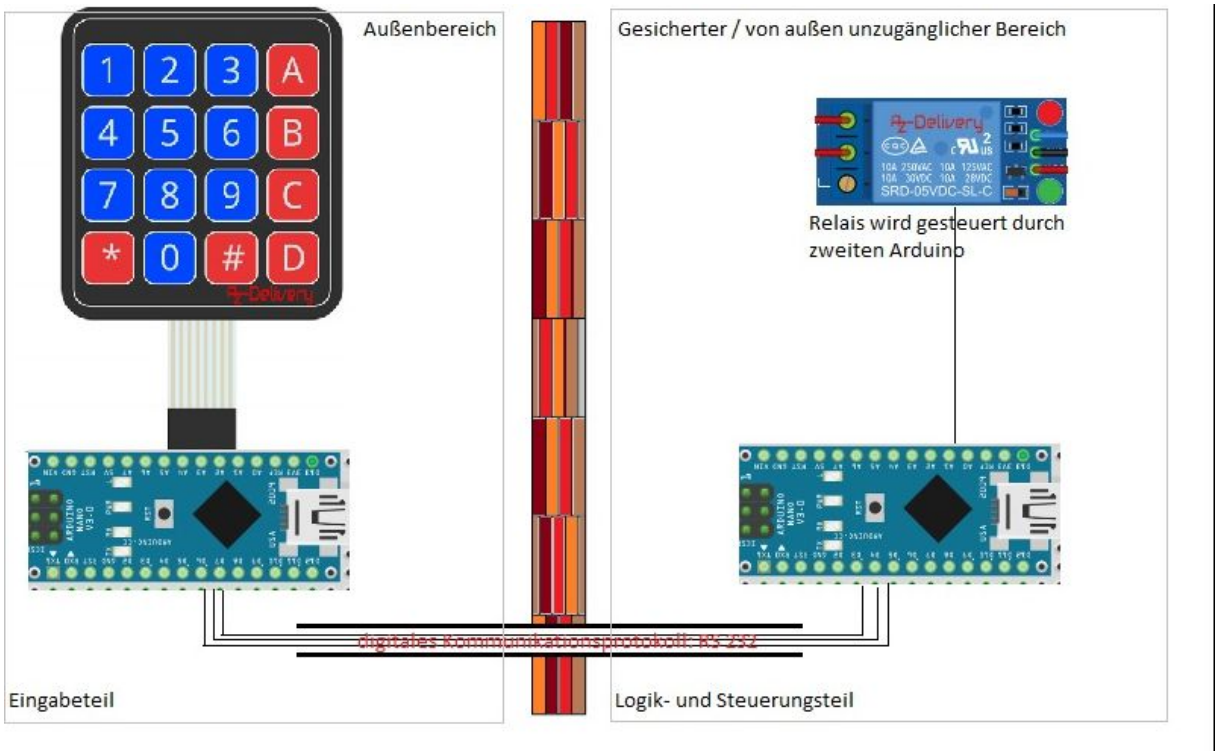
Multiprozessorkommunikation für unser Codeschloss (Teil 2)

Willkommen zum zweiten Teil unserer Codeschloss Reihe. Im heutigen Teil unserer Codeschlossreihe möchte ich auf ein mögliches, ernstes Sicherheitsproblem bei sehr einfach mechanisch und elektronisch aufgebauten Codeschlössern (zu denen auch unser Codeschloss im Teil 1 gehört) hinweisen und gleichzeitig im heutigen Teil der Reihe eine interessante erste Verbesserungsmöglichkeit vorstellen. Es soll zunächst jedoch einmal das Sicherheitsproblem von dem Codeschloss des erstens Teils beschrieben werden. Dies offenbart sich, wenn man sich die Funktionsweise und den lokalen Aufbau des Codeschlosses sich vor Augen führt.
Zur Funktionsweise: Der Arduino bzw. der Controller mit sämtlicher Auswertungslogik und einem Schaltrelais, dass die zu steuernde Last schaltet, ist oft in unmittelbarer räumlicher Nähe zu dem oft frei zugänglichen Eingabefeld angeordnet.
Ist das selbstgebaute Codeschloss inklusive Elektronik und Steuerrelais nun darüber hinaus nicht mechanisch gesichert oder in einem relativ einfachen (Plastik)-Gehäuse an dem zu sichernden Objekt angebracht, können unbefugte evtl. mit einfachen Werkzeugen an die innere Elektronik des Codeschlosses herankommen.
Dies kann sogar so weit gehen, dass das das Codeschloss nach Freilegung der Elektronik in seiner Funktionsweise nicht beeinträchtigt ist und noch funktioniert! In diesem Fall könnte ein technisch versierter unbefugter Nutzer das Lastrelais unter Umgehung der Codeeingabe direkt mit Strom versorgen oder den Strom abklemmen und so den Verbraucher unbefugt ein oder ausschalten. Eine Möglichkeit, dies zu erschweren (nicht aber zu verhindern), stellen ich heute in diesem Blog vor.

Ein Ansatz ist, den Eingabeteil von dem Logik- und Steuerungsteil zu trennen und räumlich getrennt, sicher zu platzieren. (Beispielsweise hinter einer Wand, einer Stahlverkleidung, oder auch in einem gesicherten Schaltkasten). Wichtig dabei ist, dass ein potentieller Angreifer baulich bedingt, keinen physischen Zugang zu dem Logik- und Steuerungsteil bekommt. Nachfolgende Grafik veranschaulicht das Prinzip:



Wie zu erkennen ist, wird zwischen dem Eingabeteil und dem Logik- und Steuerungsteil, die jeweils aus einem Arduino Nano bestehen, eine auf dem RS232 Protokoll basierende Kommunikation eingerichtet. So verarbeitet das Eingabeteil des Codeschlosses lediglich Tastendrücke und schickt diese ohne weitere Analyse dieser 1:1 nach einer beiden Seite bekannte, einfachen Codetabelle an den Logik- und Steuerungsteil. Dieser wertet die Tastenreihenfolge aus und schickt in Abhängigkeit des Ergebnisses wiederum Steuerbefehle für die RGB LED Anzeige zurück an das Eingabeteil und steuert das Verbraucherrelais Die RGB LED Anzeige dient dabei zur Rückmeldung an den Benutzer über den aktuellen Schaltzustand.
(in der oben gezeigten Prinzip Darstellung wurde die RGB Led am Eigebeteil zur Verständniserleichterung weggelassen).

Hinweis:
Obwohl wir und mit diesem Teil und auch mit den folgenden Teilen mit dem Thema der Sicherheit beschäftigen, und diese auch im nächsten Teil noch steigern werden, beachtet bitte beim Nachbau und beim Einsatz folgendes: Absolute Sicherheit gibt es nicht! Auch im heutigen Teil des Codeschlosses gibt es Schwachpunkte des Systems. Daher empfiehlt der Autor des Blogs keine Nutzung des Codeschlosses in von fremden zugänglichen Bereichen, in sicherheitskritischen Bereichen oder zur Absicherung von Wertgegenständen.
Die Haftung für Schäden wird ausgeschlossen.

Kommen wir aber nun zum Aufbau unseres heutigen Projektes. Wir benötigen im heutigen Teil an Hardware:

Anzahl	Bezeichnung	Anmerkung
1	Relais Modul	
2	Arduino Nano	
1	4x4 Keypad	
3	Widerstände 120 Ohm	
1	RGB Led	

Wir bauen die Schaltung wie auf folgender gezeigter Fritzing Zeichnung auf und verbinden die beiden Controller über eine Softwareseitige Serielle Schnittstelle:

Zu beachten ist das TX des Controllers 1 an RX des Controllers 2 und vice versa angeschlossen werden muss.

TX <-> RX
RX <-> TX

Nachdem wir alles verkabelt haben können wir nun folgenden Code auf Controller 1 (Eingabeteil) hochladen:

```
// Codeschloss Tobias Kuch 2020 GPL 3.0
#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] = {
                         {1,2,3,13},
                         {4,5,6,14},
                         {7,8,9,15},
                         {10,11,12,16},
                        };
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);

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
 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 = 10 ^ 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)
    {
    mySerial.write(key);
    if((key == 10) | (key == 12))
      {
        RGBControl(RGBSHORTBLACK,true);
        tone(PIEZOSUMMER, 988, 100);
        delay(120);
        CodeEnterSeqence = false;
        if(key == 10)
          {
          if (PinEnteredFalseBefore)
            {
            RGBControl(RGBMAGENTA,true); // NORMAL MODE
            } 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 ();
}
```

Danach kann der Code von Controller 2 (Logik- und Steuerungsteil) hochgeladen werden:

```
#include <SoftwareSerial.h>
#define RELAIS_A A0
#define Interval1 1000
#define MAXDelayStages 7

const byte MaxPinCodeLength = 20;
SoftwareSerial mySerial(2, 3); // RX, TX

byte KeyPadBuffer[MaxPinCodeLength];
byte PinCode[MaxPinCodeLength] = {1,2,3,13}; // Standard Pincode: 123A  - Bitte Ändern gemäß Beschreibung -
byte BufferCount = 0;
byte a;
bool AcceptCode =false;
long previousMillis = 0;

void setup()
{
 Serial.begin(9600);
 mySerial.begin(9600);
 pinMode(RELAIS_A,OUTPUT); //Relais Output
 digitalWrite(RELAIS_A,HIGH); //LOW Aktiv
}

void loop()
{
if (mySerial.available())
  {
   byte DeCodedKeyStroke = mySerial.read();
     if(DeCodedKeyStroke == 10)   // Clear Keypad Buffer  Key: *
     {
      for (a = 0; a <= MaxPinCodeLength -1; a++)
       {
       KeyPadBuffer[a] = 0;
       }
      Serial.print("Clear ");
      Serial.println(BufferCount);
      mySerial.write(23);
      BufferCount = 0;
     } else
    if(DeCodedKeyStroke ==12)   // Enter Keypad Buffer  Key: #
     {
       Serial.println("Auswertung gestertet");
       Serial.println(BufferCount);
       AcceptCode = true;
       for (a = 0; a <= MaxPinCodeLength -1 ; a++)
         {
         if (!(PinCode[a] == KeyPadBuffer[a])) {AcceptCode = false; }
         Serial.print(PinCode[a]);
```

```
          Serial.print(";");
          Serial.print(KeyPadBuffer[a]);
          Serial.println(" ");
          }
        Serial.println("END");
        if (AcceptCode)
          {
          mySerial.write(20);
          digitalWrite(RELAIS_A,(!digitalRead(RELAIS_A)));
          AcceptCode = false;
          } else
          {
          mySerial.write(21);
          }
        for (a = 0; a <= MaxPinCodeLength -1; a++) { KeyPadBuffer[a] = 0; }
        Serial.println("Clear all Memory");
        BufferCount = 0;
      } else
      {
       KeyPadBuffer[BufferCount] = DeCodedKeyStroke;
       if (BufferCount < MaxPinCodeLength ) { BufferCount++; }
       mySerial.write(22);
      }
  }
}
```

Nun können wir durch die Eingabe des Codes 123A das Relais ein und ausschalten. Der Pincode zum schalten des Relais kann in der Zeile:

```

```

Im Code von Controller 2 (Logik- und Steuerungsteil) geändert werden.

Ich wünsche viel Spaß mit unserem in der Sicherheit verbesserten Codeschloss und freue mich schon euch eine weitere Verbesserung der Sicherheit im nächsten teil vorstellen zu können.