1-Click Connect Report

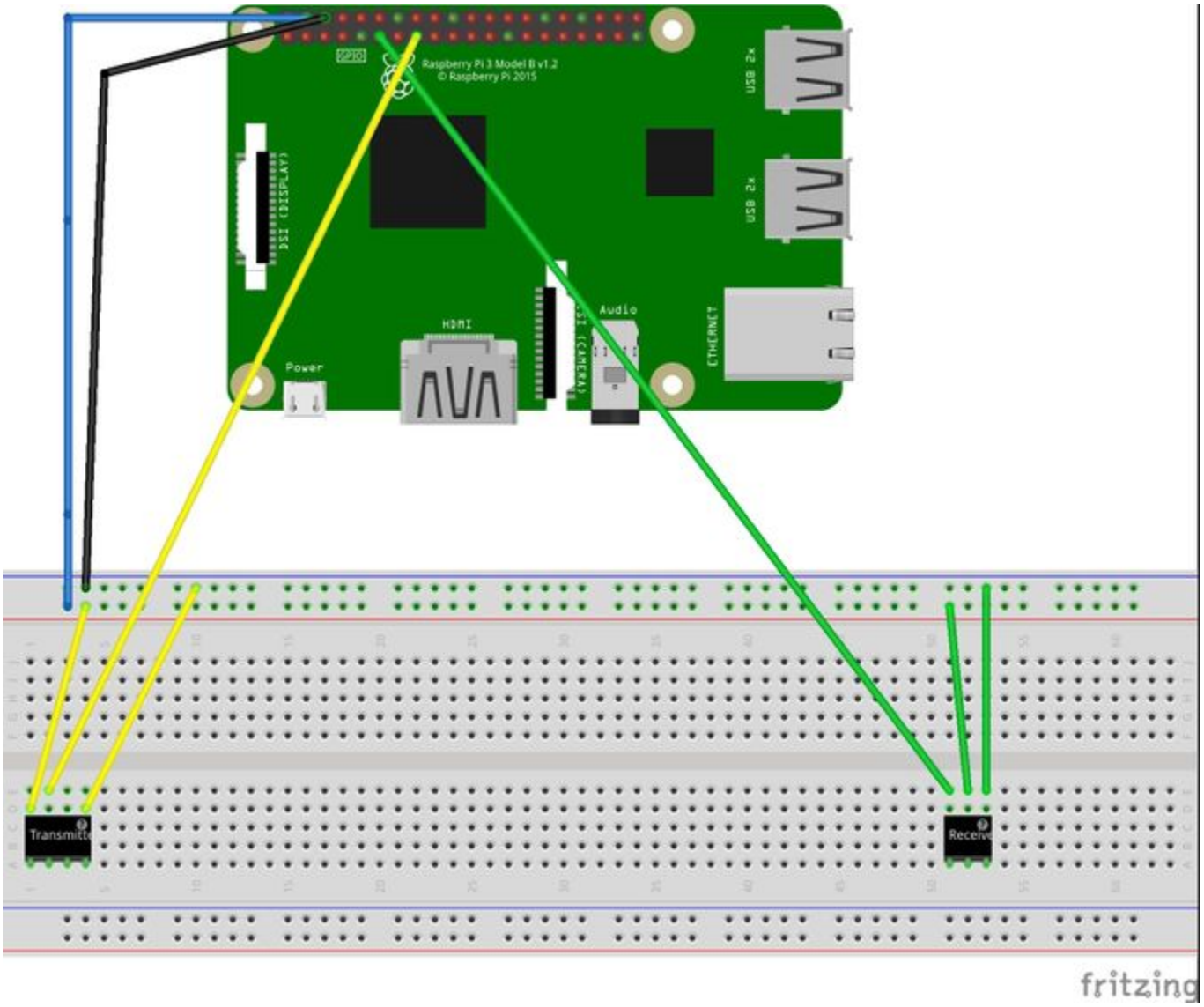
Introduction

Having in mind ease of use in the LazzyRoom project, a method to easily connect custom devices to our hidden network had to be created. The idea formulated was that of a one-click connect. The user would click a button connected to the custom device and this would initialize Radio Frequency (RF) communication between it and the Raspberry Pi master device. The hidden network's SSID (name) and Password would then be sent to the custom device and it would connect to the network. This communication would later be used to also send data that would establish a secure connection between the devices, but that is outside of the scope of this report and will be implemented in the future. In this document I will go over how the communication between the two devices was established through RF and how one can use this protocol to remotely connect the custom device to a network of their choosing.

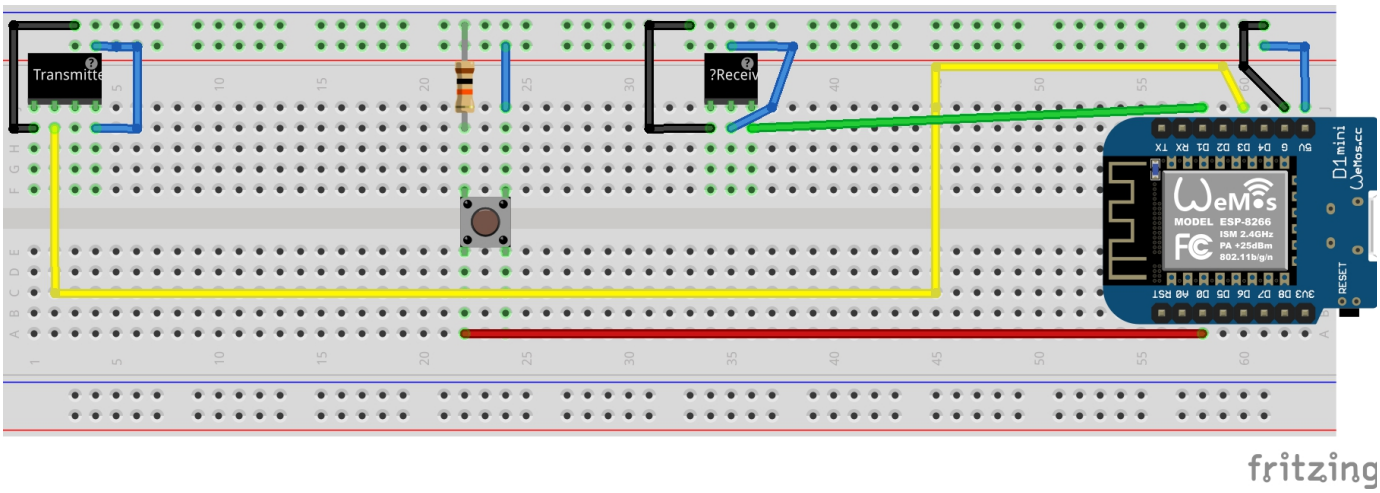
Process

Physical

The physical connection of the devices is rather simple. On one side you have the raspberry Pi, which is connected to a 433Mhz RF transmitter and a receiver of the same bandwidth.



On the other side you have a Wemos D1 Mini with a ESP8266 microchip with WiFi capabilities. It would also be connected to the same receiver and transmitter pair, but with the addition of a button as well.



Code

Initial research

The process of creating code so that both devices can communicate with each other was rather arduous. Due to the wireless nature of RF one has to create their own mini protocol, so that the receiver may know that the messages are for him. At first this was not known to the author and different libraries (RadioHead for the D1 mini and this open-source repository for the Raspberry Pi) were used for both devices. This meant that no information could be shared as these libraries had their own protocols. Upon further investigation the different pieces of such a protocol were established:

- A signature byte array of various size

This would be used as a message identifier. The receiver knows what signature to expect and it parses the message only if it has this signature. Each byte is a number in hexadecimal.

- Message body

Each library may use different methods to describe a message. For example the library that was used most for inspiration would send the signature, then the message length, the message itself and finally a checksum that would identify whether the message was sent correctly.

- Timing

Each library sends the bits in a different way and in different periods

Due to the above-mentioned points it was impossible to make use of two different libraries to communicate with each other. This was a big problem, as there was not one library that works for both a Raspberry Pi and an Arduino-like device like the D1 mini. Furthermore, most examples for the Raspberry Pi would be written in Python, which introduced another problem as the D1 mini used Arduino C++. This issue will be described further, later in this report.

Implementation

The conclusion from the above research was that to reach the desired functionality a barebones program with no RF library had to be created from scratch. Thankfully, the code found in the linked repository above, or specifically in **Pi433MHzRx** and **Pi433MHzTx** appeared simple enough to be reused in this specific case. Although this assumption was true in the end, the process of reusing the code had to go through 4 phases before it was in a fully finished state.

1. Translating the transmitter code

Reasoning

As the logic behind the code was rather convoluted, the author decided first to try and translate the Python code for the transmitter into C++ and put it into the D1 Mini. Theoretically, this would mean that both codebases would use the same protocol and thusly be able to communicate with each other. The receive code would stay in python and be used as is on the Raspberry Pi.

Conclusion

Sadly, after implementing this and testing, the communication still did not work. The author noticed that the amount of packets received on the receiver were noticeably different when the transmitter is transmitting, but the data was nonsensical.

2. Translating the receiver code

Reasoning

After researching further a theory behind the problem in phase 1 was formed - C++ was much faster than Python. A simple proof can be found here. Having that in mind, it was possible that the difference in speed was the issue. Looking deeper into the code proved this as it turned out that the logic behind the programs was time-based. This meant that the 0s and 1s sent through RF are not actual data, but the periods in between them defined what type of bit is coming in. For this reason, speed was of the essence and the receiver code was translated to C++. One issue faced was that non-arduino C++ as is the one on the Raspberry Pi did not have methods to write to the general-purpose input/output (**GPIO**) pins. This was done by using digitalWrite on the D1 mini, where one could specify whether a 0 or a 1 should be sent through RF. For this purpose the library WiringPi was used on the Pi, which closely mimics Arduino C++, having exactly the same methods, together with Timing functions like delays and milli/microsecond getters which were also needed in the receiver code.

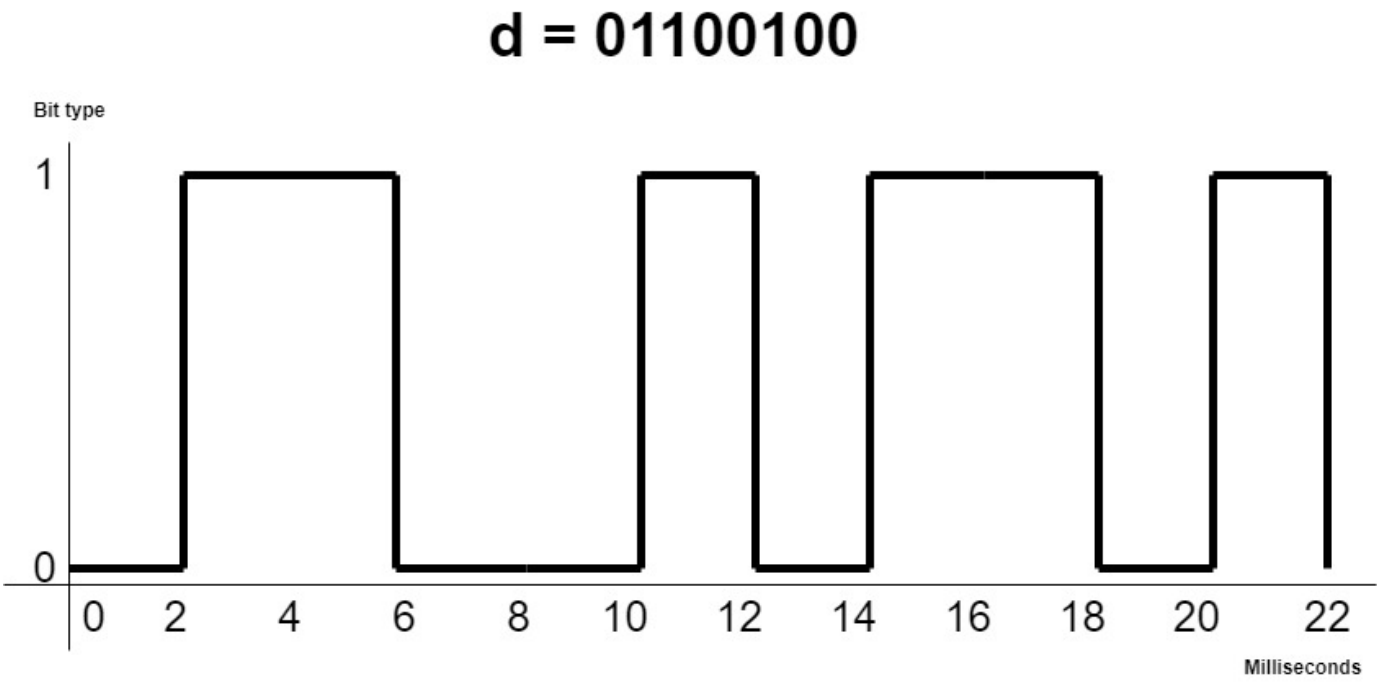
Conclusion

The results of this phase were quite promising, as data was discernable, although still not fully correct. The packet signature would sometimes be correctly received, but the data following it would rarely be right. This lead to the conclusion that there is a mistiming in the programs still. Although this phase was definitely providing better outcomes, a reliable communication was still needed, so more augmentations of the code had to be done.

3. Manipulating the timing variables

Reasoning

At this point it was futile to continue without understanding the code better. As said earlier, time was used instead of actual data to send out messages. In its original state, the transmitter would send out a start bit 1 then wait for 2 milliseconds to pass, after which it will start toggling a bit and sending it out for either 2 milliseconds or 4 milliseconds, where the former signifies a data bit 0 and the latter a data bit 1. A crude example for passing the character 'd' can be seen below:



After a whole byte, in the figure above's example - 'd', has been passed a bit 0 is sent and no information is passed again for the next 10 miliseconds, which can technically be seen as a stop bit.

In the meantime the receiver evaluates each bit it receives (keeping in mind that there is plenty of data noise in the air, which means it receives more data than just the one coming from the D1 mini) and disregards the data that come in faster than 5 and 500 **micro**seconds respectively for a single bit and a full byte. This means that in a system as fast as the D1 mini there is a lot of error margin, as it will be very precise in its timing, so a lot of garbage data can sneak in between each bit. For this reason the timing had to be changed. Empirically, it was found out that transmitting a byte every 40 milliseconds and disregarding any data that comes before the 40 milliseconds mark and subsequently transmitting a bit every 2 milliseconds, but disregarding any bits that come before a 1.5 millisecond mark makes the communication near instantaneous and all data is correctly received.

Conclusion

Understanding the code for both receiver and transceiver better helped a lot and manipulating the time margins by hand, although time-consuming proved to be the right choice, as data flow was established

4. State behaviour and final touches

Reasoning

Even though communication was established, this was still a first step towards the end goal. After all the above three steps only made sure that the devices can talk to each other one way, but this had to be done bidirectionally and furthermore connection to WiFI had to be set up. Adding state behavior was loosely based on the state machine, (which probably had to be a sequence diagram) created by the author beforehand. An issue in in the receiving state in the D1 Mini was observed, where the loop implemented was too time-consuming and was blocking background processes. To this end a yield() is called in the middle of the code. It acts similarly to a scheduler, where the background processes are allowed to continue for a brief period. This adds about 7-8 microseconds to the communication, but thankfully this has not been observed to cause a problem. In the end, the D1 Mini has 5 states, while the Raspberry Pi has only 2. In the former the first state waits for a button to be pressed, afterwards it goes to the transmit state, where it transmits its start-of-communication string, which is currently aptly chosen to be “Hello“. This transmission lasts for 5 seconds, before it automatically goes to the next state in which it will wait to receive a packet from the Pi. This timeout is chosen with a simple state change in mind. After it has received the packet, the D1 Mini will go to a state in which it will connect to the wifi, the information for which it gets in the previous state. Finally it will go to the connected state, which currently has no implementation, but is there to symbolise the end of the current communication. The Raspberry Pi on the other hand will first go to the receiving state and after receiving “Hello“ to transmitting. In transmission it will send out a string that that holds the WiFi credentials in the following form “<SSID>:<Password>”, without the captions. The transmission will end after 10 seconds, when the Raspberry Pi program will exit gracefully.

Conclusion

To surmise, this phase ends the implementation of the 1-Click-Connect, which can easily be modified in the future to send out different messages as for example a Private Key for a secure connection through WiFi.

Conclusion and recommendations

In conclusion, these two programs enable the user to make their custom device connect to the WiFi with only one click. Although there is definitely room for improvement, as there is a small chance that the communication coming from the D1 Mini might not be acquired in the 5-seconds margin that was implemented. It is important to note that if both devices are transmitting at the same time the data will get jumbled, so if an improvement of the above-mentioned problem is made in the form of threading, that will need to be kept in mind. Other improvements are for example adding standard encryption to the data the same way it is done in the programs used for inspiration and also adding a PK to the Wifi string, so as to ensure secure connection later in the Wifi stage. Lack of time was the main reason, why these easy updates have not been added to the code. Nvertheless, the two programs give the desired functionality in a clean and barebone way. The challenge that came with this assignment was very much welcomed by the author and helped him learn a great deal about the world of communication protocols.