Mika Tuomainen

Yleiskäyttöisen käyttöjärjestelmän sovittaminen uuteen laiteympäristöön

Insinöörityö
Kajaanin ammattikorkeakoulu
Tekniikan ja liikenteen ala
Tietotekniikan koulutusohjelma
Kevät 2007

Kajaanin ammattikorkeakoulu

OPINNÄYTETYÖ
TIIVISTELMÄ

Koulutusala Tekniikan ja liikenteen ala	Koulutusohjelma Tietotekniikka
Tekijä(t) Mika Tuomainen	
Työn nimi Yleiskäyttöisen käyttöjärjestelmän sovittaminen uuteen laiteympäristöön	
Vaihtoehtoiset ammattiopinnot Ohjelmointi	Ohjaaja(t) Arto Partanen
	Toimeksiantaja Elektrobit Wireless Communications Ltd, Kajaani
Aika Kevät 2007	Sivumäärä ja liitteet 40

Tiivistelmä

Insinöörityön tavoitteena oli sovittaa yleiskäyttöinen Zero-käyttöjärjestelmä Xilinxin ML403 -sulautettujen järjestelmien kehitysympäristöön, sekä toteuttaa tälle ympäristölle sarjaportin ja I/O-pinnien laiteajurit. Työn teoriaosuudessa käsitellään käyttöjärjestelmien tarkoitusta ja toimintaperiaatetta, sekä sulautettuja järjestelmiä käsitteenä. Teoriaosuudessa myös käydään arkkitehtuuritasolla läpi, kuinka Zero-käyttöjärjestelmän yleiskäyttöisyys on saavutettu ja mitä toimintoja joudutaan implementoimaan jokaiseen ympäristöön erikseen.

Zero-käyttöjärjestelmä on yleiskäyttöinen ja tarkoitettu nimenomaan sulautettujen järjestelmien käyttöjärjestelmäksi. Sulautettu järjestelmä on kuitenkin niin laaja käsite ja käsittää niin monia erilaisia järjestelmiä, että täydellisen yleiskäyttöisyyden saavuttaminen on ainakin erittäin vaikeaa, jos ei jopa mahdotonta. Muun muassa muistinhallintaan, ajastimiin ja keskeytyksiin liittyvät toiminnot on yleensä toteutettava jokaiseen järjestelmään erikseen. Tämän työn ensisijainen tarkoitus oli toteuttaa nämä laitteistoriippuvaiset toiminnot ML403-ympäristöön. Testausta ja jatkokehitystä varten toteutettiin myös sarjaportin ja I/O-pinnien laitteistoajurit. Laitteistoriippuvaisten ohjelmamoduulien kehityksessä käytettiin Xilinx Platform Studio -ohjelmaa. Zero-käyttöjärjestelmä on sovitettu toimimaan myös Windows-käyttöjärjestelmässä, joten järjestelmän konfiguroimiseen ja käytettyjen testipetien kehitykseen voitiin käyttää Microsoftin Visual Studio 2005 -ohjelmistonkehitysympäristöä. Järjestelmän testaukseen käytettiin HyperTerminal- ja ZeroTracer-tietoliikenneohjelmia.

Työn toteutuksessa havaittiin, kuinka suuri merkitys hyvällä suunnittelulla on ohjelmistonkehityksessä. Zero-käyttöjärjestelmä on erityisesti hyvän suunnittelutyön tuloksena saatu helposti järjestelmästä toiseen siirrettäväksi. Zero-käyttöjärjestelmän suunnittelussa on myös painotettu ohjelmakomponenttien uudelleenkäytettävyyttä, mikä nopeuttaa myös järjestelmän siirtämistä uuteen laiteympäristöön.

Kieli	Suomi
Asiasanat	käyttöjärjestelmä, yleiskäytöisyys, sulautettu järjestelmä
Säilytyspaikka	☒ Kajaanin ammattikorkeakoulun Kaktus-tietokanta ☒ Kajaanin ammattikorkeakoulun kirjasto

Kajaanin ammattikorkeakoulu
University of Applied Sciences

THESIS
ABSTRACT

School	Degree Programme
School of Engineering	Information Technology
Author(s) Mika Tuomainen	
Title Adapting a General-Purpose Operating System to a New Device Environment	
Optional Professional Studies Programming	Instructor(s) Arto Partanen
	Commissioned by Elektrobit Wireless Communications Ltd, Kajaani
Date 8 April 2007	Total Number of Pages and Appendices 40

The purpose of this Bachelor's thesis was to adapt the general-purpose operating system called Zero to the Xilinx ML403 Evaluation Platform. The device drivers for the serial port and the I/O pins were also developed for this platform. The theoretical part of the thesis includes information about the meaning and basic operational principles of operating systems. It also discusses embedded systems in general and the ML403 Evaluation Platform individually. It is also explained, in an architectural level, how the Zero operating system works.

The core of the Zero operating system is mainly hardware independent but the device drivers and some features need to be done separately to every system. Some configuring is also required when the Zero operating system is being adapted to a different environment. The device dependent software components were developed by using the Xilinx Platform Studio software development environment and the modules were tested by using the HyperTerminal and the ZeroTracer communication software.

The result of the project was that the Elektrobit Company got an operating system which works on the Xilinx ML403 Evaluation Platform and is able to send and receive data via a serial communication port. The system I/O pins can also be configured and controlled by the operating system.

Language of Thesis	Finnish
Keywords	operating system, general-purpose, embedded system
Deposited at	☒ Kaktus Database at Kajaani University of Applied Sciences ☒ Library of Kajaani University of Applied Sciences

ALKUSANAT

Yleiskäyttöisen käyttöjärjestelmän sovittaminen uuteen laiteympäristöön oli erittäin mielenkiintoinen ja haastava projekti toteutettavaksi. Haluan kiittää Elektrobittiä, joka on mahdollistanut tämän työn tekemisen. Erityisesti haluan kiittää Kajaanin yksikön MTS Engineering Teamin johtajaa Joona Tolosta, joka on henkilökohtaisella panoksellaan vienyt tätä projektia eteenpäin ja täten tehnyt työn tekemisen mahdolliseksi. Haluan myös kiittää insinöörityöni ohjaajana yrityksen puolesta toiminutta Kalle Ahokasta arvokkaista neuvoista, joita olen saanut työtä tehdessäni. Insinöörityön ohjaajana Kajaanin ammattikorkeakoulun puolesta toimi Arto Partanen, jota haluan myös kiittää saamastani ohjauksesta ja materiaalin etsintään liittyvistä vinkeistä.

Kajaanissa keväällä 2007

Mika Tuomainen

SISÄLLYS

TERMILUETTELO

ASIC	Application-Specific Integrated Circuit, asiakaskohtainen järjestelmäpiiri
FIFO	First In, First Out, ensimmäisenä tullutta palvellaan ensimmäisenä
FPGA	Field Programmable Gate Array, ohjelmoitava logiikkaverkko
GPIO	General Purpose Input/Output, yleiskäyttöinen portti
HDL	Hardware Description Language, laitteistonkuvauskieli
I/O	Input/Output, luku/kirjoitus
INTC	Interrupt Controller, keskeytyspiiri
OOC	Object Oriented C, olio tyyppinen ohjelmointi C-kielellä
Pollaus	Jatkuva tilan kysely
RAM	Random Access Memory, käyttömuisti
ROM	Read Only Memory, lukumuisti
Skedulointi	Suoritettavan prosessin vaihto
SoC	System-on-a-Chip, järjestelmäpiiri
UART	Universal Asynchronous Receiver Transmitter, sarjaliikennepiiri

1 JOHDANTO

Sulautetut järjestelmät ovat kasvaneet paljon viime vuosina ja tulleet entistä monipuolisemmiksi. Perinteisesti sulautettujen järjestelmien ohjelmat eivät ole olleet yleiskäyttöisiä vaan ne on räätälöity jotain tiettyä tarkoitusta ja järjestelmää varten. Sulautettujen järjestelmien suorituskyky on kuitenkin kasvanut merkittävästi viime vuosina ja niihin voidaan kehittää entistä monimutkaisempia ohjelmia, joita voidaan käyttää mitä moninaisimpiin tehtäviin. Ohjelmistojen kehityksessä pyritäänkin siihen, ettei samaa ohjelmistoa tarvitsisi toteuttaa jokaiselle laitteistolle erikseen. Tähän tarkoitukseen tarvitaankin käyttöjärjestelmää, joka toimii linkkinä laitteiston ja sovellusten välissä. Luonnollisesti myös käyttöjärjestelmän tulisi olla mahdollisimman helposti siirrettävissä laitteistosta toiseen.

Elektrobit-konsernin Kajaanin toimipisteessä on tehty jonkin aikaa tällaisen käyttöjärjestelmän kehitystyötä. Zero-käyttöjärjestelmä on yrityksessä kehitetty yleiskäyttöinen ja joustava käyttöjärjestelmä sulautettuihin järjestelmiin. Se on kehitetty nimenomaan työntekijöiden oman osaamisen ja luovuuden kehittämiseen. Sen joustavuus perustuu omiksi lohkoikseen eritytettyihin ohjelmakokonaisuuksiin ja sen keskeiset toiminnot on keskitetty Zero Controlleriin, jonka ympärille voidaan kehittää mitä moninaisimpia sulautettuja ohjelmistoja. Yleiskäyttöisyys perustuu käyttöjärjestelmästä eritytettyihin laitteistoajureihin, jotka toteutetaan jokaiselle laitteistolle erikseen ilman, että rajapinta käyttöjärjestelmän ja laitteistoajurin välillä muuttuu.

Tällainen suuri oppimisprojekti, joka koostuu rajattomasta määrästä osaprojekteja, on syntynyt siitä tarpeesta, että projektien väliin jäävälle ajalle olisi jotain mielekästä tekemistä. Aikaisemmin vanhan projektin loppumisen ja uuden projektin alkamisen väliin jäävälle ajalle ei ole ollut mitään hyödyllistä tekemistä. Toinen syy on päättötöiden aiheen hankala löytyminen luottamuksellisten asiakasprojektien yhteydestä. Samalla uudelle työntekijälle tulee tutuksi yrityksen prosessi- ja dokumentointikäytännöt. Tämä on myös erittäin hyvä tilaisuus antaa vähän kokeneemmalle työntekijälle projektinvetovastuuta ja opettaa tällä tavalla projektipäällikölle kuuluvia tehtäviä turvallisessa ympäristössä.

Aiheen insinöörityölle sain työskennellessäni yrityksessä harjoittelijana. Insinöörityön tarkoituksena on sovittaa Zero-käyttöjärjestelmä Xilinxin valmistamaan ML403 sulautettujen järjestelmien kehitysympäristöön. ML403-ympäristö perustuu Virtex-4 FPGA -piiriin, jolle on

integroitu PowerPC-prosessori. Samaan ympäristöön tullaan myöhemmissä projekteissa toteuttamaan useita eri toimintoja, kuten USB Host Controller- ja VGA-näytönohjainlaiteajurit. Tästä johtuen käyttöjärjestelmään päätettiin jo sovitusvaiheessa toteuttaa laitteistoajurit, joiden kautta sovelluskehittäjät voivat saada informaatiota ohjelman toiminnasta. Käyttöjärjestelmän oletustulostusväylä on sarjaportti, joten oli luonnollista toteuttaa sitä ohjaava laitteistoajuri. Tämän lisäksi haluttiin myös laiteajuri, joka mahdollistaa piirilevyllä olevien LEDien ja nappien ohjauksen.

2 KÄYTTÖJÄRJESTELMÄN TARKOITUS JA TOIMINTAPERIAATE

Useimmille ihmisille sana käyttöjärjestelmä tuo mieleen lähinnä jonkin komentokehotteen tai Windowsin ikkunoidun käyttöliittymän. Tämä on luonnollista, koska käyttöliittymä on yleensä käyttäjälle kaikkein silmäänpistävin osa käyttöjärjestelmästä. Kuitenkin se on vain pieni osa käyttöjärjestelmää. Käyttöjärjestelmän päätehtävä on toimia liitäntänä sovellusohjelmien ja laitteiston välillä. Tämä rajapinta antaa käyttöjärjestelmälle sen ominaiset piirteet, joita kuorrutus eli käyttöliittymä käyttää. [1.]

Käyttöjärjestelmä on siis ohjelmisto, joka tarjoaa sovelluskehittäjälle rajapinnan laitteistoon. Käyttöjärjestelmän tarjoamien palvelujen avulla sovellukset voivat kommunikoida järjestelmän kanssa ja muun muassa käynnistää ohjelman suorituksen. Suorituksen aikana käyttöjärjestelmä tarjoaa sovellukselle oheislaitteiden ohjaukseen, muistinhallintaan ja resurssien hallintaan liittyviä palveluita. Suoritettaessa samanaikaisesti useita sovelluksia käyttöjärjestelmä huolehtii laitteistoresurssien jakamisesta ohjelmille. [1.]

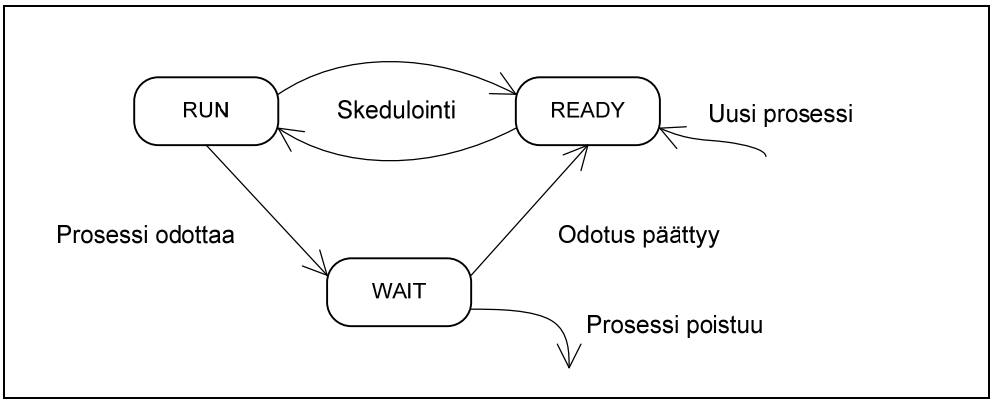
Erään määritelmän mukaan käyttöjärjestelmä on joukko integroituja systeemikutsuja, joiden päätehtävänä on tietokoneen resurssien hallinta. Tällaisia resursseja ovat mm. I/O:n hallinta ja virheistä toipuminen, prosessorin hallinta, muistinhallinta, prosessien skedulointi ja suojaukset. [2.]

Monissa yhteyksissä joukkoa käyttöjärjestelmän keskeisimpiä toimintoja kutsutaan käyttöjärjestelmän ytimeksi eli kerneliksi. Yleispätevää käyttöjärjestelmän ytimen määritelmää ei voida antaa, vaan se riippuu olennaisesti tarkasteltavasta järjestelmästä. Ytimen toimintoihin voidaan luetella ainakin ensitason keskeytyskäsittely, prosessien hallinta ja oheislaitteiden ohjaus. [1.] [3.]

Ensitason keskeytyksen käsittelyksi lasketaan yleensä ympäristön talletus ja oikeaan keskeytyskäsittelijään haarautuminen. Ohjelmallisten toimintojen määrä määräytyy sen mukaan, miten käytettävissä olevan laitteiston keskeytystoiminta on määritelty. Järjestelmissä, joissa laitteisto tallettaa ympäristön automaattisesti ja keskeytysvektori ohjaa kunkin laitteen keskeytyksen oikealle ytimen rutiinille, ei ohjelmoijan vastuulle jää ensitason käsittelijässä juuri mitään. Jos laitteisto tallettaa ympäristön automaattisesti keskeytyksen sattuessa, niin se myös palauttaa sen keskeytyksestä poistuttaessa. Ympäristön talletuksen ollessa ohjelmiston vastuulla, tulee sen palautus hoitaa ohjelmallisesti myös. [1.] [3.]

Käyttöjärjestelmän toiminnan ymmärtämisen kannalta tärkein asia on ymmärtää prosessikäsite, joka on tietokoneessa moniajon perusyksikkö. Yleisen määritelmän mukaan prosessikäsitteellä tarkoitetaan suorituksessa olevaa ohjelmaa. Käyttöjärjestelmän prosessien hallinnan tehtäviin kuuluu muun muassa uusien prosessien luominen, skedulointi, lopettaminen ja ajastaminen.

Käyttöjärjestelmän kannalta prosessi on jokaisena ajan hetkenä jossain ennalta määrätyssä tilassa. Tilat voivat olla esimerkiksi kuvassa 1 esitetyt RUN, READY ja WAIT. Tilassa RUN ovat juuri suoritettavat prosessit, joita voi olla maksimissaan yhtä monta kuin järjestelmässä on suorittimia. WAIT-tilassa ovat prosessit, jotka odottavat jotain tapahtumaa, esimerkiksi I/O-operaation päättymistä. READY-tilassa ovat prosessit, jotka odottavat vain suoritukseen pääsemistä, eli niillä on kaikki muut tarvittavat resurssit suoritinta lukuun ottamatta. Prosessien skeduloinnilla tarkoitetaan menetelmää, jolla valitaan seuraavaksi suoritettava prosessi READY-tilassa olevista prosesseista. Uudet prosessit ilmestyvät aina ensin READY-tilaan, ja prosessien voidaan ajatella poistuvan vaikka WAIT-tilan kautta. Käyttöjärjestelmän tehtävänä on ohjata prosessien etenemistä tilamallin edellyttämällä tavalla. [1.]



Kuva 1. Prosessin tilakaavio

Prosessien skedulointiin liittyviä algoritmeja on varmaan yhtä paljon kuin on käyttöjärjestelmiäkin, mutta ne kaikki perustuvat muutamien perusmenetelmien kombinaatioihin. FIFO-menetelmässä prosessit otetaan suoritukseen siinä järjestyksessä, jossa ne tulevat READY-tilaan. Menetelmän hyvänä puolena voidaan pitää sitä, että kaikki prosessit pääsevät yhtä usein suoritukseen ja, että menetelmä on erittäin helppo toteuttaa. Huonona puolena voidaan mainita esimerkiksi se, että ikuinen silmukka jossain prosessissa saa koko järjestelmän

pysähtymään. Hyvänä puolena mainittu kaikkien prosessien saman arvoisuus voidaan kokea myös huonona asiana, koska tärkeät prosessit eivät pääse vähäpätöisiä useammin ajoon. [1.]

Tämä asia voidaan korjata prioriteettiin perustuvalla menetelmällä, joita voidaan ajatella olevan kahdenlaisia, kiinteään ja vaihtelevaan prioriteettiin perustuvia. Kiinteässä prioriteetissa jokaisella prosessilla on ennalta määrätty prioriteetti ja READY-tilasta suoritukseen pääsee aina suurimman prioriteetin omaava prosessi. Menetelmän hyviä puolia on ennustettavuus, eli tiedetään varmasti, milloin mikäkin prosessi pääsee suoritukseen. Menetelmän haittapuolena voidaan mainita se, että korkean prioriteetin prosessit voivat varata kaiken suoritinajan, jolloin matalan prioriteetin prosessit eivät pääse suoritukseen olleenkaan. Tämä voidaan estää käyttämällä vaihtelevan prioriteetin menetelmää, jossa prosessin prioriteettia nostetaan tai lasketaan sen mukaan, kuinka paljon se on saanut suoritinaikaa. Tämä yleensä lisää myös suoritustehoa ja parantaa vasteaikaa, mutta samalla menetetään skeduloinnin ennustettavuus. [1.]

Viimeisenä perusmenetelmänä mainittakoon kiertovuorottelu, joka toimii aivan kuin FIFO, mutta jokaiselle prosessille annetaan tietty aikaviipale, jonka se saa olla suorituksessa kerrallaan. Menetelmän hyvä puoli on reiluus, koska jokainen prosessi saa olla suorituksessa yhtä kauan. Käytännössä todelliset sovellukset ovat aina näiden perusmenetelmien yhdistelmiä. Tavallinen yhdistelmä on käyttää käyttöjärjestelmän omille prosesseille kiinteän prioriteetin menetelmää ja käyttäjän säikeille vaihtuvan prioriteetin menetelmää. Tämän lisäksi saman prioriteetin omaavien prosessien kesken sovelletaan kiertovaihtelumenetelmää. [1.]

3 TOIMINTAYMPÄRISTÖ

3.1 Sulautettu järjestelmä

Sulautetuksi järjestelmäksi kutsutaan sellaista laitetta, jossa mikrotietokone on osana jotain suurempaa järjestelmää. Yleensä sulautettu järjestelmä sisältää vähintään mikrokontrollerin ja sille kirjoitetun ohjelman. Perinteisesti sulautetut järjestelmät eivät ole olleet yleiskäyttöisiä vaan ne on räätälöity jotain tiettyä tarkoitusta varten. Mikrokontrollerin lisäksi sulautettu järjestelmä tarvitsee myös muistia, joka yleensä jakaantuu kahteen osaan: ohjelmamuistiin ja käyttömuistiin. Ohjelmamuisti on yleensä ROM-muistia (engl. Read Only Memory), jolle on tyypillistä, että se säilyttää tietonsa käyttöjännitteen katkeamisen jälkeenkin. Ohjelma ladataan muistiin yleensä tuotteen valmistuksen yhteydessä, eikä käyttäjän tarvitse sitä vaihtaa. Käyttömuisti taas on yleensä RAM-muistia (engl. Random Access Memory), josta tiedot häviävät käyttöjännitteen katkeamisen jälkeen. Käyttömuistissa säilytettään yleensä tietoja, joita ohjelma tarvitsee toimiessaan. [4.]

Nykyään monet sulautetut järjestelmät sisältävät SoC-järjestelmäpiirin (engl. System on Chip), jolle on integroitu kaikki järjestelmän tärkeimmät toiminnot. Tyypillisesti järjestelmäpiiri sisältää ainakin yhden tai useamman mikrokontrollerin tai prosessorin, välimuistin, väylät sekä mahdolliset oheislaitteet ja ohjaimet. Tyypillisiä järjestelmäpiirille integroituja oheislaitteita ovat mm. verkko- ja äänikortit. Järjestelmäpiirit on yleensä toteutettu joko ASIC-asiakaspiirillä (engl. Application-Specific Integrated Circuit) tai ohjelmoitaviin logiikkaverkkoihin perustuvalla FPGA-piirillä. Tyypillisesti massatuotantoon tarkoitettavat järjestelmät toteutetaan ASIC-piireillä, koska FPGA-piirit ovat niitä hitaampia eivätkä pysty käsittelemään yhtä monimutkaisia järjestelmiä. FPGA-piirien etuihin sen sijaan voidaan laskea lyhyempi kehitysaika ja mahdollisuus ohjelmoida piiri uudelleen. [5.] [6.]

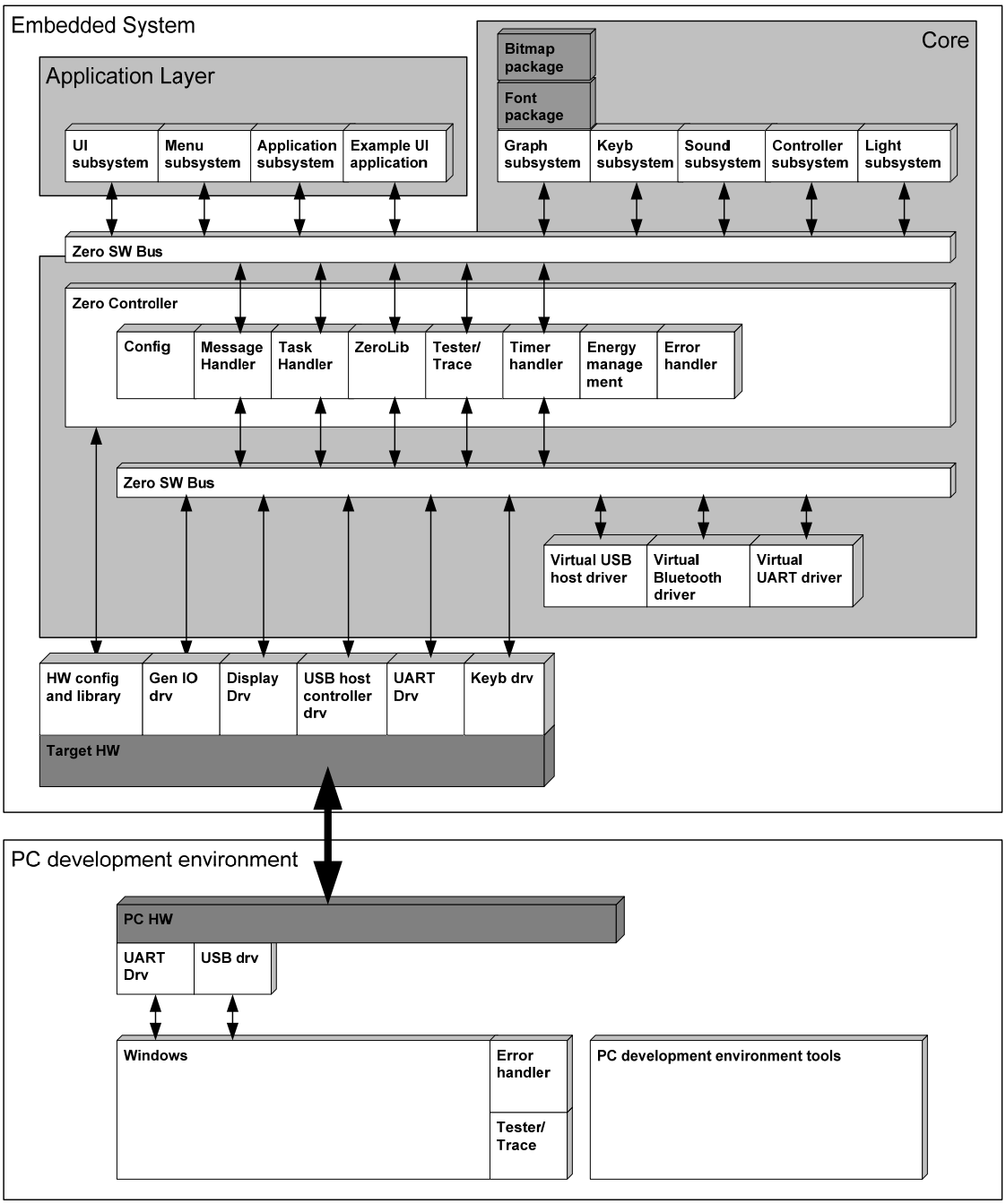
Järjestelmäpiirille voidaan toteuttaa kahdenlaisia prosessoreita, kovia (hard) ja pehmeitä (soft). Kovat prosessorit on rakennettu piirille valmiiksi tehtaalla, eikä niitä voi jälkikäteen muuttaa. Pehmeät prosessorit ovat HDL-kielellä (engl. Hardware Description Language) kirjoitettua koodia, jota voidaan kehittää yhdessä muun järjestelmän kanssa. [7.]

Sulautetun järjestelmän ohjelma on yksinkertaisimmillaan pelkkä ohjaussilmukka. Tällöin ohjelma pyörii ikuisessa silmukassa, jossa kutsutaan aliohjelmia, jotka suorittavat joitain en-

nalta määrättyjä tehtäviä. Tällainen tehtävä voi olla esimerkiksi lämpötila-anturin lukeminen ja lämpötilan tulostaminen näytölle. Hieman kehittyneempi sulautetun järjestelmän ohjelma on keskeytysohjattu. Tämä tarkoittaa sitä, että jokin järjestelmän suorittamista tehtävistä aiheuttaa prosessorille keskeytyksen, joka pitää käsitellä. Tällaisen keskeytyksen voi aiheuttaa esimerkiksi ajastin tai sarjaporttiin vastaanotettu data. Tällaisessa mallissa on yleensä myös ohjaussilmukka, mutta siinä ei suoriteta mitään tehtäviä, joiden suoritus häiriintyy yllättävästä viiveestä. Keskeytyksen käsittelevän aliohjelman tulee olla suoritusajaltaan mahdollisimman lyhyt, jotta keskeytyksiin reagointi ei viivästy liikaa. Kauan aikaa vievät keskeytyksiin liittyvät toiminnot tulisikin suorittaa ohjaussilmukassa, jota keskeytyskäsittelijä informoi, milloin käsittely tulee suorittaa. Suuremmat sulautetut järjestelmät ovat jo prosessien moniajoon perustuvia. Niissä matalan tason koodi vaihtaa suoritettavaa prosessia ajastimen mukaan. Tällöin järjestelmällä katsotaan olevan käyttöjärjestelmä. Yleensä käyttöjärjestelmä kehitetään erilleen muusta ohjelmasta tai ostetaan joltain kolmannelta osapuolelta. [8.]

3.2 Yleiskäyttöinen Zero-käyttöjärjestelmä

Zero-käyttöjärjestelmän suunnittelun lähtökohtana on ollut yleiskäyttöisyys ja joustavuus, eli se tulisi mahdollisimman helposti olla siirrettävissä eri laiteympäristöihin. Nämä ominaisuudet on toteutettu kuvan 2 mukaisella arkkitehtuurilla, jossa jokaiseen laiteympäristöön toteutetut laiteajurit on sijoitettu omaksi lohkokseen kohdeympäristön yläpuolelle. Käyttöjärjestelmän ydin (core) koostuu kontrollerista sekä laiteriippumattomista ”subsystemeistä” ja virtuaaliajureista. Kaikki informaatio laiteajureiden ja ytimen välillä kulkee Zero SW -väylän kautta.



Kuva 2. Zero-käyttöjärjestelmän ohjelmistoarkkitehtuuri [9]

Subsystemit ovat itsenäisiä ohjelmistokomponentteja, jotka tarjoavat sovelluksille laitteistoriippumattomia palveluita. Ohjelmistokehittäjät voivat valita vain tarvittavat subsystemit projektiinsa, ja ylimääräiset osat voidaan poistaa käytöstä ilman suuria ohjelmistomuutoksia. Subsystemit voivat myös rekisteröityä järjestelmään ytimen ulkopuolelta lähettämällä rekisteröintiviestin Controllerille käynnistyessään. Tämän jälkeen ne voivat toimia järjestelmässä samalla tavalla kuin ytimen sisäiset komponentit.

Laiteajurit tarjoavat kohdelaitteistoriippuvaisia palveluita. Siirrettäessä Zero-käyttöjärjestelmä uuteen laiteympäristöön, tulee ohjelmistonkehittäjän tehdä uudelle ympäristöllä omat laitteistoajurit. Tämän jälkeen käyttöjärjestelmä toimii taas ilman, että siihen on täytynyt tehdä suuria ohjelmistomuutoksia. Virtuaaliajureilla on samanlainen rajapinta, ja ne toimivat samalla tavalla kuin vastaava laitteistoajuri. Olennaisin ero on, että ne ovat laitteistoriippumattomia. Virtuaaliajureiden avulla voidaan toteuttaa kerroksellinen ajurirakenne, joka tarjoaa monia etuja esimerkiksi USB- tai Bluetooth-yhteyden toteutuksessa. Sovelluskerroksen komponentit sisältävät käyttöliittymän ja käyttöliittymäsovellukset. Ne toimivat samalla tavalla kuin subsystemit, mutta sisältävät sovelluskohtaisia toimintoja.

Zero SW Bus on väylä, jonka avulla ohjelmamoduulit kommunikoivat. Käytännössä se tarkoittaa Zero-viestirajapintaa ja OOC-funktiokutsurajapintaa, jotka ohjelmamoduulien tulee toteuttaa.

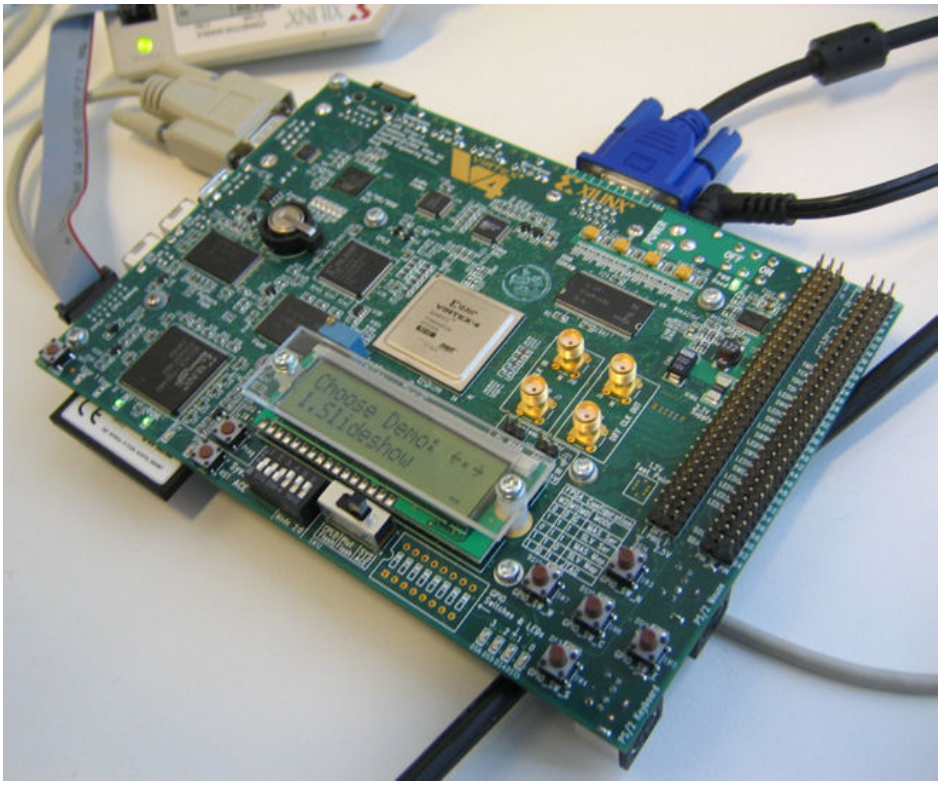
Zero-käyttöjärjestelmässä prosessien skeduloinnin hoitaa Controlleri, joka on tämän järjestelmän kerneli. Skedulointi perustuu FIFO-menetelmään, jolla saavutetaan näennäinen moniajo. Menetelmässä jokainen prosessi on samanarvoinen ja niitä skeduloidaan saapumisjärjestyksessä. Prosessin suoritusta ei voida ohjelmallisesti keskeyttää, vaan prosessi suoritetaan loppuun, jonka jälkeen valitaan seuraava prosessi ajoon. Prosessin suorituksen voi kuitenkin keskeyttää suorittimelle tuleva keskeytys, jolloin siirrytään suorittamaan keskeytyksenkäsittelijää. Controlleri vastaa myös muistinhallinnasta ja sanomienvälityksestä.

PC-kehitysympäristö tarjoaa ohjelmistonkehittäjälle työkaluja tracen eli ohjelman toiminnasta kertovan tulostuksen ja virheilmoitusten tulkintaan. Zero SW lähettää tracea ja virheilmoituksia jonkin tietoliikenneportin välityksellä, jonka kautta Windows-ohjelmisto vastaanottaa nämä viestit. Windows-ohjelmistot voivat myös lähettää rekisteröintiviestin, jolloin ne voivat toimia Zero-ympäristössä subsystemien tapaan.

PC-kehitysympäristöstä löytyy myös ZeroConfig-työkalu, joka on komentorivipohjainen apuväline kohdeympäristön konfiguroimiseen. Sille annetaan käynnistysparametrina ympäristö, jolle käyttöjärjestelmä halutaan konfiguroida. Se etsii halutun ympäristön projektikansiosta tarvittavat otsikkotiedostot ja kopioi ne include-kansioon, jossa kaikki ohjelmamoduulien käyttämät otsikkotiedostot sijaitsevat. ZeroConfig myös generoi käyttöjärjestelmän käyttämät prosessi- ja viestitunnukset.

3.3 Xilinx ML403 Evaluation Board

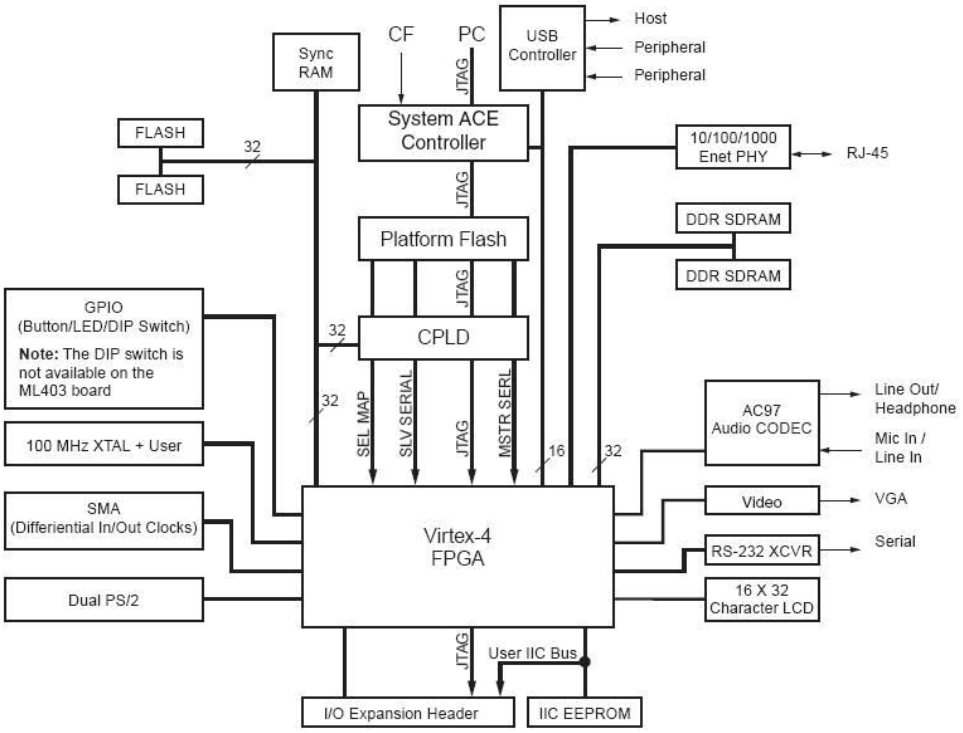
Kohdeympäristöksi valittiin kuvassa 3 esitetty Xilinx ML403 Evaluation Board, joka on erityisesti sulautettujen järjestelmien kehitykseen tarkoitettu ympäristö. Kortilla on 64 MB DDR-muistia, AC97-äänipiiri, RS-232-sarjaportti, VGA-ulostulo ja PS-2-portit näppäimistölle ja hiirelle. Siinä on myös kahden rivin LCD-näyttö ja CF-muistikorttipaikka. Lisäksi kortilla on 5 nappia ja 9 LEDiä, joita voidaan ohjata ohjelmallisesti.



Kuva 3. Xilinx ML403 Evaluation Board

Ympäristön ”sydän” on Virtex-4 FPGA -piiri, jolle voidaan ladata erilaisia laitteistokomponentteja. Kuvassa 4 on esitetty järjestelmän lohkokaavio. Ohjelma voidaan ladata piirille joko muistikortilta tai JTAG-väylän kautta System ACE Controller -piirin avulla. Kuvassa on merkitty nuolilla laitteiston liitännät.

FPGA-piiri sisältää PowerPC 405 ”hard” -prosessorin, ja sille voidaan ladata myös MicroBlaze ”soft” -prosessori.

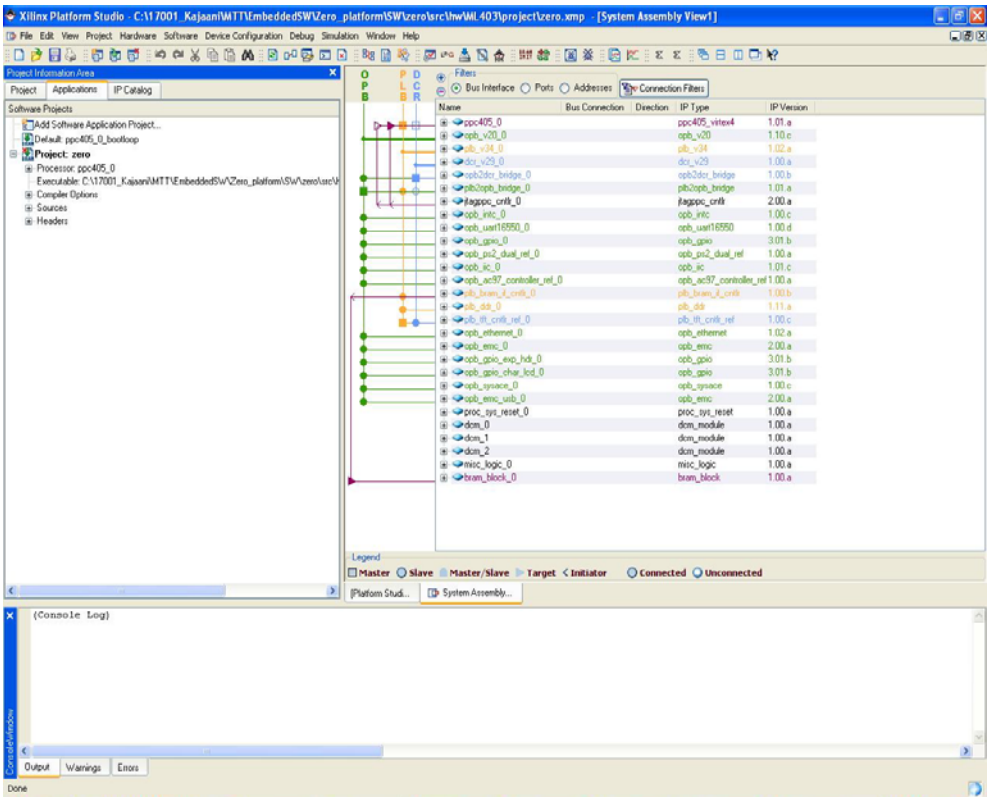


Kuva 4. Xilinx ML403 -kehitysympäristön lohkokaavio [10]

4 OHJELMISTONKEHITYSYMPÄRISTÖ

4.1 Xilinx Platform Studio

Laitteistoriippuvaisten ohjelmamoduulien kehitykseen ja kääntämiseen käytettiin Xilinx ML403 -kehitysympäristön mukana toimitettavaa Platform Studio -ohjelmistoa (XPS). XPS-ohjelma tarjoaa integroidun kehitysympäristön ohjelmistojen ja laitteistojen kehitykseen sulautettuihin PowerPC- ja MicroBlaze-prosessoriympäristöihin. Laitteiston konfigurointi, sekä prosessorin, oheislaitteiden ja väylien kytkeminen tapahtuu kuvassa 5 esitetyn graafisen käyttöliittymän kautta. Se myös tarjoaa editorin ja projektinhallintarajapinnan lähdekoodien luomiseen ja editointiin. Ohjelmistojen kääntämiseen ja linkkaukseen se käyttää GNU C-kääntäjää. Suoritettava ohjelmatiedosto voidaan ladata piirikortille XPS-ohjelmalla JTAG väylän kautta. JTAG-väylä mahdollistaa myös sulautetun ohjelmiston debuggauksen ja suorittamisen. [11.]

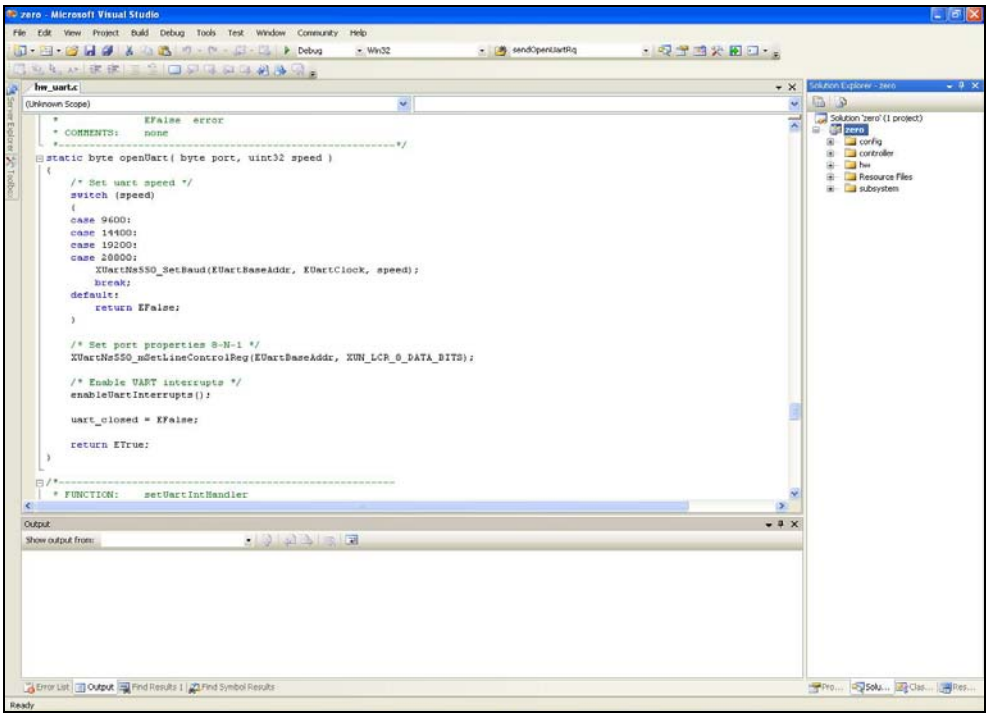


Kuva 5. Xilinx Platform Studio

4.2 Microsoft Visual Studio 2005

Lähdekoodien luomiseen ja editointiin päätettiin käyttää kuvassa 6 esitettyä Microsoftin Visual Studio 2005 -ohjelmistonkehitysympäristöä. Visual Studio on ohjelmistonkehitysympäristö, joka tukee useita eri ohjelmointikieliä, kuten Visual Basic, C#, C++ ja J#. Sillä voidaan tehdä esimerkiksi Windows-, web- ja mobiilisovelluksia ja siihen voidaan integroida monia täydennyksiä ja työkaluja. Visual Studiossa erityisesti graafisten sovellusten luominen on helppoa, mikä selittää suurelta osin sen suosion. Siinä on myös erittäin hyvä editori varustettuna Intellisense-toiminnolla, joka on todella hyvä apu ohjelmistonkehityksessä. [12.]

Visual Studiota päätettiin käyttää lähdekoodien editoimiseen lähinnä sen hyvän editorin ja projektinhallintaominaisuuksien vuoksi. Zero-käyttöjärjestelmästä on kehitetty myös Windowsin alla toimiva versio, jolla voidaan emuloida sen toimintaa sulautetussa järjestelmässä. Tällöin voidaan käyttää Visual Studion kehittyneitä debuggaus-toimintoja aivan kuten kehitettäessä Windows-ohjelmia.

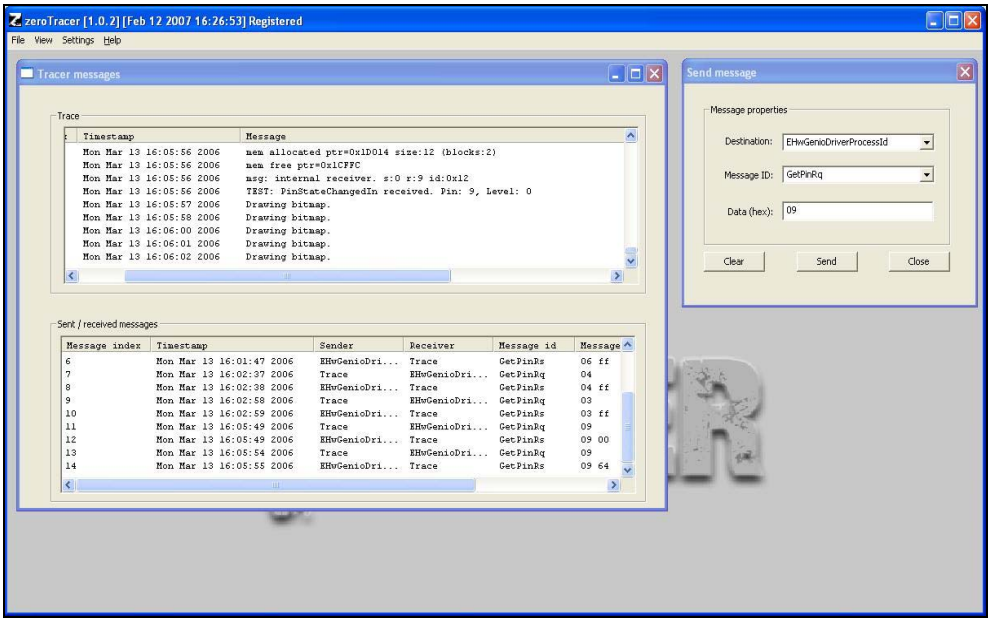


Kuva 6. Visual Studio 2005 -ohjelmistonkehitysympäristö

4.3 ZeroTracer

Trace-käsitteellä tarkoitetaan ohjelmistonkehityksessä ohjelman toiminnasta kertovaa tulostusta. Siitä on paljon apua vian etsinnässä, mutta sitä voidaan käyttää myös ohjelman toiminnan verifioinnissa. Ohjelma voi pyöriessään tulostaa tietoa esimerkiksi muistin varauksesta, minkä avulla voidaan mahdollisesti havaita muistivuodot.

Kuvassa 7 esitetty ZeroTracer on Windows-ohjelma, joka rekisteröityy Zero-käyttöjärjestelmään subsystemiksi sarjakaapelin kautta. Se on tarkoitettu käyttöjärjestelmän lähettämän tracen vastaanottoon, ja se mahdollistaa myös Zero-viestien lähettämisen käyttöjärjestelmälle. Viestien avulla voidaan ohjata käyttöjärjestelmän toimintaa, kuten esimerkiksi sytyttää LED-valo. Käyttöjärjestelmä reitittää ZeroTracer-ohjelmalle menevät ja sieltä tulevat viestit sarjaportin laiteajurin kautta.

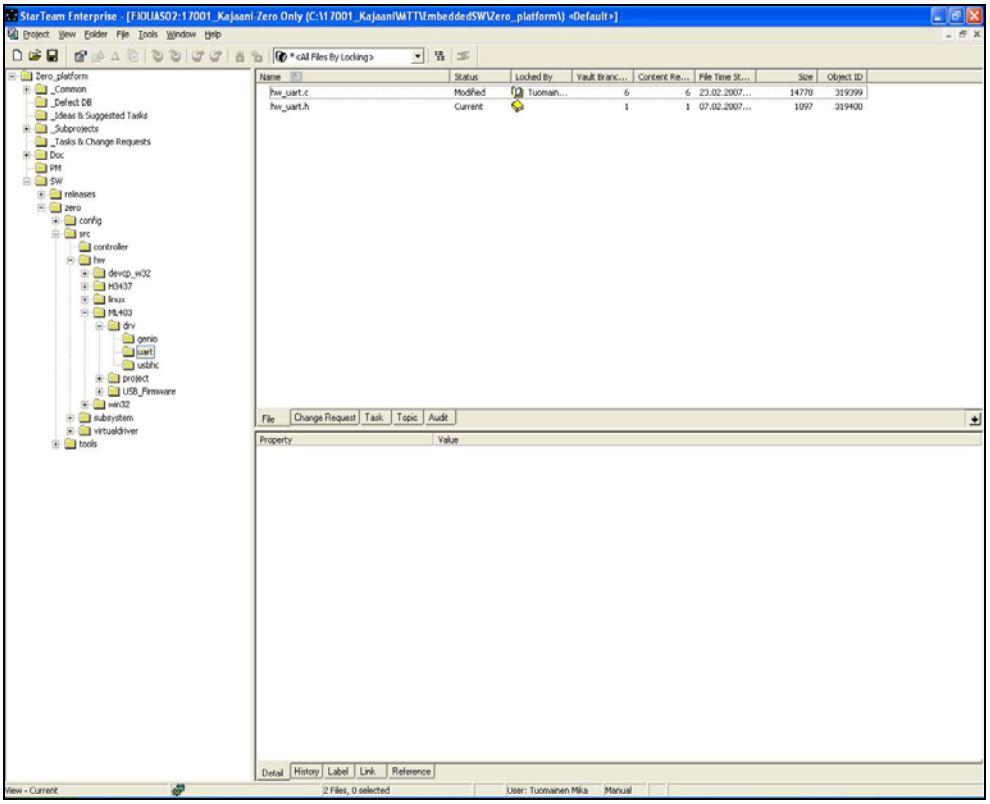


Kuva 7. ZeroTracer-debuggaustyökalu

4.4 Borland StarTeam -versionhallintatyökalu

Useiden ihmisten työskennellessä samassa projektissa kasvaa versionhallinnan merkitys. Versionhallinta mahdollistaa ohjelmiston kehityskaaren seurannan ja turvallisen kehityksen. Se

mahdollistaa myös kehityksen etenemisen eri suuntiin, mikä on usein välttämätöntä räätälöitäessä ohjelmistosta eri versiota eri alustoille ja asiakkaille. Versionhallinta mahdollistaa myös aiempaan kehitysversioon palaamisen, joka voi olla toivottavaa esimerkiksi silloin, kun siinä on suorituskyvyn tai ominaisuuksien kannalta joitakin haluttavia piirteitä, joita uusimmassa kehitysversiossa ei enää ole. Tässä projektissa käytettiin kuvassa 8 esitettyä Borland StarTeam -versionhallintatyökalua sekä dokumenttien että lähdekoodien versionhallintaan.



Kuva 8. StarTeam-versionhallintatyökalu

StarTeam toimii asiakas/palvelin-periaatteella, eli sen avulla käyttäjä voi tallentaa tiedostoja palvelimelle, jossa säilytetään kaikki sinne tallennetut versiot. Tämä mahdollistaa vanhojen versioiden palautuksen tarvittaessa. Tiedoston muokkausta varten käyttäjä hakee tiedoston palvelimelta omalle koneelle. Ennen tiedoston muokkausta käyttäjän tulee lukita tiedosto, mikä estää muita käyttäjiä muokkaamasta samaa tiedostoa yhtä aikaa. Tarvittavien muutosten jälkeen tiedosto tallennetaan uudestaan palvelimelle ja lukitus poistetaan, jonka jälkeen se on myös muiden käyttäjien muokattavissa. StarTeam-ohjelman avulla voidaan myös hallita muutospyynnöt ja avoimet tehtävät. Sen avulla voidaan myös vertailla eri tiedostoversioiden eroja

ja tutkia, mitä muutoksia on tehty sen jälkeen kun kovalevyllä oleva tiedosto on haettu palvelimelta. Tiedostojen sijainti palvelimella mahdollistaa myös varmuuskopioiden säilyttämisen. Tiedostot voidaan palauttaa palvelimelta, vaikka ne olisi tahattomasti poistettu työasemalta.

5 OHJELMISTONMÄÄRITTELY

Työn tekeminen aloitettiin ohjelmistonmäärittelyllä, jossa päätettiin, mitä ominaisuuksia projektissa tullaan toteuttamaan ja mitkä ominaisuudet jätetään projektin ulkopuolelle. Tämä on erittäin tärkeä työvaihe projektin onnistumisen kannalta. Liian moni ohjelmistoprojekti on epäonnistunut sen takia, ettei heti alussa ole tehty selvää suunnitelmaa toteutettavista ominaisuuksista. Huono suunnittelu yhdistettynä siihen tosiasiaan, että ohjelmistoprojekteissa hyvin harvoin on aloitettaessa selvillä mitä kaikkea vaadittujen ominaisuuksien toteuttaminen vaatii, on erittäin huono yhdistelmä.

Määrittelyvaiheessa päätettiin, että käyttöjärjestelmän sovittamisen lisäksi Xilinx ML403 -alustalle tullaan toteuttamaan laiteajurit sarjaportin ja I/O-liitäntöjen, kuten LED tai näppäin, ohjaukseen. Näistä sarjaportti on ehdottomasti tärkein, koska sen kautta ohjelmiston kehittäjä voi saada tietoa ohjelman toiminnasta. Näiden tietojen pohjalta aloitettiin projektisuunnitelman laatiminen. Projektisuunnitelman tärkein tehtävä on määritellä toimintaresurssit ja aikataulu. Aikataulun suunnittelu toteutettiin MS Project -ohjelmalla, johon ensin listattiin työn toteuttamiseen vaadittavat työvaiheet karkealla tasolla. Tämän jälkeen työvaiheita vielä tarkennettiin, jotta niiden toteuttamiseen kuluvan ajan arvioiminen olisi helpompaa. Jokaiseen työvaiheeseen kuluva aika arvioitiin, ja projektille määritettiin kolme tärkeää etappia, jossa aikataulua vielä tarkennettaisiin. Tämä on tärkeää, koska projektin alkuvaiheessa on lähes mahdotonta arvioida tarkkaan tiettyyn työvaiheeseen kuluvaa aikaa.

Projektissa toteutettavien toiminnallisuuksien ja niihin käytettävissä olevan ajan selvittyä, voitiin aloittaa vaatimusten määrittely toteutettaville toiminnoille. Periaatteena on, ettei toiminnoille asetettuja vaatimuksia muuteta enää projektin aikana, ellei muutos ole ohjelmiston toiminnan kannalta välttämätön. Määrittelyssä esitettyjen vaatimusten tulee olla sellaisia, että ne voidaan myöhemmin testata toimiviksi. Hyvin tehtyä määrittelyä voidaan hyödyntää suoraan testauksen määrittelyssä. Testauksen määrittelyn lähtökohtana on, että jokainen vaatimus tulee myös testattua toimivaksi.

Käyttöjärjestelmän sovitus

Zero-käyttöjärjestelmä on suunniteltu yleiskäyttöiseksi, mutta silti on joitain toimintoja, jotka ovat laitteistoriippuvaisia ja ne tulee toteuttaa jokaiselle laitteistolle erikseen. Seuraavassa on esitelty nämä toiminnot.

Zero-käyttöjärjestelmä tarvitsee toimiakseen kaksi ajastinta, Common- ja Callback-ajastimet, joista Common-ajastinta käytetään käyttöjärjestelmän tarjoamien ajastimien tahdistukseen ja Callback-ajastinta niiden keskeytysten hoitamiseen. Myös muistinvaraus ja -vapautus ovat laitteistoriippuvaisia toimintoja, jotka on konfiguroitava jokaiselle laitteistolle. Näiden lisäksi laitteistoajureilla tulee olla mahdollisuus rekisteröidä keskeytyskäsittelijä Zero-käyttöjärjestelmään, jotta ne voivat reagoida laitteiston keskeytyksiin. Laitteistoajurien täytyy pystyä myös sallimaan ja kieltämään keskeytystensä tapahtuminen.

Sarjaportin laitteistoajuri

Ajurin täytyy pystyä lähettämään ja vastaanottamaan sarjamuotoista dataa RS-232-portista. Data voi olla ennalta määriteltyjä Zero-viestejä tai raakadataa. Avattaessa porttia tiedonsiirtoa varten tulee siirtonopeus pystyä valitsemaan. Laiteajurin täytyy tukea seuraavia nopeuksia: 9600, 14400, 19200 ja 28800 bps. Portti täytyy myös pystyä sulkemaan, jonka jälkeen dataa ei enää voida lähettää eikä vastaanottaa suljetun portin kautta.

I/O-pinnien laitteistoajuri

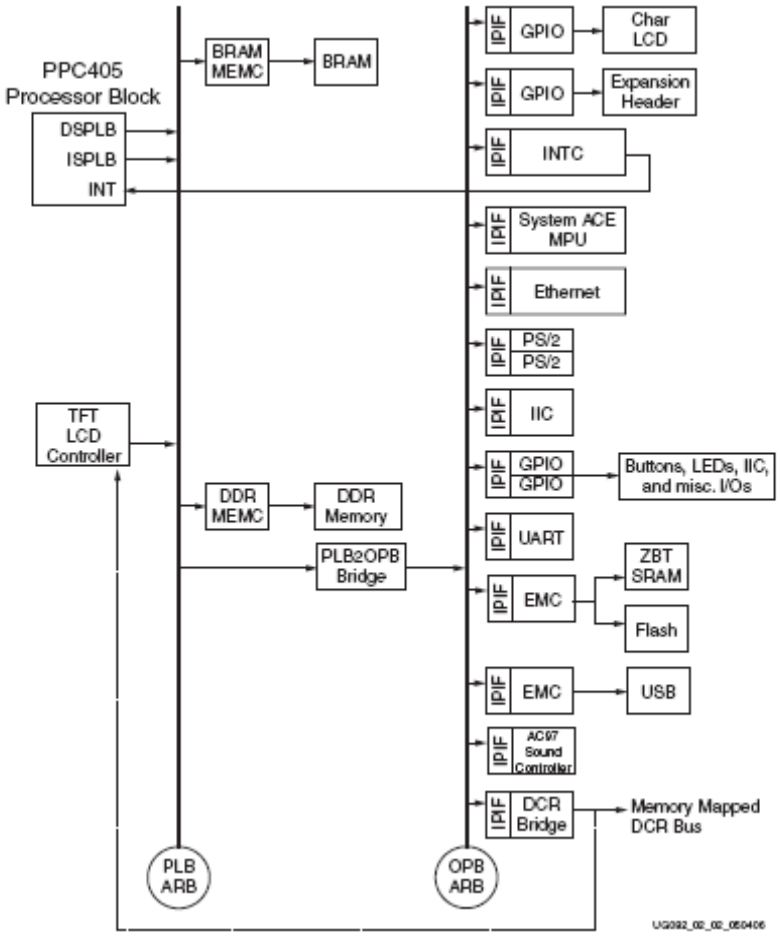
Jokaiselle I/O-pinnille täytyy pystyä konfiguroimaan suunta eli ovatko ne luku- vai kirjoitustyyppisiä. Lukutyyppisten pinnien arvo täytyy pystyä lukemaan ja kirjoitustyyppisten pinnien arvo täytyy vastaavasti pystyä asettamaan. Pinnien arvot ilmaistaan prosenttiarvona välillä 0–100 %. Ajurin tulee tarkastaa suunta jokaisen operaation yhteydessä. Sen tulee tukea useita pinnejä, ja sen toiminta ei saa häiritä käyttöjärjestelmän toimintaa, vaikka pinnin arvon lukeminen tai asettaminen epäonnistuisi.

6 ZERO-KÄYTTÖJÄRJESTELMÄN SOVITTAMINEN XILINX-LAITEYMPÄRISTÖÖN

Zero-käyttöjärjestelmän sovittaminen uuteen laiteympäristöön koostuu laitteistoriippuvaisten toimintojen uudelleentoteutuksesta. Seuraavana on selostettu kaikki Zero-käyttöjärjestelmän sovitukseen tarvitut työvaiheet, niihin liittyvät tehtävät ja saavutetut tulokset.

6.1 Toteutuksen suunnittelu

Xilinx ML403 -kehitysympäristö koostuu Virtex-4 FPGA -piiristä sekä siihen liitetyistä laitepiireistä ja liitännöistä. Ensimmäinen tehtävä ohjelmistoa suunniteltaessa olikin päättää, kuinka FPGA-piirille ladattava laitteisto tullaan toteuttamaan. Tämän insinöörityön tarkoituksena oli toteuttaa tarvittava ohjelmisto, jotta Zero-käyttöjärjestelmää voidaan käyttää kohdeympäristössä. Tästä johtuen oli luonnollista rajata laitteiston toteuttaminen tämän työn ulkopuolelle. Laitteistona päätettiin käyttää kuvassa 9 esitettyä valmistajan tarjoamaa PowerPC-prosessoriin perustuvaa referenssilaitteistoa.



Kuva 9. Kohdeympäristössä käytettävän laitteiston lohkokaavio [13]

Valittu laitteisto perustuu kahteen erilliseen väylään, jotka ovat Processor Local Bus (PLB) ja On-Chip Peripheral Bus (OPB). PLB on väylistä nopeampi ja on tarkoitettu PowerPC-prosessorin ja muistien kytkemiseen laitteistoon. OPB on hieman hitaampi, mutta myös yksinkertaisempi toteutukseltaan ja on tarkoitettu oheislaitteiden kytkemiseen.

Kuvan 9 IPIF-lohkot ovat valmiita FPGA-piirille HDL-kielellä toteutettuja komponentteja. Tässä työssä tullaan käyttämään kuvassa esitetyistä komponenteista UART- (Universal Asynchronous Receiver Transmitter), INTC- (Interrupt Controller) ja GPIO- (General Purpose Input Output) lohkoja, sekä totta kai PPC405-prosessoria.

PowerPC-prosessori tarjoaa vain yhden ohjelmallisesti asetettavan ajastimen, jota voidaan käyttää Zero-käyttöjärjestelmän tarpeisiin. Zero-käyttöjärjestelmä tarvitsee kuitenkin toimiakseen kaksi erillistä ajastinta. Ongelma ratkaistiin määrittelemällä yksi yhden millisekunnin

mittainen laitteistoajastin, josta muodostetaan ohjelmallisesti käyttöjärjestelmän vaatimat kaksi ajastinta.

Oheislaitteiden keskeytysten käsittely on toteutettu järjestelmässä Interrupt Controller -keskeytyspiirin avulla. Vastaanottaessaan keskeytyksen joltain siihen kytketyltä oheislaitteelta keskeytyspiiri aiheuttaa keskeytyksen prosessorille. Käyttöjärjestelmän laiteajureiden täytyy pystyä itse käsittelemään kyseessä olevan laitteen keskeytykset. Tästä johtuen Zero-käyttöjärjestelmän HW-kirjastomoduuliin päätettiin lisätä funktiot, joiden avulla laiteajurit voivat rekisteröidä keskeytysaliohjelmia käsittelemään oheislaitteiden keskeytyksiä. HW-kirjastomoduuliin tulee myös muistinvarauksen ja -vapautuksen toteuttavat makrot sekä laitteiston ja ajastimien alustukset.

6.2 Ohjelmistoprojektin luominen

Ohjelmiston kehitys toteutettiin Xilinx Platform Studio -kehitysympäristössä käyttäen C-ohjelmointikieltä. Kehitystyö aloitettiin luomalla uusi ohjelmistoprojekti, johon lisättiin käyttöjärjestelmän laitteistoriippumattomat ohjelmamoduulit. Laiteriippuvaisesta HW-kirjastomoduulista tehtiin ns. stub eli tynkätoteutus, jossa rajapinnan vaatimista funktioista toteutettiin pelkkä tyhjä runko.

Sulautettujen järjestelmien ohjelmistoprojektien luonnin yhteydessä täytyy lähes aina määritellä käytettävän ohjelmistokääntäjän asetukset itse. Tyypillisiä asetuksia, jotka kääntäjälle täytyy määritellä, ovat mm. muistialueiden koot, ohjelman käyttämät polut ja käännetyn tiedoston nimi.

Ohjelmamoduulit voivat varata muistia sekä staattisesti pinosta (stack) että dynaamisesti keosta (heap). Pinon kooksi määritettiin kahdeksan kilotavua ja keon kooksi 64 kilotavua. Yleensä suuret muistialueet varataan dynaamisesti ajon aikana, ja tästä johtuen täytyy keon olla pinoa suurempi. Kääntäjän asetuksiin määriteltiin myös Zero-käyttöjärjestelmän include-kansio, josta löytyy kaikki ohjelmamoduulien tarvitsemat otsikkotiedostot. Otsikkotiedostot kopioitiin include-kansioon ZeroConfig-työkalun avulla. Työkalu täytyy suorittaa uudestaan aina, kun järjestelmään halutaan lisätä uusia ohjelmistomoduuleja tai jos jo olemassa olevien ohjelmistomoduulien otsikkotiedostoihin tehdään muutoksia.

6.3 Keskeytysten alustus

PowerPC 405 -prosessorissa on kahdenlaisia keskeytyksiä, kriittisiä ja ei-kriittisiä. Kriittiset keskeytykset suoritetaan ensin ja ne voivat keskeyttää ei-kriittisen keskeytyksen käsittelyn. Tästä johtuen ne käyttävät eri rekistereitä ympäristön talletukseen kuin ei-kriittiset keskeytykset.

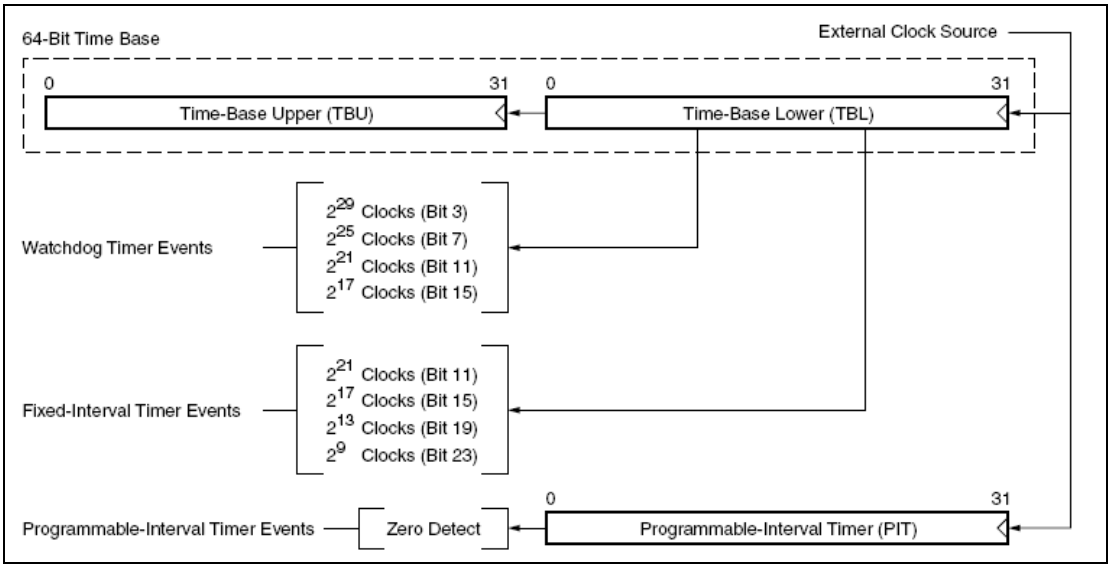
Molempien keskeytysten ensitasonkeskeytystenkäsittelyn hoitaa prosessori. Ohjelmiston tehtäväksi jää ainoastaan kuitata keskeytys sen aiheuttaneelle laitteelle, sekä keskeytysaliohjelmien asettaminen ja tarvittavien keskeytysten salliminen. Keskeytyksen sattuessa prosessori tallettaa automaattisesti ympäristön ja siirtää ohjelman suorituksen kyseessä olevan keskeytyksen keskeytysaliohjelmaan. Keskeytysaliohjelmasta poistuttaessa prosessori palauttaa ympäristön ja siirtää ohjelman suorituksen keskeytyneeseen prosessiin takaisin.

Keskeytyksien alustuksessa asetetaan ensin kaikille prosessorin keskeytyksille keskeytyskäsittelijäksi tyhjäfunktio eli niin sanottu tynkä. Tällä varmistetaan se, ettei pääse syntymään sellaista tilannetta, jossa prosessorille tulevalle keskeytykselle ei ole määritelty keskeytyskäsittelijää. Tällainen tilanne aiheuttaa yleensä aina vakavan virheen järjestelmässä.

Kuten kuvassa 9 on esitetty oheislaitteiden aiheuttamat keskeytykset tulevat prosessorille keskeytyspiirin (INTC) kautta. Keskeytyspiiri on kytketty prosessorin ulkoiseen keskeytykseen, joka on ei-kriittinen ja järjestelmän käynnistyessä estetty. Ulkoiset keskeytykset täytyy sallia ennen kuin oheislaitteiden aiheuttamat keskeytykset voivat aiheuttaa keskeytyksen prosessorille. Ennen sallimista täytyy ulkoiselle keskeytykselle asettaa keskeytysaliohjelma, joka käsittelee keskeytyksen. Keskeytyspiiriltä on keskeytysten aiheuttaminen myös estetty järjestelmän käynnistyessä ja ne täytyy sallia ennen kuin piiri voi aiheuttaa keskeytyksen prosessorille. Piiriltä on käynnistyksen jälkeen myös laitteistokeskeytykset estetty. Ennen kuin laitteistokeskeytykset sallitaan, on piirille mahdollisuus aiheuttaa ohjelmallinen keskeytys, jota voidaan käyttää esimerkiksi testaus tarkoitukseen. Normaalikäytössä, sekä laitteistokeskeytykset, että keskeytysulostulo sallitaan keskeytysten alustuksen yhteydessä. Piiriin liitettyjen laitteiden keskeytykset täytyy vielä sallia erikseen niiden alustuksen yhteydessä, mutta laitteistonkeskeytysten alustuksen yhteydessä ne jätetään estetyiksi.

6.4 Ajastimet

PowerPC 405 -prosessori tarjoaa kuvan 10 mukaiset ajastinresurssit. Se sisältää kaksi ajastinta, 64-bittisen ylöspäin laskevan Time Base -ajastimen ja 32-bittisen alaspäin laskevan Programable-Interval Timer -ajastimen (PIT). [14.]



Kuva 10. PowerPC 405:n ajastinresurssit [14]

Time Base -ajastimelle voidaan asettaa kahdenlaisia tapahtumia. Wathdog-timer-tapahtuma tarjoaa mahdollisuuden asettaa kriittisen keskeytyksen, jossa voidaan esimerkiksi yrittää toipua järjestelmävirheestä. Fixed-interval timer -tapahtuma tarjoaa mahdollisuuden asettaa keskeytyksen, joka toistetaan ennalta määrätyin väliajoin. Sitä voidaan käyttää esimerkiksi kellonajan määritykseen. [14.]

Kumpikaan näistä ei sovellu käytettäväksi Zero-käyttöjärjestelmän ajastimena, koska ne molemmat ovat kiinteämittaisia. Käyttöjärjestelmän ajastimeksi tarvitaan ajastin, jolle voidaan ohjelmallisesti asettaa tarvittava viive, jonka jälkeen ajastin aiheuttaa prosessorille keskeytyksen. PIT-ajastimelle asetettava tapahtuma on juuri tällainen. PIT-ajastin on 32-bittinen rekisteri, jonka arvoa vähennetään kellopulssin tahdissa. Se alkaa laskea alaspäin, kun sille ladataan nollasta eroava luku, ja se lopettaa vasta, kun sen sisältö on jälleen nolla. PIT-ajastimen arvon ollessa yksi aiheutuu PIT-tapahtuma seuraavan vähennyksen yhteydessä. PIT-ajastimen arvo vähennyksen jälkeen riippuu siitä onko se asetettu auto-reload-moodiin. Auto-reload-moodissa PIT-ajastimelle ladataan automaattisesti sama luku kuin edellisellä lataus-

kerralla. Ilman auto-reload-moodia PIT-ajastimen arvoksi tulee nolla ja laskeminen pysähtyy. Kaikkien ajastintapahtumien ohjaus tapahtuu Timer-Control-rekisterin (TCR) avulla ja tapahtumien tilat voidaan lukea Timer-Status-rekisteristä (TSR). [14.]

PIT-tapahtuma aiheuttaa prosessorille PIT-keskeytyksen vain, jos PIT-keskeytykset on sallittu TCR-rekisteristä ja ulkoinen keskeytys on sallittu Machine-State-rekisteristä (MSR). Ulkoiset keskeytykset on sallittu jo keskeytysten alustuksen yhteydessä, mutta PIT-keskeytykset ovat oletustilassaan estetty. Ennen PIT-keskeytysten sallimista tulee suorittaa ajastimen alustustoimet. Ajastimen alustuksessa PIT-keskeytykselle asetetaan keskeytysaliohjelma, joka suoritetaan keskeytyksen sattuessa ja ajastimen arvoksi asetetaan nolla, jolla varmistetaan se että laskenta on pysähdyksissä. Tarkoituksena on toteuttaa 1ms jatkuvasti toistuva ajastin, joten auto-reload moodi asetetaan päälle. [14.]

Alustustoimien jälkeen sallitaan PIT-keskeytykset, mutta laskeminen ei ala ennen kuin ajastimelle ladataan alkuarvo. Ajastimen alkuarvo voidaan laskea kaavasta

$$Alkuarvo = Kellotaajuus * Viive ,$$

jossa kellotaajuus on ajastimen laskentanopeus ja viive on laskennassa kuluva aika sekantteina. Prosessorin kellotaajuus on 300MHz ja se tulee suoraan kuvassa 10 esitettyyn ajastimen External Clock Source -nastaan. Ajastimen arvoa siis vähennetään 300 miljoonaa kertaa sekunnissa. Millisekunnin viive saadaan siis aikaan lataamalla ajastimeen luku 300 000. Laskenta alkaa välittömästi alkuarvon lataamisen jälkeen.

Kuten ohjelmistonmäärittely kappaleessa sanottiin, Zero-käyttöjärjestelmä tarvitsee kaksi ajastinta toimiakseen. Nämä kaksi tarvittavaa ajastinta muodostetaan PIT-keskeytyksen keskeytysaliohjelmassa ohjelmallisesti. PIT-keskeytys aiheutuu millisekunnin välein, joten ohjelmallisesti siitä voidaan muodostaa useita erimittaisia yli millisekunnin kestäviä viiveitä. PIT-keskeytyskäsittelijä hoitaa ohjelmallisten ajastinten hallinnan ja niille määrättyjen funktioiden kutsumisen. Lopuksi PIT-keskeytyskäsittelijä kuittaa PIT-keskeytyksen käsitellyksi.

6.5 Oheislaitteiden keskeytysaliohjelmien rekisteröinti

Oheislaitteiden keskeytysnastat on järjestelmässä kytketty keskeytyspiirin kautta PowerPC-prosessorin keskeytykseen ja tälle keskeytykselle on asetettu keskeytyskäsittelijä keskeytysten

alustuksen yhteydessä. Tämän keskeytyskäsittelijän tehtävänä on kuitenkin käsitellä keskeytyspiirin aiheuttama keskeytys, eikä keskeytyspiiriin liitetyn oheislaitteen keskeytys. Keskeytysohjainpiirin tehtävä onkin ottaa selvää, mikä laite aiheutti keskeytyksen keskeytyspiirille ja siirtää keskeytyksen käsittely kyseisen laitteen laiteajurille. Käyttöjärjestelmän tulee siis tarjota laiteajureille mahdollisuus rekisteröidä keskeytyskäsittelijä, jota keskeytyspiirin keskeytyskäsittelijä kutsuu. Tämän lisäksi käyttöjärjestelmän tulee tarjota laiteajureilla palvelu, jonka avulla ne voivat sallia ja estää tietyn keskeytyksen aiheutuminen.

Nämä palvelut päätettiin toteuttaa HW-kirjastomoduuliin, koska ne ovat laitteistoriippuvaisia toimintoja ja täytyy toteuttaa joka laitteistolle erikseen. Keskeytyskäsittelijää rekisteröitäessä laitteistoajurin on ilmoitettava käyttöjärjestelmälle sen keskeytyksen tunnus, jolle keskeytyskäsittelijää ollaan rekisteröimässä. Käyttöjärjestelmä tarvitsee tämän tiedon, jotta se osaa liittää käsittelijän oikeaan keskeytykseen. Sovelluskehittäjien työn helpottamiseksi päätettiin kaikkien keskeytyspiiriin liitettyjen laitteiden keskeytystunnukset määritellä HW-kirjastomoduuliin. Laitteistoajurit voivat käyttää näitä määriteltyjä tunnuksia mystisten heksadesimaalilukujen sijaan. Keskeytysten sallimisessa ja estämisessä laiteajurin tulee välittää käyttöjärjestelmälle erityinen maskitavu, josta käyttöjärjestelmä tietää mitä keskeytyksiä ollaan estämässä tai sallimassa. Myös nämä maskitavut määriteltiin HW-kirjastomoduuliin, josta ne ovat laitteistoajurien käytettävissä.

6.6 Moduulitestaus

Moduulitestaus tässä tapauksessa tarkoittaa HW-kirjastomoduuliin testausta, johon laitteistoriippuvaiset käyttöjärjestelmän toiminnot on toteutettu. Testattavat toiminnot ovat muistinvaraus ja -vapautus, ajastimet ja keskeytykset. Ongelmaksi muodostuu se miten nämä toiminnot pystytään testaamaan. Zero-käyttöjärjestelmässä on toiminto, jossa se tulostaa automaattisesti toiminnastaan kertovaa informaatiota. Tämä informaatio sisältää mm. muistinvaraus- ja -vapautus tiedot, sekä tiedon subsystemien käynnistämien ajastinten laukeamisesta ja käynnistyksestä. Oletuksena Zero-käyttöjärjestelmä käyttää tulostuskanavana sarjaportin laiteajuria, jota ei vielä tässä vaiheessa ole toteutettu. Yksi mahdollisuus olisi toteuttaa sarjaportinlaiteajuri ensin ja testata HW-kirjastomoduuli vasta sen jälkeen. Tämä ei kuitenkaan ole kovin käyttökelpoinen idea, koska sarjaportinlaiteajuri tarvitsee HW-kirjastomoduulin palveluja toiminnassaan.

Ratkaisu tähän ongelmaan löytyi ML403-ympäristön kirjastomoduuleista. Se tarjoaa mahdollisuuden tulostukseen sarjaportin kautta käyttäen sen omaa sarjaportin laiteajuria. Tämähän on juuri sopiva työkalu käytettäväksi moduulitestaukseen. ML403-ympäristö voidaan kytkeä sarjakaapelilla PC-tietokoneeseen ja näyttää tulostukset sen näytöllä.

Emme kuitenkaan voi käyttää PC:ssä ZeroTracer-työkalua tulostuksien vastaanottoon, koska sen täytyy pystyä rekisteröitymään Zero-käyttöjärjestelmään. Rekisteröityminen ei ole mahdollista ML403-ympäristön kirjastomoduulin sarjaporttiajurin välityksellä vaan rekisteröitymiseen vaaditaan Zero-viestejä tukeva laiteajuri. HW-kirjastomoduulin testauksessa emme tarvitsekaan ZeroTracer-ohjelman kehittyneitä toimintoja vaan meille riittää se, että saamme tulostettua sarjaportista vastaanotetun datan näytölle. Tähän tarkoitukseen sopiva ohjelma on Windows-käyttöjärjestelmään integroitu HyperTerminal.

Nyt meillä on tulostuskanava ja ohjelma, jolla tulostukset saadaan näytölle. Vielä ennen testauksen aloittamista meidän täytyy muuttaa Zero-käyttöjärjestelmän Tester/Trace-moduulia siten, että se käyttää tulostukseen ML403-ympäristön kirjastomoduulin tulostusfunktiota käyttöjärjestelmän oman tulostusfunktion sijasta.

Nyt käynnistettäessä ohjelma se alkaa välittömästi tulostaa tietoja toiminnastaan sarjaporttiin, jotka voidaan lukea HyperTerminal-ohjelmalla PC:ssä. Muistinhallinnan toiminta voidaan varmistaa tulostuksista, jotka tulostetaan aina muistia varatessa ja vapautettaessa. Muistinvarauksen tulostuksessa näkyy varatun muistipaikan osoite ja vastaavasti vapautuksen yhteydessä näkyy vapautetun muistipaikan osoite. Tulostuksesta varmistetaan, että jokainen varattu-muistipaikka myös vapautetaan.

Zero-käyttöjärjestelmä myös tulostaa tiedon ajastimen käynnistyksestä ja laukeamisesta. Tästä tulostuksesta pystyttiin varmistamaan, että ajastimen käynnistys ja laukeaminen toimi niin kuin pitääkin. Tulostuksesta ei kuitenkaan pystytä varmistamaan, että ajastimelle asetettu viive on oikean mittainen. Ajastimen viiveen määritystä varten lisättiin PIT-keskeytyksen keskeytysaliohjelmaan ylimääräinen tulostus. Jos tulostus suoritettaisiin joka kerta, pitäisi tulostusten väli olla 1ms. Millisekunnin viiveen mittaaminen on kuitenkin käytössä olevilla työkaluilla lähes mahdoton tehtävä. Kymmenen sekunnin viive pystytään jo mittaamaan riittävän tarkasti. Kymmenen sekunnin viive tulostusten väliin saadaan tulostamalla teksti vain joka 10 000:s kerta.

Keskeytysaliohjelmien rekisteröinnin testaamista varten täytyy toteuttaa tynkälaiteajuri, jossa on keskeytysaliohjelmat jokaiselle keskeytyspiirin keskeytykselle. Tynkälaiteajurin ainoa toteutus on, että se alustuksen yhteydessä rekisteröi jokaisen keskeytysaliohjelman. Keskeytysaliohjelmissa tulostetaan tieto, mikä keskeytys on kyseessä. Keskeytysalustusta muutetaan siten, että ennen laitteistokeskeytysten sallimista aiheutetaan ohjelmallisesti jokainen mahdollinen keskeytys piirille. Nyt keskeytysten rekisteröinnin toiminta voidaan varmistaa HyperTerminal-ohjelman tulostuksesta.

7 LAITEAJURIEN IMPLEMENTOINTI

Laiteajuri sisältää laitteenohjauksessa käytettävät tietorakenteet ja tarjoaa käyttöjärjestelmälle sen tarvitsemat palvelut. Tarvittavia palveluja ovat mm. laitteen alustus, luku ja kirjoitus, ohjausrutiinit ja laitteeseen liittyvät keskeytystenkäsittelyrutiinit. [1.]

Zero-käyttöjärjestelmässä prosessien skedulointi perustuu FIFO-menetelmään, joten myös laiteajuriprosesseilla on sama prioriteetti kuin kaikilla muillakin prosesseilla. Laiteajuriprosessit kyllä alustetaan ennen muita prosesseja ja ne pääsevät myös ensimmäisinä käynnistyksen jälkeen ajoon, mutta tämän jälkeen ne eivät eroa muista prosesseista mitenkään. Kaikkien Zero-käyttöjärjestelmän laiteajureiden tulee toteuttaa Zero-viestirajapinta ja OOC-funktiokutsurajapinta, jotta ne voivat toimia järjestelmässä. Zero-käyttöjärjestelmän laitteistoriippumattomat komponentit käyttävät näitä rajapintoja kommunikointiin laitteiston kanssa, joten on erittäin tärkeää, että nämä rajapinnat pysyvät muuttumattomina eri laitteistoympäristöjen välillä.

7.1 Sarjaportin ohjaus

Sarjaportin laiteajurin tehtävänä on avata sarjaportti oikealla nopeudella, lähettää ja vastaanottaa dataa, sekä sulkea portti. Zero-käyttöjärjestelmän sarjaportin laiteajurin vaatimuksena oli lisäksi, että sen pitää tukea raakadatan lisäksi myös erityisiä Zero-viestejä. Zero-viestit ovat järjestelmän sisäisiä viestejä, joiden avulla ohjelmamoduulit kommunikoivat keskenään. Viesti voidaan osoittaa joko tietylle moduulille, tai voidaan lähettää niin sanottu indikaatio viesti, jolla ei ole erityistä vastaanottajaa. Zero-viesti koostuu otsikko-osasta ja data-osasta, joista otsikko-osa sisältää tiedot lähettäjästä, vastaanottajasta, viestin pituudesta ja viestintyypistä. Otsikko-osaa seuraa data-osa, jossa on varsinainen tieto siitä mitä viestin vastaanottajalle halutaan kertoa.

7.1.1 Suunnittelu

Sarjaportin laiteajurin toteuttaminen aloitettiin suunnittelemalla toteutettava ohjelmisto. Suunnittelu aloitettiin tutustumalla kuvassa 9 esitetyn UART-piirin (Universal Asynchronous Receiver/Transmitter) dokumentaatioon. Tärkeimmät selvitettävät asiat olivat mitä alustuksia laitteelle tulee tehdä, kuinka nopeus voidaan asettaa, miten keskeytykset saadaan käyttöön, miten laitteeseen kirjoitetaan ja miten sieltä luetaan dataa.

Zero-käyttöjärjestelmän vaatimat Zero-viestirajapinta ja OOC-funktiokutsurajapinta on toteutettu jo aikaisemmissa projekteissa. Tässä projektissa sarjaportin laiteajurille täytyi vain toteuttaa rajapinnat toteuttavat laitteistoriippuvaiset funktiot.

Laiteajurin toteutukseen on yleensä kaksi tapaa. Pollaukseen eli jatkuvaan laitteen tilan tiedusteluun perustuva tapa on yleensä helpompi toteuttaa, mutta jatkuva laitteen tilan kysely kuormittaa järjestelmää aivan tarpeettomasti. Myös laitteen tilan muutoksiin reagointi voi olla hidasta, jos järjestelmässä on monia tähän menetelmään perustuvia laiteajureita. Yleensä onkin suositeltavampaa käyttää keskeytykseen perustuvaa toteutusta, jossa laite aiheuttaa keskeytyksen kun sen tilassa tapahtuu muutos. Tällä vältetään tarpeeton tilan kysely ja tilan muutokseen voidaan reagoida välittömästi. Tässä projektissa päätettiin käyttää keskeytykseen perustuvaa toteutusta datan vastaanotossa ja pollaukseen perustuvaa tapaa

7.1.2 Sarjaportin avaus

Alustettaessa sarjaportin laiteajuria sarjaportti avataan käyttäen järjestelmän oletusnopeutta, jolloin hoidetaan myös sarjaliikenteestä vastaavan Uart-piirin alustus. Zero-käyttöjärjestelmän ohjelmamoduulit voivat muuttaa käytettävää nopeutta avaamalla portin uudestaan haluamallaan nopeudella.

Portin avauksessa ensimmäinen tehtävä on rekisteröidä keskeytyskäsittelijä Uart-piirin keskeytyksille. Keskeytyskäsittelijän rekisteröinnin jälkeen asetetaan portin nopeus. Ohjelmiston määrittely vaiheessa (Kappale 5) tuettavien nopeuksien määrä rajattiin neljään, joka helpottaa erityisesti testaustyötä. Tilanteessa jossa yritetään asettaa virheellinen nopeus, portin avaus keskeytetään ja tulostetaan virheilmoitus.

Sarjaportin avauksessa tulee myös asettaa portin asetukset. Portin asetuksilla määritetään kuinka montaa data- ja lopetusbittiä käytetään, sekä millainen pariteettitarkistus halutaan käyttöön. Portin asetuksiksi asetettiin 8 databittiä, 1 lopetusbitti ja pariteettitarkistus poistettiin käytöstä.

Viimeinen tehtävä portin avauksessa on sallia tarvittavat keskeytykset. Uart-piiri voi aiheuttaa vastaanotto, lähetys ja modeemin hallinta keskeytyksiä. Nämä keskeytykset voidaan sallia tai estää yksitellen. Nyt toteutettava laiteajuri tarvitsee vain vastaanotosta ilmoittavaa keskeytystä, joten se sallitaan ja muut estetään. Uart-piirin keskeytykset täytyy vielä sallia myös keskeytyspiiriltä ennen kuin ne pääsevät prosessorille asti.

7.1.3 Datan lähetys

Zero-käyttöjärjestelmän ohjelmamoduulit voivat lähettää sarjaporttiajurin avulla ulkoisille ohjelmamoduuleille tarkoitettuja Zero-viestejä tai vaihtoehtoisesti aivan pelkkää raakadataa. Toteutettaessa sarjaportinlaiteajuria ei kuitenkaan tarvitse ottaa kantaa siihen millaista dataa ollaan lähettämässä. Sarjaporttilaiteajuri tarjoaa ulospäin rajapinnan, jonka kautta ohjelmamoduulit voivat välittää ajurille haluamansa määrän dataa. Laiteajurin tehtävänä on kirjoittaa data Uart-piirille.

Uart-piirille voidaan dataa kirjoittaa vain merkki kerrallaan, jonka piiri lähettää sarjaväylän kautta vastaanottajalle. Vasta lähetyksen päätyttyä voidaan piirille kirjoittaa uudestaan. Myös tässä voitaisiin käyttää keskeytyspohjaista ratkaisua, jossa Uart-piiri ilmoittaisi keskeytyksellä milloin lähetys on valmis ja uusi tavu voidaan kirjoittaa. Tässä tapauksessa keskeytysten käytöllä saavutetaan kuitenkin hyvin vähän hyötyä, koska yhden tavun lähettämiseen kuluu verrattain vähän aikaa ja lähetettävät datamäärät eivät ole kovin suuria. Datan lähetyksessä päätettiin käyttää menetelmää, jossa merkin kirjoituksen jälkeen pollataan laitetta kunnes lähetys on valmis ja uusi merkki voidaan kirjoittaa.

7.1.4 Datan vastaanotto

Datan vastaanotossa keskeytyspohjainen ratkaisu on ehdottomasti paras ratkaisu, koska ei voida ennalta tietää milloin dataa täytyy vastaanottaa. Portin avauksen yhteydessä laitteistoon

rekisteröitiin keskeytysaliohjelma, joka käsittelee sarjaportin vastaanoton keskeytykset. Sarjaportin laiteajurilla on käytössään lukupuskuri, johon vastaanotettu data luetaan Uart-piiriltä keskeytysaliohjelmassa.

Sarjaportin vaatimuksena oli, että sen pitää pystyä vastaanottamaan sekä erityisiä Zero-viestejä että sarjaporttiin tulevaa raakadataa. Sarjaportin laiteajuriprosessissa vastaanotettu viesti lähetetään Message Handler -moduulille, jossa tunnistetaan onko kyseessä Zero-viesti vai raakadata. Jotta Zero-viestin tunnistus olisi mahdollista, täytyy koko viesti olla vastaanotettuna ennen kuin se välitetään Message Handler -moduulille. Sarjaportinlaiteajuri odottaa, että dataa on vastaanotettu vähintään Zero-viestin otsikko-osan verran. Otsikko-osa sisältää tiedon data-osan pituudesta, josta saadaan tieto kuinka paljon dataa täytyy olla vastaanotettuna, jotta koko viesti on vastaanotettu. Message Handler -moduuli tunnistaa, mikä viesti on kyseessä ja osaa reitittää sen edelleen.

Raakadataa vastaanotettaessa on mahdollista, että ollaan vastaanottamassa pienempi määrä dataa kuin mitä Zero-viestin koko on. Tällaista tilannetta varten sarjaportin laiteajuriin määriteltiin aikakatkaisu datanvastaanottoon. Aikakatkaisu määrittelee maksimi ajan, joka voi kulua vastaanotettujen merkkien välillä. Tämän ajan ylittyessä data tulkitaan raakadataksi ja laiteajuri lähettää raakadatan edelleen indikaatioviestinä. Ohjelmamoduulit, jotka haluavat vastaanottaa raakadataa, voivat rekisteröityä kuuntelemaan näitä raakadatan vastaanoton indikaatioviestejä.

7.1.5 Moduulitestaus

Sarjaporttilaiteajurin testattavat toiminnot ovat sarjaportin avaus eri nopeuksilla, datanlähetys, Zero-viestien vastaanotto ja raakadatanvastaanotto. Portin avauksen ja Zero-viestien vastaanoton testaukseen voidaan käyttää ZeroTracer-työkalua. ZeroTracer-työkalussa voidaan valita käytettävä siirtonopeus vastaamaan portin avauksessa käytettyä nopeutta. Tätä ominaisuutta voidaan hyödyntää portin avauksen testauksessa. ZeroTracer-työkalulla voidaan lähettää järjestelmän ohjelmamoduuleille Zero-viestejä. Vastaanottaessaan viestin moduuli lähettää kuittauksen viestin vastaanottamisesta sen lähettäjälle. ZeroTracer-työkalun vastaanottaessa kuittausviestin voidaan sarjaporttilaiteajurin Zero-viestien tukeminen todeta toimivaksi. Samalla tulee testattua myös viestinlähetys, koska käyttäähän Zero-käyttöjärjestelmä kuittauksen lähettämiseen sarjaporttilaiteajurin datanlähetys ominaisuutta.

Raakadatan lähettämisen testaaminen ei onnistu ellei järjestelmässä ole ohjelmamoduulia, joka kuuntelee raakadataviestejä. Tämän ominaisuuden testaamista varten jouduttiinkin kehittämään niin sanottu testipeti. Testipeti on ohjelma, joka on suunniteltu pelkästään jonkin tietyn asian testaamista varten. Raakadatanvastaanoton testausta varten toteutettiin subsystem, joka kuuntelee raakadataviestejä sarjaportista. Vastaanottamansa datat se kaiuttaa takaisin sarjaporttiin. Tämän jälkeen raakadatanvastaanotto pystyttiin testaamaan lähettämällä HyperTerminal-ohjelmalla merkki sarjaporttiin, jonka toteuttamamme testipeti kaiuttaa takaisin ja HyperTerminal-ohjelma tulostaa sen näytölle.

7.2 I/O-pinnien ohjaus

I/O-pinnien ohjauksen hoitavan laiteajurin eli GenIO-laiteajurin tehtävänä on tarjota järjestelmälle mahdollisuus lukea ja kirjoittaa I/O-pinnien tiloja. Jokainen ajurin ohjaama pinni voi olla joko luku- tai kirjoitustyyppinen. Lukutyyppisiä pinnejä ovat muun muassa laitteiston näppäimet ja kirjoitustyyppisiä LEDit. GenIO-laiteajuri tarjoaa rajapinnan, jolla Zero-käyttöjärjestelmän ohjelmamoduulit voivat asettaa tai lukea pinnin tilan sen tunnuksen perusteella. GenIO-laiteajuri ei ota kantaa siihen mitä fyysistä laitetta tietty tunnus tarkoittaa, vaan se on ohjaavan sovelluksen tiedossa. Laiteajuri tietää kuitenkin onko kyseessä luku- vai kirjoitustyyppinen pinni ja sen fyysisen osoitteen.

7.2.1 Suunnittelu

I/O-pinnien ohjaus tapahtuu kuvassa 9 esitetyn GPIO-piirin (General Purpose Input Output) kautta. Suunniteltaessa GenIO-laiteajuria täytyi ensimmäisenä selvittää kuinka GPIO-piirin alustus, siitä lukeminen ja sille kirjoittaminen hoidetaan. Tämän laiteajurin vaatimuksena oli, että sen täytyy tarjota Zero-käyttöjärjestelmän ohjelmamoduuleille rajapinta, jonka avulla ne voivat kirjoittaa ja lukea pinnien arvoja. Zero-käyttöjärjestelmän vaatimat viestirajapinta ja OOC-funktiokutsurajapinta on toteutettu myös tälle laiteajurille aikaisemmissa projekteissa, joten vain rajapinnat toteuttavat funktiot täytyi toteuttaa tässä projektissa.

Kuten kuvassa 9 on esitetty, GPIO-piiri on kytketty OBP-väylään muiden oheislaitteiden tavoin. GPIO-piiri on yleiskäyttöinen IO-porttien ohjaukseen käytettävä piiri. Tässä laitteis-

tossa GPIO-piiriä käytetään LEDien ja näppäimien ohjauksen lisäksi myös LCD-näytön ja laajennusliitännän ohjaukseen. GPIO-piirin ohjaus tapahtuu 32-bittisen ohjaussanan avulla, jossa jokainen bitti edustaa yhtä pinniä, eli yhdellä ohjaussanalla voidaan maksimissaan ohjata 32:ta pinniä.

Ohjelmistoa määriteltäessä päätettiin, että pinnejä täytyy pystyä ohjaamaan prosentuaalisella arvolla välillä 0–100 %. LEDin tila voi kuitenkin olla vain päällä tai pois päältä, joten päätettiin että nollasta eroavalla arvolla LED sytytetään ja nollalla se sammutetaan.

7.2.2 Alustus

Jokaisella I/O-pinnillä täytyy olla määriteltynä tunnus, jolla sitä pystytään ohjaamaan ja suunta eli onko se kirjoitus- vai lukutyyppinen. Kohdeympäristön I/O-pinneille määriteltiin tunnus ja tyyppi HW-kirjastomoduuliin, josta ne ovat myös sovellusten käytettävissä. Pinnien tunnukset asetettiin juoksevan numeroinnin mukaan ja kirjoitustyyppisille pinneille määriteltiin myös oletusarvo, eli mikä on pinnin arvo järjestelmän käynnistyessä. HW-kirjastomoduuliin määriteltiin mitä pinniä mikäkin bitti ohjaussanassa ohjaa.

Alustettaessa GenIO-laiteajuria kirjoitetaan HW-kirjastomoduulissa määritetyt suunta- ja oletusarvotiedot myös GPIO-piirille. Tietojen kirjoitusta varten muodostetaan ohjaussanat molemmille kirjoitettaville tiedoille. Suuntatietojen kirjoitusta varten muodostetaan ohjaussana, jossa ”1”-tilassa oleva bitti tarkoittaa lukutyyppistä pinniä ja ”0”-tilassa oleva bitti kirjoitustyyppistä. Suuntatietojen kirjoituksen jälkeen voidaan kirjoittaa GPIO-piirille kirjoitustyyppisten pinnien oletusarvot. Oletusarvojen kirjoitusta varten muodostetaan uusi ohjaussana, jossa jokaisen bitin arvo tarkoittaa kyseisen pinnin oletusarvoa. Bitin arvo voi olla vain 0 tai 1, mutta laiteajuri ohjaa pinnejä prosentuaalisilla arvoilla. Ennen kuin arvo voidaan asettaa ohjaussanaan, täytyy se muuntaa binääriseksi. Muunnos tehtiin suunnitteluvaiheessa päätetyn periaatteen mukaisesti, jossa kaikki nollasta eroavat arvot tulkitaan ykköseksi. Ohjaussanassa lukutyyppisten pinnien arvoa ilmaisevat bitit asetettiin nollaksi. Niiden arvolla ei kuitenkaan ole mitään merkitystä, koska kirjoittamisella lukutyyppiseen pinniin ei ole mitään vaikutusta. Tästä johtuen on myös erittäin tärkeää, että pinnien suuntatieto on asetettu ennen oletusarvon kirjoittamista.

7.2.3 Arvon asettaminen ja lukeminen

Zero-käyttöjärjestelmään liitetyt ohjelmamoduulit voivat asettaa kirjoitustyyppisten pinnien arvon. Niiden täytyy välittää GenIO-laitteistoajurille asetettavan pinnin tunnus ja asetettava arvo prosenttilukuna. GenIO-laitteistoajurissa tarkistetaan ennen pinnin asetusta, että asetettava pinni on, määritelty HW-kirjastomoduulissa, on kirjoitustyyppinen ja asetettava arvo on välillä 0–100.

Argumenttien tarkastusten jälkeen muodostetaan GPIO-piirille välitettävä ohjaussana. Ohjaussanan muodostuksessa tulee huomioida se, että kirjoitettaessa ohjaussana piirille se asettaa kaikkien piirin ohjaamien pinnien arvon. Kuitenkin tarkoituksena on asettaa vain yhden pinnin arvo ja jättää muiden arvot koskemattomiksi. Sen tekeminen ilman, että tiedetään muiden pinnien arvot, on kuitenkin mahdotonta. Ohjaussanan muodostusta varten onkin ensin luettava piiriltä pinnien nykyiset arvot. Tämän lisäksi täytyy muodostaa niin sanottu maski, jossa vain haluttu bitti on ”1”-tilassa ja kaikki muut bitit ovat ”0”-tilassa. Tarvittava ohjaussana yhden pinnin asettamiseen ”1”-tilaan saadaan suorittamalla AND-operaatio GPIO-piiriltä luettujen arvojen ja maskin välillä. ”0”-tilaan asettamista varten täytyy maskin bitit invertoida, jonka jälkeen nollauksen suorittava ohjaussana saadaan suorittamalla OR-operaatio piiriltä luettujen arvojen ja invertoidun maskin välillä. Haluttu pinni saadaan asetettua kirjoittamalla muodostettu ohjaussana GPIO-piirille.

Ohjelmamoduulien lukiessa lukutyyppisten pinnien arvoa, täytyy GPIO-piiriltä luettu binäärinen arvo muuttaa uudestaan prosenttiluvuksi. ”1”-tila tarkoittaa päällä olevaa ja siten sille annetaan muutoksessa arvo 100 % ja ”0”-tila tulkitaan luonnollisesti arvoksi 0 %. Vastaavasti kuten pinnin arvoa kirjoitettaessa niin myös lukiessa on tarkastettava, että luettava pinni on määritelty HW-kirjastomoduulissa. On myös varmistettava, että ollaan lukemassa lukutyyppisen pinnin arvoa. Luettaessa GPIO-piiri palauttaa kaikkien pinnien arvot ohjaussanana. Ohjaussanasta täytyy vielä selvittää mikä on juuri halutun pinnin arvo. Tämä onnistuu siirtämällä bitinsiirto operaatiolla ohjaussanan bittejä halutun bitin sijainnin verran oikealle. Tämän jälkeen haluttu bitti on ensimmäisenä oikealta päin luettaessa. Halutun bitin edessä olevien bittien arvoa ei kuitenkaan tiedetä, joten ne täytyy nollata ennen kuin voidaan selvittää halutun bitin arvo. Bittien nollaaminen saadaan aikaiseksi suorittamalla AND-operaatio ohjaussanan ja numeron yksi välillä. Nyt ohjaussana on joko yksi tai nolla, ja kun se kerrotaan luvulla sata,

saadaan arvo skaalattua halutulle alueelle. Funktion lopuksi vielä palautetaan pinnin arvo sitä kysyvälle ohjelmamoduulille.

7.2.4 Moduulitestaus

GenIO-laiteajurin moduulitestausta varten toteutettiin pieni testausohjelma, jota käytettiin myös demo-ohjelmana projektin tuotosten esittelyssä. Demo-ohjelma lukee GenIO-laiteajurin lukutyyppisten pinnien tiloja aina saadessaan suoritusvuoron. Laitteiston näppäimet on määritelty lukutyyppisiksi ja käyttäjänohjelmoitavat LEDit kirjoitustyyppisiksi. Demo-ohjelma tietää näppäinten tunnukset, ja osaa siten tulkita eri näppäinten painallukset eri tavalla. Tämän demo-ohjelman pääasiallinen tarkoitus onkin testata GenIO-laiteajurin käytettävyyttä enemmän kuin itse moduulin toimintaa, ja tästä johtuen se ei yksinään riitä moduulitestauksen hyväksymiseen.

GenIO-laiteajuri testattiin demo-ohjelman lisäksi myös ZeroTracer-työkalulla. Kuten sarjaportin laiteajurin moduulitestauksessa kerrottiin, ZeroTracer-ohjelmalla voidaan lähettää Zero-käyttöjärjestelmän sisäisiä viestejä suoraan tietylle ohjelmamoduulille. GenIO-laiteajurin testauksessa lähetettiin sille viesti, jossa pyydettiin asettamaan LEDiksi määritelty pinni päälle. Laitteistosta tarkastettiin, että oikea LED syttyi palamaan. ZeroTracer-ohjelmasta varmistettiin, että GenIO-laiteajuri lähetti LEDin sytytysviestiin vastauksen, jossa se ilmoittaa pinnin arvon asetuksen onnistuneen. Vastaava toimenpide toistettiin jokaiselle järjestelmän LEDille. Järjestelmään liitettyjen näppäinten arvon lukeminen testattiin lähettämälle GenIO-laiteajurille viesti, jossa luetaan yhden näppäimen arvo. GenIO-laiteajurin tulee oikein toimiessaan vastata tähän viestiin lähettämällä vastausviesti, josta selviää luetun pinnin tunnus ja arvo. Näppäimen ollessa painettuna arvon tulee olla 100 %, ja näppäimen ollessa normaaliasennossa arvo on 0 %. Kaikki järjestelmän näppäimet testattiin ensin ilman, että näppäin on painettu, jonka jälkeen näppäin painettiin pohjaan ja varmistettiin arvon muuttuminen.

8 ANALYSOINTI JA POHDINTA

Projektin lopputuloksena Zero-käyttöjärjestelmä toimii Xilinxin ML403 -sulatetun järjestelmän kehitysympäristössä. Käyttöjärjestelmään toteutettiin kaksi laiteajuria, jotka ohjaavat sarjaporttia ja järjestelmän I/O-pinnejä. Kaikki työlle asetetut vaatimukset saavutettiin projektisuunnitelmassa laaditun aikataulun määräaikaan mennessä. Aikataulua jouduttiin tarkentamaan projektin edistyessä joidenkin käyttöjärjestelmän sovitukseen liittyvien tehtävien osalta, mutta menetetty aika pystyttiin ottamaan takaisin laiteajurien implementoinnissa. Projektin alkaessa suurimmaksi riskiksi arvioitiinkin juuri aikataulun onnistuminen käyttöjärjestelmän sovituksen osalta, koska siihen liittyi niin paljon etukäteen tuntemattomia asioita. Erityisesti huolta aiheutti se, ettei kohdeympäristöstä ollut aikaisempaa kokemusta. Aikataulun ylitykset sovitusvaiheessa olivat kuitenkin pienempiä mitä alun perin pelättiin, eikä niiden kiinniottamisessa projektin loppuvaiheessa ollut ongelmia.

Käyttöjärjestelmän sovitukseen kuuluu yleensä olennaisena osana järjestelmätestaus. Järjestelmän kattava testaus on erittäin tärkeää, koska projektin tarkoituksena on siirtää koko järjestelmä. Esimerkiksi muistinvaraukseen liittyvät toiminnot pystytään moduulitasolla testaamaan vain pintapuolisesti. Moduulitestauksessa verifioitiin, että muistilohkojen varaus onnistuu, ja että varattu muisti pystytään myös vapauttamaan. Kuitenkaan emme pysty varmistamaan, että muistia on riittävästi kun useampi ohjelmamoduuli toimii yhtä aikaa. ”Laitteistoriippumattomien” ohjelmamoduulien toiminnassa voi myös esiintyä yllättäviäkin ongelmia uudessa laitteistossa. Kattavan järjestelmätestauksen suunnittelu ja toteutus tämän koko luokan ohjelmistossa vaatii kuitenkin paljon aikaa. Tämän projektin aikataulutuksessa huomattiin, ettei kattavan järjestelmätestauksen toteuttamiseen ollut riittävästi aikaa. Kuitenkin projektin lopputuloksena syntynyt ohjelmisto otettiin käyttöön useammassakin Zero-käyttöjärjestelmän kehitysprojektissa heti valmistuttuaan. ML403-kehitysympäristöön kehitettiin muun muassa näytönohjainlaiteajuri, jossa erityisesti muistinvaraukseen ja -vapautukseen liittyvät toiminnot tuli kattavasti testattua.

Testauksen luonne tällaisessa projektissa, jossa on paljon uudelleenkäytettäviä ohjelmistokomponentteja, painottuu paljon järjestelmätestaukseen. Tässä projektissa voitiin keskittyä vain ohjelmiston toimintojen testaukseen, koska tarkoitus ei ole tuotteistaa ohjelmistoa. Tuotteistukseen tähtäävässä projektissa olisi otettava huomioon myös esimerkiksi tietoturvan testaus, jonka merkitys tulevaisuudessa kasvaa merkittävästi. Myös SoC-piirit asettavat oman

haasteensa ohjelmistontestaukseen. Periaatteessa ohjelmiston ei pitäisi olla riippuvainen SoC-piirille ladatusta laitteistosta vaan sen tulisi mukautua laitteiston muutoksiin. Tässä työssä tämä otettiin huomioon ohjelmistoa toteutettaessa, mutta kattava testaaminen jätettiin järjestelmätestauksen tapaan tämän projektin ulkopuolelle.

Projektissa saavutettiin kaikki sille asetetut tavoitteet ja valmis ohjelmisto oli vaatimusten mukainen. Projektissa opittiin muun muassa kuinka tärkeä asia hyvä suunnittelu on ohjelmistonkehityksessä. Zero-käyttöjärjestelmässä, jokaiselle ohjelmakomponentille on oma paikkansa, ja erityisesti hyvän suunnittelun ansiosta Zero-käyttöjärjestelmä on helposti siirrettävissä uuteen laitteistoon. Hyvä suunnittelu mahdollistaa myös ohjelmakomponenttien uudelleen käytettävyyden. Esimerkiksi laitteistorajapintaa ei samalle laitteelle tarvitse tehdä kuin kerran, jonka jälkeen samaa rajapintaa voidaan hyödyntää jokaisessa laitteistossa. Käytettävissä oleva muisti vaihtelee eri järjestelmien välillä hyvinkin paljon, joka asettaa käyttöjärjestelmän yleiskäyttöisyydelle suuren haasteen. Zero-käyttöjärjestelmän vaatimaa muistin määrää voidaan muuttaa muun muassa muuttamalla käytettävien puskurien kokoa ja tarpeettomia ohjelmakomponentteja voidaan helposti poistaa käytöstä. Ilman hyvää suunnittelua näiden asioiden teko vaatii hyvinkin paljon työtä ja monesti jopa uudelleensuunnittelua.

9 YHTEENVETO

Sulautetut järjestelmät ovat kasvaneet paljon viime vuosina ja tulleet entistä monipuolisemmiksi. Tarvitaan myös ohjelmistoja, jotka pystyvät vastaamaan tähän kehitykseen. Yhden ohjelman täytyy pystyä suoriutumaan mitä moninaisimmista tehtävistä ja pystyä toimimaan kaikenlaisissa järjestelmissä. Tätä varten tarvitaan käyttöjärjestelmiä, jotka yhdistävät käytettävän laitteiston ja laitteistoriippumattomat sovellukset toisiinsa.

Sanaan käyttöjärjestelmä mielletään usein Windows-käyttöjärjestelmän ikkunat ja napit, tai Dosin komentokehote. Nämä ovat kuitenkin vain käyttöjärjestelmän kuorrutus, käyttöliittymä, jolla ohjataan käyttöjärjestelmän toimintaa. Itse käyttöjärjestelmä on paljon muuta, eikä se välttämättä tarvitse käyttöliittymää ollenkaan. Käyttöjärjestelmän päätehtävänä onkin toimia liitäntänä sovellusohjelmien ja laitteiston välillä.

Työssä sovitettiin yleiskäyttöinen Zero-käyttöjärjestelmä toimimaan Xilinxin ML403 -sulautettujen järjestelmien kehitysympäristössä. Jatkokehitystä ja testausta varten toteutettiin myös laiteajurit sarjaportin ja I/O-liitäntöjen ohjaukseen. Työtä tehdessä perehdyttiin myös SoC-piirien tuomiin mahdollisuuksiin ja niiden käyttöön osana sovelluskehitystä.

Zero-käyttöjärjestelmässä laitteistoriippuvaisia toimintoja ovat muun muassa muistinhallinta, ajastimet ja keskeytysten käsittely. Sovittaminen uuteen laiteympäristöön koostuu näiden toimintojen uudelleen toteuttamisesta. Käyttöjärjestelmä kommunikoi laitteiston kanssa laiteajurien välityksellä, jotka eivät ole laitteistoriippumattomia. Laiteajurit täytyy siis toteuttaa jokaiselle laitteistolle erikseen. Kommunikointiin laiteajurien kanssa käyttöjärjestelmä käyttää ennalta määritettyjä rajapintoja, jotka pysyvät muuttumattomina eri laitteistojen välillä.

Työtä tehdessä havaittiin kuinka merkittävä asia suunnittelu on ohjelmiston kehityksessä. Erityisesti hyvä suunnittelu on mahdollistanut Zero-käyttöjärjestelmän helpon siirrettävyyden eri laitteistojen välillä. Ohjelmakomponenttien uudelleenkäytettävyyttä voidaan myös lisätä erittäin paljon hyvällä suunnittelulla. Nämä kaksi asiaa tuovat erittäin paljon säästöjä kehitettäessä samantyylisiä ohjelmistoja. Säästöjen lisäksi nämä asiat nopeuttavat ohjelmistonkehitysprosessia erityisesti dokumentointi työn vähenemisenä.

Uudelleenkäytettävien ja helposti siirrettävien komponenttien käyttö ei kuitenkaan välttämättä vähennä testaustyötä ainakaan merkittävästi. Testaustyön luonne muuttuu ja aikaisemmin

paljon aikaa vieneen moduulitestauksen tilalle tulee uudenlaisia testaustarpeita. Monien eri moduulien sovittaminen yhteen lisää integrointitestauksen tarvetta. Myös tietoturvariski kasvaa, koska yleensä ohjelmistojen tietoturva-aukot sijaitsevat juuri rajapinnoissa, joka lisää tarvetta tieturvan testaamiseen myös ohjelmistojen kehityksen yhteydessä.

LÄHTEET

1. Haikala, I. & Järvinen, H-M. 2003. Käyttöjärjestelmät. Jyväskylä. Gummerus.

2. Silberschatz, A. 2002. Operating System Concepts, New York. John Wiley & Sons, Inc.

3. Koskinen, J. 1986. Käyttöjärjestelmistä ymmärrettävästi. Helsinki. ATK-instituutti : Valtion painatuskeskus.

4. Koskinen, J. 2004. Mikrotietokonetekniikka Sulautetut järjestelmät. Keuruu. Otava

5. System-on-a-chip - Wikipedia. Luettu: 9.3.2007. [WWW-dokumentti] http://en.wikipedia.org/wiki/System-on-a-chip

6. Field-programable gate array - Wikipedia. Luettu: 5.3.2007. [WWW-dokumentti] http://en.wikipedia.org/wiki/FPGA

7. Embedded System Design in an FPGA - Wikipedia. Luettu: 5.3.2007. [WWW-dokumentti] http://en.wikipedia.org/wiki/Embedded_System_Design_in_an_FPGA

8. Embedded system - Wikipedia. Luettu: 5.3.2007. [WWW-dokumentti] http://en.wikipedia.org/wiki/Embedded_system

9. Zero SW Architectural Description

10. Xilinx - ML401/ML402/ML403 Evaluation Platform User Guide UG080 (v2.5) May 24, 2006, http://direct.xilinx.com/bvdocs/userguides/ug080.pdf

11. Xilinx - Embedded System Tools Reference Manual UG111 (v6.0) June 23, 2006 http://www.xilinx.com/ise/embedded/edk82i_docs/est_rm.pdf

12. Visual Studio - Wikipedia. Luettu: 14.3.2007. [WWW-dokumentti] http://fi.wikipedia.org/wiki/Visual_Studio

13. Xilinx - ML40x EDK Processor Reference Design User Guide for EDK 8.1 UG082 (v5.0) June 30, 2006, http://www.xilinx.com/bvdocs/userguides/ug082.pdf

14. Xilinx- PowerPC Processor Reference Guide UG011 (v1.2) January 19, 2007