Eine Netzwerkanbindung für unser Codeschloss (Teil 5)

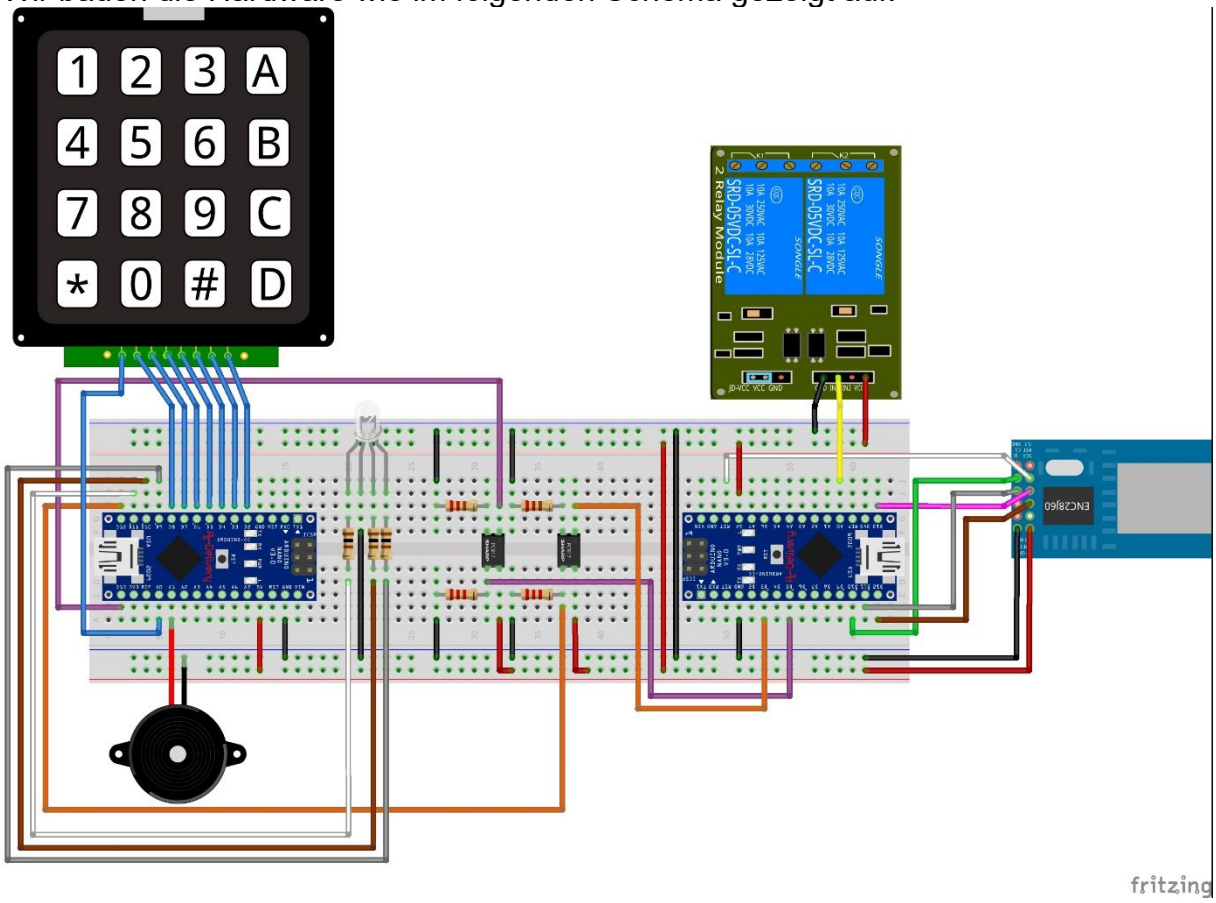
Hallo und willkommen zum vorletzten Teil unserer Codeschlossreihe. In den vorherigen Teilen sind wir hauptsächlich auf das Thema Sicherheit eingegangen und haben unserem Codeschloss ein paar interessante Sicherheitsverbesserungen, wie eine Trennung der Eingabe von der Steuereinheit, galvanische Trennung der Signalleitungen und eine proprietäre Verschlüsselung spendiert. Nun wollen wir uns in den letzten beiden Teilen mehr dem Benutzerkomfort in der Konfiguration des Systems widmen. Ein entscheidender, bisher noch gebliebener Nachteil unseres großartigen Codeschlosses werden wir im heutigen Teil der Reihe begegnen. Der starren unveränderlichen Konfiguration des Zugangspins! Denn:
Bisher ist es erforderlich, für jede Änderung des Pins die Firmware des Steuerteils neu zu kompilieren und auf den Steuerteil hochzuladen. Dies kann natürlich nicht auf Dauer so durchgehalten werden. Daher soll der gültige Codeschloss Pin im heutigen Teil per Netzwerk beliebig und jederzeit geändert werden können. Darüber hinaus soll der geänderte Pin dauerhaft im internen EEPROM gespeichert werden um beim nächsten Systemstart wieder zur Verfügung stehen. Um dieses hochgesteckte Ziel zu erreichen, spendieren wir unserer Auswerte Einheit das ENC28J60 Ethernet Shield für den Arduino Nano in einer der beiden Ausführungen:

1.) ENC28J60 Ethernet Shield für Arduino Nano V3.0
2.) ENC28J60 Ethernet Shield LAN Netzwerk Modul für Arduino

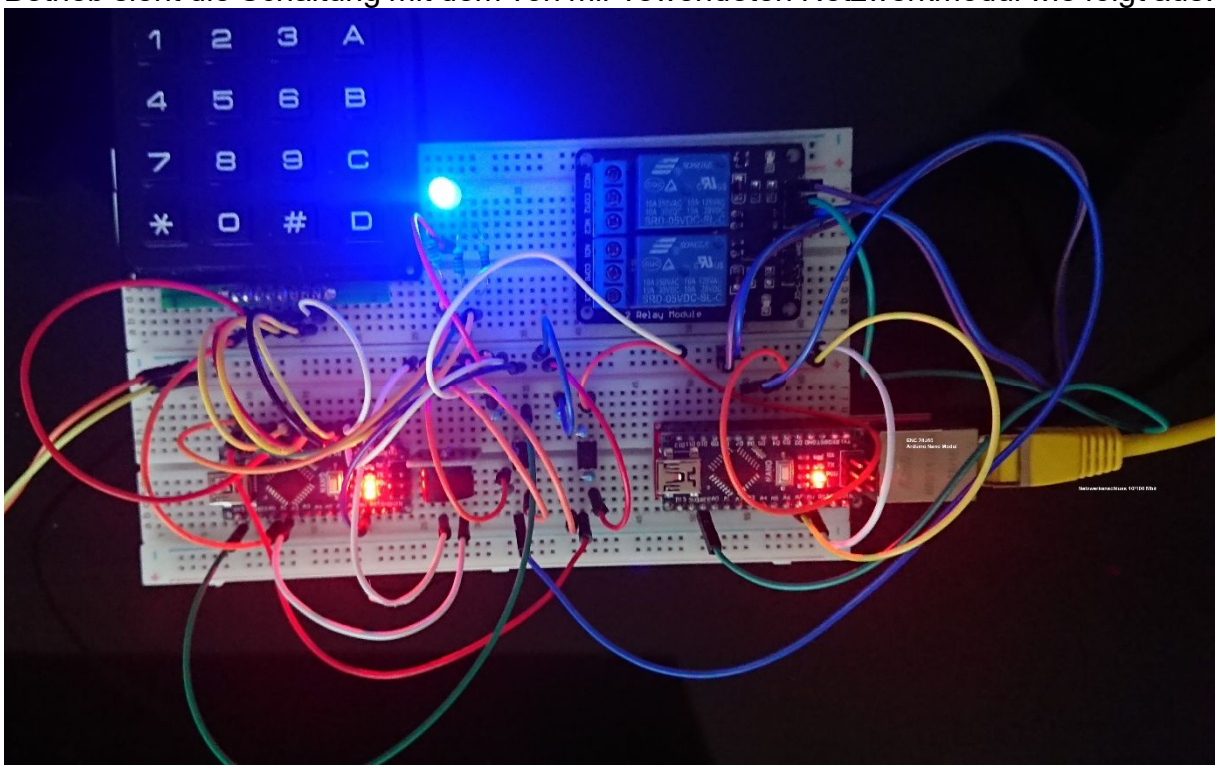
Wir brauchen also für den vorletzten Teil der Reihe folgende Hardware:

Anzahl	Bezeichnung	Anmerkung
1	Relais Modul	
2	Arduino Nano	
1	4x4 Keypad	
3	Widerstände 120 Ohm	
1	RGB Led	
2	Optokoppler Sharp PC817	
2	Widerstände 220 Ohm	Strombeg. Eing. Optokoppler
2	Widerstände 1 KOhm	Strombeg. Ausg. Optokoppler
1	ENC28J60 Ethernet f. A. Nano	
1	ENC28J60 Ethernet f. Arduino	Alternative

Wir bauen die Hardware wie im folgenden Schema gezeigt auf:



Ich empfehle die Verwendung des ENC28J60 Ethernet Shield für Arduino Nano in Version 3.0, da dieses Modul einen recht einfachen und platzsparenden Aufbau ermöglicht. Machfolgend verwende ich auch dieses Modul, da die komplette Verkabelung des Netzwerkmoduls entfällt. Aufgebaut auf einem Breadboard und in Betrieb sieht die Schaltung mit dem von mir vewendeten Netzwerkmodul wie folgt aus:



Wenn wir nun die Hardware funktionsfähig aufgebaut haben, können wir uns nun um die Programmierung des Ethernet Shields kümmern. Dazu benötigen wir zunächst die Bibliothek „UIPEthernet“ die wir von GitHub als .zip Datei herunterladen können. Sobald diese heruntergeladen wurde, können Sie die Bibliothek in Ihrer IDE unter: Sketch” ⇒ “Bibliothek einbinden” ⇒ “. ZIP-Bibliothek hinzufügen…” einbinden.
Damit stehen unserem Arduino nun die ganze Welt der Netzwerkkommunikation offen. Diese nutzen wir in unserem Falla dafür, um per Telnet oder Putty beliebig per Kommandozeile den Pin des Codeschlosses zu jedem Zeitpunkt ändern zu können.
Der neu definierte Pin wird danach im EEPROM des Auswerteteils abgespeichert, und steht bei einem Reset des Gesamtsystems wieder zur Verfügung.

Um nun einen neuen Code für das Codeschloss vergeben zu können, schließen wir im ersten Schritt das Codeschloss an das Netzwerk an und verbinden uns per seriellem Port mit 9600 Baud auf den Auswerteteil. Nach einer Weile wird die per DHCP bezogene Netzwerkadresse im Seriellen Monitor ausgegeben. Diese notieren wir uns für die Verbindung mithilfe des Programms „Putty“. In meinem Falle hat der Controller die IP-Adresse „172.20.0.158“ bekommen.

Achtung!

Sollten Sie kein Netzwerk während des Einschaltens des Codeschlosses angeschlossen haben, kann es etwas mehr als eine Minute dauern, bis das Codeschloss einsatzbereit ist, da in diesem Falle aufgrund einer Bibliothek internen Timeout Funktion die Begin Funktion der „UIPEthernet“ Bibliothek solange wartet, bis das Programm fortgesetzt wird. Um dies zu umgehen, kann auch eine feste IP-Adresse für das Codeschloss vergeben werden.

Wir verbinden und nun per „Putty“ oder einem anderen Telnet Programm auf Port 22 auf die IP-Adresse des Codeschlosses. Es öffnet sich ein schwarzes Fenster.
Nun geben Sie den Befehl CODEA: gefolgt von einem eigenen 1-20-stelligem Code ein und drücken Enter. Bei gültigem Zeichensatz und gültiger Länge wird ein "OK" ausgegeben, andernfalls ein "ERROR".

Beispiel oben: Ein gültiger Befehl währe CODEA:123 ein ungültiger dagegen währe COE. Gültige Zeigen für einen Code sind 0-9 und A-D, da die Eingabetastatur nur die Buchstaben A-D als Eingabe Bereich zur Verfügung stellt.

Nachdem ich nun die prinzipielle Funktionsweise und Administration des Codeschlosses erläutert habe, laden Sie bitte den jeweiligen aktualisierten Code auf den Controller:

Eingabeeinheit:

```
// Codeschloss Tobias Kuch 2020 GPL 3.0
// tobias.kuch@googlemail.com

#include <Keypad.h>
#include <SoftwareSerial.h>

#define RGBLED_R 11
#define RGBLED_G 10
#define RGBLED_B 9
#define RGBFadeInterval1  10       // in ms
#define KeybModeTimeInterval1 5000 // in ms
#define PIEZOSUMMER A1
#define CyclesInBlackMax 20

#define RGBOFF 0
#define RGBSHORTBLACK 8
#define RGBRED 1
#define RGBGREEN 2
#define RGBBLUE 3
#define RGBWHITE 4
#define RGBYELLOW 5
#define RGBCYAN 6
#define RGBMAGENTA 7


const byte ROWS = 4;
const byte COLS = 4;
const byte MaxPinCodeLength = 20;

SoftwareSerial mySerial(12, 13); // RX, TX

char keys[ROWS][COLS] = {
                         {49,50,51,65},
                         {52,53,54,66},
                         {55,56,57,67},
                         {58,48,59,68},
                        };
byte colPins[COLS] = {A0,8,7,6}; //A0,8,7,6;
byte rowPins[ROWS]= {5,4,3,2}; // 5,4,3,2}

byte RGBValue_R = 0;
byte RGBValue_G = 0;
byte RGBValue_B = 0;
byte RGBFadeValue_R = 0;
byte RGBFadeValue_G = 0;
byte RGBFadeValue_B = 0;
```

```
bool RGBFadeDir_R = true;
bool RGBFadeDir_G = true;
bool RGBFadeDir_B = true;


byte key = 0;
bool InSync = true;
bool CodeEnterSeqence = false;
bool CodeEnterSeqenceOLD = false;
bool InputBlocked = false;
bool PinEnteredFalseBefore  = false;
bool RGBFadeEnabled = true;

long previousMillis = 0;
long previousMillisKeyBoard = 0;
byte EnCodedKeyStroke = 0;
byte inByte = 0;
int CyclesInBlack = 0;
byte RecInititalKeyLength = 0;
unsigned long InititalKey = 0;


Keypad keypad = Keypad(makeKeymap(keys), rowPins, colPins, ROWS, COLS);

union foo {
  byte as_array[4];
  long as_long;
} d;


void setup()
{
 mySerial.begin(9600);
 Serial.begin(9600);
 pinMode(RGBLED_G,OUTPUT); // Ausgang RGB LED Grün
 pinMode(RGBLED_R,OUTPUT); // Ausgang RGB LED Rot
 pinMode(RGBLED_B,OUTPUT); // Ausgang RGB LED Blau
 pinMode(PIEZOSUMMER,OUTPUT); // Ausgang RGB LED Blau
 digitalWrite(PIEZOSUMMER,LOW); // Ausgang RGB LED Blau
 RGBControl(RGBWHITE,false); // NORMAL MODE
 RecInititalKeyLength = 0;
 do
  {
  if (mySerial.available())
    {
    inByte = mySerial.read();
    d.as_array[RecInititalKeyLength]=inByte;   //little Endian
    RecInititalKeyLength++;
    }
  } while (RecInititalKeyLength < 4);
 InititalKey = d.as_long;
```

```
 randomSeed(InititalKey);
 RGBControl(RGBBLUE,true); // NORMAL MODE
}

void RGBControl(byte function, bool fadeit)
{
 if (function == RGBOFF)
  {
  RGBValue_R = 0;
  RGBValue_G = 0;
  RGBValue_B = 0;
  RGBFadeValue_R = 0;
  RGBFadeValue_G = 0;
  RGBFadeValue_B = 0;
  RGBFadeDir_R = true;
  RGBFadeDir_G = true;
  RGBFadeDir_B = true;
  }
 if (function == RGBRED)
  {
  RGBValue_R = 255;
  RGBValue_G = 0;
  RGBValue_B = 0;
  RGBFadeValue_R = 255;
  RGBFadeValue_G = 0;
  RGBFadeValue_B = 0;
  RGBFadeDir_R = false;
  RGBFadeDir_G = true;
  RGBFadeDir_B = true;
  }
 if (function == RGBGREEN)
  {
  RGBValue_R = 0;
  RGBValue_G = 255;
  RGBValue_B = 0;
  RGBFadeValue_R = 0;
  RGBFadeValue_G = 255;
  RGBFadeValue_B = 0;
  RGBFadeDir_R = true;
  RGBFadeDir_G = false;
  RGBFadeDir_B = true;
  }
 if (function ==  RGBBLUE)
  {
  RGBValue_R = 0;
  RGBValue_G = 0;
  RGBValue_B = 255;
  RGBFadeValue_R = 0;
  RGBFadeValue_G = 0;
  RGBFadeValue_B = 255;
  RGBFadeDir_R = true;
```

```
 RGBFadeDir_G = true;
 RGBFadeDir_B = false;
 }
if (function == RGBWHITE)
 {
 RGBValue_R = 255;
 RGBValue_G = 255;
 RGBValue_B = 255;
 RGBFadeValue_R = 255;
 RGBFadeValue_G = 255;
 RGBFadeValue_B = 255;
 RGBFadeDir_R = false;
 RGBFadeDir_G = false;
 RGBFadeDir_B = false;
 }
if (function == RGBCYAN)
 {
 RGBValue_R = 0;
 RGBValue_G = 255;
 RGBValue_B = 255;
 RGBFadeValue_R = 0;
 RGBFadeValue_G = 255;
 RGBFadeValue_B = 255;
 RGBFadeDir_R = true;
 RGBFadeDir_G = false;
 RGBFadeDir_B = false;
 }
if (function == RGBYELLOW)
 {
 RGBValue_R = 255;
 RGBValue_G = 255;
 RGBValue_B = 0;
 RGBFadeValue_R = 0;
 RGBFadeValue_G = 0;
 RGBFadeValue_B = 0;
 RGBFadeDir_R = true;
 RGBFadeDir_G = true;
 RGBFadeDir_B = true;
 }
 if (function == RGBMAGENTA)
 {
 RGBValue_R = 255;
 RGBValue_G = 0;
 RGBValue_B = 255;
 RGBFadeValue_R = 255;
 RGBFadeValue_G = 0;
 RGBFadeValue_B = 255;
 RGBFadeDir_R = false;
 RGBFadeDir_G = true;
 RGBFadeDir_B = false;
 }
```

```
 if (function == RGBSHORTBLACK)
  {
  analogWrite(RGBLED_R, 0);
  analogWrite(RGBLED_G, 0);
  analogWrite(RGBLED_B, 0);
  }
RGBFadeEnabled = fadeit;
if (!(RGBFadeEnabled))
  {
  analogWrite(RGBLED_R, RGBValue_R);
  analogWrite(RGBLED_G, RGBValue_G);
  analogWrite(RGBLED_B, RGBValue_B);
  }
}


void SerialHandler ()
{

if (mySerial.available())
  {
  inByte = mySerial.read();
  if (inByte == 30) // Eingabe gesperrt Zeitschloss aktiv
    {
    InputBlocked = true;
    RGBControl(RGBRED,true);
    }
  if (inByte == 40) // Eingabe entsperrt Zeitschloss deaktiviert
    {
    RGBControl(RGBMAGENTA,true);
    InputBlocked = false;
    tone(PIEZOSUMMER, 880, 100);
    delay(120);
    }
  if (inByte == 20) // Code Correct
    {
    RGBControl(RGBGREEN,false);
    tone(PIEZOSUMMER, 1200, 200);
    delay(2000);
    PinEnteredFalseBefore  = false;
    RGBControl(RGBBLUE,true); // NORMAL MODE
    } else
    if (inByte == 21) // Code falsch
      {
      analogWrite(RGBLED_R, 255);
      analogWrite(RGBLED_G, 0);
      analogWrite(RGBLED_B, 0);
      tone(PIEZOSUMMER, 400, 300);
      delay(500);
      RGBControl(RGBRED,true);
      InputBlocked = true;
```

```
      PinEnteredFalseBefore  = true;
      }
  if (inByte == 25) // Out of Sync
    {
    RGBControl(RGBYELLOW,true);
    InSync = false;
    InititalKey = 0; // Delete Encryption Key
    }
  if (inByte == 23) //Clear ausgeführt
    {
    inByte = 0;
    }
  if (inByte == 22) // EIngabe azeptiert
    {
    inByte = 0;
    }
  }
}


void TimeMgmnt ()
{
if ((millis() - previousMillisKeyBoard > KeybModeTimeInterval1) &
CodeEnterSeqence & InSync) // Auto Reset KEyboard Input
  {
  previousMillisKeyBoard = millis();
  tone(PIEZOSUMMER, 988, 100);
  delay(110);
  if (PinEnteredFalseBefore)
    {
    RGBControl(RGBMAGENTA,true); // NORMAL MODE - Pin entered false before
    } else
    {
    RGBControl(RGBBLUE,true); // NORMAL MODE
    }
  CodeEnterSeqence = false;
  previousMillisKeyBoard = millis();
  byte randNumber = random(0, 254);
  EnCodedKeyStroke = 58 ^ randNumber;
  mySerial.write(EnCodedKeyStroke);
  }
if (millis() - previousMillis >  RGBFadeInterval1) //Fadint LEd's
  {
  if (RGBFadeEnabled)
    {
    previousMillis = millis();   // aktuelle Zeit abspeichern
    if (RGBValue_B > 0)
      {
      if (RGBFadeDir_B)
        {
        RGBFadeValue_B++;
```

```
        if ( RGBFadeValue_B >=  RGBValue_B) {RGBFadeDir_B = false; }
        } else
        {
        RGBFadeValue_B--;
        if ( RGBFadeValue_B < 1) {RGBFadeDir_B = true; }
        }
      } else { RGBFadeValue_B = 0; }
    if (RGBValue_R > 0)
      {
        if (RGBFadeDir_R)
        {
        RGBFadeValue_R++;
        if ( RGBFadeValue_R >=  RGBValue_R) {RGBFadeDir_R = false; }
        } else
        {
        RGBFadeValue_R--;
        if ( RGBFadeValue_R < 1) {RGBFadeDir_R = true; }
        }
      } else { RGBFadeValue_R = 0; }
    if (RGBValue_G > 0)
      {
      if (RGBFadeDir_G)
      {
      RGBFadeValue_G++;
      if ( RGBFadeValue_G >=  RGBValue_G) {RGBFadeDir_G = false; }
      } else
      {
      RGBFadeValue_G--;
      if ( RGBFadeValue_G < 1) {RGBFadeDir_G = true; }
      }
      } else { RGBFadeValue_G = 0; }
    analogWrite(RGBLED_R, RGBFadeValue_R);
    analogWrite(RGBLED_G, RGBFadeValue_G);
    analogWrite(RGBLED_B, RGBFadeValue_B);
    }
  }
}

void KeyboardHandler(bool NotEnabled)
{
key = keypad.getKey();
if((key)) // Key Entered
  {
  if (!NotEnabled)
    {
    byte randNumber = random(0, 254);
    EnCodedKeyStroke = key ^ randNumber;
    mySerial.write(EnCodedKeyStroke);
    if((key == 58) | (key == 59))
      {
        RGBControl(RGBSHORTBLACK,true);
```

```
        tone(PIEZOSUMMER, 988, 100);
        delay(120);
        CodeEnterSeqence = false;
        if(key == 58)
          {
          if (PinEnteredFalseBefore)
            {
            RGBControl(RGBMAGENTA,true); // NORMAL MODE - Pin entered false
before
            } else
            {
            RGBControl(RGBBLUE,true); // NORMAL MODE
            }
          }
      } else
      {
        RGBControl(RGBSHORTBLACK,true);
        tone(PIEZOSUMMER, 880, 100);
        delay(120);
        CodeEnterSeqence = true;
        RGBControl(RGBCYAN,true);
        previousMillisKeyBoard = millis();
      }
    }
  }
}


void loop()
{
if (InSync)
  {
  KeyboardHandler(InputBlocked);
  }
TimeMgmnt ();
SerialHandler ();
}
```

Auswerteeinheit

```
// Codeschloss Tobias Kuch 2020 GPL 3.0
// tobias.kuch@googlemail.com

#include <SoftwareSerial.h>
#include <UIPEthernet.h>
#include <EEPROM.h>

#define RELAIS_A A0
#define RELAIS_B A1
#define MACADDRESS 0x00,0x01,0x02,0x03,0x04,0x05
#define MYIPADDR 192,168,1,6
```

```
#define MYIPMASK 255,255,255,0
#define MYDNS 192,168,1,1
#define MYGW 192,168,1,1
#define LISTENPORT 23
#define UARTBAUD 9600

#define Interval1 1000
#define MAXDelayStages 7

const byte MaxPinCodeLength = 20;
const byte DelayInterationsInSec[MAXDelayStages] = {1,5,10,20,30,45,60};

SoftwareSerial mySerial(5, 3); // RX, TX
EthernetServer server = EthernetServer(LISTENPORT);
EthernetClient client;
EthernetClient newClient;

byte KeyPadBuffer[MaxPinCodeLength];
//byte PinCode[MaxPinCodeLength] = {1,2,3,13}; // Standard Pincode: 123A  - Bitte
Ändern gemäß Beschreibung -
byte BufferCount = 0;
byte a;
bool InSync = true;
bool AcceptCode =false;
byte ErrorCounter = 0;
long previousMillis = 0;
byte InputDelay = 0;
byte RecInititalKeyLength = 0;
uint8_t mac[6] = {MACADDRESS};
unsigned long CommuncationKey = 902841;

union foo {
  byte as_array[4];
  long as_long;
} convert;

struct EEPromData
  {
   byte PinCodeA[MaxPinCodeLength];
   char ConfigValid[3]; //If Config is Vaild, Tag "TK" is required"
  };

EEPromData MyPinCodes;


void SaveCodesToEEPROM ()
{
  for (int i = 0 ; i < sizeof(MyPinCodes) ; i++)
     {
      EEPROM.write(i, 0);
     }
```

```
  strncpy(MyPinCodes.ConfigValid , "TK", sizeof(MyPinCodes.ConfigValid) );
  EEPROM.put(0, MyPinCodes);
}

bool GetCodesFromEEPROM ()
{
 bool RetValue;
 EEPROM.get(0, MyPinCodes);
 if (String(MyPinCodes.ConfigValid) == String("TK"))
  {
    RetValue = true;
  } else
  {
    RetValue = false;
  }
  return RetValue;
}

void setup()
{
 Serial.begin(9600);
 mySerial.begin(9600);
 pinMode(RELAIS_A,OUTPUT); //Relais Output
 digitalWrite(RELAIS_A,HIGH); //LOW Aktiv
 if (GetCodesFromEEPROM())
  {
    Serial.println (F("Codes from EEPROM loaded."));
  } else
  {
    Serial.println (F("Empty EEPROM found. Set default Code."));
    MyPinCodes.PinCodeA[0] = 49; // Default Pincode: 123
    MyPinCodes.PinCodeA[1] = 50;
    MyPinCodes.PinCodeA[2] = 51;
  }
 BufferCount = 0;
 for (a = 0; a <= MaxPinCodeLength -1 ; a++)
  {
  KeyPadBuffer[a] = 0;
  }
 convert.as_long = CommuncationKey;
 RecInititalKeyLength = 0;
 Ethernet.begin(mac);
 server.begin();
 Serial.println(Ethernet.localIP());
 do
  {
  mySerial.write(convert.as_array[RecInititalKeyLength]);   //little Endian
  RecInititalKeyLength++;
  } while (RecInititalKeyLength < 4);

 randomSeed(CommuncationKey);
```

```
}

void loop()
{
 if (client = server.available())
  {
  byte Position = 0;
  bool CommandReceived = false;
  char EtherNetCommand[MaxPinCodeLength + 7] = "";
  while((client.available()) > 0)
    {
    byte thisChar = client.read();
    if ((thisChar < 123) & (thisChar > 47) & (Position < MaxPinCodeLength + 6) ) //
Sonderzeichen ausfiltern
      {
      EtherNetCommand[Position] = thisChar;
      Position++;
      if (Position > 1) { CommandReceived = true; }
      }
    }
     if (CommandReceived)
      {
       String s(EtherNetCommand);
     //  EtherNetCommand[MaxPinCodeLength + 7] = "";
      if (s.startsWith("CODEA:"))
        {
          s.remove(0, 6);
          Serial.println(F("Neuer Code"));
          byte a = s.length();
          bool CodeOk = true;
          for (Position = 0;Position < a;Position++) // Check auf gültige Zeichen
            {
            if ((char(s[Position])<48) | (char(s[Position])>68)| ((char(s[Position])>57) &
(char(s[Position])<65))) { CodeOk = false; }
            }
          if (CodeOk)
            {
            client.println(F("OK"));
            Serial.println(s);
            for (Position = 0;Position < MaxPinCodeLength;Position++) // Check auf
gültige Zeichen
              {
                 MyPinCodes.PinCodeA[Position] = 0;
              }

            for (Position = 0;Position < a;Position++) // Check auf gültige Zeichen
              {
                 MyPinCodes.PinCodeA[Position] = s[Position];
              }
            SaveCodesToEEPROM ();
            } else { client.println(F("ERROR")); }
```

```
        }
       }
    }

 if (client && !client.connected())
  {
  Serial.print(F("Client Disconnected"));
  client.stop();
  }

if (mySerial.available())
  {
   byte randNumber = random(0, 254);
   byte key = mySerial.read();
   byte DeCodedKeyStroke = key ^ randNumber;
   if (((DeCodedKeyStroke > 47) & (DeCodedKeyStroke < 69))  & InSync)
    {
     if(DeCodedKeyStroke == 58)   // Clear Keypad Buffer  Key: *
      {
       for (a = 0; a <= MaxPinCodeLength -1; a++)
        {
        KeyPadBuffer[a] = 0;
        }
       Serial.print(F("Clear "));
       Serial.println(BufferCount);
       mySerial.write(23);
       BufferCount = 0;
      } else
     if(DeCodedKeyStroke ==59)   // Enter Keypad Buffer  Key: #
      {
       if (InputDelay == 0)
         {
         //Serial.println("Auswertung gestartet");   // Zu Debugzwecken
         // Serial.println(BufferCount);
         AcceptCode = true;
         for (a = 0; a <= MaxPinCodeLength -1 ; a++)
           {
           if (!(MyPinCodes.PinCodeA[a] == KeyPadBuffer[a])) {AcceptCode = false; }
           //Serial.print(MyPinCodes.PinCodeA[a]); // Zu Debugzwecken
           //Serial.print(";");
           //Serial.print(KeyPadBuffer[a]);
           // Serial.println(" ");
           }
         // Serial.println("END");  // Zu Debugzwecken
         if (AcceptCode)
           {
           mySerial.write(20);
           digitalWrite(RELAIS_A,(!digitalRead(RELAIS_A)));
           ErrorCounter = 0;
           InputDelay = 0;
           AcceptCode = false;
```

```
          } else
          {
          mySerial.write(21);
          if ( ErrorCounter < MAXDelayStages - 1) { ErrorCounter++; }
          InputDelay = DelayInterationsInSec [ErrorCounter];
          }
        for (a = 0; a <= MaxPinCodeLength -1; a++) { KeyPadBuffer[a] = 0; }
        Serial.println(F("Clearing Memory"));
        BufferCount = 0;
        } else
        {
         Serial.println(F("Delay Mode Active"));
         mySerial.write(30);  // Delay Mode
         for (a = 0; a <= MaxPinCodeLength -1 ; a++) { KeyPadBuffer[a] = 0; }
         BufferCount = 0;
        }
      } else
      {
       KeyPadBuffer[BufferCount] = DeCodedKeyStroke;
       if (BufferCount < MaxPinCodeLength ) { BufferCount++; }
       if (InputDelay == 0) { mySerial.write(22); } else  { mySerial.write(30); }
      }
      } else
      {
        //Out of Sync
        Serial.print(F("Out of sync Data: "));
        Serial.println(DeCodedKeyStroke);
        mySerial.write(25);
        if ( ErrorCounter < MAXDelayStages - 1) { ErrorCounter++; }
        InSync = false;
      }
  }

if (millis() - previousMillis > Interval1)
{
  // Auto Reset KEyboard Input
 ;
   previousMillis = millis();
   if (InputDelay > 0)
    {
      if (InputDelay == 1)
      {
      Serial.println (F("Release"));
      mySerial.write(40);  // Delay Mode End
      }
     InputDelay = InputDelay - 1;
    }
  }
}
```

Ich wünsche viel Spaß mit dem Codeschloss und freue mich, demnächst Ihnen den letzten Teil der Reihe vorstellen zu dürfen, indem wir userem Codeschloss weiter Komfort-Administrationsmöglichkeiten über Netzwerkeingabe zur Verfügung stellen werden.