Távvezérlés TV távirányító jelének a feldolgozásával.

SAPIENTIA
ERDÉLYI MAGYAR
TUDOMÁNYEGYETEM
Marosvásárhelyi Kar

Szerzők:

- **Szép Szilárd**
- **Varga Rajmond**

Évfolyam:

- **Automatizálás III-B**

Bevezető ……

Célkitűzések ……..

A rendszer működése, felhasználói követelmények ……..

Activity diagramm ……..

Use-Case diagramm ……..

Rendszer követelmények ……..

A) Funkcionális ……..

B) Nem funkcionális ……..

A rendszer felépítése és kivitelezése ……..

A) Architektúra ……..

B) A rendszer működése ……..

C) A rendszerkomponensek működése ……..

1) IR szenzor ……..

2) Bemeneti arduino ……..

3) Küldő ESP ……..

4) Wi-Fi router ……..

5) Server ……..

6) Fogadó ESP ……..

Összegzés és konklúzió ……..

Bibliográfia ……..

Bevezető:

Világunkban otthonainkat egyre inkább a kényelem jellemzi. Az irány jelenleg egyértelműen az, hogy minél nagyobb kontrollunk egyen otthonunk, berendezéseink felett, anélkül, hogy akár egy lépést is tennünk kellene.

Rengeteg megoldás született már erre a célra, a legtöbb valamilyen telefonos alkalmazáson keresztül éri el eszközeinket, esetleg valamilyen virtuális segéd segítségével, hangvezérléssel.

Ezen rendszerek legnagyobb problémája jelenleg, hogy sok esetben egy zárt ökoszisztémában működnek, mint például az Apple smart home rendszere, és egy ilyen rendszer kiépítése komoly befektetéssel jár. A nyíltabb ökoszisztémájú rendszerek sem olcsóak, és ott is könnyedén felléphetnek kompatibilitási problémák.

Célkitűzések:

Mi ezen rendszerekkel ellentétben egy rendkívül olcsó, képlékeny és egyszerű rendszert szerettünk volna létrehozni, mely könnyedén módosítható úgy, hogy minden felhasználó specifikus igényeit elégítse ki. A rendszer rendkívül alap eszközökből épül fel, és még egy telefon sem szükséges a működtetéséhez.

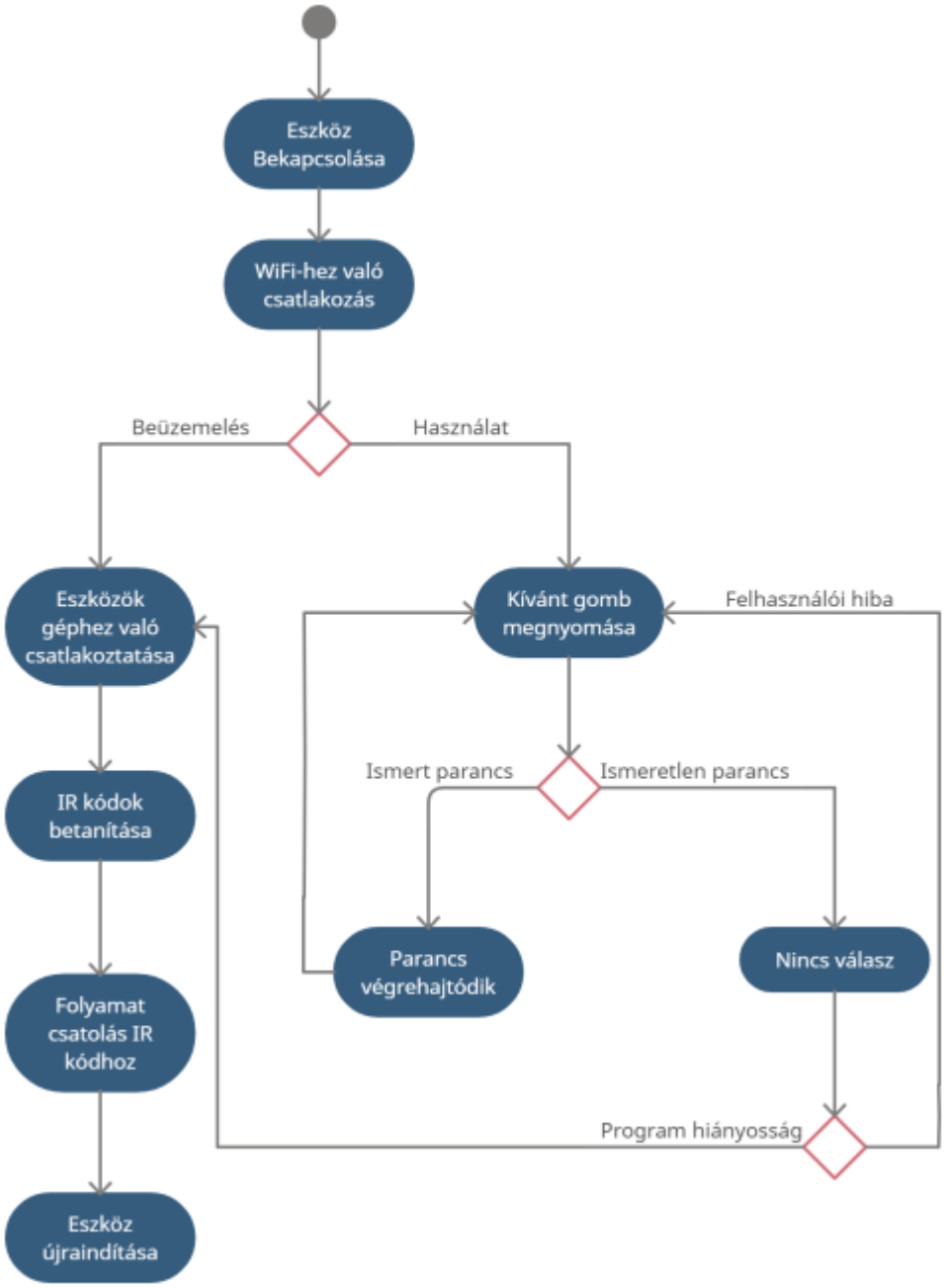
Az elképzelés az, hogy fogjuk azt a vezérlő eszközt, ami minden háztartásban megtalálható, és annak a jelét feldolgozva kapcsolókat vezérlünk esetlegesen potenciométereket, de akár motrokat is. A projekt célja egy vezérlő váz, amely könnyedén módosítható abba az irányba, amelybe a felhasználó kívánja.

A vezérlő jelet adó eszköz egy TV távirányító lenne. Minden háztartásban megtalálható, a jele könnyen feldolgozható, és mindig a felhasználó keze ügyében van.

A rendszer működése, felhasználói követelmények:

Az alap rendszer működése rendkívül egyszerű lenne. A beüzemeléskor, leolvassuk a távirányító által kiadott jeleket, azon gombok lenyomásakor, melyet a felhasználó szeretne, ezeket bevisszük a rendszerbe.

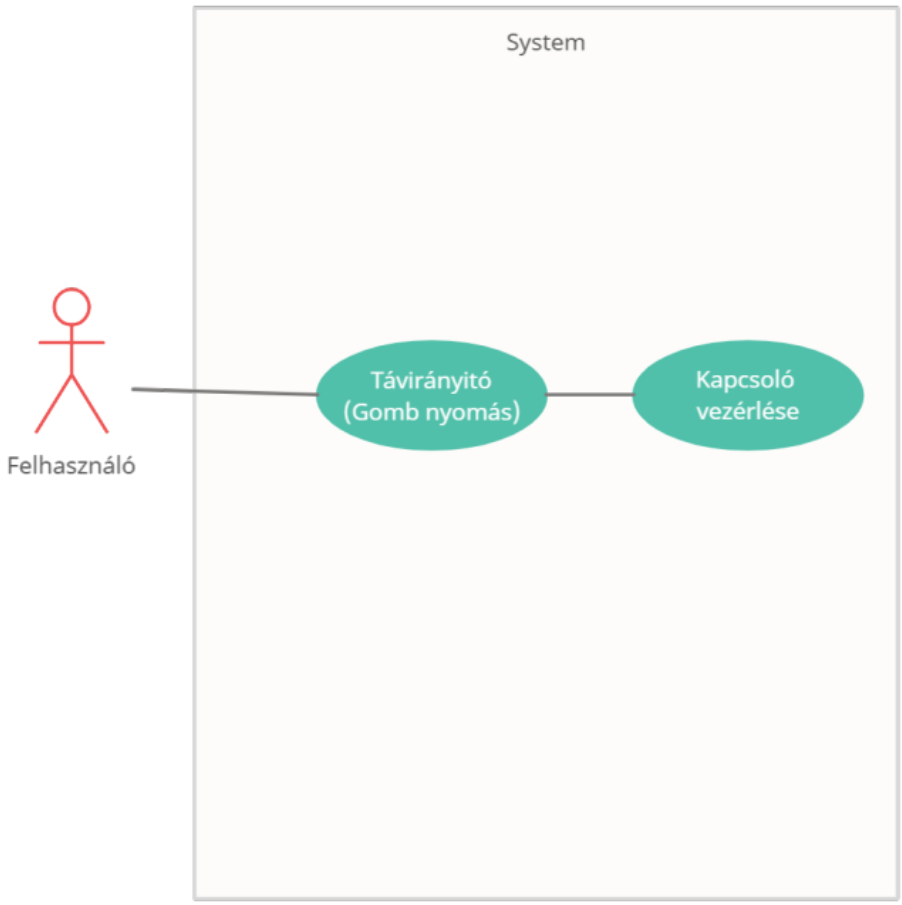
Activity diagramm:



Ezt követően helyet kapnának a vezérlendő kapcsolók (Az alap rendszerben csak kapcsolók lennének, de ez könnyedén kibővíthető lenne vezérelhető potenciométerekkel, esetleg motorvezérléssel).

A beszerelés után a rendszer irányítása annyiból állna, hogy a felhasználó megnyomja a távirányítón a kívánt gombot, a lakásban, a WI-Fi jelszóró hatáskörén belül pedig kapcsolna a kívánt kapcsoló, így irányíthatná akár a lakásban a megvilágítást, de akár eszközök áramellátását is.

Use-Case diagramm:



Rendszer követelmények:

A) Funkcionális:

- Adott gomb megnyomásával adott kapcsoló ki/be kapcsolása.

B) Nem funkcionális:

- Távirányító
- Legalább 2 ESP8266
- Legalább 2 Arduino Uno
- TSOP 1238 vagy hasonló IR dekóder
- Minden ESP-Arudino pároshoz kell egy tápegység (Például 9V elem) és egy 3,3 V kimeneti feszültségű feszültségstabilizátor (Például LF33CV).
- Wi-fi Router
- Amennyiben több mint 2 ESP8266-ra van szükség, kell egy számítógép, amint a server futhat.

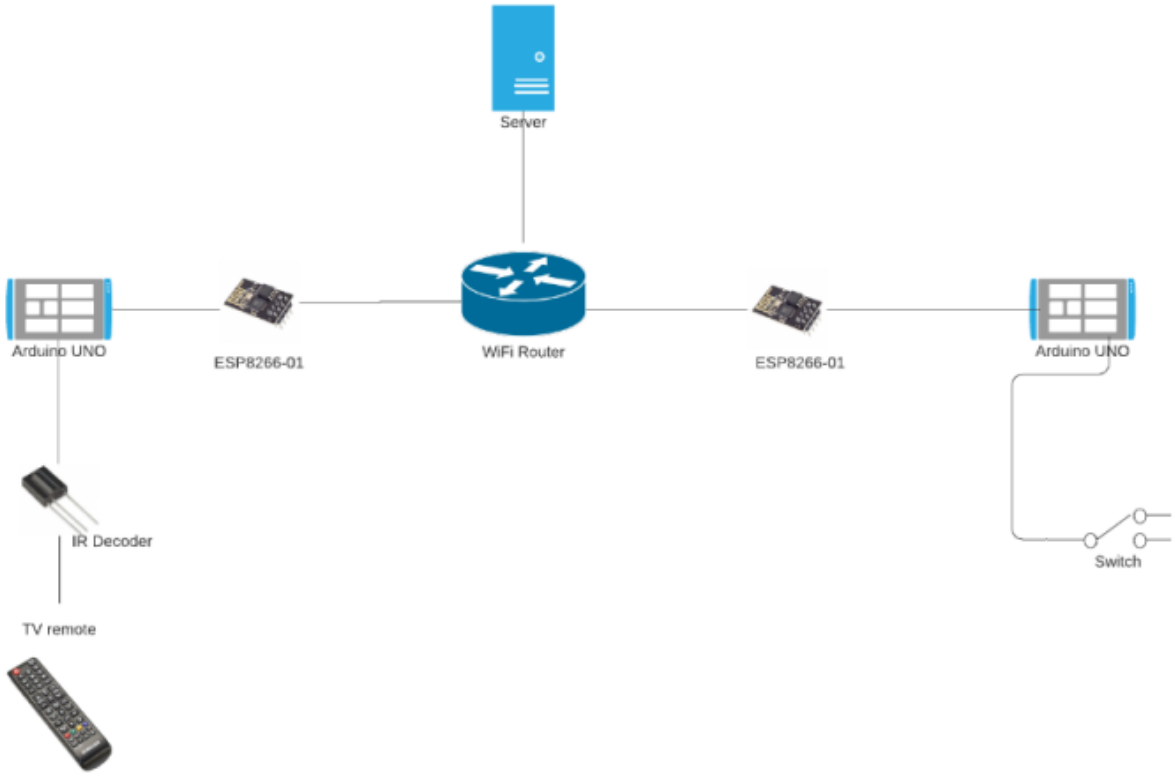
A tervezéshez és beüzemeléshez, felprogramozáshoz szükséges programok:

- Visual Studio
- Arduino IDE

A rendszer felépítése és kivitelezése:

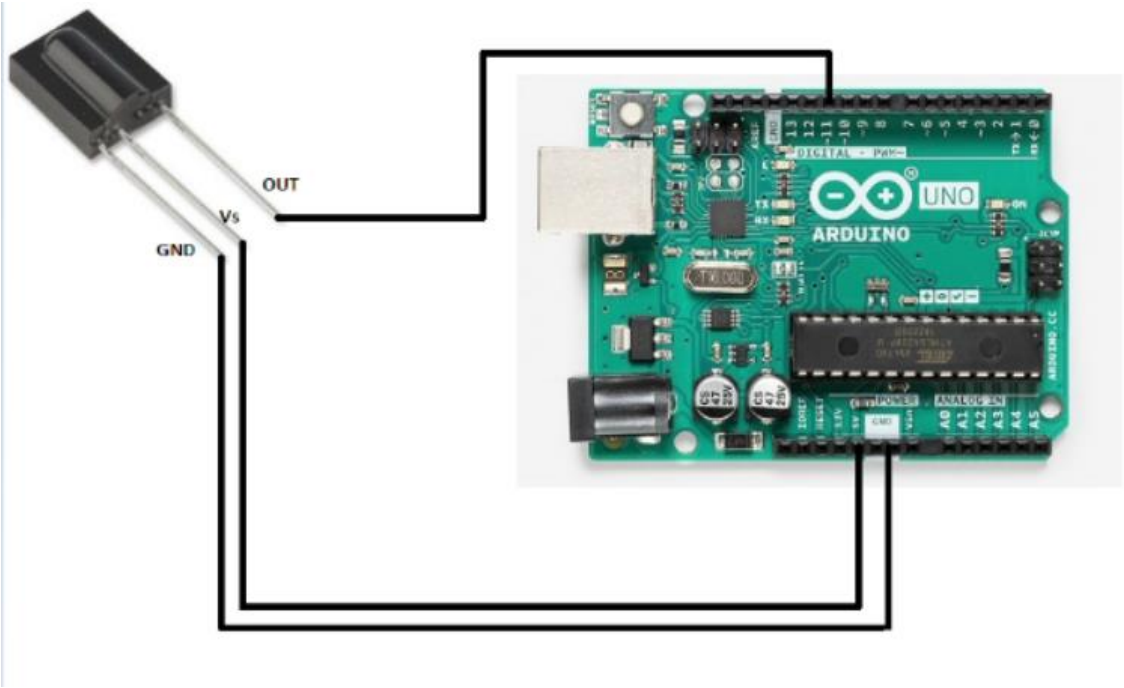
A) Architektúra:

A rendszer felépítés majdhogynem szimmetrikus, “középen” a Wi-Fi routerrel és a szerverrel, amennyiben szerver indolkolt.

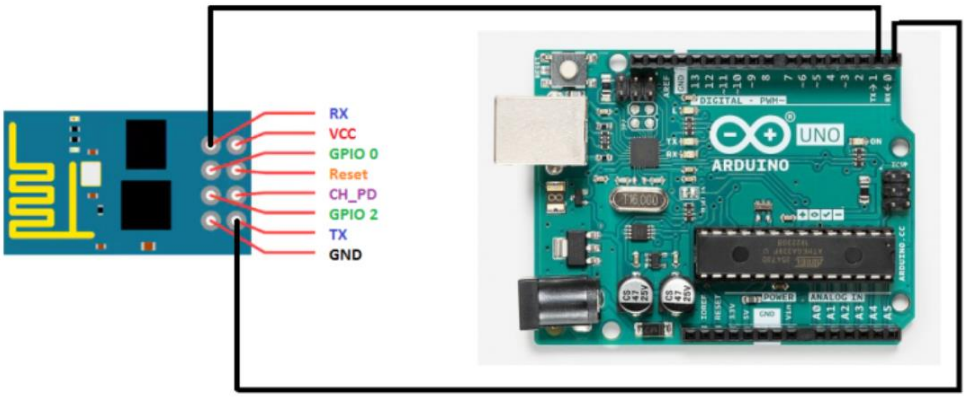


A rendszer két végpontján két arduino, egy bemeneti és egy kimeneti arduino foglal helyet, és ők bonyolítják a kimenetet a “külvilággal”.

A bemeneten egy TSOP 1238 IR dekóder van az arduinora kötve. A dekóder a tápját az arduino 5V-os ki menetéből nyeri, a digitális kódot az arduino 11-es digitális bemenetén küldi be.



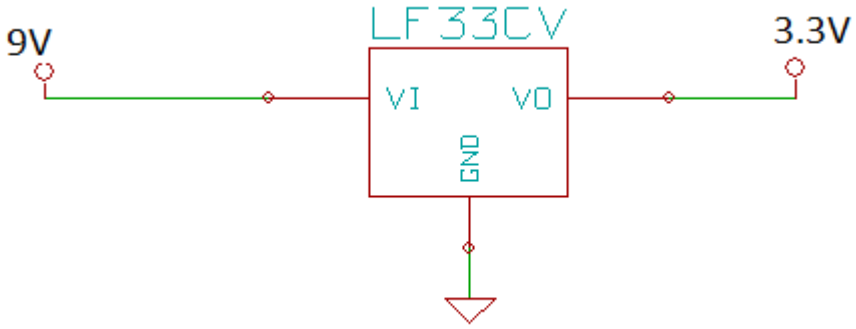
Az arduinohoz még egy ESP 8266 van kötve, az RX és TX soros kommunikációs ki/bemeneti portokon keresztül.



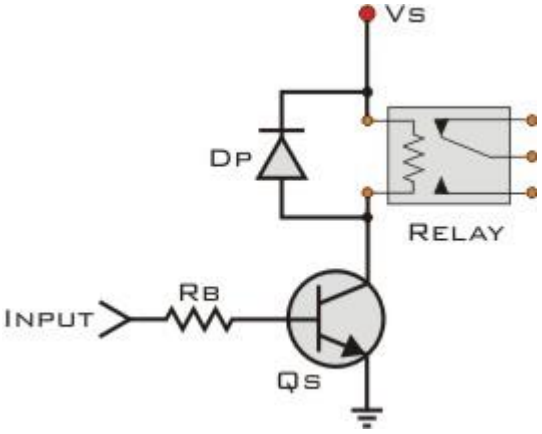
A kimeneten, az arduinora kötve a legegyszerűbb esetben egy vagy több relévezérlő áramkör van, a relé szolgál kapcsolóként.

Ehhez az arduinohoz is ugyanúgy egy ESP 8266 van kötve, mint a bemeneti arduinohoz.

Az ESP táplálásához az LF33CV feszültségstabilizátort a következő módon kell bekötni:

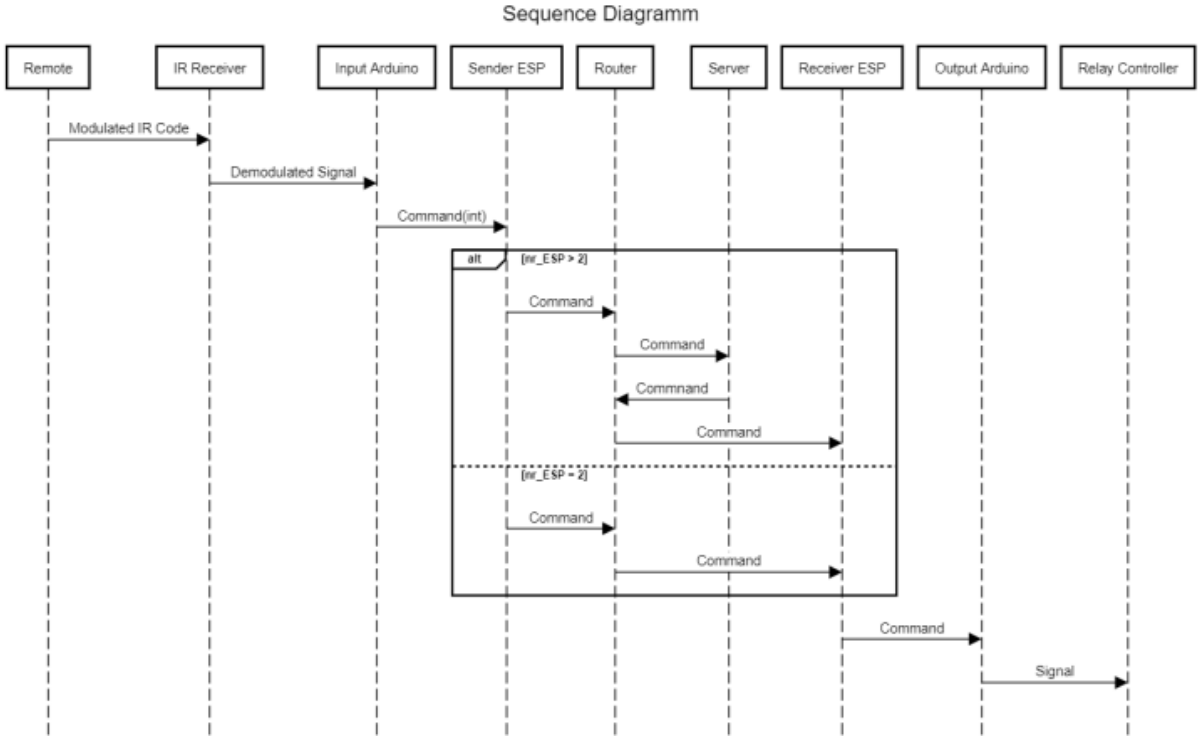


A relé vezérlő a következő módon valósítható meg:



B) A rendszer működése:

A rendszer működését a következő diagram írja le:



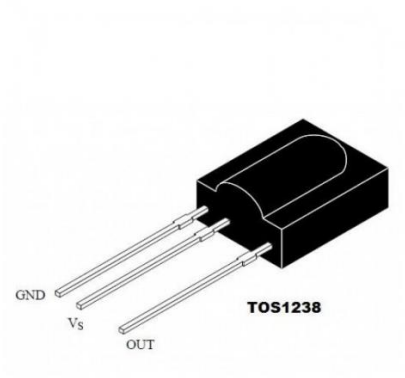
Először a távirányírtó kibocsát egy 38kHz-en modulált infra jelet. Az IR szenzor érzékeli és demodulálja ezt, majd továbbítja az arduino 11-es digitális bemenetére. Az arduino feldolgozza azt és a rx/tx soror kommunikációs portjain keresztül elküldi az ESP-nek. Az ESP két féle képpen lehet programozva. Amennyiben rajta kívül csak 1 ESP van a rendszerben, tehát csak a ház egy adott pontján szeretnénk irányítani valamit, akkor elküldi a jelet a fogadó ESP-nek a wi-fi routeren keresztül, amelyben nyitottunk egy portot előzetesen. Amennyiben több fogadó ESP is helyet kap a rendszerben, abban az esetben a küldő ESP egy servernek küldi el a jelet, amely tárolja az összes fogadó ESP IP címét, és mindenkinek elküldi az adatot a routeren keresztül. A fogadó ESP miután fogadja a jelet ugyancsak az RX/TX soros kommunikációs portjain keresztül küldi a jelet a kimeneti arduinonak, aki értelmezi a kódot, és megvezérli a megfelelő kimenetét, ami a relé vezérlő áramkörön keresztül kapcsolja a relét.

A wi-fi kommunikációhoz UDP protokollt választottunk, mert ugyan ez a protokol nem megbízható, de az biztos, hogy a csomag egy darabban fog megérkezni, ha megérkezik. Jelen esetben nem volt szükségünk nagyfokú megbízhatóságra, hiszen a folyamatok amelyeket irányítunk vele elég jól láthatóak ha nem mennek végbe, és ekkor egyszerűen újra meg kell nyomni a gombot. Nem jár különösebb problémával, ha például valami hiba folytán a villany nem kapcsolódna fel első megnyomásra.

Azonban a probléma esélye csekély, hiszen az üzenetek nem igényelnek nagy adatforgalmat, mivel kis csomagokat küldünk viszonylag ritkán, és különösen biztonságos, ha egy saját routert használunk a rendszerhez, amihez nem férnek hozzá más felhasználók.

C) A rendszerkomponensek működése:

1) IR szenzor:



A szenzor fogadja a modulált jelet a távirányítóból, demodulálja és elküldi az ardunio 11-es digitális bemenetére.

A táplálása az arduino 5V-os kimenetéről történik, ez elegendő a meghajtásához, csak 1,5 mA-áramot vesz fel maximálisan.

A fenti szenzon 40 kHz el működik, számunkra 38 kHz lenne az ideális, de megfelelően működik egy hasonló szenzorral is.

Absolute Maximum Ratings

T_{amb} = 25 °C, unless otherwise specified

Parameter	Test condition	Symbol	Value	Unit
Supply Voltage	(Pin 2)	V_S	- 0.3 to + 6.0	V
Supply Current	(Pin 2)	I_S	5	mA
Output Voltage	(Pin 3)	V_O	- 0.3 to + 6.0	V
Output Current	(Pin 3)	I_O	5	mA
Junction Temperature		T_j	100	°C
Storage Temperature Range		T_{stg}	- 25 to + 85	°C
Operating Temperature Range		T_{amb}	- 25 to + 85	°C
Power Consumption	(T_{amb} ≤ 85 °C)	P_{tot}	50	mW
Soldering Temperature	t ≤ 10 s, > 1 mm from case	T_{sd}	260	°C

2) Bemeneti arduino:

Egy arduino uno tökéletesen megfelel céljainknak, mivel nem kell sok számítást végezni az elvégzendő feladatok során, de kellő mozgásteret biztosít, ha a kimenetek kapcsolgatása helyett a későbbiekben valami komolyabb vezérlési feladatot szeretnénk beiktatni a rendszerbe.

Az arduinot 7-12 V közötti feszültségről ajánlott táplálni, a mi választásunk a tesztjeink során egy 9V-os elemre esett, mivel az uno evergiaigénye nem túl nagy.

A ki/benetei több mint elegendőek:

I/O:

- 6 analóg input.
- 14 digitális input/output és ebből 6 darab PWM

A mikrokontroller egy: ATmega328P.

A felprogramozásához az Arduino IDE programot használtuk.

Program:

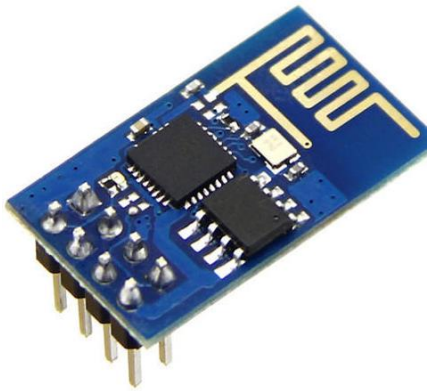
Az IR szenzor által adott jelet az IRremote.h nevű könyvtárat használtuk. Ez tökéletesen megfelelt a céljainknak a benne levő függvények képesek beolvasni a soros digitális jelet a bemenetről és int típussá alakítják.

```
1. #include <IRremote.h>
2. int RECV_PIN = 11;
3. // Beállítjuk, hogy melyik bemenetről érkezik a digitális kód.
4. IRrecv irrecv(RECV_PIN);
5. // Létrehozunk egy IRecive obijektumot és átadjuk neki a bemenő pint.
6. decode_results results;
7. // Ebben az obijektumban tárolódik el a kód, és kerül a value operandusába a kódhoz társított
   int érték.
8.
9. void setup()
10. {
11.   Serial.begin(9600);
```

```
12.   irrecv.enableIRIn();
13.  // Elindítjuk a soros portot az ESP felé és elindítjuk  Rcivert.


14.  }
15.
16.  void loop()
17.  {
18.    if (irrecv.decode(&results))
19.      {
20.   // Értelmezzük a kódot, ha érkezett a bemeneten.
21.       if(results.value==573503)
22.         {
23.           Serial.print("1");
24.         }
25.       if(results.value==532703)
26.         {
27.           Serial.print("2");
28.         }
29.       if(results.value==565343)
30.         {
31.           Serial.print("3");
32.         }
33.  // Elküldjük a parancsot az ESP-nek.
34.       irrecv.resume();
35.  // Olvassuk a bemenetről a következő kódot.
36.     }
37.  }
```

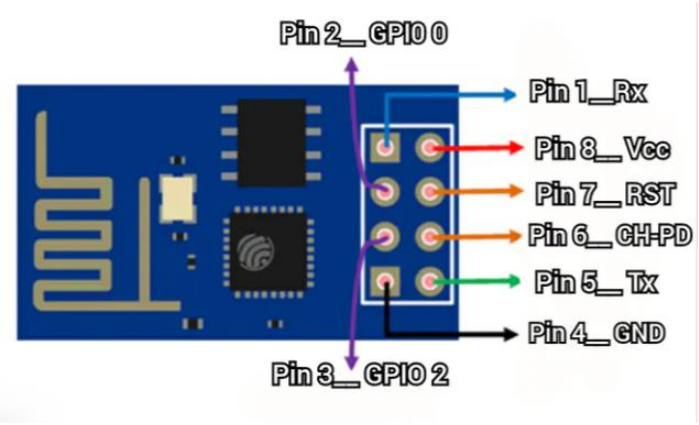
3) Küldő ESP:

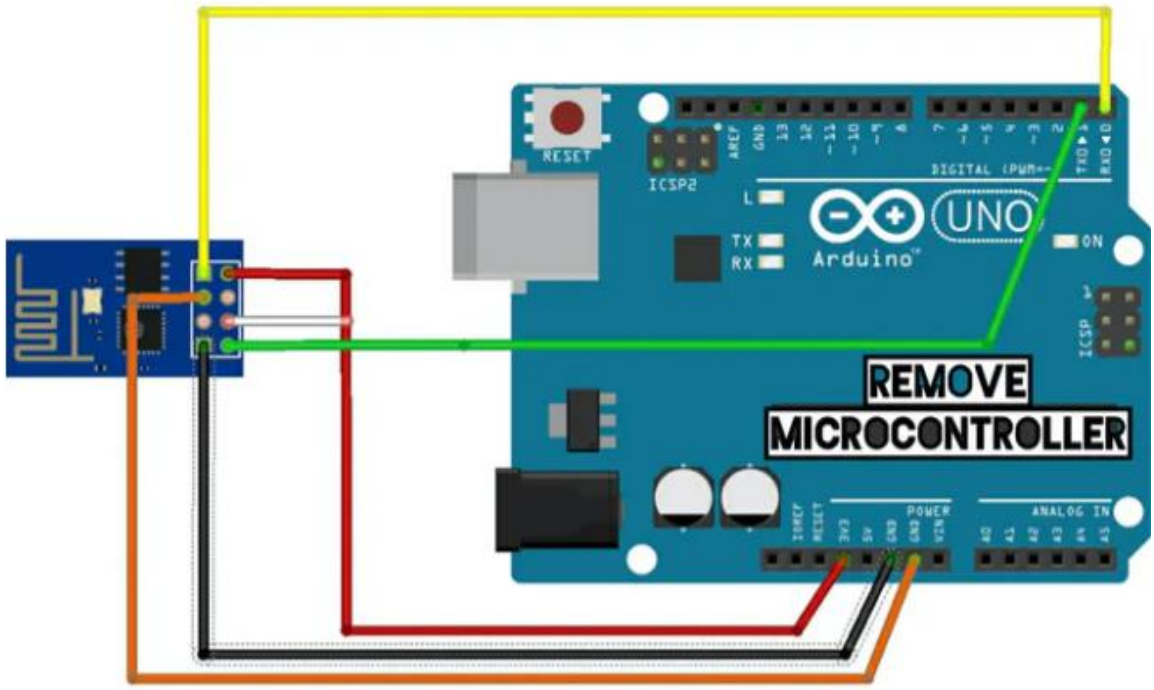


Ez a wifi modul megfelelt a céljainknak, könnyedén kommunikál az aurduinoval az RX/TX soros kommunikációs portjain keresztül, valamint alkalmas minimális számítások elvégzésére és wi-fin való kommunikációra is.

A táplálása nem történhet az Arduino 3,3V-os feszültség kijövetelén keresztül, mivel a wi-fi kommunikációhoz való energia igénye túl nagy, ezért ugyanarról forrásról érdemes táplálni, de fontos, hogy kell beiktatni egy 3,3V-os feszültség stabilizátort, mert ez a modul nagyon érzékeny a túlfeszültségre.

A felprogramozása igencsak nehézkes. Az általunk talált módszer a programozásához az volt, hogy veszünk egy arduino uno-t, eltávolítjuk belőle a mikrokontrollert, és a következő módon kötjük össze őket:





Fontos az, hogy az ESP kettes lába csak addig kell a földre kötve legyen, amig programozzuk, miután megvan, az ESP 8-as lábát ki majd vissza kell kötni a tápjára.

Amennyiben ezalatt az ESP nem az arduinotól kapja az áramellátást, akkor a földjeiket van, hogy össze kell kötni, de a tapasztalataink alapján nem mindig. Valamint van, hogy csak elég sokadik próbálkozásra kezdi el feltölteni a kódot, volt, hogy segített az, ha a folyamat alatt, ami alatt a számítógép csatlakozni próbált az ESP-hez, kihúztuk majd visszadugtuk az ESP tápját, így újraindítva azt. Sajnos ez a megoldás sem használ mindig, sokat kell próbálkozni.

Az alábbi kóddal valami probléma van, sajnos nem tudtunk rájönni, hogy miért nem érkezik meg az adat, amit el szeretnénk küldeni. Annyi biztos, hogy a bemeneti arduino által adott kódot megkapja, és értelmezi is, ha az 1 es kódot kapja meg le/fel kapcsolja a rajta levő LED-et.

A kód most a szervernek kellene hogy küldjön.

```
1.  #include <ESP8266WiFi.h>
2.  #include <WiFiUdp.h>
3.  #include <SoftwareSerial.h>
4.   // A fentebbi könyvtárakat használtuk a komunikációhoz.
5.  SoftwareSerial ESPserial(1, 5);
6.   //RX/TX soros portok helyének megadása.
7.  const char* ssid = "Tenda_2731F0";
8.  const char* password = "12345678";
9.   //Wi-fi adatok.
10. WiFiUDP Udp;
11. //Létrehozunk egy Udp obijektumot,
12. unsigned int localUdpPort = 10999;
13. int messagePacket;
14. //Port megadása és küldő buffer létrehozása.
15.
16. void setup() {
17.
18.   Serial.begin(9600);
19.   ESPserial.begin(115200);
20. //Előkészülés a soros komunikációra.
21.   pinMode(LED_BUILTIN,OUTPUT);
22. //A beépített ledet használjuk, hogy jelezzünk vele a külvilág fele.
23.   WiFi.begin(ssid, password);
24. //Rácsatlakozunk wifire;
25.   while (WiFi.status() != WL_CONNECTED)
26.   {
27.     delay(500);
28.    digitalWrite(LED_BUILTIN, !digitalRead(LED_BUILTIN));
29.   }
30. //Várunk a csatlakozásra.
31.   Serial.println(" connected");
32.
33.   Udp.begin(localUdpPort);
34.   //Inicializálja az UDP libraryt.
35. }
36. void loop() {
37.   if(Serial.available()){
38.   messagePacket = Serial.read();
39. // Ha a soros porton elérhető az információ, akkor beolvassuk.
40.   if(messagePacket=='1')
41.   {
42.     digitalWrite(LED_BUILTIN, !digitalRead(LED_BUILTIN));
43.    }
44. //Ha az 1-es gomb volt lenyomja változik a beépített led állapota (Működés ellenőrzés).
45.   Udp.beginPacket(Udp.remoteIP(), Udp.remotePort());
46.   Udp.write(messagePacket);
```

```
47.   Udp.endPacket();
48. //Elküldjük a csomagot.      Lehet hogy itt a probléma ami miatt nem megy.
49.   }
50. }
```

4) Wi-Fi router:

Bármilyen router megteszi, portot kell nyitni rajta.

5) Server

A servert c++ nyelven írtuk, Visual Studio 2019-ben. A szerver UDP protokollt használ.

A terv az volt, hogy indításkor a server minden fogadó ESP től fogad egy jelet, így ismerve meg az IP címüket. Természetesen ehhez a szerverbe előre bele kellett írni a fogadó ESP-k számát.

Ezt követően a server a küldő ESP től mindig kapta volna az adatot, majd küldte volna tovább az összes fogadó ESP-nek.

```
1.  #include <iostream>
2.  #include <stdio.h>
3.  #include "winsock2.h"
4.  #include "ws2tcpip.h"
5.  #pragma comment(lib, "Ws2_32.lib")
6.   //Szükséges könyvtárak a socketek használatához windows környezetben.
7.
8.  int main()
9.  {
10. WSADATA wsaData;
11. SOCKET ListenSocket;
12. sockaddr_in ServAddr;
13. sockaddr_in RecvAddrCommander;
14. int Port = 10999;
15. char SendBuf[4];
16. char FAR RecBuf[4];
17. int BufLen = 4;
18. int fromlen, CommandLen=-1, NumberOfClients=1;
19. SYSTEMTIME st, lt;
20.
21.
22. WSAStartup(MAKEWORD(2, 2), &wsaData);
23.       //Winsocket inicializálása
24. ListenSocket = socket(AF_INET, SOCK_DGRAM, IPPROTO_UDP);
25.           // Halgató soxket létrehozása, protokol megadása.
26. ServAddr.sin_family = AF_INET;
27. ServAddr.sin_port = htons(Port);
28. //ServAddr.sin_addr.s_addr = htonl(INADDR_ANY);
29. inet_pton(AF_INET, "127.0.0.1", &ServAddr.sin_addr);
```

```
30.         //Server címének a lefoglalása
31. if (bind(ListenSocket,
32. (SOCKADDR*)&ServAddr,
33. sizeof(ServAddr)) == SOCKET_ERROR) {
34. printf("bind() failed.\n");
35. closesocket(ListenSocket);
36. return 0;
37. }
38.         //Ezzel állandósítjuk a socketünket, hogy ne változhasson.
39. //------------------------
40.
41. sockaddr_in RecvAddr[1024];
42. for (int i = 0; i < NumberOfClients; ++i)
43. {
44. while (CommandLen < 0) {
45. fromlen = sizeof(RecvAddr[i]);
46. CommandLen = recvfrom(ListenSocket, RecBuf, BufLen, 0, (SOCKADDR*)&(RecvAddr[i]),
    &fromlen);
47. printf("%i %c", CommandLen, (char)(*RecBuf));
48. }
49. }
50.         // Itt várnánk a fogadó kliensekre, hogy megismerjük őket.
51. printf("Receiving datagrams...\n");
52.
53. int code;
54.
55. while(9)
56. {
57. while (CommandLen < 0)
58. {
59. CommandLen = recvfrom(ListenSocket, RecBuf, BufLen, 0,
    (SOCKADDR*)&RecvAddrCommander, &fromlen);
60. }
61. memcpy(&code, RecBuf, 4);
62. printf("%i %i\n", code, CommandLen);
63.
64. for (int i = 0; i < NumberOfClients; ++i)
65. {
66. CommandLen=sendto(ListenSocket, SendBuf, CommandLen, 0, (SOCKADDR*)&(RecvAddr[i]),
    sizeof(RecvAddr[i]));
67. printf("%i %i\n", code, CommandLen);
68. }
69. CommandLen = -1;
70. }
71.         //olvassuk a kódot, majd küldjük ki minden fogadónak
72.
73. //-------------------------------------------
74. // Clean up and quit.
```

```
75. closesocket(ListenSocket);
76. printf("Exiting.\n");
77. WSACleanup();
78. return 0;
79. }
```

Írtunk egy UDP teszt klienst, akivel kipróbáltuk hogy a server és a fogadó ESP is működik, legalábbis fogadnak adatot, de egyik ESP-ről sem sikerült adatot kiküldenünk.

6) Fogadó ESP:

A komponens feladata, hogy fogadja az adatot a servertől vagy a küldő ESP-től, majd azt a soros portjain keresztük küldje el a vezélő arduinonak.

A kód nagyrésze hasonlít a küldő ESP-éhez.

```
1.  char incomingPacket[4]
2.
3.
4.  void setup() {
5.
6.    Serial.begin(9600);
7.    ESPserial.begin(115200);
8.    //Serial.println();
9.    pinMode(LED_BUILTIN, OUTPUT);
10.   //Serial.printf("Connecting to %s ", ssid);
11.   WiFi.begin(ssid, password);
12.   while (WiFi.status() != WL_CONNECTED)
13.   {
14.     delay(500);
15.     digitalWrite(LED_BUILTIN, !digitalRead(LED_BUILTIN));
16.     Serial.print(".");
17.   }
18. // Villog a leg, amég várunk a csatlakozásra.
19.   digitalWrite(LED_BUILTIN,HIGH);
20.   Serial.println(" connected");
21.   digitalWrite(LED_BUILTIN,LOW);
22.   Udp.begin(localUdpPort);
23.   Serial.printf("Now listening at IP %s, UDP port %d\n", WiFi.localIP().toString().c_str(),
      localUdpPort);
24.
25.   }
26.
27. void loop() {
28.
29.   int packetSize = Udp.parsePacket();
30.
```

```
31.
32.   if (packetSize)
33.   {
34.     Serial.printf("Received %d bytes from %s, port %d\n", packetSize,
      Udp.remoteIP().toString().c_str(), Udp.remotePort());
35.  //Ha számítógéphez csatakoztatjuk ellenőrizhetjük hogy megy e, az erduino ezt egyszerűen
      //nem érti majd meg, nem okoz problémát
36.     int len = Udp.read(incomingPacket, 4);
37.     if (len > 0)
38.     {
39.       incomingPacket[len] = 0;
40.       delay(500);
41.       digitalWrite(LED_BUILTIN, !digitalRead(LED_BUILTIN));
42.       Serial.printf("UDP packet contents: %s\n", incomingPacket);
43.         if(Serial.available()){
44.     Serial.write(incomingPacket);
45.   }
46.  //Olvassuk az adatot, olvasáskor változik a led állapota, majd küldjük az adatot az arduinonak
47.     }
48.   }
49. }
```

7) Fogadó Arduino:

Az egység egyetlen feladata az, hogy a fogadó ESP-től fogadja a kódot, majd a kód függvényében a megfelelő digitális kimenetet megfordítja.

```
1.  // 2, 4 ,7 Digital Output Pins
2.  const int Out_1 = 2;
3.  const int Out_2 = 4;
4.  const int Out_3 = 7;
5.  const int ledPin = 13;
6.  int command;
7.
8.  void setup() {
9.    Serial.begin(9600);
10. }
11.
12. void loop() {
13.
14.   command = Serial.read();
15.   switch(command){
16.     case '1': digitalWrite(Out_1, !digitalRead(Out_1));
17.           digitalWrite(ledPin, !digitalRead(ledPin));
18.           break;
19.     case '2': digitalWrite(Out_2, !digitalRead(Out_2));
20.           break;
21.     case '3': digitalWrite(Out_3, !digitalRead(Out_3));
22.           break;
```

```
23.   }
24.
25.  }
```

Összegzés és konklúzió:

A projektet sajnos nem tudtuk sikerrel zárni, és bár sokat haladtunk előre, de végül egy adott problémán nem sikerült túljutni.

Ehhez nagymértékben hozzájárult, hogy rosszul osztottuk be az időnket, nem számoltunk azzal, hogy mennyi komplikáció lép fel majd közben és mennyi idő elfog telni az egyszerű problémák megoldásával is. Ilyen probléma volt az, hogy valahogyan sikerült elveszítenünk a meglevő IR szenzort, és mivel az ünnepek miatt minden zárva volt, fél napot vett igénybe, mire egy rossz TV-ből sikerült előhalászni egy hasonlót. Hasonló probléma volt, még az, hogy a megvásárolt 3 ESP8266-ból, csak 2 volt működőképes, és egyikük valamiért gyakran ejtette el a Wi-Fi jelet.

Ha néhány nappal több időt hagyunk rá, akkor valószínűleg meg tudtuk volna oldani, de végül időszűkében csak eddig jutottunk vele, sehogy sem sikerült rájönni, hogy mit hibáztunk, hogy az ESP-ről sehogy sem tudtunk adatot küldeni, csak fogadni.

A korona vírus ellehetetlenítette, hogy a projekt bármely részét is személyesen, együtt csináljuk meg, ez egy kis nehézséget okozott, de rengeteg software a segítségünkre volt ez ügyben. Elsősorban kommunikálásra Discordot használtunk, mivel mindketten ismerjük, relatív könnyű kezelni, használható fájlmegosztásra, videó chatre, képernyő megosztására is. Egy másik legtöbbet használt platform a GitHub volt. Ide inkább már elkészített, lefutó program fájlokat töltöttünk csak fel. Kicsit nehezünkre esett megtanulni, hogy hogyan is használjuk minél hatékonyabban. Sokat segített abban, hogy lássuk, hogy haladunk, mi van meg, mi van még hátra, lépéspontokat létrehozva.

Az Arduino Uno-inkat és az ESP8266-kat Arduino IDE segítségével programoztuk fel, itt kompiláltuk a kódokat. Talán ez volt a projektünk legfőbb lényege, hogy tanuljunk az Arduinokról, dolgozzunk velük, ismerjük meg hogyan is működnek, mire használhatóak akár a mindennapi életben. Az ESP-k összeköttetését szerveren keresztül oldottuk meg, ez a szerver Visual Studioban lett megírva C++ nyelvben. Hálózatok tantárgyból tanultunk szerver létrehozást, ezért maradtunk ennél a software-nél, ismerős volt és az alapok is megvoltak már.

Végül is, mindezeket figyelembe véve, bár a projektet nem sikerült működésbe hozni, így egyáltalán nem lehet sikernek nevezi, de nem könyveltük el teljes kudarcként, hiszen rengeteget tanultunk az időbeosztás fontosságáról, valamint az Arduino-k és ESP-k programozásáról, a wifi hálózaton való kommunikációról. Ezenkívül megismerkedtünk a Githubbal is, valamint néhány diagram készítő programmal.

Bibliográfia:

Arduino UNO I/O:

- **https://components101.com/microcontrollers/arduino-uno**

ESP8266 bekötése:

- **http://www.teomaragakis.com/hardware/electronics/how-to-connect-an-esp8266-to-an-arduino-uno/**
- **https://create.arduino.cc/projecthub/Niv_the_anonymous/esp8266-beginner-tutorial-project-6414c8**
- **https://arduino-esp8266.readthedocs.io/en/latest/index.html**
- **https://create.arduino.cc/projecthub/PatelDarshil/how-to-communicate-with-esp8266-via-arduino-uno-f6e92f**

UDP kommunikáció:

- **https://arduino-esp8266.readthedocs.io/en/latest/esp8266wifi/udp-examples.html**

IR Decoder:

- **https://www.youtube.com/watch?v=e8DfjMVShec**