Balogh Róbert Máté

Flexibilisen alkalmazható ipari mérés adatgyűjtő rendszerek kialakítása TCP/IP hálózat felhasználásával

Szak: Mérnök Informatikus BSc

Konzulens:

Tihanyi Attila

Budapest, 2011

Nyilatkozat

Alulírott Balogh Róbert Máté, a Pázmány Péter Katolikus Egyetem Információs Technológiai Karának hallgatója kijelentem, hogy ezt a szakdolgozatot meg nem engedett segítség nélkül, saját magam készítettem, és a szakdolgozatban csak a megadott forrásokat használtam fel. Minden olyan részt, melyet szó szerint, vagy azonos értelemben, de átfogalmazva más forrásból átvettem, egyértelműen a forrás megadásával megjelöltem. Ezt a Szakdolgozatot más szakon még nem nyújtottam be.

Budapest, 2011. december 01.

.....
Balogh Róbert Máté

Tartalomjegyzék

1 Bevezetés 8

2 Irodalomkutatás 10

2.1 Ipar 10

2.2 Internet Protokoll kialakulása 13

2.2.1 IPv4 14

2.2.2 Az Internet növekedése 16

2.2.3 IPv6 17

2.2.4 TCP/IP protokoll felépítése 18

2.3 Modbus protokoll 20

2.3.1 Bevezetés 20

2.3.2 Modbus protokoll felépítése 22

2.3.2.1 Átviteli módok 23

2.3.2.2 Hiba detektálás 24

2.3.2.3 CRC Hiba ellenörző eljárás 26

2.3.2.4 LRC Hiba ellenörző eljárás 26

2.3.2.5 ASCII keret 27

2.3.2.6 Remot Terminal Unit (RTU) keret 27

2.3.2.7 Cím mező 28

2.3.2.8 Funkció mező 28

2.3.2.9 Adat mező 29

2.3.2.10 Hiba ellenőrző mező 29

2.3.2.11 Megvalósított funkció parancsok 29

2.3.2.11.1 Read holding registers(code: 03) 29

2.3.2.11.2 Preset multiple registers(code: 16) 30

2.4 Adatbázis rendszerek 32

2.4.1 Adatbázis rendszer felépítése 33

2.4.1.1 Mérő berendezés 34

2.4.1.2 Java applikáció 34

2.4.1.3 Adatbázis 34

3 Felhasznált eszközök 34

3.1 Felhasznált hardver eszközök 34

3.2 Felhasznált szoftver eszközök 35

4 Elért eredmények 36

4.1 Modbus protokoll implementálása 37

4.2 Áramkör megvalósítása 40

4.2.1 Bevezetés 40

4.2.2 Felépítése 40

4.2.2.1 Mikrokontroller 40

4.2.2.2 Reset áramkör 42

4.2.2.3 Hőmérséklet mérő áramkör 42

4.2.2.4 Soros port illesztő áramkör 43

4.2.2.5 Külső EEPROM memória áramkör 43

4.2.2.6 RS485 meghajtó áramkör 43

4.2.2.7 Tápegység 44

4.2.2.8 Feszültség mérő áramkör 44

4.2.2.9 Áram mérő áramkör 44

4.2.2.10 Programozó port 45

4.2.2.11 Galvanikusan leválasztott kétállapotú bemenet 45

4.2.2.12 Galvanikusan leválasztott impulzus bemenet 45

4.2.2.13 Open kollektoros kétállapotú kimenet.....46

4.2.2.14 Ethernet csatlakozó áramkör.....46

4.2.2.15 WiFi modul illesztő port 47

4.2.2.16 Memória kártya illesztő port 47

4.2.2.17 Diagnosztikai modul.....47

4.3 Vezérlő szoftverének megvalósítása 51

4.4 Memória kártya implementálása 52

4.4.1 FAT16 és FAT32 összehasonlítása.....52

4.4.2 MMD fájl rendszer 52

4.5 Adatbázis rendszer megvalósítása 58

4.6 WiFi implementálása 62

4.8 Feszültség mérés megoldása 65

4.9 Weblap tárolása 66

4.10 Eszköz használata 66

5 Összefoglaló 68

5.1 Mérési eredmények 68

5.2 Továbbfejlesztési lehetőségek 71

Rövidítésjegyzék 72

Irodalomjegyzék 73

Abstract

During when I was working on my thesis my main goal was to get familiar with the industrial measuring and data collecting systems and get to know more about of the micro controllers. My plan was to make a combine of the micro controllers and the very important TCP/IP network systems.

In my thesis I provide a thorough overview of the industrial measure systems and those technologies what are involved in these kind of systems. The following parts contain the documentation of the technical work I have done: in the firs chapter I provide a short overview of the task. In the second chapter I provide a deep overview of waterworks system of Erd and It`s technology, and about of the TCP/IP protocol family, mainly focus on the IPv4 and IPv6 protocols. I will mention about the operating of the Modbus protocol and the database systems. In the third chapter I will provide an overview of what hardware and software components I have used during the working progress of the task. In chapter number four I will provide an overview of results. The last chapter provides a short summary about the results, experiences and possible future developments.

Áttekintés

A szakdolgozat elkészítése során a célom az volt, hogy betekintést nyerjek az iparban használatos mérés-adatgyűjtő, folyamat irányító technológiák világába. Továbbá szerettem volna megismerkedni a mikrokontrollerekkel és a bennük rejlő lehetőségekkel. Munkám során tanulmányoztam a mikrokontrollereknek a napjainkban fontossá vált TCP/IP hálózatban történő felhasználását és ezen technológiák kombinálását egy rendszer kifejlesztése során.

A szakdolgozatom áttekintést ad az iparban használatos mérés-adat gyűjtő technológiákról és a következő módon épül fel: Az első fejezetben röviden összefoglalom a feladatot. A második fejezetben részletes betekintést nyújtok az Érdi vízműrendszerben alkalmazott mérés-adatgyűjtő technológiáról és bővebben kifejtem a TCP/IP protokoll-család tulajdonságait, az IPv4 és IPv6 szabványokra koncentrálva. A Modbusz protokoll működését bővebben kifejtem, majd az adatbázis rendszerekbe nyújtok betekintést. A harmadik fejezetben egy rövid összefoglalót adok a feladat megoldása során felhasznált hardver, illetve szoftver eszközökről. A negyedik fejezetben az elért eredményeket fejtem ki bővebben, a Modbus protokoll implementálását és a megvalósított funkciókat mutatom be részletesen, majd a mérő áramkör megtervezését és működését írom le. Ezt követi a mikrokontrollert vezérlő szoftver implementálása és működésének az összefoglalója, az adatbázis rendszer implementálása, felépítéses és működése, WiFi kapcsolat implementálása, memóriakártya olvasó implementálása, mérőkörök implementálásának az összefoglalója. Végül az utolsó fejezet összefoglalja az elért eredményeket, tapasztalatokat és a lehetséges továbbfejlesztési irányokat.

1 Bevezetés

Napjainkban a technológia ugrásszerűen fejlődik, Moor törvénye szerint, az integrált áramkörök összetettsége körülbelül 18 havonta megduplázódik. Egyre komplexebb, fejlettebb integrált áramkörökkel ellátott eszközök látnak napvilágot. Ezek alól az mikrokontrollerek sem kivételek, melyek egyre inkább alkalmasak komplexebb feladatok megoldására is. Ez inspirált, amikor úgy döntöttem, hogy belefogok és mélyebben megismerkedek a mikrokontrollerek világával. Manapság a kommunikációban kimagasló szerepet játszik az Internet, amely a 21. századra globális méretűvé, a hétköznapi emberek, illetve a gazdasági élet, az ipar kommunikációját is napi szinten kielégítő létfontosságú hálózattá vált. A legelterjedtebben használt kommunikációs protokoll az IPv4. Sok lehetőség rejlik az olyan számítástechnikai eszközök területén, melyek kihasználjak a TCP/IP hálózati protokoll család által nyújtott szolgáltatásokat és a mikrokontrollerek kompaktságát az ipari felhasználás során.

Logikus döntésnek bizonyult a két technológia ötvözése, ez arra sarkalt, hogy elkészítsek egy olyan eszközt, amely ötvözi a mikrokontrollerek és a TCP/IP hálózatok rugalmasságát.

Egy olyan üzemben, ahol valamilyen termeket állítanak elő és ez a folyamat automatizálva van, akkor kézenfekvő egy olyan eszközt alkalmazni, ami egyesíti a TCP/IP hálózat és a mikrokontrollerek sokrétű felhasználhatóságát. A gyártás során lehetne olyan mérés adatgyűjtő berendezést használni, amely azáltal, hogy kulcsfontosságú méréseket végez, kiszűri a lehetséges hibákat és így javítja a termék minőségét. Erre kiválóan alkalmas lenne egy olcsó, alacsony energia fogyasztású flexibilisen alkalmazható mérőberendezés, amely a már kiépített TCP/IP hálózathoz csatlakozna, így olcsóbbá és gyorsabbá tenné a kiépítést.

Egy betongyárban, ami akár lehet mobilis is, kézenfekvő lenne használni egy olyan mérőberendezést, amely méri a beton hőmérsékletét, nedvesség tartalmát, mert ezek az adatok fontosak a megfelelő minőségű beton gyártásához. Amennyiben elektromos árammal korlátozottan ellátott környezetben működő mobilis üzemről beszélünk, hasznos lehet kis energia fogyasztású kompakt mérőberendezés használata, mely TCP/IP hálózaton keresztül kommunikál a vezérlő berendezéssel, tehát az energia ellátása megoltható UTP kábelen keresztül (Power over Ethernet), vagy akár egy kis teljesítményű akkumulátorról is. Használhatunk a kommunikációhoz Wireless adatátvitelt, amennyiben a kábeles kiépítés nehezen

megoldható. A lehetőségek tárháza szinte kimeríthetetlen, a legfontosabb szempontok a rugalmasság, a kompaktság és az energiatakarékosság.

Feladat rövid ismertetése

Az általam választott feladat témája a Flexibilisen alkalmazható ipari mérés adatgyűjtő rendszerek kialakítása TCP/IP hálózat felhasználásával.

Feladatom, hogy megvalósítsak egy mérés adatgyűjtő berendezést mikrokontroller felhasználásával, továbbá, hogy ez az eszköz képes legyen a TCP/IP hálózaton kommunikálni.

Előzményképpen az önálló laboratórium keretein belül kezdtem el ezzel a témával foglalkozni. Végeredményként sikerült egy mérőberendezést megvalósítani. Amil az eszköz által mért eredményét TCP/IP hálózat és a mikrokontrolleren implementált web szerver segítségével megjelenítette egy Web-es felületen. Az eszköz képes volt hőmérséklet, áram(0-20 mA, 4-20 mA) és feszültség(0-5V, 0-10 V) mérésre. Az összes mérés feszültség mérésre vezethető vissza, melyet a mikrokontroller A/D átalakítójával mértem meg. Ezeket a valós idejű méréseket egy Web-es felületen jelenítettem meg.

A továbbiakban feladataim a következők:
Irodalom kutatás:

- Az iparban már meglévő mérési megoldások, technológiák, protokollok tanulmányozása, a nemzetközi irodalom és esettanulmányok felhasználásával.
- A TCP/IP, valamint a mérőeszközök kommunikációs protokoll felépítésének tanulmányozása, különös tekintettel arra, hogy mely elemek használhatók egy mérésadat-gyűjtő rendszerben.
- A vezeték nélküli hálózati kapcsolat technológiáit tanulmányozása.
- Megvalósítandó mérési módszerek tanulmányozása az egyes mérőkörökön.
- A memóriakártyára való rögzítés tanulmányozása.
- Az SQL adatbázis tulajdonságait tanulmányozása.

Megvalósítás:

- A mérő eszközökkel való kommunikációs protokoll meghatározása.
- Mérés adatgyűjtő áramkör megtervezése.
- A mérés-adatgyűjtő áramkört vezérlő szoftverének tervezése, implementálása.

• A memóriakártyára való rögzítés implementálása.

• Az adatok adatbázisba való rögzítéséért felelős rutinok implementálása..

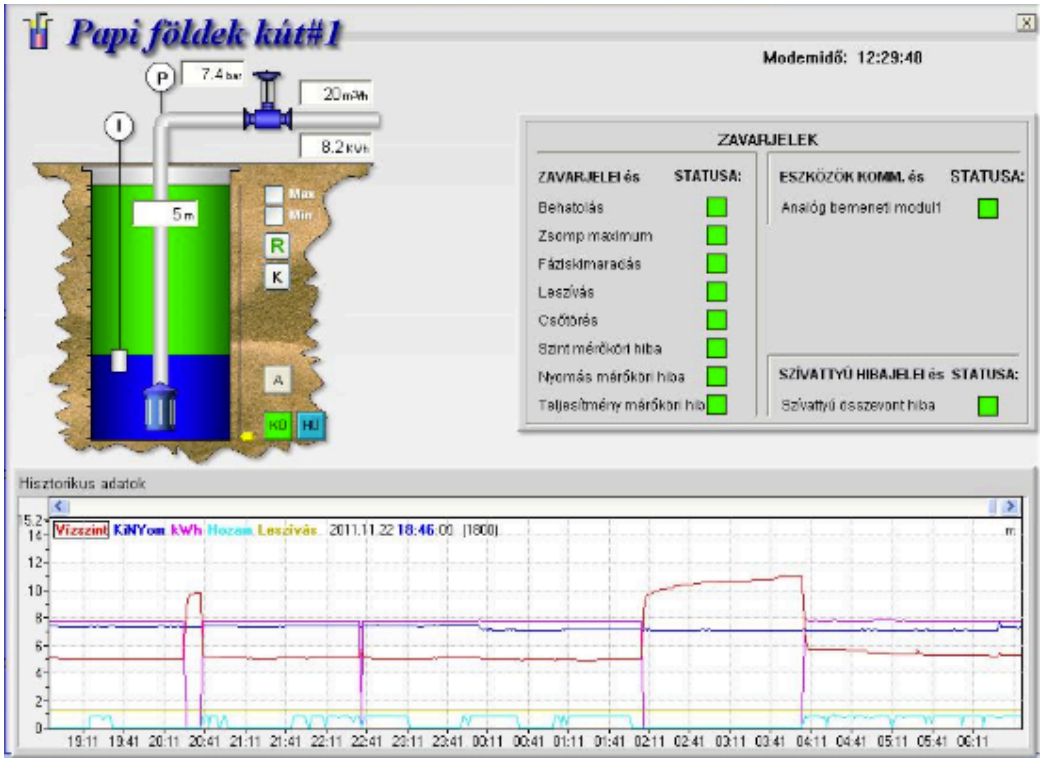
• Tesztelése a rendszer működésének és statisztikák készítése a mérési eredményekről.

2 Irodalomkutatás

2.1 Ipar

Az alábbiakban az Érdi Vízművek folyamatirányító és vezérlő rendszerébe szeretnék betekintést nyújtani, azon belül a Papi Földek ivóvíz termelő kútját mutatom be.

Papi Földek: Ivóvíz termelő kút, mely a lakosságot látja el ivóvízzel.



1. ábra: Papi földek első kútjának séma rajza

Az ábrán látható:

• Szivattyú:

Ez egy animáció, melyet egy 2 állapotú bemeneti jel vezérel (A szivattyú üzem állapotát mutatja.).

• Vízszint mérés:

Ez egy analóg jel, mely egy távadótól érkezik, méréshatára 4-20mA (az offset és a linearitás segítségével lehet paraméterezni). Mennyiségben (m^3) kifejezi a kút vízszintjét.

•Kimenő nyomás:

Ez egy analóg jel, mely egy nyomás távadótól érkezik, méréshatára 4-20mA (az offset és a linearitás segítségével lehet paraméterezni). Mennyiségben (bar) fejezi ki a kimenő nyomás értékét.

•Pillanatnyi termelt mennyiség:

Ez egy származtatott érték, ami a mennyiségmérőtől származik, impulzus jelek alapján számolja ki a hozamot (adott az impulzus/liter, a két impulzus közötti eltelt idő alapján határozza meg az átfolyó mennyiséget).

•Az állomás pillanatnyi energia fogyasztása:

Analóg jel, ami egy távadótól érkezik, méréshatára 4-20mA (az offset és a linearitás segítségével lehet paraméterezni). Mennyiségben (kWh) fejezi ki az állomás pillanatnyi energia fogyasztását.

•Zavarjelek mezőben látható elemek (2 állapotú, galvanikusan leválasztott bemenetek):

• Behatolás jelzése:

Két állapotú jel, ami jelzi, hogy történt-e illetéktelen behatolás az állomás területére.

• Zomp maximum jelzése:

Két állapotú jel, ami jelzi, hogy elöntötte-e a víz az állomás területét.

• Fázis kimaradás jelzése:

Két állapotú jel, ami jelzi, hogy megszűnt-e az állomás energia ellátása.

• Leszívás:

Két állapotú jel, ami jelzi, hogy a kút vízszintje a kritikus határ alá csökkent.

• Csőtörés:

Származtatott érték, ami a kimenő nyomás kritikus szint alá való leeséséből adódik.

• Szint mérőköri hiba:

Származtatott érték, ha 4mA-nál kevesebb, vagy 20mA-nál több áram folyik a mérőkörben, akkor generálódik ez a hiba.

• Nyomás mérőköri hiba:

Származtatott érték, ha 4mA-nál kevesebb, vagy 20mA-nál több áram folyik a mérőkörben, akkor generálódik ez a hiba.

• Teljesítmény mérőköri hiba:

Származtatott érték, ha 4mA-nál kevesebb, vagy 20mA-nál több áram folyik a mérőkörben, akkor generálódik ez a hiba.

• Analóg bemeneti modul státusza:

Származtatott érték,a Modbus-os mérőberendezések kommunikációjának az állapotát jelzi.

• Szivattyú összevont hiba:

Származtatott érték, mely a zavarjelekből tevődik össze.

• Grafikon:

A grafikon a következő jeleket ábrázolja az idő függvényében: vízszint, kimenő nyomás, kimenő teljesítmény, kimenő hozam, leszívás. Sok szempontból hasznos a grafikon, például egy pillantással át lehet tekinteni, hogy megfelelően üzemel az állomás.

Valójában hasonló állomásokból épül fel Érd és térsége vízműrendszer, a legtöbb állomáson ugyanezeket a technológiákat alkalmazzák.

A következőkben egy rövid áttekintést nyújtok az Internet protokollok kialakulásáról.

2.2 Internet Protokoll kialakulása

A Internet Protokollok kialakulásához hosszú út vezettet, aminek a megértéséhez elengedhetetlen megismernünk a kialakulásának történetét. Az amerikai védelmi minisztérium megbízásából jött létre az a társaság(ARPA), akik meghatározták, hogy a csomagkapcsolt számítógép hálózatnak úgy kell felépülnie, hogy alhálózatokat és hostokat tartalmazzon, ezzel a hálózat megbízhatóságát biztosítva. A kezdetekben a hálózat ARPANET néven egyetemek között működött. A hálózat kifejlesztésében részt vettek a Stanford Research Institute (SRI), University of California, Los Angeles (UCLA), University of California, Santa Barbara (UCSB), és az University of Utah (UTAH) egyetemeket. [1] A hálózati protokoll legfontosabb feladata az volt, hogy biztosítsa a hostok egyediségét és azonosíthatóságát a hálózatban. Sok kisebb hálózat összekapcsolásával történ az Internet megszületése, mely során az azonosíthatóság kulcsszerepet játszott végig. [2] Az első dokumentumot ami az Internet Protokollokat hivatott leírni, 1980-ban alkották meg, ez volt az első RFC. [3] Ebben a dokumentumban olyan fontos fogalmakat definiáltak mint a datagram, fregmentálás, flow control. További fontos tartalma, hogy a datagrammok hostok közötti információcserét valósítják meg és önálló entitások. Továbbá nincs lehetőség nagy adatblokkok fregmentására, nincs megoldást az adatok megbízható átvitelére [4].

A szolgáltatások a következő funkciók segítségével valósíthatók meg, Options, Type of Service, Time to Live és Header Checksum.
A hostok címe a következőképpen van definiálva: 32 bit hosszú cím, 4 részre tagolódik, az első a hálózat címe, a maradék a hostok címe. [5]
Nem telt el sok idő míg ezt a módszert leváltották, egy év múlva megjelent a ma is használatos verzió, melyben a címzés szerkezetén változtattak. 1981-ben az IETF publikálta az RFC 791-es szabványt, ami a mai legelterjedtebb protokoll, az IPv4-et írja le. [6]

2.2.1 IPv4

Az RFC 760-ban ajánlott módszer a hostok címzésére nagyságrendileg 200 alhálózat megkülönböztetését tette lehetővé. 1971-ben a hálózat (ARPANET) 15 alhálózatból állt, 1972 évek közebén pedig 34 alhálózatból, ami annak köszönhető, hogy az egyetemi munkatársak hamar felismerték a számítógépek távoli elérésében rejlő előnyöket, ez a lehetőség nemcsak a kutatással foglalkozó szakemberek kiváltsága volt, hanem egyéb szakterületen dolgozók is igényt tartottak rá. [2] A hálózat növekedése arra késztette az ARPANET fejlesztımérnökeit, hogy továbbfejlesszék a címzési rendszert.

Az új RFC 791 szabván a következő újításokat tartalmazza:

- Csomagkapcsolt kommunikációt folytató összekapcsolt számítógépek számára készült.
- Fix címmel azonosított hostok közötti információ küldését valósítja meg.
- Képes a nagy adatblokkok fragmentálására.
- Nem kínál mechanizmust a megbízható átvitelre, flow control, sorrendiség megtartására.
- Minden egyes datagrammot önálló egységeként kezel, függetlennek a többitől.
- A következő funkciók segítségével valósítja meg szolgáltatásait: Type of Service,Time to Live, Options, és Header Checksum.

A két alapfunkciója a címzés és a fregmentáció. Az internet modulok az internet header-ben hordozott címeket használják a datagrammok továbbítására a cím felé a lehetséges útvonalak közüli választási mechanizmus neve útválasztás. A hálózaton kommunikáló modulok közös elvek alapján végzik az Internet Protokoll működésével kapcsolatos funkciójukat.

A Type of Service az alkalmazni kívánt szolgáltatást jellemző egység, paraméterek egy csoportja. A Time To Live érték a csomagra jellemző egyedi szám, amelyet a küldő állít elő. Minden egyes hálózati csomópont eggyel csökkenti az értékét továbbküldéskor, ha nem éri el a célját, mielőtt nulla lenne az értéke, megsemmisítik, ezzel megelőzve a hálózatokban a hurok kialakulását.

Az Options mező lehetséges módokat tartalmaz az időbélyegekre, biztonságra, és speciális routingra vonatkozóan. A Header Checksum lehetőséget kínál arra, hogy ellenőrizzük, a csomag helyesen továbbítódott-e.

Az esetleges hibákat az ICMP protokoll segítségével deríthetjük fel részletesebben, ami a hierarchiában az IP mellett helyezkedik el. A kommunikáció modellje a következőkben képzelhető el: a küldő alkalmazás előkészíti a küldendő adatokat és a kapcsolat paramétereit és továbbítja őket a IM-nek.
Az IM elkészíti a datagrammot a meghatározott paramétereket a fejlécbe illesztve, cím az átjáró. A helyi hálózat interfésze elvégzi az alsóbb rétegbeli csomagolási feladatokat. A datagramm az átjárót ilyen módon becsomagolva éri el, ahol az kicsomagolja, majd a célcím mezőből meghatározza, hová kell továbbítania.
A cél hálózatban ugyanezek a folyamatok játszódnak le. A lényeges újítás, ami 30 éve meghatározza az internetes kommunikációt, az 1981-ben bevezetett címzési eljárás. A cél az volt, hogy a címek rugalmas kiosztásával lehetővé váljon különböző méretű hálózatok kialakítása, a következő változatokkal: kevés hálózat-sok hostal vagy közepes számú hálózat-közepes számú hostal vagy sok hálózati cím-kevés hostal. [6]

2.2.2 Az Internet növekedése

Az 1980-as évek közepén az amerikai egyetemek részéről hatalmas igény érkezett a csomagkapcsolt hálózattal történő kommunikációra, információ megosztásra. Sajnálatos módon a DoD által kifejlesztett hálózaton erre nem volt lehetőség biztonsági okokból.
A probléma megoldására létrejött egy az ARPANET-től független csomagkapcsolt hálózat, melyet az NSF épített ki. Ez a hálózat az ARPANET mintájára jött létre, 56 kb/s-os bérelt vonalak segítségével valósították meg. Ez a hálózat már a kezdetektől TCP/IP alapú kommunikációt valósított meg

Az újonnan megalakult helyi hálózatok egyre nagyobb sebességű gerinchálózatokkal kapcsolódtak össze, ennek a folyamatnak a biztosítása kezdetben állami feladat volt, de idővel nem csak egyetemek és múzeumok érdeklődtek a hálózati információ csere iránt, hanem kereskedelmi szervezetek is részt szerettek volna venni benne, illetve más országokban is megkezdték a hálózatok kiépítését az NSFNET mintájára.

A csomópontok számának növekedése exponenciális jellegű volt, ez hamar ahhoz az elkerülhetetlen problémához vezetett, hogy az RFC791-ben definiált rugalmas címkiosztás ellenére is az IP címek száma véges. [2] Megoldást nyújt a problémára a NAT módszere (RFC 1631). Hosszú és rövid távú megoldási lehetőséget, rövid távon a CIDR módszert javasolja, hosszú távon pedig egy új IP verzió kidolgozását, amelyben hosszabb címek vannak. 1994-ben, amikor ezt az RFC-t kiadták, lehetségesnek tartották, hogy a CIDR nem lesz képes biztosítani a megfelelő számú címet, amíg nem sikerül bevezetni az új címzési módszert, ezért egyéb megoldást is javasol, ez a NAT. A megoldás lényege, hogy egy címfordító egységet helyezünk a hálózat határára, amely egy táblázatot tart nyilván a hálózatban használatos helyi IP címekről és a hozzájuk társított globálisan egyedi címekről. A hálózatban használt címek tehát nem jelentenek globális egyediséget, azokat egyéb domaine-ben újra felhasználják. Az erre a célra használható címek a következőek: 10.0.0.0/8, 172.16.0.0/12, 192.168.0.0/24.

A rendszer kliens-szerver modell, a kliens gépek a magán hálózaton találhatóak, a szerver gép a nyilvános hálózaton, ottani címmel rendelkezik. A NAPT megoldásban a szervergép rendelkezik nyilvános IP címmel, a mögötte található gépek forgalmát a TCP/UDP port számaihoz hozzárendelt módon bonyolítja. A CIDR a skálázási problémára nyújt megoldást. [7] A CIDR azon az elven alapszik, hogy az eddig ki nem osztott címeket változó méretű blokkokban osszák ki:

• 194.0.0.0 – 195.255.255.255 Európához
• 198.0.0.0 – 199.255.255.255 Észak-Amerikához
• 200.0.0.0 – 201.255.255.255 Dél és Közép Amerikához
• 202.0.0.0 – 203.255.255.255 Ázsiához és Óceániához tartozzanak. [2]

A skálázási probléma a következőket jelenti:

• A B osztályú címtartomány kimerülése. A probléma az, hogy a legtöbb szervezet számára a B osztályú címtartomány túl nagy, a C pedig túl kicsi.
• A routerek akkori kapacitását kezdte meghaladni a feladat, hogy az egyre növekvő méretű routing táblákat kezeljék.
• A 32 bites címtartomány kimerülése. [8]

A NAT módszere azonban számos hátránnyal jár, a NAT szerver által módosított üzeneteken ugyanis nem minden típusú forgalom képes átjutni. AZ RTP/RTCP üzenetformái, amelyek a multimédia alkalmazások eszközei, dinamikusan foglalják le a szükséges portokat. A Kerberos autentikáció a forrás címet megköveteli, a NAT-olt forgalom esetében azonban ez a mező az üzenetformában a NAT szerver címe. [7] Az IPv6 megoldás képes e hátrányos megoldás kiküszöbölésére.

2.2.3 IPv6

Az IPv6 kifejlesztésének egyik fő oka a címzés területén jelentkező problémák kiküszöbölése. 1990-ben kezdte el a fejlesztést az IETF az új verzión. A legfontosabb cél annak a biztosítása, hogy a címek ne jelentsenek többé véges erőforrást. A routing táblák mértének csökkentése egy hierarchikus címkiosztási módszerrel. Jobb biztonsági mechanizmusok beépítése, multicast, unicast megoldások támogatása, illetve az, hogy az új és a régi protokoll egymás mellett működhessenek az átmeneti időszakban. [2]

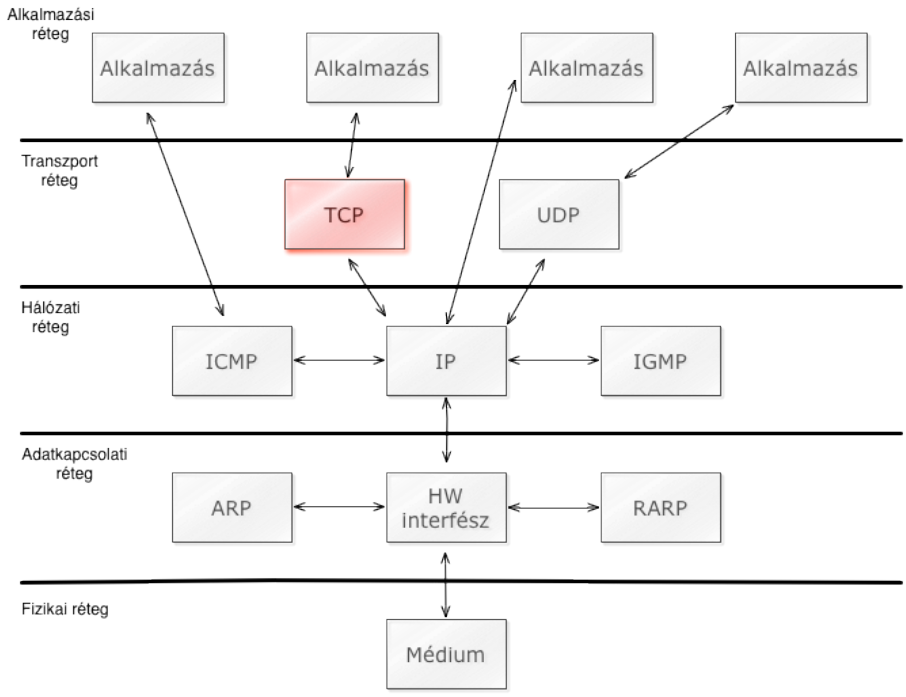
Az IPv6-ban hosszabb címeket definiáltak, 128 bites címmezőt (így 2128 db címet biztosítva), hogy többszintű hierarchiát képezhessenek a címek, több csomópont kaphasson címet, illetve egyszerűsödjön a címek automatikus kiosztása. Egy új típusú címzési lehetőséget is definiálnak az RFC2460-ban, az anycast típusú címzést (megtartva tehát a multicast és az unicast lehetőséget), az ilyen módon címzett csomagot a csomópontok egy csoportjából bármely csomópont megkaphatja. A broadcast üzenettípust azonban túl nagy erőforrás igényesnek találták, így a multicast címzést részesítették előnyben.

A fejrész egyszerűsödése az jelenti, hogy elhagytak bizonyos mezőket (pl.: protocol, type of service), másokat opcionálissá tettek. A biztonság területén nagy előrelépés, hogy az IPv6 támogatja az autentikációt és az adat integritást.[9]

A routing protokollok tekintetében nem szükséges lényeges változtatásokat tennünk, az IPv4-es protokollokat ugyanaz a felépítés, gyakran ugyanaz az implementáció jellemzi, sajnos ugyanazok a hibák is. [9] Az IPv6 bevezetésének ezek tehát a kiváltó okai, azonban van negatív oldala is, mégpedig az, hogy az időközben kialakult technológiák előnyeit így elveszítenénk. Konkrétan egy NAT-olt hálózatot, ha a hálózaton kívülről közelítünk meg, lehetetlen felderíteni, hogy a hálózatban pontosan hány gép van, milyen címen, illetve a hálózat szerkezetét. Az IPv6 bevezetésével azonban nincs szükség NAT-ra, ami azt jelenti, hogy egy strukturált hálózat szerkezetét, a gépek számát a hálózaton kívülről teljes egészében felderíthetőek.

2.2.4 TCP/IP protokoll felépítése

A TCP/IP protokollt csomagkapcsolt hálózatok adatátviteli protokolljára hozták létre. Ez egy öt réteget tartalmazó hálózati protokoll rendszer, szemben az ISO OSI hét rétegű modelljével.



2. ábra: TCP/IP modell

Physical: Fizikai réteg, lehet többek között réz, üveg, telefon,rádió. [10]

Data-Link: Ez a szint az OSI modell fizikai és adatkapcsolati rétegéhez áll közel, feladata a kapott byte-ok átvitele. A fölötte levő rétegek csak azt várják el tőle, hogy bájt folyamokat tudjon fogadni, illetve átadni. [10]

Network: Ez a réteg felelős az adatok átviteléért a hálózaton, a gépek közötti kapcsolat összeköttetés mentes és nem tudja biztosítani a réteg az adatok korrekt átvitelét sem. Ezen a szinten több protokoll is megtalálható, de ezek közül a legfontosabbak: IP(Internet Protocoll), ARP(Adress Resolution Protocol), ICMP (Internet Control Message Protocol). [10]

Internet Protokol: Ez a protokoll gondoskodik a csomagok átviteléről a hálózaton, az összes kommunikáció a host-ok között IP csomagok formájában történik. Ez egy kapcsolat nélküli protokoll, a kommunikációhoz nem szükséges előzetes kapcsolat felvétel. Adatátvitel szempontjából nem megbízható, a csomagokkal bármi megtörténhet: elveszhetnek, megsérülhetnek, sorrendjük összekeveredhet. [10]

Transport: ez lehet TCP vagy UDP. [10]

TCP: Felfelé megbízható adatátvitelt biztosít úgy, hogy alatta egy megbízhatatlan protokoll található. Felülről bájt folyammal lehet táplálni (stream orientált), full duplex átvitelt biztosít, kapcsolat orientált. Kliens- szerver kapcsolatok kialakítására képes, bárki lehet egyszerre kliens és szerver is. A kapcsolat felvétele mindkét oldalon nyugtázva történik. A kezdeményező küld egy kérés csomagot, a címzett válaszol a kérésre, majd a kezdeményező küld egy nyugtát arról, hogy vette a címzett válaszát. A kapcsolatok azonosítására szolgálnak a portok, a szerver egy adott porton érkező kérésekre figyel, és az ott érkező kéréseket kiszolgálja. A TCP portok közül az első 1024 foglalt a standard alkalmazások számára. A megbízhatóság érdekében a protokoll a teljes TCP csomagra számol ellenőrző összeget. A megbízható kapcsolat kialakítására pozitív nyugtázást használ. A küldő oldal nyugtát vár minden egyes elküldött TCP csomagról. Amennyiben egy később küldött csomagról előbb kap nyugtát, mint egy korábban küldött csomagról, akkor a korábban küldött csomagtól kezdve megismétli az adást. Egy TCP csomag szintén két részből áll egy header-ből, és az áthiendő adatokból. [10]

UDP: Ez sokkal gyorsabb protokoll, mint a TCP protokoll, viszont nem megbízható adatátvitel szempontjából. Nem kapcsolat orientált, nincs hibajavítás, nincs nyugtázás. Tulajdonképpen az IP szint által biztosított szolgáltatásokat nyújtja felfelé. Akkor szokták használni, ha az adatátvitel sebessége a legfontosabb, minden többi feladatot a felette elhelyezkedő réteg lát el. Tipikusan a DNS-ek (Domain Name Server), real-time alkalmazások, játékok szokták használni (egy játékban vagy real-time hang átvitel esetén ha egy csomag rossz akkor ott legfeljebb döccen egyet, de ez még mindig kisebb baj, mintha az adott pontnál megállna és onnantól elkezdené újra adni a csomagokat). A szegényesebb szolgáltatásból adódóan sokkal egyszerűbb az UDP header. [10]

Application: Ezen a szinten helyezkednek el az alkalmazások. Az adatok átvitelét TCP, vagy UDP porton keresztül történő hívásokkal valósítják meg. A portok tulajdonképpen a kommunikációs csatorna egy végpontjának az azonosítására szolgálnak a szabvány 65535 TCP és 65535 UDP portot engedélyez, ezek közül az első 1024 foglalt szabványosított protokollok számára (pl.: Telnet, SMTP, POP3, Rlogin, FTP SNMP, STMP, HTTP). Az összeköttetés kliens szerver alapon valósul meg. Az egyes szerver alkalmazások az adott host gép operációs rendszerénél bejegyeztetik magukat, hogy egy adott TCP, vagy UDP portra érkező kérések kiszolgálásáért az adott alkalmazás a felelős. Az egyes kliens alkalmazások, amikor megszólítják a szerver gép adott partját akkor az ott futó operációs rendszer értesíti a szerver alkalmazást arról, hogy kérés érkezett az adott partra. Azaz amikor egy szolgálat azonosítására van szükség a hálózaton akkor nem elég a szerver gép IP címét megadni, hanem szükséges az adott szolgáltatáshoz kapcsolódóan megadni a szerver adott TCP, vagy UDP partját. [10]

A következőkben egy rövid áttekintőt szeretnék nyújtani az Modbus protokoll kialakulásáról.

2.3 Modbus protokoll

2.3.1 Bevezetés

Sok, az irányítástechnikában is használt ipari készülék a Modbus protokollt alkalmazza RS-232/485 vagy Ethernet közegen.

Modicon cég a hetvenes években az ipar számára fejlesztette ki a nyílt forráskódú Modbust protokollt, PLC-k kommunikációjához.

Főbb specifikációi:

- Működési elv: master/slave.
- A kapcsolat kezdeményezését a master végzi, aki egy kérést vagy egy parancsot küldi a slave-nek.
- Topológia: pont-pont, busz multipont(vonali lezárásokkal).
- Átviteli sebesség: 4800/9600/19200 bit/sec.
- Maximális távolság : 1200m.
- Formátum: 1 start bit + 8 adatbit + paritás + 1 vagy két stop bit.
- Maximális csomag méret: 246 adat byte.
- Maximum 255 cím adható ki egy Modbus hálózaton(1byte).
- Átvitel biztonsága: CRC vagy RLC.

A soros Modbusz kapcsolatnak két típusú adat átviteli módja van, ASCII és RTU. Ez a két mód az Modbus üzenetek kódolására utal. ASCII kódolás esetében az üzenetek ASCII formátumban vannak, míg RTU esetében bináris kódolást használ, ami meggyorsítja az adatátvitelt.

Modbus/RS485/RS232

A Modbus protokoll elterjedését a nyílt forráskód és az egyszerű kialakítása tette lehetővé. A magas megbízhatóság miatt kiválóan alkalmas ipari mérés-adatgyűjtő rendszerek közötti kommunikációra. Az RS485 kommunikációs interfész fizikai tulajdonságai miatt inkább kisebb távolságok áthidalására képes. Ennek a hálózattípusnak a legnagyobb előnye az egyszerű és olcsó kivitelezhetőség. Kommunikációs médium a sodort érpár, ún. 2-vezetékes vagy 4-vezetékes módon.

Főbb specifikációi:

- Elméletben maximum 255 cím adható ki egy Modbus hálózaton, de a valóságban a RS485-ös kommunikációs interfész fizikai tulajdonságai miatt csak 32 eszköz megcímzésére van lehetőség.
- Topológia: pont-pont, busz multipont.

• Átviteli sebesség: 4800/9600/19200 Bit/sec.

Modbus/TCP

Napjainkban egyre jobban elterjed az Ethernet hálózat az ipari alkalmazások területén is. Egyre több PLC, ipari berendezés támogatja az Ethernet feletti Modbus/ TCP protokollt. A megvalósítást költségét sok esetben csökkenti a már kiépített Ethernet hálózatba való integrálhatóság.

A Modbus/TCP alapvetően egy Modbus keretet ágyaz be egy TCP keretbe. Ez egy kapcsolat-orientált tranzakció, amely azt jelenti, hogy minden egyes kérés választ vár. A nyílt forráskódú Modbusz alkalmazása TCP csomagban ágyazva széleskörű megoldást kínál egyaránt kisebb és nagyobb méretű hálózatok kialakítására.

2.3.2 Modbus protokoll felépítése

Bizonyos részei a Modbus protokollnak rögzítettek, mint például az üzenet keret formátuma, kommunikációs hibák kezelése.

Egyéb részei a protokollnak viszont felhasználó igényei alapján módosíthatók, mint például a médium kiválasztása, baud rate, karakter paritása, stop bit-ek száma, adatátviteli mód(ASCII, RTU). Ezeket a paramétereket minden egyes állomásnál be kell állítani és a rendszer működése közben nem lehet változtatni rajtuk.

Ahhoz hogy a hardver üzenetet tudjon küldeni a vonalon a címzettnek, az üzenetet egy borítékba kell csomagolni. Ez a boríték elhagyja a hardvert egy port-on keresztül és a médiumon halad végig a cél eszközhöz. Ebben az esetben a protokoll biztosítja a borítékot, amit egy kerettel(frame) reprezentál, ez a keret több komponensből áll , mint például: címzett, parancs típusa, adat és hiba ellenőrzés is.

Amikor az üzenet elérkezik a cél slave eszköz interfészét, akkor mint a küldés esetén egy port-on keresztül jut be a cél eszközbe. Ekkor az slave eszköz eltávolítja a borítékot és elolvassa az üzenetet és ha nem detektált hibát, akkor elindítja a válasz metódusát. Ekkor kicseréli az üzenetet a borítékban és vissza a feladónak. A vissza küldött csomag üzenet tartalmazza a küldő slave címét, az elvégzett utasítás kódját, adatot ami a parancs végett generálódott és hibaellenőrzést. Abban az esetben nem keletkezik válasz, ha a kiküldött üzenet broadcast(üzenet az összes slave-nek) típusú volt. Ezt a 0-ás cím indikálja.

A legtöbb esetben a master azonnal tud újabb üzenetet küldeni akármelyik slave-nak, ha kapott valós választ, illetve ha eltelt egy bizonyos idő amit a felhasználó adhat meg a válaszra várva(time out).

Az kiküldött üzenetek lehet lekérdezések, mely slave válaszokat generál. Illetve csak egyetlen esetben küldhető olyan üzenet, melyre nem érkezik igazoló válasz és ez nem más mint a broadcast üzenet. Ez azon esetben alkalmazható, ha például időt szeretnénk szinkronizálni egyszerre az összes slave-en. [11]

3. ábra: Modbus Master-Slave kérés/válasz ciklus

2.3.2.1 Átviteli módok

Characteristic	ASCII	RTU
Coding System	hexadecimal (users ASCII printable caharacters: 0-9, A-F)	8 bit binary
Number of bit per character:		
start bits	1	1
data bits	7	8
parity	1 (1 bit send for even or odd parity, no bits for no parity)	1 (1 bit send for even or odd parity, no bits for no parity)
stop bit	1 or 2	1 or 2
Error checking	LRC	CRC

4. ábra: Modbus átviteli módok összehasonlítása

Az ASCII nyomtatható karaktereket könnyű értelmezni, így hiba keresésnél kiválóan alkalmazható.

RTU módban az adat 8 bit-es bináris ként van elküldve. ASCII módban minden egyes RTU karakter ketté van osztva két 4 bit-es részre és a hexadecimális megfelelőjükkel van reprezentálva és ezek a hexadecimális értékek alkotják az elküldendő üzenetet. Az ASCII mód kétszer annyi karaktert használ fel mint az RTU , de a dekódolása és a kezelése sokkal könnyebb mint az RTU mód esetén. RTU mód esetén az üzenetet alkotó karakterek egy folyamot alkotnak. ASCII mód esetén akár 1 sec-es szünetek is lehetnek két karakter küldése között, így lehetőség van lassabb master eszköz használatára. [11]

2.3.2.2 Hiba detektálás

Két típusú hiba következhet be egy üzenet küldése során. Az egyik az átviteli hiba, a másik a működési hiba. A Modbus rendszer lehetővé teszi mind a két hiba detektálását.

A kommunikációs hibát gyakran leragadt bitek váltják ki, például az eredeti üzenet 0001 0100 helyett 0001 0110 érkezne meg az adat vonalon, ami a decimális 20-t 22-re változtatná meg. Sokkal valószínűtlenebb az az eshetőség amikor egy plusz bit adódik az üzenethez vagy elvész egy bit az üzenetből.

Ezen hibák leggyakoribb kiváltó oka a zaj, a nem kívánatos elektromos jelek a kommunikációs csatornán. A kommunikációs hibákat több módon is detektálni lehet, karakter framing, paritás ellenőrzés, redundancy check.

Amikor az ellenőrző metódusok detektálnak egy hibát, akkor az üzenet feldolgozása megáll azonnal és eldobódik az üzenet. (Ugyan ez történik ha nem létező című slave-tol jön egy üzenet).

Ha egy sérült üzenet érkezik, akkor az üzenet tartalma megbízhatatlan, így a fogadó slave nem lehet biztos egyáltalán abban sem, hogy neki címezték az üzenetet. Kommunikációs hibára utalhat, ha a master által küldött üzenetre nem érkezik válasz az általa megcímzett slave-től egy bizonyos időn belül. Ez az idő intervallum függ a baud rate-től, üzenet típusától, a slave feldolgozási idejétől. Amint meghatározásra kerül ez az idő intervallum, be lehet programozni a master az üzenet újra küldésére, ha nem érkezik válasz ezen időn belül.

Mind a két átviteli mód, RTU és ASCII, magába foglal egy opcionális paritás bit-et. RTU módban, 9 bit-es az adat mező(8 bit az adat és 1 bit a paritás bit). ASCII módban 8 biz az adat mező(7 bit az adat és egy a paritás bit). Ha a paritás bit nincs használva, akkor nem küldődik le. A paritás bit használata opcionális a Modbusz rendszerben. Installálásnál a slave eszközöknél páros, páratlan vagy kikapcsolható a paritás bit. Ami viszont fontos, hogy az összes eszközt ugyan úgy kell konfigurálni.

A paritás bit 1 bit hibát képes detektálni. A következő képen határozza meg a Modbus rendszer, hogy a paritás bit páros vagy páratlan legyen:

• Összeadja az 1-esek számát az adatban.

• Meghatározza, hogy ez az összeg páros-e vagy páratlan.

Páros paritás esetén:

• Ha a szám páros, akkor hozzá ad egy nullát, mint paritás bit, hogy az egyesek számát párosan tartja.

• Ha a szám páratlan, akkor hozzá ad egy egyest, mint paritás bit, így az adatban szereplő egyesek száma páros lesz.

• Például: Páros paritás esetén a 0110 1000 adatnál 1 lenne a paritás bit értéke, míg a 0110 1010 adatnál 0 lenne a paritás bit értéke.

Páratlan paritás esetén:

• A master hozzáad egy egyet vagy nullát, mint paritás bit, úgy hogy az egyesek száma az adatban páratlan legyen.

Ha két hiba volt az üzenetben, akkor sajnos nem minden esetben tudja a paritás észlelni a hibát. Például: 0010 0000 adatból 0010 0011 lett, mind a két esetben az adatban szereplő egyesek száma páratlan. Látható, hogy ebben az esetben nem lehet megállapítani a hibát.

A Modbusz protokoll több különböző szintű hiba detektálási módszert is kínál, hogy kellő minőségű adatátvitelt biztosítson. Abban az esetben, amikor hiba esetén a paritás nem változik, a redundancia ellenőrzés nyújthat megoldást ez lehet CRC vagy LRC. Hogy melyik redundancia ellenőrzés használatos, azt az átviteli mód határozza meg. Az RTU a CRC-t, míg az ASCII az LRC-t használja. [11]

2.3.2.3 CRC Hiba ellenörző eljárás

Az üzenet(csak az adat bitek, figyelmen kívül hagyva a start/stop és az opcionális paritás bitet) melyre úgy tekintünk mint egy bináris számra(tehát az adatbitek sorozata képez egy bináris számot), ahol az MSB bit az első küldendő elem.

Az üzenet meg kell szorozni x^16-al(16 bit-el balra eltolva), ezután elosztani x^16+x^15+x^2+1-al, így egy bináris számkén(11000000000000101) kifejezve.

A kitevőt elhanyagoljuk és a 16 bit-es maradékot hozzáfűzzük az üzenethez(MSB az első),két CRC byte formájában. Tehát a kimenetünk egy üzenet ami tartalmazza a CRC-t is, ha azt ugyanazzal a polinommal(x^16+x^15+x^2+1) osztjuk el, akkor a maradék nulla lesz, ha nem történ hiba az átvitel során(Az üzenetet fogadó slave újra kiszámítja a CRC-t és összehasonlítja a fogadottal).

CRC generálás lépései:

1. 16 bit-es regiszter feltöltése csupa egyessel.

2. A bejövő adat alsó 8 bit-ét XOR-olom a 16 bit-es regiszter felső 8 bit-évél , az eredmény rögzítése a 16 bit-es regiszterbe.

3. A 16 bit-es regiszter eltolása egy bit-el jobbra.

4.1 Ha a kitolódott bit az egy(átvitel keletkezik), akkor XOR-om a 16 bit-es regisztert a polinommal(1010 000 000 0001).

4.2 Ha a kitolódott bit az nulla, akkor visszatérünk a 3. fázishoz.

5. Ismételjük a 3-as és 4-es lépést nyolcszor.

6. XOR-oljuk a soron következő adat 8 bit-ét a 16 bit-es regiszterrel.

7. Ismételjük 3-6-ig a lépéseket, amíg az üzenet összes byte-ja XOR-olva nem lesz a 16 bit-es regiszterrel.

8. A 16 bit-es regisztert tartalma lesz a 2 byte-os CRC. [11]

2.3.2.4 LRC Hiba ellenörző eljárás

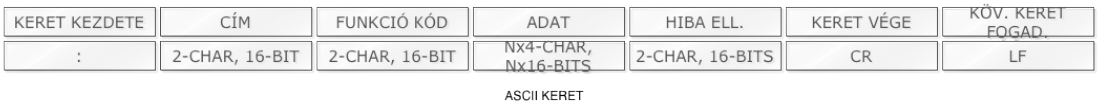
Az ASCII átvitel hiba ellenőrző eljárása az LRC. A hiba ellenőrzés egy 8 bit-es bináris szám formájában van reprezentálva, és két hexadecimális ASCII karakter formájában kerül átküldésre. A hiba ellenőrző eljárás során átkonvertáljuk a hexadecimális karaktereket bináris formátumba. Összeadjuk a bináris karaktereket carry átvitel nélkül, majd az eredmény kettes komplemensét vesszük. A fogadó oldalon újra

kiszámítjuk az LRC-t és össze hasonlítjuk a fogadott LRC-vel. A kettőspont, CR, LF, és egyéb beágyazott nem ASCII hexadecimális karakter figyelmen kívül kell hagyni az LRC generálásakor. [11]

2.3.2.5 ASCII keret

Ahogy a bevezetőben is olvastuk a protokoll kommunikációs szabályok összessége master és slave eszközök között. A végfelhasználó eszközök számára transzparens a kommunikáció. A fő különbség a két átviteli mód hiba ellenőrző eljárásában van. ASCII esetében LRC, míg az RTU esetében CRC a hiba ellenőrző eljárás.

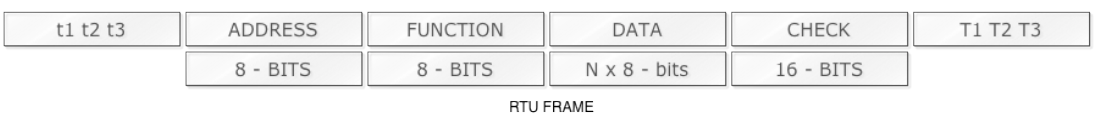
ASCII üzenet kerete kettősponttal kezdődik és a CR(carriage return) és a LF(line feed) karakterrel zárul. A LF karakter szinkronizálási szerepet is betölt, azt jelzi, hogy az átviteli fázis készen áll fogadni a válasz üzenetet. [11]



5. ábra: Modbus ASCII keret

2.3.2.6 Remot Terminal Unit (RTU) keret

RTU átviteli mód esetén a keretek szinkronizációját a keretek folyamatos egymás utáni küldésével lehet biztosítani. A fogadó eszköz figyeli a fogadott karakterek közötti eltelt időt. Ha három és fél karakter idő eltelt fogadott karakter nélkül vagy keretlezáró karakter nélkül, akkor a fogadó eszköz elküldi a válasz üzenetet és arra számít, hogy a következő fogadott byte egy cím lesz. [11]



6. ábra: Modbus RTU keret

2.3.2.7 Cím mező

Az adat mező rögtön az üzenet keret legelején áll, RTU esetén 8 bit, ASCII esetén 2 karakter. A cél slave eszköz címét reprezentálja, akinek az üzenet címezve van.

Minden egyes slave eszköznek egyedi címmel kell rendelkezzen és csak a megcímzett eszköz válaszolhat a kiküldött üzenetre. Amikor egy slave eszköz küld egy válasz üzenetet, melyben szerepel a saját címe, így a fogadó master eszköz be tudja azonosítani, hogy kivel kommunikál. Broadcast üzenet esetén a nullás cím van beállítva, minden egyes slave a rendszerben megkapja az üzenetet és feldolgozza azt, semmilyen esetben nem érkezik válasz rá. [11]

2.3.2.8 Funkció mező

A funkció kódok arra szolgálnak, hogy a megcímzett slave eszköznek megmondja milyen utasítást hajtson végre.

1. Kód: 01(read coil status)
    - kimenet státuszának leolvasása
2. Kód: 02(read input status)
    - diszkrét bemenő értékek jelenlegi értékének kiolvasása
3. Kód: 03(read holding register)
    - regiszterek blokkos olvasása
4. Kód: 04(read input registers)
    - bemenő regiszter jelenlegi értékének kiolvasása
5. Kód: 05(force single coil)
    - a kimenetek szelektív átlátása

1. Kód: 06(present single register)
    - adott érték beírása adott regiszterbe
2. Kód: 07(read exception status)
    - 8 db. kimenet státuszának olvasása
3. Kód: 08(loopback diagnostic test)

•Diagnosztikai teszt, olyan üzenet, mellyel le lehet tesztelni a kommunikációt a master és a slave között.

4. Kód: 16(preset multiple registers)

•regiszterek blokkos írása [11]

2.3.2.9 Adat mező

Olyan adatot tartalmaz, amelyre a slave eszközöknek van szüksége, a kívánt funkció végrehajtásához vagy azokat az adatokat tartalmazza, amelyeket a slave gyűjtött és válaszul küldte vissza a master eszköznek. Ezek az információk lehetnek értékek, cím referenciák, határértékek. Például arra utasítja a slavet, hogy olvassa ki az egyik regiszter tartalmát, ebben az esetben arra használatos az adat mező, hogy megmondja melyik regisztertől kezdje el a kiolvasást és mennyi adatot olvasson ki. [11]

2.3.2.10 Hiba ellenőrző mező

Ez a mező lehetővé teszi mind a master, mind a slave számára az üzenet ellenőrzését a tranzakció során. Esetenként előfordulhat, hogy az üzenet megsérülhet a tranzakció során a médiumon fellépő elektromos zajok miatt, egyéb interferenciák miatt. A hiba ellenőrző metódusok lehetővé teszik a master, illetve a slave számára a hibák felismerését, így nem reagálnak a hibás üzenetekre, egyszerűen eldobják azt. Ezzel is javítva a biztonságosságát, effektivitását a Modbus rendszernek.

ASCII átvitel esetén LRC hiba ellenőrző eljárást alkalmaz, míg RTU átvitel esetén CRC hibaellenőrző eljárást. [11]

2.3.2.11 Megvalósított funkció parancsok

Feladatom során két funkció kódot valósítottam meg, a Read holdong registers(code: 03) és a Preset multiple registers(code: 16). Ezekről bővebben az alábbiakban térnék ki.

2.3.2.11.1 Read holding registers(code: 03)

Kimeneti regiszterek tartalmának kiolvasására ad lehetőséget ez a parancs. Tehát a felhasználó a megcímzett slave kimeneti regisztereinek értékét tudja kiolvasni.

Lehetőség van maximum 125 regiszter egyidejű megcímzésére minden egyes kérésben. Viszont a címzett slave eszköznek lehetnek korlátai a maximálisan megengedett regiszterek egyidejű megcímzésére.

Ezen parancs esetében a broadcast mód nem használható.

CÍM	FUNKC.	KEZDŐ ADAT REG HO	KEZSŐ ADAT REG LO	ADAT REG SZÁM HO	ADAT REG SZÁM LO	HIBA ELL.	
11	03	00	6B	00	03	7E	CRC

READ OUTPUT REGISTER üzenet

7. ábra: 03-as funkció kódú Modbus üzenet

Az alábbi példa bemutatja, hogy épül fel az üzenet keret. Regiszter olvasás a 40108-40110 regiszterekből, az 584-es című slave eszköztől.

Válasz

A megcímzett slave válaszol, a válasz üzenet tartalmazni fogja a slave címét, funkció kódot és az információ mezőt. Az információs mező tartalmazni fog 2 byte adatot, ami a visszaküldött információ mennyiségét tartalmazza. Továbbiakban az információ mező tartalmazni fogja a kért adatot 2 byte-os blokkokban bináris formában, minden pár egy karaktert reprezentál, az első a magas helyi értékű biteket, míg a másodig az alacsony helyi értékű biteket tartalmazza.

Az alábbi példában a 40108-40110 regiszterek decimális értéke 555, 0, 100. [11]

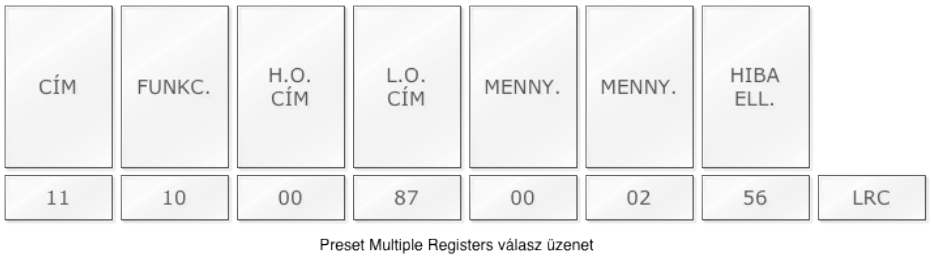
CÍM	FUNKC.	BYTE SZÁM	ADAT KIM. REG H.O. 40108	ADAT KIM. REG L.O. 40108	ADAT KIM. REG H.O. 40109	ADAT KIM. REG L.O. 40108	ADAT KIM. REG H.O. 40110	ADAT KIM. REG L.O. 40110	HIBA ELL.
11	03	06	02	2B	00	00	00	64	55

Read Output Register Response üzenet

8. ábra: 03-as funkció kódú Modbus üzenetre a válasz

2.3.2.11.2 Preset multiple registers(code: 16)

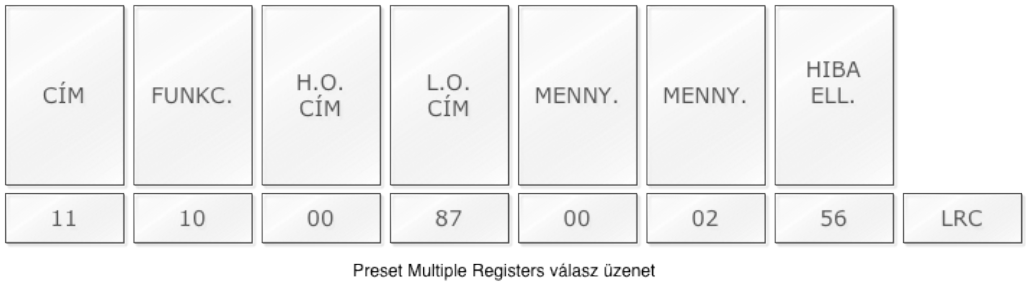
Ezzel a paranccsal tulajdonképpen a cél slave eszköz regisztereit tudjuk írni (maximum 60 regisztert). Nullás címzés esetén az összes slave eszköz megkapja ezt az üzenetet és a tartalmának megfelelően módosítják az adott regiszterek tartalmát.



9. ábra: 16-os funkció kódú Modbus üzenet

Válasz

Egy normál válasz a 16-os funkció kódot tartalmazó üzenetre, a következőket tartalmazza: kezdeményező fél címe, funkció kód, kezdő cím és a regiszterek mennyisége. [11]



10. ábra: 16-os funkció kódú Modbus üzenetre a válasz

A következőkben egy rövid áttekintőt szeretnék nyújtani az adatbázis rendszerekről.

2.4 Adatbázis rendszerek

Napjainkban egyre nagyobb szerepet játszanak az adatbázis rendszerek, szerves részt vesznek ki az információ áramlásban, mivel a fő feladatuk az adatok tárolása.

Több adatbázis rendszert is megvizsgáltam: MySql, SQLite, PostgreSQL. Ezek közül igyekeztem kiválasztani feladat számára a legideálisabbat. Az alábbiakban egy rövid áttekintőt szeretnék nyújtani ezen adatbázisok tulajdonságairól.

MySql

A MySQL egy gyors, több szálas, több felhasználós SQL adatbázis-szerver, ami open source. A Unix, OS/2 platformok alatt ingyenesen használható, míg Windows platform alatt meg kel vásárolni a szoftvert. A jelenlegi verziója a 3.21.29a. A MySQL tábla lock-olást használ, ami azt jelenti, hogy ha az adott táblán valamilyen tranzakciót hajtanak végre akkor az egész táblára egy lock kerül, tehát más tranzakció számára nem lesz elérhető az adott tábla, míg fel nem oldódik róla a lock. A tábla lock-olás technika biztosítja, hogy a tábla adatai konzisztensek maradjanak, de ugyanakkor más táblákon párhuzamosan folyhat tranzakció, így az adatbázis műveletek közepes sebességűek.[12]

SQLite

Egy nyílt forráskódú adatbázis motor, ellentétben a legtöbb adatbázis szoftverrel az SQLite nem használ szerver folyamatot, direktben írja/olvassa az lemez fájlokat. A teljes adatbázis(táblák, triggerek) egy fájlban tárolódnak a lemezen. Ezek a fájlok szabadon hordózhatók a platformok között, legyen az 32bit vagy 64bit-es rendszer.
Adatbázis szintű lock-olást használ, ami azt jelenti, hogy az adatbázis tranzakciók lassúak.[13]

PostgreSQL

A PostgresSQL egy objektum relációs adatbázis kezelő rendszer(ORDBMS), ami a Postgres alapjaira épül, a kaliforniai Berkeley egyetemen fejlesztettek ki.
A PostgreSQL az eredeti Berkeley open source leszármazottja. Támogatja a legtöbb standard SQL utasítást, és rengetek új lehetőséget biztosít, például: komplex lekérdezések, idegen kulcsok használata, triggerek, view-k, tranzakciós integrálás, multiversion konkurencia szabályozás
Továbbá, PostgreSQL kibővíthető a felhasználó által sok új kiegészítővel, mint például: adminpack(rengeteg extra szolgáltatást, távoli management), isn, lo, stb.

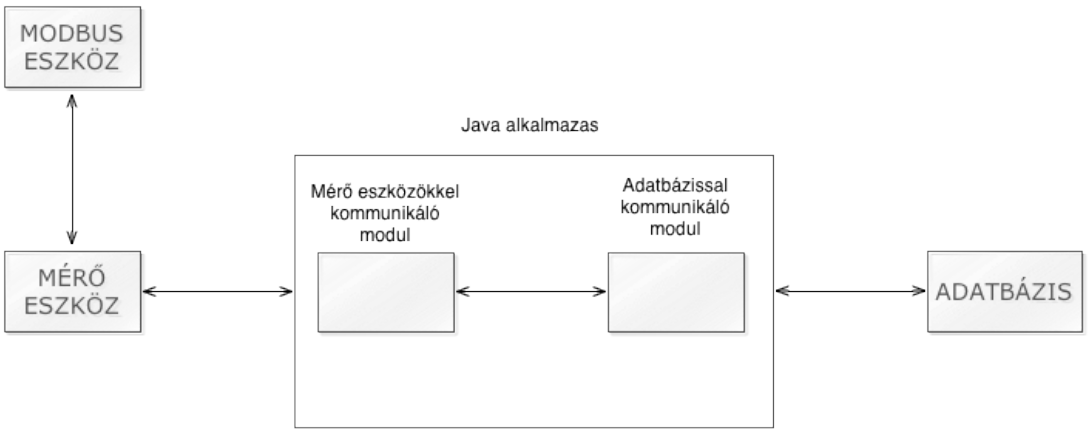
Sor szintű lock-olás. Ez a leggyorsabb mind a három módszer közül, hiszen ebben az esetben nem az egész adatbázist, illetve tábla kerül lock-olás alá hanem csak az adott sor az adott táblában. Ez azért előnyös mert párhuzamosan több tranzakció is tudja használni a táblát. Az én alkalmazásom esetében, tegyük fel, ha több mérő berendezés szeretne adatot küldeni az adatbázis ugyanazon táblájában, akkor ezek a műveletek párhuzamosan is végre tudnak hajtódni, míg a másik két adatbázis esetében ezeknek a tranzakcióknak meg kell várniuk amíg a másik végez és feloldja a lock-olás, tehát sokkal gyorsabb a művelet végzés a PostgreSQL esetében.

A választásom a PostgreSQL-re esett éppen azon tulajdonságából adódóan, hogy sor szintű lock-olást használ, ugyanis az általam megvalósítani kívánt rendszer számára ez a tulajdonság jelenti a gyorsaságot. Adott esetben egyszerre több mérőberendezés is tud adatot beírni az adatbázisba, ami ugrásszerűen meggyorsítja a feldolgozást.[14]

2.4.1 Adatbázis rendszer felépítése

Az általam felépített rendszer megvalósítást szeretném bemutatni a következőkben. A logikai felépítésre térnék csak ki, majd részeltessen a későbbiekben beszélnék róla.

A rendszer 3 elemből épül fel: mérő berendezés, Java applikáció, adatbázis.



11. ábra: Adatbázis rendszer felépítése

2.4.1.1 Mérő berendezés

Ez az eszköz végzi a mérést és az adat gyűjtést, majd bizonyos indközönként az adatokat rögzíti az adatbázisban. Ez a folyamat úgy épül fel, hogy az eszköz elvégzi a mérést-adatgyűjtést, abból generál egy üzenetet egy egyedi protokollnak megfelelően, nyit egy TCP socket-et a Java-s applikációval és elküldi az üzenetet, majd a Java-applikáció feldolgozza azt.

2.4.1.2 Java applikáció

Ez az applikáció a mérőberendezés és a adatbázis szerver között helyezkedik el. Tehát ez az applikáció a közvetítő modul a mérőberendezés és az adatbázis között. A mérő berendezéstől begyűjti az adatokat TCP socket-en keresztül, egy egyedi protokollt felhasználva, majd feldolgozza az üzenetet és elküldi a megfelelő parancsot az adatbázisnak, hogy rögzítésre kerüljön az adat.
Továbbá azért van szükség erre a köztes applikációnak a beiktatására, hogy az adatbázis szerver ne legyen kapcsolatban direktben az internettel, ez egy biztonsági megfontolás.

2.4.1.3 Adatbázis

Direktben nem kapcsolódik az internethez biztonsági okokból, közvetlen kapcsolata csak a Java-s applikációval van.

3 Felhasznált eszközök

3.1 Felhasznált hardver eszközök

A fejlesztés során egy Microchip PICDEM.Net 2 [15] fejlesztő eszközt használtam és egy PicKit3 [16] típusú programozót. MPLAB Ide fejlesztő környezettel dolgoztam és a 5.31-as verziójú Microchip Ip Stack-et [17] használtam a munkán során.

A fejlesztő eszköz kiválasztása során két lehetséges jelöltre szűkítettem le a kört. Az egyik rendszert a CCS C míg a másodikat a Microchip gyártja. Azért esett a választásom erre a Microchip fejlesztő eszközére, mert hosszas utánajárást követően, arra jutottam, hogy ennek az eszköznek a legjobb a támogatottsága.

Megfelelő dokumentáció elérhető, számos példa alkalmazás illusztrálja, hogy milyen feladatokat lehet megvalósítani vele.

Ez az eszköz a Microchip által nyújtott egyik hálózati feljeleztő környezete. Többek között azért is esett a választásom erre a developper board-ra, mert két RJ45-ös hálózati ajzat is található rajta, az egyik vezérlését a lapon dedikált PIC 18F97J60-as [18] mikrokontroller végzi, melyben egy beépíttet Ethernet vezérlő található. A másik csatlakozó vezérléséért egy külső Ethernet vezérlő felelős az ENC28J60[19] vezérlő. Azért előnyös számomra ez a felépítés, mert rálátást nyerhetek arra hogy, hogyan lehet használni egy külső Ethernet vezérlőt egy Microchip PIC mikrokontrollerrel összehangolva, és tanulmányozhatom a belső Ethernet vezérlő működését is.

A fejlesztő eszközhöz sok kiegészítő érhető el. Amit a feladat megvalósítása során felhasználtam az a MRF24WB0MA WiFi modul [20], ami könnyen csatlakoztatható a fejlesztő eszköz PicTail Plus portjához, így kézenfekvő volt a választás. A másik kiegészítő a mit felhasználtam az a AC164122 PicTail Board for SD & MMC Cards[21]. Ez nem más mint egy a PicTail portra illeszkedő SD és MMC memóriakártya olvasó.

A fejlesztés ezekkel az eszközökkel volt a legkézenfekvőbb kivitelezni, mert nem kell hardver elemeket legyártani, így leegyszerűsítette a folyamatot. Majd amikor már kialakul a végső megoldás a feladatra sokkal könnyebb volt megtervezni az áramkört, hiszen ekkora már a pontos specifikációja, megoldása a feladatnak már körvonalazódott.

3.2 Felhasznált szoftver eszközök

A mérőberendezés vezérlő szoftverének a kifejlesztését a Microchip MPLAB IDE fejlesztő környezetben végeztem el a Microchip által nyújtott C18-as C fordító [22] segítségével. Továbbá a Microchip TCP/IP Stack keretrendszer segítségével valósítottam meg a mikrokontroller hálózati kommunikációjához szükséges rutinokat.

Az adatbázist illesztő Java alkalmazást az Eclipse IDE fejlesztő környezettel valósítottam meg. Az adatbázis szerverként a PostgrSQL adatbázis rendszert használtam.

Az áramkör megtervezéséhez az OrCAD áramkör tervező alkalmazást használtam.

A fent említett szoftvereket főleg azért választottam mert díjmentesen elégetőek az OrCAD kivételével.

4 Elért eredmények

Elsőként meg kellet ismerkednem a Pic 18F97J60-as mikrokontroller tulajdonságaival, szerencsére a MICROCHIP által nyújtott dokumentáció igen alapos, majd a fejlesztő eszköz áramkörének a tanulmányozása következett.

Következő lépésként össze kellet állítanom a fejlesztő környezetet, be kellet kalibrálnom a fordító programot, IP Stacket. Már ez a része a projektnek sem volt olyan könnyű feladat, mint számítottam rá, sok kompatibilitásból fakadó problémát kellet elhárítanom, amire le tudtam fordítani az első saját alkalmazásomat és le tudtam programozni a mikrokontrollert. Ezt a problémát a Microchip hiányos dokumentációnak tudható be, és a legfrissebb kiadású PIC C18-as fordító program a PIC 18F mikrokontroller család számára hibás volt, sajnálatos módon az általam használt mikrokontrollert is érintette ez a hiba. A hiba miatt a mikrokontroller másodpercenként újraindult és ezt az állapotát nem lehetett megszakítani. Hosszas utánaolvasással megtaláltam a fordító hibás header fájl-t és ezt egy előző verziójú fordítóprogram header fájl segítségével kijavítottam. Sajnos a Microchip meg nem javította ki a hibás fordító programot. Ezek után már csak a megfelelő típusú IP stack-et kellet installálnom. Itt is sokadik próbálkozásra sikerült eredményt elérnem, több verzióval is próbálkoztam, bizonyos funkciók nem működtek megfelelően az általam módosított példa programmal.

Majd tanulmányoztam a példa alkalmazásokat. A Microchip példa alkalmazásai között ráleltem egy példa web szerver alkalmazásra. Ezt az alkalmazást tanulmányozva próbáltam megérteni a program működését. További fejlesztéseket kellet végeznem, hogy a hőmérséklet mérés algoritmusát implementálni tudjam és a web szervert fel tudjam készíteni a mérési eredmények megjelenítésére a web oldalon. Majd a potméteres feszültség mérést is implementálnom kellett, itt a mikrokontroller AD konverterének a működését kellet megvizsgálnom, természetesen a hőmérséklet mérés esetén is.

Áttekintettem a HTML nyelv szintaktikáját. El kellet készítenem egy egyszerű HTML oldalt ahol meg tudom jeleníteni a mért adatokat, tanulmányoznom kellet a HTML dinamikus változók tulajdonságait is. Ennek a részfeladatnak a megoldása során is egy példa alkalmazásból indultam ki, melyet áttanulmányozva el tudtam készíteni a saját alkalmazásomat.

Soron következett a memóriakártyára való rögzítés részfeladatának a tanulmányozása, itt is egy példa feladatból indultam ki, de voltak komplikációk.

Sajnos az általam használt mikrokontrollerre nem volt elérhető működő példa alkalmazás, ezért egy másik mikrokontrollerre szánt alkalmazást kellett átírnom. Első lépéskén egy új linker fájlt kellet készítenem amiben az alkalmazáshoz szükséges memóriaterületeket kellet lefoglalni úgy, hogy figyelembe vettem a mikrokontroller arhitetktúráját.

A következőkben a WiFi modul implementálásának a részfeladatát oldottam meg, itt is egy példa alkalmazásból indultam ki. A megfelelő beállítások alkalmazásával gyorsan működésre bírtam az eszközt.

Tulajdonképpen ezen részfeladatok megoldásával, teljesen beüzemeltem a fejlesztő eszköz minden funkcióját és nekiláthattam a továbbiakban a fő feladat megoldásának. Az alábbiakban részletesen bemutatom a folyamatot.

4.1 Modbus protokoll implementálása

A bevezetésben részletesen kifejtettem a Modbus protokoll működését, az alkalmazásban a 03(read holding register) és a 16(preset multiple registers) funkciókat valósítottam meg a protokollból. Ezen két funkció lehetővé teszi a kommunikációt bármely Modbus képes eszközzel. Ez a két funkció lehetőséget ad arra, hogy egy Modbus képes eszköz regisztereit írni, illetve olvasni tudjuk.

A Modbusz protokollt a Modbus.h, Modbus.c fájlban implementáltam. A Modbus.h header fájlban definiáltam azt a négy függvényt, ami tulajdonképpen megvalósítja a két funkciót.

Tagfüggvények:

• void DoPowerUpModMast(void);

- Inicializálja a Modbus változókat. InbA_i, InbA_o, OutbA_i, OutbA_i bemeneti és kimeneti mutatókat. Ezek a segédváltozók a Modbus üzenet össze állításánál szükségesek.

• char ModMastTx(unsigned char statId, unsigned char honnanAddr, unsigned char hovaAddr, unsigned char db);

- Ez a függvény a 16-os funkciót valósítja meg a címzett Slave eszköz regisztereinek az írását.

A függvénynek 4 bemenő paramétere van: statID(Slave címe), honnanAddr (melyik regsiztertől fogunk írni), hovaAddr(melyik regiszerbe fogunk írni), db(mennyi regisztert fognk írni).

Első lépéskén összeállítjuk a Modbus csomagot, ezt az OutbA tömbbel reprezentáljuk.

```
OutbA_i = OutbA_o = 0;
OutbA[OutbA_i++] = statId;
OutbA[OutbA_i++] = 0x10;
OutbA[OutbA_i++] = 0x00;
OutbA[OutbA_i++] = hovaAddr;
OutbA[OutbA_i++] = 0x00;
OutbA[OutbA_i++] = db;
OutbA[OutbA_i++] = db*2;
```

Majd egy ciklussal betöltjük a küldendő adatokat és legvégül betöltjük a Crc-t.

```
OutbA[OutbA_i++] = CrcL;
OutbA[OutbA_i++] = CrcH;
```

Ekkor összeállt az üzenet, amit az OutbA tömb rprezentál. Készen áll a küldésre. Megnézzük, hogy foglalt-e az USART, addig várunk amíg fel nem szabadul és elküldjük az üzenetet.

Ezek után várjuk a Slave eszköz nyugtáját. Beállítunk egy TimeOut változót, ez a változó fogja azt az időt reprezentálni, ameddig várunk a Slave válaszára, ha ezen időn belül érkezik válasz a Slave-től, akkor a Slave válaszát eltároljuk az InbA tömbbe, megvizsgáljuk hogy attól az eszközöl kaptuk-e a nyugtát, akinek elküldtük az adatot, ha az ellenőrzés helyes, akkor a tranzakció sikeres volt. Sikertelen tranzakció esete, a Master újra megpróbálja elküldeni az adatokat.

• char ModMastRx(unsigned char statId, unsigned char honnanAddr, unsigned char hovaAddr, unsigned char db);

- Ez a függvény a 03-os funkciót valósítja meg a címzett Slave eszköz regisztereinek az olvasását.

A függvénynek 4 bemenő paramétere van: statID(Slave címe), honnanAddr (melyik regisztertől fogunk olvasni), hovaAddr(melyik regiszertl fogunk olvasni), db(mennyi regisztert fog olvasni).

Első lépéskén összeállítjuk a Modbus csomagot, ezt az OutbA tömbbel reprezentáljuk.

```
OutbA_i = OutbA_o = 0;
OutbA[OutbA_i++] = statId;
OutbA[OutbA_i++] = 0x03;
OutbA[OutbA_i++] = 0x00;
OutbA[OutbA_i++] = honnanAddr;
OutbA[OutbA_i++] = 0x00;
OutbA[OutbA_i++] = db;
OutbA[OutbA_i++] = CrcL;
OutbA[OutbA_i++] = CrcH;
```

Ekkor összeállt az üzenet, amit az OutbA tömb reprezentál, készen áll a küldésre.
Megnézzük, hogy foglalt-e az USART, addig várunk amíg fel nem szabadul és elküldjük az üzenetet.

Ezek után várjuk a Slave eszköz válaszát. Beállítunk egy TimeOut változót, ez a változó fogja azt az időt reprezentálni, ameddig várunk a Slave válaszára, ha ezen időn belülérkezik válasz a Slave-től, Megvizsgáljuk hogy annyi karaktert kaptunk vissza mint amennyit kértünk, ha nem akkor eldobjuk a csomagot, különben sikeres volt a tranzakció és visszatérünk a vett darab számával és eltároljuk a Modbus regiszterekbe. Ha az adott időn belül nem érkezik válasz, akkor meghiúsult a tranzakció és újra meg fogjuk kísérelni azt.

• void CrcGen(unsigned char);

- A bevezetőben említett algoritmus alapján számolom ki a Crc-t.

4.2 Áramkör megvalósítása

4.2.1 Bevezetés

A mérés adatgyűjtő áramkör tervezését és az áramkör működését fogom bemutatni a következőkben.

4.2.2 Felépítése

Az áramkör az alábbi modulokat tartalmazza:

- Mikrokontroller
- Reset áramkör
- Hőmérséklet mérő áramkör
- Soros port illesztő áramkör
- Külső EEPROM memória áramkör
- RS485 meghajtó áramkör
- Tápegység
- Feszültség mérő áramkör
- Áram mérő áramkör
- Programozó port
- Galvanikusan leválasztott kétállapotú bemenet
- Galvanikusan leválasztott impulzus számláló bemenet
- Open collectroros kétállapotú kimenet
- Ethernet csatlakozó áramkör
- WiFi modul illesztő port
- Memória kártya illesztő port
- Diagnosztikai modul

4.2.2.1 Mikrokontroller

Azért esett a választásom a PIC18F97J60 típusú mikrokontrollerre, mert rendelkezik egy beépített Ethernet meghajtóval. Mivel az alkalmazásom lényegi része a TCP/IP hálózat felhasználása a feladat megoldásához, így kézenfekvő volt a választás.

Ez a mikrokontroller a PIC18 családhoz tartozik, melyeknek a legfontosabb tulajdonsága a magas számítási sebesség mellet az alacsony energia fogyasztás, így ezek a mikrokontrollerek kiválóan alkalmasak az általam megvalósítani kíván feladatra.

Elegendő adatmemóriával rendelkezik a működtetető szoftver számára, 96Kbyte áll rendelkezésre.

A PIC18F97J60 család elegendő adat memóriát biztosít a dinamikus alkalmazások számára, 3808 byte RAM.
Abban az esetben, ha a 96Kbyte program memória nem lenne elegendő az alkalmazás számára, akkor a PIC18F97J60 család rendelkezik egy külső memória busszal, ami lehetővé teszi a CPU program counter-ének, hogy akár 2MB memóriát is megcímezhessen.

Főbb jellemzők:

•Kommunikáció:

A PIC18F97J60 család rendelkezik többféle soros kommunikációs perifériával. Két független USART modul, két Master SSP modul, melyek képesek mind SPI, illetve I^2C módú kommunikációra. Az egyik általános célú I/O port-ot be lehet konfigurálni, mint egy 8 bites párhuzamos Maser/Slave port, ami közvetlen processzor-processzor kommunikációt tesz lehetővé.

•CCP modulok:

A processzor család tagjai mind rendelkeznek két Capture/Compare/ PWN (CCP) és három Enhanced CCP(ECCP) modulokkal, annak érdekében, hogy maximalizálják a flexibilitást olyan alkalmazásokban, ahol a folyamatok vezérlése a kulcs feladat.

• Mikrokontroller működési frekvenciája: 41.67Mhz
• Program memória: 96K.
• Adat memória: 3808 Byet.
• Megszakítási források(interrupt): 16 darab.
• I/O portok: A, B, C, D, E, F, G, H, J
• I/O lábak: 70 darab.
• Időzítők: 5 darab.
• Capture/Compare/PWN modulok: 2.
• Enhanced Capture/Compare/Pwm modulok: 3.
• Soros kommunikáció: MSSP (2), Enhanced USART (2).
• Ethernet meghajtó.
• Parallel Slave Port Communications (PSP).
• External Memory Bus.
• A/D Konverter: 10 Bites, 16 csatornás.
• Utasítás készlet: 75 utasítás, 83 ha engedélyezve van a kibővített utasítás készlet.

•A mikrokontroller két órajel generátort tartalmaz, az elsődleges a rendszer órajelét generálja, a másodlagos pedig a dátum-idő előállításához szükséges. [18]

4.2.2.2 Reset áramkör

Ez egy költség hatékony rendszer figyelő áramkör, amit arra terveztek, hogy monitorozza a tápfeszültséget a rendszerben és ha szükséges, akkor egy reset jelet adjon a mikrokontrollernek és ezáltal a mikrokontrollert alaphelyzetbe állítja.

Bekapcsoláskor a tápfeszültség a küszöb szint felé emelkedik, akkor maximum 100ms hosszú aktív-alacsony szintet generál.
A táp feszültség megszűnésekor a feszültség a küszöb szint alá esik, akkor legfeljebb egy 20us-os aktív-alacsony szintet generál.

A feladatot az U11 jelzésű TC1275 DC típusú integrált áramkörrel valósítottam meg.

Főbb jellemzői:

- tápfeszültség 3.3V
- küszöbszint 1.2V
- az áramkör a tápfeszültség tranziensekre kevésbé érzékeny [23]

4.2.2.3 Hőmérséklet mérő áramkör

U10-es jelű, TC1047 típusú integrált áramkör.
Ez egy TC1047 típusú feszültség kimenetű, hőmérséklet érzékelő áramkör. Az eszköz egy lineáris feszültség kimenetű hőmérséklet szenzor, aminél a kimeneti feszültség direktben reprezentálja a mért hőmérsékletet. A szenzor hőmérséklet érzékelő intervalluma: -40°C -+125°C. A tápfeszültség: 2.5V - 5.5V.

Főbb jellemzői:

- Tápfeszültség: 3.3V.
- Hőmérési tartomány: -40°C - +125°C. [24]

4.2.2.4 Soros port illesztő áramkör

U5 jelzésű, MAX3232 típusú integrált áramkör egy RS232 típusú meghajtó ami magában foglal egy kapacitív feszültség generátort a TIA/EIA-232-F szabványú feszültség szint 3.3V-ból való előállítására.

Főbb jellemzői:

- Tápfeszültség: 3.3V
- Műveleti sebesség: 120kbit/sec. [25]

4.2.2.5 Külső EEPROM memória áramkör

U6 jelzésű, 24LC512 típusú CMOS soros külső EEPROM memória, 64K x 8 (512 Kbit) kapacitással rendelkezik. Működéshez szükséges tápfeszültség: 1.8V - 5.5V. I^2C buszon csatlakozik a mikrokontrollerhez.

Főbb jellemzői:

- Tápfeszültség: 1.8V - 5.5V.
- Tároló kapacitás: 64K x 8 (512 Kbit).
- 128 byte méretű lap kezelés.
- I^2C busz kompatibilitás.
- Maximum 5ms-os írási ciklus.
- Hardveres írás védelem akár egész tömbökre.
- 1.000.000 írás/törlés ciklus. [26]

4.2.2.6 RS485 meghajtó áramkör

U7 jelzésű, MAX3485 integrált áramkör egy RS485 típusú meghajtó.

Főbb jellemzői:

- Tápfeszültség: 3.3V
- Műveleti sebesség: 2.5Mbps.
- Félduplex kommunikáció.

Részegységei:

- D10,11,12,13 jelzésű diódák és az R79, 80 jelzésű ellenállások az RS485-ös vonal túlfeszültség védelmét látják el.
- R65,66,69 jelzésű ellenállások, az RS485-ös vonal lezáró ellenállások. [27]

4.2.2.7 Tápegység

Feladata, hogy ellássa különböző feszültség szintű áramköröket energiával.

Részegységei:

- D1 jelzésű dióda fordított polaritás ellen védi a készüléket.
- U1, U2 jelzésű 78M05 típusú három pontos pozitív szintű feszültség stabilizátor.
- U3, U4 jelzésű TC1262-33VDB típusú 3.3V kimeneti feszültségű, nagy pontosságú alacsony maradék feszültségű(dropuot) szabályzó.

Működése:

A tápfeszültség előállítása két lépcsőben történik.

Az U1, U2 jelzésű integrált áramkör feladata, hogy az U3, U4 jelzésű integrált áramkörök számára előállítsa a +5V-os működési feszültséget.

U3, U4 jelzésű áramkörök feladata, hogy előállítsa a 3.3V-os működési feszültséget. [28], [29]

4.2.2.8 Feszültség mérő áramkör

Feladata a bemeneti tápfeszültség mérése.

Részegységei:

R3, R4 precíziós feszültség osztó és C56 szűrő kondenzátorból épül fel, mely a készülék tápfeszültség bemeneti pontján levő dióda katódjához csatlakozik.

Működése:

Az R3, R4 ellenállásokból felépített feszültség osztó úgy van beállítva, hogy 20V bemenő feszültséget 3.3V-ra osztja le. Az analóg/digitális átalakító referencia feszültsége is 3.3V, így a méréshatár 20V-os.

4.2.2.9 Áram mérő áramkör

Feladata 0-20mA vagy 4-20mA típusú távadók(nyomás távadó, teljesítmény távadó) áramának a mérése.

Részegységei:

R54, R56 mérő ellenállások. R53, C53 és a R55, C54 szűrő áramköri elemek.

Működése:

A R54, R56 ellenállások értéke úgy van megállapítva, hogy 20mA átfolyó áramnál 3.3V feszültség essen rajtuk. Az analóg/digitális átalakító referencia feszültsége is 3.3V, így a méréshatár 20mA lesz. Az RC tag a külső tranziensek csillapítására szolgál.

4.2.2.10 Programozó port

Feladata, hogy a mikrokontrollerbe le lehessen tölteni a működtető szoftvert.

Kompatibilis:

•PicKitX programozó eszközökkel.

4.2.2.11 Galvanikusan leválasztott kétállapotú bemenet

Az interfész feladata, hogy a beérkező diszkrét jeleket elválassza a belső áramkörtől, annak védelme érdekében. További feladata, hogy indikálja az adott bemenet aktív-e vagy sem(LD1,2,3,4,7,8,9,10 LED diódákkal).

Részegységei:

• LD1,2,3,4,7,8,9,10 led diódák jelzik, hogy az adott bemenet aktív-e vagy sem.

• OP1,2,3,4,6,7,8,9 optikai csatolók biztosítják a galvanikus leválasztást.

• DC1 jelzésű DC/DC átalakító biztosítja a +12V galvanikusan leválasztott feszültséget a bemeneti interfész számára.

Működése:

Az OP1,2,3,4,6,7,8,9 optikai csatolók kimenetei a mikrokontroller input bemenetére csatlakozik, így a bemeneteket a mikrokontroller ciklikusan ellenőrzi és a változásokat rögzíti.

4.2.2.12 Galvanikusan leválasztott impulzus bemenet

Feladata az impulzus kimenettel rendelkező távadók jeleinek fogadása. Például mennyiségmérők: víz, gáz.

Részegységei:

•LD9 led dióda jelzi, hogy az impulzus bemenet aktív-e vagy sem.

• OP9 optikai csatoló biztosítja a galvanikus leválasztást.

• DC1 jelzésű DC/DC átalakító biztosítja a +12V galvanikusan leválasztott feszültséget az impulzus bemeneti interfész számára.

Működése:

Az OP9 optikai csatoló kimenete a mikrokontroller interrupt bemenetére csatlakozik, így a bemenet változását a mikrokontroller érzékeli és rögzíti egy számlálóban.

4.2.2.13 Open kollektoros kétállapotú kimenet

Feladata a mikrokontrollerhez csatlakoztatott open kollektoros kimeneteken keresztül az eszközök vezérlése(ON/OFF).

Részegységei:

•Q1-Q3 jezésű ZVN3306F típusú FET tranzisztorok.

Működése:

A mikrokontroller a Q1-Q3 jelzésű FET tranzisztorok Gate-it vezérli(ON/OFF). [30]

4.2.2.14 Ethernet csatlakozó áramkör

J5 jelzésű, RJ45 típusú Ethernet csatlakozó. Ami tartalmazza a vonal illesztő transzformátort és a kommunikációs LED diódákat.

Főbb jellemzői:

• Adatátviteli sebesség: 10 MBit/sec.

• Hőmérési tartomány: -40°C - +85°C.

Részegységei:

• LD1,2 LED diódák.

• Vonal illesztő transzformátor.

• R11,12,13,14 jelzésű ellenállások valamint a C50,51 jelzésű kondenzátorok alkotják a mikrokontroller és a leválasztó transzformátor illesztését.

• R15,16 jelzésű áramköri elemek a LED előtét ellenállások.

4.2.2.15 WiFi modul illesztő port

Feladata, hogy az opcionálisan használható WiFi modult energiával ellássa és csatlakoztassa a mikrokontrollerhez.

4.2.2.16 Memória kártya illesztő port

Feladata, hogy az opcionálisan használható SD memória kártyát energiával ellássa és csatlakoztassa a mikrokontrollerhez.

4.2.2.17 Diagnosztikai modul

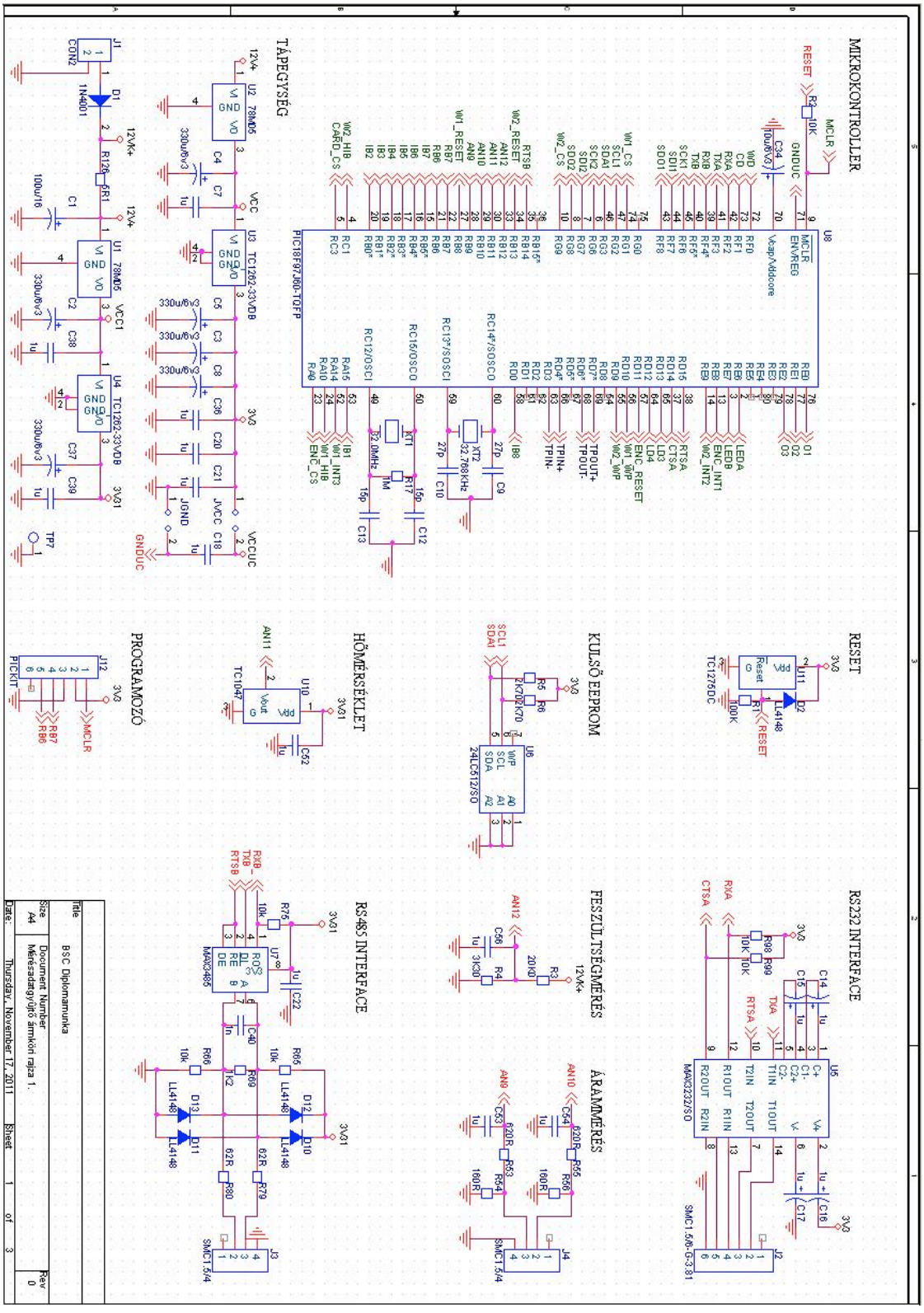
Feladata állapotok jelzése LED diódákkal.

Részegységei:

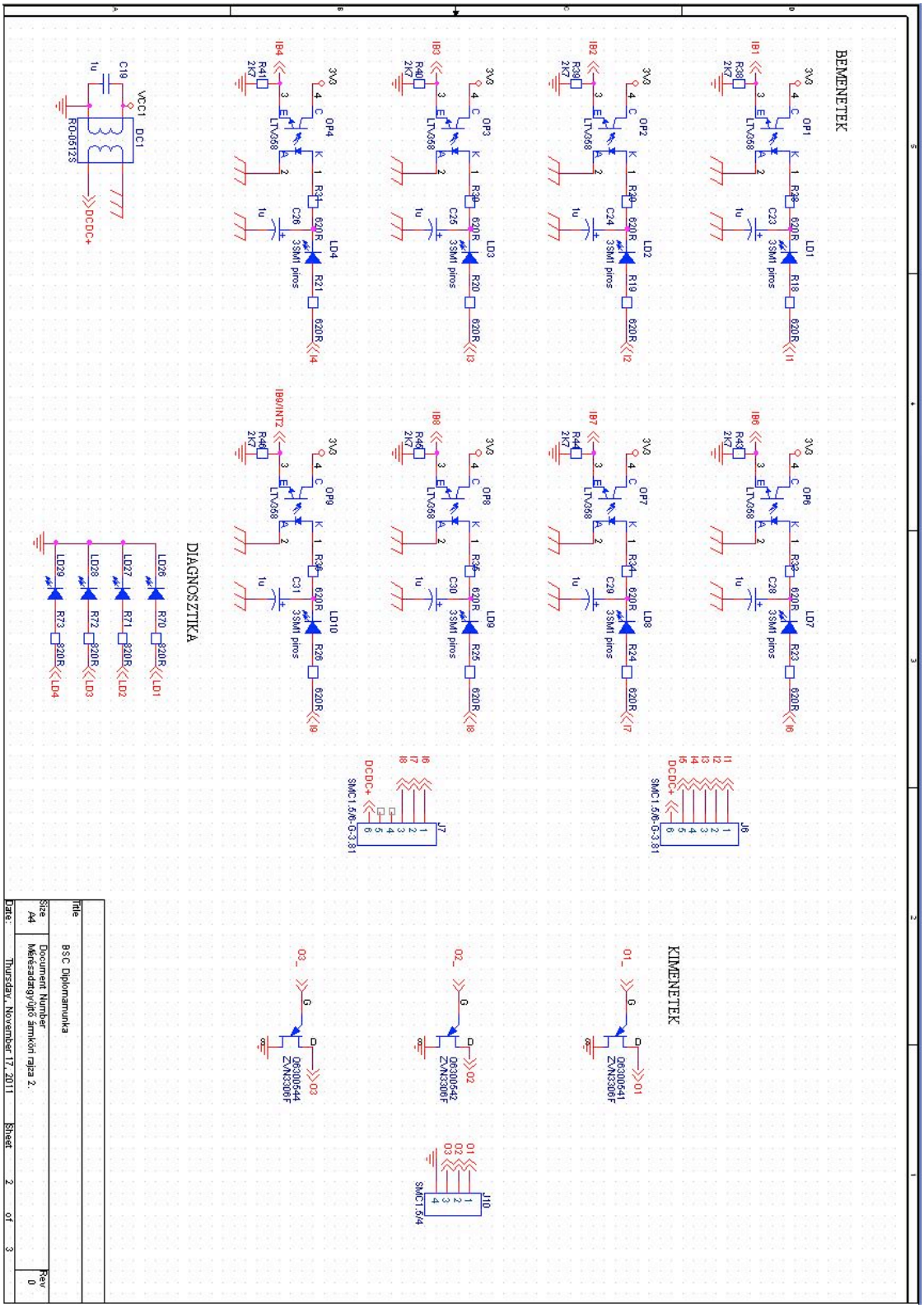
- LD26-LD29 LED diódák.

Működése:

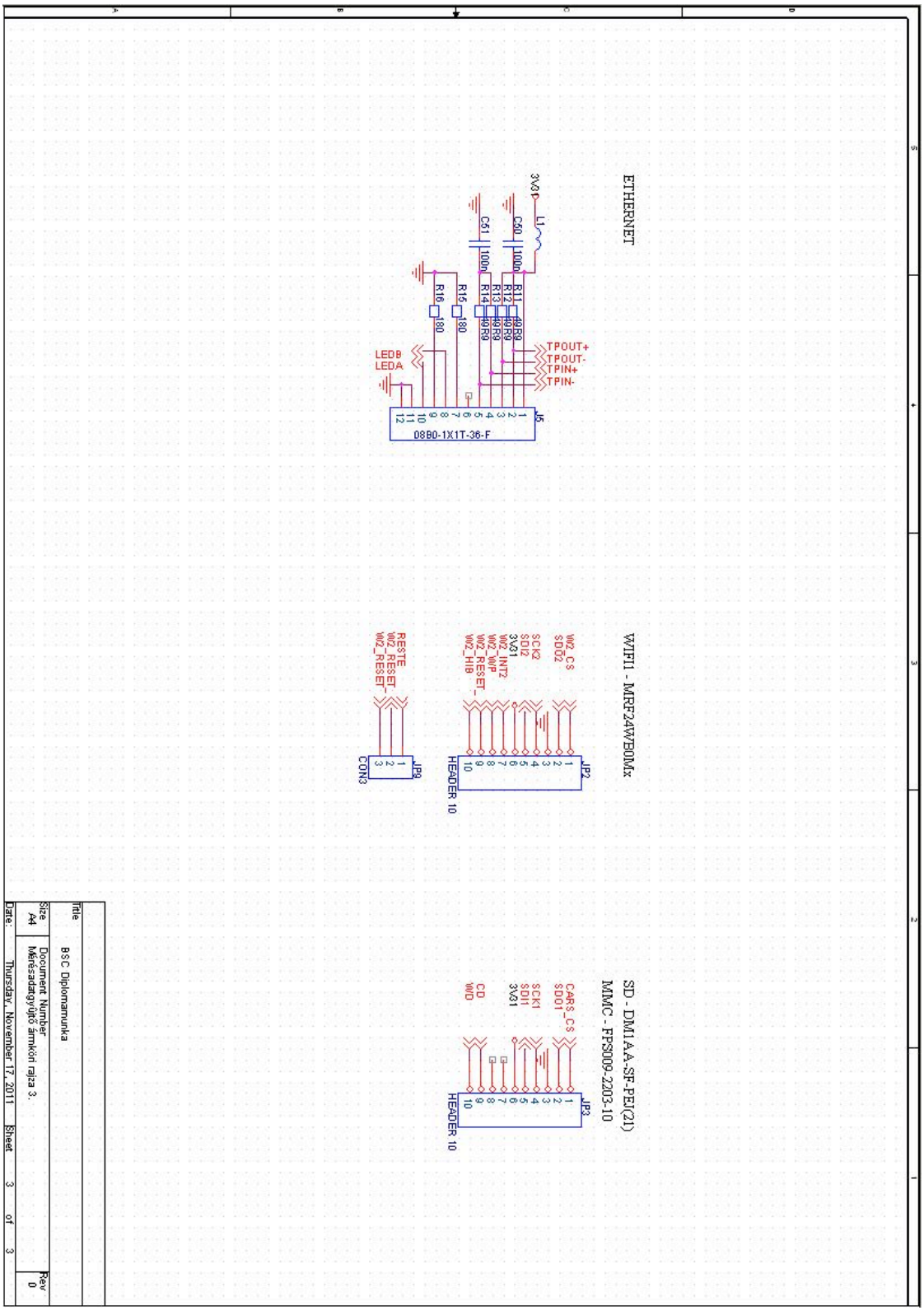
- LD26: Modbus kommunikáció jelzése. Ha HIGH-nincs és nem is volt, ha LOW-volt és most nincs, ha BLINK-folyamatban van a kommunikáció
- LD27: ugyan az ami fölül csak a rákötött eszközre nézve
- LD28: Van-e memória kártya csatlakoztatva készülékhez.
- LD29: Van-e WiFi modul csatlakoztatva a készülékhez.



12. ábra: Áramkör kapcsolási rajz első része



13. ábra: Áramkör kapcsolási rajz második része



14. ábra: Áramkör kapcsolási rajz harmadik része

4.3 Vezérlő szoftverének megvalósítása

Eben a fejezetben a mikrokontroller vezérlő szoftverének a működését fogom bemutatni.

A fejlesztés során a Microchip MPLAB IDE fejlesztő környeztetettel dolgoztam, mivel 8 bit-es mikrokontrollerrel valósítottam meg a feladatot ezért Microchip c18 fordítóprogramot használtam. Továbbiakban a Microchip TCP/IP Stack keretrendszert használtam fel a munkám során.

A főprogramnak amit a Main.c-ben implementáltam a lehető legegyszerűbbnek kellett lenni, hogy minél gyorsabban le tudjon futni a fő ciklus, mivel ciklikusan hívja meg a tagfüggvényeket, melyek elvégzik az adott feladatot.

Bekapcsolás után inicializálja a program működésének az összes paraméterét. A TCP/IP Stack-et, Mikrokontrollert, WiFi modult, Memóriakártya olvasót és az összes egyéb paramétert ami a működéshez szükséges.

A fő program ciklikusan hívja meg a feladatokat megvalósító rutinokat:

1. Saját kétállapotú bemeneteit beolvassa.

2. Beolvassa az A/D értékeket(hőmérséklet, feszültség, áram).

3. Lekéri a mérő átalakítóktól a mérési eredményeket(Modbusz).

4. Ezek után a kapott értékekből összeállít egy rekordot.

5. A rekordot rögzíti a memóriakártyán.

6. Az utoljára berögzített rekordot küldi el az adatbázis szervernek.

4.4 Memória kártya implementálása

4.4.1 FAT16 és FAT32 összehasonlítása

A két fájl rendszer nevében szereplő szám a fájl allokációs tábla méretét fejezik ki, azaz a FAT16 16 bites fájl allokációs táblát használ, míg a FAT32 esetében az első 4 bit le van foglalva ami azt jelenti, hogy 28 bit áll rendelkezésre a fájl allokációs tábla reprezentálására.

A FAT16 maximum 65.535 klasztert támogat kötetenként, maximális kötet méret: 2GB. A FAT32 esetén maximum 4.177.918 kalszter használható, maximális kötet méret: 32GB.

A FAT32 lehetővé teszi a finomabb felosztást a kötetnek(körülbelül 4 millió egység/ kötet). Ami nagyon fontos különbség a két fájl rendszer között, hogy a FAT32 esetében a gyökér mappa egy általános klaszter lánc, ebből adódóan bárhol allokálható és nem korlátozza a bejegyzések számát a gyökér mappában a FAT16-al szemben.[31]

4.4.2 MMD fájl rendszer

Ezen részfeladat során a Microchip által nyújtott Microchip Memory Disk Driver fájl rendszer keretrendszerét használtam fel.

Főbb tulajdonságok:

• ISO/IEC 9293 szabvány.

• FAT16 fájlrendszer alkalmazása.

• 2GB-os SD, MMC memóriakártyák használhatók. Az SD és az MMC kártyák igen hasonlóak, annyi a különbség van köztük, hogy SD kártyák esetében lehetőség van titkosításra.

• Interfész: SPI buszon kommunikál a mikrokontroller a memóriakártyával. Négy alapvető kommunikációs lábat használ fel: data in(SDI), data out(SDO), clock(SCK), chip select(CS) és az SD kártya esetében van még két további jel a card-detect és a write-protect, ami felhasználó számára jelzi, hogy fizikailag be van-e helyezve a memóriakártya az olvasóba vagy esetleg írás védett a kártya.

• A ISO/IEC 9293 szabvány szerint szektorokban tárolódnak az adatok, 512 byte egy szektor mérete.

• A memória terület szektorokra van fel osztva, az első szektor a MBR(Master Boot Record), a mi logikai információkat tartalmaz a memória kártyáról, és ezeket a szektorokat partícióknak hívjuk. A partíciók például a következők lehetnek: Boot szektor, FAT régió, Gyökér mappa, Adat régió.

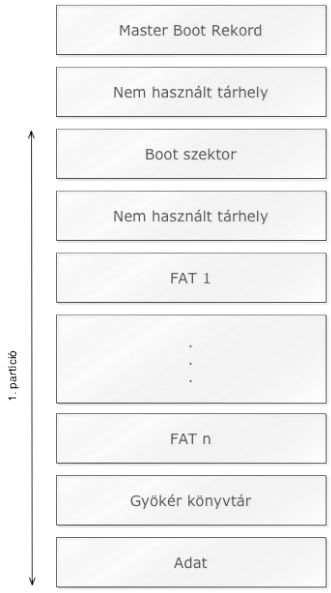
A Boot szektor az első régió és információkat tartalmaz a fájlrendszer típusáról.

A FAT régió tulajdonképpen egy térkép, ami megmutatja a kalszterek allokációját. Továbbá a FAT régióban található meg egy másolat is önmagáról az esetleges adat sérülések miatt.

A gyökér mappa régió következik a FAT régió után. Egy táblázatot tartalmaz, amiben bejegyzések találhatók a memóriakártyán lévő könyvtárakról és fájlokról.

Az első három régiót szokás rendszer területnek nevezni, hiszen többek között információkat tartalmaznak a rendszerről. A fennmaradó rész az adatok számára van fenntartva azaz ez a adat régió. Az adatok ebben a régióban inaktívak maradnak még törlés esetén is mindaddig, amíg felül nem írjak azokat.

A FAT16 fájl rendszer 16 bit-es FAT bejegyzéseket használ, ami 2^16-on klaszter reprezentálásra ad lehetőséget. Egy byte a boot szektorban megadja, hogy a szektorok számát klaszterenként a memóriakártyán. Ennek a byte-nak az értéktartománya: -128 - 127. A FAT16 rendszerben csak a pozitív kettes hatványi (1,2, 4, 8, 16, 32, 64) használhatóak. Ez azt jelenti, hogy egy standard 512 byte-os szektor méret esetén a FAT16 fájl rendszer maximum 2GB tárkapacitás támogat. [32]



15. ábra: SD memória kártya blokkvázlata

Master Boot Rekord

Ez a régió olyan információkat tartalmaz ami a kártya boot-olásához szüksége, illetve a kártyán levő partíciókról. Ezek az információk a gyártáskor programozzák be a gyártó cégek. Bármilyen kísérlet ezen információk átírására a kártyát használhatatlanná teheti. [32]

Offset	Leírás	Méret
000h	Boot kód	446 byte
1BEh	1. Partició bejegyzés	16 byte
1CEh	2. Partició bejegyzés	16 byte
1DEh	3. Partició bejegyzés	16 byte
1EEh	4. Partició bejegyzés	16 byte
1FEh	Boot aláírás kódja	2 byte

16. ábra: MasterBoot Record blokkvázlata

Partició bejegyzések a Master Boot Rekordban

A memóriakártyán lévő partíciókról az információk egy táblázatban vannak eltárolva a MBR-ban, amit Partició Tábla Bejegyzésnek hívunk. A táblázat egyik eleme a Fájl Rendszer Deszkriptor, ami azt jelzi, hogy milyen típusú a fájl rendszer az adott partíción. Például FAT16 esetén: 04h(16 bit FAT, <32M), 06h(16 bit FAT, >=32M). SD kártyák esetén általában egy aktív partíció van. [32]

Offset	Leírás	Méret
00h	Boot Deszkriptor	1 byte
04h	1. Partició szelektor	3 byte
05h	Fájl rendszer deszkriptor	1 byte
08h	Szektorok száma MBR és az első szektor között a partición	4 byte
0Ch	Szektorok száma a partición	4 byte

17. ábra: Partition Entry blokkvázlata

Boot szektor

Ez az első szektora egy partíciónak, rendszer információkat tartalmaz, mint például a mutatókat. Az első bejegyzés a szektorban az a Jump parancs, ami a boot információk átugrását jelenti. [32]
Példák a tárolt ionformációkra:

Offset	Leírás	Méret
00h	Jump parancs	3 byte
03h	OEM név	8 byte
0Bh	Byte/szektor	2 byte
0Dh	Szektor/klaszter	1 byte
0Eh	Összes lefoglalt szektor mérete	2 byte
10h	Fájl allokációs táblák száma	1 byte
11h	Gyökér mappa bejegyzések száma	2 byte
13h	Szektorok száma	2 byte
15h	Media Descriptor	1 byte
16h	Szektor/FAT	2 byte

18. ábra: Boot szektor blokkvázlata

Gyökér mappa

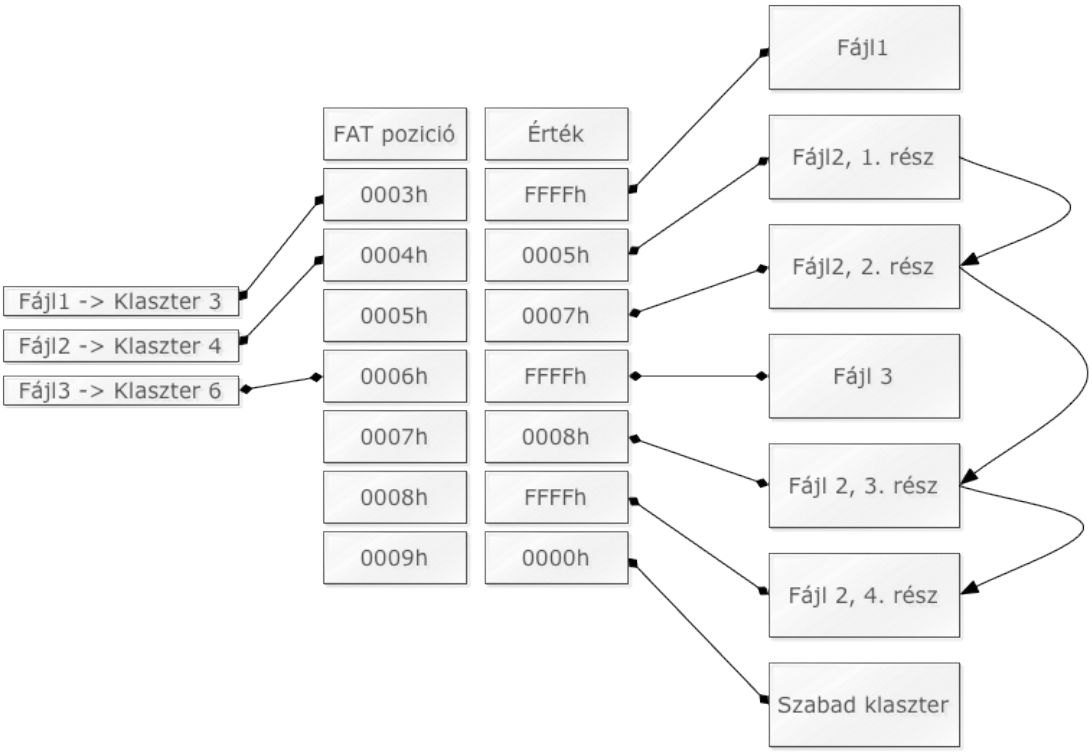
A Gyökér mappa a FAT régió után következik és nem más mint egy táblázat, ami információkat tárol a kártyán lévő mappákról és fájlokról 32 byte-os egységekben. Ez a bejegyzés tartalmazza a fájl nevét, méretét és azt a dátumot amikor létrehozva/ módosítva volt. [32]
A táblázat teljes tartalma:

Offset	Leírás	Méret
00h	Fájl név	1 byte
08h	Fájl kiterjesztése	3 byte
0Bh	Fájl attributúmok	1 byte
0Ch	Lefoglalt	1 byte
0Dh	A fájl létrehozási ideje(ms)	1 byte
0Eh	A fájl létrehozási ideje(óra,perc, másodperc)	2 byte
10h	Fájl létrehozás dátuma	2 byte
12h	Utlosó módosítás dátuma	2 byte
14h	Kiterjesztett cím index	2 byte
16h	Utolsó frissítés ideje	2 byte
18h	Utolsó frissítés dátuma	2 byte
1Ah	Első klaszter	2 byte
1Ch	Fájl mérete	4 byte

19. ábra: Gyökér mappa blokkvázlata

Fájl allokációs tábla

A FAT fájl rendszer 2 byte méretű klaszterekett alkalmaz. Minden fájlhoz legalább egy klaszter van hozzá rendelve. Ha az adott fájl mérete kisebb mint a hozzárendelt klaszter mérete, akkor a FAT bejegyzés ami ahhoz a klaszterhez tartozik, az tárolni fogja az utolsó kalszter értékét, ami azt jelenti, hogy nem tartozik már ahhoz a fájlhoz több kalszter. Egyéb esetben a FAT bejegyzés a következő kalszter értékét fogja tárolni. A klaszterek össze fűzésével a FAT nagyobb méretű fájlokat is tud tárolni. [32]
Példa FAT klaszterek összefűzésére:



20. ábra: Példa FAT klaszterek összefűzésére

Mappák

Ebben a fájl rendszerben a mappákat is úgy hozhatjuk létre mint a fájlokat, kivéve a gyökér mappát. Minden egyes mappa rendelkezik egy vagy több klaszterrel, mappa bejegyzéssel és a FAT bejegyzések láncolatával. Ha a negyedik bit be van állítva a mappa bejegyzésbe, akkor az azt jelöli, hogy a bejegyzés egy mappához tartozik. Abban különböznek a mappák a fájloktól, hogy nincs kiterjesztésük. Minden egyes mappához tartozik egy 32byt-os mappa bejegyzés. Két mappa bejegyzés a pont és a pont-pont jelen vannak minden egyes mappában kivétel a gyökér mappában. A pont bejegyzés az első bejegyzés minden alkönyvtárban, az érték ebben az bejegyzésben egy sima pont(2Eh) amit 10 space karakter(20h) követ. A klaszter arra a mappára fog mutatni amiben van a mappa.

Amikor inicializáljuk a kártyát az FSInit paranccsal, akkor a jelenlegi mappa amibe írni fogunk a gyökér mappa lesz, majd ezek után ezt meg tudjuk változtatni a FSchdir paranccsal.

Néhány fontosabb függvény

FSInit: Inicializálja a memória kártyát, betölti a Master Boot Recor-ot és a Boot szektort és ezek alapján beállítja a paramétereket.

FSfclose: Ez a parancs frissíti a fájl információkat, beírja a maradék adatot a fájlba, végül frissíti a fájl időbélyegjét.

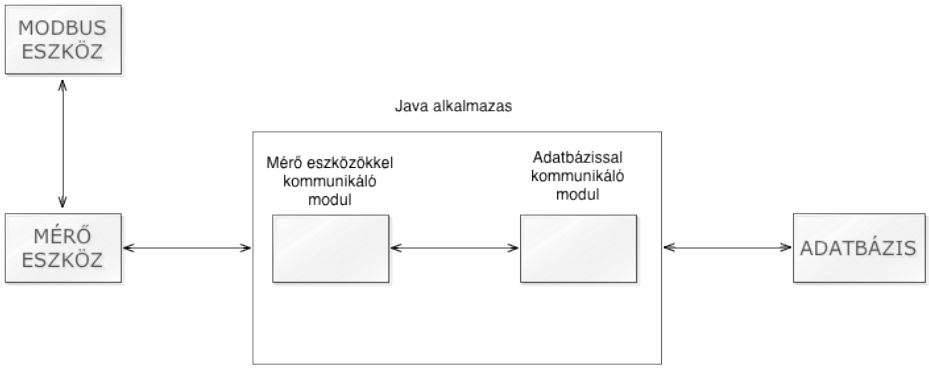
FSread: Ez a parancs adatokat olvas ki a már előzőleg megnyitott fájlból és azokat eltárolja egy bufferbe. Visszatérési érteke: kiolvasott adat objektumok száma.

FSfopenpgm: Ez a parancs megnyitja az SD kártyán lévő fájlt, csak PIC18-as mikrokontrollereknél használhat.

FSfwrite: Ez a parancs információt ír a már előzőleg megnyitott fájlba egy bufferből. [32]

4.5 Adatbázis rendszer megvalósítása

Az általam megvalósított rendszer felépítését fogom bővebben a következőkben taglalni. 3 fő komponensből áll az a rendszer, ami az adatok adatbázisban való rögzítéséért felelős.



21. ábra: Adatbázis rendszer felépítése

Mérőeszköz

A mérőeszközön történik az adatok mérése és gyűjtése, a mérőeszköz ciklikusan elvégzi a megfelelő méréseket, azokból összeállít egy üzenetet és lerögzíti a memória kártyára, majd kezdeményez egy TCP kapcsolatot a Java applikáció szerver program egységével. Az üzenet összeállítása egy egyedi nagyon egyszerű protokoll szerint történik.

> Üzenet:
> Az üzenet a következő képen épül fel:
> “Eszköz ID-Temperature-mért érték-Pot-mért érték-Pactime-érték-ModReg-Modbus regiser értéke”
>
> Például: “Pic1-Temp-23-Pot-58-Pactime-1223-ModReg-23”

Az üzenet elemei egy char típusú karakter tömbben tárolódnak és ez kerül elküldésre a Java-s applikáció számára.

Java applikáció

Ahogy a nevében is szerepel, Java nyelven íródott s az Eclipse IDE-ben feljlesztettem a szoftvert.
Logikailag az alkalmazás a mérőeszköz és az adatbázis között helyezkedik el. Fő feladata, hogy kapcsolatot tartson mind a mérőeszközzel és az adatbázissal is. Ez azért fontos, hogy nem lássa direktben egymást az adatbázis és a mérőeszköz. Továbbá az is a szerepe, hogy leválassza az adatbázist az internetről, tehát közvetlenül nem látható az internet felől az adatbázis, ez biztonsági szempont miatt lett így kialakítva.

Három fő komponensből áll az applikáció, a szerver rész, adatbázisoshoz csatlakozó rész és a koordinátor rész.

Szerver

Az alkalmazás ezen részegysége felelős azért, hogy figyelje a bejövő kéréseket, azaz egy előre definiált porton(11113 port) figyeli a csatlakozni kívánó mérő eszközöket. Amint észlel egy kérést, TCP socket nyílik meg és fogadja feldolgozásra e bejövő adatot, ami már a koordinátor rész feladata lesz.

Az alkalmazás indításánál a Coordinator osztály elindítja szervert és bemenő paraméterkén átadja neki a figyelendő port számát. Ekkor a server figyel az adott porton és ha érzékel egy igényt a TCP socket nyitására, akkor ennek eleget tesz, de oly módon, hogy elindít egy szálat. Tehát ezek után párhuzamosan fut tovább a alkalmazás, a sevrer továbbra is figyel az adott porton a lehetséges igényeket, míg az elindított szálon jön létre a TCP socket az alkalmazás és a mérőeszköz között, ahol megtörténik az adatátvitel és a Coordinátor osztály innentől kezdve átveszi a feladatot és feldolgozza az üzenetet, eközben az adat beérkezésese után zárul a TCP socket az alkalmazás és a mérőberendezés között.

Koordinátor

A Coordinator osztályban valósítottam meg, fő feladata, hogy össze hangolja a komponenseket és koordinálja az üzenet feldolgozását és az adatbázisiba való rögzítést.

Rendelkezik egy public void receiveMessageFromNetwork(String s) metódussal, aminek a bemenő paramétere egy string, ez a metódus a szálból hívódik meg amikor megérkezett az üzenet a TCP socketen, tehát a bemenő paramétere a nyers üzenet ami a mérőberendezés felől érkezik.
Első lépésben feldolgozza a beérkezett nyers üzenetet, majd egy olyan kompatibilis formára hozza ami már az adatbázis számára is feldolgozható.
Ezek után kapcsolódik az adatbázis szerverhez, és elküldi a megfelelő SQL parancsot a megfelelő tartalommal.
Az adatbázishoz kapcsolódás a ConnectionHandler osztály segítségével történik, betölti a kapcsolódáshoz szükséges JDBC driver és az előre definiált jelszó, felhasználónév és adatbázis azonosító segítségével létrehozza a kapcsolatot, majd végül átadja a Connection típusú objektumot, ami az élő kapcsolatot reprezentálja az adatbázissal, a Coordinátor osztály receiveMessageFromNetwor(String s) függvényének.

```
//measurement(id serial, deviceid text, temp float, pot float, packtime float, createdat timestamp);
String statem = "INSERT INTO measurement (deviceid, temp, pot, packtime) VALUES ( '" + strings[0] + "', " + temp + ", " + pot + ", " + packtime + " )";
```

A fenti példából is látszik, hogy egy String típusú váltózó formájában küldi el az SQL utasítást és már szerepelnek a megfelelő adatok az üzenetben. Majd ezt a String

típusú statem változót küldi el az adatbázisnak az előzőleg paraméterként átvett Connection típusú objektum segítségével.

ConnectionHandler

A ConnectionHandler osztály egyedüli feladata az, hogy csatlakozzon előre definiált paraméterek alapján az adatbázishoz.
Első lépésként betölti a csatlakozáshoz szükséges JDBC drivert, ami illeszti a Java-s alkalmazást az PostgreSQL adatbázishoz. Az előre definiált jelszó, felhasználónév és adatbázis azonosítóval kacsolódik az adatbázishoz és a kapcsolat készen áll az üzenetek küldésére.

```
private ConnectionHandler(String _usrname, String _password) throws
ClassNotFoundException, SQLException
{
Class.forName("org.postgresql.Driver");
conn = DriverManager.getConnection("jdbc:postgresql://localhost/
PicData","felhasznalonev", "jelszo");
}
```

Ahogy a fenti kód részletben is láthatjuk localhost címen keresztül kapcsolódik az alkalmazás az adatbázishoz. Az adatbázis úgy van konfigurálva, hogy csak a localhost címen elérhető, ezzel biztosítjuk azt, hogy az adatbázis el van szeparálva teljesen a külvilágtól és csak a java-s alkalmazáson keresztül lehet elérni.

Adatbázis

A PostgreSQL 9.0.5-ös verzióját használtam a feladat megoldása során. OSX platformra installáltam fel, majd létrehoztam a PicData nevű adatbázist az alkalmazásom számára.
Ezek után létrehoztam az adatbázisban a táblát, amibe az eszközök adataid tárolom.
create table measurement(id serial, deviceid text, temp float, pot float, packtime float, modreg float createdat timestamp);

A fenti példában egy measuremnet nevű táblát hoztam létre és a tábla attribútumai id, deviceid, temp, pot, packtime, modreg, createdat.

• id: sor egyedi azonosítója.
• deviceid: az eszköz azonosítója, ahonnan a mért érték származik.
• temp: hőmérséklet.
• pot: potméter.
• packtime: a mérés ideje(mikrokontroller határozza meg).
• modreg: Modbus register értéke.
• createdat: az adatbázisba való rögzítés ideje(adatbázis határozza meg).

Továbbiakban létre hoztam egy trigger-t is, ami legenerálja minden beillesztésnél az egyedi azonosítót és a rögzítés dátumát is.

Példa egy trigger létrehozására:

```
CREATE OR REPLACE FUNCTION create_created_column()
        RETURNS TRIGGER AS $$
        BEGIN
                NEW.createdat = now();
        RETURN NEW;
        END;
$$ language 'plpgsql';

CREATE TRIGGER create_measurement_createdate BEFORE INSERT
ON measurement FOR EACH ROW EXECUTE PROCEDURE
create_created_column();
```

Ennek a trigger-nek a segítségével minden beszúrásnál az aktuális időt betölti a createdat változóban, tehát a rögzítés dátuma is bekerül az adat mellé.

4.6 WiFi implementálása

Ebben a fejezetben a WiFi wireles hálózati interfész implementálását szeretnem kifejteni.

A feladat megvalósítása sora a Microchip álltal gyártott MRF24WB0MA WiFi modult használtam.

A MRF24WB0MA WiFi modul alacsony fogyasztású eszköz, mai 2.4 GHz frekvencián üzemel, megfelel az IEEE szabványnak és 802.11 kompatibilis. Tartalmazza az összes RF alkatrészeket, kristály oszcillátort, MAC, teljesítmény erősítő és beépített hardveres támogatást az AES és TKIP(WEP, WPA, WPA2) titkosítások számára. Továbbá az antennát is tartalmazza a modúl.

Úgy tervezték az eszközt, hogy könnyen használható legyen a Microchip TCP/IP stack keretrendszerrel. A keretrendszer biztosít egy illesztő programot a modul számára, ami lebonyolítja az adatcsomagok menedzselését. Tulajdonképpen csak a megfelelő header fájlokat kell inkludálni, majd beállítani a WiFi kapcsolat paramétereit (SSID, titkosítás, jelszó) és készen is áll a modul a használatra.

Könnyen illeszthető a legtöbb PIC mikrokontrollerhez az SPI buszon. 3.3V a működési tápfeszültsége.

Az energiatakarékosság jegyében a modul számos üzemállapottal rendelkezik. Ezek a hibernáció, alvás és az aktív(két fázis) állapotok. Ezek az állapotok mind befolyással vannak az energia használatra. Továbbá van meg egy fázis ami a Standby, de ez ezt az állapotot a felhasználó nem tudja kezelni.

Hibernált állapot

Ez az állapot van a legközelebb a kikapcsolt állapothoz, a Hibernate lábon keresztül lehet aktiválni. Ebben az esetben az energiafogyasztás minimális, viszont ezt az állapotot teljesen a mikrokontrollernek kell felügyelnie, csak a mikrokontroller tudja kihozni ebből az állapotból. Abban az esetben, ha az energia ellátást AA elemekkel biztosítjuk és csak óránkén van adatátvitel(kevesebb mint 500 byte), akkor az üzemidő egy év is lehet.

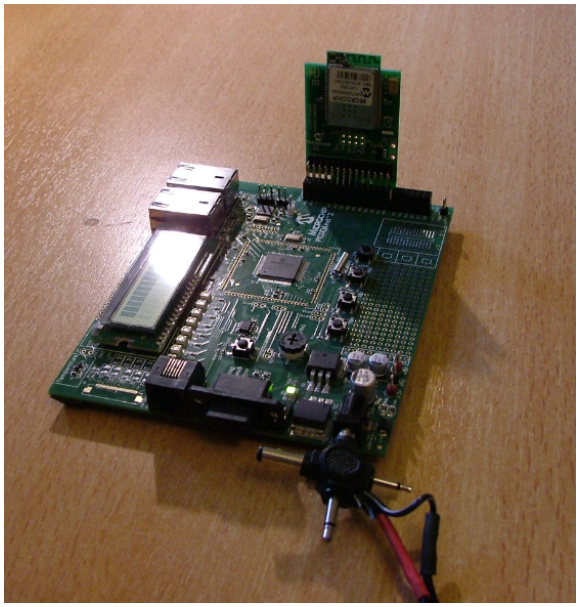
Alvó állapot

Az alvó állapot, alacsony energia fogyasztású dinamikus állapot, ami teljesíti a 802.11 energia takarékos tulajdonságokat. Minden aktív fázis után az adatátvitel befejezésével belé ebbe az állapotba. A mikrokontroller fel tudja ébreszteni bizonyos időközönként, hogy ellenőrizze volt-e igény a kommunikációra. Ez az állapot is igen

alacsony energia fogyasztású, az üzemidő elem esetén változó lehet, egy vagy több hét is az adat forgalomtól függően.

Aktív státusz

Itt adó és vevő állapotot különböztetőnk meg, ebben az esetben a legnagyobb az energiafogyasztás.



22. ábra: Mérőeszköz és a WiFi modul működés közben

4.7 Hőmérséklet mérés megoldása

A fejlesztő eszközön installált TC1047 típusú hőmérő egység, mely kimenete feszültség és referencia feszültségként a mikrokontroller tápfeszültségét kapja ami 3.3V. Majd a mért feszültséget a mikrokontroller AD konverter bemenetére van kötve a AN3-as lábra.

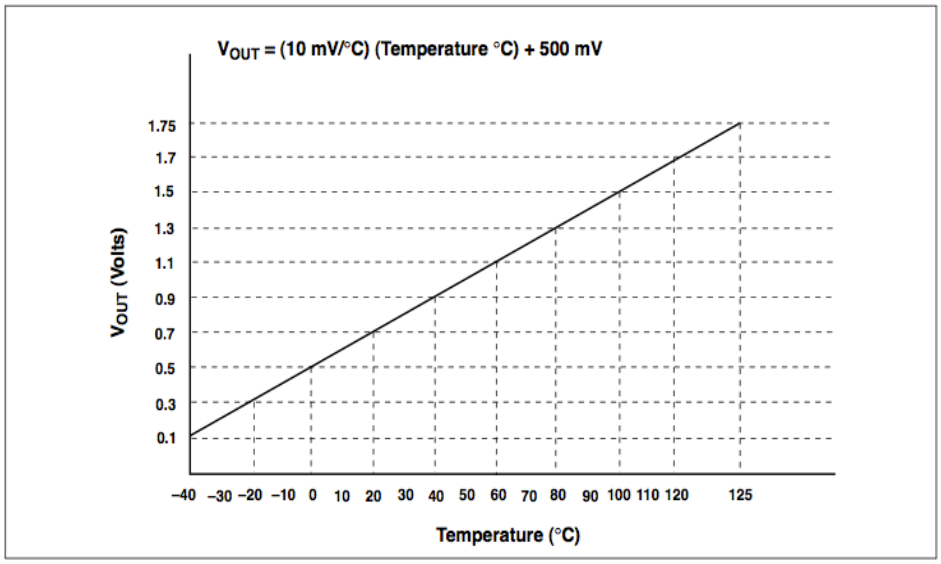


FIGURE 4-1: *Output Voltage vs. Temperature.*

23. ábra: Hőmérséklet a kimeneti feszültség függvényében

A fenti ábrán látható függvény alapján lehet kiszámolni a mért feszültségből a hőmérsékletet (Vout = (10mV/ C)(Temperature C) + 500mV). A PIC 18F97J60 –as mikrokontrollerben 10bit-es AD konverter található, 2^10=1024, tehát 1024 bit felel meg 3.3V-nak azaz 1024 bit a felbontás ebben az esetben, ami azt jelenti, hogy 1 bit = 3.22 mV-nak felel meg. Az AD konverter által mért érték az ADDRESS regiszterben tarolódik. Így könnyen meghatározható a Vout (Vout = AD konverter kimenet (bit)* 3.22mV), így már a fenti függvény alapján ki is tudjuk számolni a hőmérsékletet.

4.8 Feszültség mérés megoldása

Az áramkörön installálva van egy potméter is, amely működési elve hasonlít az előbb említett hőmérséklet szenzor működési elvéhez. Referencia feszültségnek a potméter is 3.3V feszültséget kap. Majd a kimeneti feszültség a mikrokontroller AD konverter bemenetére van kötve a AN2-as lábra. 10 bit-es AD konverter esetén 1 bit = 3.22mV-nak felel meg, mint a fent említett esetben. Az AD konverter által mért érték az ADDRESS regiszterben tarolódik szintén.

Ezzel a méréssel szimulálhatom a feszültség bemeneteket (0-20 mA, 4-20 mA, 0-10 V), ami azt jelenit hogy megvan az összes típusú feszültség mérés illetve áram mérés feszültség konverter illetve áram konverter esetleges felhasználásával.

4.9 Weblap tárolása

Továbbá az áramkörön installálva van egy külső EEPROM tároló is, a mi a Web lapok tarolására szolgál. A Web lapok bináris formában tarolódnak, szerencsére a MICROCHIP rendelkezésre bocsájtott egy olyan konverter programot, ami a HTML oldalt átalakítja bináris formára és ezt a binárist lehet letölteni a külső EEPROM tárolóba. A web oldal feltöltésére több lehetőségünk is van FTP-n keresztül tudjuk feltölteni a fájlt, továbbá elérhető egy online feltöltő opció is a web böngészőbe begépeljük, hogy http://192.168.1.1/mpfsupload ahol a 192.168.1.1 az eszköz IP címe és ekkor megjelenik egy online feltöltő ablak, ahol ki tudjuk választani a feltölteni kívánt file-t.

24. ábra: Weblap

4.10 Eszköz használata

A PICDEM.net 2 fejlesztő eszközt csatlakoztatni kell a helyi számítógépes hálózatba, akár egy router-en keresztül. Így az eszköz automatikusan megkapja az IP címét ami majd megjelenik az LCD kijelzőn is és már elérhetővé vált, vagy akár közvetlenül hozzá lehet csatlakoztatni a megfelelő kábellel a számítógépünk hálózati interfészéhez, de előbb ki kell kapcsolni az eszközben az active DHCP-t és be kell állítanunk egy fix IP címet, arra figyelve, hogy egy hálózati szegmensben legyen a

számítógépünk hálózati interfészével. Azért van szükség az aktív DHCP kikapcsolására az eszközben, mert amíg él ez a beállítás, addig az eszköz arra vár hogy kiosszák neki az IP címét, ami például egy router feladata lenne, de a mi számítógépünk ezt nem teszi meg. Majd be kell állítani az adatbázis szerver IP címét és portját.

A mérő berendezés soros portjára kell csatlakozni a Modbusz-os mérő eszközöket.

Amint csatlakoztattuk az eszközt a hálózathoz, a fent említett bármely módon és az LCD kijelzőn leolvasható az eszköz IP címe, akkor már elérhetővé vált a helyi hálózatban. Begépelhetjük az eszköz IP címét egy web böngészőbe, ekkor megjelenik a készülék kezdőlapja, ahol máris leolvashatjuk a pillanatnyi hőmérsékletet és a potméter által mért feszültség értékét és láthatjuk az LCD kijelzőn megjelenített karaktereket. Akár írhatunk egy üzenetet is a webes felületen, ami megjelenik az áramkör LCD kijelzőjén.

Ha az interneten keresztül szeretnénk elérni a Web lapot, ahhoz át kell konfigurálnunk a helyi hálózatot. Elsősorban a routert-ben konfigurálni kell a Port Forwarding opciót a az áramkör IP címének a 80-as port-ját kell továbbítani, tehát ezzel azt érjük el, hogy az internet felől az eszközünk 80-as TCP port-ja fog látszani, ami mögött az eszközünkön implementált web szerver fogja a kéréseket kiszolgálni. Továbbá egy Domain nevet kell hozzárendelnünk az helyi hálózatunk külső internet felőli IP címéhez. Ebben az esetben a Dyndns.org ingyenes szolgáltatását vettem igénybe, azért esett erre a szolgáltatásra a választásom, mert az otthoni helyi hálózatomnak nincs fix külső internet felőli IP címe, ezért a dyndns.org elérhetővé tesz egy olyan alkalmazást (Dyndns Updater), ami frissíti a hálózat külső internet felőli IP címét az általunk lefoglalt Domain névhez, amit szintén ingyenesen megtehetünk a dyndns.org-nál. Így már csak annyi a dolgunk, hogy egy web böngészőbe begépeljük az általunk kiválasztott Domain nevet(http://meroberendezes.dyndns.org).

Ha az interneten keresztül akarjuk elérni az adatbázis szervert, akkor nincs más dolgunk mint egy olyan hálózathoz csatlakoztatjuk a mérő eszközt ami rendelkezik internet kapcsolattal. Viszont az adatbázis rendszert kiszolgáló helyi hálózatot úgy kell bekonfigurálni, hasonló képen mint a fent említett esetében, hogy a Port Forwarding opciót helyesen be kell konfiguráljuk. Így az adatbázis szerver bárhol lehet,nem szükségszerű, hogy egy helyi hálózatban legyen a mérő berendezéssel.

5 Összefoglaló

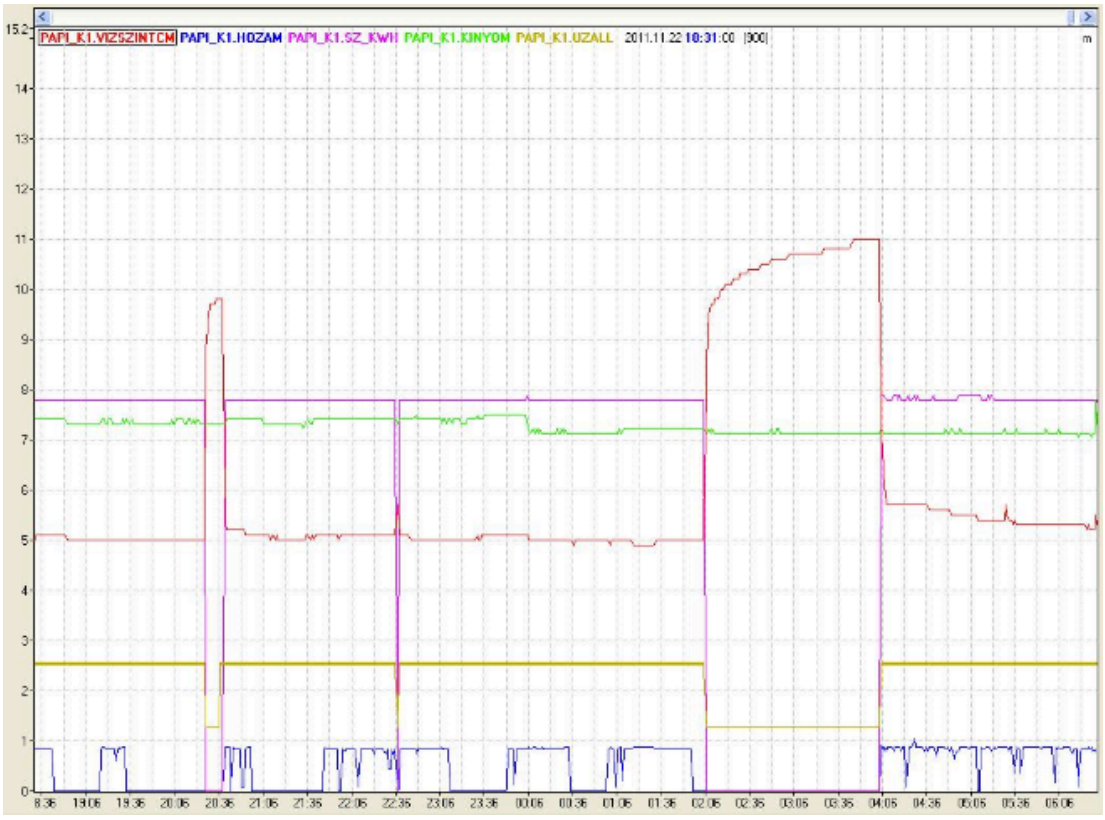
5.1 Mérési eredmények

A mérési az alábbiak szerint lett elvégezve:

25. ábra: Első mérés logikai vázlata

A Érd és térsége Vízművekhez tartozó Papi Földek nevű telepen az 1-es számú kútjánál végeztem el a mérést. A helyszínen lévő mérés-adat gyűjtő PLC soros portjára csatlakoztattam az általam elkészített mérő-adatgyűjtő berendezést és TCP/IP hálózaton keresztül csatlakozott egy laptophoz, melyen a Vison mérés-adatgyűjtő szoftver [33] futott. Egy napig futott a mérés-adatgyűjtés és az összes PLC által gyűjtött adat rögzítésre került a Vison adatbázisába. A mérő berendezés szoftverén módosítani kellett(a szükséges függvények már rendelkezésre álltak, csak annyit kellet tenni, hogy a TCP socket-en küldött üzenetet úgy kellet módosítani, hogy egy Modbus RTU formátumú üzenetet tudjon elküldeni). Tulajdonképpen a TCP csomagba ágyazott Modbus üzenet került elküldésre, ennek az oka az, hogy a Vison-hoz így lehetett illeszteni a berendezést. Enne a mérésnek az a hátrány, hogy nem tesztelte az általam kialakított adatbázis rendszert, annak tesztelésére egy másik mérést alakítottam ki.

Az alábbi grafikonon láthatók a mért értékek, ami a Vison mérés-adatgyűjtő szoftver segítségével készült:



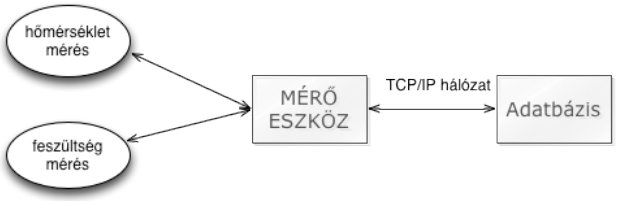
26. ábra: A mérésekből származtatott grafikon

Piros - vízszint(cm)
Kék - pillanatnyi termelt víz mennyiség(m^3/h)
Rózsaszín - állomás pillanatnyi teljesítmény felvétele(kWh)
Zöld - kimenő nyomás(bar)
Sárga - Szivattyú üzemállapota.(on/off)

Grafikon elemzés:

20:36 perckor a szivattyú megállt(sárga), melynek következtében a vízszint(piros) megemelkedett, a pillanatnyi termelt víz mennyiség(kék) leesett nulla szintre, a felvett villamos teljesítmény(rózsaszín) is minimális szintre esett vissza. A kimenő nyomás (zöld) is minimálisan csökkenésének az oka, hogy a kút szivattyú maximális emelő magassága nem sokkal több, mint a hálózatban lévő ellennyomás.

Az általam készített adatbázis rendszer tesztelése az alábbiak szerint történt. A mérő berendezés szoftverét úgy módosítottam, hogy 5 másodpercenként küldje el a mért értékeket az adatbázisba és rögzítse a memóriakártyára.



27. ábra: Második mérés logikai vázlata

Az alábbi képen látható az adatbázisban tárolt adatok:

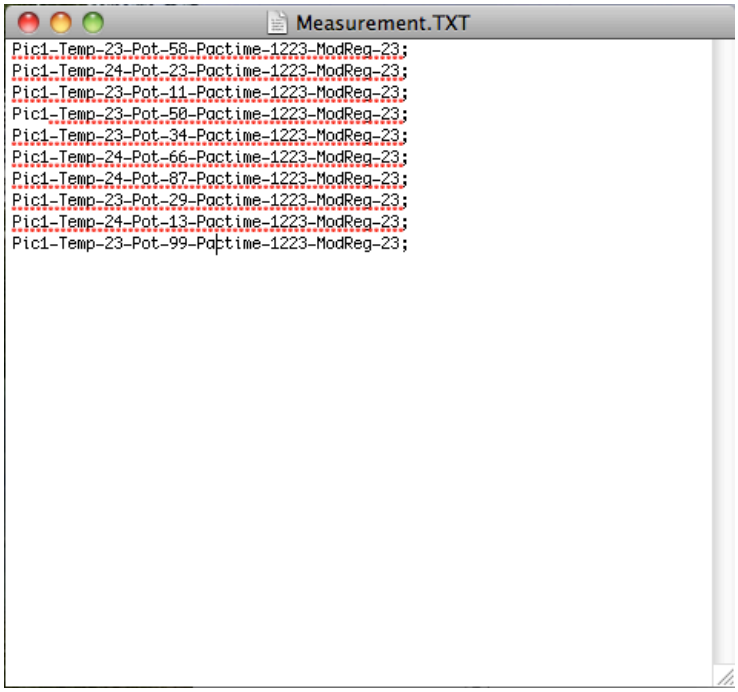
Terminal — more — 80×24

id	deviceid	temp	pot	packtime	createdat
1	Pic1	3	12	230	2011-10-27 20:14:01.433158
2	Pic1	3	12	230	2011-10-27 20:14:06.343097
3	Pic1	3	12	230	2011-10-27 20:14:11.983441
4	Pic1	3	12	230	2011-10-27 20:14:17.310718
5	Pic1	3	12	230	2011-10-27 20:14:22.633699
6	Pic1	3	12	230	2011-10-27 20:14:28.078151
7	Pic1	3	12	230	2011-10-27 20:14:33.278298
8	Pic1	3	12	230	2011-10-27 20:14:38.603645
9	Pic1	3	12	230	2011-10-27 20:14:44.244538
10	Pic1	3	12	230	2011-10-27 20:14:49.558256
11	Pic1	3	12	230	2011-10-27 20:14:55.705401
12	Pic1	3	12	230	2011-10-27 20:15:01.235763
13	Pic1	3	12	230	2011-10-27 20:15:06.554838
14	Pic1	3	12	230	2011-10-27 20:15:11.889745
15	Pic1	3	12	230	2011-10-27 20:15:17.212525
16	Pic1	3	12	230	2011-10-27 20:15:22.532915
17	Pic1	3	12	230	2011-10-27 20:15:27.858884
18	Pic1	3	12	230	2011-10-27 20:15:33.176016
19	Pic1	3	12	230	2011-10-27 20:15:39.021618
20	Pic1	3	12	230	2011-10-27 20:15:44.55037
21	Pic1	3	12	230	2011-10-27 20:15:49.878746

:

28. ábra: A mérésből származó adatbázisban szereplő adatok

Az alábbi képen látható a memória kártyára lementett adatok:



28. ábra: A mérésből származó adatbázisban szereplő adatok

5.2 Továbbfejlesztési lehetőségek

Továbbra is szeretnék ezzel a témával foglalkozni az MSC diplomamunka keretén belül is. Ami nem valósult meg: a mérő eszköz legyártása és az óra áramkör implementálása. Az utóbbi azért volna nagyon fontos, hogy a mérőberendezés egy pontos időbélyeggel tudja ellátni a méréseket, sajnos jelenleg csak az adatbázis tud időt hozzá rendelni az adatokhoz, de az nem kellően pontos. Ezeket a funkciókat meg szeretném valósítani a közel jövőben. Továbbiakban kiszeretnék alakítani egy olyan mérést, amiben több mérőberendezés is szerepel, így pontosabban tudnám tesztelni a rendszer képességeit, esetleges gyenge pontjaira könnyebben fény derülhetne. Ehhez elengedhetetlen a berendezés legyártása.

Ezúton szeretnék köszönetet mondani Tihanyi Attila tanár úrnak, aki rengeteget segített a feladat megoldása során, sok ötletet, instrukciót kaptam tőle.

Rövidítésjegyzék

TCP/IP - Transmission Control Protocol/Internet Protocol

UDP - User Datagram Protocol

UTP - Unshielded Twisted Pair

PoE - Power over Ethernet

SQL - Structured Query Language

ARPA - Address and Routing Parameter Area

IETF - Internet Engineering Task Force

ICMP - Internet Control Message Protocol

DoD - States Department of Defense

NSF - National Science Foundation

CIDR - Classless Inter-Domain Routing

OSI - Open Systems Interconnection

CRC - cyclic redundancy check

LRC - Longitudinal Redundancy Check

RTU - remote terminal unit

ASCII - American Standard Code for Information Interchange

CR - carriage return

LF - line feed

ORDBMS - object-relational database management system

EEPROM - Electrically Erasable Programmable Read-Only Memory

USART - universal asynchronous receiver/transmitter

SPI - Serial Peripheral Interface Bus

WEP - Wired Equivalent Privacy

WPA - Wi-Fi Protected Access

SSID - service set identifier

LCD - liquid crystal display

PLC - programmable logic controller

Irodalomjegyzék

[1] Network Timetable http://www.ietf.org/rfc/rfc4.txt utolsó módosítás: 1997. november 19.

[2] Andrew Stuart Tannenbaum: Számítógép hálózatok, Budapest, Panem, ISBN 9635652136.

[3]The Day the Infant Internet Uttered its First Words: http://www.cs.ucla.edu/~lk/LK/Inet/1stmesg.html utolsó módosítás: 1999. október 26.

[4] Implementation of the Host - Host Software Procedures in GORDO http://tools.ietf.org/html/rfc11 utolsó módosítás: 2010. december 27.

[5] DoD Standard Internet Protocol http://tools.ietf.org/html/rfc760 utolsó módosítása: 2011. február 3.

[6] Internet Protocol DARPA Internet Program http://tools.ietf.org/html/rfc791 utolsó módosítása: 2011. február 3.

[7] The IP Network Address Translator (NAT) http://www.rfc-editor.org/rfc/rfc1631.txt utolsó frissítés: 1994. május 18.

[8] Classless Inter-Domain Routing (CIDR): an Address Assignment and Aggregation Strategy: http://www.rfc-editor.org/rfc/rfc1519.txt utolsó módosítás: 1993. szeptember 23.

[9] Internet Protocol, Version 6 (IPv6)Protocol definition: http://tools.ietf.org/html/rfc2460 utolsó módosítás: 2011. február 3.

[10] Transmission Control Protocol DARPA Internet Program Protocol Specification http://www.rfc-editor.org/rfc/std/std7.txt utolsó módosítás: 1981. szeptember 1.

[11] Gould Modbus Protocol Feference Guide, Gould Inc. Programmable Control Division P.O. Box 3083 Andover, Massachusetts, 01810. Kiadás dátuma: 1985 január.

[12] Cím: MySQL 5.5 Reference Manual, URL: http://dev.mysql.com/doc/refman/5.5/en/

[13] Cím: SQLite dokumentációk, URL: http://www.sqlite.org/docs.html

[14] Cím: PostgreSQL 8.1.0 Documentation, URL: http://www.postgresql.org/files/documentation/pdf/8.1/postgresql-8.1-US.pdf, utolsó módosítás: 2010.

[15]Cím: PicDem.net2 Internet/Development Board Users's Guide, URL: http://ww1.microchip.com/downloads/en/DeviceDoc/51623a.pdf, utolsó módosítás: 2006.

[16] C1m: PICkit3 Programmer/Debugger User's Guide, URL: http://ww1.microchip.com/downloads/en/DeviceDoc/51795B.pdf,utolsó módosítás: 2010.

[17] Cím: The Microchip TCP/IP Stack, URL: http://ww1.microchip.com/downloads/en/appnotes/00833b.pdf, utolsó módosítás: 2002.

[18] Cím: Pic18f97j60 Family Data Sheet, URL: http://ww1.microchip.com/downloads/en/DeviceDoc/39762a.pdf, utolsó módosítás: 2006.

[19] Cím: ENC28j60 Data Sheet, URL: http://ww1.microchip.com/downloads/en/devicedoc/39662a.pdf, utolsó módosítás: 2004.

[20]Cím: MRF24WB0MA/MRF24WB0MB Data Sheet, URL: http://ww1.microchip.com/downloads/en/DeviceDoc/70632A.pdf, utolsó módosítás: 2010.

[21] Cím: PICtailTM Daughter Board for SDTM and MMC Cards Data Sheet, URL: http://ww1.microchip.com/downloads/en/DeviceDoc/DS-51583b.pdf, utolsó módosítás: 2008.

[22] Cím: MPLAB C18 C COMPILER USER'S GUIDE, URL: http://ww1.microchip.com/downloads/en/devicedoc/51288f.pdf, utolsó módosítás: 2005.

[23]Cím: TC1275/TC1276/TC1277 3-Pin Reset Monitors for 3.3V Systems Data Sheet, URL: http://ww1.microchip.com/downloads/en/DeviceDoc/21383c.pdf, utolsó módosítás: 2007.

[24] Cím: TC1047/TC1047A Precision Temperature-to-Voltage Converter Data Sheet, URL: http://ww1.microchip.com/downloads/en/DeviceDoc/21498C.pdf, utolsó módosítás: 2005.

[25] Cím: MAX3232 3V TO 5.5V MULTICHANNEL RS232 LINE DRIVER/RECEIVER Data Sheet, URL: http://www.ti.com/lit/ds/symlink/max3232.pdf, utolsó módosítás: 2000.

[26] Cím: 24AA512/24LC512/24FC512 512K I2CTM CMOS Serial EEPROM Data Sheet, URL: http://ww1.microchip.com/downloads/en/devicedoc/21754g.pdf, utolsó módosítás: 2005.

[27] Cím: 3.3V-Powered, 10Mbps and Slew-Rate-Limited True RS-485/RS-422 Transceivers Data Sheet, URL: http://www.eec.uestc.edu.cn/zlxz/yqj/downloadInfo/serinterface(3)/max3485.pdf, utolsó módosítás: 1994.

[28] Cím: MC78M05/LM78M05 3-Terminal 0.5A Positive Voltage Regulator Data Sheet, URL: http://www.fairchildsemi.com/ds/MC/MC78M05.pdf, utolsó módosítás: 2011.

[29] Cím: TC1262 500mA Fixed Output CMOS LDO Data Sheet,URL: http://ww1.microchip.com/downloads/en/devicedoc/21373b.pdf, utolsó módosítás: 2002.

[30]Cím: SOT23 N-CHANNEL ENHANCEMENT MODE VERTICAL DMOS FET Data Sheet, URL: http://www.diodes.com/datasheets/ZVN3306F.pdf, utolsó módosítás: 1996.

[31] Cím: FAT16 vs. FAT32 URL: http://technet.microsoft.com/en-us/library/cc940351.aspx

[32] Cím: mplementing File I/O Functions Using Microchip's Memory Disk Drive File System Library, Szerző: Peter Reen Microchip Technology Inc, URL: http://ww1.microchip.com/downloads/en/AppNotes/01045a.pdf, utolsó módosítás: 2006.

[33] URL: http://www.provicon.hu/