VILNIAUS UNIVERSITETAS
FIZIKOS FAKULTETAS
RADIOFIZIKOS KATEDRA

Adolis Vaisieta

Duomenų perdavimas HTTP protokolu įterptinėse sistemose

Magistrantūros studijų baigiamasis darbas
(studijų programa – TELEKOMUNIKACIJŲ FIZIKA IR ELEKTRONIKA)

Studentas	Adolis Vaisieta
Darbo vadovas	doc. Vytautas Jonkus
Recenzentas	doc. Mindaugas Viliūnas
Katedros vedėjas	habil. dr. Jūras Banys

Vilnius 2017

Turinys

ĮVADAS

Kai žmonės išgirsta žodį mikroprocesorius dažnam pirma į galvą šovusi mintis yra nešiojamas kompiuteris ar išmanusis telefonas. Nors šios sritys yra populiarios, daugelis net nesuvokia, kaip dažnai tenka susidurti su mikroprocesoriumi to net nežinant. Nuo namuose esančio dūmų detektoriaus iki mikrobangų krosnelės, mikroprocesorius galima sutikti beveik kiekviename žingsnyje. Jų įterpimas į kasdieninę įrangą nėra naujas dalykas, prasidėjęs dar prieš išpopuliarėjant personaliniams kompiuteriams. Tokiose įterptinėse sistemose yra panaudojama didžioji dalis pagaminamų mikroprocesorių. Pavyzdžiui, naujas automobilis savyje gali turėti virš 50 mikroprocesorių valdančių sudėtingas variklio, stabdžių sistemas, atsakingus už laiku išskleistas, gyvybę saugančias oro pagalves ar tiesiog komforto lygį gerinančią oro kondicionavimo sistemą. Įterptinės sistemos atlieka darbus apie kuriuos dažnas žmogus net nesusimąsto.

Dar vienas dalykas, su kuriuo susiduriame kasdien yra internetas. Be jo pasaulis pasidarė nebeįsivaizduojamas, tai technologija persmelkusi bene visas sritis. Internetu dabar naudojasi beveik pusė pasaulio gyventojų. Atsiradus internetui, atsirado vis didesnis poreikis keistis informacija, nuo dešimtis gigabaitų užimančių didelės raiškos filmų iki paprasto laiškelio senai sutiktam draugui, žmonės generuoja vis daugiau ir daugiau duomenų. Pasak puslapio internetlivestats.com kiekvieną minutę yra išsiunčiama daugiau nei 200 milijonų laiškų elektroniniu paštu, įkeliama apie 50 valandų video į puslapį „YouTube“ bei atliekama 2 milijonai puslapio Google paieškų ir ką atlikdami kiekvieną iš šių operacijų žmonės greičiausiai mato prieš akis? Akronimą HTTP (iš angl. HyperText Transfer Protocol – hiperteksto perdavimo protokolas) – kas yra interneto, kurį mes taip gerai pažįstame, kertinis akmuo.

Per paskutiniuosius 10 metų vis labiau populiarėja šių dviejų sričių sujungimas – įterptinės sistemos su interneto prieiga. Toks vystymasis leidžia stebėti ar netgi valdyti įvairių įrenginių darbą per atstumą, nenaudojant daug resursų ir neišleidžiant didelės sumos pinigų.

Šio darbo tikslas yra suprojektuoti ir pagaminti įterptinės sistemos schemą, kuri naudojant TCP/IP bibliotekas galėtų realizuoti Web serverio funkcionalumą ir perduoti duomenis HTTP protokolu.

1. ĮTERPTINĖS SISTEMOS SU INTERNETO PRIEIGA

1.1. Įterptinių sistemų įvadas

Įterptinė sistema (iš angl. Embedded System) – mikroprocesoriumi paremta sistema, kuri sukurta atlikti vieną ar kelias funkcijas, kurios negali būti perprogramuotos galutinio vartotojo. Kitaip nei naudojant personalinį kompiuterį vartotojas turi pasirinkimų dėl sistemos funkcionalumo, bet negali jo pakeisti įrašant naują programinę įrangą. Personalinis kompiuteris vieną minutę gali būti naudojamas tekstui rašyti, kitą minutę – žaisti žaidimams ir tai pasiekiama pakeičiant programinę įrangą. Pagrindinės įterptinių sistemų dalys yra procesorius, atmintis bei įvesties bei išvesties įrenginiai.

Įterptinių sistemų pagrindinės dalys – procesoriai. Jie yra skirstomi į dvi plačias kategorijas. Paprasti mikroprocesoriai, kurių atmintis bei įvesties bei išvesties įrenginiai yra atskiri integriniai grandynai bei mikrovaldiklius. Mikrovaldiklius galima laikyti kaip savarankišką sistemą, kuri savo mikroschemoje turi procesorių, atmintį ir įvesties bei išvesties įrenginius. Tokiu būdu yra sumažinamos energijos sąnaudos, dydis bei kaina. Įterptinių sistemų CPU (iš angl. Central Processing Unit – centrinis procesorius) naudojama daugelis bazinių CPU architektūrų. Taip yra todėl, kad programinė įranga yra specialiai parašyta būtent tai įterptinei sistemai. Naudojama tiek Von Neumann, tiek Harvard architektūros. Procesorių žodžių ilgis svyruoja nuo 4 iki 64 bitų.

1.2. Įterptinės sistemos su interneto prieiga

Įterptinės sistemos su interneto prieiga nėra nauja idėja, tiesiog ji ilgą laiką nebuvo patraukli dėl didelių kaštų ir poreikio nebuvimo. Dėl TCP/IP steko programinės įrangos pajėgumo reikalavimų mažiems 8 ar 16 bitų mikrovaldikliams įterptinių sistemų prijungimas prie interneto buvo atmetamas, nes su juo siejami pliusai tiesiog nepateisindavo sąnaudų padidėjimo. Tokia programinė įranga vystymo pradžioje užimdavo daug tiek ROM, tiek brangios RAM atminties.

Dabar situacija yra pasikeitusi. Per paskutiniuosius metus atlikti programinės įrangos ir pačios mikrovaldiklių technologijos proveržiai leidžia internetą panaudoti netgi labai mažuose ir pigiuose mikrovaldikliuose. Ši programinė įranga yra labai efektyvi ir nereikalauja daug resursų. Galimybė plačiai panaudoti internetą duomenų perdavimui įterptinėse sistemose atveria daug galimybių.

Atsiranda poreikis šalia įterptinių sistemų programavimo, programuoti ir programinę įrangą patrauklią vartotojui, kurią naudojant būtų galima interaktyviai bendrauti su įterptine sistema per nuotolį. Tokio tipo programavimas ankščiau buvo reikalingas tik personalinių kompiuterių programuotojams. Tačiau įterptinėse sistemose ši užduotis dar sudėtingesnė, nes kuriant programinę įrangą reikia kuo labiau taupyti resursus.

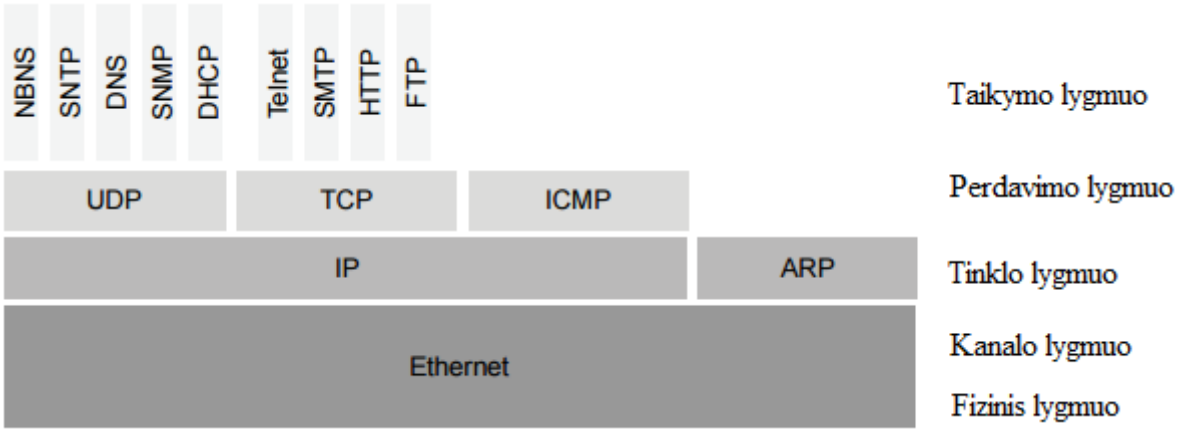
Pastaraisiais metais pradeda įsitvirtinti sąvoka „Daiktų internetas“, kuris apima bet kokį daiktą, kuris yra prijungtas prie interneto. Daiktų internetas leidžia valdyti įvairius įrenginius per jau įsitvirtinusią infrastruktūrą – tai yra internetą. Ši rinka labai sparčiai plečiasi ir ekspertų teigimu iki 2020 metų gali išsiplėsti iki 30 milijardų įrenginių. Dėl atpigusios galimybės įterptinę sistemą prijungti prie interneto, prie jo pradėta jungti viskas, nuo laikrodžio, automatiškai atnaujinančio laiką iki ištiso namo, kuriame internetu galima valdyti viską, nuo šviesų iki temperatūros nustatymų.

Tolesnei šios rinkos plėtrai reikalingas tolesnis technologijos vystymasis. Reikalinga siekti kuo pigesnių, mažiau galios naudojančių įrenginių, kurie gali patenkinti vis labiau augančius galios reikalavimus. Įtaisai turi būti vis galingesni, o kaina turi nedidėti, kas nėra lengvai pasiekiama. Taip pat, reikia siekti kuo mažesnių matmenų bei kainos modulių, kurių pagalba įterptines sistemas galima prijungti prie interneto. Šalia šių modulių turi būti daug resursų nereikalaujanti programinė įranga, palaikanti TCP/IP steką. [1] [2]

2. ETHERNET TECHNOLOGIJA

2.1. Paketų apgaubimas

Paketų apgaubimas remiasi interneto protokolų rinkiniu ir jo lygmenimis. Kiekvienas lygmuo yra atsakingas už tam tikrą funkcionalumą. Pavyzdžiui, fizinis lygmuo yra atsakingas už elektrinį bitų perdavimą per ryšio linijas. Kiekvienas aukštesnysis lygmuo panaudoja žemiau esančius nepriklausomai vienas nuo kito, tai yra lygmenys nepersidengia savo funkcijomis.



1 pav. Interneto protokolų rinkinys [3]

Dažniausiai lygmenys atlieka šias funkcijas:

- Taikymo lygmuo (iš angl. application layer) – duomenys užkoduojami;
- Perdavimo lygmuo (iš angl. transport layer) – duomenys suskaidomi į mažesnius gabalus, pridedama TCP prievado informacija;
- Tinklo lygmuo (iš angl. network layer) – pridedama IP adresų informacija, kuri pažymi kur ir iš kur keliauja duomenys;
- Kanalo lygmuo (iš angl. data link layer) – pridedama MAC (iš angl. Media Access Control – terpės prieigos kontrolė) adresų informacija, kuri specifikuoja iš kurios ir į kurią aparatinę įrangą keliauja duomenys.
- Fizinis lygmuo (iš angl. Physical Layer) – apibūdina jau pačios bitų sekos perdavimą per fizinę terpę tarp mazgų.

Pavyzdžiui, turime duomenis *Pastraipa1* ir *Pastraipa2.* Taikymo žingsnyje juos užkoduojame taip, kad gavėjas galėtų juos interpretuoti, šiuo atveju HTML kalba: * Pastraipa1 Pastraipa2 *.

Perdavimo lygmenyje siunčiami duomenys suskaidomi į mažesnius gabalus, vadinamus paketais, ir sunumeruojami eilės tvarka. Kiekvienam gabalui priskiriama prievado informacija, dažniausiai pagal tai, kokia programa naudos duomenis.

Perdavimo antraštė	Duomenys
:80 | 1/2	Pastraipa1
:80 | 2/2	Pastraipa2

Tinklo lygmenyje pridedama siuntėjo bei gavėjo IP adresų informacija. Kartu su prievado informacija IP adresai sudaro taip vadinamąsias siuntimo bei gavimo jungtis (iš angl. socket). Gavėjo IP adresas rašomas pirmiau, tada – siuntėjo.

Tinklo antraštė	Perdavimo antraštė	Duomenys
192.168.1.20 192.168.1.5	:80 | 1/2	Pastraipa1
192.168.1.20 192.168.1.5	:80 | 2/2	Pastraipa2

Galiausiai sąsajos lygmenyje prie jungčių informacijos pridedami gavėjo bei siuntėjo MAC adresai.

Kanalo antraštė	Tinklo antraštė	Perdavimo antraštė	Duomenys
11-22-33-44-55 FF-EE-DD-CC-BB	192.168.1.20 192.168.1.5	:80 | 1/2	Pastraipa1
11-22-33-44-55 FF-EE-DD-CC-BB	192.168.1.20 192.168.1.5	:80 | 2/2	Pastraipa2

Pridedant visą šią informaciją siunčiami duomenys pasiekia savo paskyrimo vietą (dėl jungties suteikiamos informacijos), randa tinkamą tinklo sąsajos įrenginį (dėl MAC adreso), randa tinkamą taikymą (dėl prievado informacijos), gali būti surinkti reikiama tvarka (dėl perdavimo antraštėje suteikiamos informacijos) ir perduoda norimą informaciją (dėl tinkamo užkodavimo taikymo lygmenyje).

Kiekviename lygmenyje naudojami adresai nėra susiję vienas su kitu. Šiame pavyzdyje, TCP paketas naudoja prievado numerį, kuris paprastai priskiriamas priklausomai nuo to, koks protokolas naudojamas taikymo lygmenyje. IP priskiria IP adresą, kuris dinamiškai priskiriamas iš visų galimų interneto adresų, o MAC paketas naudoja MAC adresą, kuris priskiriamas kiekvienam tinklo įrenginiui. [3] [4]

2.2. Ethernet paketo forma

Paprasčiausias 10/100 Ethernet paketas susideda iš šių iš eilės einančių dalių:

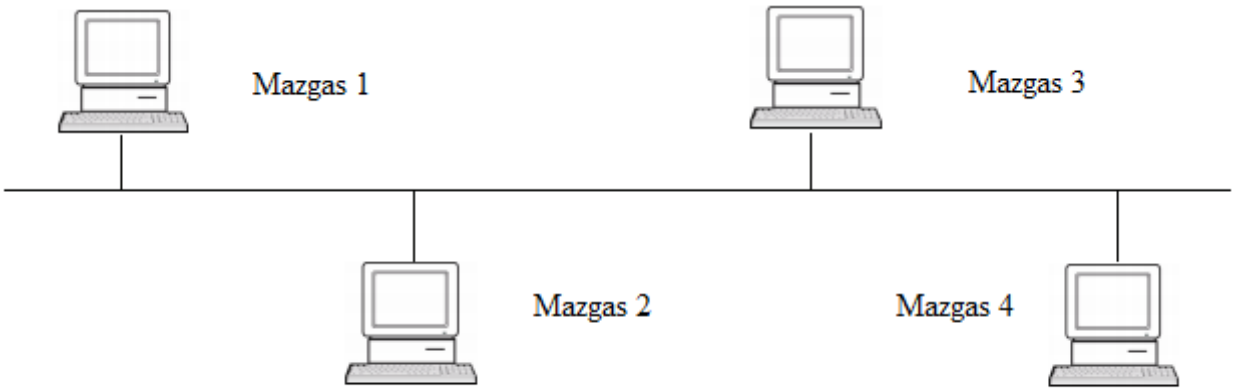
- Preambulės: 7 baitai 0x85 dvejetainiu kodu, tai yra 01010101. Ji skirta tam, kad gavėjas spėtų sureaguoti į informacijos gavimą prieš gaunant pagrindinę informaciją.
- Paketo pradžios skyriklis: dvejetainis skaičius 10101011 reiškiantis paketo pradžią.

- Gavėjo adresas: 6 baitų gavėjo MAC adresas.
- Siuntėjo adresas: 6 baitų siuntėjo MAC adresas.
- Ilgis/tipas: jei šioje dviejų baitų dalyje skaičius yra mažesnis nei 1500 jis reiškia siunčiamos informacijos ilgį, jei didesnis – tipą. Dažniausiai naudojamų tipų skaičiai: IPv4 = 0800h; IPv6 = 86DDh; ARP = 0806h; RARP = 8035h.
- Informacija: mažiausias informacijos kiekis 46 baitai, didžiausias – 1500. Jei informacijos kiekis nesiekia 46 baitų, tada ji specialiai užpildoma, kad jį pasiektų.
- Paketo patikrinimo seka: 4 baitų ilgio seka, kuri naudojama klaidoms aptikti. [3] [4]

2.3. Daugkartinės prieigos su nešlio kontrole ir konfliktų aptikimu metodas

Originaliai Ethernet buvo sukurtas naudoti bendros magistralės topologijoje. Šioje topologijoje, kiekvienas mazgas turi tą pačia prieigą prie bendros magistralės, bet tik vienas iš jų gali siųsti informaciją vienu metu. Jei keli mazgai siunčia informaciją vienu metu ji sugadinama ir prarandama. Tam kad informacija būtų perduodama be klaidų buvo sukurtas CSMA/CD (iš angl. Carrier Sense Multiple Access with Collision Detect - daugkartinės prieigos su nešlio kontrole ir konfliktų aptikimu metodas) metodas. Naudojant šį metodą galima pakaitinio dvipusio ryšio veika.

Ethernet mazgas, prieš pradėdamas siuntimą, turi nustatyti ar magistralė yra aktyviai naudojama, ar ne (nešlio kontrolė). Jeigu ji yra aktyviai naudojama, mazgas turi palaukti, kol bus aptiktas tuštumas, tada palaukti nustatytą laiko tarpą ir tik tada gali pradėti savo siuntimą. Šis nustatytas laiko tarpas priklauso nuo naudojamo greičio.



2 pav. Bendros magistralės topologija [3]

Tačiau galima situacija, kai du ar daugiau mazgų laukia magistralės ištuštėjimo ir tokiu atveju pradeda siuntimą tuo pat metu. Tokiu atveju suveikia konfliktų aptikimas. Blogiausiu atveju turime du mazgus, kurie nori perduoti informaciją, dviejuose bendros magistralės topologijos galuose ir vienas mazgas pradeda siuntimą esant mažiausiam laiko tarpui iki signalo iš kito mazgo gavimo.

Pavyzdžiui, paveikslėlyje „Mazgas 1“ pradeda informacijos siuntimą praeina kažkoks laiko tarpas, kol šis signalas magistrale pasiekia „Mazgą 4“. Prieš pat informacijos pasiekimą „Mazgas 4“ išsiunčia savo informaciją ir tuoj pat aptinka konfliktą ir išsiunčia specialų blokavimo signalą. Dabar šis blokavimo signalas turi pereiti visą magistralę, kol pasieks „Mazgą 1“ ir šis galės aptikti konfliktą palygindamas išsiųstą ir gautą informaciją. Tai reiškia, kad reikia dviejų pilnų signalo sklidimo laikų per magistralę, tam kad užtikrinti visų magistralės mazgų konfliktų aptikimą. Todėl yra didžiausias leidžiamas magistralės ilgis, jis nustatomas pagal tai kiek trunka mažiausio paketo transliavimas.

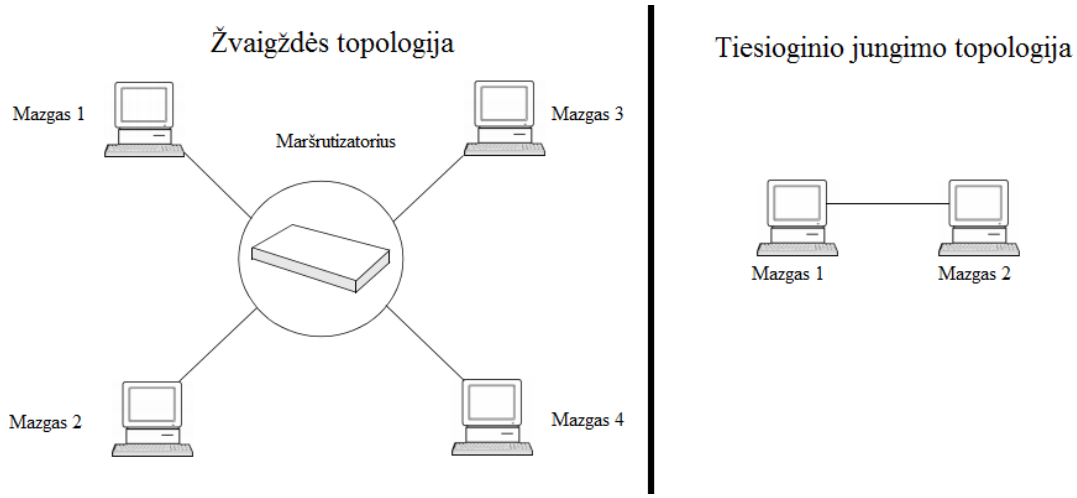
Galiausiai, reikalingas metodas, kuriuo būtų nustatoma kada mazgai gali retransliuoti savo informaciją, kitaip konfliktai niekada neišsispręstų. Ethernet panaudoją vadinamą dvejetainį eksponentinį atsitraukimo algoritmą. Jo metu pirmiausia kiekvienas mazgas atsitiktinai pasirenka laukimą iki retransliavimo (nuo 0 iki 1), jei retransliavimo metu vėl aptinkamas konfliktas, kiekvienas mazgas padvigubina laukimo intervalą (dabar gali būti nuo 0 iki 3) ir vėl atsitiktinai pasirenkamas naujas laukimas iki retransliavimo. Toks intervalo dvigubėjimas tęsiamas, kol nebeaptinkama konflikto arba pasiekiama 16 bandymų riba. Pasiekus ribą paketas nebesiunčiamas ir pranešama apie didelį konfliktų skaičių. [3] [4]

2.4. Vienalaikio dvipusio ryšio veika

Ankstyvieji Ethernet tinklai naudojo bendrą magistralės topologiją, tačiau dabartiniuose ji naudojama vis rečiau. Dažniau naudojama tiesioginio sujungimo arba žvaigždės topologijos.

Kadangi abiejose topologijose kiekvienas mazgas turi tik vieną sujungimą su kitu mazgu galima vienalaikio dvipusio ryšio veika, kadangi daugkartinės prieigos su nešlio kontrole ir konfliktų aptikimu metodo naudoti nereikia.

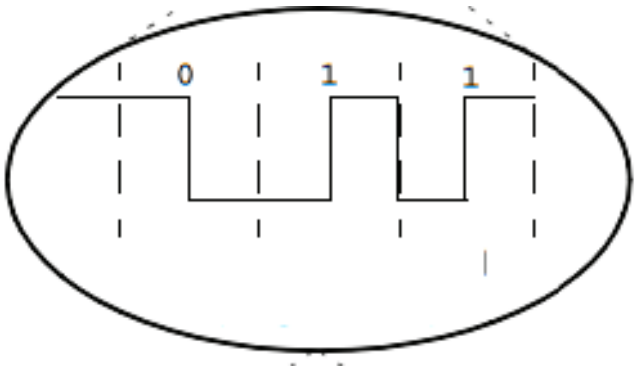
Veikiant vienalaikio dvipusio ryšio veikai tinklo išeiga padvigubinama (pavyzdžiui, iš 10 Mbit/s į 20 Mbit/s).



3 pav. Šiuolaikinės topologijos [3]

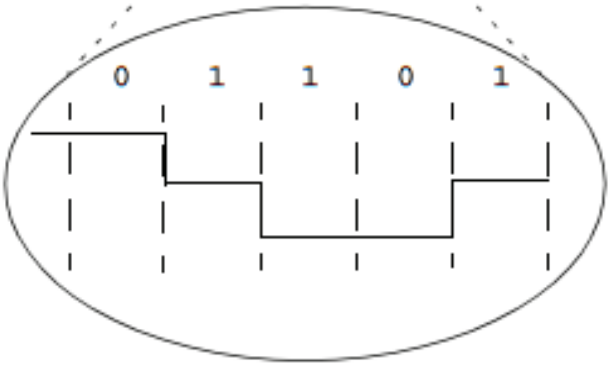
2.5. Paketo transportavimas kabeliu

Paketo transportavimas kabeliu skiriasi priklausomai nuo spartos. Naudojant 10 Mbit/s spartą naudojamas vadinamasis Mančesterio kodavimas. Ethernet aplikacijai Mančesterio kodavimas veikia taip: koduojamų duomenų loginis ‘0’ perduodamas signalo pakitimu iš aukšto lygio į žemą, viduryje bitui skirtos trukmės, o loginis ‘1’ – atvirkščiai, signalas keičiamas iš žemo lygio į aukštą. Mančesterio kodavimas naudojamas todėl, kad jis užtikrina aukštą patikimumą ir galimybę išgauti sinchronizacinio signalo dažnį iš duomenų srauto signalo.



4 pav. Mančesterio kodu užkoduotas signalas [3]

Naudojant 100 Mbit/s spartą naudojamas kitas kodavimas, dėl įvairių dėl didesnės spartos iškylančių problemų. Čia naudojamas vadinamasis trijų lygmenų perėjimo kodavimas (iš angl. MLT3 – Multi-Layer Transition 3). Jo veikimas toks: loginiai ‘1’ koduojami kaip signalo pokytis tarp trijų lygmenų (-1, 0 ir +1) į artimiausią lygmenį ir visad ta pačia tvarka (-1, 0, +1, 0, -1 ir t.t.). Loginis ‘0’ koduojamas kaip signalo nekitimas.



5 pav. Trijų lygmenų perėjimo kodavimu užkoduotas signalas [3]

Papildomai trijų lygmenų perėjimo kodavimui 100 Mbit/s spartai perduoti naudojamas papildomas loginis kodavimas vadinamas 4B/5B, kuriame 4 bitų duomenų blokas verčiamas penkiais bitais. Jis reikalingas tam, kad esant ilgai loginių ‘0’ sekai galima būtų išgauti sinchronizacinio signalo dažnį, nes dėl naudojamo kodavimo ypatybių tokia seka reiškia ilgai nesikeičiantį signalą, kurio metu siuntimo ir gavimo mazgai gali išsisinchronizuoti. Taip pat, įvedami specialios bitų sekos reiškiančios tam tikras komandas.

1 lentelė. 4B/5B kodavimas

Reikšmė	Apibūdinimas
11110	Duomenys ‘0h’
01001	Duomenys ‘1h’
10100	Duomenys ‘2h’
10101	Duomenys ‘3h’
01011	Duomenys ‘4h’
01110	Duomenys ‘5h’
01111	Duomenys ‘6h’
10010	Duomenys ‘7h’
10011	Duomenys ‘8h’
10110	Duomenys ‘9h’
10111	Duomenys ‘Ah’
11010	Duomenys ‘Bh’
11011	Duomenys ‘Ch’
11100	Duomenys ‘Dh’
11101	Duomenys ‘Eh’
11111	Duomenys ‘Fh’
11000	Siuntimo pradžia (1 dalis)
10001	Siuntimo pradžia (2 dalis)
01101	Siuntimo pabaiga (1 dalis)
00111	Siuntimo pabaiga (2 dalis)
00100	Klaida

Kaip matome iš lentelės visuose užkoduotų duomenų signaluose yra bent po 2 loginius vienetus. Tai reiškia, kad esant bet kokiems duomenims užtikrinamas signalo kitimas ir taip užtikrinama sinchronizacija. [3] [4]

2.6. Automatinės derybos

Automatinės derybos – procesas, kurio metu du mazgai pasikeičia informacija apie savo galimybes (greitį, vienalaikio dvipusio ryšio veikos palaikymą ir t.t.) tam kad išsiaiškinti geriausią

įmanomą ryšį.

Automatinės derybos vyksta panaudojant greituosius jungties impulsus (iš angl. Fast Link Pulses). Mazgai, kurie nepalaiko automatinių derybų, šiuos impulsus priima kaip įprastus impulsus ir juos ignoruoja. Mazgai, kurie jas palaiko, bet negauna jokios informacijos per greituosius jungties impulsus automatiškai naudoja žemiausio greičio nustatymus. Didžiausias jungties impulsų skaičius svyruoja ir nėra numatomas IEEE 802.3 standarto, tačiau kiekvienas automatines derybas palaikantis įrenginys būtinai turi palaikyti bazinį automatinių derybų kodinį žodį, kurio ilgis yra 16 bitų.

Šiame žodyje pirmuosiuose penkiuose bituose yra aprašoma LAN technologija (Ethernet atveju – ‘10000’. Kituose aštuoniuose – mazgo galimybės. 6-10 bitai apibūdina ryšio greitį bei vienalaikio dvipusio ryšio veikos galimybes. 11 – siuntimo pauzės galimybę, 12 – asimetrinės pauzės galimybę, 13 – naudojama papildomai informacijos buvimo ar nebuvimo galimybei (naudojama tik gigabitiniuose mazguose). 14 bitas aprašo klaidos buvimą, 15 – signalizuoja apie bazinio automatinio derybų kodinio žodžio gavimą, o 16 – apie papildomai toliau sekančius bitus, kurie nėra būtini. [3] [4]

3. HTTP PROTOKOLAS

3.1. HTTP protokolo įvadas

HTTP – protokolas, kuris yra viena iš pamatinių duomenų perdavimo internetu sudedamųjų dalių. Paprastai tariant, tai protokolas skirtas keistis ar perduodi hipertekstą. Hipertekstas – tekstas, kurio pavienės dalys yra siejamos specialiomis nuorodomis – saitais, kurių pagalba galima patekti iš vienos dalies į kitą ir jas peržiūrėti bet kuria saitais susieta eile.

HTTP veikia kaip užklausimo - atsakymo protokolas. Klientas pateikia HTTP užklausą serveriui ir serveris pateikia numatytą pranešimą, pavyzdžiui, HTML (iš angl. Hypertext Markup Language – hiperteksto žymėjimo kalba) failą ir atsako klientui su atsakymo eilute. Atsakymo eilutėje yra nurodoma pranešimo protokolo versija, sėkmės arba klaidos kodas bei serverio informacija.

HTTP ryšys dažniausiai vyksta per TCP/IP (iš angl. Transmission Control Protocol/Internet Protocol – perdavimo kontrolės protokolas/interneto protokolas) ryšį per numatytąją TCP 80 prievadą (iš angl. port). Šio protokolo naudojimas nėra absoliuti būtinybė, bet koks protokolas, kuris gali užtikrinti patikimą perdavimą, gali būti naudojamas HTTP ryšiui. [5]

3.2. HTTP užklausimo ir atsakymo žinutės

Užklausimo žinutė susideda iš:

- Užklausimo eilutės (pvz. *GET /images/logo.png HTTP/1.1* užklausa paprašo serverio resurso (šiuo atveju paveikslėlio) pavadinimu *logo.png* iš *images* direktorijos);
- Užklausimo antraštės lauko (pvz. Host: www.vu.lt užklausa nurodo serverio domeną)
- Tuščios linijos;
- Nebūtinos pagrindinės žinutės dalies, kurioje gali būti patalpinama papildoma informacija apie klientą, norimą atsakymą ir panašiai. (pvz. *Accept: text/plain* užklausos eilutė pasako serveriui, kad priimtinas atsakymas yra tik paprasto teksto tipo).

Atsakymo žinutė susideda iš:

- Padėties eilutės, kuri susidaro iš padėties kodo ir priežasties žinutės (pvz. *HTTP/1.1 200 OK* atsakymas parodo, kad kliento užklausa yra sėkminga);
- Atsakymo antraštės lauko (pvz. *Content-Type text/html* atsakymas nurodo, kad siunčiama informacija yra tekstinio HTML formato)
- Tuščios linijos;
- Nebūtinos pagrindinės žinutės dalies (pvz. HTML puslapio). [5]

3.3. HTTP užklausų metodai

Klientui siunčiant užklausą serveriui pirmoje užklausos eilutėje yra nurodomas jos metodas. Jis parodo kliento norimą veiksmą, kurį reikia atlikti su serverio resursais.

HTTP/1.1 specifikacijoje yra apibrėžti šie metodai:

- GET – iš angliško žodžio get – gauti. Ši užklausa yra dažniausiai naudojama. Ji pareikalauja tam tikro, specifinio resurso, kuris nurodomas užklausos metu;
- HEAD – iš sutrumpinto angliško žodžio header – antraštė. Ši užklausa yra panaši į GET, bet reikalaujama tik atsakymo antraštė, o ne visas dokumentas;
- POST – iš angliško žodžio post – skelbti. Ši užklausa taip pat panaši į GET, tačiau jos metu siunčiama papildoma informacija, pavyzdžiui, kliento specifikuota žinutė;
- PUT – iš angliško žodžio put – padėti – naudojama informacijos saugojimui serveryje;
- DELETE – iš angliško žodžio delete – ištrinti – naudojama specifikuotam resursui ištrinti;
- TRACE – iš angliško žodžio trace – sekti – grąžina gautą užklausą, galima naudoti patikrinimui, kokių duomenų prideda tarpiniai serveriai;
- OPTIONS – iš angliško žodžio options – pasirinkimai – grąžina sąrašą serverio palaikomų metodų;
- PATCH – iš angliško žodžio patch – lopyti – specifikuotam resursui pritaiko dalinius pakeitimus.

HEAD, GET, OPTIONS ir TRACE metodai yra vadinami saugiais metodais, nes jie naudojami tik informacijos išgavimui ir nepakeičia serverio būsenos. Dėl to šie metodai neturi sukelti jokių pašalinių efektų. POST, PUT, DELETE ir PATCH metodai yra naudojami ir kitiems veiksmams, ne tik informacijos išgavimui, todėl gali sukelti nenumatytų pašalinių efektų serveryje. [5]

4. HTML DOKUMENTAI

4.1. HTML kalbos įvadas

HTML (iš angl. Hypertext Markup Language – hiperteksto žymėjimo kalba) – viena iš kertinių kalbų naudojamų internetinių puslapių kūrimui. Internetinės naršyklės perskaito HTML failus ir paverčia juos į matomus internetinius puslapius. HTML elementai yra pagrindinė HTML puslapių dalis. HTML elementų pagalba atsiranda galimybė įterpti paveiksliukus ir kitus objektus. Jie yra apibūdinami naudojant specialias žymes, kurios vartotojui nėra matomos. [6]

4.2. HTML elementai

HTML elementai yra HTML puslapio sudedamosios dalys. Jie yra apibūdinami HTML dokumente naudojant žymes. Paprastai HTML elementas susideda iš trijų dalių:

- Pradžios žymės - *<žymė>*;
- Turinio, kuris yra visoje žymės galiojimo srityje;
- Pabaigos žymės - *</žymė>*.

Pradžios žymė papildomai gali turėti parametrų, kurie suteikia papildomos informacijos, pavyzdžiui, papildomą žymėjimą stilizacijai ar įterpto paveikslėlio vietą serveryje. Tokia žymė galėtų atrodyti taip: *<žymė parametras=”reikšmė“>*.

Paprasčiausias HTML dokumentas turi turėti keletą būtinų žymių:

- *<html>* – visą dokumentą gaubianti žymė;
- *<head>* – metainformaciją (pavadinimą ir pan.) turintis elementas;
- *<title>* – dokumento pavadinimas;
- *<body>* – vaizduojamo turinio elementas. [6]

4.3. CSS kalba

CSS (iš angl. Cascading Style Sheets – pakopiniai stilių šablonai) yra stilių šablonų kalba labai dažnai naudojama kartu su HTML. Jos pagalba yra keičiama HTML dokumento išvaizda internetinėje naršyklėje pagal HTML dokumente nurodytą žymėjimą. CSS pagrinde naudojama atskirti dokumento turinį nuo jo prezentacijos. Toks atskyrimas palengvina dokumento išvaizdos keitimą bei tos pačios išvaizdos pritaikymą keliems skirtingiems dokumentams.

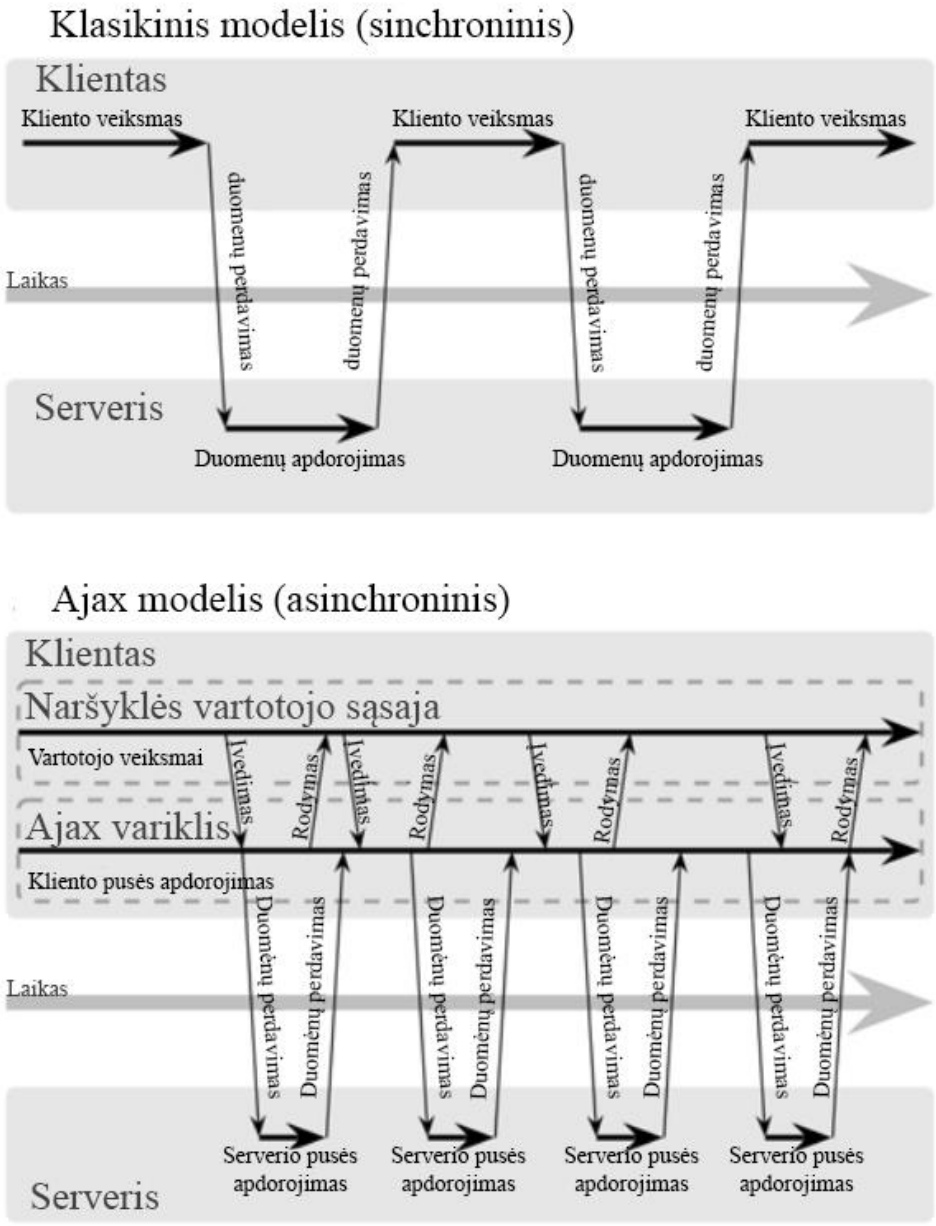
Vienas pagrindinių CSS pliusų yra paprasta jos sintaksė, nes ji tiesiog susideda iš angliškų žodžių kuriais specifikuojama įvairūs stilistiniai pasirinkimai. Stiliaus šablonas susideda iš taisyklių sąrašo. Kiekviena taisyklė susideda iš vieno ar daugiau identifikatorių ir savybių blokų.

CSS identifikatoriumi gali būti bet kuris HTML elementas ar elemento apibrėžime parinktas parametras. Norimos stiliaus savybės rašomos į skliaustus po identifikatoriaus. Savybių sintaksė yra tokia: pirmiausia yra nusakoma savybe, tada po dvitaškio įrašoma reikšmė. Pavyzdžiui, *p{color:red}* CSS eilutė nuspalvintų HTML elemento *p* tekstą esantį tarp žymių <*p*> ir </*p*> raudonai. [7]

5. AJAX MODELIS

5.1. Ajax įvadas

Ajax (iš angl. asynchronous JavaScript and XML – asinchroninis JavaScript ir XML) – internetinių puslapių kūrimo technologijų rinkinys skirtas kurti asinchroniniams internetinių puslapių sprendimams. Naudojant Ajax vietoj to, kad kaskart vartotojui atlikus kažkokį veiksmą reikia tiesiog laukti atsakymo iš serverio, Ajax varikliklis sudaro sąlygas asinchroninei sąveikai. Tokiu būdu vartotojui nereikia žiūrėti į tuščia langą ir laukti, o bendravimas su serveriu vyksta nuošaly, vartotojui nematant.



6 pav. Klasikinio ir Ajax modelių palyginimas [8]

Kiekvienas vartotojo veiksmas, kuris paprastai generuotų HTTP užklausą, panaudoja JavaScript kreipimąsi į Ajax variklį. Kiekvieną vartotojui skirtą atsakymą į užklausą, kuri nereikalauja būtino bendravimo su serveriu, pavyzdžiui, duomenų patvirtinimas, duomenų pakeitimas vartotojo atmintyje ir netgi paprasta navigacija, variklis sugeneruoja pats. Jeigu užduodant užklausą reikalingas atsakymas iš serverio, pavyzdžiui, perduodant duomenis apdorojimui, kraunant papildomą sąsajos kodą ar gaunant naujus duomenis, variklis duoda užklausas asinchroniškai, nesustabdant vartotojo sąveikos. Tokiu būdu išgaunamas iš pažiūros nenutrūkstantis programos veikimas. [8]

5.2. JavaScript

JavaScript, kartu su HTML ir CSS, yra trečioji programavimo kalba, kurią turi žinoti kiekvienas internetinių puslapių kūrėjas. Pavadinimas JavaScript yra ganėtinai apgaulingas, apart šiokių tokių sintaksinių panašumų, ši kalba absoliučiai skiriasi nuo Java programavimo kalbos. JavaScript naudojama absoliučioje daugumoje šiuolaikinių internetinių puslapių. JavaScript kodas gali veikti lokaliai, vartotojo naršyklėje, todėl naršyklė gali atsakyti į vartotojo veiksmus greitai, be atsakymo iš serverio.

JavaScript pagalba atliekami įvairūs veiksmai su duomenimis, šios kalbos galimybės yra labai plačios. JavaScript svarbiausias duomenų tipas yra objektas, jį apsibrėžiame riestiniais skliaustais, pavyzdžiui:

```
var    knyga = {
       tema: "JavaScript",
       puslapiai: "1080"
};
```

Tada į objekto kintamąjį, jo prireikus galime kreiptis atskyrę tašku, pavyzdžiui, *knyga.tema* turėtų kintamąjį "JavaScript".

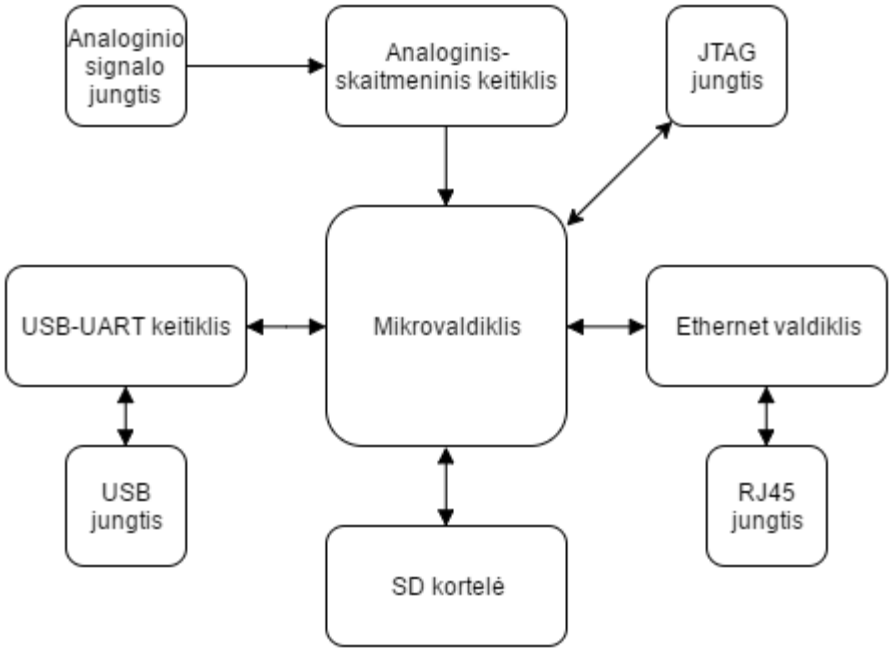
JavaScript kalba parašyti skriptai į HTML failus įterpiami panaudojant <script> žymą. [9]

5.3. XML

XML (iš angl. EXtensible Markup Language – išplečiama aprašomoji kalba) – yra bendros paskirties duomenų struktūrų ir jų turinio aprašomoji kalba, ji ganėtinai panaši į HTML. Pagrindinis XML kalbos vienetas yra elementas. Jis turi turėti vardą ir, be jo, gali turėti norimą skaičių atributų, dukterinius elementų, su elementu susijusį tekstą. Šiame darbe XML nebuvo naudojama, vietoje jos buvo naudojama HTML.

6. ĮTERPTINĖS SISTEMOS PROJEKTAVIMAS

6.1. Įterptinės sistemos blokinė schema



7 pav. Įterptinės sistemos blokinė schema

Prie mikrovaldiklio buvo numatyta prijungti RJ45, USB, analoginio signalo, JTAG jungtis bei SD kortelę. Papildomai reikėjo panaudoti USB-UART keitiklį, Ethernet valdiklį bei analoginį-skaitmeninį keitiklį. Šių dalių sujungimas su mikrovaldikliu aptariamas jiems skirtuose poskyriuose.

Suprojektuotos įterptinės sistemos tikslas yra realizuoti Web serverio funkcionalumą ir perduoti duomenis HTTP protokolu. Tuo tikslu įterptinės sistemos centre yra mikrovaldiklis, kuris veiks kaip Web serveris. Prijungtas Ethernet valdiklis su RJ45 jungtimi leidžia prie įterptinės sistemos prijungti Ethernet kabelį, kuriuo keičiamasi duomenimis su vartotoju. JTAG jungtis ir USB-UART keitiklis su USB jungtimi prijungti įvairiems veiksmams programavimo eigoje atlikti. Analoginio-skaitmeninio keitiklio su analoginio signalo jungtimi pagalba diskretizuojant analoginį signalą gaunami dinamiškai kintantys duomenys, kuriuos bus galima perduoti HTTP protokolu. SD kortelė skirta sudaryti galimybei praplėsti įterptinės sistemos atmintį.

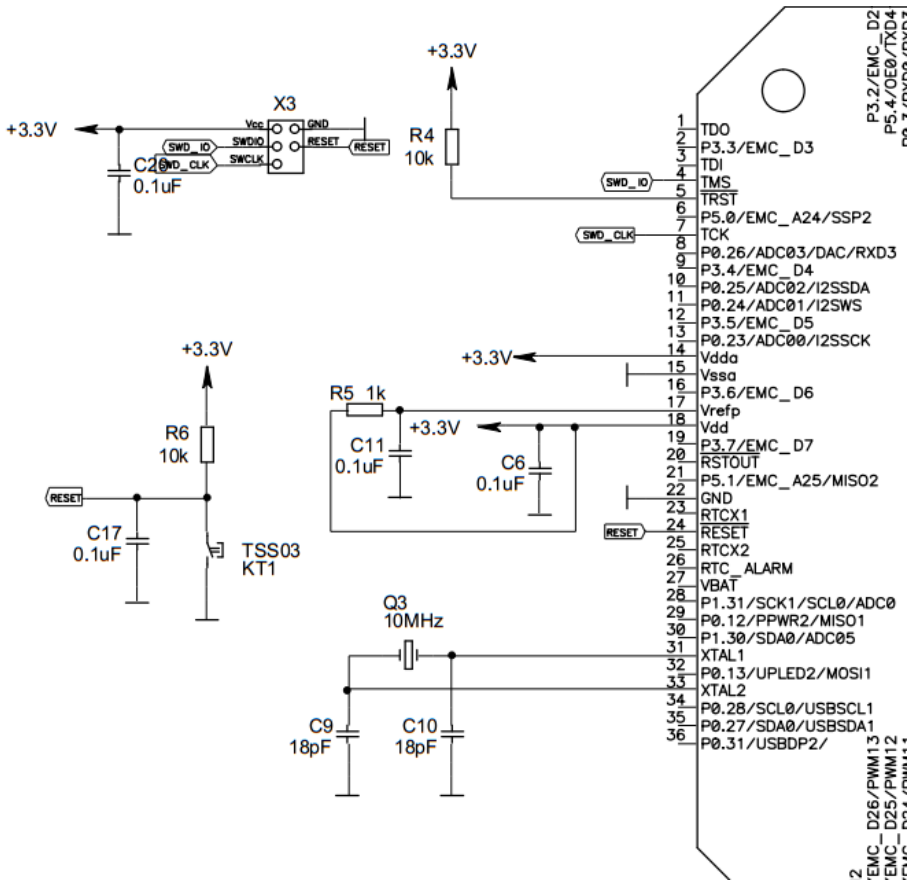
6.2. LPC1776 mikrovaldiklis

Šiame darbe naudojamas LPC1776 32 bitų mikrovaldiklis priklauso ARM (iš angl. Advanced RISC Machine – pažangus RISC (iš angl. Reduced Instruction Set Computing – supaprastintų komandų sistema) įtaisas) Cortex-M3 mikroprocesorių šeimai. Jis skirtas naudoti įterptinėse sistemose reikalaujančios didelio integracijos lygio ir mažų galios nuostolių.

ARM Cortex-M3 yra naujos kartos šerdis, kurios CPU naudoja trijų lygmenų instrukcijų

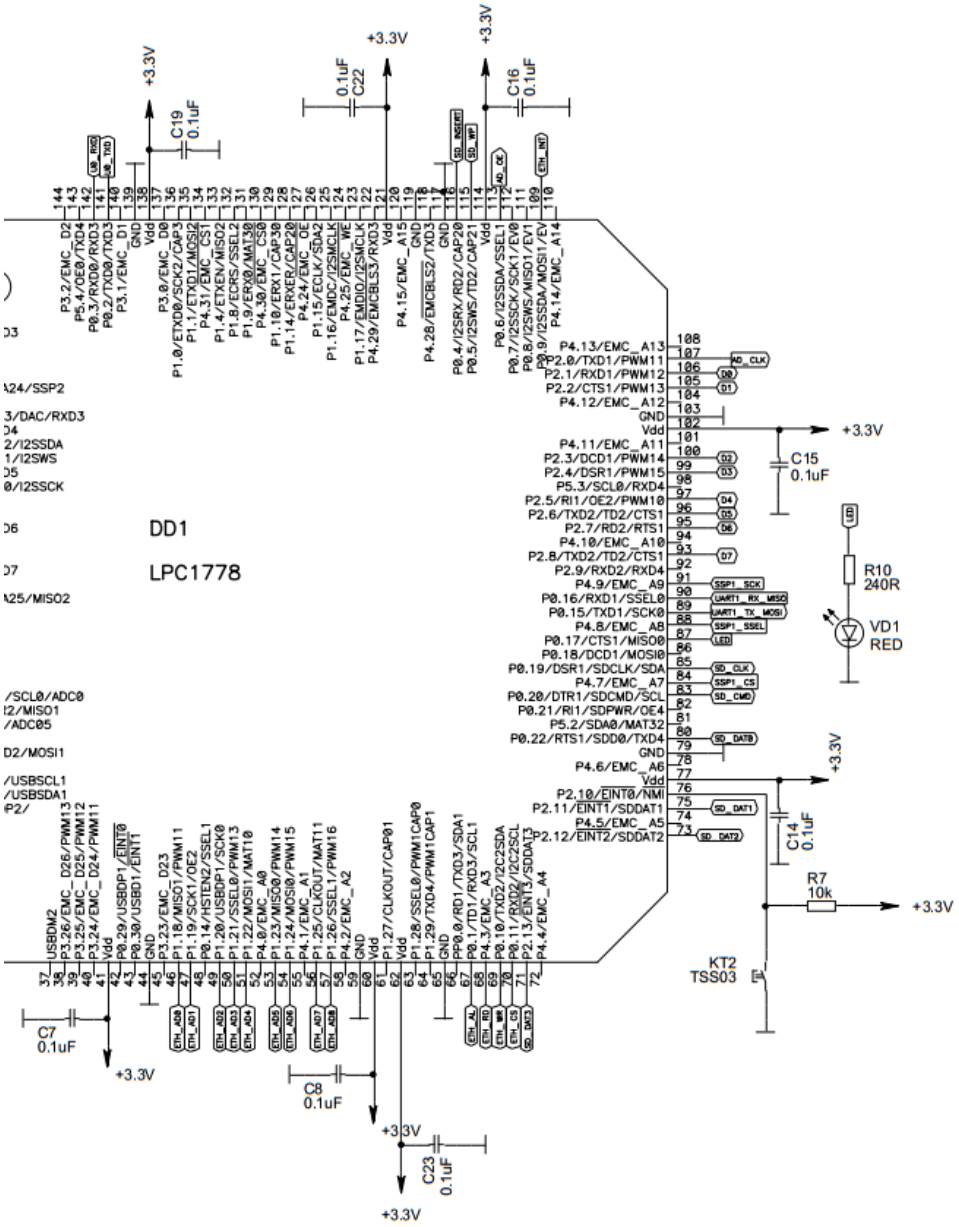
konvejeriavimo principą (iš angl. 3-stage pipeline). Trijų lygmenų konvejeriavimo principo panaudojimas padidina procesoriaus instrukcijų įvykdymo greitį. Instrukcija yra išskaidoma į tris dalis: nuskaitymą, dekodavimą ir vykdymą. Įrenginiui veikiant, kol viena instrukcija yra vykdoma, kita gali būti dekoduojama, o trečia – nuskaitoma. Taip pat naudojama Harvard architektūra su atskiromis vietinių instrukcijų bei duomenų magistralėmis ir trečia, truputį lėtesne magistrale įvesties bei išvesties įrenginių naudojimui.

LPC1776 mikrovaldiklis naudoja specializuotą flash atminties greitintuvą tam, kad užtikrintų optimalų kodo vykdymą. Mikrovaldiklis gali dirbti iki 120 MHz dažniu bei turi didelį įvesties bei išvesties įrenginių pasirinkimą: iki 512 kB flash programinę atmintį, iki 96 kB SRAM (iš angl. Static random-access memory – statinė darbinė atmintis) atmintį, iki 4032 baitų EEPROM (iš angl. Electrically Erasable Programmable Read-Only Memory – elektriškai ištrinama programuojama atmintis) atmintį, Ethernet palaikymą, USB (iš angl. Universal Serial Bus – universalioji jungtis) palaikymą, penkias UART (iš angl. Universal Asynchronous Receiver/Transmitter – universalus asinchroninis gavėjas/siųstuvas) jungtis, tris SSP (iš angl. Synchronous Serial Port – sinchroninė jungtis) valdiklius, tris I²C (iš angl. Inter-Integrated Circuit – tarpusavy integruota grandinė) magistralės sąsajas, keturis bendros paskirties skaitiklius, 165 bendros paskirties išėjimo bei įėjimo kojeles ir daug kitų ne taip dažnai naudojamų įrenginių.



8 pav. Kairės LPC1776 mikrovaldiklio pusės sujungimo schema

Prie mikrovaldiklio buvo prijungtas išorinis taktinio dažnio šaltinis – 10 MHz kvarcinis rezonatorius. Sekanti suprojektuota mikrovaldiklio dalis – perkrovimo arba „reset“ mygtukas. Jo pagalba, pasiuntus žemo lygio signalą į reikiamą mikrovaldiklio kojelę, jis persikrauna. Taip pat numatyta JTAG-SWD (iš angl. Joint Test Action Group – Serial Wire Debug – JTAG asociacijos nuoseklaus jungimo klaidų radimo sąsaja) jungtis, per kurią, prijungus specialų prietaisą, galima ieškoti klaidų, atsirandančių programavimo procese.



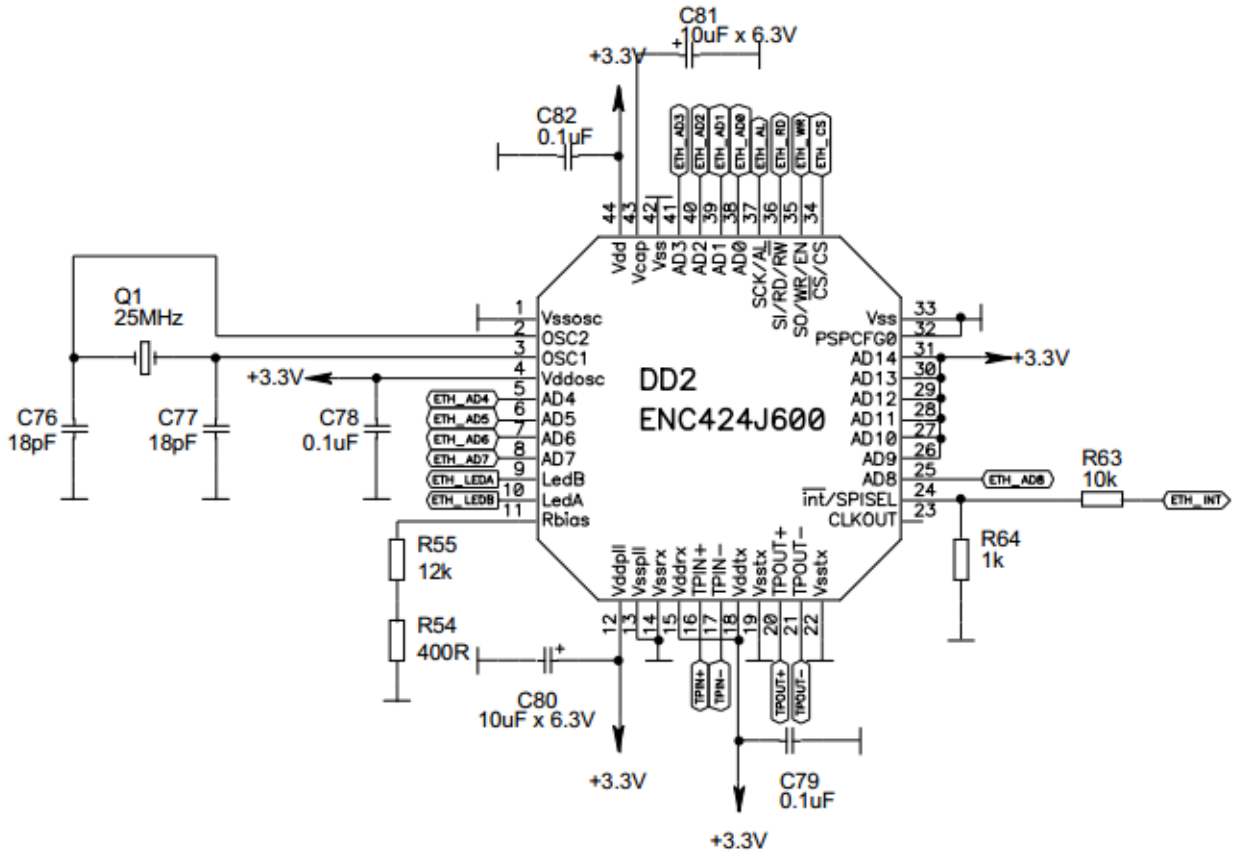
9 pav. Likusios LPC1776 mikrovaldiklio dalies sujungimo schema

Prie mikrovaldiklio buvo prijungtas dar vienas mygtukas, kurio pagalba perkrovimo metu įtaisas yra perjungiamas į programavimo rėžimą, vadinamą ISP (iš angl. In-System programming – programavimas sistemos viduje). Taip pat prijungtas šviesos diodas, kuris reikalingas programavimo procese bandant įvairius funkcionalumus. Kitos jungtys, tai papildomų jungčių ir integrinių grandynų kojelių prijungimas, kuris bus aptariamas kituose poskyriuose.

Taip pat, prie mikrovaldiklio bei kitų įrenginių V_{DD} ar V_{CC} (maitinimo) kojelių buvo prijungti mažos talpos kondensatoriai. Šie kondensatoriai atlieka dvejopą funkciją: filtruoja kintamos įtampos svyravimus, ateinančius iš maitinimo šaltinio, ir veikia kaip elektrinio krūvio talpyklos, iš kurių yra kompensuojamas staigus srovės poreikis. [13]

6.3. ENC424J600 Ethernet valdiklis

Šiame darbe papildomai su LPC1776 mikrovaldikliu yra naudojamas 10/100 Mbit Ethernet valdiklis ENC424J600. Jis su mikrovaldikliu bendrauja per SPI (iš angl. Serial Peripheral Bus – nuosekli periferinė magistralė) sąsają. Valdiklis palaiko vieną RJ45 jungtį prie kurios jungiamas Ethernet kabelis. Taip pat numatytas automatinių derybų su kitais prietaisais bei pakaitinio ar vienalaikio dvipusio ryšio veikų palaikymas.



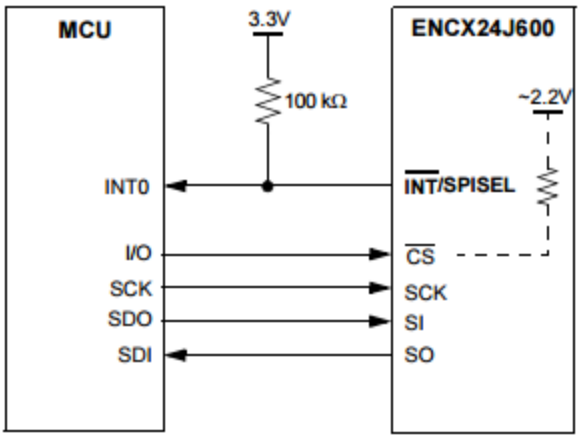
10 pav. ENC424J600 Ethernet valdiklio sujungimo schema

Ethernet valdiklis per keturias kojeles TPIN+, TPIN-, TPOUT+ ir TPOUT- gauna arba siunčia signalą per RJ45 jungtį. Informacija siunčiama panaudojant diferencinę įtampą ant kojelių poros taip formuojant Ethernet signalą.

Iš viso buvo sujungta 10 kojelių su naudojamu mikrovaldikliu. INT kojelė signalizuoja trūkio įvykį, kai ja siunčiamas žemo lygio signalas. Kitos 9 kojelės naudojamos sąsajai su mikrovaldikliu. Lankstumo dėlei galima pasirinkti SPI (iš angl. Serial Peripheral Interface – nuosekli periferinė sąsaja) arba PSP (iš angl. 16-bit parallel slave port – 16 bitų lygiagreti vedamųjų tipo jungtis) sąsajas.

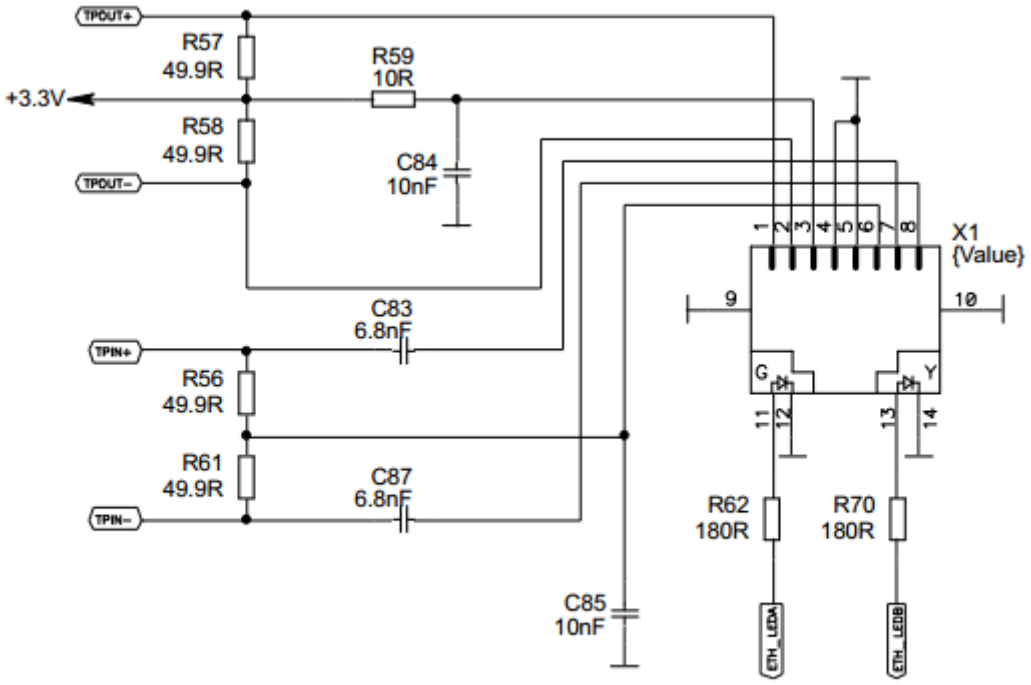
Pasirinkimas vykdomas įjungimo metu per INT kojelę. Jei ji siunčia aukštą lygį įjungimo metu – naudojama SPI sąsaja, kitu atveju PSP. Šiuo atveju naudojama bus SPI sąsaja, todėl reikalingos CS, SCK, SI bei SO kojelės.

Taip pat, šio valdiklio veikimui reikalingas išorinis taktinio dažnio šaltinis, todėl buvo prijungtas 25 MHz kvarcinis rezonatorius.



11 pav. SPI sąsajos jungtys [15]

6.4. RJ45 jungtis



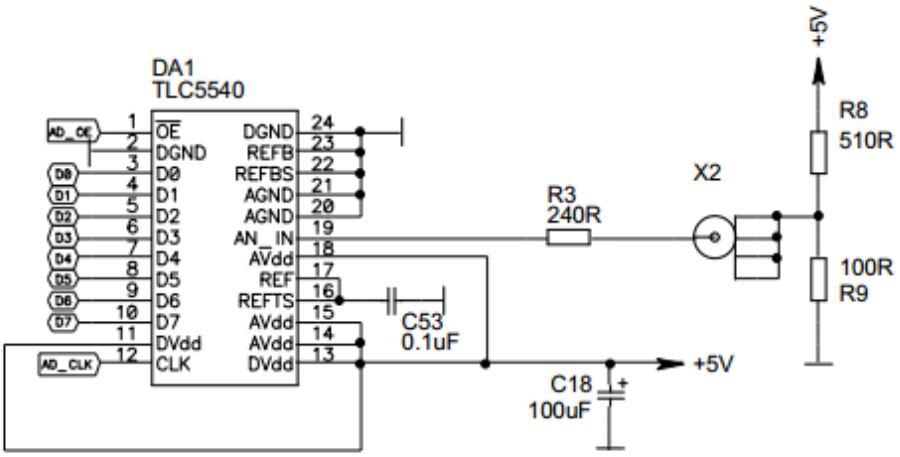
12 pav. RJ45 jungties schema

RJ45 jungtis naudojama Ethernet kabeliui prijungti. Kaip ir buvo minėta ankščiau, ši jungtis sujungiama su Ethernet valdikliu keturiomis kojelėmis. Papildomos dvi kojelės prijungtos LED

šviesėlių funkcionalumui vykdyti – perteikti vizualiai informacijai apie duomenų siuntimą ir gavimą.

Keturios signalo kojelės buvo sujungtos pagal Ethernet valdiklio gamintojo rekomendaciją. Prie TPIN kojelių poros, kurios naudojamos signalo priėmimui, buvo prijungti du 49,9 Ω rezistoriai ir du 6,8 nF kondensatoriai, kurie kartu sudaro RC filtrus, skirtus signalo triukšmo mažinimui. Prie TPOUT kojelių poros, kaip ir prieš tai, buvo prijungti du 49,9 Ω rezistoriai, kurie reikalingi signalo atspindžiams panaikinti. Per papildomą 10 Ω rezistorių centrinis perdavimo transformatoriaus kontaktas (angl. center tap), esantis RJ45 jungtyje, sujungiamas su maitinimo srove. Ši per transformatorių tekanti srovė TPOUT kojelių poros pagalba generuoja perdavimo signalą. Šis rezistorius sumažina šilumos, kurią tektų išsklaidyti ENC424J600 valdiklio Ethernet fizinio lygmens daliai.

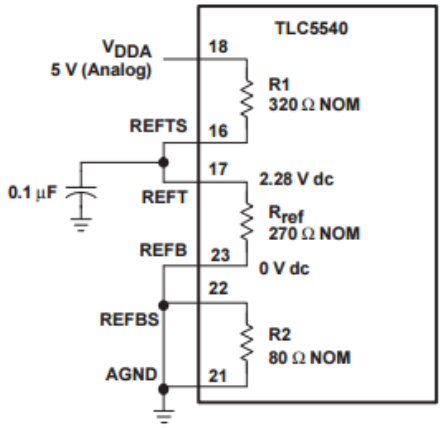
6.5. TLC5540 8 bitų analoginis-skaitmeninis keitiklis



13 pav. TLC5540 analoginio-skaitmeninio keitiklio schema

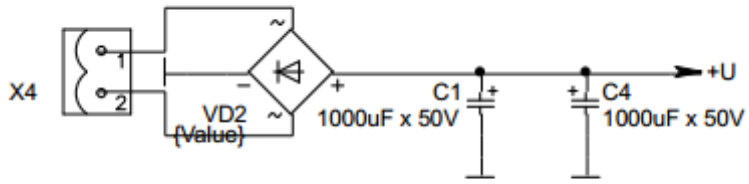
Šis keitiklis komparatorių pagalba verčia analoginį signalą į skaitmeninį. Didžiausia šio keitiklio sparta yra 40 milijonų imčių per sekundę. Prie keitiklio buvo prijungta jungtis skirta analoginio signalo įvedimui. Į mikrovaldiklį buvo nuvestos 8 linijos, kurios skirtos skaitmeninių duomenų perdavimui bei dvi papildomos linijos CLK ir OE, kurios skirtos atitinkamai sinchronizacinio dažnio įvedimui bei duomenų srauto išvesčiai įjungti bei išjungti.

Santykinių įtampų nustatymui buvo remtasi gamintojo rekomendacijomis. Panaudojant vidinius rezistorius bei tik vieną išorinį 0,1 µF talpos kondensatorių nustatomos analoginio signalo ribos – 0 – 2,28 V.



14 pav. Santykinių įtampų kojelių sujungimas pagal gamintojo rekomendacijas [16]

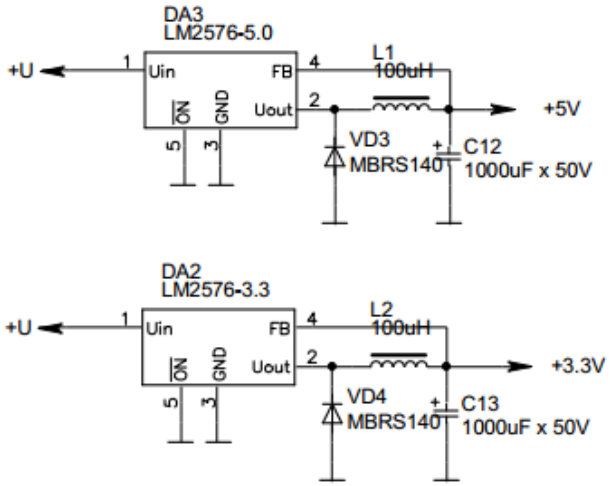
6.6. Maitinimas



15 pav. Maitinimo schema šalia jungties

Prie maitinimo jungties prijungtas diodinis tiltelis, kuris veikia kaip kintamos maitinimo srovės lygintuvas. Po jo jungiami du elektrolitiniai 1000 µF talpos kondensatoriai, kurie reikalingi diodinio tiltelio pagalba gautos išlygintos srovės pulsacijoms sumažinti, taip gaunant lygesnę nuolatinę srovę.

Toliau jungiami 5 V ir 3,3 V LM2576 įtampos reguliatoriai.



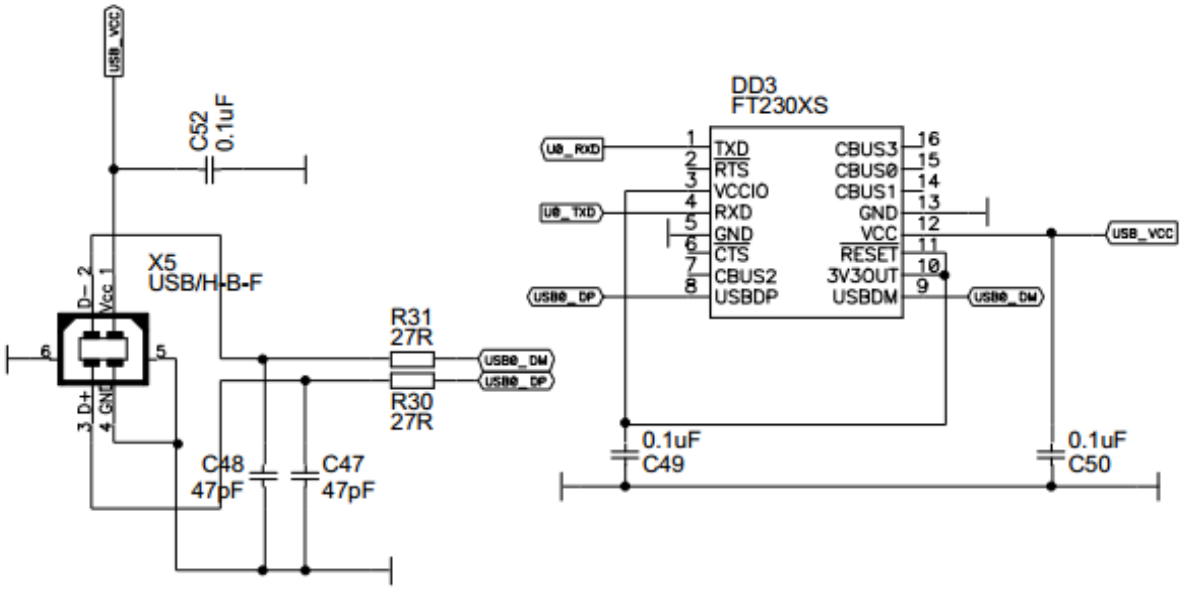
16 pav. Įtampos reguliatorių schemos

Šie reguliatoriai veikia ciklišku rėžimu, kur perjunginėjant srovės padavimą per ritę ji cikliškai įkraunama ir iškraunama, taip sumažinama išėjimo įtampos vertė palyginus ją su įėjimo įtampa. Šis rėžimas yra patrauklus tuo, kad jeigu ritė ir srovės perjungimo įrenginiai yra idealūs, tai įtampos vertės keitimo metu neprarandama nė kiek galios.

Įtampos reguliatorių veikimui reikalingi trys išoriniai elementai. Pirmas yra 100 µH ritė, kuri naudojama energijos kaupimui. 1000 µF talpos kondensatorius reikalingas išėjimo įtampos pulsacijų mažinimui bei reguliatoriaus ciklo stabilumo užtikrinimui. Pagal gamintojo specifikacijas, tokios talpos kondensatorius užtikrina ne didesnį nei 100 mV įtampos svyravimą. Paskutinis elementas – Šotkio diodas, jis reikalingas ritės srovės tekėjimui, kai pasiekiama reguliatoriaus veikimo ciklo dalis, kai sukaupta energija imama iš ritės. Šotkio diodai turi didelį persijungimo greitį, todėl juos naudojant pasiekiamas geriausias reguliatoriaus efektyvumas.

6.7. Kitos jungtys

Projektuojamo įrenginio funkcionalumui padidinti buvo suprojektuota USB jungtis, kuri panaudojant FT230XS USB į UART keitiklį leidžia prijungti kompiuterį ir palaikyti ryšį su mikrovaldikliu.

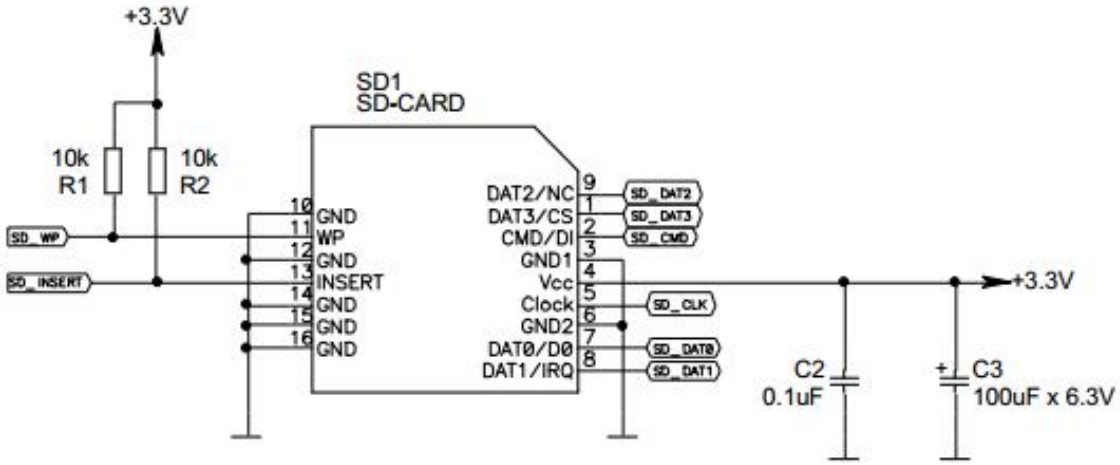


17 pav. USB jungties ir USB į UART keitiklio jungimo schemos

USB jungtis su keitikliu buvo sujungta remiantis keitiklio gamintojo siūloma jungimo schema. Ši schema nenaudoja bendro visos schemos maitinimo, o maitinimą gauna iš USB jungties. Keitiklis su mikrovaldikliu sujungtas panaudojant UART sąsajai skirtas mikrovaldiklio kojeles.

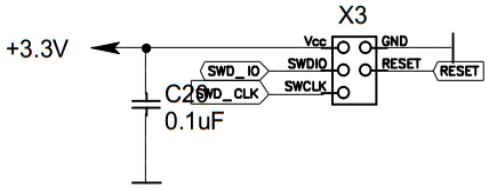
Taip pat įterptinės sistemos atminties praplėtimo galimybei buvo suplanuotas SD kortelės naudojimas. Šios kortelės reikiamos kojelės buvo sujungtos su atitinkamai numatytomis

mikrovaldiklio kojelėmis.



18 pav. SD kortelės schema

Taip pat numatytas mikrovaldiklio sujungimas su JTAG įrenginiu, kuris skirtas programavimo stadijoje padarytoms klaidoms surasti bei ištaisyti. Jis taip pat su mikrovaldikliu sujungiamas panaudojant tam skirtas mikrovaldiklio kojeles.



19 pav. JTAG jungties schema

7. ĮTERPTINĖS SISTEMOS PROGRAMAVIMAS

7.1. TCP/IP bibliotekos

7.1.1. RL-ARM biblioteka

Kadangi naudojamas mikrovaldiklis turi ARM architektūros mikroprocesorių, jam galima naudoti kompanijos Keil sukurta RL (iš angl. Real-Time Library – realaus laiko biblioteka) biblioteką. Ši biblioteka yra tarpusavy susijusių bibliotekų rinkinys, kuris sukurtas tam, kad padėtų įgyvendinti realaus laiko bei komunikacijų sprendimus įterptinėse sistemose.

Šiam projektui buvo naudotas RL-TCPnet bibliotekos komponentas. Jo pagalba pritaikomas TCP/IP protokolų rinkinys naudojant mažai vidinės atminties bei turint mažą kodo dydį. Šis bibliotekos komponentas sukurtas taip, kad veiktų be papildomos operacinės sistemos.

RL-TCPnet bibliotekos komponente yra sukurta Web serverio palaikymo galimybė. Serveris palaiko tiek statinius tiek dinaminius internetinius puslapius. Reikiami internetinio puslapio failai yra pakeičiami į C kodą panaudojant FCARM failų keitiklį. Šis keitiklis visus reikiamus failus pakeičia į vieną C failą, kuris pridedamas prie projekto ir sukompiliuojamas kartu su pagrindine programa.

Šis bibliotekos komponentas taip pat turi numatytą skriptų kalbą, kuri gali būti panaudota dinaminiams internetiniams puslapiams kurti. Web serveris vykdo skripto kodą linija po linijos ir kai reikia iššaukia CGI funkcijas (bus aprašytos žemiau). CGI funkcijų rezultatas yra siunčiamas vartotojui, kaip dalis internetinio puslapio. Kiekviena skripto eilutė prasideda su komandiniu simboliu, kuris pasako kokią komandą reikia atlikti skripto interpretatoriui.

2 lentelė. Skripto komandiniai simboliai.

Simbolis	Apibūdinimas
i	Duoda komandą skripto interpretatoriui, kad reikia pridėti failą iš virtualios failų sistemos ir rodyti jo turinį naršyklėje.
t	Duoda komandą, kad po šio simbolio einanti teksto eilutė turi būti parodyta naršyklėje
c	Iššaukia C funkciją iš HTTP_CGI.c failo. Po funkcijos gali sekti eilutė teksto, kuri perduodama į atitinkamą funkciją kaip rodyklė į kintamąjį.
#	Žymi komentarų eilutę ir yra ignoruojama skripto interpretatoriaus.
.	Žymi paskutinę skripto eilutę.

Norint RL-ARM biblioteką panaudoti su Ethernet valdikliu yra reikalingos trys funkcijos:

- init_ethernet() – Ethernet valdiklio inicializacijai;
- poll_ethernet() – paketo iš Ethernet valdiklio skaitymui;
- send_frame() – paketo rašymui;

init_ethernet() funkcija turi įjungti paketų siuntimą ir priėmimą, nustatyti Ethernet valdiklio MAC adresą, įvykdyti kitus reikiamus registrų pakeitimus tam, kad Ethernet valdiklis veiktų.

poll_ethernet() funkcija kas tam tikrą laiko tarpą turi patikrinti tam tikrą Ethernet valdiklio registrą, kuris registruoja ar yra gautas paketas. Gavus paketą, ši funkcija išskiria atminties bloką ir į jį nukopijuoja gautą paketą. Funkcija uždeda rodyklę ant atminties bloko ir tą rodyklę patalpina į gautų paketų eilę iškviesdama put_in_queue funkciją.

send_frame() funkcija siunčia Ethernet paketą panaudodama Ethernet valdiklį. Pagrindinis funkcijos argumentas *frame* rodo į Ethernet paketą sukonstruotą TCPnet sistemos. Šio argumento komponentas *length* savyje turi informaciją apie siunčiamo paketo ilgį. [19]

7.1.2. lwIP biblioteka

lwIP (iš angl. lightweight Internet Protocol – lengvasvoris interneto protokolas) biblioteka – plačiai naudojamas atvirojo kodo TCP/IP protokolų rinkinio įgyvendinimas sukurtas įterptinėms sistemoms. Ją pradžioje kūrė vienas žmogus vardu Adam Dunkels, bet dabar jos veikimą palaiko daug programuotojų iš viso pasaulio. LwIP tikslas yra TCP/IP pritaikymas naudojant kuo mažiau resursų, tai leidžia nesunkiai jį panaudoti įterptinėse sistemose, nes jo reikalavimai yra kelios dešimtys kilobaitų RAM atminties ir maždaug 40 kB ROM atminties.

Norint panaudoti lwIP biblioteką reikia sukurti tinklo sąsajos struktūrą ir pritaikyti naudojamam Ethernet valdikliui kelias būtinas funkcijas.

Sukuriant naują tinklo sąsają yra sukuriama struktūra *netif* ir iššaukiama *netif_add*:

```
struct *netif netif_add(struct netif *mynetif, struct ip_addr *ipaddr, struct ip_addr *netmask,
        struct ip_addr *gw, void *state, err_t (* init)(struct netif *netif),
        err_t (* input)(struct pbuf *p, struct netif *netif));
```

Čia įrašomas IP adresas, tinklo trafaretas (iš angl. mask) ir tinklų sietuvo adresas (iš angl. gateway address).

Init parametras specifikuoja Ethernet valdiklio inicializacijos funkciją, kurią reikia iššaukti vieną kartą, kuri paruošia Ethernet valdiklį darbui, kaip ir RL-ARM bibliotekos atveju.

Input parametras nurodo Ethernet valdikliui pritaikytą funkciją, kurią reiks kviesti, kai bus gautas paketas.

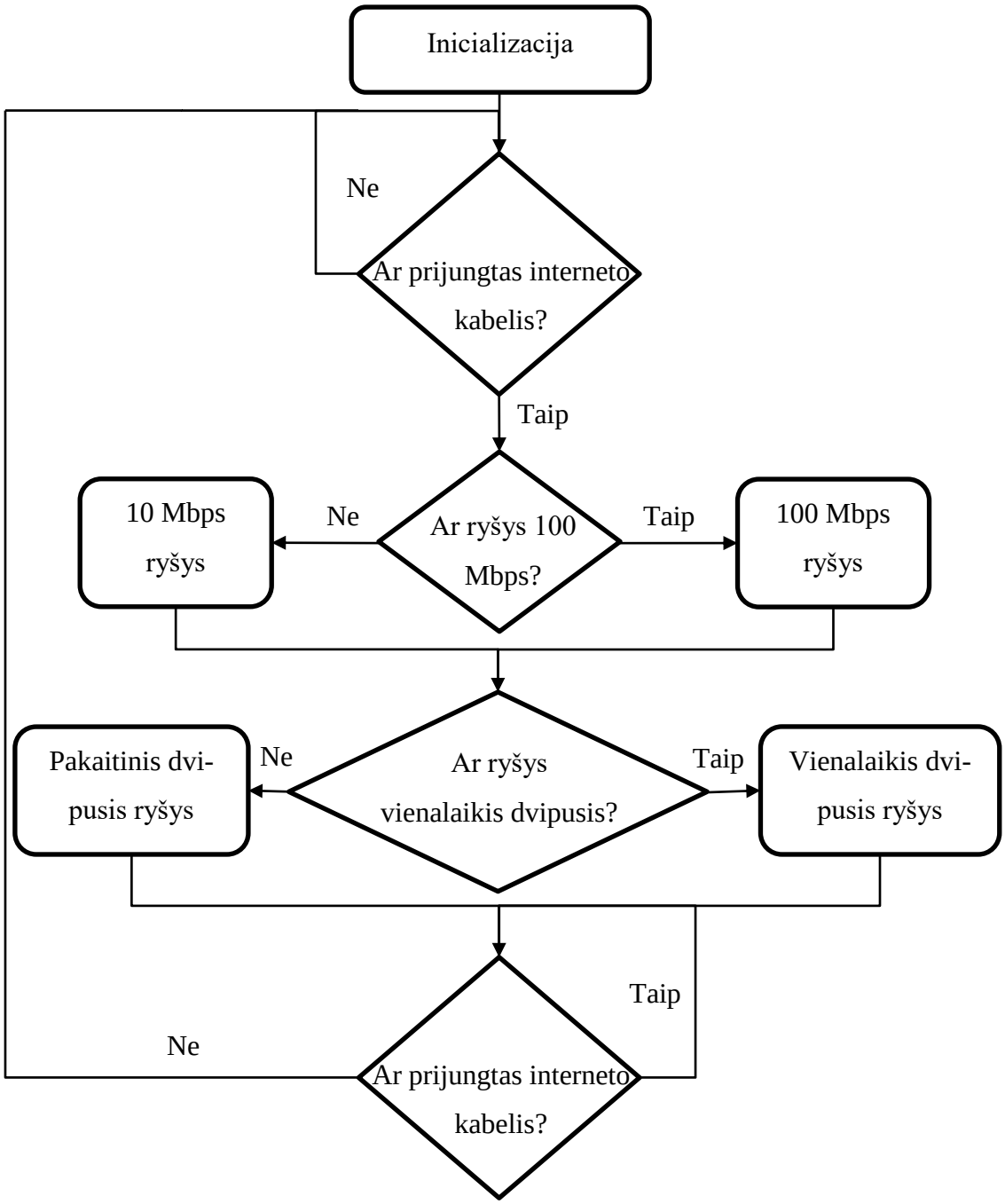
Prieš naudojant sąsają, ją reikia paleisti, tai daroma dviem būdais, priklausomai ar naudojamas DHCP (iš angl. Dynamic Host Configuration Protocol – dinaminis stočių konfigūravimo protokolas)

ar ne. Statiniam IP adresui naudojama *netif_set_up* ir *netif_set_down.* Naudojant DHCP naudojama *dhcp_start* ir *dhcp_stop*. Patikrinimui ar sąsaja veikia gali būti naudojama *netif_is_up* funkcija.

Papildomai be šios tinklo struktūros reikalinga išvesties funkcija *err_t myif_output(struct netif *netif, struct pbuf *p, ip_addr_t *ipaddr)*, kuri iššaukiama, kai paketas yra paruoštas siuntimui. [20]

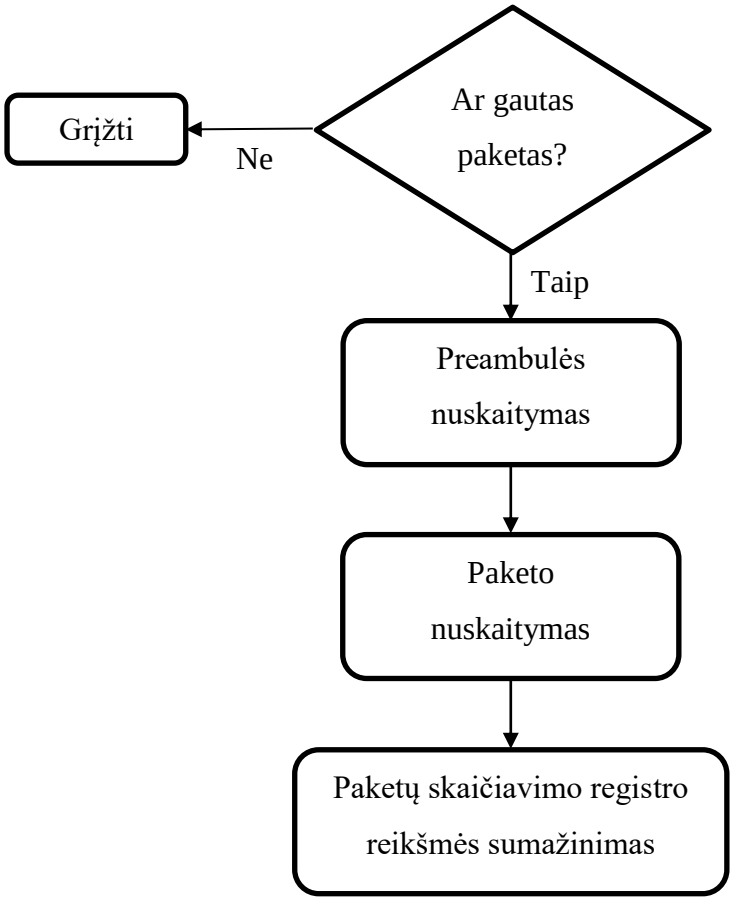
7.2. Serverio pusės programavimas

Susipažinus su LPC1776 mikrovaldiklio bei ENC424J600 Ethernet valdiklio dokumentacija jų pagalba parašytos programinės tvarkyklės (iš angl. driver) įgalinančios su RL-TCPnet bibliotekos elementu ar lwIP biblioteka išgauti duomenų perdavimą HTTP protokolu įterptinėje sistemoje.



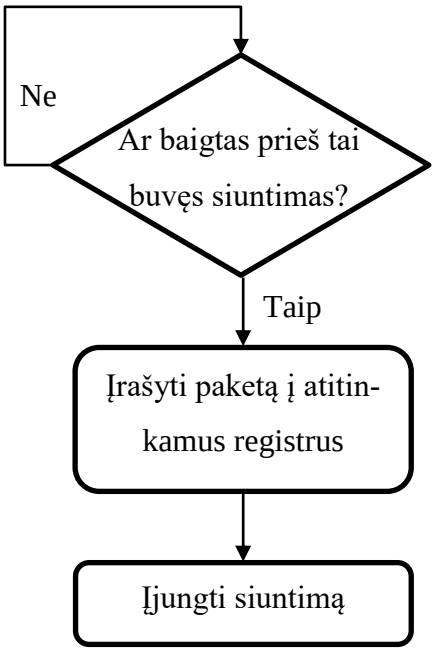
20 pav. Blokinė ryšio tikrinimo schema

Ryšio tikrinimo ciklas visų pirma įvykdo Ethernet valdiklio inicializacija įrašant reikiamas vertes į tam tikrus registrus. Toliau ciklišKai tikrinama ar prie valdiklio yra prijungtas interneto kabelis, tai vykdoma nuskaitant reikiamą registro reikšmę. Kai aptinkamas prijungtas kabelis vykdomas ryšio informacijos gavimas ir nustatomas ryšio greitis ir ryšio veikos tipas. Gavus šią informaciją vykdomas naujas ciklas, kuris vėl tikrina ar įjungtas interneto kabelis, jį ištraukus iš lizdo ciklas peršoka į pradinį interneto kabelio tikrinimą. (Žr. Priedas 5)



21 pav. Blokinė paketo priėmimo schema

Priimant paketą pirmiausia vykdomas tikrinimas ar yra gautas paketas, gavus paketą nuskaitoma jo preambulė, kurioje yra informacija apie paketą bei apie įvykusias klaidas. Toliau vykdomas pačio paketo nuskaitymas iš atitinkamų registrų. Galiausiai yra sumažinama paketų skaičiavimo registro reikšmė, jei ji pasiekia nulį, reiškia nėra likusios informacijos kurią dar reikia nuskaityti. (Žr. Priedas 6)



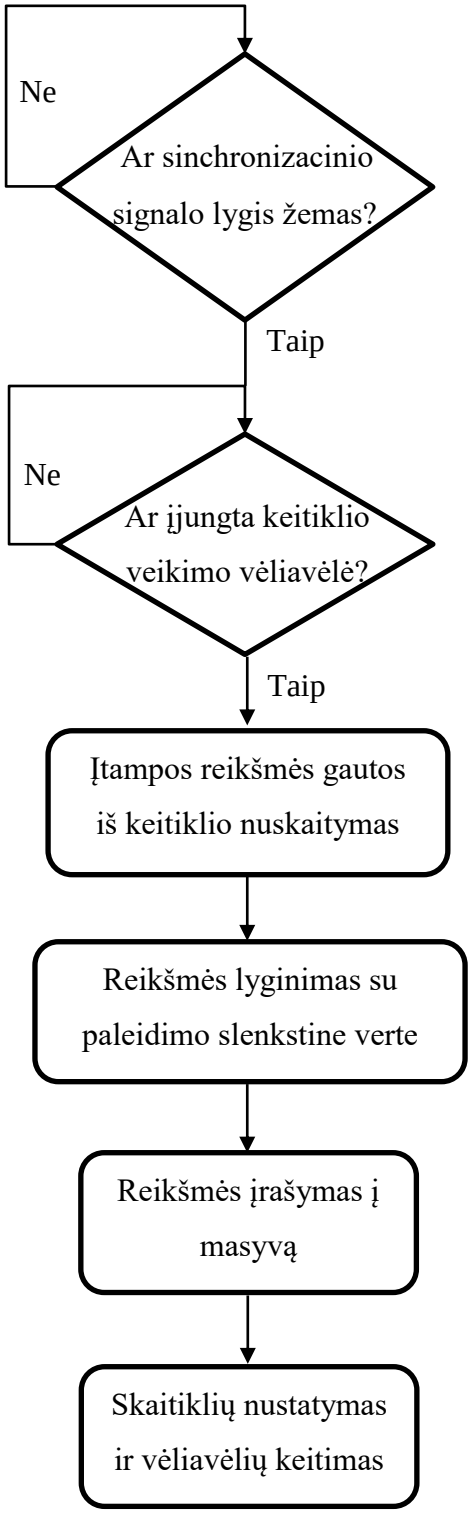
22 pav. Blokinė paketo siuntimo schema

Siunčiant paketą pirmiausia ciklišk ai tikrinama ar baigtas prieš tai buvęs siuntimas. Pasibaigus prieš tai vykdytam siuntimui įrašoma paketo informacija į atitinkamus registrus. Apart pačios informacijos, taip pat nurodomas paketo ilgis. Atlikus informacijos įrašymą pakeičiama atitinkamo registro reikšmė ir taip įjungiamas paketo siuntimas. (Žr. Priedas 7)

Susipažinus su TLC5540 analoginio-skaitmeninio keitiklio dokumentacija jos pagalba parašyti reikiami keitiklio valdikliai. Įrenginio inicializacija labai paprasta – norint įjungti duomenų nuskaitymą ir perdavimą mikrovaldiklis turi nustatyti keitiklio kojelės OE (iš angl. output enable – išvedimo įjungimas) lygį į žemą. Keitiklio veikimui reikalingas sinchronizacinis signalas, kurio generavimas parašytas panaudojant vidinį mikrovaldiklio laikmatį, kurio dažnis nustatytas 1 MHz. Kai laikmačio reikšmė lyginė – sinchronizacinio signalo kojelė pakeliama į aukštą lygį, kai nelyginė – atvirkščiai. Kai sinchronizacinis signalas yra aukštame lygyje, analoginis-skaitmeninis keitiklis pakeičia 8 duomenų kojelių lygius pagal jame esančių įtampos komparatorių duomenis. Duomenų išgavimas vykdomas panaudojant ciklą. Jo pagalba nuskaitomi visų 8 kojelių lygiai ir jie sudedami į 8 bitų ilgio dvejetainį skaičių, kurį padauginus iš apskaičiuotos konstantos gaunama į keitiklį atėjusios įtampos vertė milivoltais. (Žr. Priedas 8)

Konstanta apskaičiuota remiantis tuo, kad keitiklio įtampos priėmimo riba yra ties 2,28 volto, o keitiklis yra 8 bitų, tai yra turi 255 galimus žingsnelius. Gauname, kad vienas žingsnelis atitinka:

$$C_{keit.} = \frac{2{,}28\ \text{V}}{255} = 8{,}94\ \text{mV}$$



23 pav. Blokinė duomenų rašymo į masyvą schema

Norint vartotojui siųsti duomenis ne po vieną skaičių reikalingas duomenų surašymas į masyvą. Tai vykdoma patalpinant duomenų rašymo į masyvą funkciją į vieno iš vidinio mikrovaldiklio laikmačių trūkio valdymo funkciją, kurios pagalba kartu generuojamas ir sinchronizacinis signalas. Pirmiausia yra tikrinama ar sinchronizacinio signalo lygis yra žemas. Tada tikrinama analoginio-skaitmeninio keitiklio veikimo vėliavėlė. Ją įjungus yra nuskaitoma įtampos reikšmė. Ta reikšmė yra palyginama su nustatyta, taip vadinama paleidimo slenkstine (iš angl. trigger)

reikšme, kurios pagalba gaunamas sinchronizuoto signalo vaizdavimas vartotojo ekrane. Jei reikšmė – didesnė ir prieš tai nėra nustatyta paleidimo slenkstinės reikšmės viršijimo vėliavėlė, pirmiausia nustatoma minėtoji vėliavėlė, tada paleidimo bei pagrindinis masyvo skaitikliai nustatomi į nulinę vertę. Kitu atveju, funkcija tiesiog tęsiama toliau. Kitame žingsnyje gauta reikšmė yra įrašoma į masyvo vietą, kurią nustato pagrindinis masyvo skaitiklis. Po šio veiksmo pagrindinio skaitiklio vertė yra pakeliama vienetu, jei ji dabar viršija nustatytą masyvo dydį skaitiklis nustatomas atgal į 0. Toliau yra tikrinama ar nustatyta paleidimo slenkstinės vertės viršijimo vėliavėlė, jeigu taip – padidinama atskiro paleidimo skaitiklio reikšmė. Kai ji viršija nustatytą masyvo dydį, tai reiškia, kad visos masyvo reikšmės yra užpildytos reikšmėmis nuo paleidimo slenkstinės reikšmės viršijimo įvykio momento ir tai reiškia rašymo į masyvą pabaigą. Nustatoma pabaigos vėliavėlės vertė. Rašymas sustoja iki to momento, kol vėl nebus įjungta keitiklio veikimo vėliavėlė. (Žr. Priedas 9)

Web serverio funkcijų atlikimas išpildytas lwIP bibliotekos pagalba. Serveris gavęs užklausą kažkokiam failui gauti, to failo ieško C kalba užprogramuotame faile, kuriame užkoduoti visi reikiami internetiniam puslapiui duomenys.

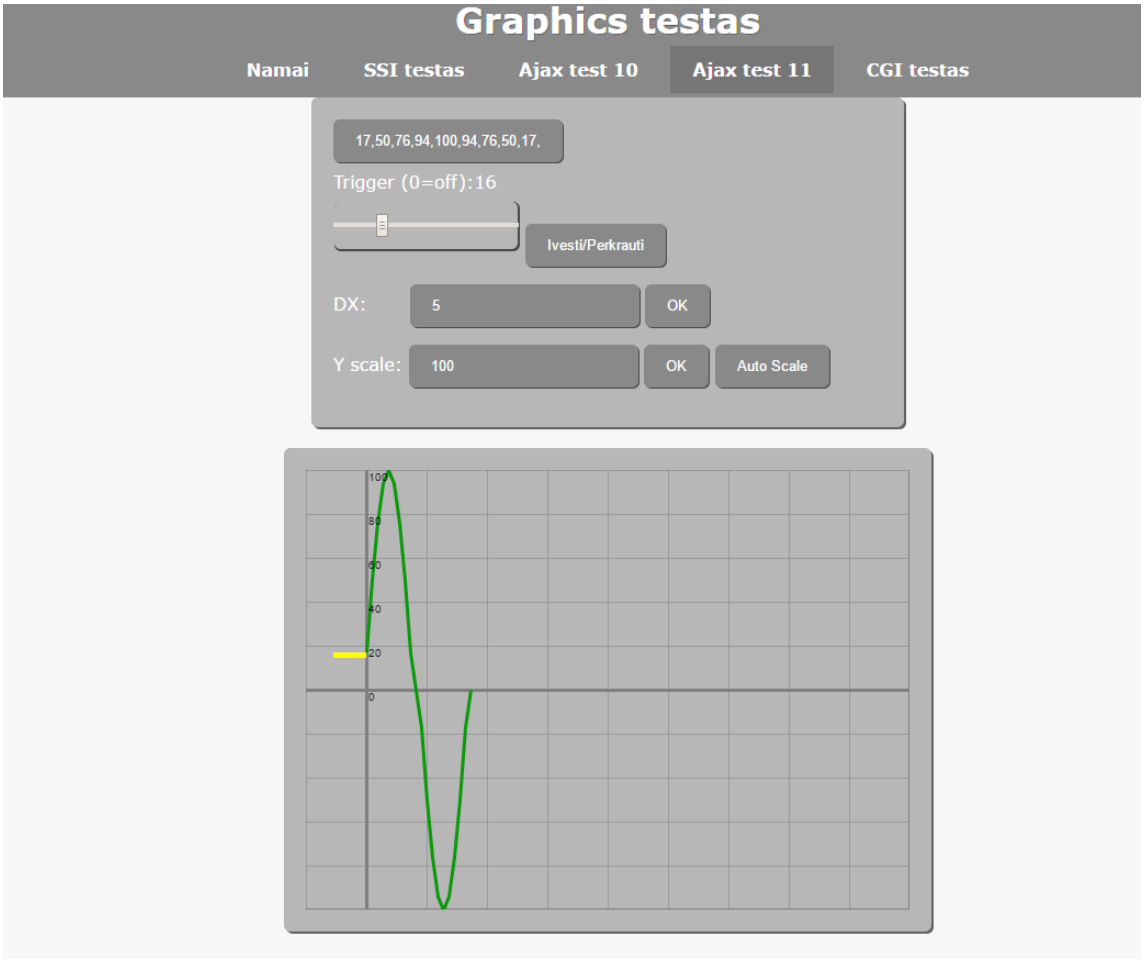
Duomenys nuskaityti iš analoginio-skaitmeninio keitiklio vartotojui perduodami Ajax užklausos metu. Ajax pagalba vartotojas duoda užklausą specialiai pavadintam failui gauti, pavyzdžiui, „1.cgx“. Serveris gavęs tokią užklausą ir radęs failo pavadinimą serverio failų sistemos paprogramėje iškviečia specialiai parašytą funkciją, kuri ciklišk ai patalpina duomenis iš skaitinio („int“ tipo) keitiklio duomenų nuskaitymo metu sukurto masyvo į simbolių („char“ tipo) masyvą, kuriame kiekvienas atskiras duomuo atskiriamas tarpu. Šis simbolių masyvas ir perduodamas vartotojui, kaip „1.cgx“ failas. (Žr. Priedas 10)

Vartotojo duomenų perdavimui į serverį naudojama CGI (iš angl. Common Gateway Interface – bendra sietuvinė sąsaja). Esant nustatytiems kriterijams (aktyvaus internetinio puslapio failo pavadinimui, GET užklausos parametrams) vartotojui siunčiant GET užklausą, yra iškviečiamas serveryje esantis CGI skriptas, kuris nuskaito užklausos parametrus ir tų parametrų reikšmes bei atlieka užprogramuotas funkcijas. Šiame darbe vartotojas serveriui perduoda nustatytą vadinamąją paleidimo slenkstinę reikšmę, kurios pagalba sinchronizuojamas signalo vaizdavimas. CGI skriptas aptikęs „trigger“ parametrą nuskaito jo reikšmę ir nustato gautą vertę, kuri naudojama prieš tai aprašytame cikle. (Žr. Priedas 11)

7.3. Kliento pusės programavimas

Kliento sąveikai su serveriu sukurtas internetinis puslapis, kuriame HTML kalba patalpintas 550 pikselių pločio ir 400 pikselių aukščio laukas, kuriame atvaizduojami gauti duomenys bei sukurti mygtukai ir laukeliai duomenų atvaizdavimo skalėms ir paleidimo slenkstinei reikšmei valdyti bei gautiems duomenims matyti. Panaudojant CSS kalbą paredaguotas šio puslapio atvaizdavimas, kad

jis būtų malonesnis vartotojo akiai bei informacija būtų lengvai matoma ir skaitoma. (Žr. Priedas 12)



24 pav. Vartotojo sąsajos atvaizdavimas

Pačiai sąveikai su serveriu ir duomenų atvaizdavimui išgauti parašytos funkcijos JavaScript kalba. Vienoje iš pagrindinių funkcijų, naudojamoje duomenų išgavimui iš serverio, pirmiausia sukuriama Ajax technologijoje labai plačiai naudojama vadinamoji XMLHTTP užklausa. Jos pagalba pilnai neperkraunant puslapio, panaudojant HTTP GET metodą, serverio paprašoma gauti prieš tai aprašytą .cgx failą, kuriame yra patalpinti iš skaitmeninio-analoginio keitiklio gauti duomenys atskirti tarpais. Gavus šiuos duomenis, kai užklausos būsena pasikeičia į „200 OK“, tekstiniai duomenys yra atskiriami panaudojant atskyrimo metodą ir patalpinami į skaitinį masyvą. Ši funkcija yra iškviečiama nuolatos su tam tikru, nustatytu intervalu. Taip duomenys yra nuolatos atnaujinami. (Žr. Priedas 12 AjaxSubmit funkciją)

Duomenys atvaizduojami nuolatos iškviečiant kitą funkciją, kurios intervalas nustatomas 1 ms. Ši funkcija ciklišk ai keldama skaitiklio reikšmę patalpina gautus duomenis iš vieno skaitinio masyvo į kitą, naują masyvą, pasiekus paskutinį skaičių duomenų masyve, skaitiklis vėl nustatomas į 0 vertę. Įvedus vertę į naujai sukurtą masyvą, jis piešiamas žalia linija. Prieš tai nupiešiamos skalės bei skalių vertės. Kiekvienam piešimo veiksmui atlikti parašytos atskiros funkcijos, kuriomis nustatomos tikslios piešimo koordinatės, spalvos ir kiti parametrai. (Žr. Priedas 12 splitData funkciją)

Paleidimo slenkstinės reikšmės vertei siųsti parašyta funkcija, kurioje vėlgi naudojama XMLHTTP užklausa. Panaudojant HTTP GET metodą yra paprašoma gauti failą .cgi, kurio pavadinimas yra nustatytas serveryje esančioje CGI skriptų tvarkyklėje, su nustatyta reikšme panaudojant GET metodo sintaksę (pvz. .cgi?trigger=500). Taip serveriui perduodama vartotojo nustatyta paleidimo slenkstinė vertė, kurią jis nuskaito CGI skripto pagalba. (Žr. Priedas 12 triggerSubmit funkciją)

REZULTATŲ APTARIMAS

Buvo suprojektuotas ir pagamintas veikiantis schemos prototipas.

Panaudojant RL-TCPnet ir lwIP bibliotekas mikrovaldiklio atmintyje buvo realizuotas Web serveris, kurio pagalba parašius HTML bei HTML+Ajax internetinius puslapius, buvo išbandyti GET bei POST HTTP užklausų metodai bei išbandyti CGI skriptai.

Panaudojant Web serverį buvo ištirta priklausomybė kaip priklauso paveikslėlio užkrovimo trukmė nuo paveikslėlio dydžio. Tuo tikslu buvo sukurti 5 paveikslėlių, kurių dydis kito nuo 10 kB iki 50 kB ir buvo fiksuojamas laikas per kurį vartotojas šį paveikslėlį išgaudavo iš mikrovaldiklyje esančio serverio. Tai pasiekta panaudojant tam skirtą „Chrome“ interneto naršyklės plėtinį. Laikas fiksuotas po 5 kartus ir išvestas vidurkis. Priderinus prie gautos kreivės tiesę, apskaičiuota realios duomenų perdavimo spartos gautos panaudojant skirtingas bibliotekas bei laikas reikalingas užklausai bei atsakymui pateikti:

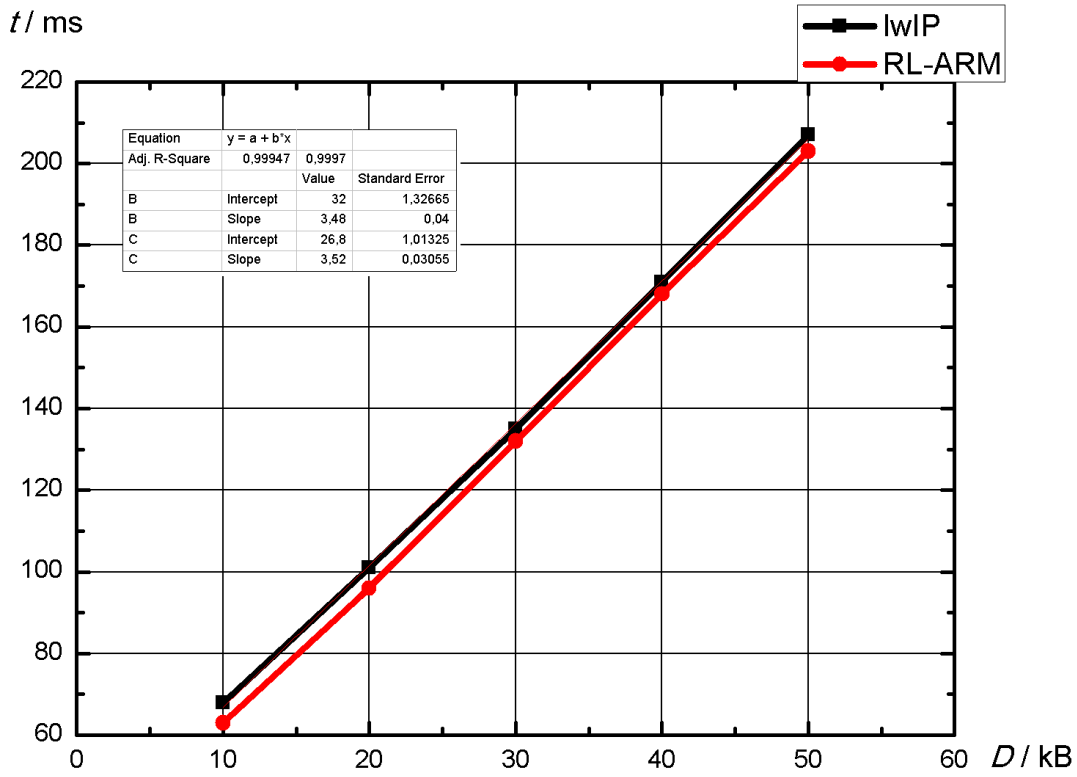
$$S_{\mathrm{lwIP}} = \frac{1}{3{,}48\,\frac{\mathrm{ms}}{\mathrm{kB}}} = 0{,}2874\,\frac{\mathrm{kB}}{\mathrm{ms}} = 287{,}4\,\frac{\mathrm{kB}}{\mathrm{s}}, \qquad T_{\mathrm{lwIP}} = 32\ \mathrm{ms}$$

$$S_{\mathrm{RLARM}} = \frac{1}{3{,}52\,\frac{\mathrm{ms}}{\mathrm{kB}}} = 0{,}2841\,\frac{\mathrm{kB}}{\mathrm{ms}} = 284{,}1\,\frac{\mathrm{kB}}{\mathrm{s}}, \qquad T_{\mathrm{RLARM}} = 26{,}8\ \mathrm{ms}$$

Čia S_{lwIP}, S_{RLARM} - realios duomenų perdavimo spartos gautos iš priderintų tiesių polinkio, T_{lwIP}, T_{RLARM} – užklausų ir atsakymų trukmės gautos iš priderintų tiesių Y ašies kirtimo vietos.

3 lentelė. Paveikslėlio užkrovimo trukmės priklausomybė nuo jo dydžio

Dydis *D* / kB	Trukmė *t* / ms (lwIP)	Trukmė *t* / ms (RL-ARM)
10	68	63
20	101	96
30	135	132
40	171	168
50	207	203



25 pav. Paveikslėlio užkrovimo trukmės priklausomybė nuo jo dydžio

Iš rezultatų matyti, kad lwIP biblioteka paremtas Web serveris pasižymi didesne duomenų perdavimo sparta, bet naudojant RL-ARM biblioteką gaunamas ženkliai geresnis laikas reikalingas užklausai ir atsakymui atlikti.

Taip pat buvo ištirta atsako trukmė ping įrankiu, esančiu Windows operacinėje sistemoje, su skirtingo dydžio paketais. Ping įrankio veikimas labai paprastas. Į nustatytą adresą išsiunčiamas pranešimas ir laukiamas atsakymas, gavus atsakymą fiksuojama trukmė. Buvo siunčiama 50 pranešimų ir fiksuojamos trumpiausios ir ilgiausios trukmės bei vidurkis. Paketo dydis apribotas ties 1280 baitais, nes serverių nustatymuose nustatytas MTU (iš angl. maximum transmission unit – didžiausias perdavimo elementas) yra 1500 baitų. MTU nustato, koks yra didžiausias leistinas duomenų kiekis perduodamas vienos tinklo transakcijos metu.

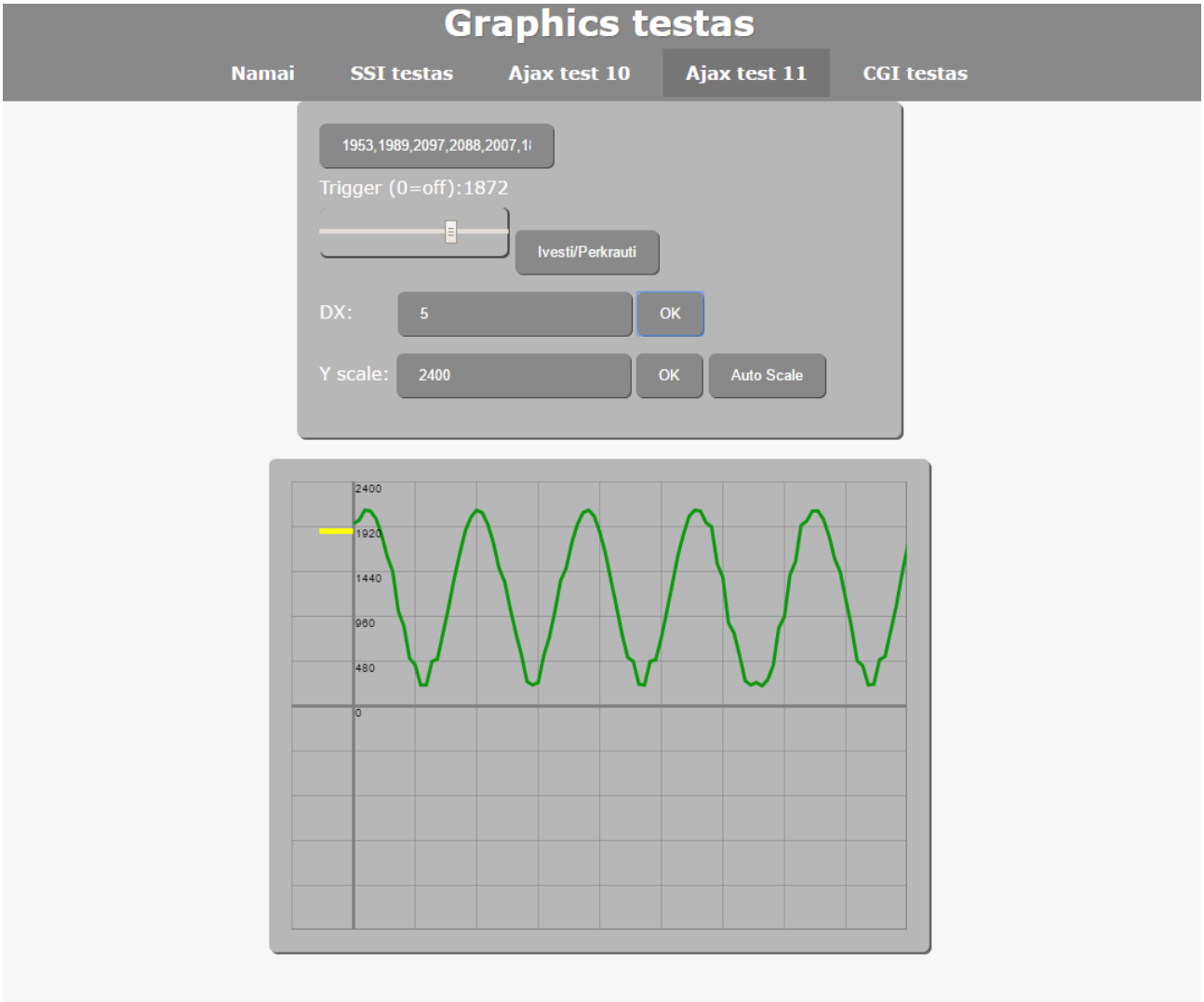
4 lentelė. Atsako trukmės gautos ping įrankiu panaudojant lwIP bei RL-ARM

lwIP			
Paketo dydis baitais	Trumpiausia	Ilgiausia	Vidurkis
32	1	11	2
128	2	16	4
512	5	19	5
1280	10	22	11
RL-ARM			
Paketo dydis baitais	Trumpiausia	Ilgiausia	Vidurkis
32	1	5	2
128	2	9	3
512	5	10	5
1280	8	15	10

Iš šių rezultatų matyti, kad atsako trukmės skiriasi labai mažai, tačiau RL-ARM biblioteka pasižymi tiek trumpesne, tiek stabilesne atsako trukme. Sukompiliavus programas gauta, kad programa naudojant RL-ARM biblioteką užima 38,40 kB, o naudojant lwIP biblioteką – 44,98 kB ROM atminties. Pasak bibliotekų aprašymų, jų RAM atminties reikalavimai turėtų būti labai panašūs. Priklausomai nuo naudojamų funkcijų kiekio – apie 20 kB. Todėl sistemos apkrova nuo to, kokia biblioteka bus naudojama, praktiškai nepasikeis.

Kadangi palyginimų rezultatai yra labai artimi buvo pasirinkta toliau naudoti lwIP biblioteką. Šios bibliotekos CGI skriptų pritaikymas pasirodė lengviau implementuotinas ir nereikėjo naudoti specialios RL-ARM CGI skriptams reikalingos kalbos. Taip pat biblioteka yra visiškai nemokama ir jos licencija yra labai atviro tipo.

Panaudojant lwIP bibliotekos pagalba mikrovaldiklio atmintyje realizuotą Web serverį buvo parašytas pilnai veikiantis osciloskopo funkciją primenantis dinaminio tipo internetinis puslapis, kuriame Ajax užklausų bei CGI skriptų pagalba vartotojas gali matyti duomenis įvestus į plokštės analoginį įėjimą bei nustatyti paleidimo slenkstinę vertę taip sinchronizuodamas vaizdą ties ta verte. Taip pat numatytas vartotojo X bei Y skalių valdymas.



26 pav. Duomenų gautų per analoginį schemos įėjimą vaizdavimas internetiniame puslapyje

Panaudojant „Chrome“ naršyklėje esančius įrankius buvo ištirta Ajax modelyje naudojamoms XMLHTTP užklausų ir atsakymų trukmė. Šių užklausų pagalba iš serverio yra gaunami analoginio-skaitmeninio keitiklio duomenys bei perduodama paleidimo slenkstinė vertė. Buvo atlikta po 50 užklausų, gautos trumpiausios, ilgiausios trukmės bei vidurkis.

5 lentelė. XMLHTTP užklausų ir atsakymų trukmės

Pavadinimas	Trumpiausia	Ilgiausia	Vidurkis
Analoginio-skaitmeninio keitiklio duomenų perdavimo užklausa ir atsakymas (832 baitai)	14 ms	28 ms	16 ms
Paleidimo slenkstinės vertės siuntimo užklausa ir atsakymas (114 baitų)	8 ms	13 ms	10 ms

Naudojamo analoginio-skaitmeninio keitiklio didžiausia sparta yra 40 milijonų imčių per sekundę. Naudojamo mikrovaldiklio didžiausias taktinis dažnis yra 120 MHz, kadangi keitiklio

perduotos reikšmės nuskaitymui ir įrašymui į masyvą reikalinga apie 30 komandų, tai parašytu kodu daugiausiai galima nuskaityti apie 4 milijonus imčių per sekundę. Realiai šis skaičius dar labiau sumažėja, nes mikrovaldiklis turi atlikti daug daugiau veiksmų nei reikšmių nuskaitymas ir įrašymas tam, kad duomenis būtų galima perduoti HTTP protokolu. Duomenų perdavimo HTTP protokolu sparta darbinę spartą įtakoja mažiau, nes viena užklausa galima perduoti reguliuojamą imčių skaičių. Tai reiškia, kad suprojektuotos įterptinės sistemos darbinę spartą, nuskaitant bei perduodant duomenis gautus per analoginį įėjimą, apriboja mikrovaldiklio parametrai. Čia darbine sparta vadinamas analoginio signalo diskretizavimas, duomenų iš analoginio-skaitmeninio keitiklio nuskaitymas, tų duomenų patalpinimas į reikiamą masyvą bei perdavimas vartotojui HTTP protokolu.

REZULTATAI IR IŠVADOS

- Naudojant lwIP ir RL-ARM bibliotekas pavyko panaudoti įterptinę sistemą kaip Web serverį ir perduoti tiek statinio, tiek dinaminio tipo internetinius puslapius;
- Gautos lwIP ir RL-ARM bibliotekomis paremtų Web serverių duomenų perdavimo spartos – atitinkamai 287,4 kB/s ir 284,1 kB/s;
- Taip pat gautos jų atsako trukmės, kurios skyrėsi nežymiai, tačiau RL-ARM biblioteka paremtas Web serveris pasižymėjo trumpesne ilgiausio atsako trukme;
- Panaudojant lwIP biblioteka paremtą Web serverį bei parašius dinaminio tipo internetinio puslapio kodą buvo gauta internetinėje naršyklėje veikianti interaktyvi vartotojo sąsaja, kuri vaizduoja per analoginį įterptinės sistemos įėjimą gautus duomenis;
- Įvertinus gautus rezultatus nuspręsta, kad suprojektuotos įterptinės sistemos darbinę spartą, nuskaitant bei perduodant duomenis gautus per analoginį įėjimą, apriboja mikrovaldiklio parametrai – taktinis dažnis bei darbinė atmintis.

LITERATŪRA

[1] Dave Evans – „How the Next Evolution of the Internet Is Changing Everything“, *Cisco Internet Business Solutions Group*, 2011

[2] Geng Wu, Shilpa Talwar, Kerstin Johnsson, Nageen Himayat, Kevin D. Johnson – „M2M: From Mobile to Embedded Internet“, *IEEE Communications Magazine*, 2011

[3] Microchip – „Ethernet Theory of Operation“, 2008

[4] W. Richard Stevens – „TCP/IP Illustrated Vol 1“, 2001

[5] Roy T. Fielding, James Gettys, Jeffrey C. Mogul, Henrik Frystyk Nielsen, Larry Masinter, Paul J. Leach, Tim Berners-Lee – „HyperText Transfer Protocol – HTTP/1.1“, *RFC 2068*, 1999

[6] T. Berners-Lee, D. Connolly – „HyperText Markup Language – 2.0“, 1995

[7] Mozilla Developer Network – „CSS developer guide“, 2015

[8] Jesse James Garrett – „Ajax: A New Approach to Web Applications“, 2007

[9] David Flanagan „JavaScript: The Definitive Guide. 6th Edition“, 2011

[10] Steve Heath – „Embedded Systems Design“, 2003

[11] Zach Shelby – „Embedded Web Services“, *IEEE Wireless Communications*, 2010

[12] I.D. Agranat – „Engineering Web technologies for embedded applications*“, IEEE Internet Computing*, 1998

[13] Vytautas Jonkus – Mikrovaldikliai elektroninėse grandinėse, 2008

[14] Mikrovaldiklio LPC1776 dokumentacija – http://www.nxp.com/documents/data_sheet/LPC178X_7X.pdf

[15] Ethernet valdiklio ENC424J600 dokumentacija – http://ww1.microchip.com/downloads/en/DeviceDoc/39935c.pdf

[16] Analoginio-skaitmeninio keitiklio TLC5540 dokumentacija – http://www.ti.com/lit/ds/symlink/tlc5540.pdf

[17] Įtampos reguliatorių LM2576 dokumentacija – http://www.ti.com/lit/ds/symlink/lm2576.pdf

[18] USB į UART keitiklio FT230XS dokumentacija – http://www.ftdichip.com/Support/Documents/DataSheets/ICs/DS_FT230X.pdf

[19] RL-ARM internetinis puslapis – http://www.keil.com/support/man/docs/rlarm/

[20] lwIP internetinis puslapis – https://savannah.nongnu.org/projects/lwip/

SUMMARY

Adolis Vaisieta

Data transfer using HTTP protocol in embedded systems

Master‘s thesis

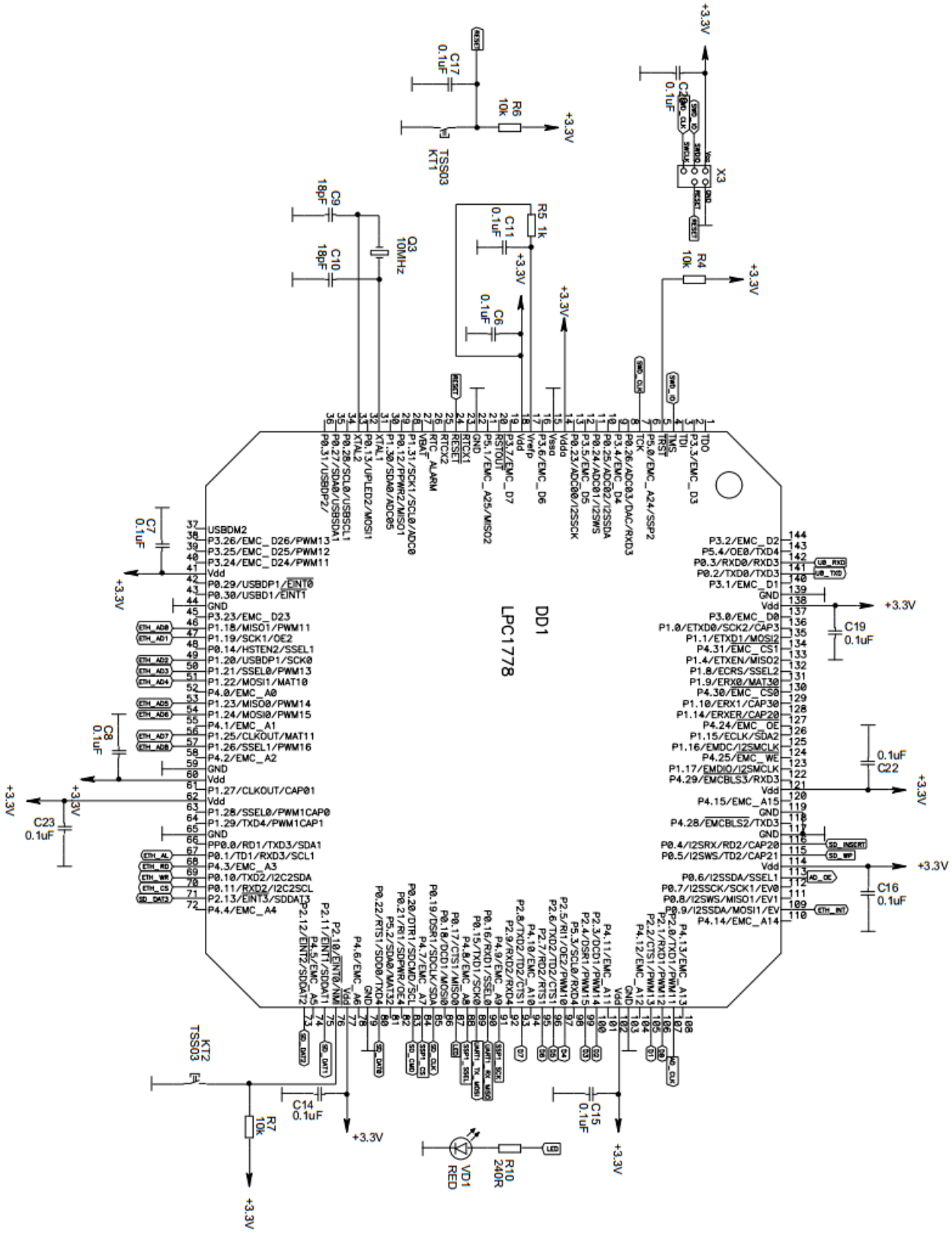
Goal of this thesis was to design and produce a circuit board which would be capable of acting as a Web server and then transfer data using HTTP.

“P–CAD” software was used to design this circuit board which was then produced and tested. Then RL-ARM and lwIP TCP/IP libraries were used to achieve functionality of Web server. Code for various functions were written in order for embedded system to work as a Web server. Then RL-ARM and lwIP Web servers were tested and their data transfer speeds and round-trip delay times were compared. Then website code was written by using HTML, JavaScript and Ajax. This website was able to display data which was being input through embedded system’s analog input. Data display style was chosen to be similar to that of an oscilloscope.

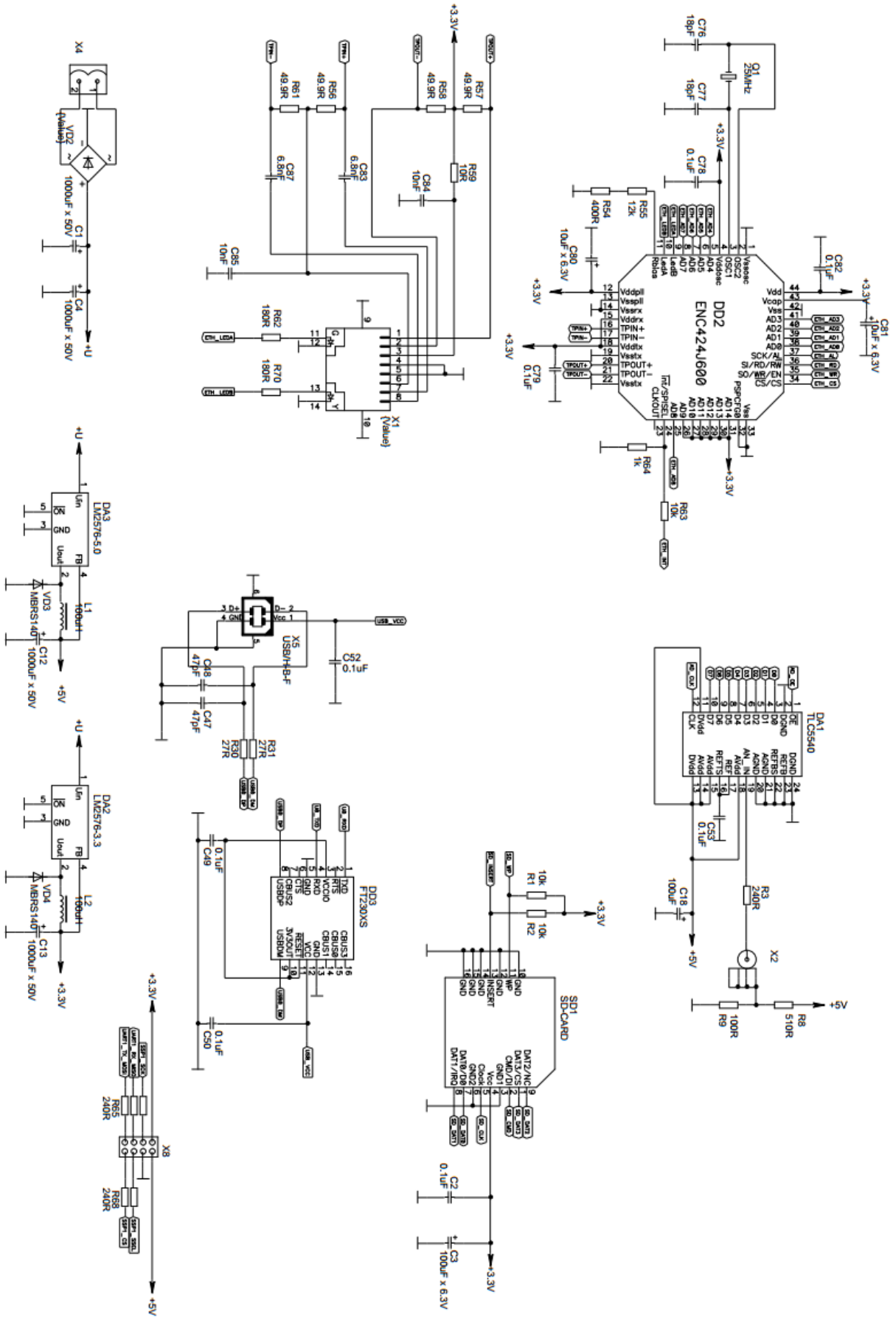
Main results and conclusions:

- Designed embedded system was successfully used as a Web server by using lwIP and RL-ARM libraries and both static and dynamic webpages were tested;
- Web servers based on lwIP and RL-ARM libraries were tested to find out their data transfer speeds – they were respectively 287.4 kB/s and 284.1 kB/s;
- Also, servers‘ round-trip delay times were tested, the differences were miniscule but RL-ARM based Web server had better maximum round-trip delay time;
- By using lwIP library based Web server and dynamic type website code, website browser based interactive user interface was achieved, by using this interface user could see data which was input through embedded system‘s analog input;
- Designed embedded system‘s speed while handling data from analog input is limited by microcontroller‘s properties – clock rate and RAM.

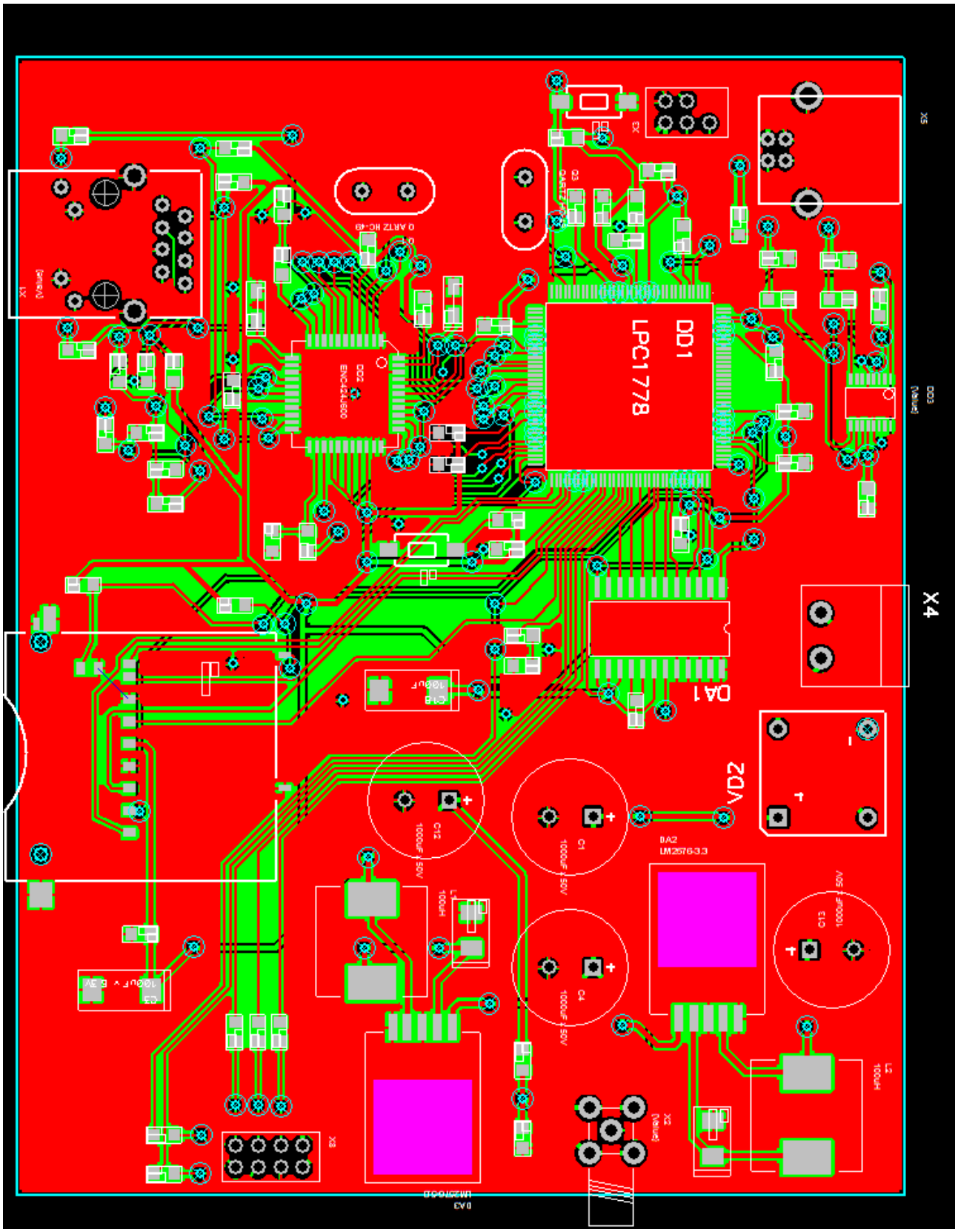
1 priedas. Mikrovaldiklio jungimo schema



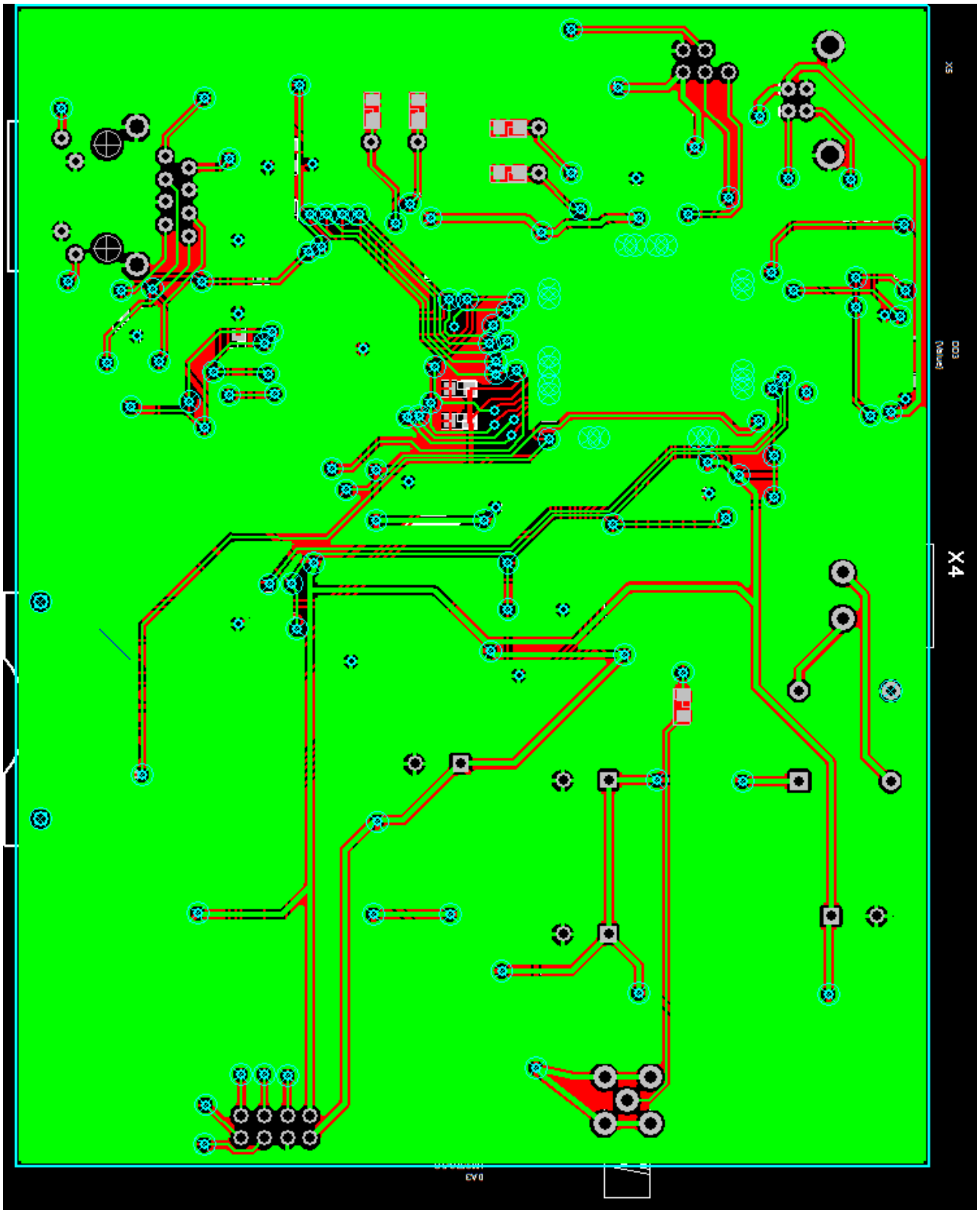
2 priedas. Papildomų įrenginių ir jungčių jungimo schemos



3 priedas. Suprojektuotos įterptinės sistemos PCB schema (viršus)



4 priedas. Suprojektuotos įterptinės sistemos PCB schema (apačia)



5 priedas. Ryšio tikrinimo kodas

```
          if(ETH_GetStatus() &  ETH_INIT_PASSED)
         {
             if(EthStatus & ETH_LINK_ON)
             {
                 if((EthStatus & ETH_LINK_ON_TASK) == 0)
                 {
                     EthStatus |= ETH_LINK_ON_TASK;
                     EthStatus &= ~ETH_LINK_OFF_TASK;
                     xfprint(OUTDEV_UART0,"\n\rLink on ");
                     ETH_PostLinkInit(ucMACAddress);
                     ETH_GetStatus();
                     if(EthStatus & ETH_SPEED) xfprint(OUTDEV_UART0,"100Mb/s ");
                     else xfprint(OUTDEV_UART0,"10Mb/s ");
                     if(EthStatus & ETH_DUPLEX) xfprint(OUTDEV_UART0,"Full duplex
");
                     else xfprint(OUTDEV_UART0,"half duplex ");
                     xfprint(OUTDEV_UART0,"\n\rIP=%s",
ipaddr_ntoa(&MyInterface.ip_addr));

xfprint(OUTDEV_UART0,"\n\rMAC=%02X:%02X:%02X:%02X:%02X:%02X",MyInterface.hwaddr[
0],

MyInterface.hwaddr[1],

MyInterface.hwaddr[2],

MyInterface.hwaddr[3],

MyInterface.hwaddr[4],

MyInterface.hwaddr[5]);

                     netif_set_up(&MyInterface);
                     netif_set_link_up(&MyInterface);
             }
             else
             {
                 if((EthStatus & ETH_LINK_OFF_TASK) == 0)
                 {
                     EthStatus |= ETH_LINK_OFF_TASK;
                     EthStatus &= ~ETH_LINK_ON_TASK;
                     EthStatus &= ~ETH_POSTINIT_PASSED;
                     EthStatus &= ~ETH_DHCP_IP_RECEIVED;
                     xfprint(OUTDEV_UART0,"\n\rLink off ");
                     netif_set_down(&MyInterface);
                 }
             }
         }
```

6 priedas. Paketo priėmimo kodas

```
void ETH_ReceiveFrame(void)
{
    unsigned char  buf[22];
    unsigned short temp;
    unsigned char  *ptr;

    #if USE_ETHERNET_MUTEX>0
        if(!osMutexWait(EthMutex, 300))
        {
            #if (USE_STREAM>0) && (USE_ERROR>0)
            xfprint(OUTDEV_ERROR,"\n\rETH_ReceiveFrame:: mutex timeout");
            #endif
            return;
        }
    #endif
    if(!(Enc424j600_Read(EIR) & EIR_PKTIF))  // ar gautas paketas?
    {
        #if USE_ETHERNET_MUTEX>0
        osMutexRelease(EthMutex);
        #endif
        return;
    }

    wCurrentPacketPointer = wNextPacketPointer;
    Enc424j600_Write(ERXRDPT, wCurrentPacketPointer);       // paketo pradzia
    Enc424j600_buf_rd(ERXDATA, buf, 20);

    // nuskaityti preambule
    wNextPacketPointer = (WORD)buf[0] | ((WORD)buf[1] << 8); // kito paketo
adresas
    temp = 0;
    if(buf[4] & 0x10) temp |= 1; // CRC error
    if(buf[4] & 0x20) temp |= 2; // Length error
    //if(buf[4] & 0x40) temp |= 4; // Length range
    if((buf[4] & 0x80)==0) temp |= 8; // not OK
    if(temp != 0)
    {
        #if (USE_STREAM>0) && (USE_ERROR>0)
            xfprint(OUTDEV_ERROR,"\n\rETH_ReceiveFrame:: err: invalid frame");
        #endif
    }
    temp =  (WORD)buf[2] | ((WORD)buf[3] << 8);             // paketo ilgis
    ptr = ETH_AllocateRxBuffer(temp);
    //ptr = NULL;
    if(ptr != NULL)
    {
        Enc424j600_Write(ERXRDPT, wCurrentPacketPointer + 8);
        Enc424j600_buf_rd(ERXDATA, ptr, temp);    // nuskaityti paketa
    }
    else
    {
        #if (USE_STREAM>0) && (USE_ERROR>0)
            xfprint(OUTDEV_ERROR,"\n\rETH_Receive_Frame::   ETH_AllocateRxBuffer
failed");
        #endif
    }

    wNewRXTail = wNextPacketPointer - 2;
    if(wNextPacketPointer == RXSTART) wNewRXTail = ENC100_RAM_SIZE - 2;

    // Decrement the RX packet counter register, EPKTCNT
```

```
    Enc424j600_bsf(ECON1, ECON1_PKTDEC);
    // Move the receive read pointer to unwrite-protect the memory used by the
    // last packet.  The writing order is important: set the low byte first,
    // high byte last (handled automatically in WriteReg()).
    Enc424j600_Write(ERXTAIL, wNewRXTail);

    #if USE_ETHERNET_MUTEX>0
    osMutexRelease(EthMutex);
    #endif

    #if ETH_DEBUG>0
        xfprint(OUTDEV_UART0,"\n\rETH_ReceiveFrame:: %d",temp);
    #endif
    #if ETH_DEBUG>1
        ETH_PrintIPHeader((char *)ptr, temp);
    #endif
    #if ETH_DEBUG>2
        ETH_PrintFrame(ptr, temp);
    #endif

    if(ptr != NULL) ETH_PutRxMessage(temp);
}
```

7 priedas. Paketo siuntimo kodas

```
void ETH_SendFrame(unsigned char *data, unsigned long length)
{
    unsigned short temp;
    temp = 0;
    #if USE_ETHERNET_MUTEX>0
        if(!osMutexWait(EthMutex, 300))
        {
            #if (USE_STREAM>0) && (USE_ERROR>0)
            xfprint(OUTDEV_ERROR,"\n\rETH_SendFrame:: mutex timeout");
            #endif
            return;
        }
    #endif
    while(Enc424j600_Read(ECON1) & ECON1_TXRTS) // ar pries tai buves siuntimas
baigtas?
    {
        if(temp == 10)
        {
            #if (USE_STREAM>0) && (USE_ERROR>0)
                xfprint(OUTDEV_ERROR,"\n\rETH_SendFrame:: failed send");
            #endif
            #if USE_ETHERNET_MUTEX>0
                osMutexRelease(EthMutex);
            #endif
            return;
        }
        ETH_DELAY( 20000 );
        temp++;
    }

    Enc424j600_Write(EGPWRPT, TXSTART);
    Enc424j600_buf_wr(EGPDATA, data, length);
    Enc424j600_Write(ETXLEN, length);
    Enc424j600_bsf(ECON1, ECON1_TXRTS); // ijungti siuntima
    #if USE_ETHERNET_MUTEX>0
    osMutexRelease(EthMutex);
    #endif
    #if ETH_DEBUG>0
        xfprint(OUTDEV_UART0,"\n\rETH_SendFrame:: send %d",length);
    #endif
    #if ETH_DEBUG>1
        ETH_PrintIPHeader((char *)data, length);
    #endif
    #if ETH_DEBUG>2
        ETH_PrintFrame(data, length);
    #endif
}
```

8 priedas. Analoginio-skaitmeninio keitiklio kodas

```
void adc_init(void)
{
    if(PinRead(PIN_AD_OE))
    {
        PinClear(PIN_AD_OE);
    }
}

int adc_read(void)
{
    int temp=0;
        for(i=0; i<8; i++)
        {
            temp |= adc_check(PIN_AD_D0 + i) << i; // Nuskaitom 1 ar 0, pastumiam
ir uzORinam

        }
    int result = temp * 9; // verciam milivoltais, nes 0-2.28 V kinta ir 255
stepai, tai 2.28/255*1000=8.941176, kolkas 9, nes nereik tikslumo
    return result;
}

int adc_check(int koja)
{

    if(PinRead(koja)) return 1;
    else return 0;
}

void adc_clock(void)
{
    if (adcTimer %2 ==0) //Lyginis PinSet -> Nelyginis PinClear
    {
        PinSet(PIN_AD_CLK);
    }
    else
    {
        PinClear(PIN_AD_CLK);
    }
}
```

9 priedas. Duomenų rašymo į masyvą kodas

```
void Timer1Func(void)
{
    int temp;
    adcTimer++;
    adc_clock();
    if (adcTimer %2 !=0)
    {
        if(AdcStatus & ADC_ENABLE)
        {
            temp = adc_read();
            if(temp > AdcTriger && (AdcStatus & ADC_TRIGER) == 0)
            {
                AdcTrigerIndex = 0;
                AdcIndex = 0;
                AdcStatus |= ADC_TRIGER;
            }
            AdcArray[AdcIndex] = temp;
            AdcIndex++;
            if(AdcIndex >= ADC_ARRAY_SIZE) AdcIndex = 0;
            if(AdcStatus & ADC_TRIGER && AdcTriger>0)
            {
                AdcTrigerIndex++;
                if(AdcTrigerIndex >= (ADC_ARRAY_SIZE))
                {
                    AdcStatus = ADC_COMPLETED;
                }
            }
        }
    }
}
void AdcStart(void)
{
    AdcIndex = 0;
    AdcStatus = ADC_ENABLE;
}

// 0 - nepabaige, <>0 pabaige
int IsAdcCompleted(void)
{
    return  AdcStatus & ADC_COMPLETED;
}

void AdcGetArray(int *val)
{
    int i;
    for(i=0; i< ADC_ARRAY_SIZE; i++)
    {
        val[i] = AdcArray[i];
    }
}
```

10 priedas. Ajax užklausos kodas

```
int fs_open_custom(struct fs_file *file, const char *name)
{
    const ROM_FILE_TABLE *pTable;
    //void *ptr;
    int i = 0;

xfprint(OUTDEV_DEBUG,"\n\rfs_open_custom %s ", name);

    //------------  ajax file  ------------------------
    file->file_type = ssi_detect_file_type(name, "cgx");
    if(file->file_type)
    {
        if (strcmp(name, "/1.cgx") == 0)
        {
            file->data       = (const char *)SSI_String;
            file->len        = AjaxHandler(0);
            file->index      = file->len;
            file->pextension = NULL;
            file->http_header_included = 1;
            xfprint(OUTDEV_DEBUG,"ajax");
            return ERR_OK;
        }
    }
    //------------  rom file  ------------------------
    while(1)
    {
        pTable = &RomFileTable[i];
        if((pTable->offset == 0) && (pTable->length == 0)) break;
        if (strcmp(pTable->name, name) == 0)
        {
            file->data       = (const char *)&RomFile[pTable->offset];
            file->len        = pTable->length;
            file->index      = pTable->length;
            file->pextension = NULL;
            file->http_header_included = 1;
            #if HTTPD_PRECALCULATED_CHECKSUM
                #error Precalculated checksum is absent!
            #endif /* HTTPD_PRECALCULATED_CHECKSUM */
            #if LWIP_HTTPD_FILE_STATE
                file->state = fs_state_init(file, name);
            #endif /* #if LWIP_HTTPD_FILE_STATE */


            xfprint(OUTDEV_DEBUG,"ok");

            return ERR_OK;
        }
        i++;
    }

    xfprint(OUTDEV_DEBUG,"fail");
    return ERR_VAL;
}

int AjaxHandler(int index)
 {
    int i;
      switch(index)
      {
        case 0:
                        AdcGetArray(ajaxArray);
```

```
                        xsprint(SSI_String, "%d ", ajaxArray[0]);
                        for(i=1;i<ADC_ARRAY_SIZE;i++)
                        {
                            xsprint(tempString, "%d ", ajaxArray[i]);
                            strcat(SSI_String, tempString);
                        }
            break;
         default:
                  xsprint(SSI_String, "??");
            break;
     }
     return(strlen(SSI_String));
 }
```

11 priedas. CGI skripto kodas

```
extern long AdcTriger;
const char *CGIHandler(int iIndex, int iNumParams, char *pcParam[],
                            char *pcValue[])
{
    unsigned int i;
    if (iIndex==0)
      {
        for (i=0; i<iNumParams; i++)
        {
              if (strcmp(pcParam[i] , "trigger")==0)
              {
              AdcStart();
                    AdcTriger = strtol(pcValue[i],NULL,10);
                    xfprint(0,"\n\rTrigger=%d",AdcTriger);
              }
        }
      }


      return "/testcgi.cgi";
}

void cgi_init(void)
{
    const tCGI CGI_Config = {"/testcgi.cgi", CGIHandler};
    CGI_TAB[0] = CGI_Config;
    http_set_cgi_handlers(&CGI_TAB[0], 1);
}
```

12 priedas. Internetinio puslapio kodas

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"><html>
<head>
    <link rel="stylesheet" type="text/css" href="style.css">
    <title>Ajax testas</title>
    <meta http-equiv="Cache-Control" content="no-store" />
    <meta http-equiv="Content-Type" content="text/html; charset=utf-8">
    <script>

var dataArray;
var dataArray2=[20];
var dataCnt=0;
var dx=5;
var dy=2;
var trigger=0;
function draw(data)
{
  var canvas = document.getElementById("canvas");
  if (null==canvas || !canvas.getContext) return;

  var axes={}, ctx=canvas.getContext("2d");
  axes.x0 = .5 + 0.1*canvas.width;  // x0 pixels from left to x=0
  axes.y0 = .5 + .5*canvas.height; // y0 pixels from top to y=0
  axes.scale = 40;                 // 40 pixels from x=0 to x=1
  axes.doNegativeX = true;
    funGraph(data,ctx,axes,"rgb(11,153,11)",3);
    if(trigger!=0){
        triggerGraph(ctx);
    }
}
function getMaxOfArray(arr) {
      return Math.max.apply(null, arr);
}

function funGraph (data,ctx,axes,color,thick)
{
  var xx, yy, x0=axes.x0, y0=axes.y0, scale=axes.scale;
  var iMax = Math.round((ctx.canvas.width-x0)/dx);
  var iMin = axes.doNegativeX ? Math.round(-x0/dx) : 0;
  ctx.clearRect(0, 0, canvas.width, canvas.height);
  showAxes(ctx,axes);
  ctx.beginPath();
  ctx.lineWidth = thick;
  ctx.strokeStyle = color;
  for (var i=0; i<data.length; i++)
  {
    xx = dx*i;
    yy = dy*data[i];
    if (i==0) ctx.moveTo(x0+xx,y0-yy);
    else      ctx.lineTo(x0+xx,y0-yy);
  }
  ctx.stroke();
  axisValues(ctx);
}
function splitData(data)
        {
            var splitTimes=1;
            var canvas = document.getElementById("canvas");
                    if(dataCnt > data.length)
                        {
                            dataCnt=0;
```

```
                    }
                if(dataCnt <= data.length)
                    {
                        dataArray2[dataCnt]=data[dataCnt];
                        draw(dataArray2);
                        dataCnt++;
                    }
        }
function showAxes(ctx,axes)
{
  var x0=axes.x0, w=ctx.canvas.width;
  var y0=axes.y0, h=ctx.canvas.height;
  var xmin = axes.doNegativeX ? 0 : x0;
  ctx.beginPath();
  ctx.strokeStyle = "rgb(128,128,128)";
  ctx.moveTo(xmin,y0); ctx.lineTo(w,y0);  // X axis
  ctx.moveTo(x0,0);    ctx.lineTo(x0,h);  // Y axis
  ctx.moveTo(xmin,y0+0.5*h); ctx.lineTo(w,y0+0.5*h);  // X axis
  ctx.moveTo(x0+0.9*w,0);    ctx.lineTo(x0+0.9*w,h);  // Y axis
  ctx.stroke();
  ctx.lineWidth = .5;
  ctx.moveTo(xmin,y0+0.1*h); ctx.lineTo(w,y0+0.1*h);  // X axis
  ctx.moveTo(x0+0.1*w,0);    ctx.lineTo(x0+0.1*w,h);  // Y axis
  ctx.moveTo(xmin,y0+0.2*h); ctx.lineTo(w,y0+0.2*h);  // X axis
  ctx.moveTo(x0+0.2*w,0);    ctx.lineTo(x0+0.2*w,h);  // Y axis
  ctx.moveTo(xmin,y0+0.3*h); ctx.lineTo(w,y0+0.3*h);  // X axis
  ctx.moveTo(x0+0.3*w,0);    ctx.lineTo(x0+0.3*w,h);  // Y axis
  ctx.moveTo(xmin,y0+0.4*h); ctx.lineTo(w,y0+0.4*h);  // X axis
  ctx.moveTo(x0+0.4*w,0);    ctx.lineTo(x0+0.4*w,h);  // Y axis
  ctx.moveTo(xmin,y0-0.1*h); ctx.lineTo(w,y0-0.1*h);  // X axis
  ctx.moveTo(x0+0.5*w,0);    ctx.lineTo(x0+0.5*w,h);  // Y axis
  ctx.moveTo(xmin,y0-0.2*h); ctx.lineTo(w,y0-0.2*h);  // X axis
  ctx.moveTo(x0+0.6*w,0);    ctx.lineTo(x0+0.6*w,h);  // Y axis
  ctx.moveTo(xmin,y0-0.3*h); ctx.lineTo(w,y0-0.3*h);  // X axis
  ctx.moveTo(x0+0.7*w,0);    ctx.lineTo(x0+0.7*w,h);  // Y axis
  ctx.moveTo(xmin,y0-0.4*h); ctx.lineTo(w,y0-0.4*h);  // X axis
  ctx.moveTo(x0+0.8*w,0);    ctx.lineTo(x0+0.8*w,h);  // Y axis
  ctx.moveTo(xmin,y0-0.5*h); ctx.lineTo(w,y0-0.5*h);  // X axis
  ctx.moveTo(x0-0.1*w,0);    ctx.lineTo(x0-0.1*w,h);  // Y axis
  ctx.stroke();
}
function triggerSubmit()
{
    var xmlHttp = new XMLHttpRequest();
    var theUrl = "/testcgi.cgi?trigger=" + trigger;
    xmlHttp.open("GET", theUrl, true); // true for asynchronous
    xmlHttp.send(null);
}

   function AjaxSubmit()
    {
      var xhr;
      try {  xhr = new ActiveXObject('Msxml2.XMLHTTP');   }
      catch (e)
      {
        try {   xhr = new ActiveXObject('Microsoft.XMLHTTP');    }
        catch (e2)
        {
          try {  xhr = new XMLHttpRequest();     }
          catch (e3) {  xhr = false;   }
        }
     }
     xhr.responseType = "text";
     xhr.onreadystatechange  = function()
```

```
     {
          if(xhr.readyState  == 4)
          {
               if(xhr.status  == 200)
               {
                   var restxt = xhr.responseText;
                   dataArray = restxt.split(" ");
                   for(var  i=0;  i<dataArray.length-1;  i++)  {  dataArray[i]  =
parseInt(dataArray[i], 10); }
                   document.getElementById('txt').value = dataArray.toString();
               }
               else
                  document.getElementById('txt').value   =   "Error   code   "   +
xhr.status;
          }
     };

    xhr.open('GET', "3.cgx",  true);
    xhr.send(null);
   }
    function dxTest()
        {
            var temp = document.getElementById("dx").value;
            dx = parseInt(temp,10);
            dataArray2 = [];
            dataCnt = 0;
            splitCnt = 0;
        }
        function dyTest()
        {
            var canvas = document.getElementById("canvas");
            var temp = document.getElementById("dy").value;
            var h = canvas.height;
            dy = (h/2) / parseFloat(temp);
            dataArray2 = [];
            dataCnt = 0;
            splitCnt = 0;
        }
        function dyScale()
        {
            var canvas = document.getElementById("canvas");
            dy = canvas.height/(getMaxOfArray(dataArray))/2;
            document.getElementById("dy").value= Math.round(h/2 / dy * 100) / 100;
            dataArray2 = [];
            dataCnt = 0;
            splitCnt = 0;
        }
    function axisValues(ctx)
        {
            h=ctx.canvas.height;
            w=ctx.canvas.width;
            ctx.fillStyle = "black";
            ctx.fillText(Math.round(h/2 / dy * 100)/100, .1*w+2 ,10);
            ctx.fillText(0.8*Math.round(h/2 / dy * 100 )/ 100, .1*w+2 ,.1*h+10);
            ctx.fillText(0.6*Math.round(h/2 / dy * 100 )/ 100, .1*w+2 ,.2*h+10);
            ctx.fillText(0.4*Math.round(h/2 / dy * 100 )/ 100, .1*w+2 ,.3*h+10);
            ctx.fillText(0.2*Math.round(h/2 / dy * 100 )/ 100, .1*w+2 ,.4*h+10);
            ctx.fillText(0, .1*w+2 ,.5*h+10);
        }
    function triggerGraph(ctx)
        {
            ctx.fillStyle = "#ffff00";
            ctx.fillRect(0.1*w-30,0.5*h-2-trigger*dy,30,5);
        }
```

```
    function triggerSlide()
        {
            var canvas = document.getElementById("canvas");
            var temp = document.getElementById("triggerSlide").value;
            trigger =  Math.round(temp * h/2 / dy)
            document.getElementById("triggerSpan").innerHTML = trigger;
        }
setInterval('AjaxSubmit();', 500);
setInterval('splitData(dataArray);', 1);
setInterval('triggerSlide();', 50);
</script>
</head>
<body>
  <h1>Graphics testas </h1>
        <div id="divhome">
        <a class="button" href="index.htm"><strong>Namai</strong></a>
         <a class="button" href="testssi.shtml"><strong>SSI testas</strong></a>
         <a class="button" href="ajax10.htm"><strong>Ajax test 10</strong></a>
            <a  class="button    active"  href="ajax11.htm"><strong>Ajax  test
11</strong></a>
         <a class="button" href="testcgi.htm"><strong>CGI testas</strong></a>
    </div>
    <div id='divmain'>
        <FORM method="POST" name='ajax' action="">
            <INPUT type="text" name="dyn" id='txt' value="">
         <br>Trigger (0=off):<span id="triggerSpan">0</span> <br>
                <input  type="range"  min="0"  max="1"  value="0"  step="0.01"
id="triggerSlide"    />    <INPUT    type="BUTTON"    value="Ivesti/Perkrauti"
ONCLICK="triggerSubmit()">
         <br>DX:          <INPUT  type="text"  name="dx"
id='dx' value="5">
                  <INPUT type="BUTTON" value="OK"  ONCLICK="dxTest()">
         <br>Y scale:   <INPUT type="text" name="dy" id="dy" value="100">
                <INPUT type="BUTTON" value="OK"  ONCLICK="dyTest()">
                <INPUT type="BUTTON" value="Auto Scale"  ONCLICK="dyScale()">

  </div>
      <br>
      <div id='divcanvas'>
      <canvas id="canvas" width="550" height="400"></canvas>
      </div>

</body>
</html>
```