Szczecińska Szkoła Wyższa

Collegium Balticum

rok akademicki: 2020/2021

Podstawy programowania układów wbudowanych
Projekt: Uniwersalny translator sygnałów IR

Dominik Dąbek

Rok studiów: **I**

Semestr**: II**

Kierunek: **Informatyka**

1 CONTENTS

1. Opis Projektu 3
1.1 Założenia projektu: 3
2 Schemat układu 3
3 BOM 4
4 Symulacja Układu 4
5 Realizacja układu 5
6 Programowanie 5
7 Podsumowanie 7
7.1 Lessons Learned: 7
7.2 Zakończenie 7
8 Dodatki 7
8.1 Spis rysunków: 7
8.2 Kod programu: 7

1. Opis Projektu

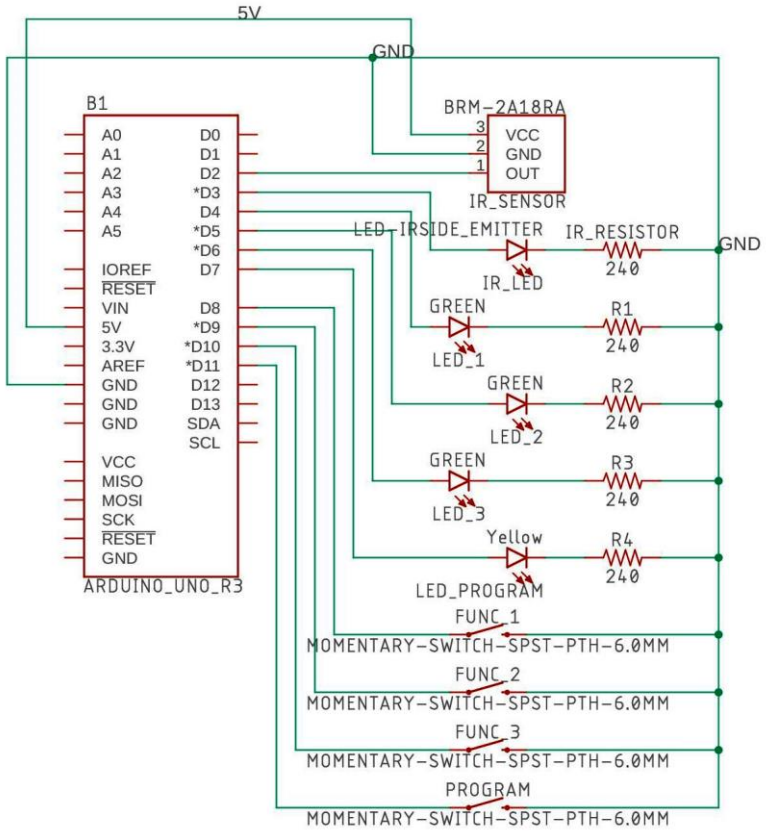
Zaprojektowanie i wykonanie urządzenia z wykorzystaniem Arduino pozwalającego sterować jednym pilotem IR dwoma urządzeniami. Po wykryciu sygnału IR, urządzenie sprawdza czy jest on zapisany w pamięci jako sygnał wyzwalający. Jeśli tak, to urządzenie wysyła przypisany do danego sygnału inny sygnał IR. Przykład: sterowanie grzejnikiem za pomocą pilota do TV: przyciski głośności +/- zmieniają temperaturę, przycisk ON/OFF włącza/wyłącza.

1.1 Założenia projektu:

Układ powinine być wyposażony w:

- Nadajnik IR
- Odbiornik IR
- Przyciski sterujące
- Diody LED sygnalizujące stan urządzenia

2 Schemat układu



Rysunek 1 Schemat układu

Układ BRM-2A18RA jest odbiornikiem IR i został podłączony do wejścia cyfrowego 2.

Dioda nadajnika podczerwieni została podłączenia do wyjścia cyfrowego 3 (narzucone przez wykorzystaną bibliotekę).

Do wyjść od 4 do 7 zostały podłączone diody LED informujące o stanie urządzania. Prąd płynący przez każdą jest ograniczony rezystorem.

Do wejść od 8 do 11 zostały podłączone przyciski do sterowania stanem urządzenia.

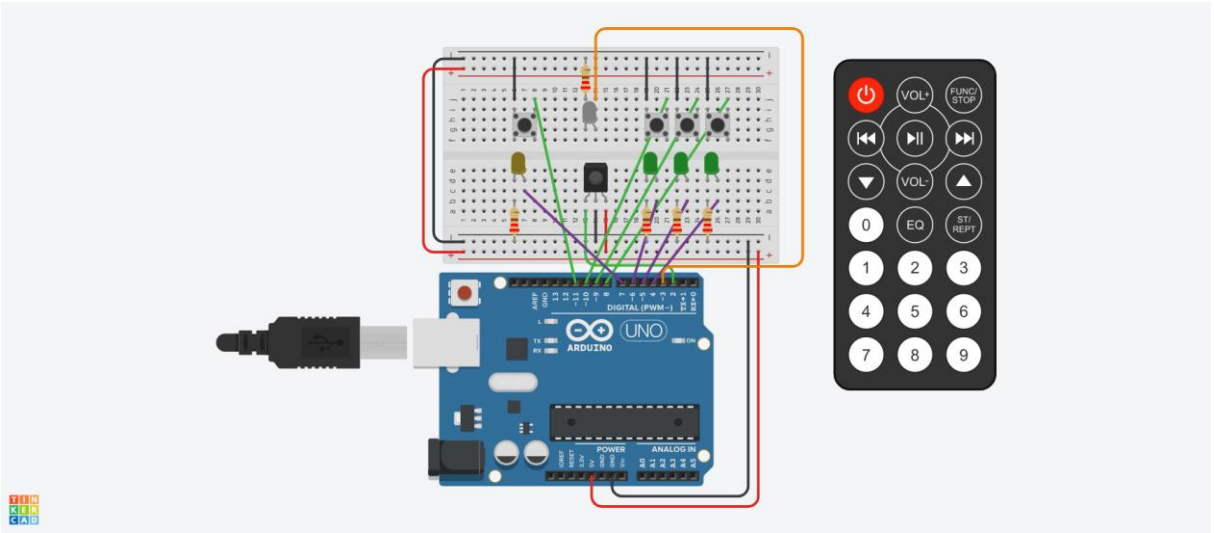
3 BOM

Build of Material:

Oznaczenie	Model	Wartość	Opis	Comment
ARDUINO_UNO_R3	Arduino	UNO	Main board	
IR_SENSOR	BRM-2A18RA		Odbiornik IR	
IR_LED	Dioda podczerwieni		Nadajnik IR	
LED_1, LED_2, LED_3, LED_PROGRAM	Dioda LED	Zielone, żółta	Diody LED	
IR_RESISTOR, R1, R2, R3, R4	Rezystor	240	Rezystor	
FUNC_1, FUNC_2, FUNC_3, PROGRAM	MOMENTARY-SWITCH-SPST-PTH-6.0 MM		Przycisk	Przycisk normalnie otwarty

4 Symulacja Układu

Symulacja została wykonana w środowisku Tinkercad.com. Zasymulowany układ:

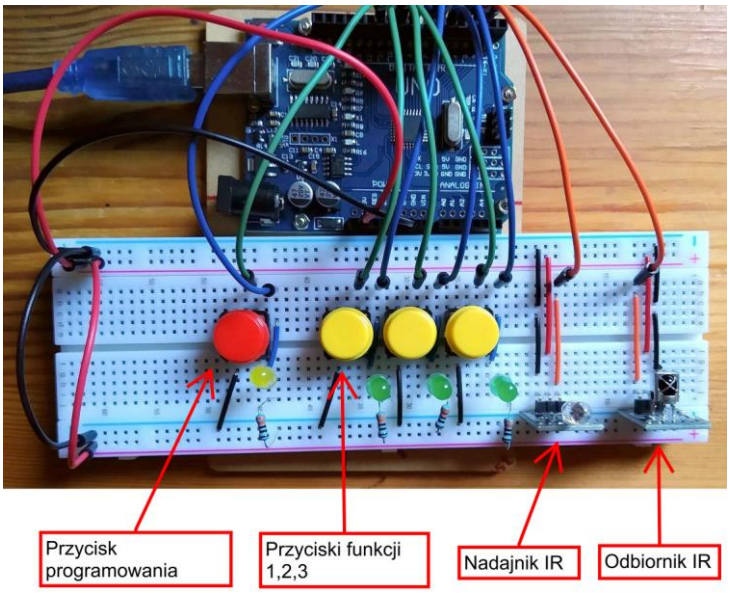


Rysunek 2 Symulacja układu

Podczas symulacji udowodniono poprawność podłączeń zaproponowanych na powyższym schemacie widocznym na Rysunek 1 Schemat układu. Przetestowano funkcję odbioru kodów IR w bibliotece IRemote.

5 REALIZACJA UKŁADU

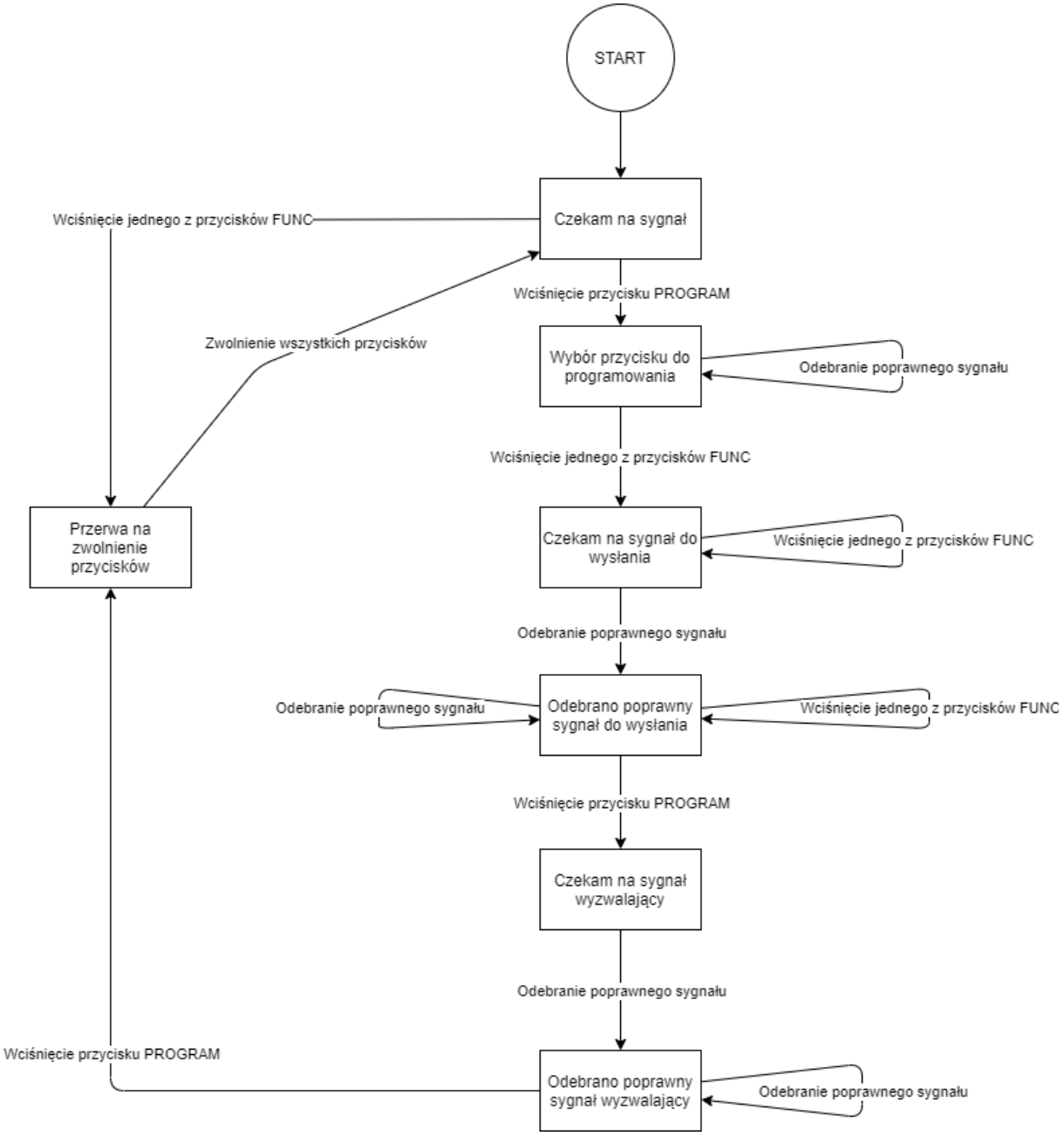
Projekt wykonano na płytce stykowej.



Rysunek 3 Zdjęcie realizacji na płytce stykowej

6 PROGRAMOWANIE

Układ został zaprogramowany za pomocą połączenia szeregowego USB i środowiska Arduino IDE. Program został wykonany jako maszyna stanów, która opisana została następującym schematem.



Rysunek 4 Schemat maszyny stanów

Układ rozpoczyna pracę od oczekiwania na sygnał bądź naciśnięcie przycisku:

- Odebranie sygnału IR zapisanego w pamięci, powoduje wysłanie przyporządkowanego do niego sygnału IR z nadajnika.
- Przyciśnięcie jednego z trzech przycisków funkcji (FUNC_1, FUNC_2, FUNC_3) powoduje wysłanie przyporządkowanego do niego sygnału IR z nadajnika.
- Przyciśnięcie przycisku programowania (PROGRAM) rozpoczyna proces programowania urządzenia.
    - Urządzenie czeka na wybór jednego z przycisków funkcji poprzez jego wciśnięcie. W tym kroku możliwa jest wielokrotna zmiana

- Jednocześnie, urządzenie odbiera sygnały IR do zapisania jako sygnał wyjściowy. Zapisany zostanie ostatnio odebrany sygnał.
- Po naciśnięciu przycisku programującego urządzenie przechodzi do odbierania sygnału wejściowego (wyzwalającego).
- Zapisany zostanie ostatnio odebrany sygnał.
- Po naciśnięciu przycisku programującego, urządzenie przypisze wybraną parę sygnałów do wybranego przycisku i będzie oczekiwać na zwolnienie wszystkich przycisków.
- Po zwolnieniu wszystkich przycisków, urządzenie wraca do stanu początkowego.

7 PODSUMOWANIE

7.1 LESSONS LEARNED:

Podczas realizacji projektu napotkałem następujące problemy:

- odbiornik odbierał sygnały niepochodzące od pilotów (światło słoneczne? pobliskie urządzenia? wadliwy układ odbiornika?),

- sygnał nie zawsze był dekodowany poprawnie (być może niepoprawne użycie biblioteki). W aktualnej wersji konieczne jest upewnienie się, czy układ odczytał właściwy kod.

7.2 ZAKOŃCZENIE

Projekt udało się zrealizować. Układ można dodatkowo rozwinąć poprzez zamianę interfejsu użytkownika na menu z wyświetlaczem.

8 DODATKI

8.1 SPIS RYSUNKÓW:

Rysunek 1 Schemat układu 3

Rysunek 2 Symulacja układu 4

Rysunek 3 Zdjęcie realizacji na płytce stykowej 5

Rysunek 4 Schemat maszyny stanów 6

8.2 KOD PROGRAMU:

```
#include <IRremote.h>

#define BUTTON_1_PIN 8

#define BUTTON_2_PIN 9

#define BUTTON_3_PIN 10
```

```
#define BUTTON_PRO_PIN 11
#define RECEIVER_PIN 12
#define BLINK_LED_PIN 13
#define LED_PROGRAM_PIN 7
#define LED_1_PIN 4
#define LED_2_PIN 5
#define LED_3_PIN 6
#define LED_ALL 1

#define Waiting_for_button 0
#define Choose_button_to_program 1
#define Receiving_code 2
#define Receiving_trigger_code 5
#define Cooldown_after_programming 3

#define LED_STATE_OFF 0
#define LED_STATE_ON 1
#define LED_STATE_BLINKING 2

#define LED_BLINKING_PERIOD_MSEC 1000

#define READ_OK 0

void setState(char newState);
void setup();
void loop();
void setLedState(char ledNumber);
void sendIRCode(int code);
void updateLeds();
bool isBlinkTime();

short state = -1;
bool button_1 = false;
bool button_2 = false;
bool button_3 = false;
```

```
bool button_program = false;
bool button_program_active = true;
short programming_button = -1;
bool repeat_last_code = false;
short led_state[4] = {0, 0, 0, 0};
short led_pins[4] = {LED_PROGRAM_PIN, LED_1_PIN, LED_2_PIN, LED_3_PIN};

struct storedIRDataStruct {
  IRData receivedIRData;
} readBuffer, receivedIRData[6]; // First 3 are output codes, last 3 are trigger codes

void storeOutputCode(IRData *aIRReceivedData, int memoryIndex) {
  storeCode(aIRReceivedData, receivedIRData + memoryIndex);
}

void storeTriggerCode(IRData *aIRReceivedData, int memoryIndex) {
  storeCode(aIRReceivedData, receivedIRData + memoryIndex + 3);
}

short storeCode(IRData *aIRReceivedData, struct storedIRDataStruct *result) {
  if (aIRReceivedData->flags & IRDATA_FLAGS_IS_REPEAT) {
    Serial.println(F("Ignore repeat"));
    return 1;
  }
  if (aIRReceivedData->flags & IRDATA_FLAGS_IS_AUTO_REPEAT) {
    Serial.println(F("Ignore autorepeat"));
    return 2;
  }
  if (aIRReceivedData->flags & IRDATA_FLAGS_PARITY_FAILED) {
    Serial.println(F("Ignore parity error"));
    return 3;
  }
  if (!(aIRReceivedData->protocol == NEC || aIRReceivedData->protocol == SONY)) {
    Serial.println(F("Ignore bad protocol"));
    return 4;
```

```
  }
  /*
    Copy decoded data
  */
  result->receivedIRData = *aIRReceivedData;

  IrReceiver.printIRResultShort(&Serial);
  result->receivedIRData.flags = 0; // clear flags -esp. repeat- for later sending
  Serial.println();
  return 0;
}

void setLedState(char ledNumber, char stateToSet) {
  switch (ledNumber) {
    case LED_PROGRAM_PIN:
      led_state[0] = stateToSet;
      break;
    case LED_1_PIN:
      led_state[1] = stateToSet;
      led_state[2] = LED_STATE_OFF;
      led_state[3] = LED_STATE_OFF;
      break;
    case LED_2_PIN:
      led_state[1] = LED_STATE_OFF;
      led_state[2] = stateToSet;
      led_state[3] = LED_STATE_OFF;
      break;
    case LED_3_PIN:
      led_state[1] = LED_STATE_OFF;
      led_state[2] = LED_STATE_OFF;
      led_state[3] = stateToSet;
      break;
    case LED_ALL:
      led_state[0] = stateToSet;
      led_state[1] = stateToSet;
```

```
    led_state[2] = stateToSet;
    led_state[3] = stateToSet;
    break;
  }
}

void setState(char new_state) {
  switch (new_state)
  {
    case Receiving_code:
      Serial.println("Receiving code to send.");
      break;
    case Receiving_trigger_code:
      Serial.println("Receiving trigger code.");
      break;
    case Choose_button_to_program:
      Serial.println("Choose button to program.");
      programming_button = -1;
      setLedState(LED_ALL, LED_STATE_OFF);
      setLedState(LED_PROGRAM_PIN, LED_STATE_BLINKING);
      break;
    case Cooldown_after_programming:
      break;
    case Waiting_for_button:
      Serial.println("Waiting for input.");
      setLedState(LED_ALL, LED_STATE_OFF);
      break;
  }
  state = new_state;
}

void updateLeds() {
  for (int i = 0; i < 4; i++) {
    if (LED_STATE_OFF == led_state[i]) {
      digitalWrite(led_pins[i], LOW);
```

```
    } else if (LED_STATE_ON == led_state[i]) {
      digitalWrite(led_pins[i], HIGH);
    } else if (LED_STATE_BLINKING == led_state[i]) {
      if (isBlinkTime()) {
        digitalWrite(led_pins[i], HIGH);
      } else {
        digitalWrite(led_pins[i], LOW);
      }
    }
  }
}

bool isBlinkTime() {
  unsigned int timeWindow = millis() % LED_BLINKING_PERIOD_MSEC;
  return timeWindow > LED_BLINKING_PERIOD_MSEC / 2;
}

bool decodeTypeIsValid() {
  return true;
}

void setup()
{
  Serial.begin(9600);
  pinMode(BUTTON_1_PIN, INPUT_PULLUP);
  pinMode(BUTTON_2_PIN, INPUT_PULLUP);
  pinMode(BUTTON_3_PIN, INPUT_PULLUP);
  pinMode(BUTTON_PRO_PIN, INPUT_PULLUP);
  pinMode(LED_PROGRAM_PIN, OUTPUT);
  pinMode(LED_1_PIN, OUTPUT);
  pinMode(LED_2_PIN, OUTPUT);
  pinMode(LED_3_PIN, OUTPUT);
  IrReceiver.begin(RECEIVER_PIN, DISABLE_LED_FEEDBACK);
  IrSender.begin(false);
  setState(Waiting_for_button);
```

```
}

void sendCode(IRData irData) {
  IrReceiver.stop();
  if (irData.protocol == NEC) {
    IrSender.sendNEC(irData.address,
                     irData.command, 1);
  }
  IrReceiver.start();
}

void loop()
{
  updateLeds();
  button_1 = digitalRead(BUTTON_1_PIN) == LOW;
  button_2 = digitalRead(BUTTON_2_PIN) == LOW;
  button_3 = digitalRead(BUTTON_3_PIN) == LOW;
  button_program = digitalRead(BUTTON_PRO_PIN) == LOW;
  if (!button_program) {
    button_program_active = true;
  }
  switch (state)
  {
    case Waiting_for_button:
      if (IrReceiver.available()) {
        if (storeCode(IrReceiver.read(), &readBuffer) == READ_OK) {
          for (int i = 0; i < 3; i++) {
            if (readBuffer.receivedIRData.protocol == receivedIRData[i + 3].receivedIRData.protocol &&
                readBuffer.receivedIRData.address == receivedIRData[i + 3].receivedIRData.address &&
                readBuffer.receivedIRData.command == receivedIRData[i + 3].receivedIRData.command) {
              Serial.print("Got code: ");
              Serial.println(readBuffer.receivedIRData.decodedRawData, HEX);
              Serial.print("Sending code: ");
              Serial.println(receivedIRData[i].receivedIRData.decodedRawData, HEX);
              delay(100);
```

```
        sendCode(receivedIRData[i].receivedIRData);
        break;
       }
     }
     delay(50);
   }
   IrReceiver.resume();
  }
  if (button_1) {
   sendCode(receivedIRData[0].receivedIRData);
   setLedState(LED_1_PIN, LED_STATE_ON);
   setState(Cooldown_after_programming);
  } else if (button_2) {
   sendCode(receivedIRData[1].receivedIRData);
   setLedState(LED_2_PIN, LED_STATE_ON);
   setState(Cooldown_after_programming);
  } else if (button_3) {
   sendCode(receivedIRData[2].receivedIRData);
   setLedState(LED_3_PIN, LED_STATE_ON);
   setState(Cooldown_after_programming);
  } else if (button_program && button_program_active) {
   button_program_active = false;
   setState(Choose_button_to_program);
  } else {
   setLedState(LED_ALL, LED_STATE_OFF);
  }
  break;
 case Choose_button_to_program:
  if (button_1 && programming_button != 0) {
   Serial.println("Programming button 1");
   programming_button = 0;
   setLedState(LED_1_PIN, LED_STATE_ON);
   setState(Receiving_code);
  } else if (button_2 && programming_button != 1) {
   Serial.println("Programming button 2");
```

```
    programming_button = 1;
    setLedState(LED_2_PIN, LED_STATE_ON);
    setState(Receiving_code);
   } else if (button_3 && programming_button != 2) {
    Serial.println("Programming button 3");
    programming_button = 2;
    setLedState(LED_3_PIN, LED_STATE_ON);
    setState(Receiving_code);
   }
   break;
  case Receiving_code:
   if (IrReceiver.available()) {
    storeOutputCode(IrReceiver.read(), programming_button);
    IrReceiver.resume();
   }
   if (button_1 && programming_button != 0) {
    Serial.println("Programming button 1");
    programming_button = 0;
    setLedState(LED_1_PIN, LED_STATE_ON);
   } else if (button_2 && programming_button != 1) {
    Serial.println("Programming button 2");
    programming_button = 1;
    setLedState(LED_2_PIN, LED_STATE_ON);
   } else if (button_3 && programming_button != 2) {
    Serial.println("Programming button 3");
    programming_button = 2;
    setLedState(LED_3_PIN, LED_STATE_ON);
   } else if (button_program && button_program_active) {
    button_program_active = false;
    Serial.print("Setting button ");
    Serial.print(programming_button);
    Serial.print(" output code to ");
    Serial.println(receivedIRData[programming_button].receivedIRData.decodedRawData, HEX);
    setState(Receiving_trigger_code);
    setLedState(led_pins[programming_button + 1], LED_STATE_BLINKING);
```

```
    }

    break;
  case Receiving_trigger_code:
    if (IrReceiver.available()) {
      storeTriggerCode(IrReceiver.read(), programming_button);
      IrReceiver.resume();
    }
    if (button_program && button_program_active) {
      button_program_active = false;
      Serial.print("Setting button ");
      Serial.print(programming_button);
      Serial.print(" output code to ");
      Serial.println(receivedIRData[programming_button + 3].receivedIRData.decodedRawData, HEX);
      setState(Cooldown_after_programming);
    }

    break;
  case Cooldown_after_programming:
    if (!(button_1 || button_2 || button_3 || button_program)) {
      setState(Waiting_for_button);
    }
  }
  delay(15);
}
```