Elmo Aho

Langaton HTTP-palvelin National Instrumentsin robotin ohjaukseen

Metropolia Ammattikorkeakoulu

Insinööri (AMK)

Sähkötekniikan koulutusohjelma

Insinöörityö

18.12.2013

Metropolia

Tekijä Otsikko Sivumäärä Aika	Elmo Aho Langaton HTTP-palvelin National Instrumentsin robotin ohjaukseen 22 sivua + 4 liitettä 18.12.2013
Tutkinto	insinööri (AMK)
Koulutusohjelma	elektroniikka
Suuntautumisvaihtoehto	elektroniikkasuunnittelu
Ohjaaja	lehtori Janne Mäntykoski

Insinöörityössä tehtiin langaton HTTP-palvelin käyttämällä mbed-mikrokontrolleria ja siihen liitettyä Wifly-moduulia. Palvelimen oli tarkoitus ohjata National Instrumentsin tarjoamaa robottia. Langattomuus toteutettiin käyttämällä Wifly WLAN -moduulia, joka käyttää IEEE 802.11 b/g -standardin mukaista langatonta verkkoa. Robotin ohjaukseen käytettiin yksinkertaista HTML-sivua.

Palvelin välitti ohjaustiedon robotille SPI-väylän kautta 8-bittisella tavulla. HTTP-palvelin saatiin toimimaan, mutta Wiflyn rajoitusten vuoksi, se ei pystynyt palvelemaan useaa asiakasta kerrallaan. Sen käyttöliittymä jäi myös suhteellisen yksinkertaiseksi robotin ja palvelimen välisten kommunikointiongelmien vuoksi.

Kommunikaatio-ongelmat saatiin selvitettyä myöhemmin, ja robottia pystyttiin ohjaamaan käyttämällä palvelinta. Tulevaisuudessa työtä voidaan kehittää tekemällä palvelimesta käyttäjäystävällisempi. Tämä voidaan saavuttaa esimerkiksi lisäämällä näyttö palvelimeen, josta nähdään palvelimeen liittyviä tietoja kuten sen IP-osoite. Lisäksi käytettyä HTML-sivua voisi kehittää paremman näköiseksi ja lisätä monimutkaisempia komentoja robotille.

Avainsanat	http-palvelin, mikrokontrolleri, mbed

Author Title Number of Pages Date	Elmo Aho Wireless HTTP-server for National Instruments robot 22 pages + 4 appendices 18 December 2013
Degree	Bachelor of Engineering
Degree Programme	Electronics
Specialisation option	Electronics design
Instructor	Janne Mäntykoski, Senior Lecturer

This final year project is about making a wireless http-server using a microcontroller and a Wifly-module. The server was made to control a robot provided by National Instruments. The Wifly-module was used to achieve a wireless connection that uses IEEE 802.11 b/g standard. The microcontroller was on an mbed Development board. The robot was controlled via a simple HTML site by using simple commands like forward and backward.

The Server communicated with the robot via SPI bus. A 8-bit byte was sent as the control information. The HTTP-server worked but it was somewhat unreliable. Its interface was also left somewhat unfinished because of communication problems between the robot and the server.

The communication problems were later solved and the robot could be controlled by the server. In future the server could be developed by making it more user friendly. This could be achieved by adding a screen to display important information about the server such as its IP address. Also the HTML site could be made to look better and more complicated commands could be added to control the robot.

Keywords	http-server, microcontroller, mbed

Sisällys

Lyhenteet ja käsitteet

ADC	Analog to Digital Converter; nasta, josta laite lukee analogisen tiedon ja muuttaa sen digitaaliseksi
ARM	Advanced RISC Machines; maailman eniten valmistettu 32-bittinen mikroprosessoriarkkitehtuuri
DRAM	Dynamic Random Access Memory; luku- ja kirjoitus muisti, jota käytetään ns. keskusmuistina
EAGLE	Easily Applicaple Graphical Layout Editor; monipuolinen piirilevynsuunnitteluohjelma
FPGA	Field-programmable Gate Array; digitaalinen mikropiiri, joka on helposti uudelleen ohjelmoitavissa
I/O nasta	Input/Output nasta; nasta, jonka kautta kirjoitetaan tai luetaan tietoa
IEEE	Institute of Electrical and Electronics Engineers; ammatillinen elektroniikan alan järjestö, jonka tarkoituksena on koulutuksen edistäminen ja standardien määrittely
Kuvauskieli	Ohjelmointikieli, joka kertoo ohjelmalle, miten dokumentti muotoillaan halutun näköiseksi
SPI	Serial Peripheral Interface Bus; synkroninen kaksisuuntainen sarjaväylä
UART	Universal Asynchronous Receiver Transmitter; asynkroninen sarjaväylä
WIFI, WLAN	Wireless Local Area Network; langaton lähiverkko, joka käyttää tavallisesti IEEE 802.11 standardia

1 Johdanto

Tässä insinöörityössä esitellään ulkoista mikrokontrolleria käyttävä langaton HTTP-palvelin, joka välittää tietoa National Instrumentsin robotille. HTTP-palvelin lähettää ja vastaanottaa tietoa SPI-väylän kautta robotin sbRIO-ohjaimen kanssa. Langattomaan kommunikointiin käytetään IEEE 802.11 b/g -standardien mukaista langatonta verkkoa. Langattomuus toteutettiin käyttämällä Wifly WLAN -moduulia. Palvelin taas toimi mbed-mikrokontrollerilla, joka ohjelmoitiin käyttämällä C++-ohjelmointikieltä.

Palvelin lähettää verkkosivun asiakaslaitteelle ja vastaanottaa robotin ohjaustiedon asiakaslaitteelta. Näin robottia voidaan ohjata esimerkiksi matkapuhelimella langattomasti verkkosivun kautta. Verkkosivu sisältää nappeja, jotka välittävät robotille yksinkertaisia käskyjä, kuten eteen ja taakse. Palvelimen virrankulutus suunniteltiin pieneksi, koska robotti on akkukäyttöinen. Robotin on tarkoitus toimia esimerkiksi messuilla National Instrumentsin esittelylaitteena.

Tässä insinöörityössä käsitellään vain robotin langatonta HTTP-palvelinta. Itse robotin ohjelmointi tehtiin Metropolia Ammattikorkeakoulussa, Aleksi Oravan insinöörityönä.

2 Verkkoprotokollat

2.1 OSI-malli

ISO:n kansainvälinen standardi OSI-malli on luotu kuvaamaan sitä miten tieto kulkee tietoverkossa paikasta toiseen. Se koostuu seitsemästä eri kerroksesta, jotka esitellään taulukossa 1. (ks. seur. s.):

Taulukko 1. OSI-mallin kerrokset ja protokollaesimerkit

Tavallisesti osana	**Kerros**	**Esimerkiksi**
ohjelmaa	7. sovelluskerros	HTTP, ftp
	6. esitystapakerros	GIF, JPG
	5. istuntokerros	Internet Socket, NetBIOS
tiedonsiirtoa	4. kuljetuskerros	TCP, UDP
	3. verkkokerros	IPv4/IPv6, IPX
	2. siirtokerros	IEEE 802 MAC LCC, Ethernet
	1. fyysinen kerros	USB, DSL, Wifi

Näistä kerroksista jokainen käyttää yhtä alemman kerroksen palveluista ja toimittaa palveluja yläpuoleiselle kerrokselle. Tieto liikkuu mallin mukaan lähetettäessä ylimmästä alimpaan käymällä kaikki kerrokset läpi ja vastaanotettaessa alimmasta ylimpään. Kohde sovellusten välillä tieto voi liikkua myös linkistä toiseen käyttäen vain tiedonsiirtokerroksia.

Työssä tehtiin http-palvelin, jonka fyysisenä ja siirtokerroksena käytettiin Wi-Fi:ä, verkkokerroksena IPv4:ää, kuljetuskerroksena TCP:tä, istuntokerroksena Internet Socketia, esitystapakerroksen useita HTML-sivusta löytyviä esitystapoja ja sovelluskerroksena HTTP:tä.

Käytännössä suuret verkot eivät toimi täysin OSI-mallin mukaisesti. Esimerkiksi mobiiliverkoista löytyy useita kerroksia verkkokerroksen alapuolella ja esimerkiksi IP-paketteja voidaan lähettää toisten IP-pakettien sisällä. [1.]

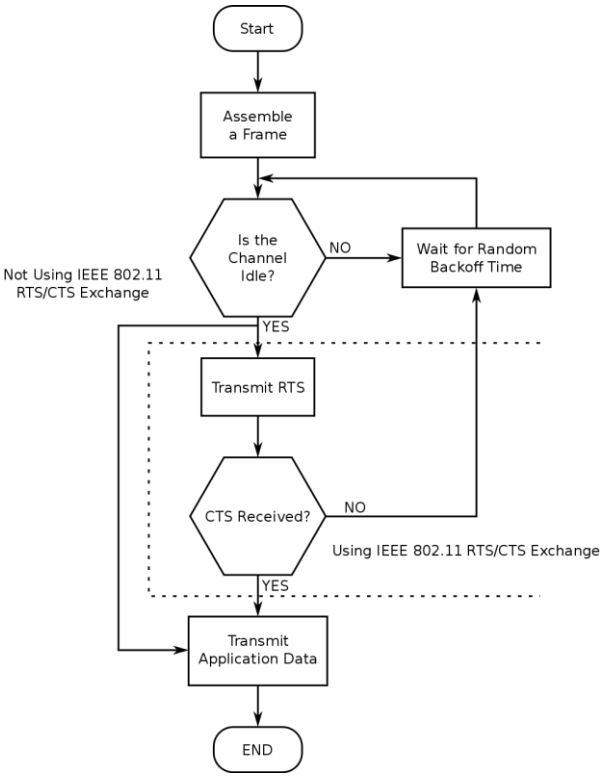
2.2 Langaton verkko

IEEE 802.11 on IEEE:n luoma ja ylläpitämä standardi langattomille lähiverkoille eli WLAN-verkoille. Se sisältää fyysisen kerroksen ja siirtokerroksen, jota kutsutaan nimellä MAC. Standardi noudattaa pääosin samaa perusprotokollaa, mutta se on jaettu eri

versioihin, jotka eivät kaikki ole keskenään yhteensopivia. Työssä käytettiin IEEE 802.11 b- ja g -versioita tukevaa WLAN-lähiverkkoa. [2.]

Fyysinen kerros, joka tunnetaan myös nimellä PHY, vastaa merkkien koodauksesta, purkamisesta, lähetyksestä ja vastaanotosta. Se toimii MAC-siirtokerroksen kanssa ja toimittaa tietopaketin eteenpäin seuraavalle laitteelle. MAC-kerros puolestaan vastaa mm. lähetetyn tiedon kehyksistä, siirtotien varausmenetelmästä, salauksesta ja korruptoituneiden kehysten uudelleen lähetyksestä.

Alkuperäinen IEEE 802.11 -standardi käytti siirtotien varaukseen CSMA/CA-menetelmää. Tätä siirtotien varausmenetelmää käyttävät kaikki muutkin 802.11 standardit. Menetelmässä kuunnellaan kanavan liikennettä ennen paketin lähettämistä, kuten nähdään seuraavassa kuvassa 1. Jos kanavalla ei ole liikennettä lähetetään RTS (*Request to Send*), ja jos vastaus CTS (*Clear to Send*) saadaan, lähetetään varsinainen data. [3.]



Kuva 1. CSMA/CA-siirtotien varausmenetelmän yksinkertaistettu toimintatapa [3]

802.11b on julkaistu vuonna 1999 parannuksena ensimmäiseen 802.11-verkoon. Se toimii 2.4 GHz:n taajuudella ja sisältää 14 mahdollista kanavaa 5 MHz:n välein. Verkon maksimi siirtonopeus on 11 Mbit/s, mutta koska se käyttää CSMA/CA-siirtotien varausmenetelmää päästään TCP/IP-protokollaa käyttämällä noin 5.9 Mbit/s:n nopeuteen.

802.11g on vuonna 2003 julkaistu kolmas 802.11-standardin mukainen langaton verkko. Se toimii samalla taajuudella ja kanavilla kuin 802.11b, mutta pystyy 54 Mbit/s:n nopeuteen. Se on täysin taaksepäin yhteensopiva 802.11b-verkon kanssa. Suuremman nopeuden mahdollistaa eri modulointimenetelmä kuin 802.11b-versiossa, mutta tarvittaessa 802.11g pystyy käyttämään myös samaa menetelmää kuin 802.11b-verkko. [4.]

Standardin mukaiset verkot toimivat topologialtaan tähtimuotoisesti tukiasemien toimiessa tähden keskustana. Langaton laite ottaa yhteyden WLAN-tukiasemaan, joka mahdollisesti taas on yhteydessä laajempiin verkkoihin joko ethernetkaapelilla tai langattomasti käyttäen Wireless Distribution Systeemiä eli WDS:sää.

2.3 Tietoturva

Työssä käytettiin WPA eli *Wi-Fi Protected Access* -salausta. Se julkaistiin IEEE 802.11i standardin luonnoksessa korjaamaan sitä edeltävässä WEP-salauksessa havaittuja haavoittuvuuksia. WPA käyttää salaukseen TKIP eli Temporal Key Integrity -protokollaa, joka parantaa turvallisuutta toimimalla kuitenkin samalla laitteistolla kuin WEP. Näin se voitiin päivittää laitteisiin, jotka käyttivät WEP-salausta ohjelmistopäivityksenä.

TKIP sekoittaa käyttäjän antaman salasanan sekoitusavaimella ennen sen lähettämistä. Tämä avain sekoitetaan uudestaan joka paketille erikseen. Lisäksi TKIP käyttää laskuria sekvensoimaan sen lähettämät paketit, joten vastaanottaja pystyy hylkäämään väärässä järjestyksessä tulleet paketit. TKIP myös kryptaa salausavaimen useasti, jolloin salauksen purkaminen vaikeutuu. [5, s. 43.]

2.4 TCP/IP-protokolla

TCP/IP on kokoelma protokollia, joista tärkeimmät ovat TCP ja IP. Sitä käytetään monissa eri sovelluksissa, suurimpana näistä mainittakoon maailmanlaajuinen internet. Kaikki TCP/IP-protokollat on määritelty kaikille saatavissa olevissa RFC-standardeissa.

IP eli *Internet Protocol* on OSI-mallin mukaan verkkokerros. Se tarjoaa perustan pakettien kuljetukseen tietoverkossa. Se toimii lähetä ja unohda periaatteella joten se ei takaa pakettien perillemenoa. Tämän hoitaa verkon TCP-osa. Käytetty IP-versio 4 käyttää 32-bittistä IP-osoitetta. OSI-mallin mukaisesti IP yhdistyy alempaan siirtokerrokseen, ja se käyttää ARP (*Address Resolution Protocol*) -protokollaa. ARP yhdistää IP-osoitteen tiettyyn MAC-osoitteeseen, jotta laitetason kommunikointi toimii. [6.]

TCP eli *Transmission Control Protocol* tarjoaa luotettavan tiedonsiirtopalvelun kahden verkossa olevan laitteen välille. Se lisää tarkistussummia, numeroi lähetetyn tiedon ja kuittaa vastaanotetut datapaketit lähettäjälle. Tämä varmistaa tiedon perille menon. TCP vastaa myös tiedon kulun nopeudesta. Se säätää lähetysnopeutta hyödyntämällä kaistaa mahdollisimman tehokkaasti.

TCP käyttää 16-bittistä porttinumeroa erottamaan sen muista protokollista ja erottamaan eri TCP:tä käyttävät ohjelmat toisistaan. Esimerkiksi työssä käytetty HTTP-protokolla käyttää tavallisesti porttia 80. [7.]

2.4.1 Verkko-ohjelmoitirajapinta

Ohjelmoitaessa ohjelmaa joka käyttää tietoverkkoa tarvitaan ohjelmointirajapintaa jolla käytetään tietoverkkoa. Ohjelmointirajapinta ohjaa ohjelman toimintaa antamalla ohjelmoijan määritellä tarvittavat tiedot kuten osoitteen, portin ja käytettävän protokollan. Tämä ohjelmointirajapinta on nimeltään *Network Socket*. Se on tavallisesti saatavilla käyttöjärjestelmässä. Työssä *Network Socket* -ohjelmointirajapinta saatiin mbedin socket.h-kirjastosta.

Mbedin *Network Socket* toimi ensiksi määrittelemällä serverin TCP-portti *server*-olion funktiolla *bind*. Porttina käytettiin yleisesti HTTP-protokollassa käytettyä porttia 80. Tämän jälkeen käskettiin ohjelman kuunnella porttia komennolla *listen*. Kun porttia kuunneltiin voitiin funktiolla *accept* hyväksyä ohjelmaan otettu yhteys. Yhteyden muodostuk-

sen jälkeen voitiin vastaanottaa tai lähettää tietoa muodostetun yhteyden kassa käyttämällä *client*-olion funktioita *receive* ja *send*. [8.]

2.4.2 DHCP-protokolla

Verkon käyttäjä saa IP-osoitteen tavallisesti DHCP eli *Dynamic Host Configuration Protocol* –palvelimelta, joka sijaitsee normaalisti reitittimellä, johon laite on yhteydessä. DHCP-serveriä käyttämällä vältytään IP-osotteiden päällekkäisyyksiltä. Osoitetta pyydettäessä käytetään UDP protokollaa, joka on kuin TCP:n yksinkertaisempi muoto. Se ei vaadi yhteyden muodostusta laitteiden välille eikä takaa viestin perillemenoa.

Työssä käytetty Wifly -moduuli käyttää oletusasetuksena DHCP-palvelinta, joka sopii hyvin työn tarkoitukseen. Wiflyyn voidaan tarvittaessa asettaa IP-osoite myös manuaalisesti.

2.5 HTTP-protokolla

Hypertext Transfer Protocol eli HTTP toimii WWW:n pohjana. Se käyttää tavallisesti TCP/IP:tä liikuttamaan tietoa päätteestä toiseen. Tieto voi olla tekstiä tai binääridataa kuten kuvia, videoita tai ohjelmia.

HTTP-palvelin kuuntelee tavallisesti päällä ollessaan pyyntöjä TCP-portilla 80. Http-tiedonsiirto alkaa, kun asiakaslaite lähettää palvelimelle metodin eli pyynnön. Protokollassa on useita eri metodeita, joiden tarkoituksena on ilmoittaa haluttu toiminto. Tavallisessa sivunhaussa käytetään GET-metodia, joka hakee pyydetyn sivun, eikä tee mitään muuta. GET-metodin pitäisi aina tuottaa sama tulos. [9, s.50.]

Muita metodeja ovat mm. POST, joka lähettää esimerkiksi sivulla täytetyn lomakkeen tiedot palvelimelle, HEAD, joka pyytää palvelimelta vain sivun otsikkotiedot ja CONNECT, joka pyytää palvelinta muodostamaan TCP/IP-tunnelin.

```
GET /index.html HTTP/1.1
Host: www.esim.fi
```

Esimerkkikoodi 1. HTTP GET -pyyntö

GET-metodi sisältää vähintään tiedon halutusta sivusta, http:n version ja palvelimen nimen (Tämä on pakollista HTTP:n versiosta 1.1 ylöspäin.). Lisäksi se voi sisältää useita muita tietoja, jotka voivat kertoa mm. käytetystä selaimesta, sallituista ohjelmista ja yhteyden käsittelytavasta.

```
HTTP/1.1 200 OK
Content-Type: text/html; charset=UTF-8
Content-Length: 105
Connection: close


<html>
<head>
  <title>Esimerkki sivu</title>
</head>
<body>
  Tässä on tekstiä.
</body>
</html>
```

Esimerkkikoodi 2. HTTP GET -pyyntöön vastaus

Palvelin vastaa, mihin tahansa pyyntöön ensimmäiseksi status-koodilla, joka kertoo onko pyyntö onnistunut tai mahdollisen epäonnistumisen syyn. Kun GET pyyntö on onnistunut palvelin vastaa http-versio numerolla ja statuskoodilla 200 OK. Tämän jälkeen palvelin lähettää vastauksen otsikko-osan, joka voi sisältää monenlaista tietoa palvelimesta ja sivusta. Otsikko-osan tiedot ovat vapaavalinnaisia, joten niitä ei teoriassa tarvita. Otsikon jälkeen palvelin lähettää html-sivun, joka alkaa kuten esimerkissä yllä <html> ja päättyy </html>. Tämän jälkeen palvelin sulkee yhteyden, ellei otsikko-osassa toisin mainita.

2.6 HTML-kuvauskieli

HTML eli *HyperText Markup Language* on kuvauskieli, jolla luodaan HTML-sivu, jonka verkkoselain taas kääntää verkkosivuksi. HTML-koodi muodostuu sisäkkäisistä ja peräkkäisistä elementeistä. Seuraavassa esimerkissä esitellään yksinkertaisen html-sivun rakenne.

```
<html>
  <head>
    <title>Otsikko on tässä</title>
  </head>
  <body>
    <p>Tämä on kappale tekstiä.</p>
```

```
  </body>
</html>
```

Esimerkkikoodi 3. HTML-sivun esimerkki

Elementit erotetaan toisistaan aloite- ja lopetustunnisteella. Tunnisteet ovat muotoa <elementti> ”itse elementti” </elementti>, jossa vinoviivalla varustettu tunniste on lopetustunniste. Esimerkissä elementti *title* kertoo sivun otsikon ja elementti <p> kertoo kappaleesta. Elementit voivat myös sisältää erilaisia ominaisuuksia, jotka vaikuttavat mm. elementin sijaintiin, kirjasintyyppiin, väriin ja toimintaan painettaessa. [10.]

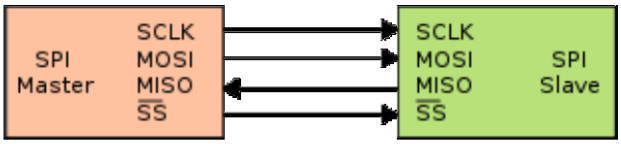
3 Sarjaväylät

3.1.1 SPI-väylä

SPI eli *Serial Peripheral Interface Bus* on synkroninen kaksisuuntainen sarjaväylä. Väylän laitteet noudattavat isäntä/orja (*master/slave*) topologiaa, jossa isäntälaite aloittaa viestinnän. Isäntälaite valitsee käytettävän orjalaitteen käyttäen SS eli *Slave Select* -signaalia. Tässä työssä käytettiin vain yhtä orjalaitetta, joten SS-signaali voitiin kytkeä pysyvästi aktiiviseksi. Muita SPI:n signaaliteitä ovat

- SCLK (*Serial Clock*); isäntälaitteen luoma kellosignaali tiedonsiirron synkronisoimiseksi
- MOSI (*Master Out, Slave In*); tieto isännältä orjalle
- MISO (*Master In, Slave Out*); tieto orjalta isännälle.

Seuraavassa kuvassa 2 esitellään signaalitiet ja niiden kulkusuunnat.

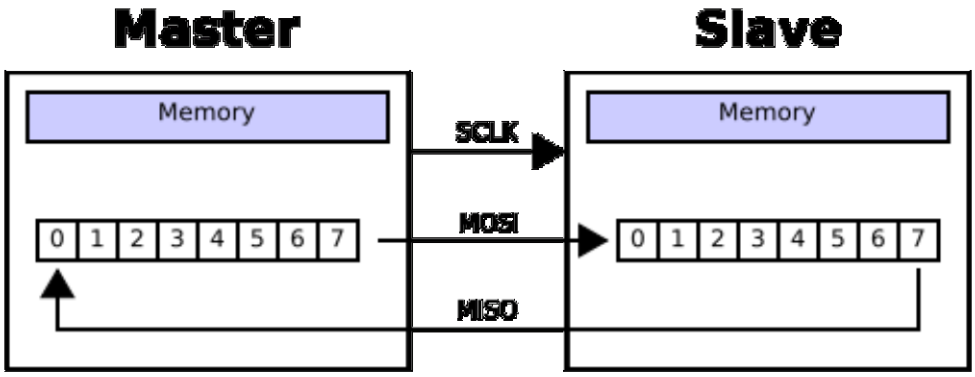


Kuva 2. SPI-väylän signaalitiet [11]

Tiedonsiirto alkaa isäntälaitteen lähettäessä loogisen nollan orjalaitteelle. Tämän jälkeen tapahtuu kaksisuuntainen tiedonsiirto, jossa isäntä lähettää bitin MOSI-

signaalitietä orjalaitteelle ja samanaikaisesti orjalaite lähettää bitin MISO-signaalitietä isäntälaitteelle. [11.]

Tieto siirtyy tavallisesti kahden siirtorekisterin välillä laitteelta toiselle ja aina kaksisuuntaisesti, vaikka toiseen suuntaan siirtyvällä tiedolla ei olisi sovelluksen kannalta mitään merkitystä. Siirto pysähtyy, kun isäntälaite pysäyttää kellon. Isäntä- ja orjalaitteen siirtorekisterin koon ja siirtokellotaajuuden täytyy olla samat. Siirtotapahtuma havainnollistetaan seuraavassa kuvassa 3.



Kuva 3. SPI-väylän tiedonsiirtotapahtuma [11]

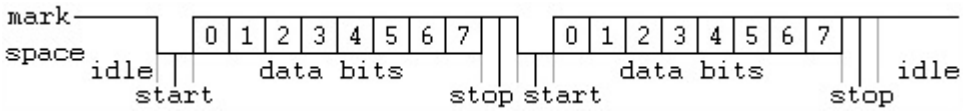
Mbedin siirtorekisterin kokoa voidaan säätää vapaasti kuten myös SPI:n kellotaajuutta. SPI-väylä ei kuitenkaan toiminut, kun kellotaajuus laskettiin alle 10 kHz:iin. SPI-väylää käytettiin sbRIO:n ja mbedin väliseen kommunikointiin. [12]

3.1.2 UART-väylä

UART (*Universal Asynchronous Receiver Transmitter*) on asynkroninen sarjaväylä, joka lähettää rinnakkaismuotoista tietoa sarjamuotoisena ja kokoaa taas lähetetyn tiedon rinnakkaismuotoiseksi paketiksi. Erilaisia UART-malleja on monenlaisia, niiden eroina on tavallisesti nopeus, puskurin koko ja mahdolliset pariteetti bitit. UART-ohjainta käytetään usein jonkin viestintä standardin mukaisen tietoliikenneväylän kanssa.

UART:n tiedonsiirtotapahtuma alkaa lähettäjän lähettäessä vastaanottajalle aloitusbitin. Tämän bitin tarkoitus on kertoa, että tiedonsiirto on alkamassa ja synkronoida vastaanottajan kello samaan tahtiin lähettäjän kellon kanssa. Tämän jälkeen siirretään haluttu tieto alkaen vähiten merkitsevästä bitistä. Vastaanottaja tarkastelee lähetetyn bitin ar-

von lähetyksen alussa synkronoidun kellon mukaan. On tärkeää, etteivät kellot poikkea toisistaan, tai tiedon siirto epäonnistuu. Kun tietobitit on lähetetty, lähetetään tarvittaessa pariteettibitit. Lopuksi lopetusbitti joka lopettaa tiedonsiirron. Siirtotapahtuma havainnollistetaan bittitasolla kuvassa 4. (ks. seur. s.). [13.]



Kuva 4. UART-tiedonsiirtotapahtuma [13]

Työssä UART:a käytettiin mbedin ja Wiflyn välillä lyhyen matkan tiedonsiirtoon, joten varsinaista tietoliikenneväylää ei tarvittu. Lisäksi mbed ja Wiflyn loogiset signaalitasot olivat samat, eikä niitä ollut tarvetta muuttaa. Mbedin ja Wiflyn UART käyttää kahta yksisuuntaista signaalitietä, tx ja rx lähetykseen ja vastaanottamiseen. [14.]

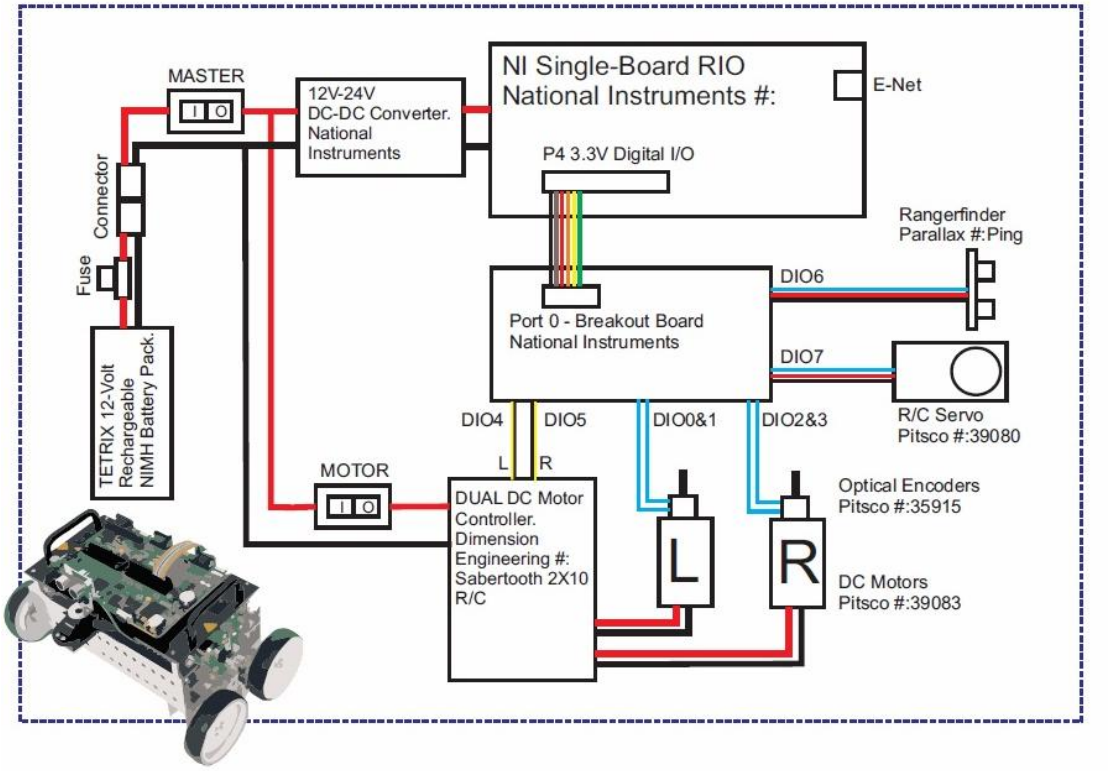
Mbedin UART-ohjaimen toimintaa voidaan säätää koodipohjaisesti. joten se toimii monien UART-mallien kanssa. Oletusarvoisena mbedin UART-ohjain toimii 9600/8N1 -asetusten mukaisesti, toisin sanoen

- 9 600 bittiä sekunnissa
- 8 tietobittiä
- ei pariteettibittiä
- 1 lopetusbitti.

4 HTTP-palvelimen fyysinen toteutus

4.1 National Instrumentsin robotti

Työssä käytettiin National Instrumentsi tarjoamaa robottirakennussarjaa. Se sisälsi Single-Board RIO 9631 - piirikortin, ultraäänisensorin ja robotin liikkumiseen tarvittavat moottorit, akut ja niiden ohjaimet. Kuvassa 5. (ks. seur. s.) esitellään robotin sisäinen rakenne. [15.]



Kuva 5. LabVIEW Robotics -rakennussarjan lohkokaavio [15]

Single-Board RIO 9631 -piirikortti on monipuolinen kehitystyökalu sulautettujen piirien ja prototyyppien valmistukseen. Kortilla on miljoonaporttinen FPGA-piiri, 266 MHz:n prosessori, 64 MB DRAM-muistia ja 128 MB flash-muistia. Lisäksi kortti sisältää 110:n 3,3 V:n digitaalista I/O-nastaa, 32 A/D-tuloa ja neljä D/A-lähtöä. Työssä käytettiin FPGA-piiriä robotin ohjaukseen, digitaalisia I/O-nastoja SPI-väylänä ja erityistä virta I/O -nastaa 5 V:n jännitelähteenä. [16.]

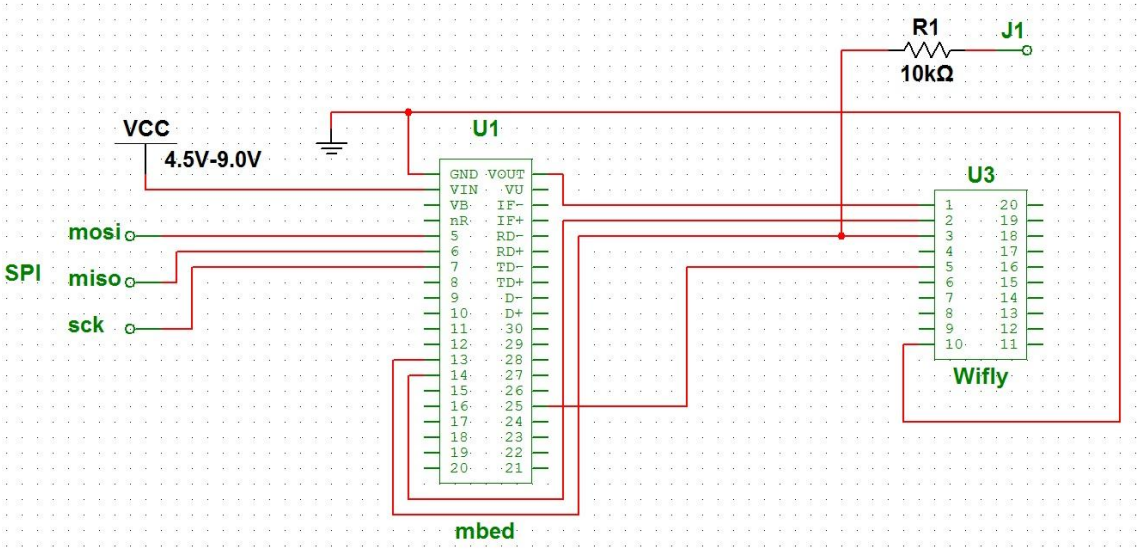
Robottiin kiinnitettiin mbed-mikrokontrolleri ajamaan http-palvelinta ja Wifly WLAN -moduuli huolehtimaan langattomasta yhteydenpidosta. Näin ohjattiin robotin toimintaa langattoman verkon kautta.

4.2 Piirilevy robottiin

Mbed-mikrokontrollerin ja Wifly-moduulin robottiin kiinnitystä varten tehtiin Single-Board RIO 9631 -piirikortin I/O-liittimiin sopiva piirilevy. Kortin teki Aleksi Orava vuonna 2013

osana omaa insinöörityötään käyttämällä EAGLE (Easily Applicable Graphical Layout Editor) - ohjelmistoa. [17.]

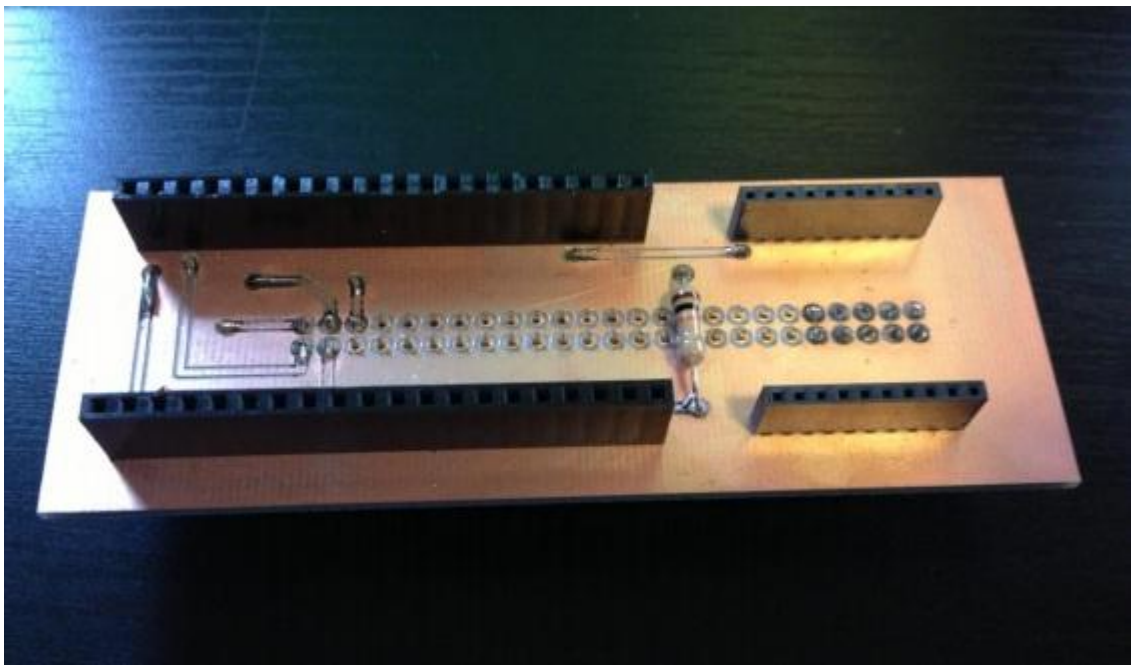
Levy tehtiin seuraavassa kuvassa 6. esiteltävän kytkentäkaavion pohjalta. Tarvittavat kytkennät olivat triviaaleja, ainoana poikkeuksena voidaan todeta Wiflyn ja Mbedin UART-väylään vaadittu ylösvetovastus. Tämän vastuksen puuttuminen aiheutti työn alkuvaiheessa ongelmia, koska kyseistä vastusta ei ollut koekytkentä levylle sopivalla reguloidulla Wiflyn jaloitukselle sopivalla aluspiirilevyllä. Vastus täytyi lisätä jotta UART yhteys saatiin toimimaan. Sen puuttuminen aiheutti ilmeisesti kohinaa UART-väylään. Reguloidussa aluslevyssä oli kyseisessä väylässä diodi ennen Wiflyn sisäistä ylösvetovastusta, joka ilmeisesti häiritsi sisäisen ylösvetovastuksen toimintaa.



Kuva 6. Piirilevyn kytkentäkaavio

Kuvasta 6 voitiin havaita, että portit 13, 14 ja 25 ovat UART-väylän TX, RX ja *reset*. Wiflyyn kytketty Vout antaa reguloidun 3,3 V käyttöjännitteen Wiflylle. SPI-väylän signaalitiet tulevat sbRIO:n I/O -porteilta.

Käyttöjännite Vcc saatiin suoraan sbRIOn käyttöjänniteeksi suunnitellusta I/O-nastasta. Tällainen I/O-nasta pystyy antamaan maksimissaan kahden ampeerin virran käytettävään ulkoiseen järjestelmään. Tämä riitti hyvin mbedille ja Wiflylle. Lisäksi National Instruments suosittaa laittamaan tarvittaessa kondensaattorin suodattamaan mahdollista epätasaisuutta virranotossa. Suodinta ei kuitenkaan tarvittu, koska se on sisäänrakennettu mbediin. Kuvassa 7. (ks. seur. s.) esitellään piirilevyn prototyyppi. [18, s. 25.]



Kuva 7. Piirilevyn valmis prototyyppi [17]

4.3 Mbed NXP LPC1768 -mikrokontrolleri

Työssä käytettiin NXP:n valmistamaa ARM Cortex™-M3 -mikroprosessorin ympärille rakennettua mbed-kehitysalustaa. Mbed valittiin, koska siinä on tarvittava määrä sisäistä muistia, riittävästi kommunikaatioväyliä ja hyvät valmiit kirjastot, jotka vähentävät ohjelmoinnin tarvetta. Kuvassa 8. (ks. seur. s.) esitellään osa mbedin ominaisuuksista. [19.]

ARM Cortex-M3 core	• 100 MHz operation • Nested Vectored Interrupt Controller for fast deterministic interrupts • Wakeup Interrupt Controller allows automatic wake from any priority interrupt • Memory Protection Unit • Four reduced-power modes: sleep, deep sleep, power-down and deep power-down
Memories	• 512 KB of Flash memory • 64 KB of SRAM
Serial peripherals	• 10/100 Ethernet MAC • USB 2.0 full-speed device/Host/ OTG controller with on-chip PHY • Four UARTs with fractional baud rate generation, RS-48, modem control, and IrDA • Two CAN 2.0B controllers • Three SSP/SPI controllers • Three I²C-bus interfaces with one supporting Fast Mode Plus (1-Mbit/s data rates) • I²S interface for digital audio
Analog peripherals	• 12-bit ADC with eight channels • 10-bit DAC
Other peripherals	• Ultra-low-power (< 1 uA) RTC • General-purpose DMA controller with eight channels • Up to 70 GPIO • Motor control PWM and Quadrature Encoder Interface to support three-phase motors • Four 32-bit general-purpose timers/counters
Package	• 100-pin LQFP (14 x 14 x 1.4 mm)

Kuva 8. Mbed-kehitysalustan ominaisuuksia [19]

Mbediä ohjelmoidaan ja käännetään selainpohjaisella käyttöliittymällä internetissä osoitteessa www.mbed.org. Ohjelmointikielenä toimii C++-kieli. Ohjelmointikäyttöliittymä toimii samalla pilvipalveluna, joten ohjelmoida voidaan millä tahansa alustalla, joka omaa tarpeeksi kehittyneen selaimen ja internet-yhteyden.

Yksi suuri syy mbedin valintaan oli sen valmis HTTP-serverikirjasto. Sitä ei kuitenkaan valitettavasti voitu käyttää, sillä se tuki vain vanhentunutta ethernet-kirjastoa. Wiflyn kirjasto oli suunniteltu tukemaan vain uusimpaa versiota mbedin ethernet -kirjastosta, joten se ei toiminut vanhemman kirjaston kanssa. [20.]

4.4 Wifly WLAN -moduuli

Wifly on Roving Networksin kehittämä sertifioitu ratkaisu IEEE 802.15.4 -standardin siirtoon IEEE 802.11 b/g -standardin langattomaan verkkoon. Se sisältää UART-väylää käyttävän langattoman verkonlisäksi muutamia I/O- ja ADC-nastoja, joita ei tässä työssä käytetty. [21.]

Wiflyä käytettiin koska mbed-mikrokontrollerilla oli sille valmis koodikirjasto, joka helpotti Wiflyn käyttöönottoa. Kirjastolla oli lisäksi sama ohjelmointirajapinta kuin mbedin ethernet -kirjastolla, joten serveriä voidaan käyttää tarvittaessa pienin muutoksin myös käyttäen ethernet-protokollaa.

Moduulin sai käyttövoimansa mbedin reguloidusta 3.3 voltin ulostulosta. Mbedin ja moduulin välinen tiedonsiirto tehtiin käyttäen UART-väylää. Väylällä pystyttiin halutessa myös muokkaamaan Wiflyn toimintaa käyttäen ns. *command modea*. Wifly vaati toimiakseen halutulla tavalla muutamien toimintaparametrien muuton. Nämä parametrit olivat

- COMM REMOTE, joka muutettiin oletusasetuksesta HELLO nollaksi. Tämä asetus vaikuttaa siihen mitä Wifly lähettää asiakkaalle avattaessa uusi yhteys. Oletusarvolla Wifly lähettää merkkijonon HELLO. Asetuksella 0 se ei lähetä mitään.
- COMM IDLE, joka määrittää ajan jonka jälkeen katkaistaan yhteys, jos TCP-väylä on toimettomana. (Ks. 4.3.)

Lisäksi Wiflyn kirjastossa olevat funktiot muuttavat näitä parametreja niihin syötettyjen tietojen perusteella. Kirjasto muuttaa tietoja kuten tukiaseman IP-osoite, salasana, käytettävä salausprotokolla ja valintaa DHCP:n käytöstä. [22.]

5 HTTP-palvelin

5.1 HTTP-palvelimen ohjelmointi

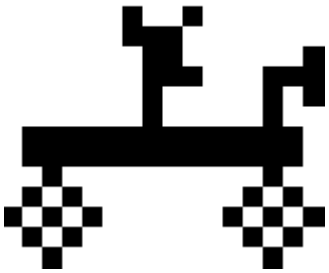
Ohjelmointi tehtiin käyttäen mbedin internet sivua C++-kielellä. Tarkoituksena oli aluksi tehdä yksinkertainen toimiva http-palvelin joka kykenee lähettämään käskyjä SPI-väylään. Se osoittautui hankalammaksi kuin oletettiin koska Wiflyn kirjasto ei tukenut valmista HTTP-palvelin kirjastoa vaan palvelin jouduttiin ohjelmoimaan itse. Palvelimen koodi esitellään liitteessä 1.

Ohjelmoitaessa käytettiin Wiflyn valmista kirjastoa nimeltä *WiflyInterface.h*. Se sisälsi tarvittava kirjastot Wiflyn ohjaukseen UART-väylällä ja TCP/IP-protokollan ohjelmointirajapinnan. Lisäksi käytettiin mbedin yleistä kirjastoa *mbed.h*:ta, joka sisälsi mm. SPI:n ohjaukseen tarvittavat funktiot.

Käytetty HTML-sivu varastoitiin mbedin sisäiselle muistille. Valitettavasti tämä aiheutti pieniä hankaluuksia sivun toimintaan koska mbed ei pysty käyttämään fseek funktiota sisäiseen flash muistiinsa, kun muisti avataan kirjoitusmoodissa. Tämä vaikeuttaa HTML-sivun muuttamista robotin tilan mukaan. Sisäiselle muistille kirjoitetaan ohjelmassa myös oma tiedosto HTTP-vastauksen otsikko-osasta, joka lähetetään vastauk-

sena HTTP GET -metodiin. Kun otsikko-osa luodaan, ohjelmassa luetaan samalla HTML-tiedoston koko. Näin ollen html-tiedostoa voidaan muuttaa ilman, että otsikko-osaan tarvitsee päivittää html-tiedoston uusi koko.

Lisäksi sisäisellä muistilla on myös favi.ico-tiedosto. Se on internetsivulle määriteltävä ikoni, joka nähdään tavallisesti sekä osoitepalkissa ennen internetsivun nimeä että talletettaessa sivusto kirjanmerkkeihin. Ikoni esitellään kuvassa 9.



Kuva 9. favi.ico suunniteltiin vastaamaan robotin ulkoista olemusta

Ohjelman käynnistyessä alustetaan SPI-väylä ja Wifly-moduuli. Seuraavaksi yritetään muodostaa WLAN yhteys ohjelmassa määriteltyyn verkkoon. Tätä tehdään niin kauan kunnes yhteys on muodostettu. IP-osoitteen Wifly hakee asetusten mukaisesti DHCP-serveriltä. Wifly myös tulostaa saadun osoitteen usb-väylään, jotta osoite olisi käyttäjän tiedossa.

Tämän jälkeen html-sivu, GET-pyynnön vastauksen otsikko-osa ja favi-iconi luetaan mbedin RAM-muistiin palvelimen toiminnan nopeuttamiseksi. Jos tiedostot luettaisiin aina käytön yhteydessä uudestaan muistiin, se hidastaisi serverin vastausnopeutta noin 100 - 300 millisekuntia. Tiedostojen muistiin luvun jälkeen ohjelma alkaa kuunnella porttia 80.

Kun porttiin 80 otetaan yhteys, talletetaan vastaanotettu tieto muistiin. Tiedosta tarkastellaan, pyydetäänkö favi.iconia, onko kyseessä get-pyyntö vai onko tieto jotain muuta. Jos kyseessä on favi.ico-pyyntö, lähetetään ikoni ja suljetaan yhteys. Jos taas vastaan otetaan GET-pyyntö, luetaan pyynnöllä haluttu osoite. Robottia ohjataan halutun osoitteen mukaan. Esimerkiksi jos haluttu osoite on /eteen, lähetetään SPI:llä robotille komento eteen ja pyynnön lähettäjälle html-sivu. Jos taas halutaan vain juuri sivu (eli http://”serverin IP-osoite”), lähetetään pyynnön lähettäjälle HTML-sivu eikä robotin toimintaan puututa. Pyynnön ollessa jotain muuta kuin GET- tai favi.ico-pyyntö vastataan

lähettäjälle virheilmoituksella *400 Bad Request*. Virheen korjausta varten, SPI käskyn lähetyksen jälkeen sytytetään tai sammutetaan ledivalo, jotta nähtäisiin että käsky on lähetetty.

5.2 HTML-sivu

HTML-sivu pidettiin hyvin yksinkertaisena, ennen kuin järjestelmä saatiin muuten toimimaan. Kuten yllä näemme, se koostui napeista jotka hakivat sivun nimensä mukaisesta osoitteesta. Lisäksi sivulla on nappi KILL joka tosin Isaac Asimovin robotiikan perus lakien mukaan kieltäytyy toteuttamasta annettua käskyä. Kuvassa 10 (ks. seur. s.) esitellään sivun aikainen versio Google Chrome -selaimella käännettynä. Se sisältää HTML version 1.1 mukaista koodia, joka nähdään liitteessä 2.

Kuva 10. HTML-sivun aikainen versio

Seuraavana nähdään sivulla esiintyvän napin HTML-koodin. Se saa selaimen hakemaan sivun osoitteesta ”sivunosoite”/eteen. Elementin ominaisuus *input type* kertoo elementin olevan nappi. Ominaisuus *value* taas kertoo napissa lukevan tekstin. *OnClick* määrittää nappia painettaessa tapahtuvan funktion.

```
<FORM1>
<INPUT TYPE="button" value="eteen" onClick="parent.location='eteen'">
</FORM1>
```

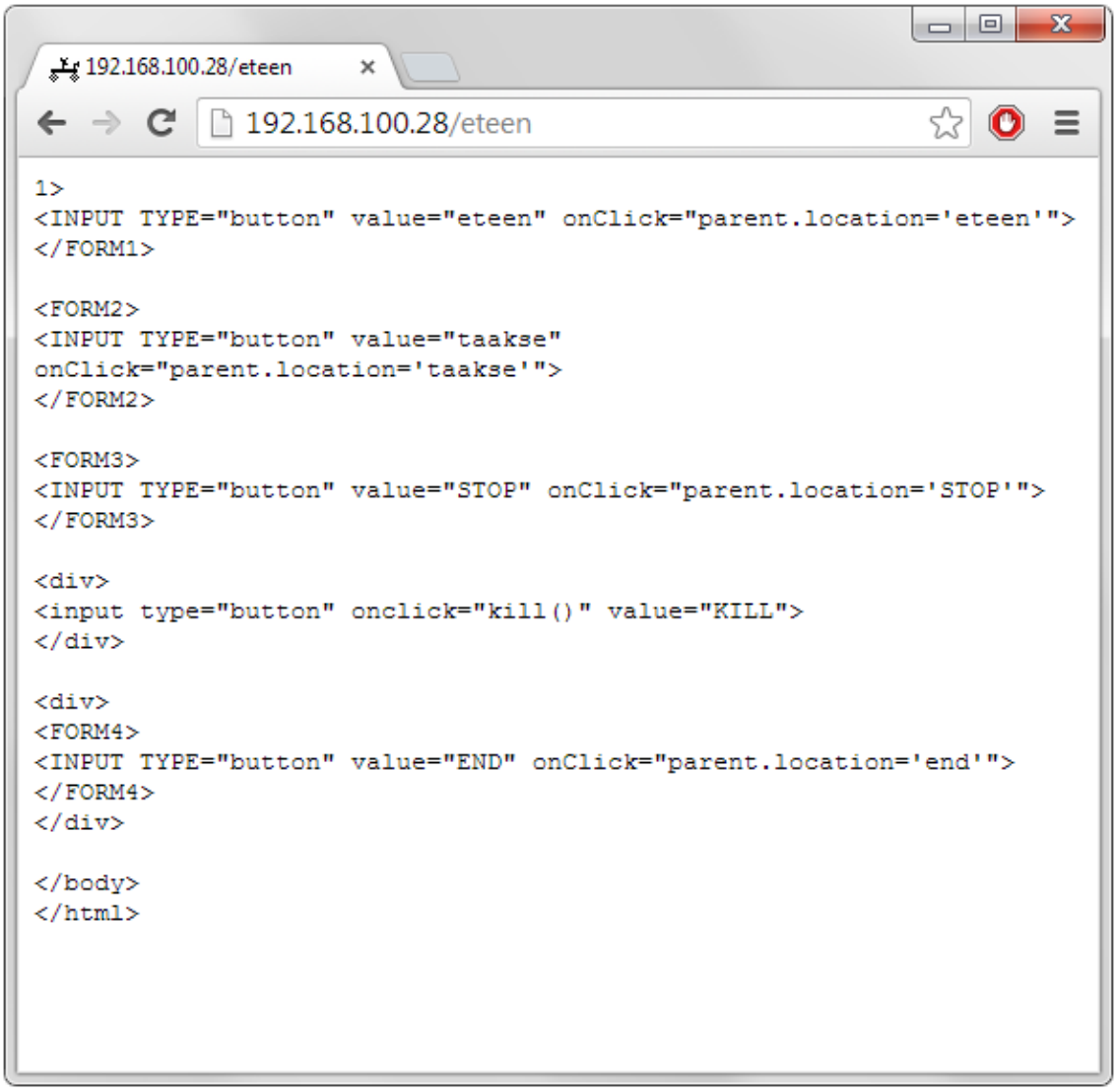
Esimerkkikoodi 4. HTML-sivussa käytetty nappi

5.3 HTTP-palvelimen toiminta

Http-palvelin saatiin toimimaan, mutta sen toiminnassa oli pieniä ongelmia. Ensinnäkin palvelin ei pysty palvelemaan useaa käyttäjää samaan aikaan. Tämä johtuu Wiflyn

puutteista, se pystyy muodostamaan vain yhden TCP-yhteyden kerrallaan. Tämän syy jäi hämärän peittoon, lisäksi mainintaa tästä ei löydetty mistään Wiflyyn liittyvistä dokumenteista. [23.]

Toisena serveri on hyvin yksinkertainen ja suhteellisen hidas. Tämä saattaa johtaa vikoihin tiedon kulussa kun pyyntöjä tulee paljon lyhyessä ajassa. Jos palvelin saa päällekkäisiä GET-pyyntöjä saattaa se lähettää sivun kerran ja ei välitä uusista pyynnöistä ennen kuin sivu on lähetetty. Näin käyttäjä luulee saavansa sivun alusta uusiksi, vaikka se saa sivun siitä kohdasta, jossa lähetys sattuu olemaan. Käyttäjä saa tällöin sivukseen vain html-koodia, jonka selain tavallisesti tulkitsee tekstiksi. Kuvassa 11 esitellään edellä mainittu vikatila. Vikaa yritettiin korjata nopeuttamalla sivun lähetystä. Aikaisessa versiossa sivu luettiin *flash*-muistista aina lähetyksen yhteydessä. Tämä toiminta tapa poistettiin ja sivu luettiin ennen palvelimen käynnistämistä RAM-muistiin. Näin nopeutettiin palvelimen toimintaa. Tällä taivoin ongelma ei kuitenkaan korjaantunut, vaikkakin sen vakavuus pieneni.



Kuva 11. Esimerkki selaimen tulkitsemasta virheellisestä sivun lähetyksestä

Kolmantena Wifly ei suostunut katkaisemaan yhteyttä katkaisukäskyn perillemenosta huolimatta. Tämä aiheutti kaksi ongelmaa. Joillain selaimilla selain ei suostunut näyttämään sivua ennen kuin yhteys oli katkaistu, vaikka GET-vastauksen otsikko-osassa oli määritelty sivun sisältävän tietynmäärän tietoa. Lisäksi, kuten edellä mainittiin, Wifly ei pystynyt käsittelemään kuin yhden TCP-yhteyden kerrallaan. Siksi yhteyden jäädessä päälle vain yhteyden muodostanut käyttäjä pystyi käyttämään palvelinta.

TCP-yhteyttä yritettiin katkaista monin eri käskyin, mutta mikään yritetyistä tavoista ei katkaissut yhteyttä. Yhteys katkesi vain idletimerin eli toimettomuusajastimeen asetetun arvon ylityttyä. Ajastin katkaisee yhteyden kun yhteys on toimettomana siihen asetetun ajan.

Ongelmaa yritettiin korjata asettamalla Wiflyn toimettomuusajastimen aika mahdollisimman pieneksi. Se johti vain uusiin ongelmiin, yhteys katkesi jo ennen kuin kaikki sivun sisältämät tiedot saatiin lähetettyä.

Palvelimen toiminnassa oli ongelmia lisäksi joidenkin puhelimien selainten kanssa. Sivu ei aluksi latautunut ollenkaan mm. Apple mallisella puhelimella. Ongelma saatiin kuitenkin korjattua lisäämällä favi.ico:n pyyntö palvelimelle. Ilmeisesti toimimattomuus johtui puhelimen selaimen tavasta pyytää ensiksi favi.ico ja vasta sitten itse HTML-sivu. Palvelin ei osannut aluksi lähettää favi.icoa, joten puhelimen selain vain odotti sitä. Varmuutta vian syyhyn ei saatu testausvälineen puuttumisen takia.

Tehty palvelin ei ole minkään HTTP-standardin mukainen. Se kuitenkin toimii yksinkertaistettuna palvelimena jossain määrin luotettavasti. Jatkokehityksenä palvelimen käyttäjäystävällisyyttä voitaisiin parantaa.

6 Yhteenveto

Tässä insinöörityössä tehtiin langaton HTTP-palvelin ohjaamaan National Instrumentsin robottia. Palvelin tehtiin käyttämällä mbed-mikrokontrolleria ja Wifly WLAN -moduulia. Palvelin saatiin toimivaksi, mutta sen luotettavuus ei ollut paras mahdollinen. Valitettavasti robotin SPI-väylää ei saatu toimimaan ajoissa, joten ohjauksen testausta ei voitu heti toteutettaa. Toinen työtyhmä sai kuitenkin SPI-väylän toimimaan myöhemmin ja robottia pystyttiin ohjaamaan palvelimen kautta. Palvelin tehtiin myös hyvin yksinkertaiseksi, jotta ohjausjärjestelmään liittäminen ei olisi tuottanut ongelmia. Koska sitä ei saatu liitettyä ohjaukseen, se myös jäi yksinkertaiseksi.

Myös palvelimen käyttäjäystävällisyydessä on tilaa jatkokehitykselle. Työssä tehdyssä versiossa käyttäjän on muutettava koodiin käsin WLAN-tukiaseman nimi ja salasana sekä katsottava DHCP-palvelimelta saatu IP-osoite terminaaliohjelmalla USB-väylän kautta. Nämä toiminnot voitaisiin toteuttaa esimerkiksi tietokoneella ajettavan palvelinta varten tehdyn ohjelman kautta tai lisäämällä palvelimeen näyttö ja fyysinen käyttöliittymä.

Wifly-moduuli osoittautui huonoksi tavaksi toteuttaa HTTP-palvelimen langattomuus sen rajoittuneen WLAN:n toiminnan takia. Se pystyi palvelemaan vain yhtä asiakasta kerrallaan. Vastaisuudessa mbedillä toteutetun palvelimen langattomuus voisi olla käytännöllisempää toteuttaa käyttämällä Ethernet WLAN -adapteria. Wifly soveltuu paremmin kahden laitteen väliseen kommunikointiin.

Työssä tehtyä palvelinta voidaan helposti käyttää myös muissa sovelluksissa, joissa on pienet käyttäjämäärät, tarvitaan vähäistä virrankulutusta ja pienikokoista langattomuutta. Pienellä jatkokehityksellä virrankulutusta saataisiin todennäköisesti vielä vähennettyä nykyisestä käyttämällä valmiustiloja sekä Wiflyssä että mbedissä palvelimen ollessa pois käytöstä.

Lähteet

1 Cisco Internetworking Basics. 2013. Verkkodokumentti. Luettu 26.10.2013. http://docwiki.cisco.com/wiki/Internetworking_Basics#OSI_Model_Physical_Layer

2 IEEE 802.11. 2013. Verkkodokumentti. Wikipedia. Luettu 26.10.2013. http://en.wikipedia.org/wiki/IEEE_802.11

3 CSMA/CA. 2013. Verkkodokumentti. Wikipedia. Luettu 27.10.2013. http://en.wikipedia.org/wiki/Carrier_sense_multiple_access_with_collision_avoidance

4 IEEE 802.11b. 2013. Verkkodokumentti. Wikipedia. Luettu 27.10.2013. http://en.wikipedia.org/wiki/IEEE_802.11b-1999

5 IEEE 802.11i standardi. 2004. Verkkodokumentti. Luettu 27.10.2013. http://standards.ieee.org/getieee802/download/802.11i-2004.pdf

6 IP. 2013. Verkkodokumentti. Wikipedia. Luettu 27.10.2013. http://en.wikipedia.org/wiki/Internet_Protocol

7 TCP. 2013. Verkkodokumentti. Wikipedia. Luettu 27.10.2013. http://en.wikipedia.org/wiki/Transmission_Control_Protocol

8 Mbed Socket. 2013. Verkkodokumentti Luettu 27.10.2013. http://mbed.org/handbook/Socket

9 RFC 2616 – Hypertext transfer protocol. 1999. Verkkodokumentti. Luettu 27.10.2013. http://tools.ietf.org/html/rfc2616

10 HTML. 2013. Verkkodokumentti. Wikipedia. Luettu 27.10.2013. http://en.wikipedia.org/wiki/Html

11 Spi-väylä. 2013. Verkkodokumentti. Wikipedia. Luettu 10.4.2013. http://en.wikipedia.org/wiki/Serial_Peripheral_Interface_Bus

12 Spi. 2013. Verkkodokumentti. Mbed. Luettu 10.4.2013. mbed.org/handbook/SPI

13 UART. 2013. Verkkodokumentti. Wikipedia. Luettu 14.4.2013. http://en.wikipedia.org/wiki/Universal_asynchronous_receiver/transmitter

14 Serial. 2013. Verkkodokumentti. Mbed. Luettu 14.4.2013. mbed.org/handbook/Serial

15 Robotic Kit. 2013. Verkkodokumentti. National Instruments. Luettu 5.9.2013. http://sine.ni.com/nips/cds/view/p/lang/fi/nid/208010

16 NI Single-board RIO. 2013. Verkkodokumentti. National Instruments. Luettu 5.9.2013. http://www.ni.com/pdf/products/us/cat_sbRIO_96xx.pdf

17 Orava Aleksi. 2013. National Instrumentsin robotista kännykällä ohjattava versio. Insinöörityö. Metropolia Ammattikorkeakoulu.

18 USER GUIDE NI sbRIO-961x/963x/964x and NI sbRIO-9612XT/9632XT/9642XT. 2010. Verkkodokumentti. National Instruments. Luettu 24.10.2013. http://www.ni.com/pdf/manuals/375052c.pdf

19 Mbed NXP LPC1768 prototypingboard. 2013. Verkkodokumentti. Mbed. Luettu 10.4.2013. http://www.nxp.com/documents/leaflet/LPC1768.pdf

20 Mokrani Samuel. 2013. Embedded Software Engineer, mbed. Verkkokommentti. Luettu 24.10.2013. http://mbed.org/questions/330/Has-anyone-succeeded-in-making-a-Wifly-H/

21 RN-XV datasheet. 2011. Verkkodokumentti. Roving Networks. Luettu 26.10.2013. http://www.rovingnetworks.com/files/resources/WiFly-RN-XV-DS.pdf

22 WiFly Command Reference, Advanced Features & Applications User's Guide. 2013. Verkkodokumentti. Roving Networks. Luettu 26.10.2013. http://ww1.microchip.com/downloads/en/DeviceDoc/rn-wiflycr-ug-v1.2r.pdf

23 Mokrani Samuel. 2013. Embedded Software Engineer, mbed. Verkkokommentti. Luettu 15.11.2013. https://mbed.org/questions/339/Multiple-mBeds-over-Wifly/

Liitteet

Http-palvelimen koodi

```
#include "mbed.h"
#include "WiflyInterface.h"
#include "AvailableMemory.h"

#define SERVER_PORT   80

DigitalOut ledi1(LED1);
DigitalOut ledi2(LED2);
DigitalOut ledi3(LED3);
DigitalOut ledi4(LED4);
Serial pc(USBTX, USBRX);
SPI spi(p5, p6, p7);

WiflyInterface wifly(p13, p14, p25, p26, "XXXX", "XXXX", WPA);

LocalFileSystem local("local");

TCPSocketServer server;
TCPSocketConnection client;

void vilkku(float nopeus)
{
    while(true) {
        ledi1=!ledi1;
        wait(nopeus);
    }
}


int Get_Size(char filename[127])
{
    char path[255];
    sprintf(path, "/local/%s", filename);

    FILE *pFile = NULL;
    pFile = fopen(path, "r");

    if(pFile==NULL) {
        pc.printf("Could not open LOCALFILESYSTEM\r");
        vilkku(1);
    }

    fseek( pFile, 0, SEEK_END );    // Asetetaan pointteri tiedoston loppuun

    int Size = ftell( pFile );      // Haetaan sivun koko.

    fseek(pFile, 0, SEEK_SET);

    fclose( pFile );                // Suljetaan tiedosto ja vapautetaan bufferi

    return Size;
}

void headerinit()      //Kirjotetaan head.txt tiedosto ja html.htm koko head.txt:hen.
{
    int Size = Get_Size("sivu.htm");
    FILE *he = NULL;
    he = fopen("/local/head.txt", "w");
```

```
    if(he==NULL) {
        pc.printf("Could not open LOCALFILESYSTEM\r");
        vilkku(2);
    }

    fprintf(he, "HTTP/1.0 200 OK\r\nContent-Type: text/html; charset=UTF-8\r\nContent-
Length: %i\r\nConnection: close\r\r\n\n",Size);
    fclose( he );

}

const char *header_s(char *sivu,int koko)       //Luetaan head.txt muistiin.
{
    FILE *he = fopen("/local/head.txt", "r");
    fread(sivu,koko,koko,he);
    fclose(he);

    return sivu;
}

//Luetaan parametrina annettu tiedosto muistiin.
const char *html_s(char filename[127], char *sivu,int koko)

{
    char path[255];
    sprintf(path, "/local/%s", filename);

    FILE *fp = fopen(path, "r");
    fread(sivu,koko,koko,fp);
    fclose(fp);

    return sivu;
}

int main()
{
    pc.printf("\r\nInit...\r\n");
    spi.frequency(10000);
    wifly.init();                    // Käyttää oletuksena DHCP:tä.
    wifly.sendCommand("set comm remote 0","YES");
    wifly.sendCommand("set comm idle 3","YES");
    wifly.sendCommand("set comm size 1420","YES");
    wifly.sendCommand("set comm time 0","YES");

    while (!wifly.connect());          // Liitytään verkkoon.
    pc.printf("IP Address is %s\r\n", wifly.getIPAddress());

    headerinit();
    int head_koko = Get_Size("head.txt");
    char head[head_koko];
    header_s(head,head_koko);

    int html_koko = Get_Size("sivu.htm");
    char html[html_koko];
    html_s("sivu.htm",html,html_koko);

    int favi_koko = Get_Size("favi.ico");
    char favi[favi_koko];
    html_s("favi.ico",favi,favi_koko);

    server.bind(SERVER_PORT);
    server.listen(4);

    pc.printf("\r\nWaiting for connection...\r\n");
```

```
    while (true) {
        if(server.accept(client)<0) {
            pc.printf("failed to accept connection.\r\n");
        } else {
            pc.printf("Connected to client\r\n");

            char buffer[1599] = {};

            client.receive(buffer, 1599);

            pc.printf("Available memory (exact bytes) : %d\r\n", AvailableMemory(1));

            if(strstr(buffer,"/favicon.ico")) {     //Lähetetään favicon.ico tarvittaessa.
                client.send(favi,favi_koko);
                pc.printf("favicon.ico Sent\r\n");
            } else

                if(strstr(buffer,"GET")) {


                    char gets[20]= {};
                    strncpy(gets,buffer,20);     //Tarkastellaan vain GET:n jälkeistä osaa.
                    pc.printf(gets);

                    if(strstr(gets,"/STOP")) {
                        ledi3=!ledi3;
                        spi.write(4);
                    }

                    if(strstr(gets,"/eteen")) {
                        ledi1=!ledi1;
                        spi.write(1);
                    }

                    if(strstr(gets,"/taakse")) {
                        ledi2=!ledi2;
                        spi.write(2);
                    }

                    if(strstr(gets,"/end")) {
                        wifly.disconnect();
                        vilkku(3);
                    }

                    wifly.flush();
                    client.send(head,head_koko);
                    pc.printf("header sent\r\n");

                    client.send(html,html_koko);
                    pc.printf("page sent\r\n");

                } else {
                    char gets[20] = {};
                    strncpy(gets,buffer,20);
                    pc.printf("else than GET recived:\r\n%s\r\n",gets);
                    client.send("\r\n400 Bad Request\r\n",19);
                }
        }
        client.close();                         //Tällä ei tunnu olevan vaikutusta.
        wifly.close();
        wifly.flush();
        pc.printf("\r\nClosed connection\r\n listening...\r\n");
    }
}
```

HTML-sivun koodi

```
<!DOCTYPE html>
<html>
<style>
body
{
margin-left:10%;
background-color:#E0E0E0;
}
h1
{
color:#0066FF;
}
p
{
font-family:"Times New Roman";
font-size:20px;
}
form4
{
margin-left:15%;
}
</style>
<script>
function kill()
{
alert("Command is in conflict with the First Law.\n\nCannot comply.");
}
</script>
<body>

<H1>ROBO-Mk1</H1>

<HR SIZE="1" COLOR="blue" ALIGN="left" WIDTH="80%">

<P>Robotin ohjaus.</P>

<FORM1>
<INPUT TYPE="button" value="eteen" onClick="parent.location='eteen'">
</FORM1>

<FORM2>
<INPUT TYPE="button"value="taakse" onClick="parent.location='taakse'">
</FORM2>

<FORM3>
<INPUT TYPE="button" value="STOP" onClick="parent.location='STOP'">
</FORM3>

<div>
<input type="button" onclick="kill()" value="KILL">
</div>

<div>
<FORM4>
<INPUT TYPE="button" value="END" onClick="parent.location='end'">
</FORM4>
</div>

</body>
</html>
```