Group T: Jirad Al Massri (32323), Nour-Hamed-Raafat Hamed (31348) and Tarik Aydin (26751)

Chicken Check

1. Introduction

Written by Nour-Hamed-Raafat Hamed

In monitoring chickens there's a lot of different parameters are measured to gain insight on chicken movement and behaviour and a more complex objective of monitoring health indicators like feeding and nesting habits. Where are the chicken? In the chicken coop? Sitting on the nest? Breeding? Outside?.

Not only monitoring the chickens but also to track the long-term development of behaviour tracking for example it can help to register alarming trends and changes early on such that appropriate actions can be taken. This makes chicken checker a topic of importance in sustainable farming The Chicken Checker project uses Radio Frequency Identification (RFID) technology to monitor and track the activity of chickens on a farm as shown in figure 1. By connecting two antennas to a ESP32 and placing them at the start and end of the entrance of the chicken coop Microcontrollers are a valuable tool for farm owners and agricultural industries as they are cheap, small in size and easy to set up and use. They offer a lot of possibilities to help maintain a sustainable farming environment like reading results and transmitting the data to database where it can be assessed. The aim of this project is to scheme and program a small monitoring station for a chicken coop and learn how such technology works and its upsides and downside and how to set it up.

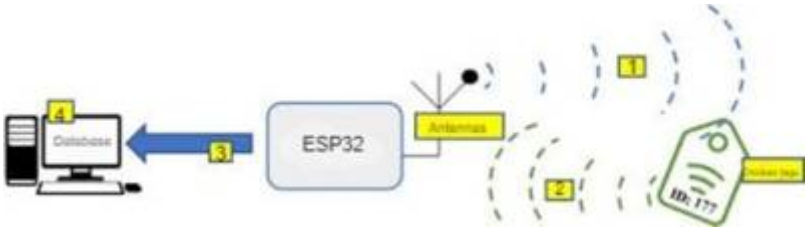

Figure 1 scheme of hardware on the farm

2. Materials & Methods

Written by Tarik Aydin

For this project the following materials were used:

1) PN532 Adafruit board

The PN532 Adafruit board operates on a 13.56MHz frequency and the PN532 chip is commonly used in phones which are able to do NFC. This board creates an electromagnetic field which can reach approximately up to 10cm range according to Adafruit [2], tested around 7.5cm where reached. In

total two board were used for this project, the boards act as gates which detect a nfc card when the nfc card passes the gates.

2) NFC cards type ISO14443 tags
The NFC cards acted passing chickens, the card consist of a small chip which is connected to a wire which is curled inside a plastic card. The card is a read by passive induction which is caused by the electromagnetic field of the PN532 board. The card can store up to 1KiloByte of data and has an unique identifier integrated in to the chip. These type of chips are usually used in train or bus passes [3].

3) Esp32 S3 Dev Module
The Esp32 S3 Dev Module worked as the brain of this project, the microcontroller controls the two PN532 boards and also sends the data to the computer. The Esp32 S3 Dev Module also has built in Wi-Fi, Bluetooth and low power functions, these function enable to collect the data automated viva Wi-Fi or Bluetooth [4].

4) Jumper wires
Jumper wires connect the parts used electronically. In total five male to male Jumper wires and twelve male to female jumper wires were used.

5) Bread board
The bread board was used to as a base to connect all the parts together.

6) USB-A to USB 2.0 micro B cable
This cable was used a simple data transmission cable from the computers serial port to the Esp32 S3 Dev module to program the microcontroller and to execute the code written in Arduino [1].

7) Soldering Iron
The PN532 boards were delivered with lose pins, in order to secure the pins in the wanted place an soldering iron was used.

8) ChatGPT
ChatGPT is an AI tool which is able to answer questions in a conversational way and allows answer follow up questions. ChatGPT enables for a faster development of the Arduino code to program the ESP32 microcontroller and the PN532 boards. But always be cautions with answer from ChatGPT, because answer can be sounding plausible which are incorrect [5].

3. Results

Written by Tarik Aydin

The code used for this project:

```
#include <Wire.h>
#include <SPI.h>
#include <Adafruit_PN532.h>

// Define the slave select (SS) pins for the PN532 modules
#define PN532_SS1 5
#define PN532_SS2 1

// Create PN532 SPI instances
Adafruit_PN532 nfc1(PN532_SS1);
Adafruit_PN532 nfc2(PN532_SS2);

struct UIDCounter {
  uint8_t uid[7];
  uint8_t uidLength;
  unsigned long count;
};

UIDCounter uidCounters1[10]; // Array to store UIDs and counts for PN532 #1
UIDCounter uidCounters2[10]; // Array to store UIDs and counts for PN532 #2

void setup(void) {
  Serial.begin(115200);
  Serial.println("Hello!");

  // Set up the SPI pins
  SPI.begin(13, 12, 11); // SCK, MISO, MOSI

  // Initialize both PN532 modules
  nfc1.begin();
  nfc2.begin();

  // Check if PN532 modules are ready
  uint32_t versiondata = nfc1.getFirmwareVersion();
  if (!versiondata) {
    Serial.println("Didn't find PN532 module #1");
    while (1);
  }

  versiondata = nfc2.getFirmwareVersion();
  if (!versiondata) {
    Serial.println("Didn't find PN532 module #2");
    while (1);
  }

  // Configure PN532 modules to read RFID cards
  nfc1.SAMConfig();
  nfc2.SAMConfig();
```

```
  // Initially set passive activation retries to zero (turn off antenna) for
both modules
  nfc1.setPassiveActivationRetries(0x00);
  nfc2.setPassiveActivationRetries(0x00);
}

void loop(void) {
  uint8_t success1;
  uint8_t uid1[7] = { 0 };  // Buffer to store the returned UID
  uint8_t uidLength1;  // Length of the UID (4 or 7 bytes depending on
ISO14443A card type)

  uint8_t success2;
  uint8_t uid2[7] = { 0 };  // Buffer to store the returned UID
  uint8_t uidLength2;  // Length of the UID (4 or 7 bytes depending on
ISO14443A card type)

  // Activate the first PN532 module and read
  nfc1.setPassiveActivationRetries(0x0A); // Set retries to 10 for the first
module
  digitalWrite(PN532_SS1, LOW);
  success1 = nfc1.readPassiveTargetID(PN532_MIFARE_ISO14443A, uid1,
&uidLength1);
  digitalWrite(PN532_SS1, HIGH);
  nfc1.setPassiveActivationRetries(0x00); // Disable antenna for the first
module

  if (success1) {
    updateUIDCounter(uidCounters1, uid1, uidLength1);
    printUIDs(uidCounters1, "PN532 #1");
  }

  // Add a small delay to ensure the second board does not activate too soon
  delay(50);

  // Activate the second PN532 module and read
  nfc2.setPassiveActivationRetries(0x0A); // Set retries to 10 for the
second module
  digitalWrite(PN532_SS2, LOW);
  success2 = nfc2.readPassiveTargetID(PN532_MIFARE_ISO14443A, uid2,
&uidLength2);
  digitalWrite(PN532_SS2, HIGH);
  nfc2.setPassiveActivationRetries(0x00); // Disable antenna for the second
module

  if (success2) {
    updateUIDCounter(uidCounters2, uid2, uidLength2);
    printUIDs(uidCounters2, "PN532 #2");
  }
```

```
  // Add a delay to control the loop timing

}

void updateUIDCounter(UIDCounter *uidCounters, uint8_t *uid, uint8_t
uidLength) {
  for (int i = 0; i < 10; i++) {
    if (uidCounters[i].count == 0) {
      // Empty slot, add new UID
      memcpy(uidCounters[i].uid, uid, uidLength);
      uidCounters[i].uidLength = uidLength;
      uidCounters[i].count = 1;
      return;
    } else if (memcmp(uidCounters[i].uid, uid, uidLength) == 0) {
      // UID found, increment counter
      uidCounters[i].count++;
      return;
    }
  }
}

void printUIDs(UIDCounter *uidCounters, const char *label) {
  Serial.print(label);
  Serial.println(" - Detected UIDs:");
  for (int i = 0; i < 10; i++) {
    if (uidCounters[i].count > 0) {
      Serial.print("UID: ");
      for (uint8_t j = 0; j < uidCounters[i].uidLength; j++) {
        if (uidCounters[i].uid[j] <= 0xF)
          Serial.print("0");
        Serial.print(uidCounters[i].uid[j], HEX);
        if (j < uidCounters[i].uidLength - 1)
          Serial.print(" ");
      }
      Serial.print(" Count: ");
      Serial.println(uidCounters[i].count);
    }
  }
  Serial.println("--------------");
}
```

Explanation of the code:

1. In the first part of the code the libraries and the chip select pins are defined.
    1. The libraries used are <Wire.h>, <SPI.h> and <Adafruit_PN532.h>
    2. The chip select pins are set at pin 5 for the first PN532 board and at pin 1 for the second PN532 board is set.

2. The Adafruit_PN532 nfc function creates instances which define which pin has to be used for which board. Two PN532 boards were used in this project so two instances have to be created. This enables to distinguish the two boards.
3. Then the struct UIDCounter is defined which enables to store read UIDs and also tracks how many times a specific UID has been read.
    1. The UIDCounter uidCounters1[10] and the UIDCounter uidCounters2[10] are storing for each board individually the UIDs detected for the corresponding board.
4. The SPI pins are defined in the set up.
    1. Pin 13 is defined for the serial clock.
    2. Pin 12 is defined for MISO.
    3. Pin 11 is defined for MOSI.
5. The nfc1.begin() and nfc2.begin() functions initializes the communication for the boards.
6. The code uint32_t versiondata = nfc1.getFirmwareVersion(); if (!versiondata) checks if the boards are connected correctly and returns an message in the serial monitor when the boards not found.
7. The functions nfc1.SAMConfig() and nfc2.SAMConfig() configures the board to use the Secure Access Module which enables to read RFID cards.
8. The last functions nfc1.setPassiveActivationRetries(0x00) and nfc2.setPassiveActivationRetries(0x00) in the void set up turns the antenna of the PN532 boards off.
9. The void loop contains variable storing functions success1, success2 as well as control of the two PN532 boards. As in 8. The function nfc.setPassiveActivationRetries() is used. Here in the beginning the function sets the PN532 board up to try ten times to read a nfc card. The digitalWrite(PN532_SS1, LOW) pulls the SPI signal down starting the SPI communication of the PN532 board. After digitalWrite(PN532_SS1, LOW) comes a variable which is called Success1 which is using the nfc1.ReadPassiveTargetID(), which will read the card when the card is close enough. After the Success1 comes the function digitalWrite(PN532_SS1, HIGH) which stops the SPI communication, after that the antenna is turned off again with the nfc.setPassiveActivationRetries() function. If a card is successfully read the functions updateUIDCounter and printUIDs , the updateUIDCounter updates the UID count at the perspective board and the function printUIDs will print the UIDs in the Serial monitor.
10. Small delay of 50ms is added to make sure that the second board starts bit later which ensures that the boards do not interfere with each other. Point nine is then done again just with changed variable names for the second board.

Generation of the code:

The code used for the project was generated in a conversation with ChatGPT, small adaptations were made in the code. The AI added a delay which wasn’t necessary, as well as the delay of the second board was set a 100ms which would lead to interference of the two boards, because the first board tries ten times a second to read a card. The delay for the second board was changed to 50ms.

Results in the serial monitor in Arduino

In figure 2 the serial monitor is displayed. Here is shown that the boards each have an individual counter for the different UIDs and that the UIDs have an individual counter as well. This enables to see in which direction the chicken walks, because one of the boards will be placed before the

exit/entrance and the other board will be placed after the exit/entrance.

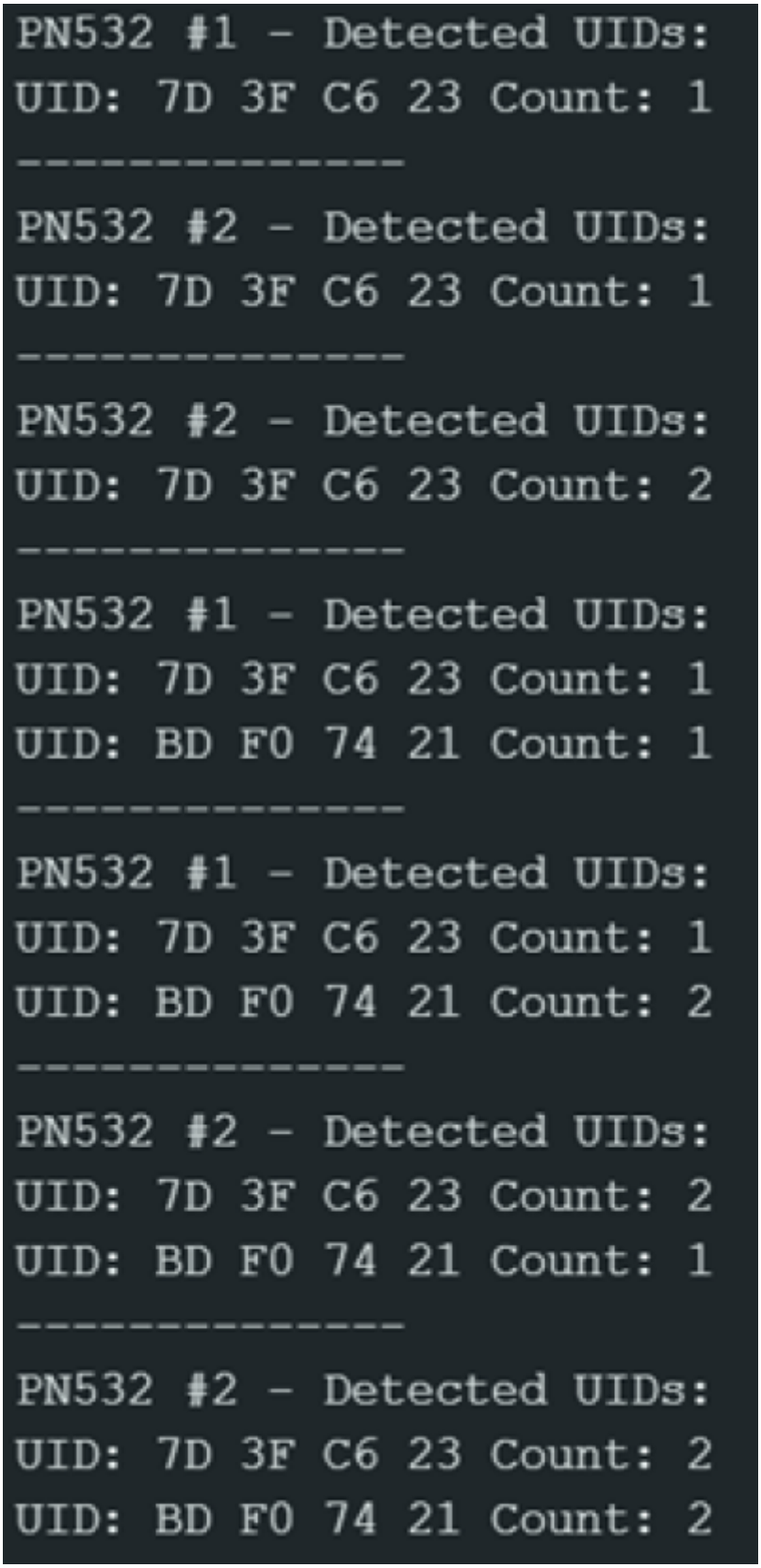

Figure 2 Serial Monitor in Arduino

Code that did not work for the project

The function nfc.readPassiveTargetID() is set as a standard to 0xFF [6], this means the function will run for ever. This is a problem when two PN532 boards are used, because in the void loop the function will be needed two times once for board one and once for board two. When the code is set up in the way that the first board comes first and the second board comes second, code will run in the sequence first board one and then board two. So, this allows only to start reading with the first boards and the code also will only read one signal at the time at one board. One way to avoid to be stuck in the sequence is to use the freeRTOS library, which will allow to create tasks in Arduino. In each task the function nfc.readPassiveTargetID() is used and still operates in the standard setting 0xFF. Here the two tasks will work in parallel, avoiding to sequence the nfc card. Using the freeRTOS library only solves the sequencing issue, the bigger issue here is then that the two boars will be permanently on. Which is causing electromagnetic interference.

4. Discussion

Written by Jihad Al Massri

First challenge was the usage of I2C, where it uses 7-bit to 10-bit addresses to make the identification of devices on the bus, with many PN532 address conflict can increase because of the limited number of unique addresses. In addition, data collision and bus contention can increase in an environment with high communication traffic because I2C use the same set of wires (SDA and SCL) to the communication of multiple devices. Switching to SPI was the choice since SPI uses separate Chip select for each PN532 that eliminate address conflict, make device management more easy. Moreover, SPI enable a fast communication between the microcontroller and the PN532 because it has a higher data rate as well as it allows simultaneous data transmission and reception. The usage of SPI facilitate the addition of new PN532 to the system since its only require to identify a specific chip select line for every new PN532.
Second challenge was the electromagnetic interference. When the PN532 where on, they were interfering with each other, causing a disruption in their operation. This interference created a dead spot in the detection area because both boards were not able to read data. Additionally, these interference has weakened the detection range of each board that reduces their efficiency and reliability. This has affected the overall performance of the system, making difficulties to have accurate reading from the devices.

5. Conclusion

Written by Jihad Al Massri and Nour-Hamed-Raafat Hamed

In this project, the initial use of I2C communication for PN532 NFC/RFID has faced a challenge, to distinguish between multiple boards. The switch to SPI communication has solved the issue by the usage separate chip select lines that made the communication more reliable and stable. Despite these improvements, the interference between the boards is still a challenge that promotes to explore techniques such as physical separation and shielding. Because of the challenges we faced, a lot of insight was gained.
Technology Integration: Learned to combine ESP32 microcontrollers and PN532 with RFID tags

technology for tracking.

Understanding Antennas :
Gained knowledge on how antennas work with RFID systems and UID detection.

UID detection:
Gained knowledge on how NFC/RFID tags interacts with readers like the PN532.

System Design:
As a group a lot of experience was gained through trail and troubleshooting integrated systems involving hardware and software parts of the project.

I2C and SPI:
I2C has a communication serial line to all antennas (sda scl), and for the SPI can distinguish between the antennas through chip select.

Future work:
Where to pick up this project? The main focus should be on advanced method to reduce electromagnetic interference as this was the main challenge in our project , improve system flexibility where it can track multiple behaviors like chicken nesting habits and health tracking, and real world testing to apply the methods and then accordingly make changes in hardware or software.

6. References

1. https://www.anker.com/blogs/cables/how-to-identify-different-types-of-usb-cables-a-brief-guide
2. https://www.adafruit.com/product/789
3. https://www.adafruit.com/product/359
4. https://docs.espressif.com/projects/esp-idf/en/stable/esp32s3/hw-reference/esp32s3/user-guide-devkitc-1.html
5. https://openai.com/index/chatgpt/
6. https://github.com/adafruit/Adafruit-PN532/blob/master/Adafruit_PN532.cpp