Budapesti Műszaki és Gazdaságtudományi Egyetem
Villamosmérnöki és Informatikai Kar
Méréstechnika és Információs Rendszerek Tanszék

Beágyazott rendszer vezérlésének implementálása USB porton IoT eszközre alapozva

Készítette
Pető Ágoston

Konzulens
Krébesz Tamás István

2019

TARTALOMJEGYZÉK

HALLGATÓI NYILATKOZAT

Alulírott Pető Ágoston, szigorló hallgató kijelentem, hogy ezt a szakdolgozatot meg nem engedett segítség nélkül, saját magam készítettem, csak a megadott forrásokat (szakirodalom, eszközök stb.) használtam fel. Minden olyan részt, melyet szó szerint, vagy azonos értelemben, de átfogalmazva más forrásból átvettem, egyértelműen, a forrás megadásával megjelöltem.

Hozzájárulok, hogy a jelen munkám alapadatait (szerző(k), cím, angol és magyar nyelvű tartalmi kivonat, készítés éve, konzulens(ek) neve) a BME VIK nyilvánosan hozzáférhető elektronikus formában, a munka teljes szövegét pedig az egyetem belső hálózatán keresztül (vagy hitelesített felhasználók számára) közzétegye. Kijelentem, hogy a benyújtott munka és annak elektronikus verziója megegyezik. Dékáni engedéllyel titkosított diplomatervek esetén a dolgozat szövege csak 3 év eltelte után válik hozzáférhetővé.

Kelt: Budapest, 2019. 12. 22.

.....

Pető Ágoston

Összefoglaló

A TRF6900 EVM FSK adó-vevő kártyát az egyetemen laboratóriumi mérések során használjuk, azonban a fejlesztőkártya eredeti vezérlőszoftvere, csak a régi, a Microsoft által ma már nem támogatott operációs rendszereken (például: Windows 95, 98 és XP) működik. A vezérlőszoftver és a TRF6900 EVM közötti kommunikáció párhuzamos porton keresztül történik, ami a mai számítógépekben már nem található meg és a legújabb operációs rendszerek nem teszik lehetővé a párhuzamos port regisztereihez való hozzáférést.

Szakdolgozatom célkitűzése, hogy a TRF6900 EVM fejlesztőkártya a továbbiakban használható legyen a mai modern operációs rendszerekkel is. Ennek elérése érdekében tanulmányoztam a kártya felépítését, protokollját és a vezérléshez szükséges kódszavak képzését. Ezek ismeretében LabVIEW környezetben egy olyan vezérlőszoftvert készítettem el, ami a Texas Instruments eredeti alkalmazásának helyébe léphet. Megvalósítottam a rendszerszintű paraméterek megadását támogató grafikus felhasználói felületet, a vezérlő kódszavak generálását és a TRF6900 EVM kommunikációs protokollját. A számítógép USB alapú soros portja és a TRF6900 EVM kártya párhuzamos portja között szükség volt egy fizikai interfészre, amihez egy NodeMCU mikrokontrollert használtam. A soros portos vezérlés mellett megvalósításra került a kártya WiFi-n keresztül történő vezérlése is.

A szakdolgozat elkészítése során oszcilloszkóppal megvizsgáltam a mikrokontroller port regiszteres és szekvenciális kimeneti értékadás közötti időkülönbséget, majd a megvalósítását követően ellenőriztem a rendszer által kiküldött jelalakokat, ami egyezett az adatlapon található ábrával.

A dolgozat végén kitérek a megoldásom továbbfejlesztési lehetőségeire.

Abstract

The TRF6900 EVM FSK transceiver board is used at the University for laboratory measurements, but the original control software for the development board only works on old operating systems that are no longer supported by Microsoft (such as Windows 95, 98 and XP). Communication between the control software and the TRF6900 EVM is via a parallel port, which is no longer found in today's computers and the latest operating systems do not allow access to parallel port registers.

The purpose of my thesis is to make the TRF6900 EVM development board usable with today's modern operating systems. To achieve this, I studied the card's structure, protocol, and the control words generation. With these knowledges, I created a control software in LabVIEW that could replace the original Texas Instruments application. I created a graphical user interface (GUI) that supports the specification of the system parameters. I also implemented the control words generation, and the TRF6900 EVM communication protocol. A physical interface was needed between the computer’s USB port and the parallel port of the TRF6900 EVM card. For this I used a NodeMCU microcontroller. In addition to serial port control, I also implemented the control of the board via Wi-Fi.

During the preparation of the thesis I examined the time difference between the microcontroller port register and sequential output assignments using an oscilloscope, and during the test phase I checked the signals sent by the system, which corresponded to the figure on the datasheet.

At the end of the thesis I discuss the possibilities of further development of my solution.

1. Bevezetés

A TRF6900 fejlesztői kártyát (Evaluation Module – EVM), a Texas Instruments gyártja, ami egy a TRF6900A chipre épülő FSK adó-vevőt, és annak programozásához szükséges kódszavak beléptetését segítő áramkörök alkotják. Az FSK (frekvenciabillentyűzés) moduláció a digitális moduláció egy fajtája, ahol az analóg szinuszos vivőjel frekvenciája hordozza a bináris információt. A TRF6900A rendszerchipnek tekinthető (System-on-a-Chip - SoC), hiszen egyaránt megvalósításra került benne egy FSK adó és vevő áramkör is [1]. Vezérlésére a Texas Instruments kifejlesztett egy szoftvert, ami lehetővé teszi a rendszerparaméterek magasszintű megadását, amiből a program előállítja a chip programozásához szükséges kódszavakat. Ezt a vezérlőprogramot azonban a modern operációs rendszerek (például Windows 10) nem támogatják.

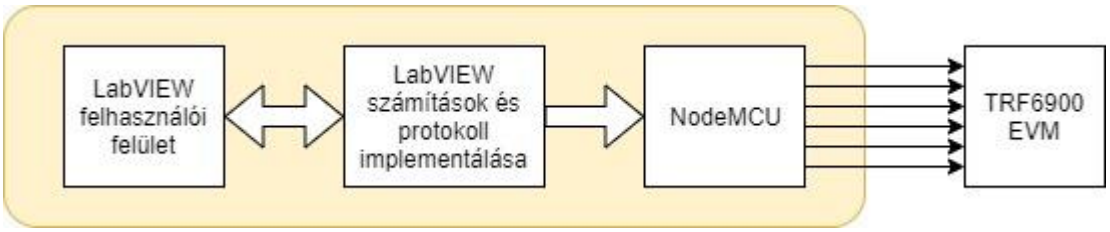
A kódszavakat a vezérlőszoftver párhuzamos porton keresztül küldi ki a TRF6900 EVM fejlesztőkártyának. A mai számítógépekben már nem található párhuzamos port, ráadásul az a legújabb operációs rendszerek nem teszik lehetővé a párhuzamos port regisztereihez való hozzáférést.

Mivel az eredeti szoftver zárt, és specifikációja nem elérhető, ezért jelen dolgozatban a kártya és a protokoll dokumentációját felhasználva megvalósításra kerül egy olyan rendszer, ami funkciójában megfelel a Texas Instruments vezérlőszoftverének. Továbbá mivel a TRF6900 EVM kártya csak párhuzamos porton keresztül vezérelhető, így implementálásra kerül a PC USB és a TRF6900 EVM kártya párhuzamos portja közötti fizikai interfész.

A vezérlőszoftvert LabVIEW-ban valósítottam meg. Az alkalmazás egy grafikus felhasználói felületet nyújt, ahol a rendszerszintű paraméterek beállíthatók, ami alapján a program a kódszavakat legenerálja. A legenerált kódszavak ezután a kártya egyedi protokolljának megfelelően kiküldésre kerülnek.

A LabVIEW-ból kiküldött adatok a fizikai interfészen keresztül jutnak el a fejlesztői kártya párhuzamos portjára. Ezt a fizikai interfészt NodeMCU mikrokontroller segítségével valósítottam meg.

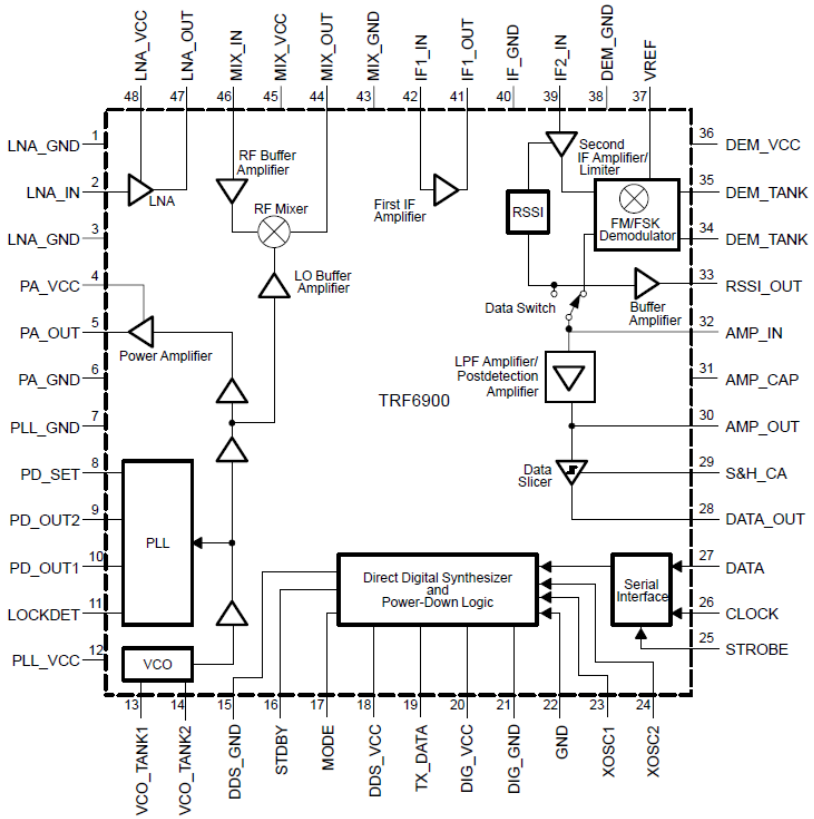
A 1-1. ábra szemlélteti a TRF6900 EVM kártya vezérlését, ami a dolgozatban megvalósításra került.



1-1. ábra: A rendszer megvalósítása

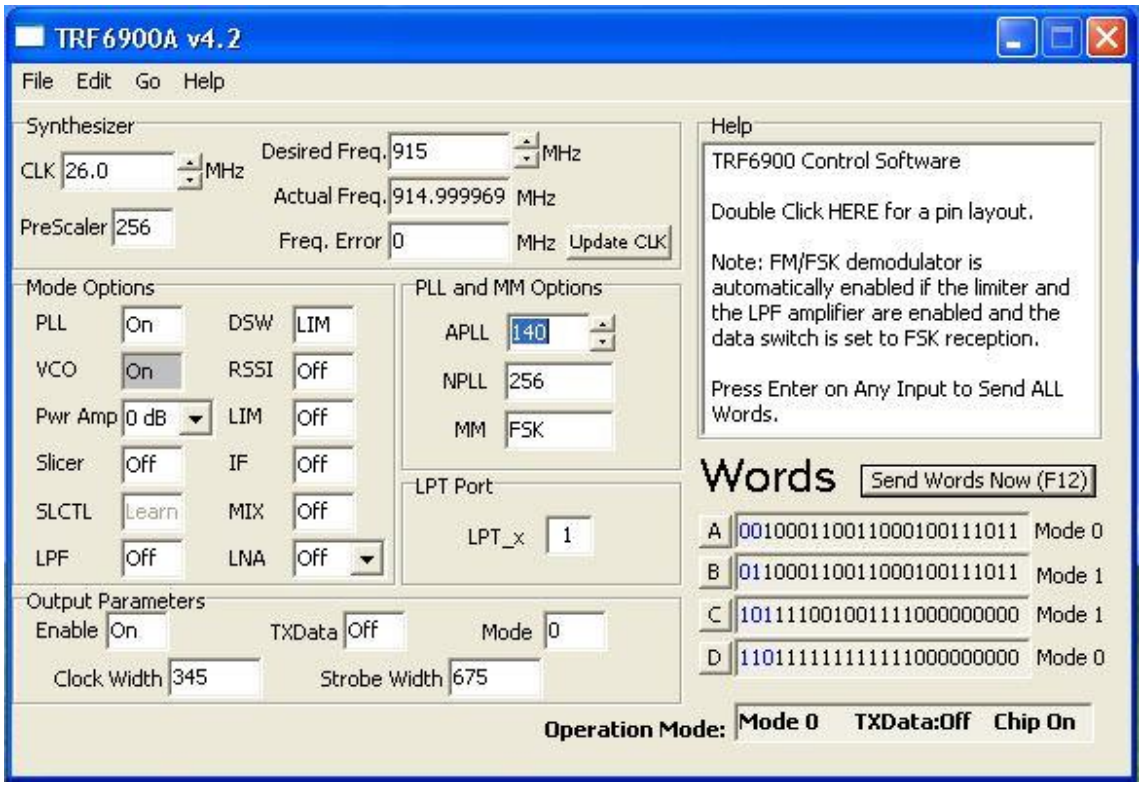
1.1. TRF6900 kártya felépítése és vezérlőszoftvere

A TRF6900A FSK adó-vevő chip különböző funkcionális blokkból épül fel, amit a 1-2. ábra szemléltet.

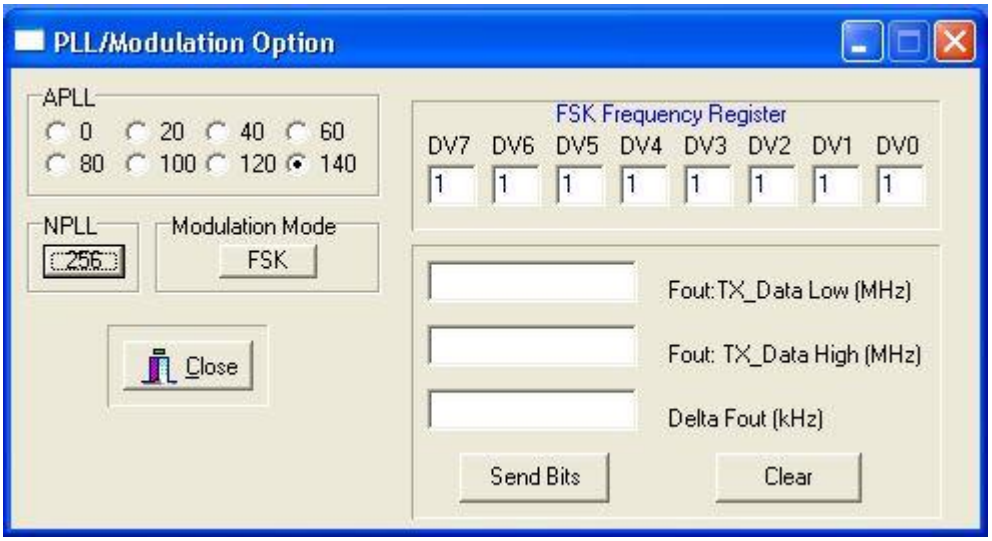


1-2. ábra: A TRF6900A blokkdiagramja

A Texas Instruments által fejlesztett eredeti vezérlőprogram az 1-3. és 1-4. ábrán látható. A kezelőfelületen a TRF6900 EVM különböző funkcionális blokkjainak értékadása, be- és kikapcsolása, illetve az FSK modulációs regiszter (DEV) megadása lehetséges, amit részletesen a 3.1 fejezetben tárgyalok.



1-3. ábra: A Texas Instruments szoftver vezérlő ablaka



1-4. ábra: A Texas Instruments szoftver PLL/FSK modulációs ablaka

1.2. NodeMCU Lua WIFI (ESP8266-CP2102)

Az 1-5. ábrán látható NodeMCU egy nyílt forráskódú firmware és hardver fejlesztési környezet, ami az Espressif által 2014-ben megalkotott ESP8266 SoC köré épült. Az ESP8266 tudásának és rendkívül alacsony árának köszönhetően nagyon népszerű IoT

eszköz. Hardverét tekintve, egy nagyon alacsony fogyasztású 32 bites Tensilica L106-os RISC processzort kapott, ami akár a 160MHz-es maximum órajelet is elérhet. Az eszköz WiFi modullal rendelkezik, ami támogatja a 802.11 b/g/n szabványt és a TCP/IP protokollt. Szintén támogatja többek között az SPI, UART és I2C kommunikációs protokollokat [2].

1-5. ábra: NodeMCU mikrokontroller

A NodeMCU egy a Lua programozási nyelven alapuló firmware, azonban a program fejlesztése során az Arduino Core könyvtárat használtam, amivel Arduino környezetben gyorsan hatásos program írható. A fejlesztéshez a Microsoft által fejlesztett, nyílt forráskódú Visual Studio Code [3] forráskód szerkesztő programot és a Platform IO [4] bővítményt használtam. A Platform IO szintén nyílt forráskódú és több mint 700 különböző beágyazott fejlesztőkártyát támogat.

1.3. LabVIEW

A LabVIEW (Laboratory Virtual Instrument Engineering Workbench) egy grafikus fejlesztő környezet, ami rendkívül sokat használt az egyes mérő és adatgyűjtő alkalmazások terén [5].

A LabVIEW 1986-ban jelent meg és egyre nagyobb népszerűségnek örvend a mérnöki világban, mivel egy gyorsabb, magasszintű fejlesztési módszert nyújt a rendszer elképzelésétől a megvalósításig, és széleskörű I/O és hardver platformmal is integrálható.

A LabVIEW-ban megírt programokat Virtual Instruments-nek (VI) nevezzük. A VI programot egy Front Panel (előlap) és egy Block Diagram alkotja. A program előlapján jelennek meg az egyes vezérlő bemenetek és kijelzők, míg a blokkdiagramon az ezek közötti logikai kapcsolatot teremtjük meg, funkcióblokkokkal és huzalozásokkal.

2. A TRF6900 EVM vezérlése párhuzamos porton keresztül NodeMCU-val

A TRF6900 EVM kártya párhuzamos porton keresztül vezérelhető. A mai számítógépek már nem rendelkeznek ilyen csatlakozóval, így a két eszköz között szükségünk van egy fizikai interfészre. A NodeMCU mikrokontroller az adatokat soros porton vagy WiFin keresztül kapja, és kimeneteit a TRF6900 EVM párhuzamos port csatlakozójának megfelelő lábaira kötve vezérli azt.

2.1. Adatok fogadása

A NodeMCU feladata ebben a rendszerben a TRF6900 EVM kártya 3.fejezetben tárgyalt vezérlőszoftvere által küldött adatok megfelelő kimenetre, vagyis a párhuzamos port megfelelő bemenetére való állítására korlátozódik.

2.1.1. Soros port

A soros kommunikáció felépítéséhez a NodeMCU oldalán az Arduino könyvtár Serial osztályát használtam, ami többféle függvényt biztosít adatok beolvasására. A `Serial.parseInt()` és `Serial.parseFloat()` függvények működése nagyon hasonló egymáshoz. Mindkét függvény nevének megfelelő adattípust vár a bemeneten, és visszatérési értéke az első érvényes adat. A függvény érvénytelen bejövő adat, vagy időtúllépés (timeout) hatására kilép. Az időlimit alapértelmezett értéke 1 másodperc, de ez a `Serial.setTimeout()` függvénnyel megváltoztatható, milliszekundumos léptékkel. A `Serial.readString()` és `Serial.readStringUntil()` függvények sztring típusú bemenet beolvasására használhatóak. Működésük annyiban tér el egymástól, hogy amíg az első az időlimit eléréséig olvassa a bejövő adatokat, addig a második függvény esetén megadhatunk egy speciális karaktert, aminek beolvasására a függvény automatikusan kilép. Teljesen ugyanez a különbség a `Serial.readBytes()` és a `Serial.readBytesUntil()` függvények között. A függvény hívásakor megadunk egy karaktertömböt, amit a függvény pufferként használ, és a beolvasandó adatok hosszát. Az Serial osztály utolsó beolvasó függvénye, amit ismertetni szeretnék, a `Serial.read()` függvény. Ez az ASCII táblázatban szereplő karakterek egyikét olvassa be a bemenetről. Mivel összesen egyetlen karakter megérkezése esetén visszatér, így nincsen várakozási idő. Az ASCII táblázatban

minden karakter 8 biten van leírva. Amint azt a későbbiekben ismertetni fogom, a TRF6900 EVM 6 vonallal vezérelhető, ezért a program megvalósításában a `Serial.read()` függvényt használtam. Itt fontos megjegyezni, hogy a LabVIEW-ban megvalósított az egyes vezérlővonalakon egy órajel ciklus alatt kiküldött 6 vezérlőbitet kiküldés előtt karakter formátumra át kell alakítanunk, hogy a beérkező adat megfeleljen a használt függvény elvárásainak [6].

2.1.2. WiFi modul

A soros kommunikáció hátránya, hogy az eszközök között fizikai kapcsolatot kell létrehozni egy kábel segítségével. Ennél a megoldásnál lényegesen kényelmesebb a kapcsolatot vezeték használata nélkül felépíteni. Az ESP8266 hardver lehetőséget biztosít a vezetéknélküli kommunikációra. A kapcsolat kiépítéséhez szükségünk van egy elérési pontra (access point), ahova a két eszköz kapcsolódik és egymással kommunikálni tud. Ez akár lehet egy harmadik eszköz is (például router), de az ESP8266 is létre tud hozni egy saját hálózatot (Soft Access Point) amire a számítógép Station módban kapcsolódni fog [7]. Ezt illusztrálja a 2-1. ábra.

2-1. ábra: Kapcsolat kiépítése a PC és a mikrokontroller között

A NodeMCU mikrokontroller feladata ebben a rendszerben a LabVIEW által kiküldött adatok fogadása és azok továbbítása. Ebben a modellben a kommunikáció kezdetét mindig a LabVIEW fogja kezdeményezni, ezért szükség van egy szerver létrehozására, ami egy általunk szabadon megválasztott porton fogadja a klienstől érkező kommunikációs kéréseket. A kapcsolódási pontot és a szervert az alábbi kódrészlet hozza létre.

```
const char* ssid = "TRF";
const char* password = "BMEVIKMIT";
WiFiServer server(23);
WiFi.softAP(ssid, password);
server.begin();
```

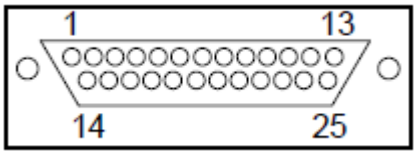
A soft access point alapértelmezett IP címe a 192.168.4.1. Amennyiben ez valamiért nem megfelelő akkor a `Wifi.softAPConfig()` függvény használatával megváltoztathatjuk [8].

A program ezt követően várja a kliens érkezését. Amennyiben a kapcsolat létrejött a szerver és kliens között, az adatokat beolvassuk és meghívjuk a kimenetekre kiíró függvényt az alábbi módon:

```
WiFiClient client;
if (server.hasClient()) {
  client = server.available();
}
while(client.available()) {
  input_data = client.read();
  WriteToPins();
}
```

2.2. Párhuzamos port interfész

A 2-2. ábrán látható párhuzamos port egy párhuzamos kommunikációra alkalmas csatlakozási pont. A szabványos párhuzamos port 8 PC kimeneti adatbittel, 4 kimeneti és 5 bemeneti kontroll bittel rendelkezik. A TRF6900 EVM ezekből csak a 2-7 kimeneti biteket és a 11 és 12 bemeneti biteket használja, azonban a vezérléshez mi csak a kimeneti biteket használjuk [9].



2-2. ábra: Párhuzamos port

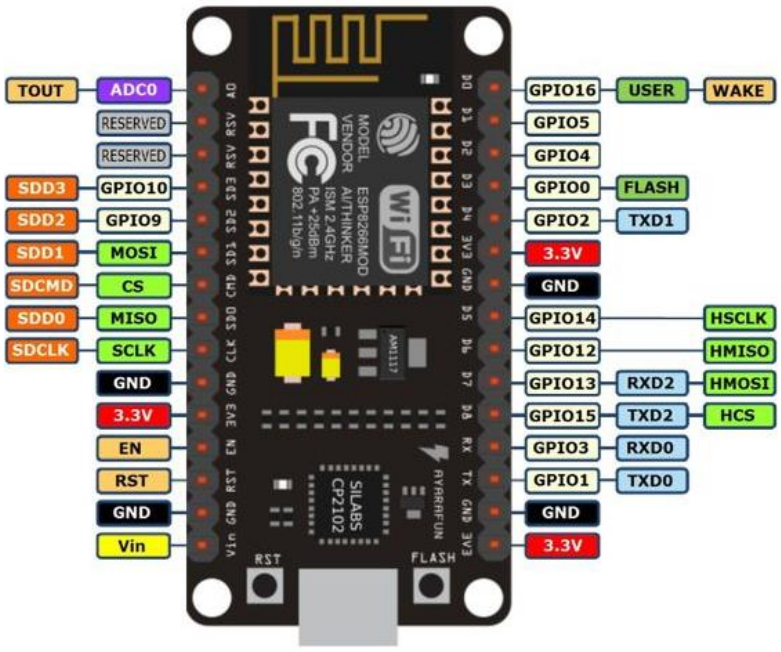
Annak érdekében, hogy a kommunikáció a kártya és a rendszer között helyesen működjön a NodeMCU kimeneteit a TRF6900 EVM megfelelő bemeneteire kell kötnünk. Ennek kiosztása a 2-3. táblázatban látható.

2-3. táblázat: Párhuzamos port bemeneti tábla

Láb	Bemenet
2	Clock
3	Data
4	Strobe
5	TX_Data
6	Enable
7	Mode

2.2.1. GPIO ki és bemenetek

A mikrokontroller kimeneteinek helyes kiválasztása nagyon fontos. Az általános célú be és kimenetek (General Purpose Input Output - GPIO) azok, amiket a program során szabadon felhasználhatunk a külvilággal történő kommunikációra. Ezek lehetnek digitális vagy analóg kimenetek. A NodeMCU ki- és bemeneteit a 2-4. ábra szemlélteti.



2-4. ábra: NodeMCU ki és bemenetei

A következő részben ismertetett port regiszter használata miatt fontos volt, hogy a regiszterben egymás mellett elhelyezkedő biteket válasszak ki, így kevesebb shift és maszkolás kell annak előállításához. Mivel a GPIO1 és GPIO3 portokat a soros kommunikáció során használjuk, ezért a kimeneteket a 2-5. táblázat szerint választottam.

2-5. táblázat: NodeMCU kimeneti tábla

GPIO	Beme-net
4	Clock
5	Data
12	Strobe
13	TX_Data
14	Enable
15	Mode

2.2.2. Port regiszter

Az alacsony szintű port regiszter rendkívül gyors kimenetírást tesz lehetővé. Alkalmazása rendkívül előnyös nagyon időkritikus alkalmazások esetén, ahol akár a mikroszekundumnál is rövidebb idő alatt kell a kimeneti lábak értékét megadni. Szintén hasznos alkalmazás lehet, ha kevés elérhető hely maradt a mikrokontroller memóriájában, hiszen, akár 16 kimenetnek is 1 parancsban be tudom állítani az értékét, a különálló értékadás helyett. Ebből adódóan viszont használata rendkívül megnehezíti a hibakeresést [10].

Én a saját megoldásomban a NodeMCU saját port regiszterét használtam. A `GPO` (General Purpose Output regiszter) egy 16 bites regiszter, ami a kimenetek értékét állítja. Egyes GPIO kimenethez tartozó bit, a GPIO sorszámával megfelelő helyiértékű bit, ezért a helyes kimeneti regiszter elérése érdekében bitmanipulációt kell végezni a beérkező adaton. A korábban beolvasott 6 bites adat két legkisebb helyiértékű bitjét az 5-dik és 6-dik bithelyre shift-elem, a többit pedig kimaszkolom. Ugyanígy járok el a felső 4 bittel és a két 16 bites adatot összeadom, így előállítva a port regiszter helyes értékét.

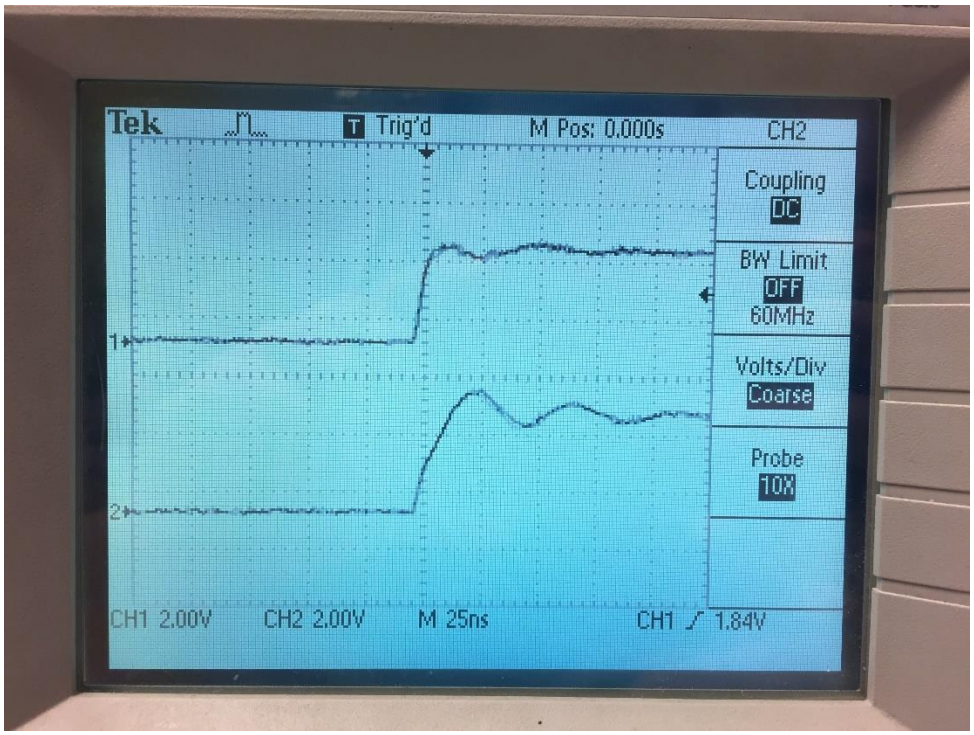
```
mask_first4bits = 0xF000;    //HEX F000 = BIN 1111 0000 0000 0000
mask_last2bits = 0x0030;     //HEX 0030 = BIN 0000 0000 0011 0000
GPO = ((input_data << 10) & mask_first4bits) + ((in-
put_data << 4) & mask_last2bits);
```

A port regiszter használata viszont felvet egy kérdést. Ténylegesen mennyivel hosszabb időt vesz igénybe egy bit kiküldése a hagyományos `digitalWrite()` függvény alkalmazásával, ezért oszcilloszkóppal megmértem a kettő közti különbséget.

```
digitalWrite(4,bitRead(input_data,0));
digitalWrite(5,bitRead(input_data,1));
digitalWrite(12,bitRead(input_data,2));
digitalWrite(13,bitRead(input_data,3));
digitalWrite(14,bitRead(input_data,4));
digitalWrite(15,bitRead(input_data,5));
```

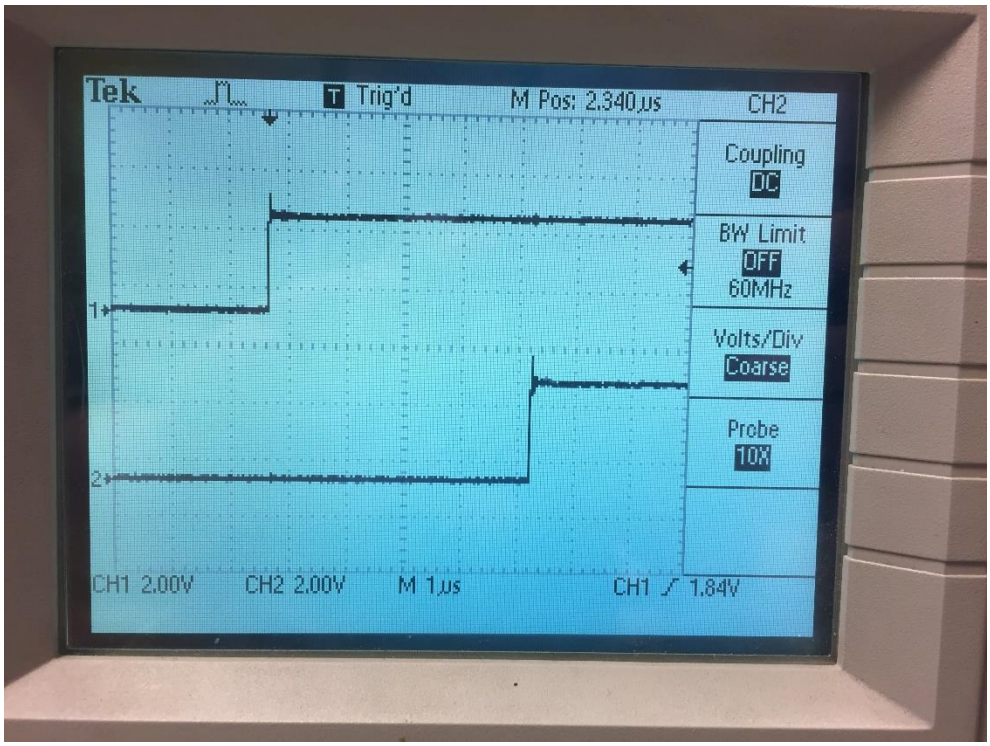
A két bit kiküldése közötti időt úgy mértem meg, hogy a megvizsgáltam GPIO4-es és a GPIO15-ös kimenet kiküldése közötti időt, és ezt öttel leosztottam, így megkaptam a `digitalWrite()` egy kimenet állításának átlagát.

A port regiszterrel történő értékadás esetén a kimenetek egyszerre változnak. Ezt mutatja a 2-6. ábra is, ahol a 25 nanoszekundumus felbontásban vizsgáltam a kimenetek értékváltását.



2-6. ábra: Értékadás port regiszterrel

A kimenetek, szekvenciális, egymás után történő értékadásánál azt tapasztaltam, hogy a kimenetek állítása lényegesen több időt vesz igénybe. A 2-7. ábrán látható, hogy a két érték felfutása között nagyjából 4.5 mikroszekundum telt el, amiből arra következtettem, hogy egy kimenet állítása jó közelítéssel 0.9 mikroszekundum.



2-7. ábra: Értékadás szekvenciálisan

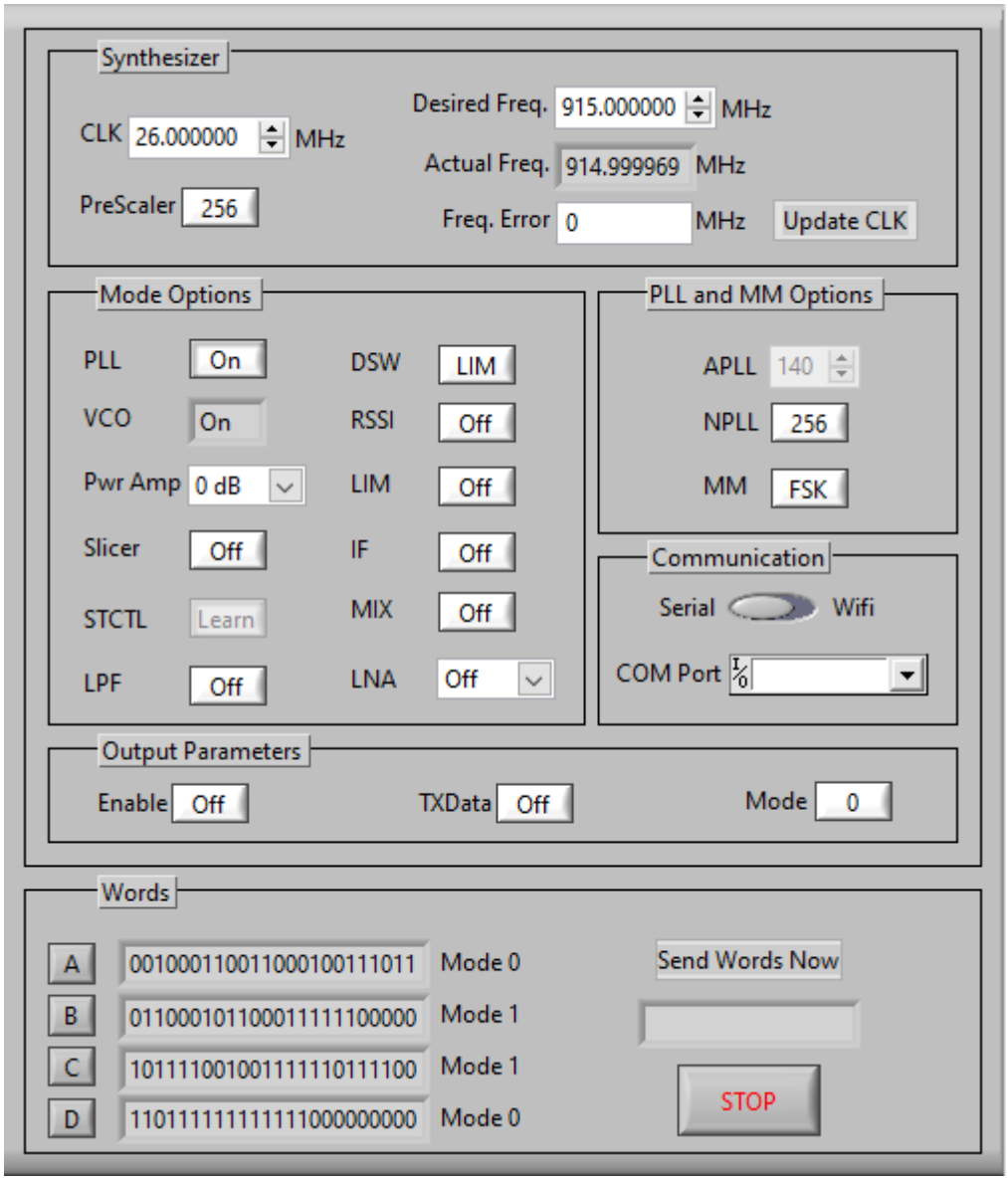
A mérést elvégeztem úgy is, hogy `bitRead()` műveleteket a kiírás előtt elvégeztem és egy-egy `bool` változóba írtam, azonban ez érdemi változást nem hozott a kimenet állításának gyorsaságában. Így is jó közelítéssel 4.5 mikroszekundumos különbséget mértem.

3. TRF6900 EVM vezérlőszoftver megvalósítása LabVIEW környezetben

A TRF6900 EVM vezérlése négy 24 bites regiszterrel történik. Ezen regiszterek megadása nehézkes és időigényes, ezért a Texas Instruments megalkotta a kártya 1-3. ábrán látható vezérlőszoftverét, ami lehetővé teszi a TRF6900 EVM szolgáltatásainak elérését magas szinten, vagyis biztosít egy olyan grafikus felhasználói felületet, amely elfedi a regiszter szintű vezérlést és a rendszerszintű paraméterek közvetlen megadását teszi lehetővé. Ez a szoftver azonban csak régi operációs rendszereken működik (pl. Windows 95, 98, XP), és a vezérlés közvetlenül párhuzamos porton keresztül működik, ami a jelenkor számítógépeinek túlnyomó többségében nincs.

Az ebben a fejezetben bemutatásra kerülő alkalmazás célja, hogy új operációs rendszerrel működő, és párhuzamos port kimenetet nem tartalmazó számítógépeken is vezérelhető legyen a TRF6900 EVM kártya. A szoftver fejlesztése során szem előtt tartottam, hogy az eredeti szoftver kezelőfelületéhez hasonlót hozzak létre, így azok, akik már megszokták a Texas Instruments által készített felületet, könnyen tudnak áttérni az általam elkészített verzióra. Ugyanazon funkciók kerültek implementálásra a kommunikációs protokoll megvalósítása, valamint a kapcsolat létrehozása során a NodeMCU mikrokontrollerrel az USB alapú soros vezérléssel és WiFin keresztül is.

3.1. Felhasználói felület



3-1. ábra: A TRF6900 EVM LabVIEW-ban megvalósított vezérlő ablaka

A szoftver kezelőfelülete hat részre oszlik [9], amelyeket alább ismertetek.

Synthesizer

- CLK: A kívánt órajel frekvencia megadása, ami beállítható a pontos érték bevitelével, a jobb oldalon található fel-le nyilakkal és az egér görgőjével.
- Desired Freq.: A frekvenciaszintézer kívánt frekvenciájának megadása, ami beállítható a pontos érték bevitelével, a jobb oldalon található fel-le nyilakkal és az egér görgőjével.

- PreScaler: A PLL frekvenciaosztójának megadása. Értéke 256 és 512 lehet, amit a kétállású kapcsolóval választhatunk ki. A PreScaler értéke mindig megegyezik az NPLL értékével (PLL and MM Options). A PLL kikapcsolt állapotában a kapcsoló letiltásra kerül.
- Actual Freq.: A frekvenciaszintézer valós frekvenciája. Ezt az értéket a program számolja a Δf alapján, amit részletesen a 3.2 fejezetben tárgyalok
- Freq.Error: A kívánt és a valós frekvencia ritkán egyezik meg. E hiba kompenzálására szolgál. Megadása az érték beírásával MHz-ben.
- Update CLK: Az órajel frekvencia frissítése a frekvenciahiba alapján a nyomógombra nyomást követően.

Mode Options

- PLL: A PLL ki és bekapcsolását állító kétállású kapcsoló.
- VCO: A VCO állapotát mutató kijelző. Mindig bekapcsolt állapotban van. Amennyiben kikapcsoljuk, akkor külső VCO használata szükséges.
- Pwr Amp: Az RF kimenő teljesítmény szintjének beállítása a legördülő menü használatával. Lehetséges értékek: Kikapcsolás, 20dB, 10dB vagy 0dB csillapítás.
- Slicer: A döntő áramkör ki- és bekapcsolását állító kétállású kapcsoló.
- SLTCL: Ha a döntő áramkör bekapcsolt állapotban van, akkor az SLTCL kétállású kapcsolóval tudunk a Tanulás (Learn) vagy Tartás (Hold) üzemmódok közül választani. Ellenkező esetben a kapcsoló le van tiltva.
- LPF: Az aluláteresztő szűrő ki és bekapcsolását állító kétállású kapcsoló.
- DSW: Kétállású adatkapcsoló. LIM beállításnál az aluláteresztő szűrő bemenetére a demodulátort, RSSI beállításnál pedig az RSSI-t állítja.
- RSSI: A vett jelszint indikátor ki- és bekapcsolását állító kétállású kapcsoló.
- LIM: A középfrekvenciás (KF) főerősítő és limiter ki és bekapcsolását állító kétállású kapcsoló.
- IF: A középfrekvenciás (KF) előerősítő ki és bekapcsolását állító kétállású kapcsoló.
- MIX: A keverő ki- és bekapcsolását állító kétállású kapcsoló.
- LNA: A rádiófrekvenciás (RF) kiszajú előerősítő ki- és bekapcsolását állító kétállású kapcsoló.

Output Parameters

- Enable: A TRF6900 EVM ki (készenléti állapot) és bekapcsolását állító kétállású kapcsoló.
- TXData: A TXData adatvonalat állító kétállású kapcsoló.
- Mode: „0” és „1” üzemmódot állító kétállású kapcsoló

PLL and MM Options

- APLL: A PLL zárthurkú befogási sávszélességének megadása. Lehetséges értékek: 0,20,40,60,80,100,120 és 140 és a jobb oldalon található fel-le nyilakkal állítható. A PLL bekapcsolt állapotában a kapcsoló letiltásra kerül.
- NPLL: A PLL frekvenciaosztójának megadása. Értéke 256 és 512 lehet, amit a kétállású kapcsolóval választhatunk ki. Az NPLL értéke mindig megegyezik az PreScaler értékével (Synthesizer). A PLL kikapcsolt állapotában a kapcsoló letiltásra kerül.
- MM: A Moduláció módját kiválasztó kapcsoló. Ebben a programban mindig FSK értékű.

Communication

- Serial-WiFi kapcsoló: Kommunikáció módjának kiválasztása. Lehetséges értékek: USB soros porti vagy WiFi kommunikáció.
- COM Port: A soros port kiválasztására. Csak abban az esetben látható, ha a Serial-WiFi kapcsoló Serial értékre van állítva.
- IP: A TCP kapcsolat IP címének megadása. Csak abban az esetben látható, ha a Serial-WiFi kapcsoló WiFi értékre van állítva.
- Port: A TCP kapcsolat Port-jának megadása. Csak abban az esetben látható, ha a Serial-WiFi kapcsoló WiFi értékre van állítva.

Words

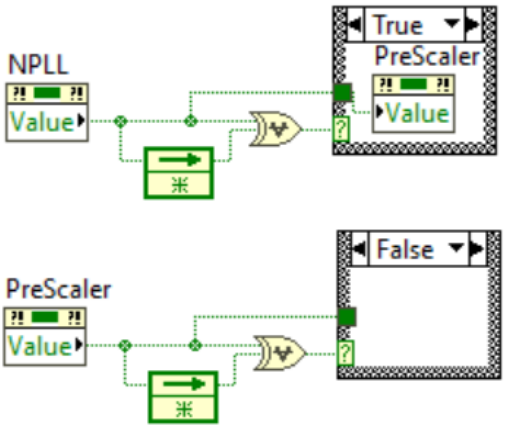
- A, B, C, D kódszavak: A program további részeiben megadott beállításokból generált kódszavak.

- A, B, C, D gombok: A megfelelő kódszó kiküldése a TRF6900 EVM kártyára.
- Send All Words: Az összes kódszó kiküldése a TRF6900 EVM kártyára.
- Kijelző: A kiküldés állapotának kiírása.
- STOP: A TRF6900 EVM vezérlőprogram leállítása.

A felhasználói felület egyes paraméterei között függőség áll fent.

- A Slicer kikapcsolt állapotában az SLTCL letiltva, bekapcsolt állapotában engedélyezve van
- A PLL kikapcsolt állapotában az NPLL és PreScaler letiltva, bekapcsolt állapotában engedélyezve van
- A PLL bekapcsolt állapotában az APLL letiltva, kikapcsolt állapotában engedélyezve van
- Az NPLL és PreScaler egyikének értékét megváltoztatom, akkor a másik is megváltozik.

Az NPLL és a PreScaler értékeinek összehangolását a 3-2. ábrán látható módon oldottam meg.

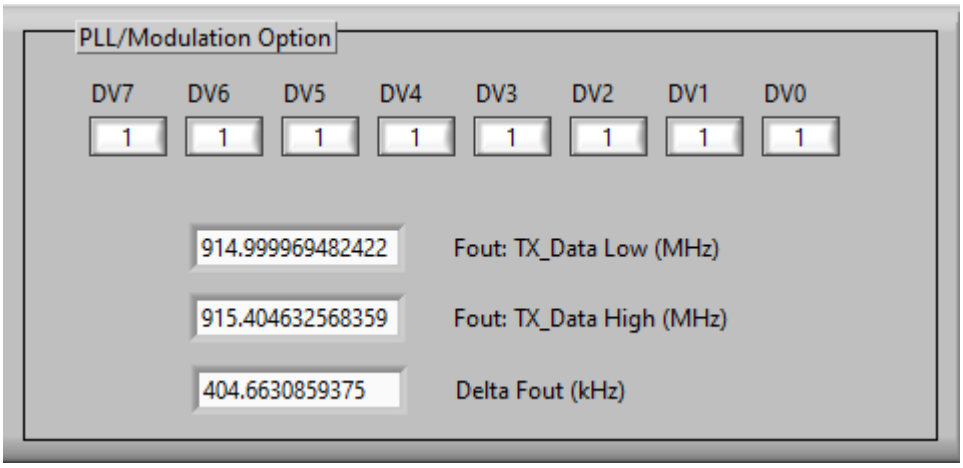


3-2. ábra: NPLL és PreScaler összehangolása

A LabVIEW programnyelvben a Feedback node megőrzi a változó, előző ciklusban felvett értékét. A kizáró vagy művelet csak abban az esetben állítja a kimenetet 1-be ha a két bemenő érték különböző. Amennyiben a jelenlegi és a Feedback node kimeneti értéke különbözik egymástól, akkor változást detektáltunk, ezért a másik kapcsoló értékét

is megváltoztatjuk. Mivel minden élváltás alkalmával a két kapcsoló értékét összehangolom, ezért, ha nem történt változás, vagyis a kizáró vagy művelet kimenete 0, akkor az NPLL és a PreScaler értéke is egyenlő, tehát nincs szükség újabb összehangolásra.

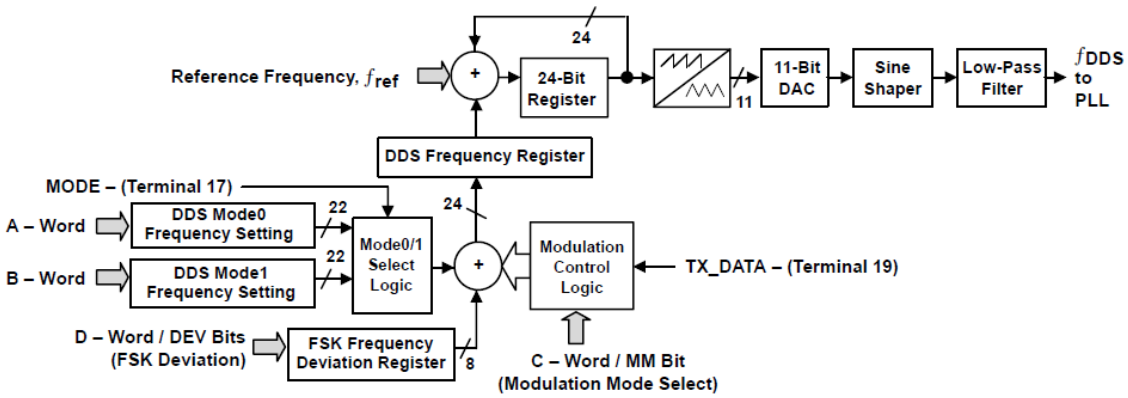
Az alkalmazás 3-3. ábrán látható FSK modulációs ablakában a frekvencialöketet meghatározó Deviation regisztert (DEV) változtathatjuk. Az aktuálisan beállított értéktől függően változik az $F_{out:TX\_Data=High}$ és a két frekvencia különbsége is. A három frekvenciaérték számítását részletesen a 3.2-es fejezetben tárgyalom.



3-3. ábra: A LabVIEW-ban megvalósított vezérlőszoftver PLL/FSK modulációs ablaka

3.2. Kódszavak generálása

A TRF6900 EVM kártyán található Direkt Digitális Szintézer (DDS) blokkvázlata a 3-4. ábrán látható. Az ábrán az f_{ref} jelölés a felhasználói felületen megadott órajel frekvenciát (CLK), míg az f_{DDS}, a DDS áramkör kimeneti és egyúttal a PLL bemeneti frekvenciáját jelöli [11].

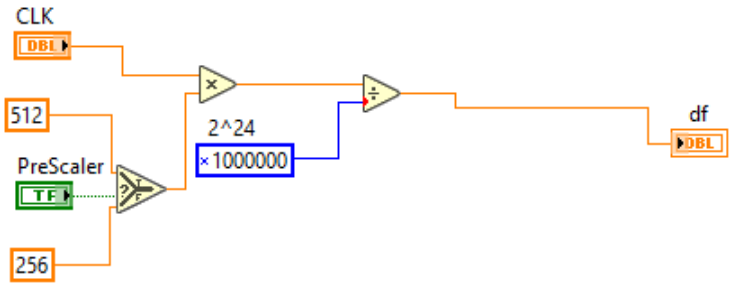


3-4. ábra: A TRF6900 EVM Direkt Digitális Szintézerének blokkdiagrammja

A referencia oszcillátor frekvenciájának (f_{ref}), és a PLL osztójának (N) ismeretében meghatározható a DDS áramkör frekvencia felbontása (Δf), ami a DDS két egymással szomszédos kimeneti értékének különbsége. Ezt frekvenciaraszternek nevezzük és alábbi kifejezéssel adható meg:

$$\Delta f = \frac{N * f_{ref}}{2^{24}} \tag{3-5}$$

A fenti egyenlet megvalósítását LabVIEW-ban a 3-6. ábra mutatja be.



3-6. ábra: Δf számítása LabVIEW-ban

A feszültség vezérelt oszcillátor (VCO) kimenete (f_{out}), nem más, mint a szintézer kívánt frekvenciája (Desired Freq.). amely függ a DDS értékétől. 0-ás és 1-es üzemmód Desired Freq. paramétere eltérhet egymástól, ezért a DDS is különböző értéket vehet

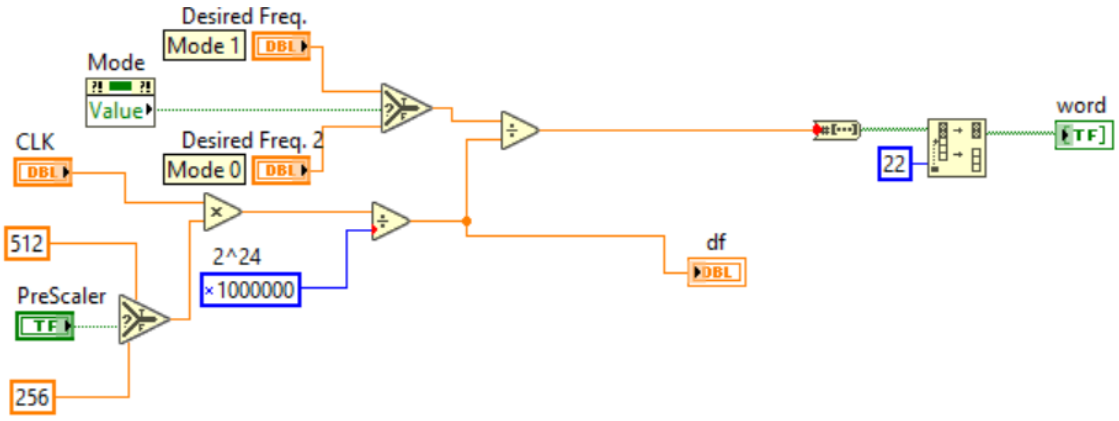
fel. Bevezetjük DDS_0-t ami Mode 0-hoz és DDS_1-t ami Mode 1-hez tartozó új frekvenciaregiszter.

$$f_{out} = DDS_x * N * \frac{f_{ref}}{2^{24}} \tag{3-7}$$

Az egyenlet átrendezésével megkapjuk az egyenletet DDS_x értékére kifejezve.

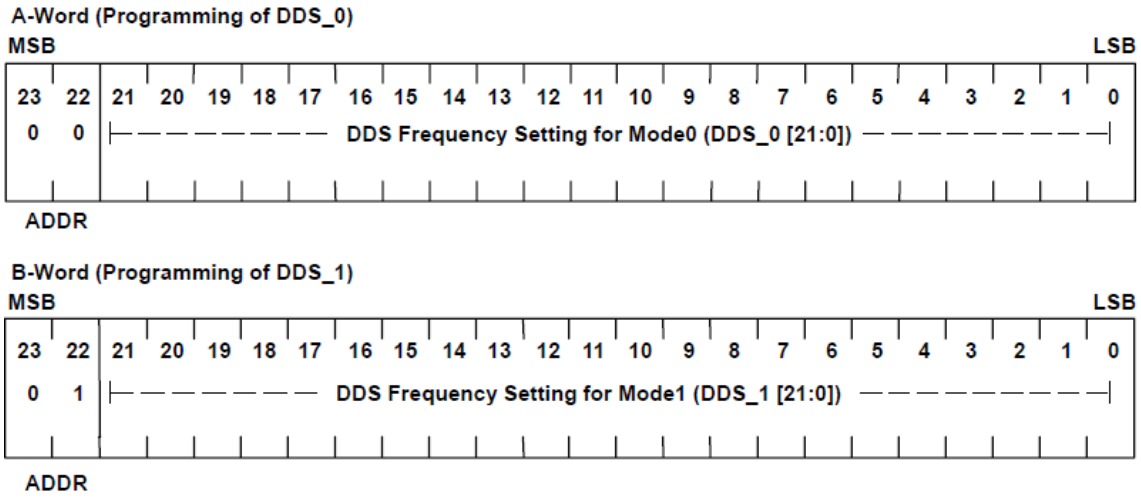
$$DDS_x = \frac{f_{out} * 2^{24}}{N * f_{ref}} \tag{3-8}$$

A vezérlőszoftverben az 3-9. ábrán látható módon, két különböző regiszterben tárolom a kívánt frekvencia értékeket. A Mode kapcsoló állapota dönti el melyik frekvenciából számolom a DDS_x-et, amit ezután egy 22 elemű bináris tömbbe írok.



3-9. ábra: A 22 bites DDS_x kiszámítása LabVIEW-ban

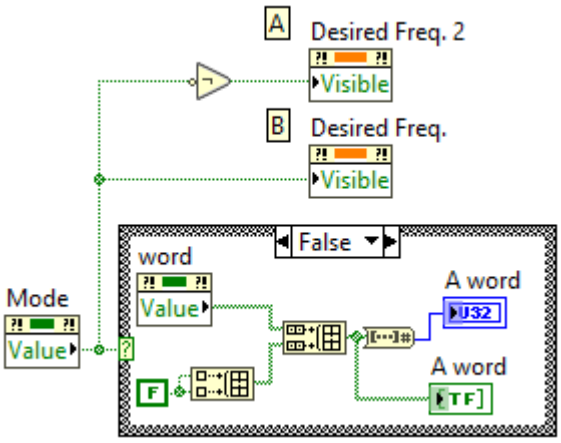
A TRF6900 EVM A és B kódszavainak összetételét a 3-10. ábra szemlélteti. A két legmagasabb helyiértékű bit a kódszavak címbitjei. Ez A kódszóban mindig 00, B esetén pedig 01 értékű. Az ezt követő 22 bit, a DDS_x értéke.



3-10. ábra: A és B kódszó felépítése

Mint azt korábban említettem a A és a B kódszóhoz használt kívánt frekvencia értékeket külön regiszterekben tárolom, amikhez egymástól független beviteli mező is tartozik. Mivel a felhasználói felületen a Mode kapcsoló értéke dönti el, hogy éppen melyik kódszót változtatom, így a két beviteli mező közül mindig csak az aktuális üzemmódhoz tartozó lehet látható. Az éppen inaktív beviteli mezőt a LabVIEW Property Node funkciójával a 3-11. ábra szerint tudom láthatatlanná tenni.

Ezután a kiszámolt DDS_x értékét a megfelelő kódszóba írom. Ennek megvalósítására egy feltételvizsgálatot kell végeznünk. Ha a Mode értéke 0, akkor a DDS_x az A, ellenkező esetben B 0-21 bitjeire írom.



3-11. ábra: DDS_x kiírása a megfelelő kódszóba

Amint azt korábban tárgyaltuk, a Δf frekvenciaraszter, a két szomszédos frekvenciaérték közti különbség. Azonban a kívánt frekvencia és a frekvenciaraszter hányadosa (ami DDS_x , a (3-5) -ös és (3-8)-as egyenletből kiszámolva) legtöbbször nem egész szám, így a valós frekvencia a kívánt értéktől $\pm\frac{1}{2}\Delta f$ értékkel eltérhet. Az egyenletben $DDS_{x\prime}$-nek nevezem DDS_x egész számra kerekített értékét.

$$f_{actual} = \Delta f * DDS_{x\prime} \tag{3-12}$$

Ha a szintézer tényleges frekvenciájának megmérjük és a kívánt frekvenciától való elérését a Freq. Error ablakba bevisszük, akkor az Update CLK lenyomásával a szoftver az alábbi módon korrigálja a kimenetet.

$$f_{desired\_korrigált} = f_{desired} - f_{error}$$

$$f_{desired\_korrigált} = DDS_x * N * \frac{f_{ref\_korrigált}}{2^{24}}$$

$$f_{ref\_korrigált} = \frac{f_{desired\_korrigált} * 2^{24}}{DDS_x * N} \tag{3-13}$$

Az FSK moduláció löketét meghatározó DEV regisztert a FSK modulációs beállítások ablakban adhatjuk meg. Az itt beállított regisztert alapul véve a program kiszámolja az FSK magas és alacsony értékhez tartozó frekvenciaszintet és a kettő közötti különbséget. A ΔF_{out} nem más, mint az FSK moduláció löketének kétszerese.

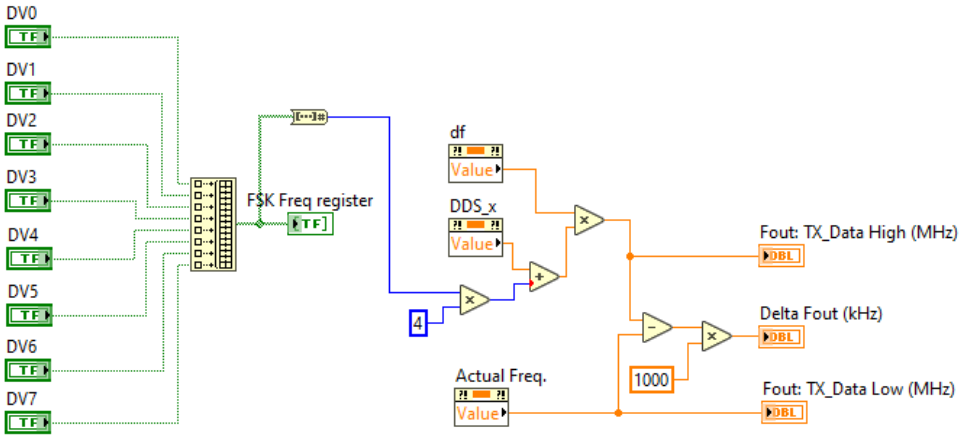
$$F_{out:TX\_Data=Low} = N * \frac{f_{ref} * DDS_x}{2^{24}} = f_{actual}$$

$$F_{out:TX\_Data=High} = N * \frac{f_{ref} * (DDS_x + 4 * DEV)}{2^{24}}$$

$$\Delta F_{out} = F_{out:TX\_Data=High} - F_{out:TX\_Data=Low} \tag{3-14}$$

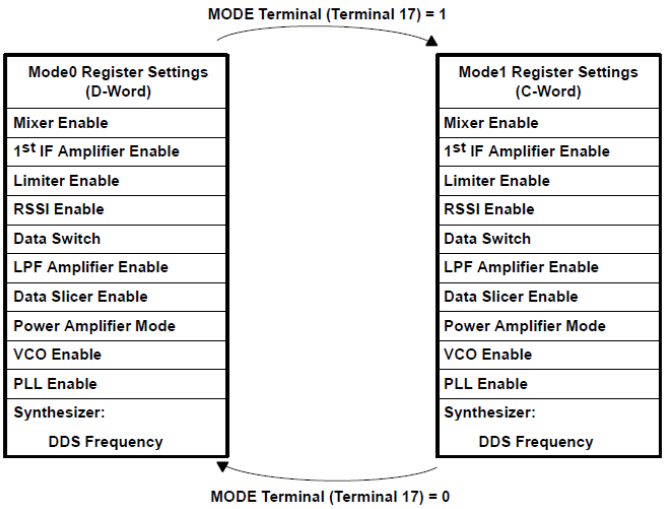
Az egyenleteket a LabVIEW programom az 3-15. ábrán látható módon oldja meg. A DEV regiszter egyes bitjeit egy-egy kétállású kapcsolóval olvassa be a felhasználói felületről. A 8 bites regisztert egy bináris tömbbe írja, és lementi azt.

A LabVIEW Bináris tömb, decimális számmá konvertáló függvénye a tömb első elemét a legalacsonyabb, az utolsót pedig legmagasabb helyiértékű bitként kezeli. Az így összeállt bináris számot decimálissá alakítja, amelyen a (3-14)-es egyenletek elvégezhetők.



3-15. ábra: FSK moduláció löketének számítása LabVIEW-ban

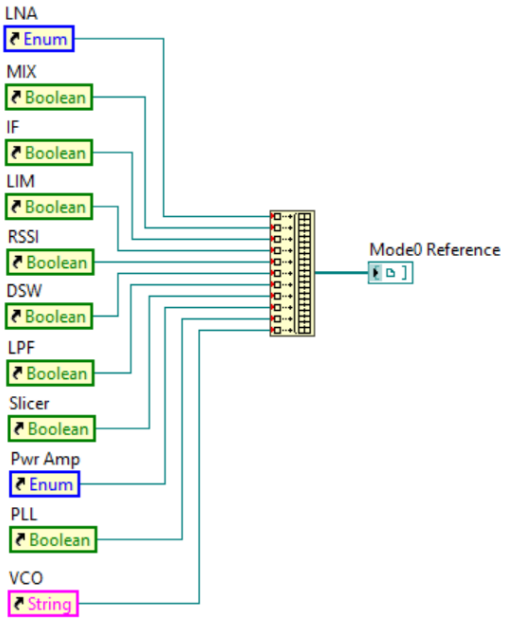
A Mode kapcsoló, két egymástól független előre megadott beállításokkal rendelkező üzemmód közti gyors váltást tesz lehetővé. Ezeknek a beállításoknak egyike, a korábban tárgyalt kívánt szintézer frekvencia (Desired Freq.), amit a 3-16. ábra is szemléltet. Szintén részei a kezelőfelület Options részén beállítható paraméterek, amelyek, Mode 1 esetén C, Mode 0 esetén D kódszó első 13 bitjét adják meg. Mivel ezek a paraméterek a két üzemmód esetében különböző értékűek lehetnek, így fontos, hogy Mode értékének ismeretében a megfelelő kódszó bitjét változtassuk és üzemmód váltáskor betöltsük az új üzemmódhoz tartozó értékeket.



3-16. ábra: Paraméterek betöltése üzemmód változtatásakor

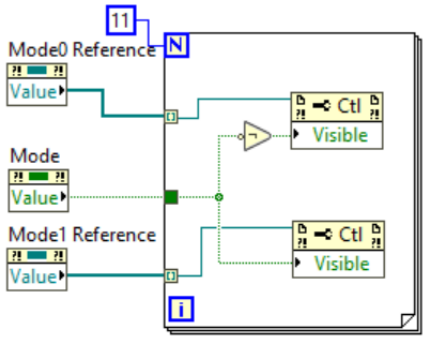
Az üzemmódok közti váltás esetén előforduló betöltést többféle módon oldhatjuk meg. Alkalmazhatnánk változás detektálást (3-2. ábrához hasonlóan), és a C és D kódszóból betölthetnénk az aktuális üzemmódhoz tartozó értéket. Ennél egyszerűbb megoldás azonban, ha Mode 0-hoz és Mode 1-hez különálló, egymástól független kapcsolókat

hozunk létre. Annak érdekében, hogy ne kelljen külön kezelőfelületet létrehozni a két üzemmódnak, a 3-11. ábrán látható megoldáshoz hasonlóan itt is az aktív kapcsolókat megjelenítjük, míg az inaktívakat eltüntetjük a felhasználói felületről. Ennek érdekében az egyes kapcsolók referenciáit, 3-17. ábrán látható módon, egy tömbbe lementem.



3-17. ábra: Mode 0 állapothoz tartozó kapcsolók referenciáit tömbbe mentem

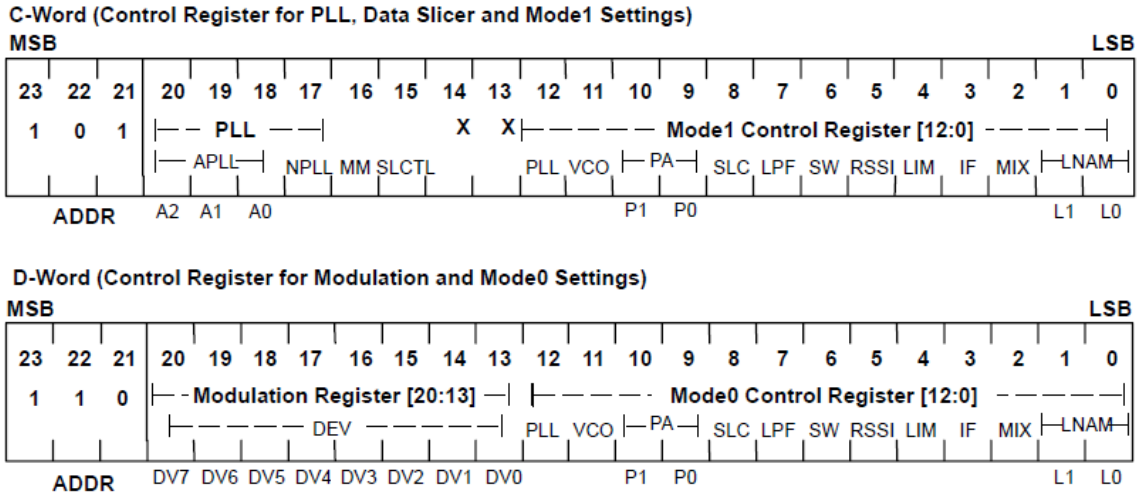
Az így kapott két tömbök elemeivel az összes kapcsoló láthatósági paraméterét ciklikusan állítom, az aktuális üzemmódnak megfelelően, amint azt a 3-18. ábra mutatja.



3-18. ábra: Kapcsolók megjelenítése és eltüntetése a kezelőfelületen

Az A és B kódszóval szemben, ahol csak a két legfelső, C és D kódszavaknál a legfelső három bitet használjuk a regiszterek címzésére. Ez C esetében 101, D esetében pedig mindig 110 értéket vesz fel, ahogy azt a 3-19. ábra szemlélteti.

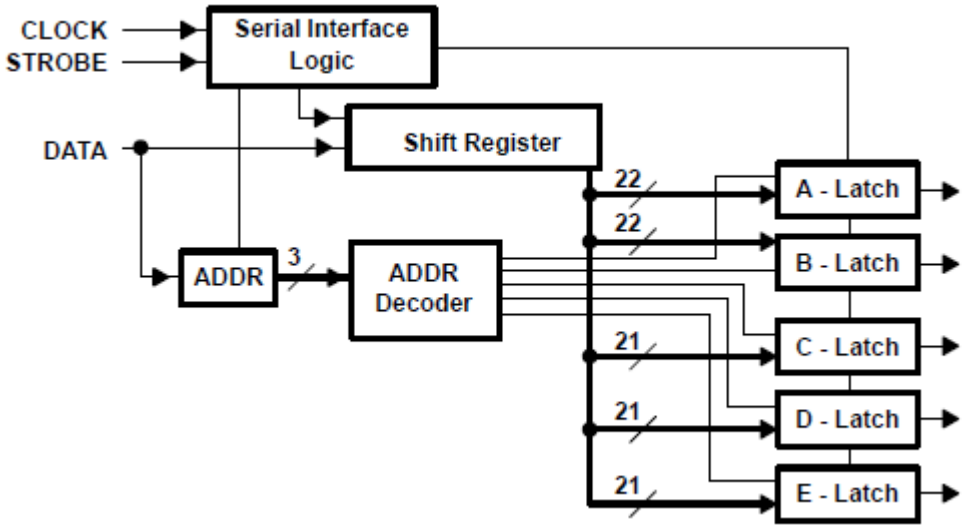
Az FSK modulációs ablakban megadott DEV regiszter a TRF6900 EVM kártya D kódszavának 13-20 bitjeiben kerül kiküldésre. A C regiszter 13 és 14-es bitjei a kártya működésére nézve nem tartalmaznak érdemi információt. A programban ezeken a helyi-értékeken 00 értéket küldünk. A 15-20 biteken PLL and MM Options ablakban megadható értékek szerepelnek. Az APLL 0-tól 140-ig vehet fel értékeket 20-as lépésközzel. Az itt megadott értéket 20-szal elosztjuk és az így megkapott értéket 3 biten tároljuk.



3-19. ábra: C és D kódszó felépítése

3.3. A TRF6900 EVM kommunikációs protokolljának megvalósítása

A TRF6900 EVM programozásához háromvezetékes egyirányú soros buszt használunk. Az órajel (CLOCK) a kommunikáció időzítésének, az adat (DATA) a 3.2 fejezetben tárgyalt kódszavak kiküldésnek, míg a STROBE az egyes kódszavak befejezését jelölő bit adatvonala. Az adat mintavételezése az órajel felfutó élénél történik, és egy 24 bites shift regiszterbe kerül beolvasásra. A STROBE magas értéke jelzi a kódszó végét. A címdekóder a legfelső három (A és B esetében csak a legfelső két) bit alapján meghatározza a Latch regisztert, amibe a shift regiszter tartalma kiírásra kerül. Amíg a STROBE jel magas, addig az órajelnek és az adatnak alacsony állapotban kell lennie. Mivel az órajel és STROBE egymás viszonylatában aszinkron jelek, így biztosítani kell, hogy a kimenet zajtól és hibától mentes legyen [11].

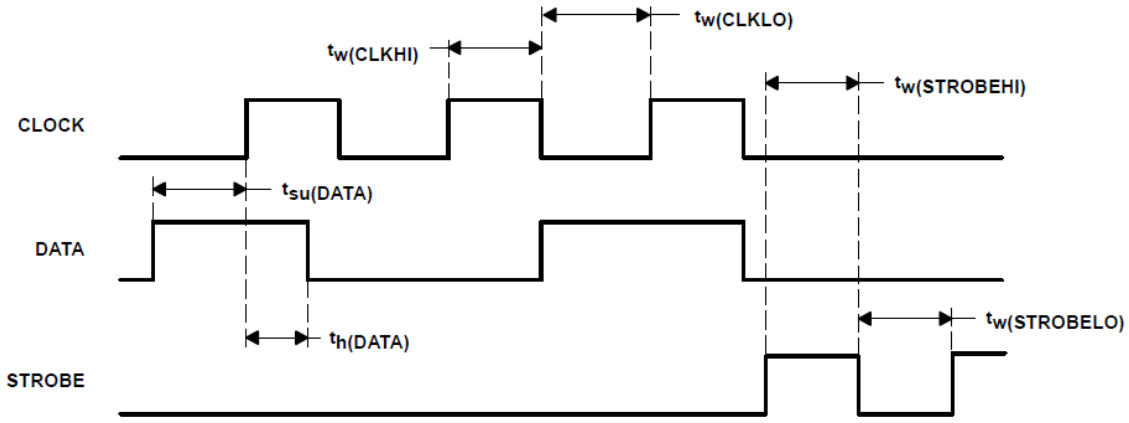


3-20. ábra: A soros interfész blokkdiagramja

A kódszavak kiküldése a legmagasabb helyiértékű bittel kezdődik. A kártya programozásához az A, B, C és D kódszavak mindegyikét ki kell küldenünk. Amennyiben a program futása közben egyes paramétereket változtatni szeretnénk, a soros interfész lehetőséget biztosít a szavak különálló kiküldésére is [11].

Az E Latch tesztelési célokra alkalmazható, amellyel jelen dolgozat nem foglalkozik.

A soros interfész időzítése a 3-21. ábrán látható módon történik. Az mintavételezés az órajel felfutó élére történik. Ezt megelőzően már az adatnak $t_{su(DATA)}$ idővel stabilnak kell lennie. Ezt az időt nevezzük beállási időnek (setup time). A mintavételezés után az adat értékét még meghatározott ideig ($t_{h(DATA)}$) stabilan tartani kell. Ezt tartási időnek (hold time) hívjuk. Az ábrán t_w időegységek, a jelek, HI index esetén magas, LO index esetén pedig az alacsony értékének minimális tartási idejét jelzik.



3-21. ábra: A soros interfész időzítése

Az ábrán jelölt időegységek alsó vagy felső korlátait a 3-22. táblázat tartalmazza.

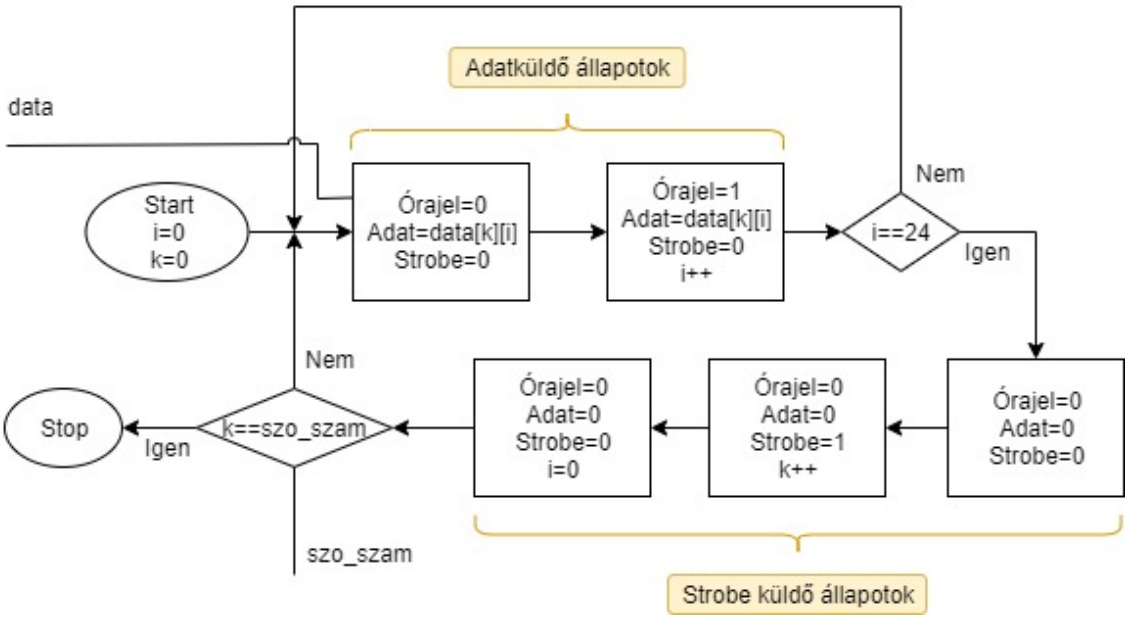
3-22. táblázat: A soros interfész korlátjai

PARAMÉTER		MIN	MAX	Mérték-egység
$f_{(CLOCK)}$	CLOCK frekvenciája		20	MHz
$t_{w(CLKHI)}$	CLOCK magas értékének impulzusideje	25		ns
$t_{w(CLKLO)}$	CLOCK alacsony értékének impulzusideje	25		ns
$t_{su(DATA)}$	DATA beállási idője CLOCK felfutása előtt	25		ns
$t_{h(DATA)}$	DATA tartási ideje CLOCK felfutása után	25		ns
$t_{w(STROBEHI)}$	STROBE alacsony értékének impulzusideje	25		ns
$t_{w(STROBELO)}$	STROBE magas értékének impulzusideje	25		ns

A fenti táblázatban meghatározott korlátokat mindig kielégíti, ha a jeleket legalább 25 ns hosszúságú ideig kitartjuk, és adat váltását az órajel lefutó élénél eszközöljük. Ezt a megoldásom jelentősen túlteljesíti, az időzítések jellemzően 10ms nagyságrendűek.

A protokoll állapotgéppel történő megvalósítását a 3-23. ábra szemlélteti. Mivel a TRF6900 EVM soros interfésze támogatja az összes kódszó egyszerre történő, és azok különálló beküldését is, így az állapotgép megkapja az elküldésre váró szavak számát (szo_szam). Ezen kívül szintén bemeneti értékként megadjuk a DATA adatvonalra kiírni kívánt kódszavak tartalmát. Ez egy kétdimenziós tömb és az állapotgép diagrammon data névvel szerepel.

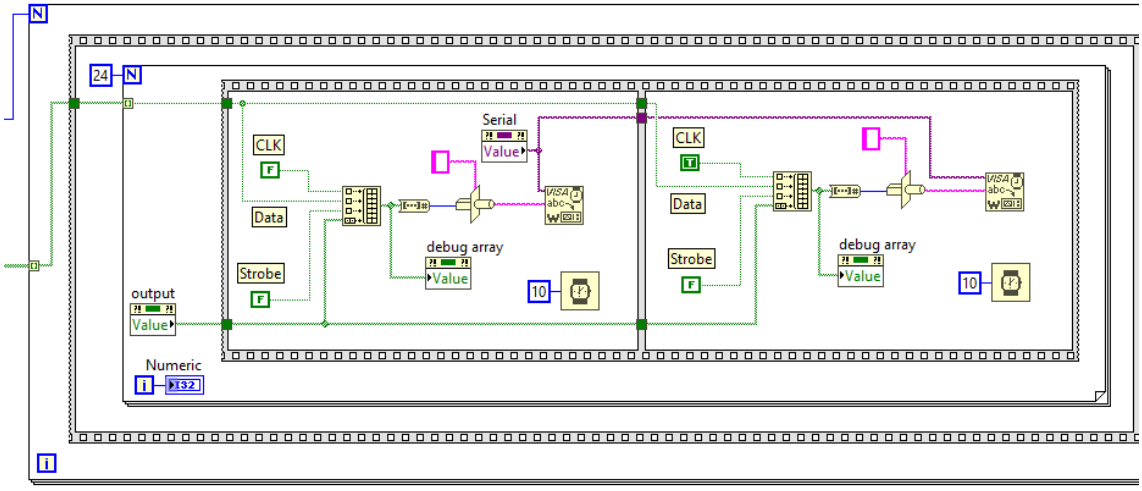
A számlálók (i és k) kezdeti értékadása után az állapotgép az első fázisba lép. Itt az órajel 0-s értéket vesz fel, az adatvonalra pedig kitesszük az első kódszó legelső elemét. Ezt követően a második állapotban az órajel felfut és az adat mintavételezése megtörténik. Itt megnövelem az i számláló értékét, hogy a következő ciklusban már az első kódszó második elemét kapjam meg. Ez a ciklus addig megy, amíg a teljes 24 bites adatot kiküldtem, vagyis, ha 24 ciklus lezajlott. Ekkor jelezzük a TRF6900 EVM kártyának, hogy az első kódszó összes bitje kiküldésre került, a STROBE adatvonal magas állapotba emelésével. A k számláló az aktuálisan már kiküldött szavak számát tárolja. Amennyiben ez egyenlő az elküldésre váró szavak számával, akkor a ciklus befejeződik, az állapotgép STOP állapotba kerül, ahonnan csak újbóli hívásra kerül megint START állapotba.



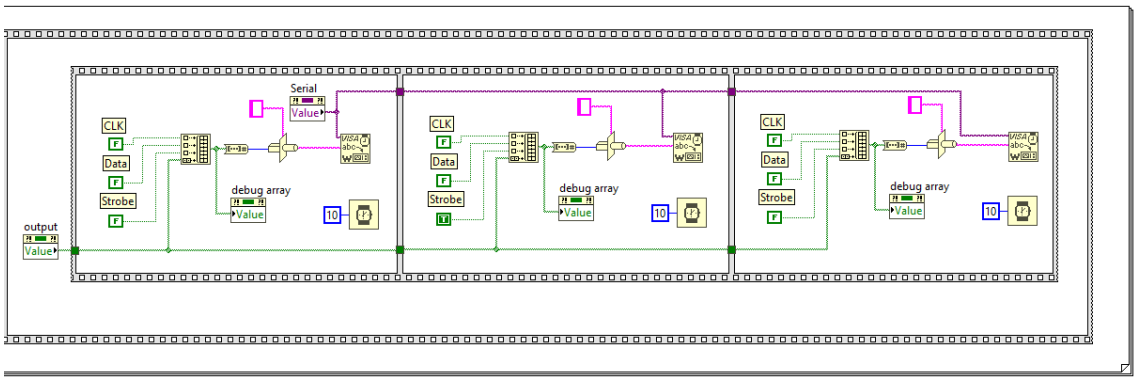
3-23. ábra: A kommunikációs protokoll állapotgépes leírása

A LabVIEW az adatfolyam modell alapján futtatja a programkódot, vagyis az egyes elemek akkor kerülnek végrehajtásra, ha az összes szükséges bemenet elérhető. Az állapotgépes leírásban az a célunk, hogy az állapotok egymás utáni sorrendben hajtódjanak végre, és a TRF6900 EVM soros interfészének korlátait betartsák, azaz, hogy az adatvonalakon az egyes biteket legalább 25 ns időtartamig tartsák. A LabVIEW programozási nyelvben is van lehetőség a szekvenciális végrehajtásra a Flat Sequence struktúrával.

A 3-24. és a 3-25. ábrán látható a protokoll megvalósítása LabVIEW-ban. Az modul megkapja a kétdimenziós tömböt és a szavak számát, ami meghatározza, hány alkalommal fog a ciklus lefutni. Ezt követően 24 bitnyi adatot kiküldünk és a Strobe jellel lezárjuk az adott kódszó küldését.



3-24. ábra: Adatküldő állapotok



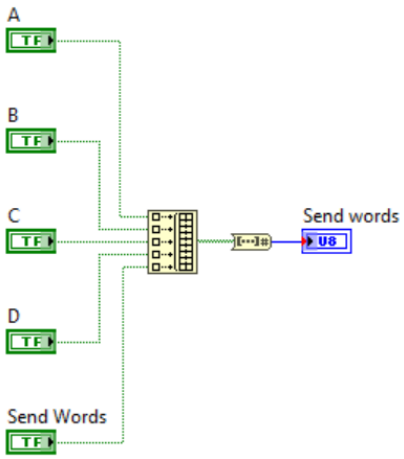
3-25. ábra: Strobe küldő állapotok

3.4. Kommunikáció megvalósítása

A LabVIEW vezérlőszoftver korábban tárgyalt részeiben a kódszavak előállításra kerültek a grafikus kezelőfelületen megadott paraméterek alapján, és a kommunikáció protokollja implementálásra került az állapotgépes struktúra szerint. Az alábbiakban az adatok NodeMCU mikrokontrollerre történő kiküldését tárgyalom.

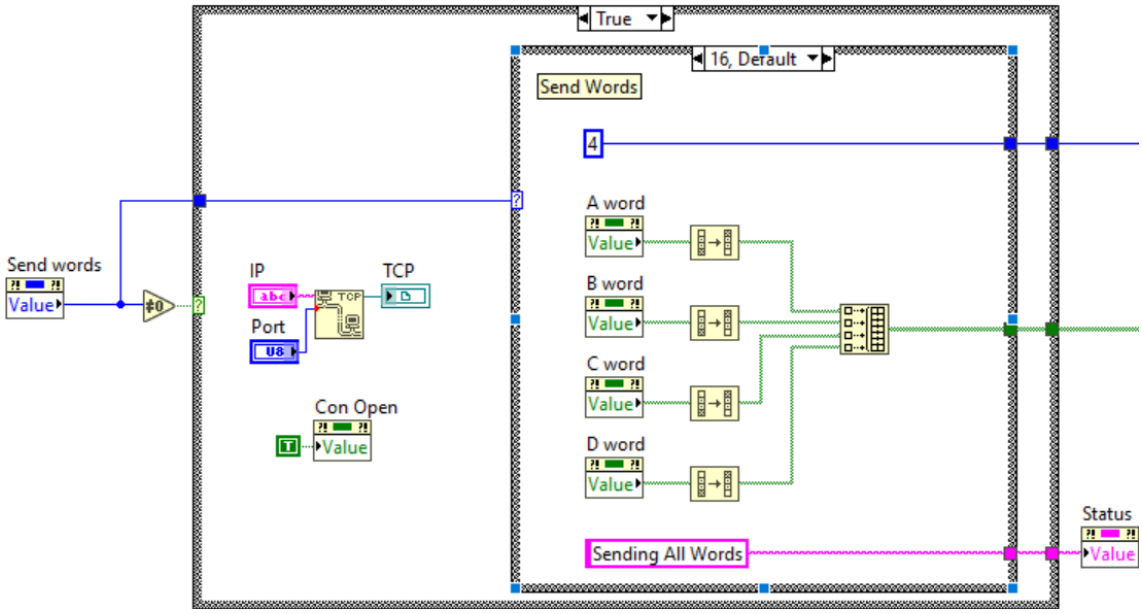
Egy vagy az összes kódszó kiküldését a felhasználói felületen öt gombbal lehet elindítani. Az A, B, C, D gombok az egyes szavak míg a Send All Words mind a négy kódszó egymás után történő kiküldését biztosítja. Ezen gombok bármelyikének lenyomása a kommunikációs csatorna megnyitását és a kódszavak küldésének megkezdését jelenti. A gombok a lenyomás után csak rövid ideig kerülnek magas állapotba, mivel egyszeri kiolvasása után értékük automatikusan visszavált alacsonyba.

A gombokat a 3-26. ábrán látható módon bináris tömbbe rendezem majd ezt decimális számmá átalakítom. Amennyiben az eredmény kettő bármelyik hatványa abban az esetben egy gomb került lenyomásra. Ha az eredmény nullától eltér, de nem kettő hatványa, úgy több gomb lett aktiválva. Ez utóbbi esetben az összes kódszó kiküldésre kerül.



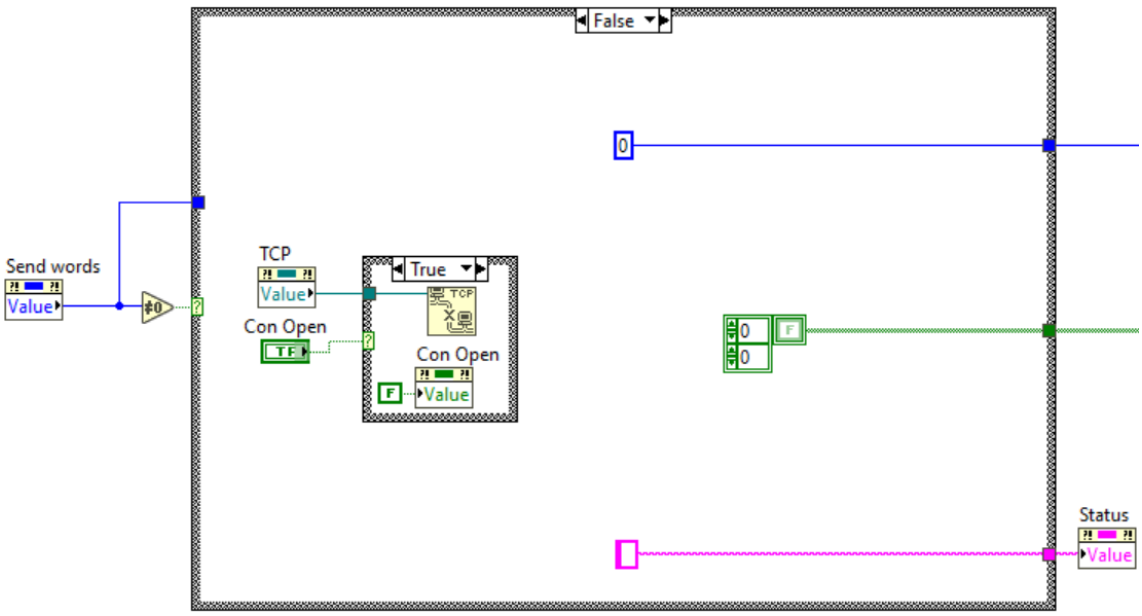
3-26. ábra: Kiküldést indító gombok beolvasása

A LabVIEW-ban megvalósított vezérlő szoftver és a NodeMCU közötti kapcsolatot kétféle módon oldottam meg. A kommunikáció történhet soros porton vagy WiFi-n keresztül is. A TCP egy kapcsolat-orientált protokoll, ami biztosítja a sorrendhelyes átvitelt az eszközök között. Mivel a LabVIEW és a NodeMCU is támogatja a TCP protokollt, így a vezetéknélküli kommunikációt ezen keresztül valósítottam meg. A 3-27. ábrán a TCP kapcsolat létrehozását láthatjuk. A gombok állapotából létrehozott szám meghatározza, hogy melyik szavak kerüljenek kiküldésre. Amennyiben ez a szám 16 (2^4), akkor a program elindítja az összes kódszó kiküldését, vagyis a 3.3 fejezetben tárgyalt protokollnak a 4 kódszót tartalmazó kétdimenziós bináris tömb és a szavak számát jelölő decimális 4-es kerül kiküldésre.



3-27. ábra: Kommunikációs csatorna megnyitása és a kétdimenziós tömb előállítása

Amennyiben az összes adat küldése befejeződött, 3-28. ábrán látható módon lezárhatjuk a kapcsolatot. Ha a gombok állapotából képzett decimális szám nulla, azaz egyik gomb sem került lenyomásra, akkor a program 0 decimális értéket küld a protokollt megvalósító állapotgépnek, ami megakadályozza a ciklus elindulását.



3-28. ábra: Kommunikációs csatorna lezárása

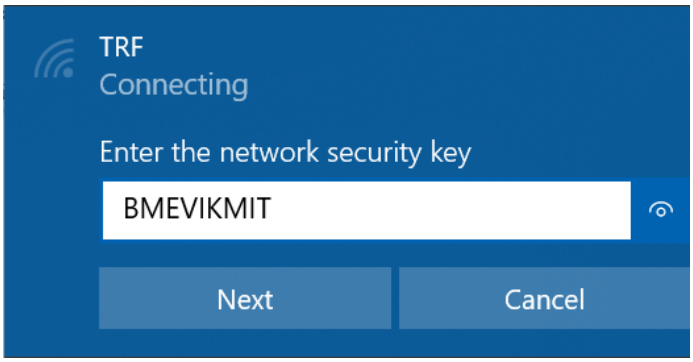
4. A vezérlő szoftver tesztelése

A tesztelés a rendszer fejlesztését kezdettől fogva végig kísérte. A kódszavak generálását a Texas Instruments eredeti vezérlő szoftvere által generált értékkekkel hasonlítottam össze. A protokoll korrekt működését segéd indikátorokkal, mikrokontroller egyes kimenetein megjelenő értékeket, a kimenetre kötött LED-ekkel, multiméterrel és oszcilloszkóppal is ellenőriztem.

A kész rendszer tesztelését az alábbi módokon végeztem el.

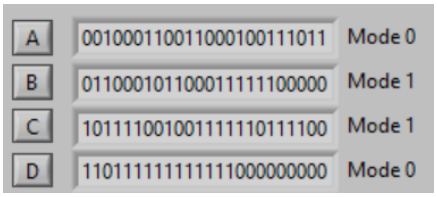
4.1. Kommunikációs protokoll tesztelése Serial Monitorral

A teszt elvégzése során, a 4-1. ábrán látható módon, csatlakoztam a NodeMCU által létrehozott Soft Access Pointhoz. A LabVIEW-ból WiFi-n keresztül beérkező adatokat a mikrokontrollerrel soros porton keresztül kiírattam és a Platform IO Serial Monitor-ján vizsgáltam.



4-1. ábra: Csatlakozás WiFi-n keresztül a NodeMCU-hoz

A LabVIEW-ból WiFi-n keresztül a 4-2. ábrán látható kódszavak kerültek kiküldésre.



4-2. ábra: WiFi-n keresztül kiküldött kódszavak

A Serial Monitoron megjelenő adatokat a 4-3. ábra mutatja. Az ábrán két 6 bites szó egy órajel ciklust jelenít meg. Mivel a mintavételezés mindig az órajel felfutó élénél történik, ezért az adat már az azt megelőző szóban kiküldésre kerül. A kódszavak végén kiküldött három 6 bites szó a Strobe jel kiküldését mutatja.

A 6 bites szavak felépítése:

MODE	ENABLE	TX_DATA	STROBE	DATA	CLK

A CLK egyes értéke jelzi az órajel felfutó élét.

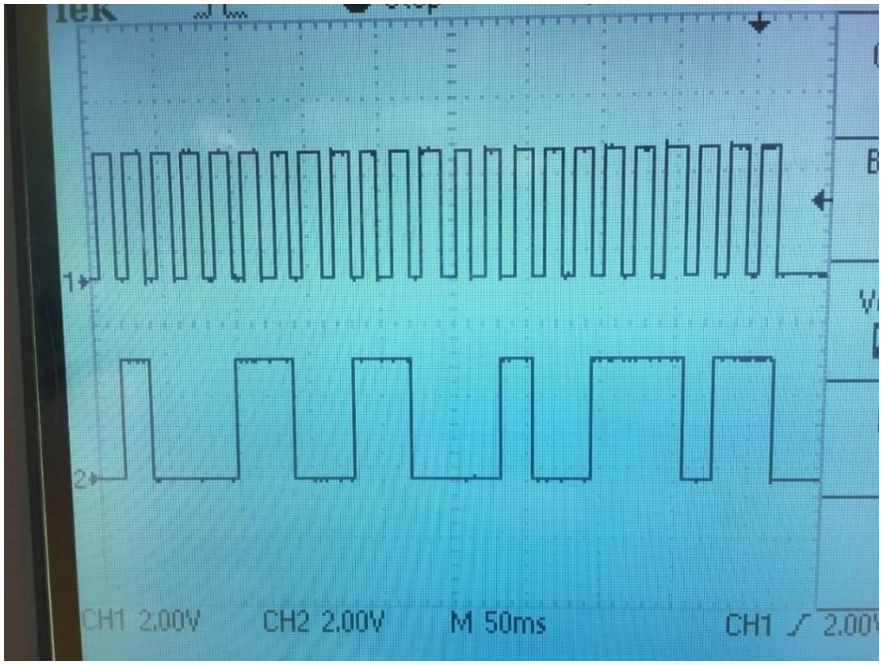
```
--- Miniterm on COM4  9600,8,N,1 ---
--- Quit: Ctrl+C | Menu: Ctrl+T | Help: Ctrl+T followed by Ctrl+H ---
Connection established.
Mode,Enable,TX_Data,Strobe,Data,CLK

000000, 000001, 000000, 000001, 000010, 000011, 000000, 000001, 000000, 000001
000000, 000001, 000010, 000011, 000010, 000011, 000000, 000001, 000000, 000001
000010, 000011, 000010, 000011, 000000, 000001, 000000, 000001, 000000, 000001
000010, 000011, 000000, 000001, 000000, 000001, 000010, 000011, 000010, 000011
000010, 000011, 000000, 000001, 000010, 000011, 000010, 000011
000000, 000100, 000000

000000, 000001, 000010, 000011, 000010, 000011, 000000, 000001, 000000, 000001
000000, 000001, 000010, 000011, 000000, 000001, 000010, 000011, 000010, 000011
000000, 000001, 000000, 000001, 000000, 000001, 000010, 000011, 000010, 000011
000010, 000011, 000010, 000011, 000010, 000011, 000010, 000011, 000000, 000001
000000, 000001, 000000, 000001, 000000, 000001, 000000, 000001
000000, 000100, 000000

000010, 000011, 000000, 000001, 000010, 000011, 000010, 000011, 000010, 000011
000010, 000011, 000000, 000001, 000000, 000001, 000010, 000011, 000000, 000001
000000, 000001, 000010, 000011, 000010, 000011, 000010, 000011, 000010, 000011
000010, 000011, 000010, 000011, 000000, 000001, 000010, 000011, 000010, 000011
000010, 000011, 000010, 000011, 000000, 000001, 000000, 000001
000000, 000100, 000000

000010, 000011, 000010, 000011, 000000, 000001, 000010, 000011, 000010, 000011
000010, 000011, 000010, 000011, 000010, 000011, 000010, 000011, 000010, 000011
000010, 000011, 000010, 000011, 000010, 000011, 000010, 000011, 000010, 000011
000000, 000001, 000000, 000001, 000000, 000001, 000000, 000001, 000000, 000001
000000, 000001, 000000, 000001, 000000, 000001, 000000, 000001
000000, 000100, 000000
```

4-3. ábra: Teszt Serial Monitoron

4.2. Tesztelés oszcilloszkóppal

A protokoll helyes működését oszcilloszkóppal is teszteltem. A 4-4. ábrán az órajel felül, míg az adatvonal alul látható. Mint azt a korábbi tesztben is láthattuk, az adat mindig

az órajel lefutó élénél vált. Továbbá ellenőriztem, hogy a 3-22. táblázatban megadott értékek a protokoll során mindvégig teljesültek. A mérés során, a jeleket stabilnak értékeltem.



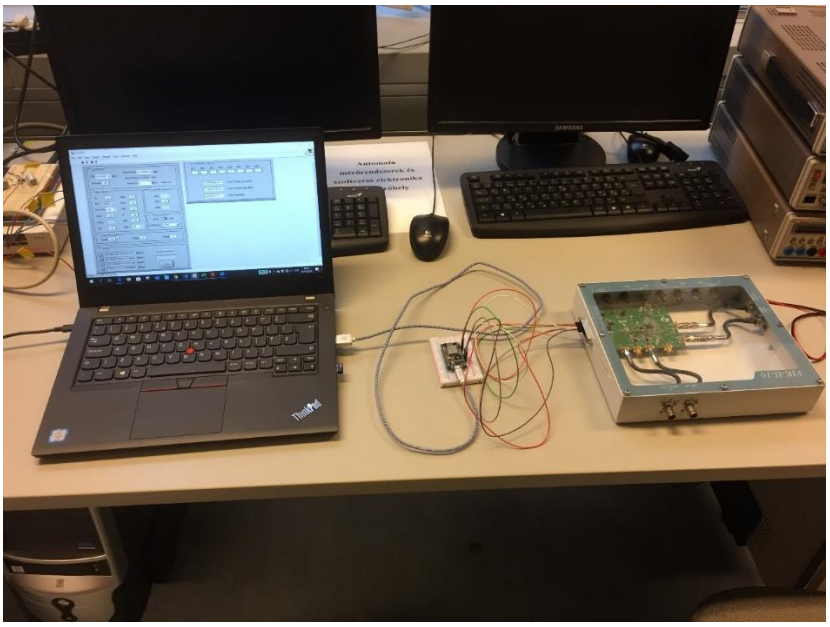
4-4. ábra: Kódszó kiküldés mérése oszcilloszkópon

A 4-5. ábrán az alsó jel a Strobe. Mint ahogy a protokollban elő van írva, látható, hogy az órajel a Strobe magas értéke során mindvégig alacsony állapotban van.

4-5. ábra: Strobe jel kiküldés mérése oszcilloszkópon

5. Összegzés

A dolgozat célja a TRF6900 EVM gyári szoftverének kiváltása volt, egy új rendszer megvalósításával. A megalkotott rendszert az 5-1. ábra mutatja.



5-1. ábra: A megvalósított rendszer

A feladat végrehajtása során megismerkedtem a NodeMCU IoT eszközzel, és ezt felhasználva létrehoztam egy olyan fizikai interfészt, ami a vezérlőszoftvertől adatokat fogad USB alapú soros portról illetve WiFi-n keresztül, és azt a párhuzamos port megfelelő bemenetére továbbítja.

Új ismeretekre tettem szert a LabVIEW használata során, amely segítségével egy olyan felhasználói felületet hoztam létre, amely elfedi a TRF6900 EVM regiszter szintű vezérlését, és a különböző paraméterek megadását közvetlenül lehetővé teszi. A LabVIEW-ban írt programban implementálásra került a TRF6900 EVM kommunikációs protokollja, amely biztosítja az adatok helyes kiküldését.

A projekt többféle módon fejleszthető tovább. Egyik ilyen lehetőség a TRF6900 EVM funkcionális blokkjainak vezérlése, egy grafikus blokkdiagramon keresztül. Érdekes feladat lehetne még a Bluetooth-on keresztüli kommunikáció megvalósítása. A Bluetooth-t a feladat során használt NodeMCU ESP-12E nem támogatja, azonban a NodeMCU másik verziója, az ESP-32S igen.

Köszönetnyilvánítás

Ezúton szeretném köszönetemet kifejezni konzulensemnek, Krébesz Tamásnak, aki a dolgozat készítése során mindvégig hasznos tanácsokkal, ötletekkel és útmutatással segítette munkámat.

Irodalomjegyzék

[1] K. Tamás, „900 MHz-es FSK adatátviteli berendezés mérése," BME-MIT, [Online]. Available: https://www.mit.bme.hu/system/files/oktatas/targyak/vedett/11031/meres10_FSK.pdf. [Hozzáférés dátuma: december 2019].

[2] „ESP8266 Datasheet," Espressif, [Online]. Available: https://www.espressif.com/sites/default/files/documentation/0a-esp8266ex_datasheet_en.pdf. [Hozzáférés dátuma: december 2019].

[3] „Visual Studio Code," [Online]. Available: https://code.visualstudio.com/. [Hozzáférés dátuma: december 2019].

[4] „Platform IO bővítmény," [Online]. Available: https://platformio.org/. [Hozzáférés dátuma: december 2019].

[5] „Labview," National Instruments, [Online]. Available: http://www.ni.com/.

[6] „Arduino Reference Guide," Arduino, [Online]. Available: https://www.arduino.cc/reference/en/. [Hozzáférés dátuma: december 2019].

[7] „ESP8266 Arduino Core documentation," [Online]. Available: https://arduino-esp8266.readthedocs.io/en/latest/index.html. [Hozzáférés dátuma: december 2019].

[8] „ESP8266 Arduino Core Github repository," [Online]. Available: https://github.com/esp8266/Arduino. [Hozzáférés dátuma: december 2019].

[9] „TRF6900 Evaluation Board User's Guide," Texas Instruments, [Online]. Available: https://www.mit.bme.hu/eng/system/files/oktatas/targyak/11051/01_trf6900_user_manual_evaluation_board.pdf. [Hozzáférés dátuma: december 2019].

[10] „Port regiszterek működése," Arduino, [Online]. Available: https://www.arduino.cc/en/Reference/PortManipulation. [Hozzáférés dátuma: december 2019].

[11] „TRF6900A Single-Chip RF Transceiver," Texas Instruments, [Online]. Available:

https://www.mit.bme.hu/system/files/oktatas/targyak/11051/02_trf6900a_chip_data_sheet.pdf. [Hozzáférés dátuma: december 2019].

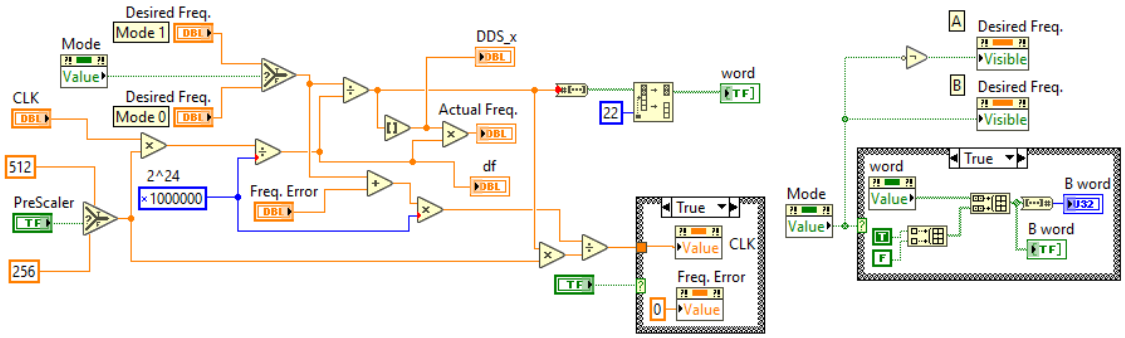
Függelék

NodeMCU programkódja

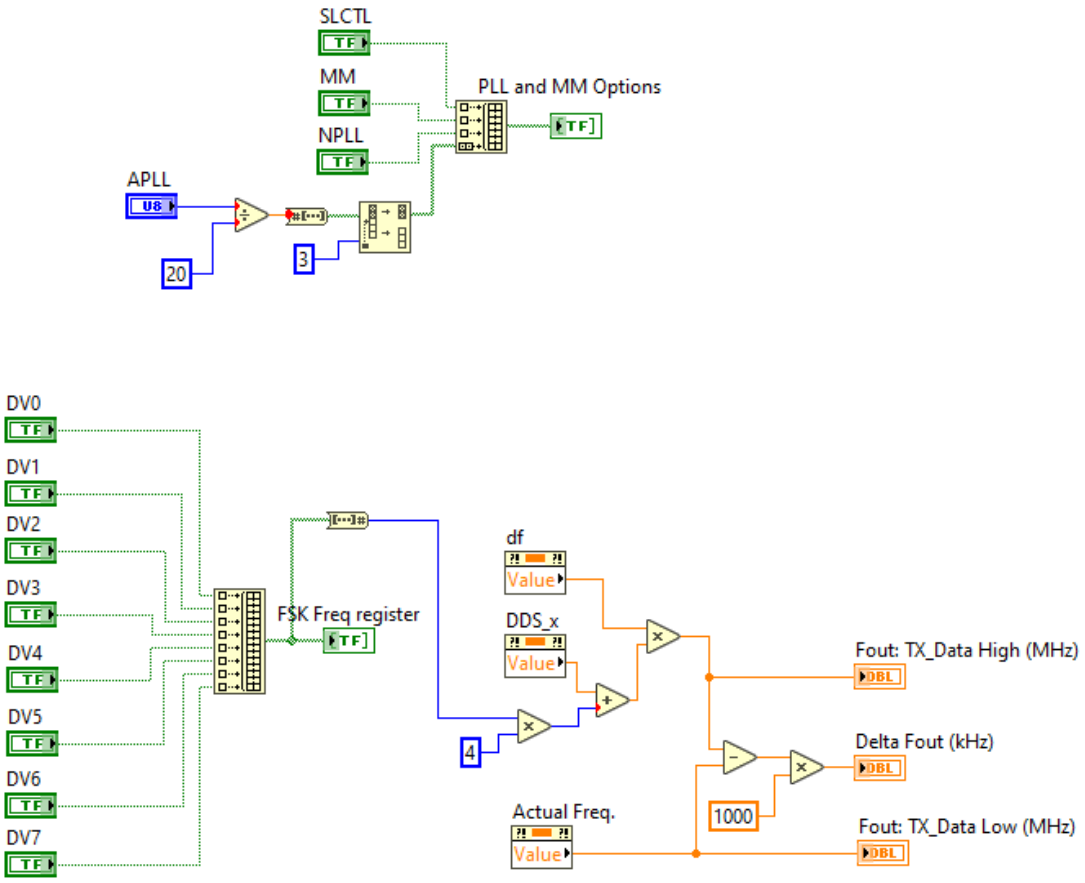
```
#include <Arduino.h>
#include <ESP8266WiFi.h>
#include <WiFiClient.h>
#include <ESP8266WebServer.h>

int mask_first4bits;
int mask_last2bits;
int input_data;

const char* ssid = "TRF";
const char* password = "BMEVIKMIT";

WiFiServer server(23);
WiFiClient client;

void WriteToPins() {
  //A beerkezo 0,1 bit 4,5 bitre shiftelese es 2-5 bit 12-15 bitre shiftelese
  GPO = ((input_data << 10) & mask_first4bits) + ((in-
put_data << 4) & mask_last2bits);
}

void setup() {
  Serial.begin(9600);

  WiFi.softAP(ssid, password);
  server.begin();
  server.setNoDelay(true);

  mask_first4bits = 0xF000;    //HEX F000 = BIN 1111 0000 0000 0000
  mask_last2bits = 0x0030;     //HEX 0030 = BIN 0000 0000 0011 0000
  input_data = 0;

  pinMode(4,OUTPUT);     //CLK
  pinMode(5,OUTPUT);     //Data
  pinMode(12,OUTPUT);    //Strobe
  pinMode(13,OUTPUT);    //TX_Data
  pinMode(14,OUTPUT);    //Enable
  pinMode(15,OUTPUT);    //Mode

  GPO = 0;               //Az osszes PIN 0-ra allitasa
}
```

```
void loop() {
  while(Serial.available()) {
    input_data = Serial.read();
    WriteToPins();
  }

  if (server.hasClient()) {
    client = server.available();
  }


  while(client.available()) {
    input_data = client.read();
    WriteToPins();
  }

}
```

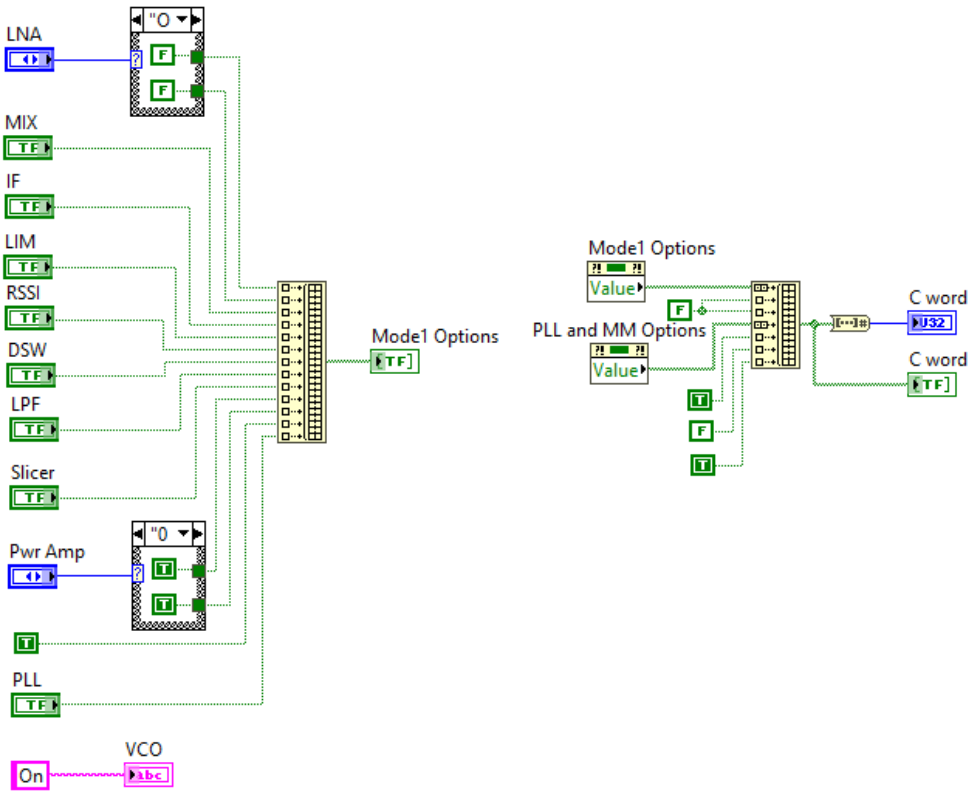
LabVIEW programkód:



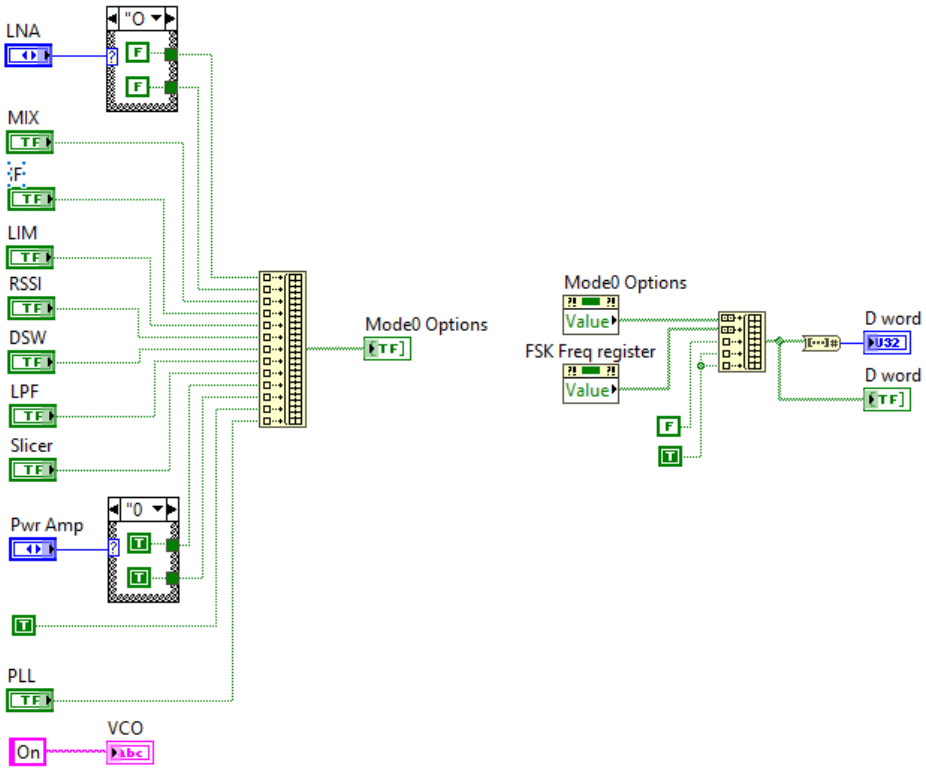
Függelék 1.ábra: A és B kódszó generálása



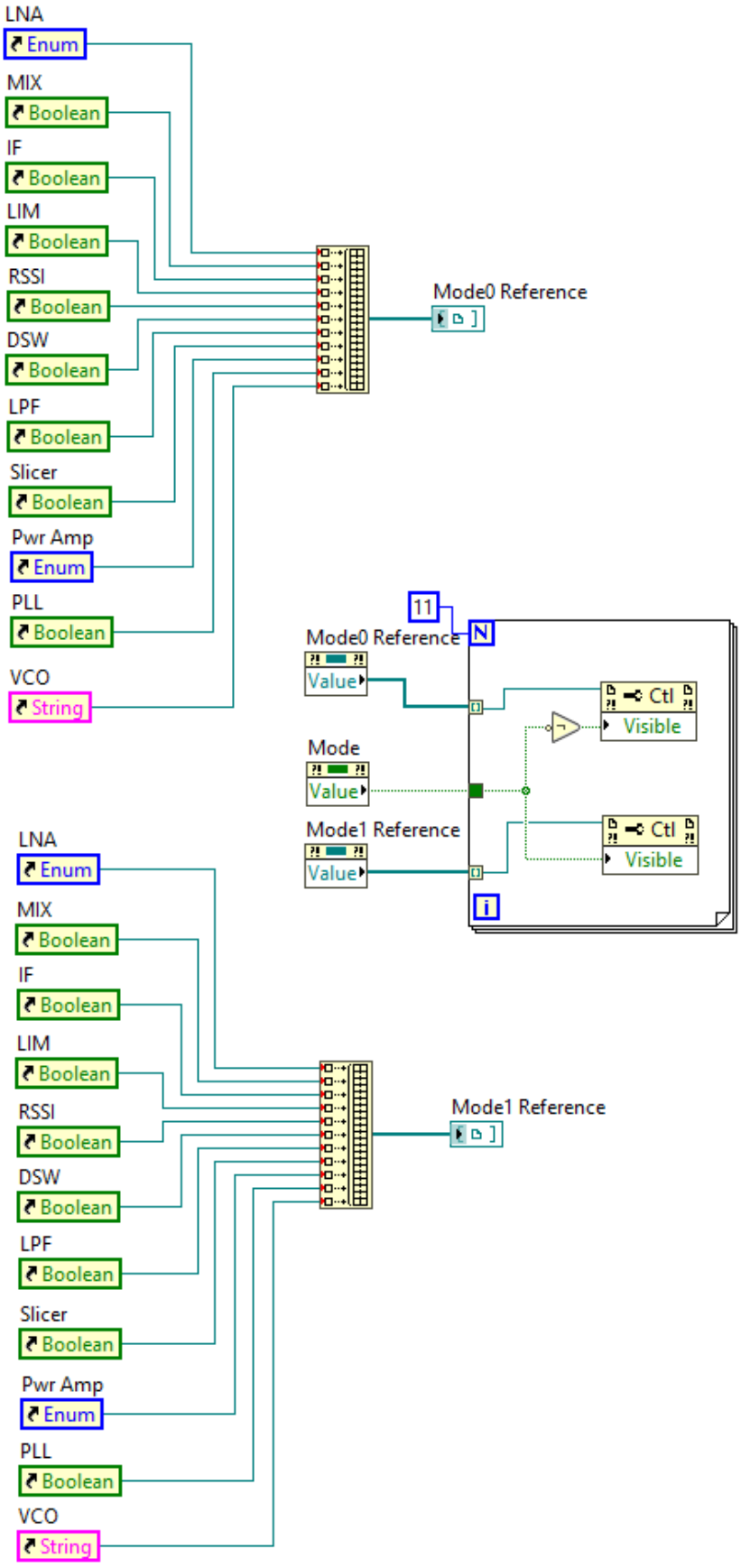
Függelék 2.ábra: PLL és FSK beállítások és számítások elvégzése



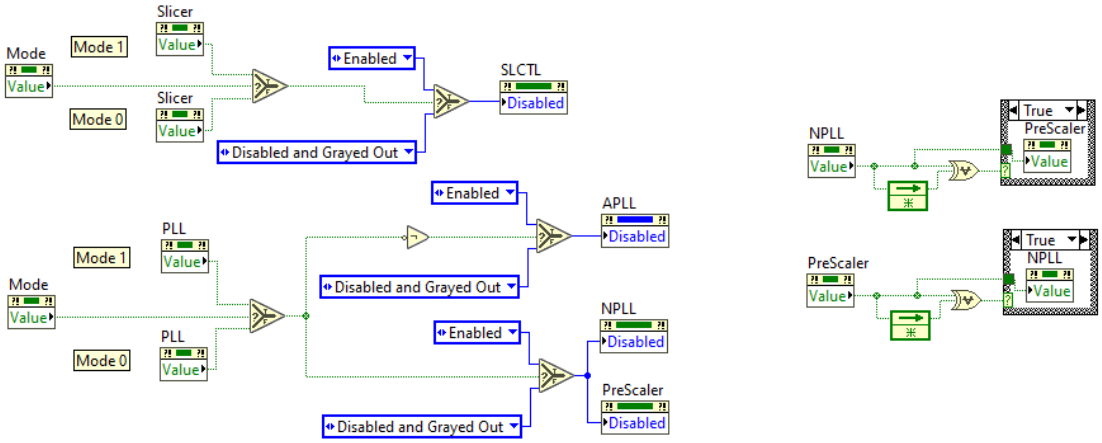
Függelék 3.ábra: C kódszó generálása



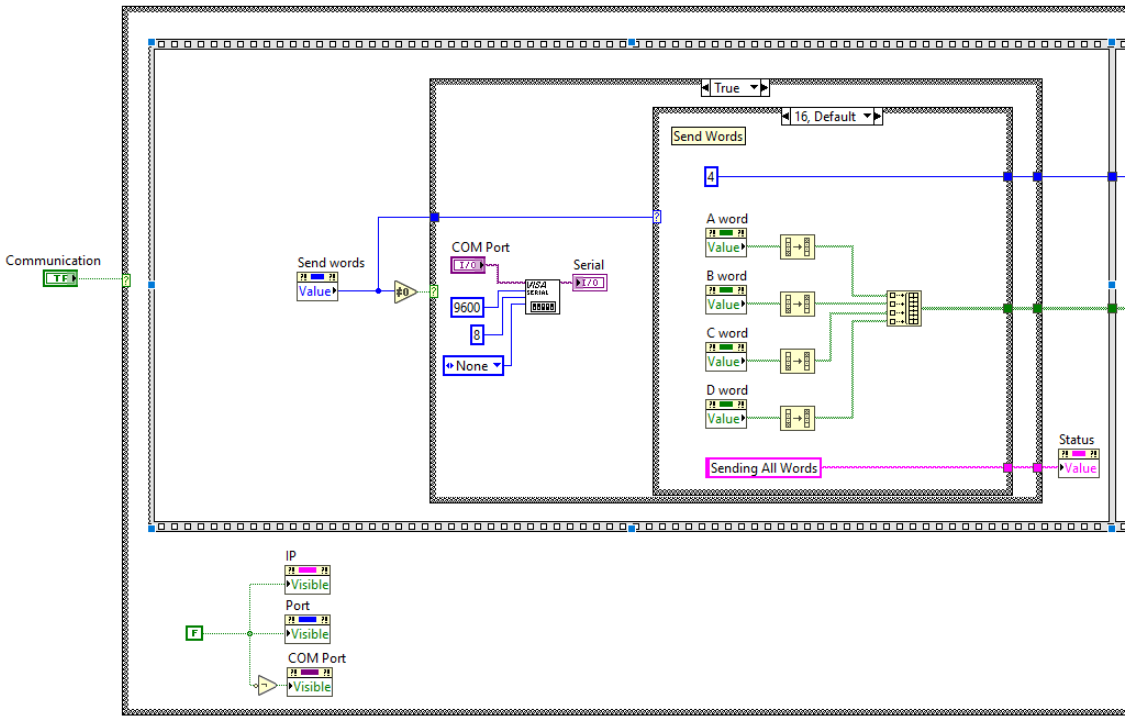
Függelék 4.ábra: D kódszó generálása



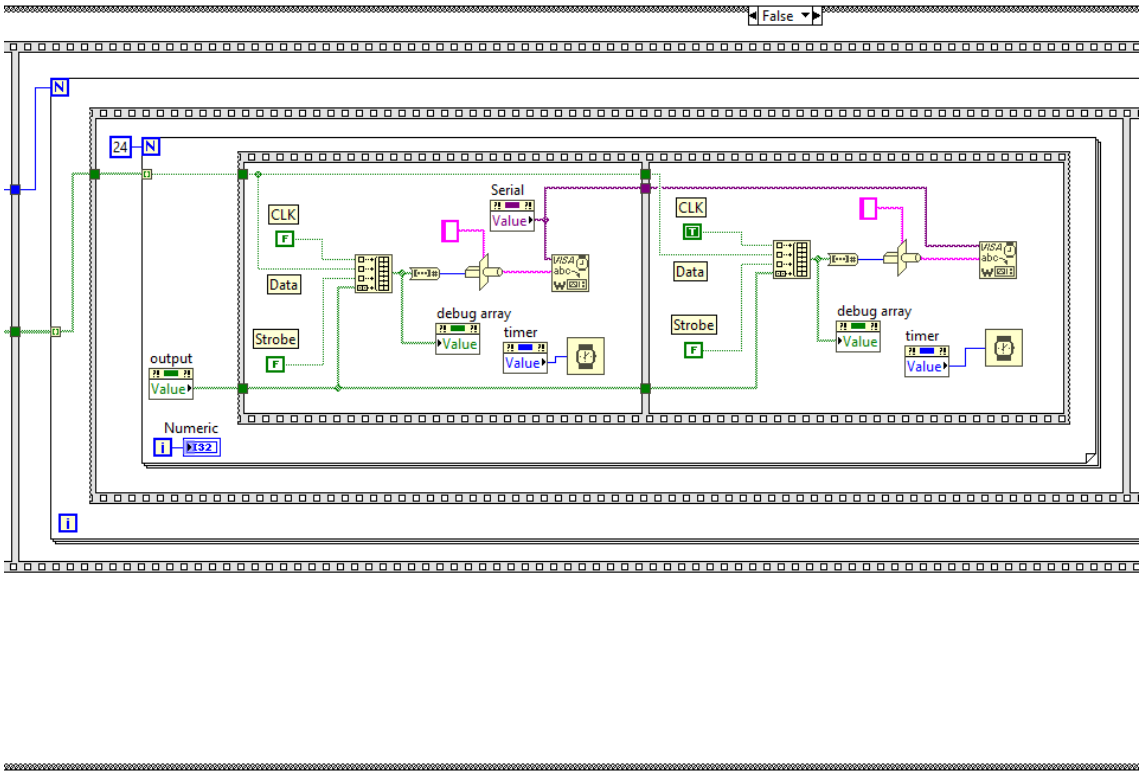
Függelék 5.ábra: Üzemmód váltás



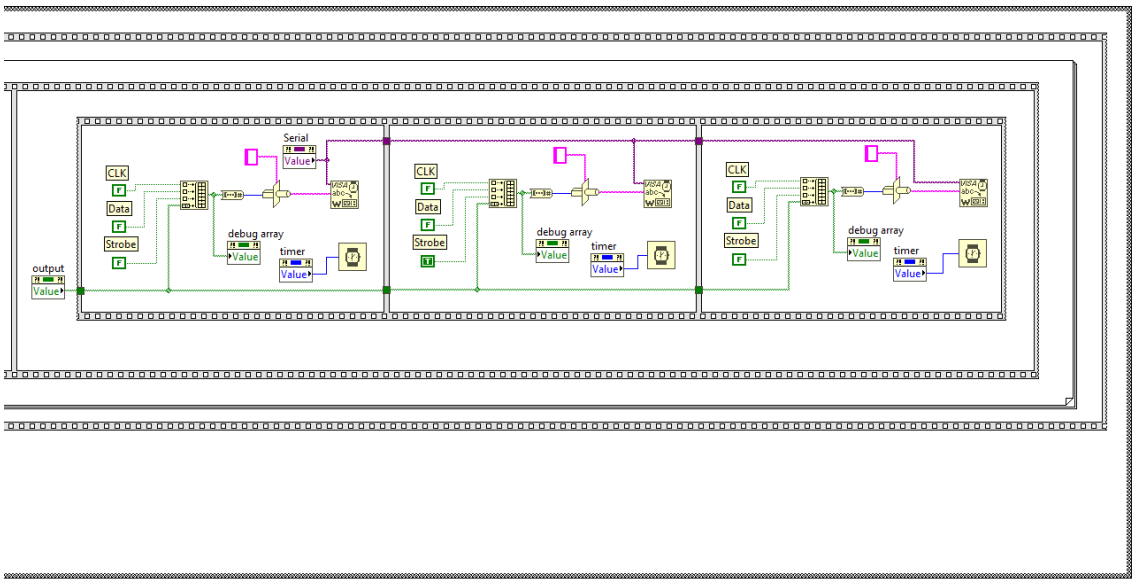
Függelék 6.ábra: Függőségek beállítása



Függelék 7.ábra: Soros porti kommunikáció 1.rész



Függelék 8.ábra: Soros porti kommunikáció 2.rész



Függelék 9.ábra: Soros porti kommunikáció 3.rész