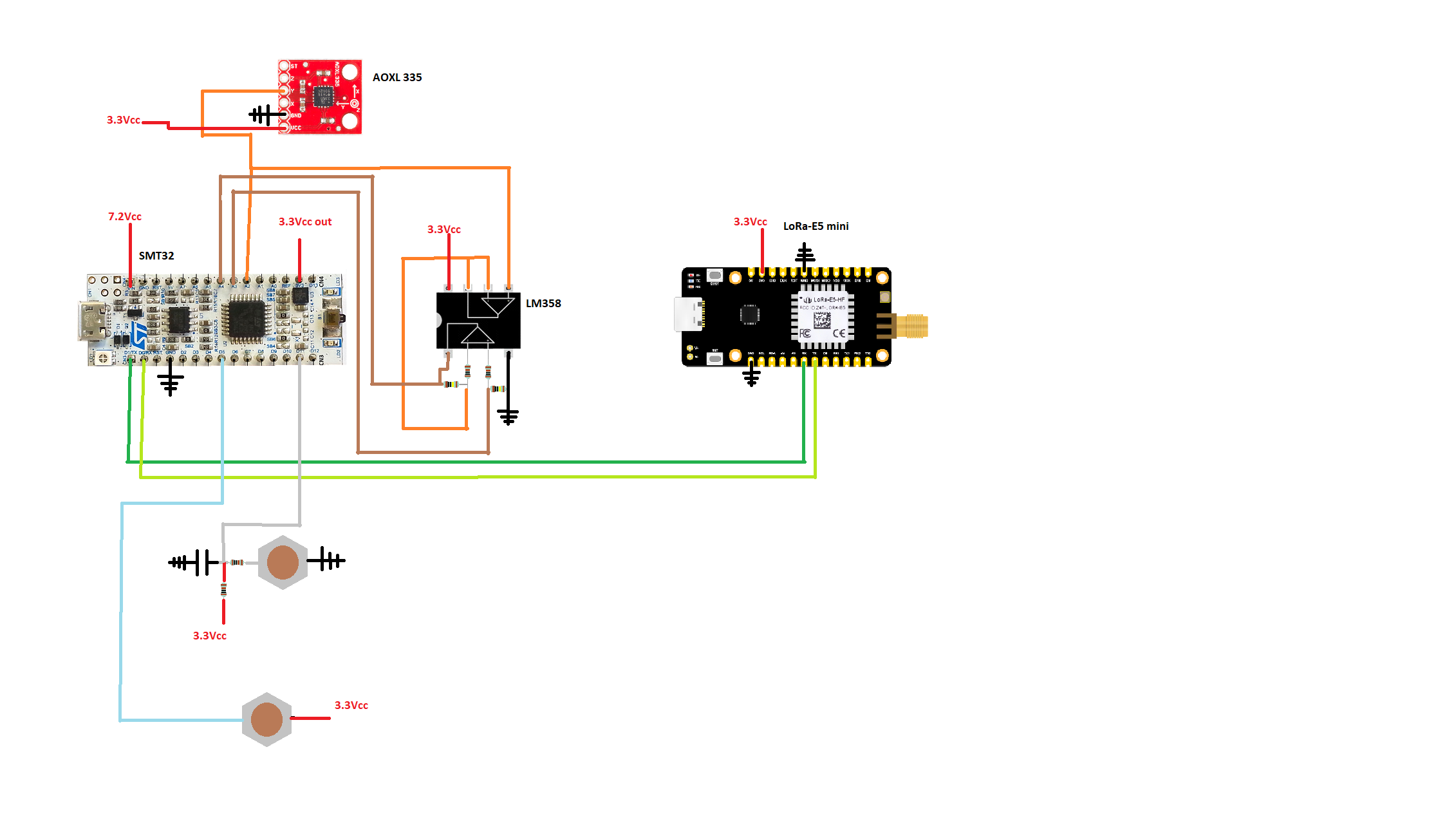
En bild som visar text, skärmbild, Teckensnitt, diagram

Automatiskt genererad beskrivning

**Technical description**

**Door module**

**Wiring diagram:**



The door module consists of an esp32 on a L432KC-board, a LM358 OP-amp, one AOXL 335 accelerometer, two buttons and some resistors and capacitors. The button in the bottom of the image is simply placed between the door and the wall, to check if the door is open. The accelerometer is used for sensing vibrations. When the system starts up the stm32 sends uart messages to the Lora E5 mini to make it connect to lora. When connection is established the uart messages from the stm32 gets sent out via LoRa through the Lora E5. Two different payloads can be sent to the receiving module. “open” if the door has been opened and “knack” if the system detects a knock on the door. In order to communicate with the LoRa E5 via uart, every message must begin with “AT+”. For the initiation we send the following messages “AT+DR=EU868”, “AT+CH=NUM,0-2", “AT+MODE=LWOTAA”,   
“AT+JOIN”. And for sending a message via Lora, simply “AT+<message>”.

Two ADC-inputs are used on the STM32. One recieves the values directly from the accelerometer. The other receives the output from the differential amplifier(LM358). When the upper button is pressed the ADC-value from the accelerometer is saved in a variable and send out though the DAC to the differential amplifier.

Using this method we can compare this set value with input from the accelerometer in the differential amplifier. This allows for greater amplification (1000 times amplification)so we get a better sensitivity.

As seen in the while loop below we first calculate a mean of the ADC-inputs from the accelerometer. Then we must press the button and send this value out on the DAC before the rest of the program starts. Then we continually check if the door is open and if a knock has been detected. If the door is open we no longer check for knocks.

A screenshot of a computer code

Description automatically generated

**Receiver module**

**Wiring diagram**

**En bild som visar text, krets, Elektrisk ingenjörskonst, diagram

Automatiskt genererad beskrivning**

**The code**

**En bild som visar text, skärmbild, Teckensnitt

Automatiskt genererad beskrivning**

The libraries we use and their functions are as follows:

**WiFi.h:** This library is employed to manage the WiFi functionalities on the ESP32. Using it, one can connect to a WiFi network, get and set IP addresses, start a server, and more. In our case, it's used to connect the device to a WiFi network.

**PubSubClient.h**: PubSubClient is an MQTT client library for Arduino. With this library, a device can publish and subscribe to messages from an MQTT broker. In our setup, it helps with connecting to an MQTT broker and listening to messages from a specific topic.

**ArduinoJson.h**: ArduinoJson is a library for handling JSON in Arduino. We use it to interpret the received JSON messages from our TTN MQTT broker and then extract payload information from these messages.

**base64.hpp:** This library assists with Base64 encoding and decoding. Base64 is commonly used to represent binary data as text strings. In our scenario, it's employed to decode payload data received in Base64 format from MQTT messages.

En bild som visar text, skärmbild

Automatiskt genererad beskrivning

The code for the ESP32 begins by initiating a connection to WiFi. Connection details, such as SSID and password, as well as MQTT authentication details, are retrieved from TTN. Once successfully connected, a session to our MQTT Explorer and the TTN MQTT broker is established using the PubSubClient library. This device then subscribes to a specific topic where data from the E5 mini is published.

En bild som visar text, skärmbild

Automatiskt genererad beskrivning

Upon receiving a message, the callback function is activated. This function processes the message, converting the incoming JSON message into a string, and then decodes its Base64 format into a decimal format. If the message content corresponds to "knock", the yellow LED is activated along with a short sound alert. If the message matches "open", the red LED gets activated, and a continuous sound alert is played.

En bild som visar text, Teckensnitt, linje, skärmbild

Automatiskt genererad beskrivning

The image above shows part of how the data looks when it's packaged in JSON format from TTN to our MQTT client. Under the "uplink message" category, we find the "frm\_payload", which contains data from the sensor module of interest, albeit in Base64 format.

En bild som visar text, skärmbild, Teckensnitt, nummer

Automatiskt genererad beskrivning

The above displays the configuration details for our device in TTN. We utilized a frequency band of 863-870 MHz for data transmission; this frequency is standard in Europe for LoRa.

We chose to work with LoRaWAN because it is designed for wireless battery-powered devices and is perfect for transmitting small amounts of data over long distances. Although LoRaWAN enables long-distance communication, it is very energy-efficient and user-friendly by adjusting the data rate depending on the device's distance to the nearest gateway.

En bild som visar text, skärmbild, linje, Teckensnitt

Automatiskt genererad beskrivning

To access the MQTT broker hosted by TTN, we use an API Key. All login details were generated by TTN. We opted for Port 1883 since we didn't need to send encrypted information.

**Power Calculations**

We assume that the sensor module sends data approximately every 10 seconds.

**Battery Life for the Receiver:**

Average Current = 0.90×40mA + 0.10×105mA

= 36mA + 10.5mA

= 46.5 mA

Battery Lifespan = Battery Capacity / Average Current

= 1700 mAh / 46.5 mA = 36.5 hours

**Battery Life for the Transmitter Module:**

Average Current = 13.5 mA + 35.2 mA

= 48.7 mA

Battery Lifespan = Battery Capacity / Average Current

= 2000 mAh / 48.7 mA = 41.07 hours

**Discussion**

This project did work pretty much as expected. However there is room for improvement. For example two way communication via LoRa (Downlink) would be a usefull addition. With that function the person on the receiving end would be able to answer to confirm they have noticed the knock and have a message sent back to the door module. A more solid casing would be usefull as well. So far we have used duct tape to stick the door module to the door.

Also if this product was to be used over a longer period a time we would have to do something about the power consumption. That could probably be solved by using a watch dog timer and have the unit only go active when messages were sent and received.