Türklingel

Übersicht

Im Hackerspace ist am Eingang Bornstr. 14-15 eine Türklingel mit Sprechanlage (Türsprechstelle) installiert. Das Haustelefon befindet sich im Workshopraum.

Es wurde eine Möglichkeit gefunden, das Klingelsignal abzugreifen und in die Räume Bornstr. 16-17 zu verlängern (E- und Kreativwerkstatt).

Optional:

Es könnte zu jeder zeit an den Empfängern eine Li-ion Akku (via JST) angeschlossen werden und somit auch mobil betrieben werden z.b bei Veranstaltungen die im Hinnenhof stattfinden um kein Klingeln zu verpassen.

Achtung bei Powerbanks!

Da sich die meisten Powerbanks bei geringer Last ausschalten, könnte es sein, dass der Empfänger bei mobilen betrieb mit Powerbanks die Funktion nach kurzer zeit Einstellen (https://apfelhirn.de/automatische-abschaltung-von-powerbanks-als-arduino-stromversorgung-bei-geringer-grundlast-verhindern/).

Sprechanlage

Die Sprechanlage nutzt ein 2-Draht-Bussystem vom Typ STR QwikBus, welches die Türsprechstelle mit den Haustelefonen verbindet. Daher ist es nicht möglich, das Klingelsignal direkt abzugreifen, da es digital über den Bus übertragen wird.

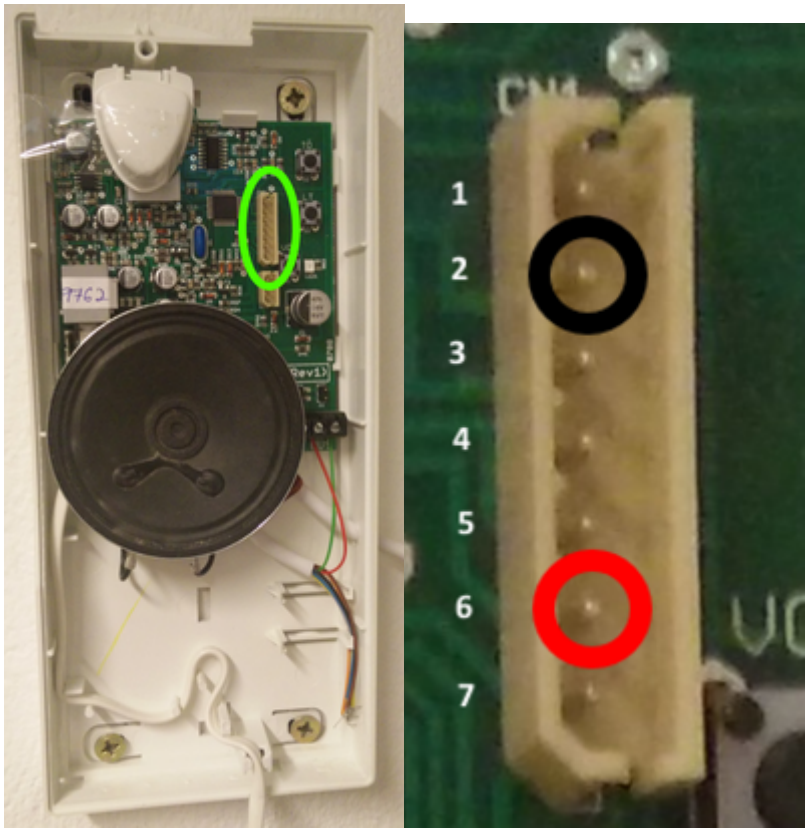
Das Haustelefon ist vom Typ STR HT 3033.

Das Steuergerät ist ein SP 333 (Doku), welches im Hauskeller im Sicherungsschrank neben dem Netzteil NH 333 installiert ist.

Pinout

Öffnet man das Haustelefon über die einzig vorhandene Gehäuseschraube (die obere Gehäusehälfte lässt sich dann herunterklappen), so kommen innen zwei Steckverbinder zum Vorschein.

Die Belegung lässt sich durch ein Foto einer Relaisbox AM333, die man hinzukaufen kann, erahnen. Ein Multimeter half bei der weiteren Analyse. Auf dem zweiten Pin von oben liegt Masse, auf dem zweiten Pin von unten wird das Klingelsignal ausgegeben (5 V, wenn ich mich recht erinnere).



Mechanisch konnten die passenden Steckerweibchen nicht ermittelt werden. Daher greifen wir das Signal über zwei IC-Beinchen-Einsätze gedrehter IC-Sockel ab, die in den Steckverbindern verlässlich halten.

Funk-Türklingel

Da uns keine Leitung zwischen den Räumen Bornstr. 14-15 sowie 16-17 zur Verfügung steht, mussten wir auf eine Funk-Türklingel mit zwei Empfängern zurückgreifen. Der Sender sowie die 2 Empfänger werden mit einem Standard 5 Volt USB Netzteil versorgt.

Änderungen am 04.02.2025:

Der Mosfet (siehe Bild) hat nicht mehr das Klingel Signal zum ESPNOW Sender weitergeleitet. Daher dedektiere ich das Klingelsignal nun über einen H11LX (Optokoppler) der beim Klingeln den Pin 04 LOW setzt und ein ESPNOW signal zu den Empfängern ,,Klingeln„ schickt.

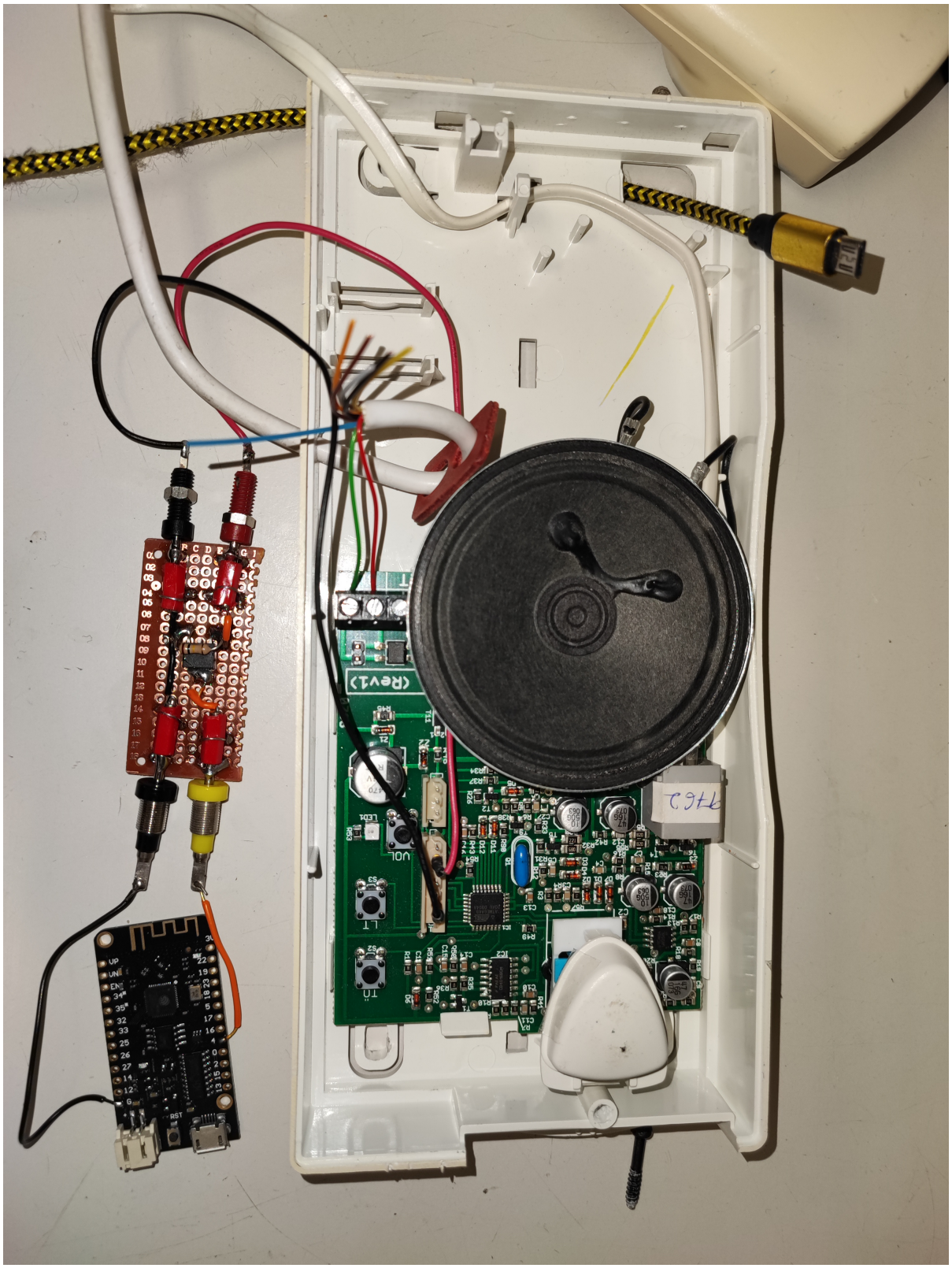
Zusätzlich habe ich den Empfängern mit einem Watchchdog und ein auto restart (jede Minute) ausgestattet.

Nachtrag am 05.02.2025:

Es kam gelegentlich zu Fehlauslösungen. Daher habe ich in der Software eingestellt, dass das Klingelsignal mindestens 1800 ms anliegen muss (In meinen Tests war das Signal nie kürzer als 2000 ms).

Sender in Haustelefon einbauen

Der Sender ist ein ESP32 Lolin Lite welcher auf ein Trigger auf Pin 4 wartet.



Sourcecode

Sender:

```
#include <WiFi.h>
#include <esp_wifi.h>
#include <esp_now.h>

// Set your new MAC Address
uint8_t newMACAddress[] = {0x30, 0xAF, 0xA0, 0x05, 0x1D, 0x63};
uint8_t broadcastAddress1[] = {0x30, 0xAF, 0xA1, 0x05, 0x1D, 0x63};
uint8_t broadcastAddress2[] = {0x30, 0xAF, 0xA2, 0x05, 0x1D, 0x63};

unsigned long TestpreviousMillis = 0;
unsigned long DebugpreviousMillis = 0;

const long Testinterval = 60000;
const long Debuginterval = 1000;

int testPin = 33;
int triggerPin = 4;

int testmode = 0;
int testmodeState = 0;
int triggerState = 0;

typedef struct bell_struct {
  uint64_t trigger;
} bell_struct;

bell_struct bell;

esp_now_peer_info_t peerInfo;

// Variable, die von der ISR gesetzt wird
volatile bool bellTriggered = false;

// Callback when data is sent
void OnDataSent(const uint8_t *mac_addr, esp_now_send_status_t status) {
  char macStr[18];
  snprintf(macStr, sizeof(macStr), "%02x:%02x:%02x:%02x:%02x:%02x",
           mac_addr[0], mac_addr[1], mac_addr[2], mac_addr[3], mac_addr[4],
mac_addr[5]);
  Serial.print("Packet to: ");
  Serial.print(macStr);
  Serial.print(" send status:\t");
  Serial.println(status == ESP_NOW_SEND_SUCCESS ? "Delivery Success" :
"Delivery Fail");
}
```

```
void setup() {
  Serial.begin(115200);

  WiFi.mode(WIFI_STA);

  // Set the ESP32 Board MAC Address
  esp_wifi_set_mac(WIFI_IF_STA, &newMACAddress[0]);
  Serial.print("[NEW] ESP32 Board MAC Address: ");
  Serial.println(WiFi.macAddress());

  if (esp_now_init() != ESP_OK) {
    Serial.println("Error initializing ESP-NOW");
    ESP.restart();
  }

  esp_now_register_send_cb(OnDataSent);

  // Register first peer
  peerInfo.channel = 0;
  peerInfo.encrypt = false;
  memcpy(peerInfo.peer_addr, broadcastAddress1, 6);
  if (esp_now_add_peer(&peerInfo) != ESP_OK) {
    Serial.println("Failed to add peer 1");
    return;
  }

  // Register second peer
  memcpy(peerInfo.peer_addr, broadcastAddress2, 6);
  if (esp_now_add_peer(&peerInfo) != ESP_OK) {
    Serial.println("Failed to add peer 2");
    return;
  }

  // Set pin 4 as an interrupt
  pinMode(triggerPin, INPUT);
  pinMode(testPin, INPUT_PULLUP);
  attachInterrupt(digitalPinToInterrupt(triggerPin), triggerbell, FALLING);
}

// Minimal Interrupt-Service-Routine
void triggerbell() {
  bellTriggered = true; // Setzt die Flag-Variable
}

void testbell() {
  bell.trigger = 4042190430;

  esp_err_t result = esp_now_send(0, (uint8_t *)&bell, sizeof(bell_struct));

  if (result == ESP_OK) {
    Serial.println("Sent with success");
```

```
  } else {
    Serial.println("Error sending the data");
  }

  Serial.print("[NEW] ESP32 Board MAC Address: ");
  Serial.println(WiFi.macAddress());
}

void loop() {
  unsigned long currentMillis = millis();

  testmodeState = digitalRead(testPin);
  triggerState = digitalRead(triggerPin);

  // Verarbeitung, wenn Interrupt ausgelöst wurde
  if (bellTriggered) {
    bellTriggered = false; // Zurücksetzen der Flag-Variable

    // Senden des Klingelsignals
    bell.trigger = 8562190430;

    esp_err_t result = esp_now_send(0, (uint8_t *)&bell,
sizeof(bell_struct));

    if (result == ESP_OK) {
      Serial.println("Sent with success");
    } else {
      Serial.println("Error sending the data");
    }

    Serial.print("[NEW] ESP32 Board MAC Address: ");
    Serial.println(WiFi.macAddress());
  }

  // Testmodus umschalten
  if (testmodeState == 0 && testmode == 0) {
    Serial.println("Klingelsendertestmodus ist aktiviert!");
    testmode = 1;
    TestpreviousMillis = Testinterval + currentMillis;
  }

  if (testmodeState == 1 && testmode == 1) {
    Serial.println("Klingelsendertestmodus wird deaktiviert!");
    testmode = 0;
    TestpreviousMillis = Testinterval + currentMillis;
  }

  // Debugging-Ausgabe
  if (currentMillis - DebugpreviousMillis>= Debuginterval) {
    DebugpreviousMillis = currentMillis;
    Serial.print("testmodeState: ");
```

```
    Serial.print(testmodeState);
    Serial.print(" - triggerState: ");
    Serial.println(triggerState);
  }

  // Testklingelsignal im Testmodus senden
  if (currentMillis - TestpreviousMillis>= Testinterval) {
    TestpreviousMillis = currentMillis;
    if (testmode == 1) {
      Serial.println("sende Testklingel signal (payload: 4042190430)");
      testbell();
    } else {
      Serial.println("warte auf klingeln, testmodus deaktiviert!");
    }
  }
}
```

Empfänger1

```
#include <WiFi.h>
#include <esp_wifi.h>
#include <esp_now.h>
#include <Ticker.h>

// Set MAC Address this device (bell receiver 1)
uint8_t newMACAddress[] = {0x30, 0xAF, 0xA1, 0x05, 0x1D, 0x63};
uint8_t addr[] = {0x30, 0xAF, 0xA0, 0x05, 0x1D, 0x63};
unsigned long lastPrintoutTime = 0;
unsigned long lastRebootTime = 0;
unsigned long lastDebugTime = 0;
const unsigned long DebugIntervall = 1000;
const unsigned long REBOOT_INTERVAL = 60000; // Periodischer Neustart nach
60 Sekunden

bool isTriggered = false;
unsigned long lastTriggeredTime = 0;

int selftest = 0;

typedef struct bell_struct {
  uint64_t trigger;
} bell_struct;

// Create a struct_message called myData
bell_struct myData;

//INITIALIZE WATCHDOG/////////
//watchdogCount==30 means it will wait 30 seconds before it will
automatically reset on a hang
//got this watchdog code from Andreas Spiess "The Guy with the Swiss Accent"
```

```
Ticker secondTick;
volatile int watchdogCount = 0;

void ISRwatchdog() {
  watchdogCount ++;
  if (watchdogCount>= 180) {
    Serial.println("crashed");
    ESP.restart();
  }
}
//END WATCHDOG

void printRestartReason() {
  esp_reset_reason_t reason = esp_reset_reason();
  Serial.print("Restart reason: ");
  switch (reason) {
    case ESP_RST_POWERON: Serial.println("Power on"); selftest = 1; break;
    case ESP_RST_EXT: Serial.println("External reset"); break;
    case ESP_RST_SW: Serial.println("Software reset"); break;
    case ESP_RST_PANIC: Serial.println("Exception/panic"); break;
    case ESP_RST_INT_WDT: Serial.println("Interrupt watchdog"); break;
    case ESP_RST_TASK_WDT: Serial.println("Task watchdog"); break;
    case ESP_RST_WDT: Serial.println("Other watchdog"); break;
    case ESP_RST_DEEPSLEEP: Serial.println("Deep sleep"); break;
    case ESP_RST_BROWNOUT: Serial.println("Brownout"); break;
    case ESP_RST_SDIO: Serial.println("SDIO"); break;
    default: Serial.println("Unknown"); break;
  }
}

void OnDataRecv(const esp_now_recv_info_t *info, const uint8_t
*incomingData, int len) {
  memcpy(&myData, incomingData, sizeof(myData));
  Serial.print("Bytes received: ");
  Serial.println(len);
  Serial.print("received command: ");
  Serial.println(myData.trigger);

  // Prüfen, ob die MAC-Adresse übereinstimmt
  if (memcmp(info->src_addr, addr, 6) == 0) {
    Serial.println("klingel device received!");
    if (myData.trigger == 8562190430) {
      Serial.println("valid Trigger code received!");
      triggerbell();
    }
  }
}

void setup() {
  Serial.begin(115200);
  Serial.println();
```

```
  secondTick.attach(1, ISRwatchdog);
  printRestartReason();

  WiFi.mode(WIFI_STA);
  Serial.print("[OLD] ESP32 Board MAC Address: ");
  Serial.println(WiFi.macAddress());

  esp_wifi_set_mac(WIFI_IF_STA, &newMACAddress[0]);
  Serial.print("[NEW] ESP32 Board MAC Address: ");
  Serial.println(WiFi.macAddress());

  pinMode(LED_BUILTIN, OUTPUT);
  pinMode(4, OUTPUT);

  if (selftest == 1) {
    Serial.println("selbsttest!");
    digitalWrite(LED_BUILTIN, LOW);
    digitalWrite(4, HIGH); // Trigger bell
    delay(100);
    digitalWrite(4, LOW);
    digitalWrite(LED_BUILTIN, HIGH);
    delay(100);
  } else {
    Serial.println("kein selbsttest!");
  }

  WiFi.mode(WIFI_STA);

  // ESP-NOW initialisieren
  if (esp_now_init() != ESP_OK) {
    Serial.println("Error initializing ESP-NOW");
    ESP.restart();
  }

  esp_now_register_recv_cb(OnDataRecv);

  lastRebootTime = millis(); // Zeit des letzten Neustarts speichern
}

void triggerbell() {
  if (!isTriggered || (millis() - lastTriggeredTime> 10000)) {
    isTriggered = true;
    lastTriggeredTime = millis();
    digitalWrite(LED_BUILTIN, LOW);
    digitalWrite(4, HIGH); // Trigger bell

    delay(100);
    digitalWrite(4, LOW);
    digitalWrite(LED_BUILTIN, HIGH);
    Serial.println("bell triggered");
  }
```

```
}

void loop() {

  // Reset der `isTriggered`-Variable nach 5 Sekunden
  if (millis() - lastTriggeredTime> 5000) {
    isTriggered = false;
  }

  // Periodischer Neustart alle 60 Sekunden
  if (millis() - lastRebootTime> REBOOT_INTERVAL) {
    Serial.println("Performing periodic reboot...");
    ESP.restart();
  }

  if (millis() - lastDebugTime> DebugIntervall) {
    lastDebugTime = millis();
    Serial.print("isTriggered: ");
    Serial.println(isTriggered);
  }

  watchdogCount = 0;
}
```

Empfänger2

```
#include <WiFi.h>
#include <esp_wifi.h>
#include <esp_now.h>
#include <Ticker.h>

// Set MAC Address this device (bell receiver 2)
uint8_t newMACAddress[] = {0x30, 0xAF, 0xA2, 0x05, 0x1D, 0x63};
uint8_t addr[] = {0x30, 0xAF, 0xA0, 0x05, 0x1D, 0x63};
unsigned long lastPrintoutTime = 0;
unsigned long lastRebootTime = 0;
unsigned long lastDebugTime = 0;
const unsigned long DebugIntervall = 1000;
const unsigned long REBOOT_INTERVAL = 60000; // Periodischer Neustart nach
60 Sekunden

bool isTriggered = false;
unsigned long lastTriggeredTime = 0;

int selftest = 0;

typedef struct bell_struct {
  uint64_t trigger;
} bell_struct;
```

```
// Create a struct_message called myData
bell_struct myData;

//INITIALIZE WATCHDOG/////////
//watchdogCount==30 means it will wait 30 seconds before it will
automatically reset on a hang
//got this watchdog code from Andreas Spiess "The Guy with the Swiss Accent"

Ticker secondTick;
volatile int watchdogCount = 0;

void ISRwatchdog() {
  watchdogCount ++;
  if (watchdogCount>= 180) {
    Serial.println("crashed");
    ESP.restart();
  }
}
//END WATCHDOG

void printRestartReason() {
  esp_reset_reason_t reason = esp_reset_reason();
  Serial.print("Restart reason: ");
  switch (reason) {
    case ESP_RST_POWERON: Serial.println("Power on"); selftest = 1; break;
    case ESP_RST_EXT: Serial.println("External reset"); break;
    case ESP_RST_SW: Serial.println("Software reset"); break;
    case ESP_RST_PANIC: Serial.println("Exception/panic"); break;
    case ESP_RST_INT_WDT: Serial.println("Interrupt watchdog"); break;
    case ESP_RST_TASK_WDT: Serial.println("Task watchdog"); break;
    case ESP_RST_WDT: Serial.println("Other watchdog"); break;
    case ESP_RST_DEEPSLEEP: Serial.println("Deep sleep"); break;
    case ESP_RST_BROWNOUT: Serial.println("Brownout"); break;
    case ESP_RST_SDIO: Serial.println("SDIO"); break;
    default: Serial.println("Unknown"); break;
  }
}

void OnDataRecv(const esp_now_recv_info_t *info, const uint8_t
*incomingData, int len) {
  memcpy(&myData, incomingData, sizeof(myData));
  Serial.print("Bytes received: ");
  Serial.println(len);
  Serial.print("received command: ");
  Serial.println(myData.trigger);

  // Prüfen, ob die MAC-Adresse übereinstimmt
  if (memcmp(info->src_addr, addr, 6) == 0) {
    Serial.println("klingel device received!");
    if (myData.trigger == 8562190430) {
      Serial.println("valid Trigger code received!");
```

```
      triggerbell();
    }
  }
}

void setup() {
  Serial.begin(115200);
  Serial.println();
  secondTick.attach(1, ISRwatchdog);
  printRestartReason();

  WiFi.mode(WIFI_STA);
  Serial.print("[OLD] ESP32 Board MAC Address: ");
  Serial.println(WiFi.macAddress());

  esp_wifi_set_mac(WIFI_IF_STA, &newMACAddress[0]);
  Serial.print("[NEW] ESP32 Board MAC Address: ");
  Serial.println(WiFi.macAddress());

  pinMode(LED_BUILTIN, OUTPUT);
  pinMode(4, OUTPUT);

  if (selftest == 1) {
    Serial.println("selbsttest!");
    digitalWrite(LED_BUILTIN, LOW);
    digitalWrite(4, HIGH); // Trigger bell
    delay(100);
    digitalWrite(4, LOW);
    digitalWrite(LED_BUILTIN, HIGH);
    delay(100);
  } else {
    Serial.println("kein selbsttest!");
  }

  WiFi.mode(WIFI_STA);

  // ESP-NOW initialisieren
  if (esp_now_init() != ESP_OK) {
    Serial.println("Error initializing ESP-NOW");
    ESP.restart();
  }

  esp_now_register_recv_cb(OnDataRecv);

  lastRebootTime = millis(); // Zeit des letzten Neustarts speichern
}

void triggerbell() {
  if (!isTriggered || (millis() - lastTriggeredTime> 10000)) {
    isTriggered = true;
    lastTriggeredTime = millis();
```

```
    digitalWrite(LED_BUILTIN, LOW);
    digitalWrite(4, HIGH); // Trigger bell

    delay(100);
    digitalWrite(4, LOW);
    digitalWrite(LED_BUILTIN, HIGH);
    Serial.println("bell triggered");
  }
}

void loop() {

  // Reset der `isTriggered`-Variable nach 5 Sekunden
  if (millis() - lastTriggeredTime> 5000) {
    isTriggered = false;
  }

  // Periodischer Neustart alle 60 Sekunden
  if (millis() - lastRebootTime> REBOOT_INTERVAL) {
    Serial.println("Performing periodic reboot...");
    ESP.restart();
  }

  if (millis() - lastDebugTime> DebugIntervall) {
    lastDebugTime = millis();
    Serial.print("isTriggered: ");
    Serial.println(isTriggered);
  }

  watchdogCount = 0;
}
```