VAASAN YLIOPISTO

TEKNILLINEN TIEDEKUNTA

AUTOMAATIOTEKNIIKKA

Otto Nurmi

FPGA-TEKNIIKAN OPETUKSEN KEHITTÄMINEN

Diplomityö, joka on jätetty tarkastettavaksi diplomi-insinöörin tutkintoa varten

Vaasassa 08.05.2015

Työn valvoja Professori Jarmo Alander

Työn ohjaaja TkT Petri Välisuo

ALKULAUSE

Tämä diplomityö on toteutettu Vaasan yliopiston teknillisen tiedekunnan sähkö- ja energiatekniikan yksikössä. Haluan kiittää työn ohjaajaa professori Jarmo Alanderia mielenkiintoisesta aiheesta sekä hänen neuvoistaan työn aikana. Lisäksi haluan kiittää TkT Birgitta Martinkauppia hänen avustaan palautekyselylomakkeen laatimisessa AUTO1010 Digitaalitekniikan perusteet kurssilla. DI Suvi Karhua haluan kiittää yhteistyöstä, jonka tuloksena työn tulokset esitellään konferenssijulkaisussa.

Vaasassa 08.05.2015

Otto Nurmi

SISÄLLYSLUETTELO **sivu**

SYMBOLI- JA LYHENNELUETTELO

ADC	Analog to Digital Converter
ASIC	Application Specific Integrated Circuit
CAD	Computer-Aided Design
CSR	Control and Status Register
DAC	Digital to Analog Converter
Deeds	Digital Electronics Education and Design Suite
DPI	Direct Programming Interface
DSO	Digital Storage Oscilloscope
DSP	Digital Signal Processor
FPL	Field Programmable Logic
FPGA	Field Programmable Gate Array
GPIO	General Purpose Input/Output
GUI	Graphical User Interface
HDL	Hardware Description Language
HDVL	Hardware Description and Verification language
HTTP	Hypertext Transfer Protocol
IEEE	Institute of Electrical and Electronics Engineers
IC	Integrated Chip
IP	Intellectual Property
I^2C	Inter-Integrated Circuit bus
JTAG	Joint Test Action Group
LA	Logic Analyzer
LabVIEW	Laboratory Virtual Instrument Engineering Workbench
LCD	Liquid Crystal Display
LUT	Lookup Table
PBL	Project Based Learning
PHP	Hypertext Preprocessor
PWM	Pulse-Width Modulation
RTL	Register Transfer Level
SCL	Serial clock line
SDA	Serial data line

SoC	System-on-Chip
SSH	Secure Shell
TTL	Transistor-Transistor Logic
UART	Universal Asynchronous Receiver Transmitter
USB	Universal Serial Bus
VD	Virtual Device
VDP	Virtual Devices Platform
VGA	Video Graphics Array
VHDL	VHSIC Hardware Description Language
VHSIC	Very High Speed Integrated Circuit
VVVF	Variable-Voltage Variable-Frequency

VAASAN YLIOPISTO
Teknillinen tiedekunta
Tekijä: Otto Nurmi
Diplomityön nimi: FPGA-tekniikan opetuksen kehittäminen
Valvojan nimi: Professori Jarmo Alander
Ohjaajan nimi: TkT Petri Välisuo
Tutkinto: Diplomi-insinööri
Koulutusohjelma: Sähkö-ja energiatekniikan koulutusohjelma
Suunta: Automaatiotekniikka
Opintojen aloitusvuosi: 2010
Diplomityön valmistumisvuosi: 2015 **Sivumäärä:** 96

TIIVISTELMÄ

Digitaalitekniikan käytön yleistyminen teknisissä tuotteissa ja sovelluksissa yhä etenevässä määrin tulee myös tekniikan alan korkeakoulutuksessa ottaa huomioon. Teknisissä tuotteissa digitaalitekniikkaa käytetään esimerkiksi älykkäissä säätöratkaisuissa sekä tuotannon automatisoinnissa, kuten robotiikassa. Digitaalitekniikkaa voidaan toteuttaa käyttäen mikroprosessoreita, mikrokontrollereita, signaaliprosessoreita, asiakaskohtaisia ja ohjelmoitavia logiikkapiirejä sekä kokonaisia piisirulle integroituja järjestelmiä.

Tässä diplomityössä tarkastellaan FPGA-tekniikan opetusta Vaasan yliopiston automaatiotekniikan opintosuunnassa, tarkasteltavana kurssina AUTO1010 Digitaalitekniikan perusteet. FPGA-piirejä käytetään kurssin opetusalustana niiden hyötyjen, kuten rinnakkaislaskennan vuoksi. Tarkoituksena on selvittää opetuksen mahdolliset kehityskohteet FPGA-tekniikan opetuksessa. Kurssin harjoitustyönä keväällä 2014 toteutettiin valourut FPGA:lla. Valourkujen toteutuksessa käytettiin laitevalmistajan suunnitteluohjelmaa, ledejä, vastuksia, koekytkentälevyjä ja kuulokemikrofonia. Toteutuksessa käytettiin VHDL-laitteistokuvauskieltä. Kurssin luennoilla ja laskuharjoituksissa käytiin tarvittava teoriaosuus lävitse, tarkoituksena antaa riittävästi pohjatietoa harjoitustyön suorittamiseksi. Harjoitustyön suorituksessa järjestettiin kolme ohjauskertaa, josta viimeisimmällä toteutettiin palautekysely koskien kurssin ja harjoitustyön suoritusta.

Kerättyä palautekyselyn tuloksia verrattiin muihin samankaltaisiin tutkimuksiin. Palautekyselyn ja vertailujen perusteella voidaan sanoa, että kurssin FPGA-tekniikan opetus on hyvällä tasolla, eikä merkittäviä muutoksia kurssin järjestelyyn tarvitse tehdä. Toteutetun palautekyselyn tuloksista julkaistiin myös konferenssiartikkeli. Tulevaisuudessa digitaalitekniikan opetukseen voisi ottaa mukaan SoC-järjestelmät, joissa ohjelmisto ja laitteisto ovat integroituja samalle piirille käyttöjärjestelmän (Linux) kanssa.

AVAINSANAT: FPGA-tekniikka, opetus, digitaalitekniikka

UNIVERSITY OF VAASA
Faculty of technology
Author: Otto Nurmi
Topic of the Thesis: Development of teaching FPGA-technology
Supervisor: Professor Jarmo Alander
Instructor: D.Sc.Tech Petri Välisuo
Degree: Master of Science in Technology
Degree Programme: Degree Programme in Electrical and Energy Engineering
Major of Subject: Automation Technology
Year of Entering the University: 2010
Year of Completing the Thesis: 2015 **Pages:** 96

ABSTRACT

As the use of digital technology becomes a common part of industrial engineering products and applications, so have also teaching of digital electronics become a vital part of engineering education. Engineering products where digital electronics is used are for example smart control systems and production automation, such as robotics. Digital technology can be implemented with using microprocessors, microcontrollers, digital signal processors, application specific circuits, programmable logic circuits and systems integrated in a chip.

In this thesis teaching of FPGA technology at Automation curriculum of University of Vaasa is examined. The course which is examined is AUTO1010 Introduction to digital electronics. FPGAs are used in course study platform because of their benefits, such as parallel computing. The purpose is to find out potential developing subjects in course teaching. The practical work of the course in Spring 2014 was light organ with FPGA. In implementation device manufacturer's software was used with leds, resistors, breadboards, and headphone microphone. The design language used in the project work was VHDL hardware description language. In the lectures and exercises necessary theory was introduced to manage carrying out the practical work. Three guidance lessons were held and in the last time a feedback survey was carried out.

The finished feedback survey was compared to other similar publications. On the basis of feedback survey and comparisons it can be said that teaching of FPGA technology is at a good level and there's no need for significant changes on the course implementation. The results from the feedback survey were also presented in a conference paper. In the future to teaching of digital technology could include the use of SoC systems where software and hardware can be integrated on the same IC-chip with operating system (Linux).

KEYWORDS: FPGA technology, teaching, digital technology

1. JOHDANTO

Digitaalitekniikan käytön yleistyessä teknisissä laitteissa tulee tulevaisuuden insinööreillä olla hyvät pohjatiedot kyseisen tekniikan suunnittelu- ja toteutusmenetelmistä. Tuotteita jotka hyödyntävät digitaalitekniikka ovat esimerkiksi suojareleet, taajuusmuuttajat, robotiikan sovellukset sekä muut automaatioratkaisut. Teknisissä sovelluksissa digitaalitekniikkaa voidaan soveltaa käyttäen mikroprosessoreita, mikrokontrollereita, digitaalisia signaaliprosessoreita (DSP), asiakaskohtaisia logiikkapiirejä (ASIC) sekä rinnakkaislaskentaan perustuvia kenttäohjelmoitavia logiikkapiirejä (FPGA).

FPGA-piirit tulivat markkinoille 1980-luvulla ja niiden käyttö eri teknisissä tuotteissa on lisääntynyt jatkuvasti. Tärkein tekijä niiden käytön jatkuvaan suosioon on niiden uudelleenohjelmoitavuus ja rinnakkaislaskenta, jolloin samaa piiriä voidaan käyttää eri sovelluksiin suhteellisen halvalla kehityskustannuksella. Viimeaikainen kehitys on mennyt kohti integroituja piirejä (IC), jossa yhdelle mikropiirille on mahdollista toteuttaa laitteisto- ja prosessorijärjestelmä (SoC). FPGA-piirit koostuvat logiikkaelementeistä, jotka on järjestetty matriisirakenteeksi. Logiikkaelementit yhdistetään ohjelmoitavan reitityksen avulla. Lisäksi piiri voi sisältää omia muisti/kertolaskuelementtejä. Piirille on mahdollisuus lisätä prosessori sulautettuna mikroprosessorilaitteina tai ohjelmistopohjaisena. FPGA-piirien digitaalinen logiikka toteutetaan käyttäen laitteistokuvauskieltä (VHDL, Verilog, SystemVerilog) ja ne soveltuvat hyvin prototyypitykseen. Niiden avulla voidaan toteuttaa esimerkiksi digitaalisia säätöratkaisuja, kuten esimerkiksi induktiomoottorin jännitteen ja taajuuden säädössä (VVVF) (Vinay, Shyam, Rishi & Moorthi 2011: 1–6).

FPGA-tekniikan opetus sisältyy monen korkeakoulun tekniikan alan opetusohjelmaan. Opetuksen toteuttamisessa osa yliopistoista käyttää etälaboratoriota, jotka toimivat internetin kautta, ja mahdollistavat opiskelijoiden joustavan työskentelymahdollisuuden. Lisäksi piirivalmistajien (Altera, Xilinx) tarjoamia kehitysalustoja käytetään opetuksessa ja prototyyppien kehityksessä. Kehitysalustat sisältävät FPGA-piirin sekä lisäkomponentteja kuten muisteja, kytkimiä ja LCD-näyttöjä. Kehitysalustojen käyttö mahdollistaa itse tehtävien ulkopuolisten kytkentöjen liittämisen kehitysalustalle niiden omien

komponenttien lisäksi. FPGA-piirien ohjelmoimiseen on piirivalmistajien tarjoamia ohjelmistoja, joiden avulla on mahdollisuus kääntää, simuloida ja syntetisoida haluttu piirikuvaus. Laitteistokuvauskielen osaaminen on yksi keskeinen taito FPGA-toteutuksia tehtäessä, ja monet korkeakoulut ovat panostaneet sen opetukseen käyttämällä valmistajien ohjelmien lisäksi omia itse toteutettuja suunnitteluohjelmia. On olemassa lisäksi kolmannen osapuolen tekemiä ohjelmistoja, joilla on mahdollisuus kääntää annettu piirikuvaus suoraan laitteistokuvauskielelle. FPGA-tekniikkaa voidaan käyttää opetusalustana digitaalisten järjestelmien, esimerkiksi digitaalisen signaalinkäsittelyn opetuksessa (Stastny, Ruckay 2006: 201–204).

Työn tarkoituksena on selvittää FPGA-tekniikan opetuksen tilaa Vaasan yliopiston automaatiotekniikan kursseissa. Tarkasteltavana kurssina on digitaalitekniikan peruskurssi AUTO1010. Vertailuaineistona käytettiin artikkeleita, jotka käsittelivät FPGA-tekniikan opetusta.

1.1. Työn rakenne

Tässä työssä toteutettiin palautekysely AUTO1010 Digitaalitekniikan perusteet kurssin toteutuksesta, erityisesti harjoitustyöstä. Kysely toteutettiin yhteistyössä TkT Birgitta Martinkaupin kanssa, jonka vastuulla oli kurssin laskuharjoitusten pitäminen. Kurssin harjoitustyönä toteutettiin FPGA sovellus, jonka nimi oli valourut. Palautekysely on liitteessä 1.

Kappaleessa 2 käydään lävitse FPGA-piirien arkkitehtuuria, suunnittelumenetelmiä, niiden hyötyjä verrattuna muihin toteutusmahdollisuuksiin sekä kolme eri laitteistokuvauskieltä. Arkkitehtuurista esitellään kaksi erilaista toteutusvaihtoehtoa. Laitteistokuvauskielistä esitellään kolme yleisintä VHDL, Verilog ja SystemVerilog. Lyhyesti käydään lävitse mikroprosessorien liittäminen FPGA-piirille. Kappaleessa 3 esitellään mahdollisuudet toteuttaa FPGA-tekniikan opetus käyttäen etälaboratoriota ja tavanomaisia laboratoriota ja opetusmenetelmiä. Lisäksi laitteistokuvauskielen opetusta selvitellään. Kappaleessa 4 esitellään kurssin laskuharjoitusten suoritus sekä harjoitustyön rakenne ja toteutus. Rakenteesta esitellään valourkujen rakenne, kehitysalustassa oleva

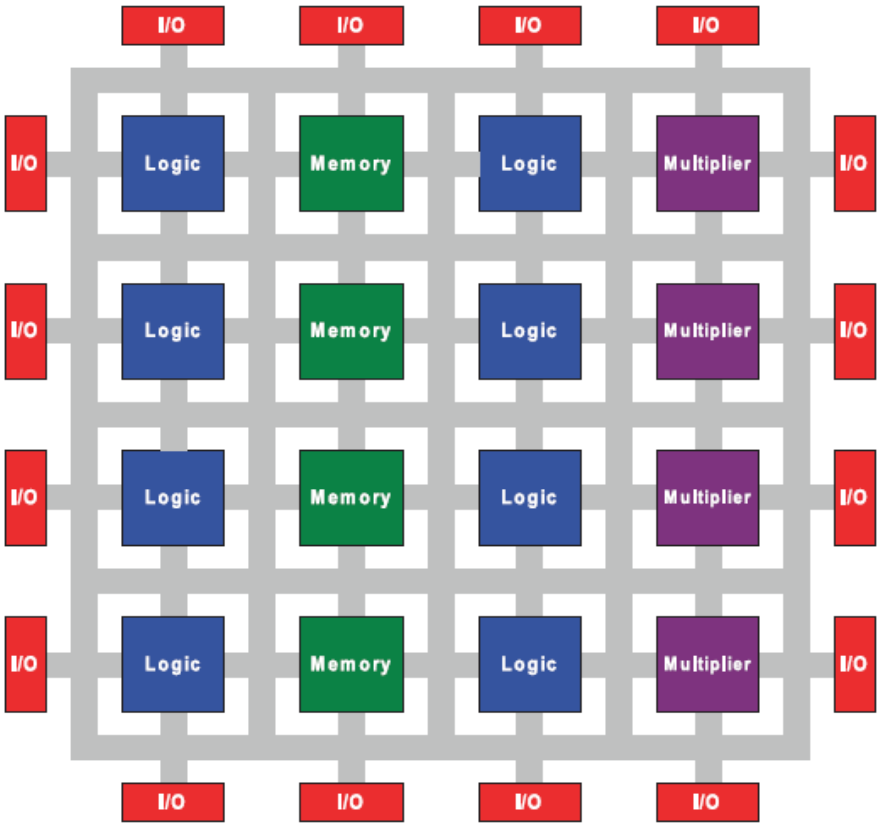
WM8731 Audio codec, FPGA-piirin ja koodekin välisen tiedonsiirron hoitava I^2C-väylä, pulssinleveysmodulaatio (PWM) joka opiskelijoiden oli tarkoitus toteuttaa omana moduulina valmiiksi annettuun projektipohjaan. Kappaleessa 5 esitellään suoritetun palautekyselyn tulokset, jotka kerättiin opiskelijoilta viimeisellä harjoitustyön ohjauskerralla. Kyselytuloksiin saatiin vastaus 15 opiskelijalta. Suoritetun palautekyselyn tuloksista kirjoitettiin konferenssiartikkeli (Karhu, Alander & Nurmi 2015), joka esiteltiin CSEDU-konferenssissa. Konferenssiartikkeli on liitteessä 2. Kappaleessa 6 pohditaan kurssin toteutusta yleisesti ja vastausten perusteella, sekä vertaillaan kirjallisuudesta löytyviin muihin toteutustapoihin. Viimeisessä kappaleessa suoritetaan työn yhteenveto.

2. FPGA

FPGA-piirit ovat digitaalisia logiikkapiirejä, jotka kuuluvat kenttäohjelmoitaviin logiikoihin (FPL). Kenttäohjelmoitavat logiikat ovat ohjelmoitavia laitteita, jotka sisältävät pieniä logiikkasoluja ja elementtejä. (Mayer-Baese 2007: 3.) FPGA-piireillä pystytään toteuttamaan monimutkaisia logiikkapiirejä, sekä suorittamaan vaativia laskutoimituksia. FPGA-piirien etuna on niiden uudelleenohjelmoitavuus, jolloin käyttäjä pystyy suunnittelemaan erilaisia toteutuksia tai päivittämään jo toimivaa sovellusta. FPGA-piiri sisältää loogisia portteja, joita voi yhdellä piirillä olla jopa miljoonia. Nämä loogiset portit ovat elementtejä, jotka voidaan konfiguroida eri tavoin. Näitä yhdistelemällä voidaan toteuttaa erilaisia loogisia toimintoja. (Lötjönen 2012: 4–5.)

2.1. Arkkitehtuuri

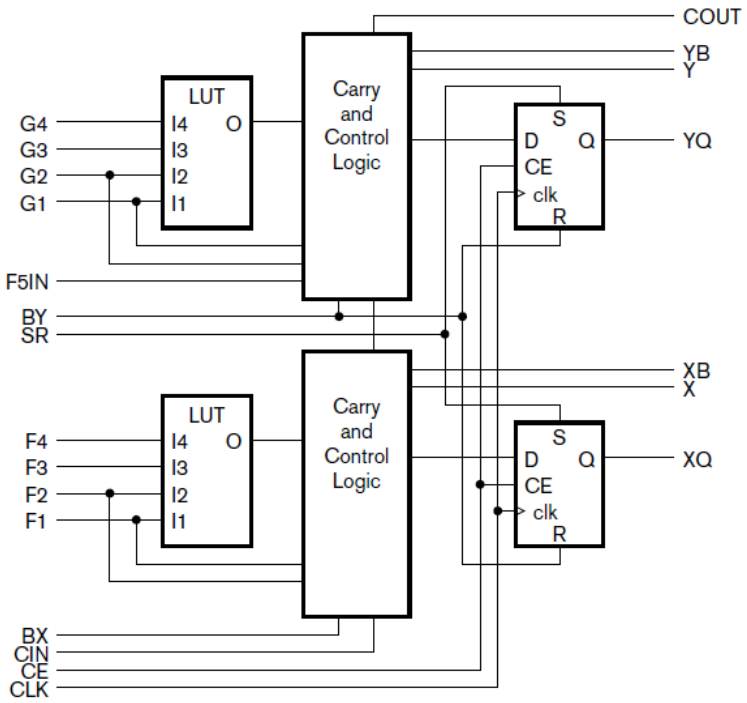
FPGA-piirin arkkitehtuuri koostuu ohjelmoitavista logiikkamoduuleista, jotka on järjestetty matriisiksi (Kuva 1). Logiikkamoduulit on yhdistetty toisiinsa kytkentäjohdoilla. (Kuon, Tessier & Rose 2007: 138.) Logiikkamoduulien ja liityntöjen lisäksi piiri sisältää I/O-moduuleja, joita on matriisirakenteen ympärillä, sekä kytkin- ja muistielementtejä. Eri piirivalmistajat nimeävät ja toteuttavat I/O- ja logiikkamoduulit eri tavalla, toiminnan ollessa kuitenkin samantapainen. (Lötjönen 2013: 5–6.) Logiikkamoduulit voivat olla toiminnaltaan erilaisia, osa voi toimia kertolaskumoduulina, osa muistimoduulina (Kuon ym. 2007: 138).



Kuva 1. FPGA-piirin arkkitehtuuri koostuu matriisirakenteena olevista logiikkamoduuleista, kuten yleislogiikka-, muisti- ja laskumoduuleista. Moduulit on yhdistetty ohjelmoitavalla reitityksellä. Matriisirakenteen ympärillä on I/O-moduulit. (Kuon ym. 2007: 138).

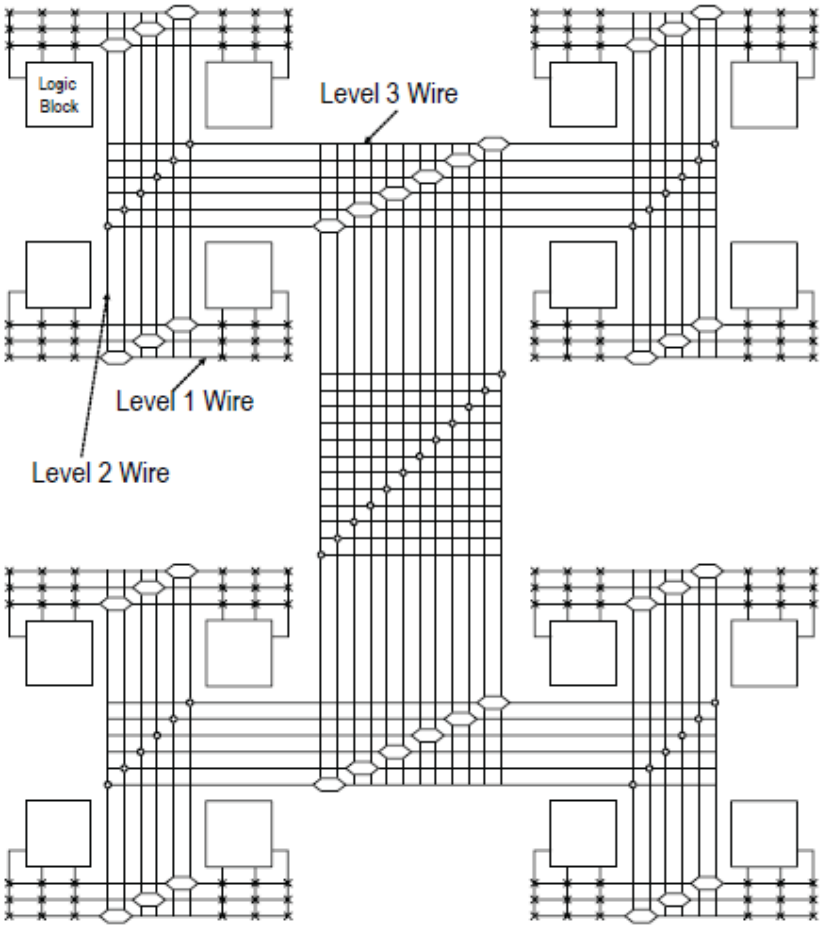
Logiikkamoduulien tarkoituksena on toimia laskenta- ja muistiyksikköinä, joita tarvitaan digitaalisissa järjestelmissä. Tarvittavien moduulien määrä pienenee niiden toiminnallisuuden kasvaessa. Toiminnallisuuden kasvu tarkoittaa samalla moduulin koon kasvamista (Kuon, ym. 2007: 151, 156).

Logiikkamoduulit koostuvat pienistä elektronisista komponenteista, kuten kiikuista, yhteenlaskimista, hakutaulukoista (LUT) ja multipleksereista (Ashenden 2008:265, Löttjönen 2013:6). Logiikkamoduuli on peruslogiikkayksikkö FPGA-piireillä, sen toiminta vaihtelee laitteiden ja valmistajien välillä. Xilinxin (2013) mukaan logiikkalohkossa on konfiguroitava kytkinmatriisi 4-6 sisääntulolla, kiikkuja ja multipleksereita. Kuvassa 2 on kuvattu osa Xilinxin Spartan-II logiikkamoduulia.



Kuva 2. Osa Xilinxin Spartan-II logiikkamoduulista. Moduuli sisältää neljällä sisääntulolla toimivan hakutaulukon (LUT), joka voidaan ohjelmoida toteuttamaan mikä tahansa sisääntulojen mahdollistama looginen funktio. Carry and Control Logic koostuu piiristä, joka liittää hakutaulukon ulostulon XOR ja AND-veräjään, jolloin voidaan muodostaa yhteenlasku- ja kertolaskupiirejä. S-moduulit ovat kiikkuja. (Ashenden 2008: 265)

Ohjelmoitavien liityntöjen avulla logiikkamoduulit, I/O-moduulit ja kytkimet voidaan yhdistää halutunlaiseksi piirikokonaisuudeksi. Liityntäarkkitehtuureja on kahdenlaisia, hierarkkisia ja saareke-tyyppisiä. Hierarkkisessa liityntäarkkitehtuurissa (Kuva 3) logiikkamoduulit on eroteltu erillisiksi ryhmiksi. Ryhmien väliset liitynnät voidaan toteuttaa johdinsegmenttien avulla liityntäarkkitehtuurin alimmalla tasolla. Etäällä toisistaan olevien moduulien yhdistämiseksi tarvitaan yksi tai useampia poikittaisia liityntäsegmenttejä. (Kuon ym. 2007: 176–177.)

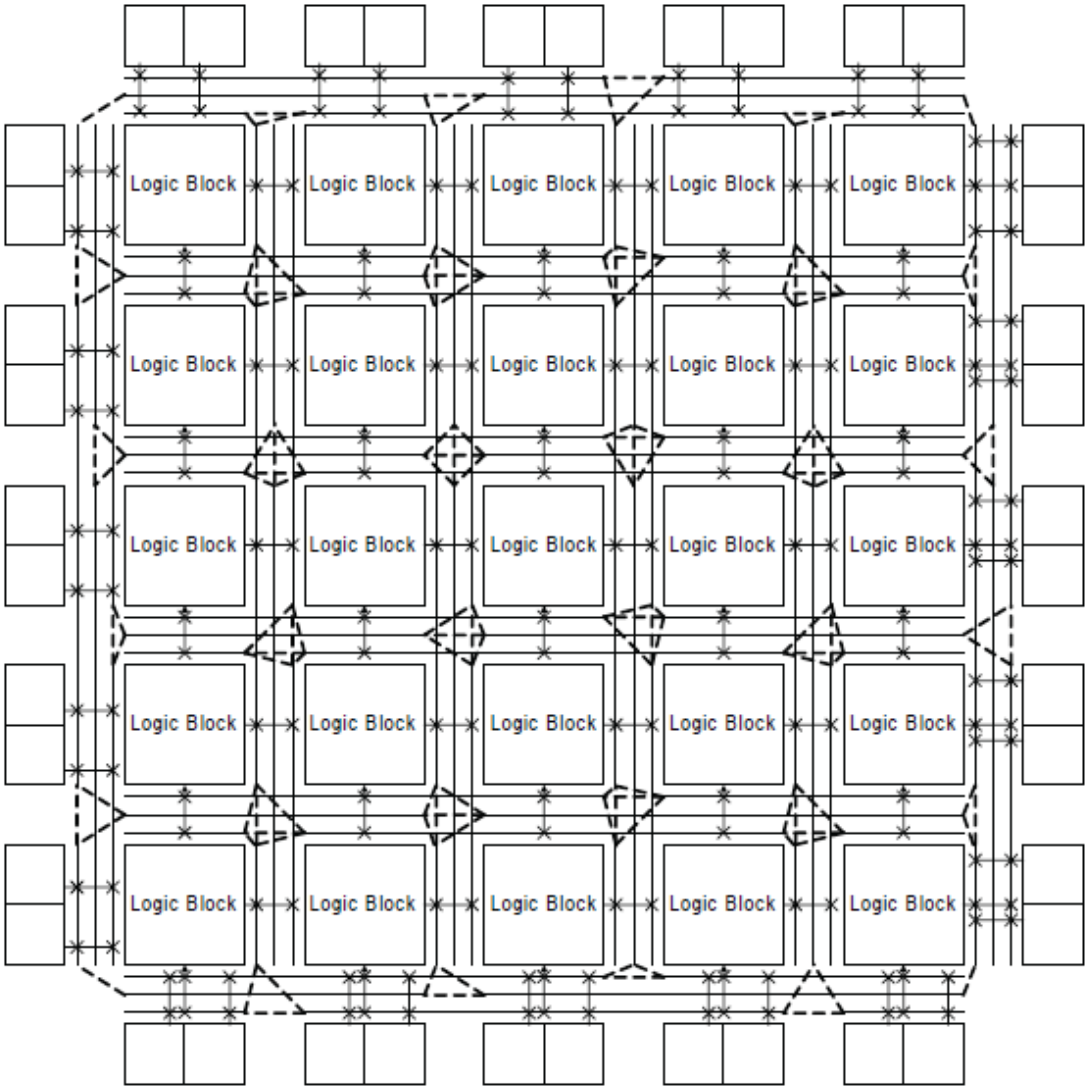


Kuva 3. Hierarkkinen liityntäarkkitehtuuri, jossa logiikkamoduulit ovat erillään olevissa ryhmissä. Ryhmät on yhdistetty johdinsegmenttien avulla. (Kuon ym. 2007: 177–178)

Kuvassa 3 tasolla yksi (Level 1) olevat logiikkamoduulit yhdistyvät suoraan toistensa kanssa. Ohjelmoitavia liityntöjä merkitään risteillä ja ympyröillä. Ristit ovat yhteyslohkoja, joiden avulla logiikkamoduulin voidaan kytkeä johdinsegmentti pysty- tai vaakasuorasti. Ympyrät on kytkinlohkoja joilla kytketään vaaka- ja pystysuorat johdinsegmentit niiden risteyskohdassa. Yhteys- ja kytkinlohkojen joustavalla käytöllä saavutetaan tehokas reititys. (Rose 1991: 277.)

Saareke-tyyppisessä liityntäarkkitehtuurissa (Kuva 4) logiikkamoduulit on järjestetty kaksiulotteiseksi verkoksi, jossa liitynnät on tasaisesti jaettu. Jokaisessa logiikkamoduulissa on kaikilla sivuilla liityntä. Tällä rakenteella pystytään muodostamaan erilaisia tehokkaita liityntöjä, koska liityntäjohdot ovat logiikkamoduulien läheisyydessä. Kyt-

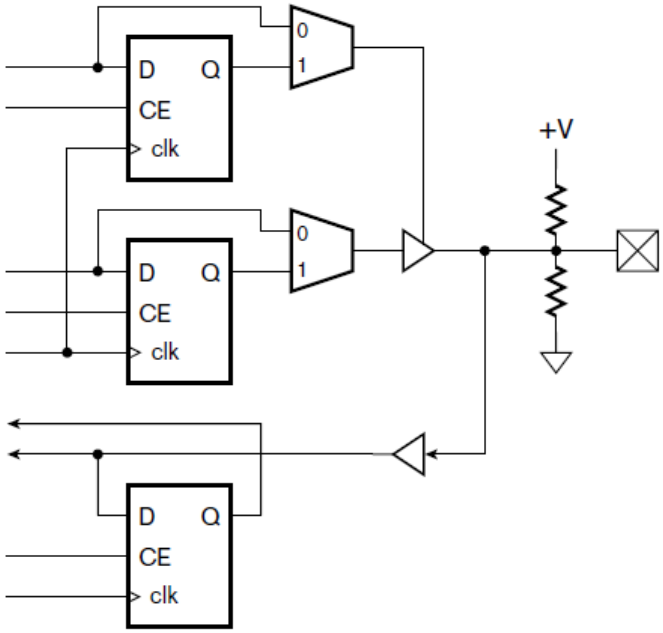
kinmoduulit muodostavat liitynnät liityntäjohtojen välillä jokaisessa vaaka- ja pystykanavan leikkauskohdassa. (Kuon ym. 2007: 180–181.)



Kuva 4. Saareke-tyyppinen liityntäarkkitehtuuri. Jokaisessa logiikkamoduulissa on reitityskanavat jokaisella neljällä puolella logiikkamoduulissa. Kolmiot ovat ohjelmoitavia reitityskytkimiä, jotka muodostavat yhteyden vaaka- ja pystysuuntaisten johdinsegmenttien risteyskohdassa. (Kuon ym. 2007: 179–181)

FPGA-piirin kommunikointi ulkoisten komponenttien kanssa tapahtuu I/O-moduulien kautta. I/O-moduulit määrittelevät kuinka paljon kommunikaatiota voi tapahtua ulkoisten komponenttien kanssa. I/O-moduulit vievät ison osan FPGA-piirin pinta-alasta. I/O-moduulit sisältävät komponentteja, jotka avustavat kommunikoinnissa muiden laitteiden

kanssa, kuten rekistereitä, puskureita sekä jännitteen kääntäjiä. (Kuon ym.2007: 55; Smith 2010: 44–45; Naser 2009: 40–41.) Kuvassa 5 on esitetty tyypillinen FPGA-piirin I/O-moduuli.



Kuva 5. FPGA-piirin I/O-moduuli. Multipleksereiden sisääntulot ohjelmoidaan säätämään ulostulot kombinaatio- tai sekvenssilogiikaksi. Ylimpänä olevat kiikku ja multiplekseri säätävät korkeaimpedanssista tilaa kolmitilaohjaimessa joka ohjaa nastan ulostuloksi. Keskimmäiset ohjaavat ulostulon arvoa. Alimpana oleva ulostuloajuri on ohjelmoitava erilaisille logiikkatasoille. Ylös- ja alasvetovastukset voidaan ohjelmoida halutulle vastustasolle. (Ashenden 2008: 266)

FPGA-piiri ei suorita toimintoja eikä ohjelmakoodia samalla tavalla kuin mikroprosessori, vaan toiminnan määrittelee muuttumaton staattinen konfiguraatio. FPGA:n toiminta perustuu massiiviseen rinnakkaisuuteen. Rinnakkaisessa toteutuksessa jokainen logiikkalohko toimii itsenäisesti käyttäen omia resurssejaan. Piirin kapasiteetti riippuu sen logiikkamoduulien määrästä, eli mitä enemmän logiikkamoduuleja sitä suurempi kapasiteetti. (Ranta 2012: 13.)

2.2. Suunnittelu

FPGA-suunnittelu voidaan jakaa 15:oista eri vaiheeseen (U.S.NRC 2009: 46–52), Qasim, Abbasi ja Almashary (2009: 313) esittävät kaavion FPGA suunnittelumetodologiasta kuvassa 6:

- suunnittelumääritelmät
- arkkitehtuurin suunnittelu
- yksityiskohtainen suunnittelu
- suunnitelman läpikäynti
- käyttäytymiskuvaus
- käyttäytymissimulointi
- logiikkasynteesi
- logiikkatason simulointi
- paikoitus ja reititys
- laitteiston toteutus
- rakenteen simulointi
- prototyypitys
- verifiointi
- laitteiston toteutus
- funktionaalinen laitteiston verifiointi.

Suunnittelumääritelmissä kuvataan FPGA-piirin haluttu toiminnallisuus. Päätarkoituksena on kuvata toiminta, mutta tiedot tehonkulutuksesta, kriittisistä ajastuksista ja piirilevyn koosta voidaan lisätä mukaan. Qasim ym. (2009: 313–314) mukaan ensimmäisessä vaiheessa (Kuva 6) kuvataan haluttu algoritmi käyttäen laitteistokuvauskieltä tai kytkentäkaaviota riippuen toteutuksen monimutkaisuudesta.

Arkkitehtuurin suunnittelussa toiminnallisuus jaetaan pienempiin osamoduuleihin. Näiden osamoduulien kuvaukset ja vuorovaikutukset määritellään. Lisäksi mahdolliset lisäominaisuudet kuten redundanssi ja virheiden korjaus suunnitellaan tässä vaiheessa.

Yksityiskohtaisessa suunnittelussa arkkitehtuurin suunnittelussa määritellyt moduulit kuvataan yksityiskohtaisesti. Asiat, jotka liittyvät esimerkiksi kellon nopeuteen, piirin liityntöihin, johdotuksiin ja muisteihin kuvataan tarkemmin.

Suunnitelman läpikäynnissä kuvataan arkkitehtuurin ja yksityiskohtaisen suunnittelun vaiheet teksti- tai kuvamuodossa. Läpikäynnin tuloksena voi tulla korjauksia alkuperäiseen suunnitelmaan.

Käyttäytymiskuvauksessa toiminta kuvataan käyttäen laitteistokuvauskieltä (HDL). Käyttäytyminen voidaan kuvata myös kytkentäkaavion avulla. Kytkentäkaavion tekemistä suositellaan turvallisuuskriittisissä sovelluksissa. Kytkentäkaavion käyttäminen on hitaampaa kuin laitteistokuvauskielen käyttäminen. Käyttäytyminen voidaan kuvata myös käyttäen korkeamman tason ohjelmointikieliä kuten C, Matlab tai LabVIEW, jonka jälkeen kuvaus käännetään laitteistokuvauskieleksi koodigeneraattorin avulla.

Käyttäytymissimuloinnissa käytetään standardeja simulointityökaluja ja se perustuu pelkästään tekstimuotoiseen tai graafiseen kuvaukseen suunnittelumääritelmistä. Syötevektorit ja odotetut vastevektorit tulisi määritellä suunnittelumääritelmien mukaisesti. Hierarkkisessa mallissa, joka sisältää useita lohkoja tulisi jokainen lohko simuloida erikseen mahdollisimman kattavan simulointituloksen saamiseksi. Lohkotason simuloinnin jälkeen voidaan suorittaa päätason (top-level) simulointi varmistamaan lohkojen välinen yhdistyneisyys.

Logiikkasynteesissä korkean tason kuvaus muutetaan logiikkakaavioksi ja netlistiksi. Groutin (2008: 341) mukaan logiikkasynteesissä laitteistokuvauskielinen kuvaus käännetään piirin netlistiksi. Netlist tarkoittaa piirin loogisten porttien ja niiden välisten kytkentöjen kuvausta. Syntetisointi suoritetaan määrittelyjen jälkeen, jolloin suunnittelijan tulee vahvistaa suunnitelman toimivuus. Tämä suoritetaan funktionaalisella tai käyttäytymissimuloinnilla (Kuva 6).

Logiikkatason simulointia käytetään varmistamaan alkuperäisten suunnittelumääritelmien oikeellisuus syntetisoinnissa. Käyttäytymissimulointi on teknologia riippumatonta,

mutta logiikkatason simulointi käyttää teknologiasta riippuvaisia elementtejä, jotka ovat FPGA kohtaisia.

Paikoitusta ja reititystä kutsutaan myös fyysiseksi implementoinniksi. Tässä osiossa syntetisoitu keskitason netlist käännetään alemman tason netlistiksi sekä bittitijonoksi. Näitä kahta voidaan tämän jälkeen käyttää konfiguroinnissa. Tässä vaiheessa netlist käännetään loogisista porteista sisääntulo- ja ulostulolohkoiksi kohde-FPGA arkkitehtuurissa (Technology Mapping). Tässä kohdassa valitaan jokaiselle lohkolle optimaalinen paikka piiriltä siten, että lohkojen välinen reititys saadaan minimaaliseksi (Placement). Reititys on laskennallisesti vaativa, koska siinä käytetään ainoastaan valmiita johdinsegmenttejä, ohjelmoitavia kytkimiä ja multipleksereitä. (Kuva 6).

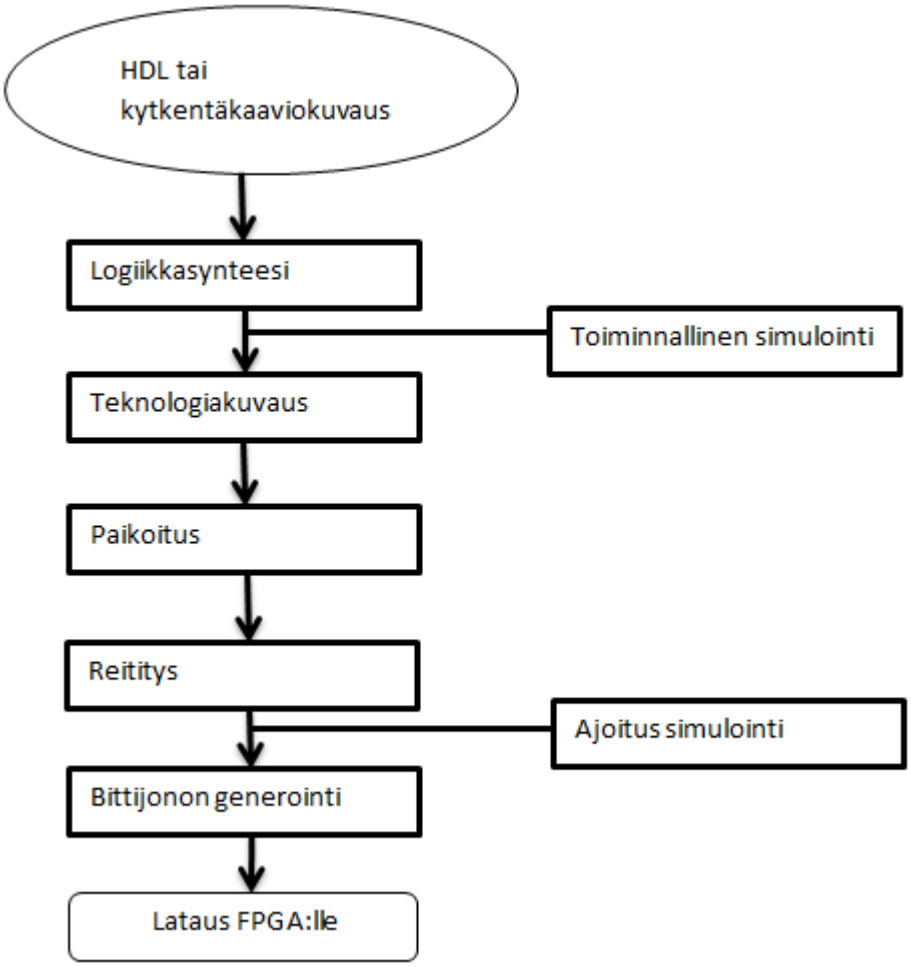
Paikoituksen ja reitityksen jälkeen suoritetaan ajoitussimulointi (Kuva 6). Sen tarkoituksena on varmistaa suunnitelman looginen virheettömyys ottaen huomioon FPGA-systeemin viipeet. Ennen viimeistä suunnitteluvaihetta toteutetaan bittijono (Kuva 6). Siinä kuvauksen sekä paikoituksen ja reitityksen toteutukset ovat sisääntuloja, joista generoidaan bittijono, jolla ohjelmoidaan haluttu logiikka ja yhteydet kohdelaitteessa. Rakenteen simulointi tulisi suorittaa käyttäen samoja sisäänmenoja kuin aikaisemmissa simuloinneissa. Simuloinnin tuloksia tulisi verrata aikaisempien kohtien simulointituloksiin.

Prototyypityksen tarkoituksena on suorittaa suunnitellun laitteiston verifiointi aiemmin suoritetun ohjelmallisen verifioinnin lisäksi. Prototyypitystä tarvitaan varmistamaan, että rakenteen simuloinnissa generoitu konfigurointitiedosto toteuttaa halutun toiminnallisuuden. Prototyypitykseen kuuluu lisäksi alustan suunnittelu, jossa FPGA-piiri laitetaan liityntäkohtaan johon se valmiissa alustassa tulisi.

Verifiointi tulisi suorittaa laitteistolla generoidulla sisääntuloilla, jotka ovat identtisiä ohjelmallisessa simuloinnissa käytettyjen sisääntulojen kanssa. Vertailemalla laitteistolla suoritetun simuloinnin tuloksia ohjelmapohjaisiin simulointituloksiin voidaan varmistua, että alkuperäiset suunnittelumääritelmät täyttyvät moduulitasolla sekä päätasolla.

Viimeisenä suunnitteluvaiheena on itse laitteiston toteutus. Laitteiston toteutuksessa suunnitellaan liitynnät ja lopullinen piiri FPGA:n käyttöön (Ranta 2012: 32). Lisäksi tulisi toteuttaa dokumentointi, joka tulisi sisältää piirin kytkentäkaavio, materiaalien hinnat ja alustalla olevien komponenttien datalehdet.

Funktionaalinen laitteiston verifiointi tulisi suorittaa varmistamaan alustan oikeanlainen toiminnallisuus operatiivisessa käytössä. Tässä verifiointi suoritetaan koko FPGA-alustalle käyttäen sisääntulokombinaatioita, jotka ovat mahdollisia tuotanto käytössä. Tässä osassa tulee ottaa myös huomioon muut alustalla olevat komponentit ja niiden toiminta, kuten ulkopuoliset kellolähteet ja sisään- ja ulostulojen logiikkatasot.



Kuva 6. FPGA:n suunnittelumetodologia. (Perustuu lähteeseen Qasim, Abbasi & Almashary 2009: 313.)

2.3. FPGA:n edut

FPGA:n rinnakkaisuudesta johtuen sen suorituskyky on sille sopivissa tehtävissä usein huomattavasti parempi kuin esimerkiksi DSP:n tai mikroprosessorin. I/O-moduulien hallinta laitteistotasolla tarjoaa enemmän prosessointitehoa ja sopii paremmin vaatimuksiin. ASIC-piireihin verrattuna FPGA:n valmistusajat ovat lyhyemmät. ASIC-piirit toteutetaan aina sovelluskohtaisiksi, jolloin niiden testauskin vie enemmän aikaa, eikä niiden toiminnallisuutta ole mahdollista muuttaa. FPGA:n toiminnallisuuden muuttaminen vie muutamia tunteja, kun taas ASIC-piirit joudutaan toteuttamaan uudelleen.

Suurien eräkokojen valmistaminen ASIC-piireillä on halpaa, mutta yksittäisen piirin hyvin kallista. Muutosten tekeminen ASIC-piireihin ei ole mahdollista, koska ne ovat kiinteitä, ei ohjelmoitavia. Tämän vuoksi piirit joudutaan suunnittelemaan ja valmistamaan uudelleen, joka on kallista ja aikavievää. FPGA-piireillä korjausten ja virheiden takia muutosten tekeminen tapahtuu ohjelmallisesti. DSP systeemin toteuttamiseen tarvitaan ylimääräisiä mikropiirejä toteuttamaan toiminnallisuutta, joka vie tilaa ja kuluttaa tehoa. Toteuttamalla DSP kiinteästi samalla piirillä FPGA:n kanssa saadaan toteutettua korkean suoritustason signaalinkäsittelyä (Skliarova, Sklyarov, Sudnitson & Kruus 2013: 479).

FPGA-piirit ovat toimintavarmempia kuin mikroprosessorit. Mikroprosessorit toimivat peräkkäisesti käsky kerrallaan. Mikroprosessoreilla usean ohjelman ajaminen rinnakkaisesti voi johtaa ongelmiin säikeiden välisistä ajoituksista johtuen. Ohjelmien suoritukseen tarvitaan käyttöjärjestelmä. FPGA:t toimivat rinnakkaisesti, eikä niissä ole käyttöjärjestelmää. FPGA:n suoritus voidaan jakaa pienempiin modulaarisiin rinnakkaisiin osiin, mikä lisää toimintavarmuutta. On mahdollista toteuttaa mikroprosessori FGPA-piirille laitteisto- tai ohjelmistopohjaisesti lisälogiikan lisäksi (Skliarova ym. 2013: 479).

FPGA-piirien pitkäaikainen ylläpidettävyys on helpompaa kuin muilla ratkaisuilla. Tarvittavat muutokset pystytään tekemään tarvittaessa myös käyttökohteessa. Myös loppukäyttäjä pystyy periaatteessa suunnittelemaan uuden toteutuksen ja siirtämään sen

FPGA-piirille. Tämä on joustavampaa ja tehokkaampaa kuin uuden piirin valmistaminen. (National Instruments 2012)

2.4. Laitteistokuvauskielet

Laitteistokuvauskielellä suunnitteluun käytetään tekstikuvaus digitaalisesta systeemistä tai logiikkapiiristä. Laitteistokuvauskielellä piiristä voidaan tehdä kuvaus eri abstraktiotasoilla. Korkeimmalla abstraktiotasolla kuvataan systeemin spesifikaatiot, sekä tekninen kuvaus algoritmeilla. Laitteistossa algoritmien rakenne kuvataan systeemin arkkitehtuurissa, jossa identifioidaan korkean tason funktionaaliset lohkot ja toiminnallisuuden yhdentyminen. Algoritmit kuvaavat systeemin käyttäytymistä. Arkkitehtuurin alapuolisella abstraktiotasolla on rekisteritaso (RTL), jossa kuvataan datan siirtyminen, tallennus ja suoritettavat loogiset operaatiot. Tätä tasoa käytetään yleensä syntetisoinnissa, jossa kuvataan piirin netlist, joka tarkoittaa piirin loogisten veräjien ja niiden välisten kytkentöjen kuvausta. Alimmalla abstraktiotasolla loogiset veräjät toteutetaan transistoreina. (Grout 2008: 193.)

2.4.1. VHDL

VHDL-laitteistokuvauskielen kehitys alkoi 1980-luvulla Yhdysvaltojen puolustusministeriön toimesta. Vaatimuksena oli digitaalipiirien suunnittelu yleisiä suunnittelumenetelmiä käyttäen, dokumentoinnin mahdollisuus sekä uudelleenkäyttö uusien tekniikoiden kanssa. VHDL:stä tuli IEEE-standardi (IEEE Std 1076–1987) vuonna 1987, joka oli myös ensimmäinen laitteistokuvauskielelle myönnetty standardi (Pedroni 2008: 492). Kieli tarkistettiin vuosina 1993, 2000, 2002 ja 2008. Viimeisin IEEE-standardi on 1076–2008. (IEEE Std 1076–2008.)

VHDL on valmistajasta ja teknologiasta riippumaton laitteistokuvauskieli, jolla kuvataan elektroniikkapiirin rakennetta tai käyttäytymistä. Sitä voidaan käyttää sekä piirin spesifiointiin, suunniteluun, syntetisoimiseen että simulointiin. (Pedroni 2008: 492.)

VHDL-kielinen ohjelma koostuu kolmesta perusrakenteesta, jotka ovat kirjaston esittely, entiteetin kuvaus sekä arkkitehtuurin toiminnan kuvaus (Kuva 7). Kirjaston esittelyssä määritellään tarvittavat kirjastot ja kokoelmat joita käyttäjä haluaa käyttää ohjelman käsittelyssä ja kääntämisessä. Peruskirjastossa (std) on määritelty perustietotyypit, joita ovat muun muassa bitti, boolean, kokonaisluku ja bittivektori. Kirjaston esittelyn jälkeen tulee esitellä tarvittavat kokoelmat, joita tarvitaan. Peruskokoelma VHDL:ssä on std_logic_1164. (IEEE Std 1076–2008: 514.)

```
library IEEE;                          --kirjastojen määrittely
use IEEE.std_logic_1164.all;

entity esimerkki is                    --entiteetin kuvaus
      generic();                       --geneeriset parametrit
      port();                          --sisään/ulostulot
end entity esimerkki;

                                       --arkkitehtuurin kuvaus
architecture arc1 of esimerkki is

      begin                            --aloitetaan suoritus

                                       --arkkitehtuurin toiminta

end architecture arc1                  --arkkitehtuurin lopetus;
```

Kuva 7. Kirjaston, entiteetin ja arkkitehtuurin määrittely. Entity sanan jälkeen tulee piirin nimi, joka on tässä tapauksessa esimerkki, jota tulee käyttää myös arkkitehtuurin kuvauksessa.

Entiteetin kuvauksessa kohdassa port (Kuva 7) kerrotaan piirin sisään- ja ulostulot, sekä niiden tilat ja tietotyypit. Mahdollisia tiloja ovat in, out, inout ja buffer. In-tilassa pystytään ainoastaan vastaanottamaan tietoa, out-tilassa tietoa pystytään ainoastaan lähettämään, inout-tilassa pystytään sekä vastaanottamaan että lähettämään tietoa ja buffer-tilassa pystytään vastaanottamaan ja lähettämään sekä sitä voidaan käyttää sisäisessä tiedonlähetyksessä (Grout 2008: 347). Sisään ja ulostulojen tietotyyppejä voivat olla muun muassa bit, bit_vector, integer, std_logic ja std_logic_vector. Entiteetissä voidaan lisäksi määritellä geneerisiä parametreja, aliohjelmien runkoja sekä samanaikaisia suorituksia. Geneerisillä parametreilla tarkoitetaan vakioita, joita voidaan käyttää useammassa paikassa samassa koodissa. Geneeriset parametrit määritellään ennen sisään- ja ulostuloja kohdassa generic (Kuva 7).

Arkkitehtuurin toiminnan kuvauksessa (Kuva 7) kuvataan piirin sisäinen toiminta, eli miten sisääntulot vaikuttavat ulostuloon. Arkkitehtuuri voidaan toteuttaa kolmella eri tavalla, jotka ovat rakenteellinen (structural), tietovuo (dataflow) ja käyttäytyminen (behavioural) (IEEE Std 1076–2008: 12). Rakenteellisessa mallissa piirin rakenne kuvataan loogisten porttien ja niiden välisten liityntöjen muodostamana kokonaisuutena. Tietovuomallissa kuvataan tiedon siirtyminen signaalien välillä ja sisääntuloista ulostuloihin. Käyttäytymismallissa kuvataan toiminta piirin kannalta ja käyttäytyminen algoritmien avulla. Tietovuo- ja käyttäytymismallin erona on process-lause, jota käytetään käyttäytymismallissa.

VHDL:ssä ohjelman suoritus tapahtuu pääsääntöisesti rinnakkaisesti. Tämä on hyvä asia, kun halutaan toteuttaa kombinaatiopiirejä, mutta sekvenssipiirejä toteutettaessa tarvitaan myös peräkkäistä suoritusta. VHDL:ssä peräkkäinen suoritus voidaan toteuttaa prosessilla (process) tai proseduureilla (procedure) sekä funktioilla (function). Funktiota ja proseduuria kutsutaan IEEE 1076-standardin mukaan aliohjelmiksi. Aliohjelmien suoritus tapahtuu rinnakkaisesti mahdollisten muiden aliohjelmien kanssa mitä ohjelma sisältää. Peräkkäiset suoritukset, jotka toimivat edellä mainittujen toimintojen sisällä voivat sisältää case-lauseita, if-then-else-lauseita tai silmukoita kuten while ja for-silmukat. (IEEE Std 1076–2008: 19–20, 29–30.)

```
--arkkitehtuurin kuvaus
architecture arc1 of esimerkki is

    begin                --aloitetaan arkkitehtuurin suoritus

    process()            --sulkuihin herkkyyslista
        variable         --paikallisten muuttujien määrittely

        begin            --aloitetaan prosessin suoritus

    end process;         --lopetetaan prosessin suoritus

end architecture arc1;
```

Kuva 8. Process-lohkon määrittely arkkitehtuurin. Process-lauseessa sulkujen sisälle määritellään muuttujat, jotka vaikuttavat sen suoritukseen. Lohkon toiminta tapahtuu begin ja end process sanojen välissä.

Prosessi-lauseessa (Kuva 8) määritellään ensimmäiseksi herkkyyslista (sulkujen sisälle). Prosessin suoritus käynnistyy joka kerta, kun jonkin herkkyyslistassa olevan signaalin tila muuttuu. Paikallisten muuttujien määrittely on mahdollista heti herkkyyslistan jälkeen. Paikallisten muuttujien tulee olla variable-tyyppisiä. (IEEE Std 1076–2008: 170–172.)

Funktioiden toteutus suoritetaan yleensä kokoelmissa (package) tai kirjastoissa. Tämä helpottaa niiden uudelleenkäytettävyyttä ja jakamista. Funktiossa ei voi määritellä paikallisia muuttujia tai signaaleja. Funktiot pystyvät palauttamaan ainoastaan yksittäisiä arvoja. Parametrit voivat olla ainoastaan vakioita tai signaaleja. Funktion kutsu voi tapahtua aliohjelmista tai varsinaisesta ohjelmasta (Kuva 10). (IEEE Std 1076–2008: 19–20.) Kuvassa 9 on esimerkki kuinka funktio voidaan määritellä kokoelmassa.

```
--kokoelman määrittely
package my_package is
--funktion esittely
      function shift_integer(signal a, b: integer) return integer;
end my_package;

--kokoelman rungon määrittely
package body my_package is

function shift_integer(signal a, b: integer) return integer is

      begin                        --funktion toiminnan aloitus

            return a*(2**b);       --palautettava arvo

      end shift_integer;           --funktion toiminnan lopetus

end my_package;
```

Kuva 9. Funktion määrittely kokoelmassa. Kokoelma määritellään sanalla package, jonka jälkeen tulee sen nimi my_package. Funktion määrittelyssä määritellään funktio ja sen parametrit sekä paluutyyppi. Kokoelman rungossa package body toteutetaan määritellyn funktion toiminta. (Pedroni 2008: 514)

```
library IEEE;                              --kirjastojen määrittely
use IEEE.std_logic_1164.all;
use work.my_package.all;                   --kokoelma, jossa funktio on

entity shifter is
      port(  input : in integer range 0 to 63;
             shift  : in integer range -6 to 6;
             output : out integer range 0 to 63);
end entity shifter;

architecture shifter of shifter is         –arkkitehtuurin määrittely

      begin                                --aloitetaan suoritus


      output<=shif_integer(input, shift);  --funktion kutsu

end architecture shifter;
```

Kuva 10. Kokoelmassa määritellyn funktion käyttö. Kokoelma otetaan käyttöön esittelemällä se samaan tapaan kuin käytettävät kirjastot (use work.my_package.all). Funktion käyttö tapahtuu arkkitehtuurissa, jossa sitä kutsutaan funktion nimellä tarvittavien parametrien kanssa. (Pedroni 2008: 514)

Proseduurin rakenne ja käyttö on samantapainen kuin funktiolla. Proseduuri voi kuitenkin palauttaa useamman kuin yhden arvon. Parametreina voidaan käyttää vakioita, muuttujia tai signaaleita. Proseduurin kutsuminen on oma lause. (IEEE Std 1076–2008: 19–20.)

2.4.2. Verilog

Verilog on VHDL:n ohella toinen suosittu laitteistokuvauskieli. Gateway Design Automation kehitti sen 1980-luvun alkupuolella. Verilogista tuli IEEE-standardi vuonna 1995 (1364–1995) ja se uusittiin vuonna 2001 (1363–2001). Verilog muistuttaa monessa suhteessa C-ohjelmointikieltä. Kieli on heikosti tyypitetty. Tämän vuoksi sijoituslauseet, jotka toimivat Verilogissa eivät välttämättä toimi VHDL:ssä. Verilogin avulla loogisia portteja voidaan kuvata transistoritasolla, tämän vuoksi sen sanotaan olevan lähempänä laitteistoa kuin VHDL. (Zwolinski 2004: 327.)

Verilogia käytetään kuvaamaan järjestelmän laitteistomoduuleja tai kokonaisia järjestelmiä (Navabi 2006: 18). Toisin kuin VHDL:ssä Verilogissa ei ole erikseen kuvausta

entiteetille ja arkkitehtuurille, eikä myöskään kirjastojen määrittelyä. Verilogissa sisään- ja ulostulot sekä toiminta kirjoitetaan yhden moduulin sisään. Moduuli määritellään avainsanojen module ja endmodule väliin. Module avainsanan jälkeen määritellään moduulin nimi, jonka jälkeen suluissa määritellään signaalien parametrit (Kuva 11). (IEEE Std 1364–2001 2001: 166.)

Moduulin rungossa määritellään signaalien tilat, jotka voivat olla input, output tai inout (Kuva 11). Signaaleilla ei ole VHDL:n tapaan tietotyyppejä, vaan niiden neljä mahdollista arvoa ovat 0, 1, Z tai X. Arvo 0 kuvaa epätosia tiloja tai loogista nollatilaa, arvon 1 ollessa sen komplementti. Tuntemattomat loogiset arvot ovat X, ja korkeaimpedanssinen tila Z. (IEEE Std 1364–2001 2001: 20.) Tietotyyppejä Verilogissa ovat net ja reg, jotka kuvaavat signaaleja ja muuttujia. Net- tietotyyppi kuvaa tiedon kuljettajia, kuten veräjien ulostuloja, puskureita ja liityntäsignaaleja. Reg- tietotyyppi määrittää muuttujat, jotka voivat toimia rekistereinä, ennen kuin ne ylikirjoitetaan. Molemmat edellä mainitut tietotyypit voidaan määritellä etumerkillisiksi avainsanalla signed. (IEEE Std 1364–2001 2001: 22–23.)

```
/*moduulin esittely*/
module And_gate(a,b,c);

   input a;
   input b;                    /*sisään ja ulostulojen määrittely*/
  output c;

   assign c = a & b;           /*moduulin toiminta*/

endmodule                      /*moduulin lopetus*/
```

Kuva 11. Moduulin määrittely, esimerkkinä ja-veräjä (And_gate). Sulkeissa olevat sisään- ja ulostulomuuttujat esitellään ensimmäisenä. Toiminta määritellään heti muuttujien esittelyn jälkeen. Jatkuva-aikainen sijoitus toteutetaan sanalla assign.

Moduulin toiminta kuvataan signaalien tilojen määrittelyn jälkeen. Tämä vastaa VHDL:n arkkitehtuurin kuvausta. Toiminnan kuvaaminen voidaan suorittaa samalla tavalla kuin VHDL:ssä. Jatkuva-aikaiset sijoitukset suoritetaan avainsanalla assign (Kuva 11). Verilogissa on lisäksi blokkaava (blocking) ja ei-blokkaava (non-blocking) sijoitus (Kuva 12). Blokkaavassa sijoituksessa sijoitus tapahtuu välittömästi, ei-blokkaavassa suoritusajan päätyttyä. (IEEE Std 1364–2001 2001: 69–71.) Standardin

mukaan blokkaavaa sijoitusta tulisi käyttää kombinatorisessa logiikassa ja ei-blokkaavaa sekvenssipiireissä.

```
c = a & b;    /*blokkaava sijoitus*/
c <= a & b;  /*ei-blokkaava sijoitus*/
```

Kuva 12. Blokkaava ja ei-blokkaava sijoitus. Blokkaavassa sijoituksessa muuttuja saa välittömästi arvon ja ei-blokkaavassa suoritusajan päätyttyä. Kummassakin yllä olevassa esimerkissä suoritetaan looginen ja-operaatio.

Peräkkäinen suoritus Verilogilla suoritetaan initial- tai always-lohkoissa (IEEE Std 1364–2001 2001: 148–150). Always-lohko vastaa VHDL:n process-lohkoa. Lohkoissa voidaan käyttää erilaisia ohjelman suoritusta ohjaavia toimintoja, kuten tapahtumakontrollia. Always-lohkossa voidaan käyttää blokkaavaa tai ei-blokkaavaa sijoitusta, riippuen halutaanko toteuttaa kombinatorista vai sekvenssilogiikkaa. Always-lohkon ulostulo tulee aina sijoittaa reg-tietotyyppiin (Kuva 13). Blokkaavaa ja ei-blokkaavaa sijoitusta ei tule käyttää samassa always-lohkossa, koska blokkaava sijoitus estää muiden sijoitusten toteuttamisen ennen kuin sen sijoitus on tapahtunut. Tämä voi lukita toiminnan, jolloin ei-blokkaavat sijoitukset eivät välttämättä saa arvoja. Tämän vuoksi blokkaavaa sijoitusta tulee käyttää kombinaatiopiireissä ja ei-blokkaavaa sekvenssipiireissä.

```
/*moduulin esittely*/
module And_gate(a,b,c);

    input a;
    input b;            /*sisään ja ulostulojen määrittely*/
    output c;

    reg c;              /*peräkkäisessä suorituksessa ulostulon
                        tietotyyppinä*/

    /*moduulin toiminta*/

    always@(a or b)     /*always-lohko, suluissa herkkyyslista*/

        begin           /*aloitetaan lohkon toiminta*/

        c <= a & b;     /*ei-blokkaava sijoitus*/

        end             /*lopetetaan lohkon toiminta*/

endmodule               /*moduulin lopetus*/
```

Kuva 13. Always-lohkon käyttö moduulissa. Always-lohkon toimintaan vaikuttavat muuttujat sijoitetaan siinä olevien sulkujen sisälle. Lohkon sisällä arvon saavien muuttujien tulee olla tietotyypiltään reg. Suoritus tapahtuu sulkujen sisällä olevien muuttujien arvojen vaihtuessa. Tässä tapauksessa sisääntulojen saadessa arvoja suoritetaan looginen ja-operaatio, jonka tulos tulee muuttujan c arvoksi.

2.4.3. SystemVerilog

SystemVerilog on Verilog-laitteistokuvauskielen (IEEE 1364–2005) standardin laajennusosa. SystemVerilog sisältää piirteitä SUPERLOG-, C-, C++, VERA- ja VHDL-kielistä. Kokonaisuus, joka SystemVerilogilla on saatu luotua, kutsutaan HDVL-kieleksi (Hardware Description and Verification Language). Tämän avulla suunnittelijoiden on helpompi toteuttaa ja testata suurempia kokonaisuuksia ja varmistua niiden toiminnasta. SystemVerilogin kehittäminen aloitettiin Accelleran standardointiorganisaatiossa Verilogin kehityksen ohessa. Ensimmäinen julkaisu on vuodelta 2002, SystemVerilog 3.0. Tätä julkaisua kutsutaan myös Verilogin kolmanneksi sukupolveksi. Nykyinen SystemVerilog-standardi on IEEE 1800–2005. (Sutherland, Davidmann & Flake 2006: 1–3.)

SystemVerilog sisältää kaikki Verilogin tietotyypit, kuten reg ja integer. Tietotyypillä ilmaistaan ovatko arvot kaksi- vai nelitilaisia. Nelitilaisia tietotyyppejä ovat logic, reg, integer ja time. SystemVerilog sisältää uusia kaksitilaisia tietotyyppejä, jotka sopivat systeemi- ja transaktiotason mallintamiseen. Kaksitilaisia tietotyyppejä ovat bit, joka vastaa yhtä bittiä, tavua vastaava byte, 16-bittinen kokonaisluku shortint, 32-bittinen kokonaisluku int sekä 64-bittinen kokonaisluku longint. Reg ja logic tietotyyppejä käytetään yleensä laitteiston mallinnuksessa peräkkäisessä suorituksessa, kuten always-lohkossa. Tietotyyppejä int ja byte voidaan käyttää yhdistämään SystemVerilogilla kirjoitetut mallit C ja C++-kielisiin malleihin käyttämällä DPI:tä (Direct Programming Interface). DPI on rajapinta SystemVerilogin ja vieraan ohjelmointikielen välillä, jossa molemmilla kielillä on oma erillään oleva kerros. Vieraana kielenä voidaan IEEE:n mukaan käyttää C-ohjelmointikieltä. SystemVerilog kerros ei ole riippuvainen vieraasta kielestä. SystemVerilog kerroksessa vieraan kielen funktionkutsuprotokolla ja argumentin syöttömekanismit ovat läpinäkyviä ja irrelevantteja SystemVerilog:lle. Vieraan kielen kerros määrittelee kuinka argumentit välittyvät, kuinka niihin päästään käsiksi, kuinka SystemVerilogin tietotyypit esitetään ja kuinka ne käännetään ennalta määritellyiksi C-kielen tietotyypeiksi. Int tietotyyppiä käytetään yleisesti for-silmukoissa samalla tavalla kuin tavanomaisissa ohjelmointikielissä. Lisäksi SystemVerilog sisältää void tietotyypin, johon ei voi tallentaa arvoa, sekä shortreal tietotyypin, johon voi tallentaa 32-bittisen liukuluvun. (IEEE 1800–2005 2005: 15–17, 399, 400–401.)

SystemVerilog sisältää tietotyyppejä, jotka ovat oletusarvoltaan etumerkillisiä. Näitä ovat aiemmin mainitut byte, shortint, int ja longint. Etumerkilliset tietotyypit voidaan muuttaa etumerkittömiksi avainsanalla unsigned. Omien tietotyyppien määrittäminen on mahdollista avainsanalla typedef. Omat tietotyypit voidaan määritellä kokoelmissa, paikallisesti tai ulkopuolella. (IEEE 1800–2005 2005: 15, 18, 24–25.)

Luetellut tyypit määrittelevät joukon kiinteitä nimivakioita. Niiden avulla voidaan muodostaa abstrakteja muuttujia, jotka voivat saada listalla kerrottuja arvoja. Jokaiselle arvolle tulee määritellä käyttäjän antama nimi. Lueteltu tyyppi määritellään SystemVerilogissa avainsanalla enum (Kuva 14). (IEEE 1800–2005 2005: 26–27.)

```
enum {red, yellow, green} Light1, Light2;
```

Kuva 14. Lueteltujen tyyppien määrittely. Light1 ja Light2 on määritelty muuttujiksi nimeämättömälle kokonaislukutietotyypille, jonka sisältää jäsenet red, yellow ja green. (IEEE 1800–2005 2005: 26)

SystemVerilog sisältää C-ohjelmointikielestä tutut struktuurit (struct) ja unionit (union). Struktuurin (Kuva 15) avulla useita samankaltaisia tietoja voidaan koota yhteen. Struktuuri määritellään samalla tavalla kuin C-kielessä avainsanalla struct. Struktuurin jäsenet voivat olla mitä tahansa muuttujia, mukaan lukien käyttäjän määrittelemät muuttujat ja vakiot. Struktuuriin viittaus on samanlainen kuin C-kielessä. Aiemmin mainittua typedef-avainsanaa käytetään määrittämään käyttäjän määrittelemiä struktuureja. (IEEE 1800–2005 2005: 31.)

```
struct {
            bit [7:0] opcode;
            bit [23:0] addr;


          } IR;
union {
          int i;
          int unsigned u;

    } Data;
```

Kuva 15. Struktuurin ja unionin määrittely. Tässä struktuurissa määritellään muuttuja IR, joka sisältää 8 ja 24 bittiset vektorit joiden tietotyyppi on bit. (IEEE 1800–2005 2005: 31). Tässä määritellään unioni nimeltään Data, joka sisältää kokonaisluvun ja etumerkillisen kokonaisluvun. (Sutherland ym. 2006: 106)

Unioni on yksittäinen tallennuspaikka, johon voidaan viedä erityyppisiä arvoja. Unionin määrittely (Kuva 15) ja viittaus sen arvoihin tapahtuu samanlaisesti kuin struktuurin määrittely, käytettävä avainsana on union. Unionin ja struktuurin erona on arvojen tallennus. Struktuuriin voidaan tallentaa useita arvoja, kun taas unioniin vain yksi arvo. Unioneissa voidaan myös käyttää avainsanaa typedef määriteltäessä käyttäjän määrittelemiä unioneita. (IEEE 1800–2005 2005: 31.)

Peräkkäinen suoritus SystemVerilogissa tapahtuu always-lohkossa samaan tapaan kuin Verilogissa. SystemVerilog sisältää tavanomaisen always-lohkon lisäksi kolme laitteistotyyppikohtaista always-lohkoa, jotka ovat always_comb kombinaatiologiikalle, always_latch säppipiireille ja always_ff sekvenssilogiikalle (Kuva 16). Tavanomaisen always-lohkon tapaista herkkyyslistaa ei tarvitse määritellä always_comb ja always_latch-lohkoissa, koska kääntäjä tunnistaa mitä logiikkaa suoritetaan. Sekvenssilogiikka suorittaessa tulee always_ff-lohkon herkkyyslistaan määritellä muuttujat, jotka vaikuttavat lohkon suoritukseen, avainsanoja posedge tai negedge tulee käyttää. (IEEE 1800–2005 2005: 145–146.)

```
always_comb begin            //kombinaatiologiikka
        a = b & c;
    end

always_latch begin           //säppipiirit
        if(ck)
        q <= d;
    end
                                //sekvenssilogiikka
always_ff@ (posedge clk iff reset==0 or posedge reset)
    begin
        r1 <= reset ? 0: r2 + 1;
    end
```

Kuva 16. Always_comb-lohkon suoritus tapahtuu kerran ajan ollessa nolla ja sen jälkeen aina kun lohkossa käytettyjen muuttujien implisiittisesti määrittämässä herkkyyslistassa tapahtuu muutoksia, toiminnan määräytyessä sen funktion mukaan, joka tässä on ja-veräjä. Always_latch toimii samalla tavalla kuin always_comb. Always_ff aktivoituu kun sen herkkyyslistassa tapahtuu muutoksia, tässä tapauksessa nousevalla clk tai reset signaalin reunalla. (IEEE 1800–2005 2005: 145–146)

SystemVerilogissa voidaan määritellä kokoelmia (Kuva 17) VHDL:n tapaan. Syntetisoitavassa kokoelmassa voi olla funktioita, muuttujia, vakioita ja käyttäjän määrittelemiä tietotyyppejä. Kokoelma on oma erillinen SystemVerilog-tiedosto, jota voidaan käyttää useamman moduulin yhteydessä. Kokoelmien avulla voidaan jakaa funktioita, parametreja, tietoa, sekvenssejä ja ominaisuuksien esittelyjä eri moduulien välillä, käyttöliittymissä ja ohjelmissa. Kokoelmien ei tule sisältää prosesseja (always-lohkoja). (IEEE 1800–2005 2005: 327.)

```
package definitions;                    //kokoelman nimi
                                        //luetellut tyypit
typedef enum {ADD, SUB, MUL} opcodes_t;
typedef struct {                        //struktuurin määrittely
  logic [31:0] a,b;
  opcodes_t opcode;
} instruction_t;
                                      //funktion määrittely
function automatic [31:0] multiplier (input [31:0] a,b);
  return a * b;                       //funktion toteutus
endfunction
endpackage                            //kokoelman lopetus
```

Kuva 17. Kokoelmassa (package) on määritelty luetellut tyypit (opcodes_t), struktuuri (instruction_t) sekä funktio (multiplier), joka laskee kahden 32-bittisen vektorin tulon. Kokoelman nimi on definitions. (IEEE 1800–2005 2005: 327–331)

Suora viittaus kokoelmaan (Kuva 17) tapahtuu ::-operaattorilla (Kuva 18), jota kutsutaan luokan näkyvyysalueoperaattoriksi. Tällä voidaan viitata kokoelmaan sen nimellä ja haluttuihin määrittelyihin kokoelmassa. Tietyt osat kokoelmasta saadaan käyttöön import-lauseella (Kuva 18). Kokoelman kaikki osat saadaan käyttöön ::*-operaattorilla (Kuva 18), jota sanotaan jokerimerkiksi. (IEEE 1800–2005 2005: 329–330.)

```
module ALU                                //struktuuri sisääntulona
(input definitions::instruction_t IW,
  input logic clock,
  output logic [31:0] result);

import definitions::ADD;                  //import-lause
import definitions::SUB;

import definitions::*;                    //kaikki osat käyttöön

endmodule
```

Kuva 18. Moduulin nimen alapuolella sisään-ja ulostulojen määrityksissä ilmoitetaan, että käytetään tiettyä kokoelman (definitions) osaa, joka tässä tapauksessa on struktuuri. Keskellä olevilla import-lauseilla saadaan käyttöön halutut osat kokoelman luetelluista tyypeistä. Viimeisellä import-lauseella saadaan käyttöön kaikki kokoelman määritykset. (IEEE 1800–2005 2005: 327–331)

2.5. FPGA:lle sulautetut mikroprosessorit

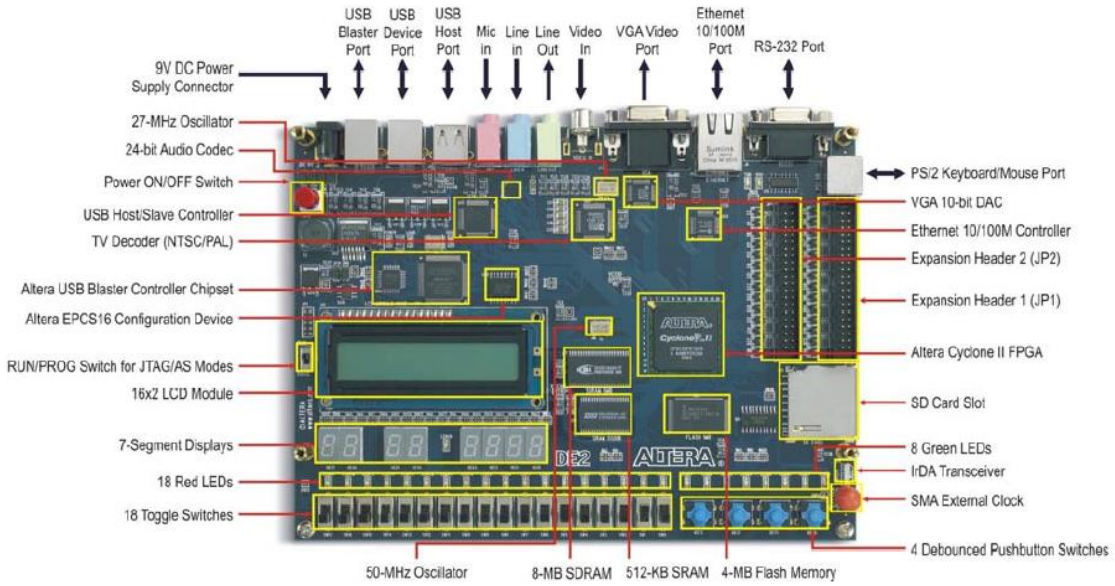
FPGA-piirille on mahdollista liittää mikroprosessori laitteistona tai ohjelmistollisesti IP-lohkolla. Laitteistona prosessori toteutetaan kiinteästi osaksi FPGA-piiriä. Ohjelmoitu prosessori toteutetaan käyttäen hyväksi FPGA-piirin logiikkaa. Ohjelmistototeutuksen etuna laitteistototeutukseen on muun muassa sen joustavuus sekä helppo muunneltavuus, jolloin prosessorista voidaan ottaa käyttöön vain tarvittavat osat. Laitteistona toteutettu prosessori vie FPGA-piiriltä resursseja, eikä se ole yhtä nopea kuin ohjelmana toteutettu prosessori.

Prosessorien ominaisuudet ja sisäiset rakenteet ovat samankaltaisia kuin tavallisilla prosessorijärjestelmillä. Perusrakenteina niissä ovat väylät, rekisterit ja aritmeettislooginen yksikkö. Etuna FPGA-piirille toteutetulla prosessorilla perinteiseen prosessorijärjestelmään verrattaessa on mahdollisuus lisätä omia lohkoja, joiden avulla ohjelmiston suoritusnopeutta voidaan lisätä suorittamalla aikaa vievät toiminnot rinnakkaisesti laitteistolla. Lisäksi koko järjestelmä pystytään toteuttamaan käyttäen ainoastaan yhtä mikropiiriä (System on Chip).

FPGA-piirien valmistajat Altera ja Xilinx tarjoavat kummatkin oman tuoteperheen sulautettuja ohjelmistoprosessoreja. Alteran tuoteperhe on Nios, jossa on vaihtoehtoina kolme erilaista prosessoriydintä, lisäksi Altera tarjoaa laitteistona ARM-prosessorin (Altera 2014b). Xilinxin vastaavat ovat nimeltään MicroBlaze ja PicoBlaze, sekä laitteistona PowerPC (Xilinx 2014). Valmistajien omien tuoteperheiden lisäksi on olemassa avoimeen lähdekoodiin perustuvia prosessoreita, joiden lähdekoodit ovat vapaasti saatavilla ja muokattavissa. (Penttinen 2005: 7; 24–29.)

3. FPGA-OPETUS/LABORATORIOT

Tässä osiossa kerrotaan miten FPGA-tekniikan opetus ja harjoitukset suoritetaan eräissä muissa yliopistoissa. Vaasan yliopistossa FPGA-tekniikan opetuksessa on käytetty kevääseen 2014 asti Alteran DE2-opetusalustaa, jota käytetään myös muissa yliopistoissa. DE2-lauta on yksi Alteran yliopisto-ohjelmassa (University program) käytetyistä alustoista.



Kuva 19. Altera DE2-kehitysalusta. (Altera 2014a)

3.1. Laitteistokuvauskielten opetus

Ellerveen, Reinsalun, Arhipovin, Ivaskin, Tammemäen, Evartsonin ja Sudnitsonin (2008; 37–38) mukaan valmistuvien opiskelijoiden kilpailukyky työmarkkinoilla on parempi, jos he tuntevat useamman laitteistokuvauskielen pääperiaatteet ja niiden suurimmat erot. Verilog ja VHDL ovat monesta syystä käytössä teollisuudessa, osa yrityksistä suosii Verilogia ja toiset käyttävät VHDL:ää. Kehitystyön tapahtuessa ympäri maailman suunnittelijan tulisi tuntea molempien kielten yleiset piirteet ja eroavaisuudet. Nämä kaksi kieltä antavat hyvän pohjan oppia myös muita laitteistokuvauskieliä.

Tyypillisiä ongelmia laitteistokuvauskielen oppimisessa ovat VHDL:ssä signaalin ja muuttujan sijoitus ja Verilogissa blokkaavan ja ei-blokkaavan sijoituksen ero. VHDL-kielen opettaminen on hyvä aloittaa rakenteellisesta kuvauksesta (structural), koska se auttaa huomaaman eron tavallisiin ohjelmointikieliin. Rakenteellisella kuvaustavalla laitteiston eri elementit on helpompi erottaa kuin käyttäytymismallista (behavioural). (Ellervee ym. 2008:39.)

Laitteistokuvauskielten opettaminen kannattaa Reinsalun, Arhipovin, Evartsonin ja Ellerveen (2007: 69–70) mukaan aloittaa VHDL-kielellä ennen Verilogia, koska VHDL on yleisin digitaalisten järjestelmien suunnittelukieli Euroopassa. VHDL:n ja Verilogin pääpiirteiden oppimisen jälkeen on helpompi siirtyä toisiin kieliin kuten esimerkiksi SystemC:hen, jolla voidaan mallintaa sekä ohjelmistoa että laitteistoa.

VHDL-laitteistokuvauskieltä käsittelevät kirjat aloittavat kielen kuvauksen esittelemällä sen käyttöä digitaalisten systeemien mallinnuksessa hyödyntäen sen monia ominaisuuksia, kuten ajoitukseen liittyvää informaatiota. Duckworthin (2005: 35–36) mukaan parempi tapa on aloittaa esittelemällä kielen syntetisoitavat osat esimerkkien avulla. Tämän jälkeen kannattaa esitellä ne osat joita käytetään simulointiin, testaukseen ja mallinnukseen. Tämä opetustapa vähentää sekaannuksen mahdollisuutta ja auttaa opiskelijoita välttämään sopimattomien lausekkeiden käyttämistä syntetisoimiseen. Opiskelijoille tulisi myös selventää mitkä ovat kombinaatio- ja sekvenssipiirien erot, eli se että sekvenssipiiri sisältää kiikkuja mutta kombinaatiopiiri ei. FPGA-piireille sulautettavat mikroprosessorit lisäävät laitteiston kehityksen, virheenkorjauksen ja hallinnan haastavuutta. Yksinkertaisten prosessorien käyttö on aluksi suositeltavaa opiskelijaprojekteissa.

Rodriguez-Poncen ja Rodriguez-Resendizin (2013: 172–177) mukaan digitaalisen suunnittelun kurssi on olennainen osa insinöörien koulutusohjelmaa, koska digitaalitekniikkaa on kaikkialla, kuten viihdeteollisuudessa, aseteknologiassa ja lääketieteellisissä laitteissa. Heidän mukaansa kuitenkin monet yliopistot toteuttavat digitaalitekniikan opetuksen vanhentunein menetelmin, kuten TTL-piirein, eikä esimerkiksi FPGA-tekniikkaa ole suuressa määrin hyödynnetty. Pelkästään uuden laitteistokuvauskieltä käsittelevän kurssin toteuttaminen ei aina ole mahdollista johtuen tiiviistä opintosuunnitelmasta, vaan kurssien toteutusta tulisi muokata sellaiseksi, että siihen saadaan mahtu-

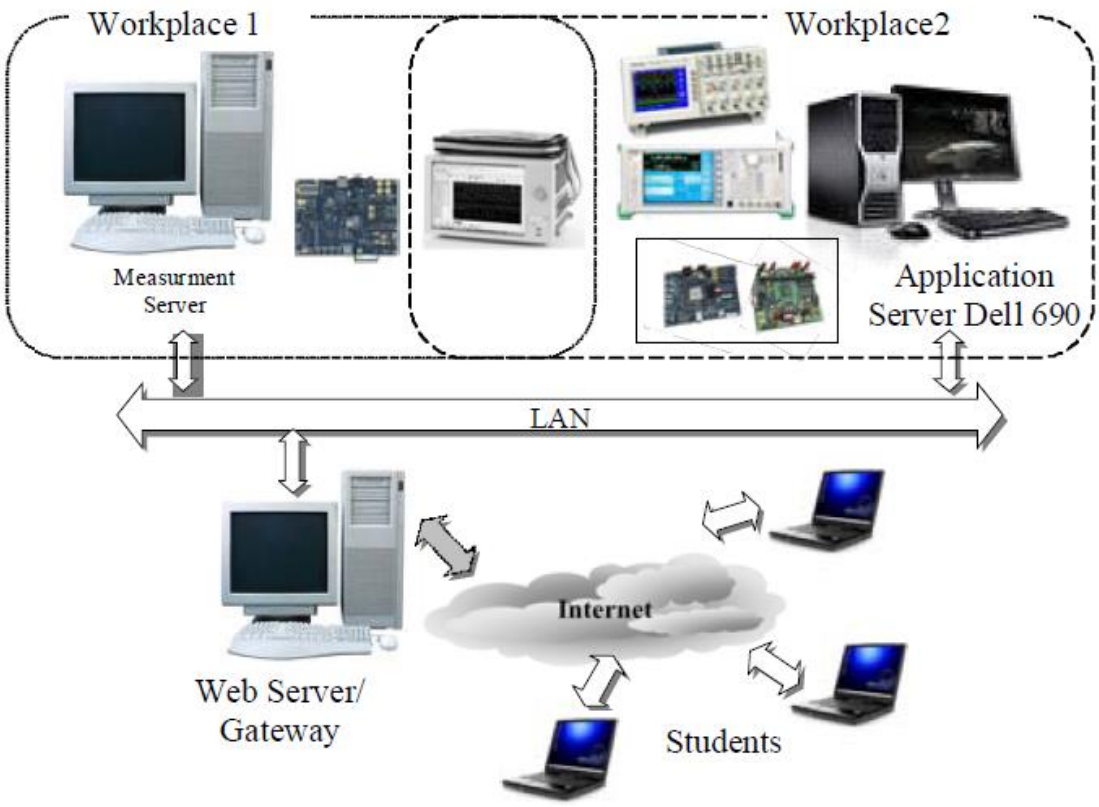
maan laitteistokuvauskieli. He toteuttivat digitaalisen suunnittelun kurssin uudella tavalla. Ennen kurssin harjoituksiin kuului ainoastaan kaksi VHDL-toteutuksen simulointikertaa ja muut harjoitukset koostuivat TTL-piirien käytöstä. Kurssin toteutusta muokattiin siten, että siihen tuli enemmän VHDL-kieltä, jota opiskelijat pystyivät käyttämään matemaattisiin operaatioihin, joita tarvitaan esimerkiksi digitaalisissa säätöalgoritmeissa. VHDL:n kaikki kuvastavat (dataflow, behavioural ja structural) käytiin lävitse kurssin aikana. Harjoitukset toteutettiin käyttäen Altera DE2-kehitysalustaa TTL-piirien sijasta. Toteutetun kyselytutkimuksen perusteella he sanovat, että VHDL:n lisääminen kurssille lisäsi opiskelijoiden ymmärrystä digitaalisesta logiikasta ja antoi enemmän valmiuksia vaativimpiin digitaalitekniikan projekteihin.

Knodelin, Zabellin, Lehmannin ja Spallekin (2014: 1–6) mukaan FPGA tarjoaa nopean kehitysalustan IC-piireille. Näiden piiritoteutusten tullessa yhä monimutkaisemmiksi tulee niiden tekniikan opetus aloittaa yhä aikaisemmin. Laitteistokuvauskielet ovat nykyaikaisin tapa kuvata, simuloida ja tarkistaa oikeellisuus digitaalisissa systeemeissä niin tutkimuksissa, teollisuudessa kuin yliopisto-opetuksessa. Heidän mukaan siirtyminen imperatiivisesta ohjelmoinnista rinnakkaiseen laitteistokuvaukseen on suurin haaste opiskelijoille. Kurssien toteutusten verifiointi tulisi toteuttaa myös oikealla laitteistolla simuloinnin lisäksi. He aloittavat VHDL:n opetuksen ohjelmoitavien logiikoiden kurssilla, jossa sen syntaksi esitellään yksinkertaisilla kombinaatio- ja sekvenssitoteutuksilla käyttäen Alteran Quartus:ta. Seuraavissa kursseissa opiskelijoiden tulee itse syventää VHDL:n osaamista, jotta he voivat tehdä harjoituksia. Kirjoittajien kokemuksen perusteella simulointi on tärkeää, koska oikean laitteiston testaus on monimutkaista. Testipenkkien käytöstä on hyötyä opiskelijoille erityisesti silloin, kun he tutustuvat ohjelmoitavaan laitteistoon.

3.2. Etälaboratoriot

Fyysisten laboratorioiden ohella osa yliopistoista on toteuttanut FPGA-opetuksen etälaboratoriona. Laboratoriotöissä opiskelijat joutuvat yleensä käyttämään kalliita laitteistoja. Usein budjettisyistä johtuen yliopistoilla ei yleensä ole useampaa tällaista tilaa, joka

sisältäisi näitä kalliita laitteistoja. Tämän vuoksi opiskelijoillakin on rajoitettu pääsy tiloihin. Laboratorion toteuttaminen etälaboratoriona voi olla ratkaisu tähän ongelmaan. Etälaboratorio perustuu oikeaan laitteistoon, kun taas virtuaalilaboratoriot perustuvat erilaisiin ohjelmamalleihin. Etälaboratoriot ovat käteviä ja tehokkaita laitteiston suunnittelukursseille. (Drutarovsky, Saliga, Michaeli & Hroncova 2009: 54.)



Kuva 20. Etälaboratorion arkkitehtuuri. Mittauksia voidaan suorittaa pelkästään digitaalisesti, tai käyttäen analogia- ja digitaalielektroniikkaa yhdessä. Ethernetin kautta mittalaitteet on yhdistetty mittauspalvelimelle. (Drutarovsky ym. 2009: 55)

Drutarovskyn ym. (2009: 55–56) ehdottaman etälaboratorion arkkitehtuuri on kuvassa 20. Etälaboratorio koostuu muun muassa LabVIEW-pohjaisesta internet- ja mittauspalvelimesta, FPGA-kehitysalustoista, logiikka-analysaattorista (LA), digitaalisesta oskilloskoopista (DSO) ja signaaligeneraattorista. Ensimmäinen työasema (Workplace1) on tarkoitettu yksinkertaisiin FPGA toteutukseen. Toisella työasemalla (Workplace2) on tarkoitus tehdä monimutkaisempia toteutuksia, kuten digitaalisten signaaliprosessorilohkojen (IP) liittämistä FPGA-piirille.

Internetin kautta toimivat etälaboratoriot antavat opiskelijoille mahdollisuuden joustavasti valita ajan ja paikan laboratorioharjoituksille. Persiano, Rapuano, Zoino, Morganella & Chiusolo (2007: 2–4) esittävät oman vaihtoehtonsa etälaboratoriolle, jonka tarkoituksena on FPGA-sovellusten etähallinta ilman käyttäjän tietokoneelle asennettavaa ohjelmaa. Heidän etälaboratorio on internetiin perustuva sivusto, jonka tarkoitus on opettaa käyttäjälle kaikki tarvittavat vaiheet FPGA-suunnittelusta. Käyttäjän on mahdollista ohjata useampaa mittalaitetta tilanteen mukaan, suorittaa omia testauksia mittalaitteilla tai seurata opettajan suorittamaa testausta. Käyttäjän ei tarvitse asentaa liitännäisiä tai muita ohjelmia, vaan hän pääse käsiksi kaikkiin resursseihin, kuten ohjelmistoon ja laitteistoon. Käyttäjä tarvitsee ainoastaan internetyhteyden ja -selaimen. Käyttöliittymänä toimii LabVIEW:llä toteutettu järjestelmä jolla FPGA-sovelluksia tarkastellaan kameran välityksellä.

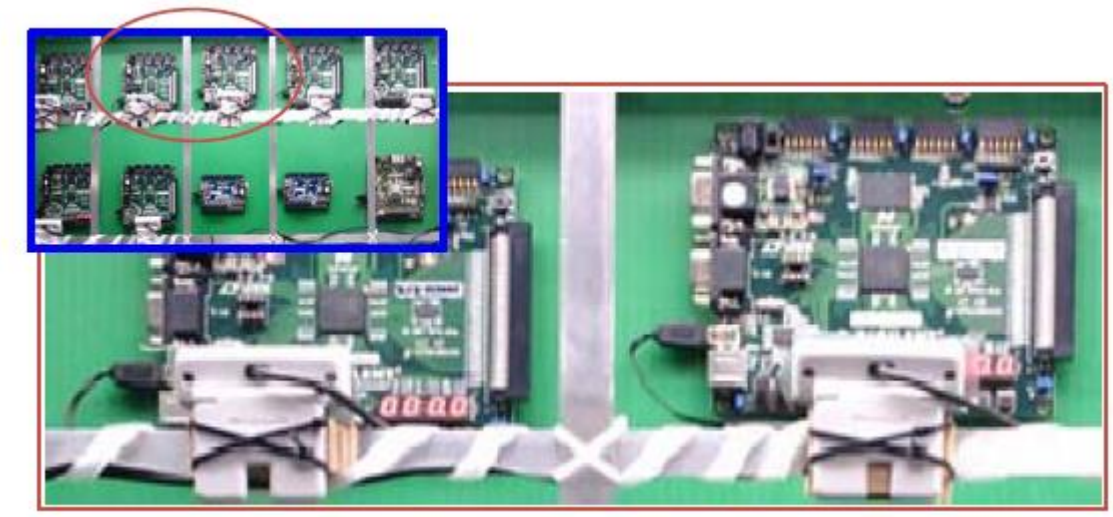
Dattan ja Sassin (2007: 343) ideana on tehdä 64 Xilinxin ML-310 FPGA-kehitysalustaa saavutettavaksi internetin kautta käyttäen SSH-palvelinta. Käyttäjällä on mahdollisuus kytkeä kehitysalustat päälle ja pois, lähettää/ladata tiedosto, konfiguroida alustat ja suorittaa syntetisointi.

El-Medanyn (2008: 107) esittämä etälaboratorio koostuu 20 tietokoneesta ja Spartan 3E FPGA-kehitysalustasta. Tietokoneet on kytketty kampuksen lähiverkkoon, jonka kautta käyttäjät voivat konfiguroida ja testata alustojen toiminnan. Käyttäjä pystyy graafisen käyttöliittymän kautta simuloimaan alustan kytkinten ja ledien toimintaa, sekä kontrolloimaan sen muita toimintoja. Tulevaisuudessa on tarkoitus vielä lisätä kamera jokaiseen kehitysalustaan, jolla käyttäjä voi tarkkailla alustan toimintaa.

Rajasekharin, Kritikosin, Schmidtin ja Sassin (2008: 687–689) mukaan tietokonesimuloinnilla ei aina saavuteta riittävää tarkkuutta joita FPGA-toteutuksissa on, eikä sillä pystytä havaitsemaan kaikkia ongelmia. Toteutusten koon kasvaessa simulointikin yleensä hidastuu. Aiemmin simulointia käytettiin vaihtoehtona tavanomaisille laboratorioharjoituksille. Heidän mukaansa etälaboratorion tulee tarjota useita toiminnallisuuksia kuten kehitysalustan käynnistys/sammutus, virheenkorjaus ja uudelleen konfigurointi sekä olennaisten tulosten tallentaminen. Kustannusten minimoimiseksi heidän etälaboratoriossaan kaikki tukitoiminnot on liitetty yhdelle työasemalle. Etälaboratorioon

sisäänkirjautuminen suoritetaan käyttäen serverin komentotulkkia, josta on mahdollista hallita kehitysalustojen virransyöttöä ja yhteydenpitoa.

Morganin ja Cawleyn (2011: 1–3) toteuttamassa etälaboratoriossa (remoteFPGA) jokainen FPGA-kehitysalusta on liitetty omaan nettikameraan, jotka näyttävät reaaliaikaista kuvaa jokaisesta aktiivisesta kehitysalustasta (Kuva 21). USB palvelin varaa resurssit FPGA:lle pääpalvelimella. Laboratorion palvelin jakaa pääsyn FPGA:n resursseihin yksittäiselle käyttäjälle. RemoteFPGA:n arkkitehtuuri koostuu FPGA-kehitysalustoista, sovelluspalvelimesta, verkkosivusta ja verkkokameroista, verkko- palvelimesta ja USB-palvelimesta. (Morgan, Cawley, Callaly, Agnew, Rocke, O'Halloran, Drozd, Kepa & McGinley 2011: 496–497.)



Kuva 21. FPGA-kehitysalustat ja kamerat. (Morgan ym. 2011: 1)

Etälaboratorio mahdollistaa kehitysalustan kaukokonfiguroinnin, tiedonsiirron sisään ja ulos USB kontrolli- ja statusrekisterin kautta (CSR). RemoteFPGA-IP-ydin muodostaa rajapinnan USB-oheislaitteeseen ja hallinnoi ydintä kontrolli- ja statusrekisterillä. Ytimen ohjaus- ja statusrekisteri käsittelee käyttäjän kellon valintaa ja kellon generointia. Verkkosovellus liittää FGPA-kehitysalustan ja verkkokameran käyttäjän sessioon ja hallinnoi siihen liittyvää informaatiota tietokannassa. Tietokanta säilyttää jatkuvasti käyttäjätilit, ladatut bittivirrat ja systeemin diagrammit. FPGA-kehitysalustojen konfi-

gurointi tapahtuu verkkosovelluksella käyttäen vaadittavia konfigurointibittejä. USB-palvelin hallinnoi datan kirjoitusta/lukemista FPGA:lla olevan USB-rajapinnan kautta. Etälaboratorion verkkosivu kommunikoi kehitysalustojen kanssa USB-palvelimen kautta ja käyttää HTTP-protokollaa reaaliaikaisen käyttöliittymän näyttämiseen.

Soaresin ja Lobon (2011: 95–97) mukaan etälaboratorioissa haasteena on käyttäjän ja laitteiston välisen vuorovaikutuksen toteuttaminen mahdollisimman realistisena internet-rajapintana, jolloin käyttäjillä on mahdollisuus toimia kehitysalustan kanssa mistä tahansa. He toteuttivat sovelluksen käyttäen PHP:ta, joka mahdollistaa DE2-kehitysalustan ja kameran etäkäytön. Kehitysalustalla olevat painonapit (4 kpl) ja kytkimet (18 kpl) on toteutettu virtuaalisesti. Kehitysalustan verkkopalvelin sijaitsee kampuksen lähiverkossa, joka on yhdistetty USB:n kautta kehitysalustan JTAG-liitäntään. Myös kamera on kytketty palvelimeen USB:n kautta. Etälaboratorio soveltuu sellaisten kombinaatio- ja sekvenssipiiritoteutuksiin, joissa on yksinkertainen käyttöliittymä. Kehitysalustan FPGA-resurssit rajoittavat piiritoteutusten monimutkaisuutta. Käytettäessä virtuaalisia painonappeja ja kytkimiä tuloksen näkeminen kamerasta viivästyy noin sekunnin johtuen viiveestä kytkinten ja kehitysalustan välillä. Tällä hetkellä on mahdollista käyttää virtuaalisia kytkimiä ja painonappeja ja nähdä tulos esimerkiksi ledeistä ja näytöstä. Tulevaisuudessa kirjoittajat aikovat lisätä enemmän ominaisuuksia, kuten äänen, VGA-näytön, toisen kameran ja useampia kehitysalustoja.

Oritin, Julianin, Zulaican ja Cristin (2010: 2229–2236) mukaan laboratorioharjoituksilla on tärkeä osa insinöörikoulutuksessa. Etäoppimisen lisääntyessä laboratorioharjoitusten mahdollistaminen etäopiskelijoille on tullut yhä tärkeämmäksi. Laitteistolaboratoriot tarjoavat kokonaisvaltaisen oppimiskokemuksen kokeellisine mittauksineen, visuaalisena palautteena ja instrumentointina. Etälaboratorioita on kehitetty, jotta etäopiskelijat saisivat samat laboratoriokokemukset kuin paikanpäällä olevat opiskelijat. Kirjoittajat esittelevät FPGA-etälaboratorion, joka on suunniteltu tehoelektroniikan ja sähkönkäytön tutkimukseen. Laboratoriota on mahdollista käyttää paikanpäällä tai etänä internetin kautta. Yhteen laboratorioasemaan kuuluu Xilinxin Virtex IV kehitysalusta, virtalähde, piirilevy joka sisältää virta- ja jänniteantureita, piirilevy jossa on ADC/DAC-muuntimet, oskilloskooppi mittauksia varten sekä kameran joka näyttää reaaliaikaista

kuvaa etäkäyttäjille. Kommunikointi kehitysalustan ja tietokoneen välillä tapahtuu JTAG-kaapelia pitkin. Laboratoriossa voi suorittaa mittauksia ainoastaan (paikalla/etänä) silloin kun laboratorioteknikko on paikalla, lisäksi ohjauspaneeli estää laitteiden ylikuormittamisen. Etäkäyttö tapahtuu verkkopalvelimen kautta. Verkkorajapinta on suunniteltu sellaiseksi, että käyttäjä tarvitsee ainoastaan tietokoneen ja internetyhteyden testien suorittamiseen. Käyttäjien ei tarvitse asentaa mitään ohjelmia tietokoneelleen, vaan ainoastaan HTTP-porttia käytetään. Jokainen laboratoriokerta sisältää simuloinnin ja testauksen laitteella. Tarkoituksena on, että opiskelijat vertaavat simulointituloksia testaustuloksiin. Oskilloskoopista on mahdollista siirtää dataa Matlabille oskilloskoopin palvelimen kautta. Kirjoittajien suorittamien kyselyiden mukaan opiskelijat ovat olleet tyytyväisiä laboratorion käyttömahdollisuuksiin, samoin myös osallistumismäärät ovat kasvaneet. Taulukossa 1 esitetään yhteenveto etälaboratoriototeutuksista. Lisäkomponenteilla/työkaluilla tarkoitetaan FPGA:sta erillistä ohjelmaa tai komponenttia ja omilla kytkennöillä sitä onko mahdollista liittää komponentteja fyysisesti FPGA:lle.

Taulukko 1. FPGA-etälaboratorioiden toteutukset.

Viite	Toteutus	FPGA	Lisäkomponentit/ työkalut	Kehitys-ympäristö	Omat kyt-kennät
Drutarovsky ym. 2009	FPGA-etälaboratorio	EP2C35F672 & EP2S60F1020C 4	Logiikka-analysaattori, Oskil-loskooppi, Signaali-genetaattori	Lab-VIEW-ohjelma GUI:lla	Ei mahdolli-suutta omien kytkentöjen tekoon
Persiano ym. 2007	FPGA-etälaboratorio	EPM7128SLC8 4-7 & EPF10K70RC2 4-0-4	LabVIEW:tä käyte-tään tulosten visu-aaliseen näyttämi-seen webbikameran kautta	Quar-tus2ohjelma n käyttö internetin kautta	Ei mahdolli-suutta omien kytkentöjen tekoon
Datta ym. 2007	FPGA-etälaboratorio	Xilinx ML-310 kehitysalusta	Ei lisäkomponentte-ja	Kehitys-alustojen käyttö ssh-palvelimelta	Ei mahdolli-suutta omien kytkentöjen tekoon
El-Medany 2008	FPGA-etälaboratorio	Spartan 3E kehitysalusta	Tulevaisuudessa tarkoitus lisätä webbikamera	Xilinxin suunnittelu-ohjelmiston käyttö inter-netin kautta	Ei mahdolli-suutta omien kytkentöjen tekoon
Rajasekhar ym. 2008	FPGA-etälaboratorio	Xilinx ML-310 kehitysalusta	Ei käytössä lisä-komponentteja	Oma ohjel-ma, käyttö internetin kautta	Ei mahdolli-suutta omien kytkentöjen tekoon
Morgan ym. 2011	FPGA-etälaboratorio	Xilinx Spartan-3E kehitysalusta	Webbikamerat	Oma ohjel-ma, jota käytetään internetin kautta. Xili-xin suunnit-telu-ohjelmalla HDL-kuvaukset	Ei mahdolli-suutta omien kytkentöjen tekoon
Soares ym. 2011	FPGA-etälaboratorio	Altera DE2-kehitysalusta	Webbikamera. Komponentti, jolla kehitysalustan kyt-kimet ja painonapit näytetään virtuaali-sesti	Quartus2-ohjelmalla tulee tehdä ohjelmointi-tiedosto (.sof), joka siirretään palvelimelle	Ei mahdolli-suutta omien kytkentöjen tekoon
Oriti ym. 2010	FPGA-etälaboratorio	Xilinx Virtex IV kehitysalustat	Piirilevyt, joissa ADC/DAC-muuntimet, jännite ja virta-antureita sekä enkooderi. Webbikamerat sekä oskilloskooppi	Matlab/ Simu-link/Xi-linx suunnittelu-ohjelmisto	Ei mahdolli-suutta omien kytkentöjen tekoon

3.3. Tavanomaiset opetuslaboratoriot

Sulautettujen järjestelmien opetus aloitetaan yleisesti kursseilla, joissa kehitysalustassa on mikrokontrolleri. Haastavampien kurssien tarjonta painottuu opintojen loppuvaiheeseen, mikä voi jättää aukkoja osaamiseen. Tarjontaan on kuitenkin tullut kursseja, jotka hyödyntävät alustoja joissa on mahdollisuus laitteiston ja ohjelmiston yhteistoteutukseen. (Schneider, Bezdek, Zhang, Zhang & Rover 2005: 53.)

Schneider ym. (2005: 53–54) esittelevät julkaisussaan laboratoriotilan, jossa on 15 työasemaa varustettuna Xilinxin Virtex 2 Pro alustalla. Alustassa on tehokas prosessori ja mahdollisuudet lisälaitteiden kytkemiseen. Suunnitteluohjelmana käytetään EDK/ISE ohjelmaa, jossa prosessorin ohjelmointiin voidaan käyttää C/C++-ohjelmointikieltä. Laitteisto kuvataan ohjelmassa erillisinä IP-lohkoina, joka antaa opiskelijoille kokemusta systeemitason suunnittelusta. Laboratorioharjoituksissa he korostavat laitteiston ja ohjelmiston yhteistoteutusta. Harjoituksissa käytetään reaaliaikakäyttöjärjestelmää, jota on tarkoitus käyttää monisäikeisissä sulautetuissa järjestelmissä. Reaaliaikakäyttöjärjestelmän avulla opiskelijat pystyvät optimoimaan systeemin suorituskykyä monisäikeisellä toteutuksella ja systeemitason profiloinnilla.

Quintas, Valdes, Moure, Fernandez-Ferreira ja Mandado (2005) ovat toteuttaneet sovelluksen itseopiskeluun. Oletuksena on, että sovelluksen käyttäjät tuntevat digitaalitekniikan peruskomponentit kuten veräjät, kiikut, multiplekserit, muistit, laskurit sekä VHDL:n perusteet. Tarkoituksena on tehdä helpoksi FPGA-systeemien suunnittelumenetelmien oppiminen, syventää VHDL-kielen osaamista ja käyttää suunnitteluun Alteran tarjoamaa Quartus-ohjelmistoa. Toteutus on hypermediasovellus, johon sisältyy ohjeistus, ohjelmisto, FPGA kehitysalusta ja lisälaitealusta. Ohjelmistona on Alteran Quartus. Toteutuksessa on myös virtuaalinen logiikka-analysaattori, jonka avulla voidaan varmistaa suunnitellun systeemin toimivuus.

Yuxin, Lin, Junin, Jinpingin ja Zulin (2010: 414–417) mukaan FPGA-piirien avulla opiskelijat voivat käytännössä toteuttaa sovelluksia digitaalisesta signaalinkäsittelystä käyttäen apuna muita suunnitteluohjelmia, kuten Matlab:ia. Digitaalisen signaalinkäsittelyn sovelluksista he antavat esimerkkinä tutkan ominaisuuksien opettamisen Matlabin

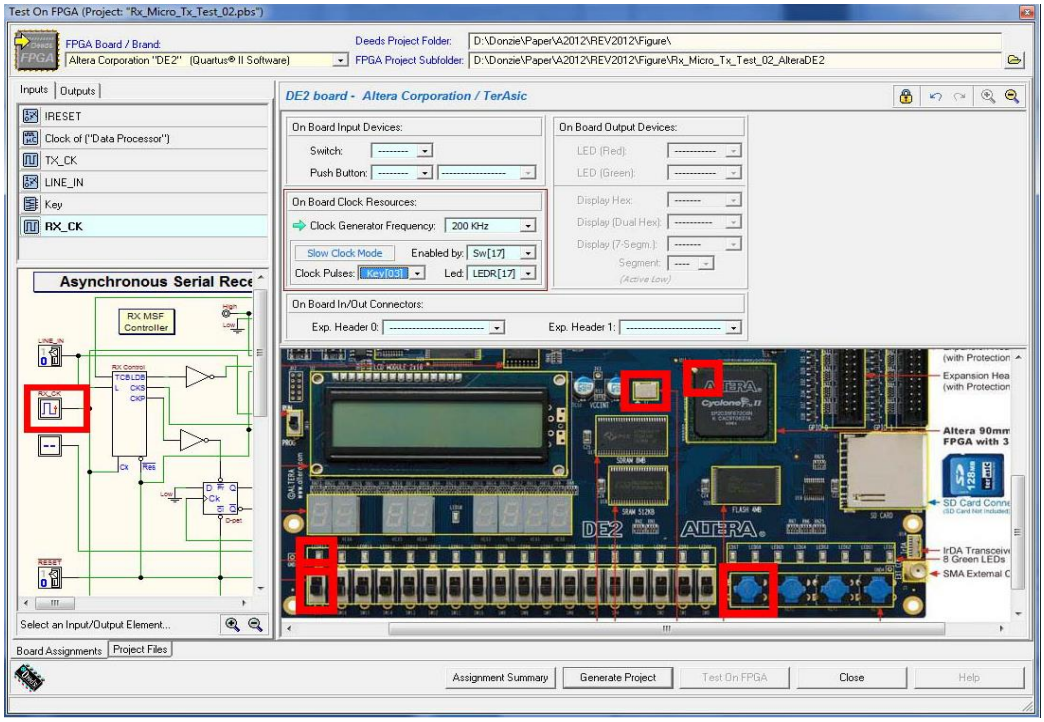
ja FPGA:n avulla. Matlab:ia käytetään tutkan ominaisuuksien simulointiin, jolloin tulokset saadaan näkyviin graafisesti. Simuloidun mallin toteutus on tarkoitus toteuttaa FPGA:lla. Matlab:ia käytetään toteutuksen muidenkin ominaisuuksien toteuttamiseen, kuten suodatinten kertoimien laskemiseen.

DE2-115 kehitysalustaan perustuva ja Alteran Cyclone IV FPGA-piirin sisältävä t-pad on sopiva alusta käytännön sulautettujen järjestelmien opetukseen. Nykyaikaisten sulautettujen järjestelmien suunnitteluun tarvitaan ohjelmiston ja laitteiston yhteissuunnittelua, IP-lohkojen käyttöä, prosessorin integrointia piirille sekä I/O-standardien hallitsemista. FPGA-piirit tarjoavat hyvän ja käytännöllisen alustan prosessorien integroimiseen piirille. (Mahmoodi, Montoya, Franco, Rodriguez, Carrillo, Goel, Chen, Enriquez, Jiang, Pong & Shanasser 2012: 1.)

T-pad alusta sisältää LCD-kosketusnäytön, kameran, USB:n, Ethernet liitynnän, Video Graphics Array (VGA) liitynnän ja ulkopuolisia liityntämahdollisuuksia, sekä muutenkin kokonaisvaltaisen alustan sulautettujen järjestelmien toteutukseen. Alustan mukana on kirjasto, jossa on valmiita IP-lohkoja eri I/O-kytkentöihin. Suunnittelutyökalujen manuaalit ovat hyvin kattavia, eivätkä ne välttämättä sovellu opiskelijoiden ohjeiksi. Tämän vuoksi kirjoittajat ovat laatineet itseopiskeluohjeita opiskelijoille, jotka käyvät lävitse koko suunnittelukaavion. Ensimmäinen ohje sisältää alustan ja sen ominaisuudet, toinen keskittyy ainoastaan laitteiston suunnitteluun, kolmas laitteisto-ohjelmisto- yhteissuunnitteluun ja viimeisenä LCD-kosketusnäytön käytön opettaminen. Saadun palautteen perusteella kirjoittajat suosittelevat t-pad alustan käyttöä opetuksessa. (Mahmoodi ym. 2012: 1–6.)

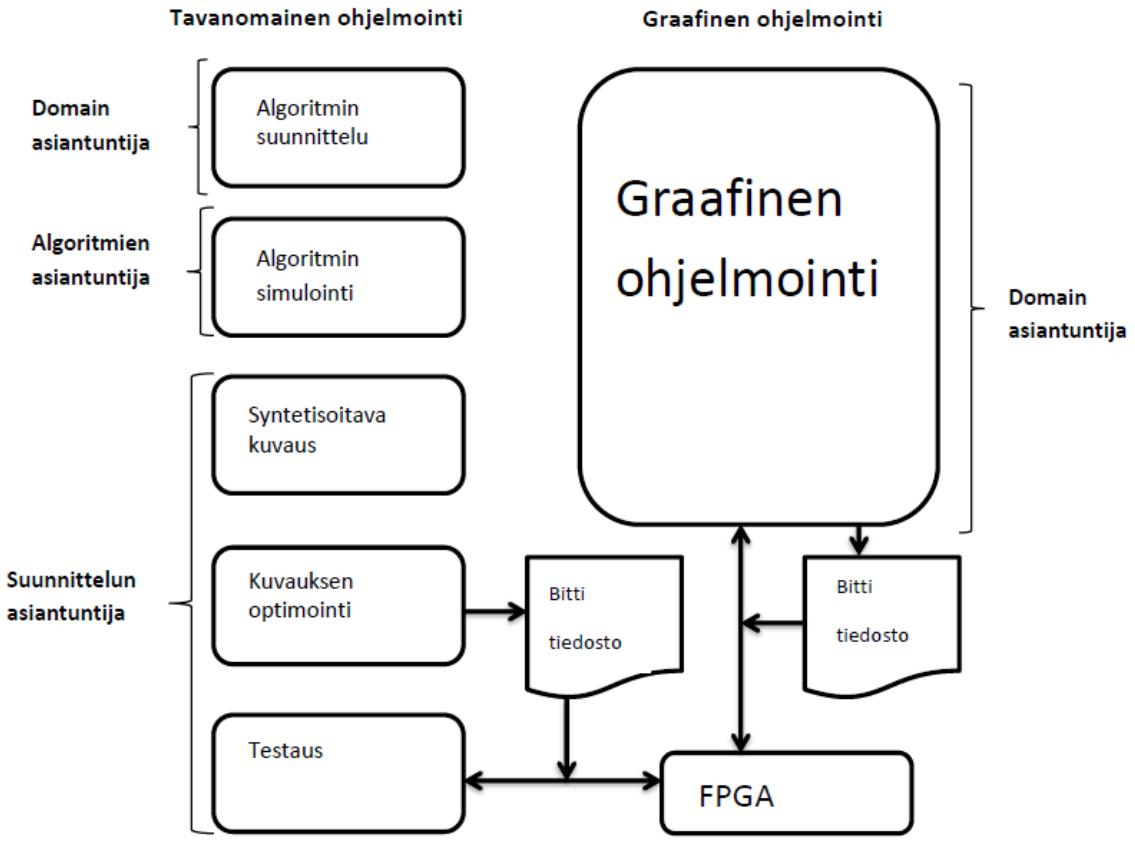
Donzellini ja Ponta (2012: 1–4) ovat kehittäneet digitaalisten piirien suunnittelualustan jonka nimi on Deeds. Alusta antaa opiskelijoille mahdollisuuden toteuttaa suunnitelmansa myös FPGA-piirille, minimoiden samalla laitevalmistajien tarjoamien suunnitteluohjelmien (CAD) käytön. Deeds kattaa sekvenssi- ja kombinaatiopiirit, tilakoneen suunnittelun, assembly kielen ohjelmoimisen sekä mikrotietokoneen liittämisen. Lisäksi se sisältää simulointityökalut piireille, mikrotietokoneelle ja tilakoneelle. Deeds eroaa muista suunnitteluohjelmista siinä, että se on tarkoitettu opetuskäyttöön, sekä helposta käyttöliittymästään (Kuva 22). Se tarjoaa myös paljon opiskelumateriaalia vanhojen

projektien muodossa. FPGA:n ja Deeds:llä toteutetun projektin integrointi tapahtuu saumattomasti, jolloin FPGA-piirin konfigurointi on helppoa. Lisäominaisuuden avulla on mahdollista ennalta määritellä liitynnät ja niiden ominaisuudet. Koodigeneraattori yhdistää sisään- ja ulostulot FPGA-piirille sekä asettaa kytkemättömiin liityntöihin oletusarvot. Koodigeneraattori luo myös muut tarvittavat projektitiedostot, joita tarvitaan FPGA-piirin ohjelmointiohjelmistossa. Koodin ja tiedostojen luonnin jälkeen käyttäjä pystyy ajamaan tiedostot FPGA-piirille käyttäen esimerkiksi Altera Quartus II-ohjelmaa. Valmistajien ohjeistukset ovat yleensä tarkoitettu ammattikäyttöön, jolloin opiskelijat näkevät tuloksia vasta käytyään monta askelta läpi. Suunnittelualustan lisäominaisuuden avulla opiskelijat voivat suorittaa ainoastaan tarvittavat toimenpiteet. Lisäominaisuudessa kytkentäkaavioita käytetään opetuksessa laitteistokuvauskielten ja FPGA-piirien kanssa. Laboratoriotilassa on 25 työasemaa, joissa jokaisessa on oma tietokone, Altera Quartus II ohjelmisto, Deeds-sovellus ja DE2 kehitysalusta (Donzellini & Ponta 2013: 47.)



Kuva 22. Käyttöliittymä. Kellonmäärittelyssä määritellään käytettävä taajuus. Punaisella merkittyjen kohtien toiminnallisuus voidaan määritellä käyttäjän toimesta käyttöliittymässä. (Donzellini ym. 2012: 6)

Balidin ja Abdulwahedin (2013: 24–27) mukaan sulautettujen FPGA sovellusten suunnittelussa kannattaa käyttää graafista ohjelmointikieltä, koska se on viisi kertaa nopeampaa kuin tekstipohjainen ohjelmointi, on helppo oppia eikä se vaadi erityistä ohjelmointikokemusta. Graafiset ohjelmointikielet perustuvat tietovuoparadigmaan, jossa ohjelma mallinnetaan sarjana operaatioita. Graafisten ohjelmointikielin vahvuutena on, että käyttäjät voivat käyttää samaa mallia useammassa alustassa vähäisillä muutoksilla. Käyttäjien ei tarvitse käyttää rajapintaa alemman tason ohjelmointikieliin liitynnöissä, jotka generoidaan automaattisesti. Kuvassa 23 esitetään tarvittavien askelien määrä tavanomaisella ja graafisella menetelmällä toteutettaessa FPGA-sovellusta. Kirjoittavat ovat kehittäneet laboratorion joka perustuu LabVIEW ohjelmalla tapahtuvaan graafiseen ohjelmointiin, Spartan-3E alustaan ja oheislaitteisiin.



Kuva 23. Tarvittavien suunnitteluaskelien ero ohjelmoitaessa FPGA tavanomaisin- ja graafisin menetelmin. (Perustuu lähteeseen Balid ym. 2013: 27.)

Kiray, Demir ja Zhaparova (2013: 445–447) suosittelevat digitaalitekniikan opettamiseen projektiperusteista (PBL) – ja mikro-opetusta. Projektiperusteisen opettamisen tarkoituksena on jakaa projekti sopiviin osiin parantamaan oppimista ja sopivuutta ryhmätyöskentelyyn. Mikro-opettaminen käsittää pienet opetusaiheet ja lyhyen aikavälin oppimistoimet. Esimerkkiprojektissaan heidän tarkoituksena on saada projektin osat suoritettua valmiiksi tietyssä ajassa ja samalla parantaa opiskelijoiden motivaatiota ja ymmärrystä. Mikro-opetusvaiheessa heidän mukaansa tulee määritellä aiheet jotka tulee oppia, opetusaika ja tarpeeksi kattavat testit oppimisen testaamiseksi. Nämä soveltuvat heidän mukaan FPGA-perusteisen digitaalitekniikan opetukseen. Projekti- ja mikroperusteisella opetuksella opiskelijat saavat mahdollisuuden toistaa ja yhdistellä vaiheita, joita he ovat oppineet vaihe kerrallaan.

Hocenski, Aleksi ja Sruk (2013: 22, 24–25, 28) ovat kehittäneet FPGA-verifiointialustan opiskelijoiden avuksi FPGA-piirien digitaaliseen suunnitteluun. Verifiointialusta (VDP) on toteutettu Spartan 3 XC3S200 FPGA-piirille ja käyttää 14 %:a sen kapasiteetista. Alusta koostuu FPGA:n ja tietokoneen välisestä tiedonsiirrosta ja tietokoneen graafisesta käyttöliittymästä (GUI). Alustan laitteiston verifikaatio suoritetaan ennalta määritetyillä sisään- ja ulostulomoduuleilla, jotka mahdollistavat reaaliaikaisen sisäisten signaalien tarkastelun. Sisään- ja ulostulomoduulit kytketään kahdeksan bittisiin virtuaalilaitteisiin (VD). Jokaisella virtuaalilaitteella on tietty fyysinen osoite. Tietokoneen ja FPGA:n välinen kommunikaatio tapahtuu RS232 full-duplex protokollalla. Kommunikaatiosta huolehtii piirillä olevat kaksi PicoBlaze mikroprosessoria ja UART-piiri.

Graafinen käyttöliittymä (GUI) on toteutettu Visual Basic ohjelmointikielellä. Käyttöliittymä on tehty käyttäen dynaamista ohjelmointitekniikkaa. Se toimii osana, joka kommunikoi FPGA:n kanssa RS2323 sarjaportin kautta. Käyttöliittymän toimintoja ovat sarjaliikenteen hallinnointi, sisään- ja ulostulosignaalien esittäminen tietokoneen näytöllä, tiedonsiirto ja vastaanotto virtuaalilaitteelta, virtuaalilaitteiden dynaaminen luonti ja poisto sekä satunnaisen osoitteen valitseminen tietylle virtuaalilaitteelle. Käyttäjät voivat käyttöliittymässä valita testausmoduuliin erilaisia sisääntuloja (kytki-

met/painonapit) ja ulostuloja (led). Käyttöliittymässä signaalien sisäiset tilat ovat koko ajan näkyvillä.

Karyano ja Wicaksana (2013: 1–3) ovat käyttäneet FPGA-tekniikkaa apuna opettaessaan mikrokontrollereiden ja mikroprosessoreiden perusteita. Kirjoittajien mukaan opiskelijat saavat tämän kautta kokemusta oikeasta laitteistosta, eikä pelkästään simuloinnista tai ohjelmiston kehittämisestä. Prosessorin osat kuten aritmeettis-looginen- ja ohjausyksikkö sekä muisti voidaan toteuttaa yksittäisinä moduuleina. Helpoin tapa toteuttaa mikroprosessori FPGA-piirille on käyttää lohkokaaviota, muut tavat ovat laitteistokuvauskieli ja tilakaavio. Lohkokaavion käyttö on yleistä suunniteltaessa digitaalisia järjestelmiä. FPGA tekniikan käyttö mikroprosessoreiden ja mikrokontrollereiden toiminnan opetuksessa antaa opiskelijoille yksityiskohtaista osaamista.

Brekkenin ja Mohanin (2006: 1–2, 4) mukaan tehoelektroniikan komponenttien ominaisuuksien tullessa muun muassa säätöominaisuuksiltaan yhä kehittyneemmiksi tulee insinööreillä olla kyky käyttää digitaalisia järjestelmiä, kuten FPGA:ta niiden hallintaan. Laitteisto perustuu Spartan 3 FPGA kehitysalustaan. FPGA:ta käytetään muuntajapiirin ohjauksessa, jolla voidaan esimerkiksi ajaa moottoria tai passiivista kuormaa suljetussa tai avoimessa silmukassa. FPGA:n lisäksi laitteistoon kuuluu kuorma, kokosilta-piiri, liitinlevy ja ajuri-piiri. Spartan-3-levyllä olevia kytkimiä voidaan käyttää reaaliaikaisesti esimerkiksi pulssileveysmodulaatiossa (PWM) pulssileveyden säätämiseen. Laitteisto on kirjoittajien mukaan joustava ja edullinen vaihtoehto digitaalisiin tehoelektroniikan sovelluksiin.

Tampereen teknillisessä yliopistossa toteutettiin vuonna 2008 projekti, jonka tarkoituksena oli parantaa ja modernisoida tietokonetekniikan laitoksen kurssien laboratorioharjoituksia. Tarkoituksena oli käyttää perättäisissä kursseissa samaa laitteistoa, jolloin opetuksessa voidaan keskittyä olennaiseen eikä opetella yhä uusien laitteiden tai työkalujen toimintaa. Laitteistoksi valittiin Altera DE2. Tietokonetekniikan laitoksella yhdeksän kurssia käyttävät samaa alustaa. Yksi alusta on hyödyllinen opiskelijoille, koska heidän ei joka kerta tarvitse opetella uuden alustan toiminnallisuutta. (Vainio, Salminen ja Takala 2010: 137–138, 140.)

Kurssien laboratoriotöissä opiskelijat törmäävät usein ongelmiin, jotka liittyvät uusien laitteiden käyttöön, harjoitusten aikatauluihin sekä ylläpidettävyyteen. Näiden ongelmien ratkaisemiseksi Kastelan, Majstorovic, Nikolic, Eremic ja Katona (2012: 1113, 1115) ehdottavat ristikkäistä sulautettujen järjestelmien opetuslaitteistoa, joka perustuu FPGA:han, ja jonka opiskelijat pystyvät konfiguroimaan suunnitelmallaan. Tarkoituksena on, että se kattaa sulautettujen järjestelmien koko opetusprosessin, mukaan lukien yhtymäkohdat eri teknologioiden välillä. Heidän suunnittelemat harjoitukset alustalle kattavat kaikki digitaalitekniikan perusasiat, kuten tilakoneet ja kombinaatio- ja sekvenssipiirit käyttäen VHDL:ää.

Haban (2014: 794) mukaan nykyään FPGA-piirien suorituskykyä arvioidaan logiikkalohkojen lisäksi I/O-liityntöjen, muistilohkojen, laskentalohkojen ja prosessorilohkojen ominaisuuksien mukaan. Hänen mukaansa nämä ominaisuudet mahdollistavat sen, että FPGA-pohjaiset alustat voivat korvata laboratoriotöissä aiemmin käytetyt mikrokontrollerit, prosessorit ja muut vastaavat laitteet. FPGA-piirien uudelleenohjelmoitavuus on yksi tärkeimmistä syistä miksi ne erittäin hyvin soveltuvat digitaalitekniikan opetukseen. FPGA:lla on etuna se, että opiskelijat voivat käyttää samaa laitetta, ohjelmistoa ja suunnittelutyökaluja monen kurssin harjoituksissa.

Laakkonen, Rauma, Ikonen ja Pyrhönen (2006: 1–4) ovat toteuttaneet FPGA-pohjaisen alustan moottorin säädön ja digitaalisen signaalinkäsittelyn algoritmien opetukseen. FPGA:n käyttö tarjoaa joustavan ja tehokkaan tavan algoritmien toteutukseen. Aiemmin käytetty DSP on korvattu sulautetulla prosessorilla, joka voidaan ohjelmoida C-kielellä. Opetuskäyttöön alusta tarjoaa valmiita kirjoittajien tekemiä VHDL-moduuleita. Vaatimuksena alustalle oli joustavuus, halpa hinta, korkea laskentateho, modulaarisuus ja ohjelmoinnin helppous. Alustan laitteisto koostuu Xilixin VirtexII-piiriin perustuvasta säätöpiiristä ja tehopiiristä. Opiskelijoille valmiit VHDL-moduulit on tarkoitettu esimerkiksi taajuusmuuttajan toteuttamiseen. Osaa moduuleista on pakko käyttää, kuten suojaus ja tyhjennysmoduulia, muut käytettävät moduulit opiskelijat voivat suunnitella itse. Alustan sisäinen kommunikointi tapahtuu OKITO-väylällä, jossa on mahdollista määritellä väylän nopeus, leveys sekä moduulien ja rekistereiden määrä. Ulkoisen käyttöliittymän kanssa kommunikointi voidaan toteuttaa RS232:lla. Moottorinsäätöalgorit-

mit voidaan toteuttaa C-kielellä käyttäen MicroBlaze prosessoria, tai VHDL:llä käyttäen piirin logiikkaa. Kirjoittajien mukaan alusta on sopiva VHDL:n opettamiseen moottorinsäätöalgoritmien kanssa.

Matilaisen, Salmisen ja Hämäläisen (2014: 37–42) mukaan isot harjoitukset kannattaa jakaa pakollisiin pieniin viikkoharjoituksiin, joista on mahdollista saada bonuspisteitä. Automaattiset testipenkit ja aloitusesimerkit ovat hyödyllisiä opetuksessa. SoC-laitteet ovat monimutkaisia integroitujapiirejä (IC), jotka sisältävät erilaisia toiminnallisia elementtejä käyttäen heterogeenisia IP-lohkoja, kuten ohjelmoitavia prosessoreita, muisteja, kiihdyttimiä ja useita muita ulkopuolisia liityntöjä. Tällaisen laitteen suunnitteluprojekti on laaja ja haasteellinen. Perustuvanlaatuinen ongelma on miten perehdyttää opiskelijat tällaisen systeemin suunnitteluun. Harjoitusten tarkoituksena on opettaa tärkeimmät konseptit ja vaiheet systeemin suunnittelussa, erityisesti IP-lohkojen uudelleenkäyttöä, sovellusalustaperusteista suunnittelua ja laitteisto/ohjelmisto yhteissuunnittelua. IP-lohkojen uudelleenkäyttöön käytetään avoimeen lähdekoodiin perustuvaa Kactus-ohjelmaa, joka suorittaa kelpoisuuden tarkastuksen ja pystyy generoimaan rakenteellisen VHDL-kuvauksen, C tunnisteet ja suorittamaan projektin syntetisoinnin. Kirjoittajien mukaan uusien työkalujen käytön opettelu vie aikaa, joten perustyökalujen kuten ModelSim simulointiohjelman käyttö tulisi säilyä samana peräkkäisissä kursseissa, he esittelivät ainoastaan Kactus2-ohjelman uutena ohjelmana. Heidän mukaan opiskelijat olivat tyytyväisiä ja pystyivät käyttämään työkaluja monimutkaisia järjestelmiä suunniteltaessa. Valmiiden IP-lohkojen käytössä voi esiintyä ongelmia. Tämä kuvastaa tosielämän suunnitteluskenaariota, jota ei tulisi tapahtua sattumalta. Kunnollinen kvalifiointi, ymmärrys ja IP-lohkojen uudelleenkäytettävyyden mahdollisuus tarvitaan ennen kuin niitä voidaan sujuvasti käyttää opetuksessa.

FPGA tarjoaa opetuksessa erilaisia niin analogisen kuin digitaalisen elektroniikan käyttömahdollisuuksia. Sen käyttö opetuksessa on hyödyllistä sovelluksissa missä tarvitaan tehokasta laskentaa, kuten robotiikassa, sulautetuissa järjestelmissä ja bioinformatiikassa. FPGA-pohjaisen järjestelmän suunnittelu vaatii pohjatietoja elektroniikan ja tietotekniikan opinnoista, kuten suunnittelukielistä, syntetisoinnista, suunnitteluohjelmista, rajapinnoista jne. (Skliarova, Sklyarov, Sudnitson & Kruus 2014: 460–469.)

Skliarova ym. (2014: 460–469) esittelevät artikkelissaan kaksi FPGA-järjestelmien opetusmetodia. Ensimmäisessä opetusvaiheet esitellään ja selitetään alhaalta-ylös- menetelmällä käyttäen apuna esimerkkejä yksinkertaisista ja monimutkaisista järjestelmistä. Kirjoittajien mukaan oppiminen on tehokasta, jos on riittävä määrä teoreettisia ja käytännöllisiä luentoja. Lisäksi FPGA-järjestelmistä on hyötyä muillekin kursseille, joissa toteutukset ovat monimutkaisia. Toinen metodi mahdollistaa monimutkaisten projektien toteuttamisen suhteellisen nopeasti. Siinä jokainen suunnitelma suoritetaan opiskelijoiden toimesta erillisenä osana. Opiskelijat toteutettavat osat selitetään tarkasti, ja heidän tulee toteuttaa ydinkomponentit ja yhdistää ne valmiiksi annettuihin komponentteihin, joista saadaan testattava FPGA-sovellus. Tehokkaiden ja luotettavien digitaalisten järjestelmien kehitys vaatii laitteiston ja ohjelmiston yhteissuunnittelua ja simulointia. Opiskelijoiden harjoitustyönä toteutettiin laitteisto/ohjelmisto-yhteistoteutus, johon kuului tietokoneella oleva ohjelma, joka on vuorovaikutuksessa FPGA-kehitysalustan kanssa. FPGA:ta käytetään tehokkaan lajittelualgoritmin toteutukseen. He käyttivät toteutuksessa toista opetusmetodia, jossa opiskelijoille annettiin osa tarvittavista komponenteista valmiina ja osan he joutuivat tekemään itse. Tämänkaltainen toteutus sallii heidän mukaan projektin toteutuksen monipuolisuuden ja estää ratkaisujen kopioimisen. Kirjoittajien käyttämän opetusmetodin avulla yhä useammat ovat saaneet projektin suoritetuksi samalla kun arvosanat ovat parantuneet, vaikka projektit ovat olleet haastavampia. Lisäksi heidän mukaan ensimmäisen vuoden opiskelijoille piiritoteutusten tekeminen on tehokkaampi tapa oppia, kuin pelkästään perinteisellä tavalla. Taulukossa 2 esitetään tavanomaisten FPGA-laboratorioiden toteutukset. Lisäkomponenteilla/työkaluilla tarkoitetaan omaa suunnitteluohjelmaa tai valmiita laajennuskortteja ja omilla kytkennöillä mahdollisuutta kytkeä omia komponentteja, kuten ledejä FPGA-kehitysalustalle.

Taulukko 2. Tavanomaisten laboratorioiden toteutukset.

Viite	Toteutus	FPGA	Lisäkomponentit/työkalut	Kehitysympäristö	Omat kytkennät
Schneider ym. 2005	Tavanomainen opetuslaboratorio	Xilinx Virtex2 Pro kehitysalusta (15 kpl)	Reealiaikakäyttöjärjestelmä	Xilinxin suunnitteluohjelmisto	Mahdollisuus komponenttien kytkentöihin GPIO:n kautta (160 nastaa)
Quintas ym. 2005	Tavanomainen opetuslaboratorio	APEX EP20K100EQC240-2X (Altera)	Hypermediasovellus. Laajennuskortti, jossa mm. LCD-näyttö. Lisäksi virtuaalinen logiikka-analysaattori	Quartus2 suunnitteluohjelmisto	Mahdollisuus komponenttien kytkentöihin (3 x 64 nastaa)
Mahmoodi ym. 2012	Tavanomainen opetuslaboratorio	Altera DE2-115 t-pad kehitysalusta	Kamera ja LCD-kostetusnäyttö	Quartus2 suunnitteluohjelmisto	Mahdollisuus komponenttien kytkentöihin GPIO:n kautta
Donzellini ym. 2012	Tavanomainen opetuslaboratorio	Altera DE2-kehitysalusta	Deeds suunnittelualusta	Quartus2 suunnitteluohjelmisto ja Deeds suunnittelu-alusta	DE2-kehitysalustan GPIO
Balid ym. 2013	Tavanomainen opetuslaboratorio	Spartan 3E kehitysalusta	Erilaisia laajennuskortteja DC-moottorin säätöön, lämpötilan mittaukseen ja valon intensiteetin mittaukseen	LabVIEW-pohjaisella ohjelmalla tapahtuva graafinen ohjelmointi	Kehitysalustan GPIO
Hocenski ym. 2013	Tavanomainen opetuslaboratorio	Spartan3 XC3S200 kehitysalusta	Graafinen käyttöliittymä tietokoneella	Visual Basicilla toteutettu graafinen käyttöliittymä	Kehitysalustan GPIO
Karyano ym. 2013	Tavanomainen opetuslaboratorio	Altera DE1 kehitysalusta	Ei lisäkomponentteja	Alteran suunnitteluohjelmat	Kehitysalustan GPIO
Brekken ym. 2006	Tehoelektroniikan opetuslaboratorio	Xilinx Spartan3 kehitysalusta	Kokosilta-piiri, ajuri-piiri, kuorma ja liitinlevy	Xilinxin suunnitteluohjelma	Kuormaan mahdollista liittää esimerkiksi DC-moottori
Vainio ym. 2010	Tavanomainen opetuslaboratorio	Altera DE2-kehitysalusta	Ei lisäkomponentteja	Quartus2 suunnitteluohjelmisto	Kehitysalustan GPIO
Kastelan ym. 2012	Tavanomainen opetuslaboratorio	FPGA-piiri ja ARM-prosessori sekä muita komponentteja	Ei lisäkomponentteja	Nimeämätön sunnitteluohjelmisto	Alustan GPIO

Viite	Toteutus	FPGA	Lisäkomponentit/työkalut	Kehitysympäristö	Omat kytkennät
Haba 2014	Tavanomainen opetuslaboratorio	Xilinxin kehitysalusta	Ei lisäkomponentteja	Xilinxin suunnitteluohjelmisto	Kehitysalustan GPIO
Laakkonen ym. 2006	Digitaalisen signaalin-käsittelyn ja moottorin ohjauksen laboratorio	Xilinx Virtex2-kehitysalusta	Teholauta, joka sisältää kolme vaihetasasuuntaajaa ja kolme IBGT transistoria	Xilinxin suunnitteluohjelmisto sekä C-kääntäjä	Mahdollisuus kiinnittää sähkömoottori alustaan
Matilainen ym. 2014	SoC-järjestelmien laboratorio	Altera DE2-kehitysalusta	Avoimen lähdekoodin Kactus2-ohjelma	Quartus2 suunnitteluohjelma ja Kacktus2	Kehitysalustan GPIO
Skliarova ym. 2014	SoC-järjestelmien laboratorio	Xilinxin kehitysalustat (Spartan-3E, Spartan-6, Artix-7)	Tietokone, joka kommunikoi FPGA:n kanssa UART:n välityksellä (full-duplex)	Xilinxin suunnitteluohjelma	Kehitysalustalla olevat valmiit komponentit ja GPIO

4. LASKUHARJOITUKSET JA HARJOITUSTYÖ

Tässä kappaleessa kerrotaan miten digitaalitekniikan opetus toteutetaan Vaasan yliopistossa, tarkasteltava on kurssi AUTO1010 digitaalitekniikan perusteet keväällä 2014. Kurssi toteutettiin luennoilla ja laskuharjoituksilla, jolloin oli myös tietystä luentoalueesta koostuva mikrotentti, sekä erillisestä harjoitustyöstä jossa tarkoituksena oli toteuttaa valourut-sovellus käyttäen FPGA:ta. Harjoitustyöstä esitellään siinä tarvittavat komponentit ja toiminnot.

4.1. Laskuharjoitukset

Laskuharjoitukset digitaalitekniikan perusteet kurssilla etenevät perusteista alkaen. Ensimmäisissä 1-2 harjoituksessa käydään lävitse digitaalitekniikan peruskomponentteja, kuten veräjiä ja sievennetään funktioita Karnaughin karttojen avulla. Harjoituskertojen edetessä otetaan käyttöön VHDL-kieli. Ensimmäiset VHDL-ohjelmointitehtävät ovat peruskomponenttien toteuttamista. Esimerkiksi keväällä 2014 ensimmäinen ohjelmointitehtävä oli ekvivalenssin toteuttaminen käyttäen AND, OR ja NOT veräjiä (Alander 2015). Toteutusten toimivuus testataan harjoituksissa simuloinnilla. Ohjelmoimiseen ja simulointiin käytetään Alteran Quartus ohjelmistoa. Harjoituksissa käytetään Alteran DE2-kehitysalustaa, jolle toteutetaan ohjelmoitu digitaalinen logiikka. Samaa kehitysalustaa käytettiin myös harjoitustyössä. Lisäksi harjoituksissa käytettiin koekytkentälevyjä johon toteutettiin erilaisia kytkentöjä käyttäen ledejä, vastuksia, TTL-piirejä sekä kytkentäjohtoja. Tarkoituksena oli opettaa opiskelijoille yksinkertaisten kytkentöjen tekemistä koekytkentälevylle, sekä FPGA:n käyttöä näiden ledien ja TTL-piirien ohjaamiseen kehitysalustalla olevien ledien sijasta.

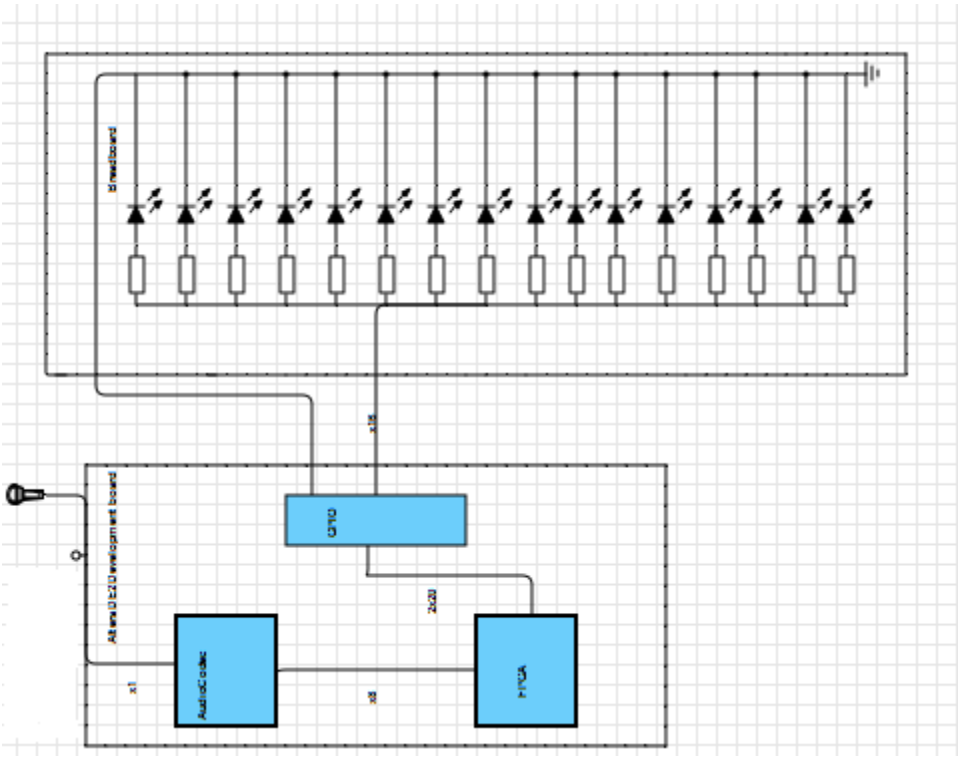
4.2. Harjoitustyö

Keväällä 2014 AUTO1010-kurssin harjoitustyön aiheena olivat valourut. Valourkuihin (Kuva 24 & 29) kuuluivat 16 lediä ja vastusta, koekytkentälevy, kytkentäjohtoa, Altera DE2-kehitysalusta sekä kuulokemikrofoni. Tarkoituksena oli tehdä toteutus jossa ledit

ja vastukset kytketään koekytkentälevylle joka edelleen kytkentäjohdolla yhdistetään kehitysalustalla olevalle GPIO:lle. Kuulokemikrofoni kytkettiin kehitysalustalle käyttäen siinä olevaa Wolfson WM8731 / WM8731L Audio CODEC:ia. Opiskelijoiden oli tarkoitus toteuttaa valourkujen ledien ja vastusten kytkentä koekytkentälevylle, liittää se kehitysalustalle, sekä toteuttaa 5 bittinen pulssinleveysmodulaatio (PWM, joka käydään lävitse seuraavassa aliotsikossa), jolla ledien valaistuksen intensiteettiä voitiin säädellä. Kuulokemikrofonista tuleva äänisignaali ohjaa ledejä äänen voimakkuuden mukaan. PWM:llä ei vaikuteta siihen kuinka monta lediä valaistaan, ainoastaan valaistuksen intensiteettiin. Harjoitustyötä varten opiskelijoille annettiin valmis projektipohja, joka sisälsi viisi VHDL-moduulia. Moduuleissa oli alustettu WM8731 toiminta, kommunikointi FPGA-piirin kanssa (I^2C), sekä tarvittavat laskutoimitukset.

Keväällä 2014 harjoitustyö eteni vaiheittain seuraavasti:

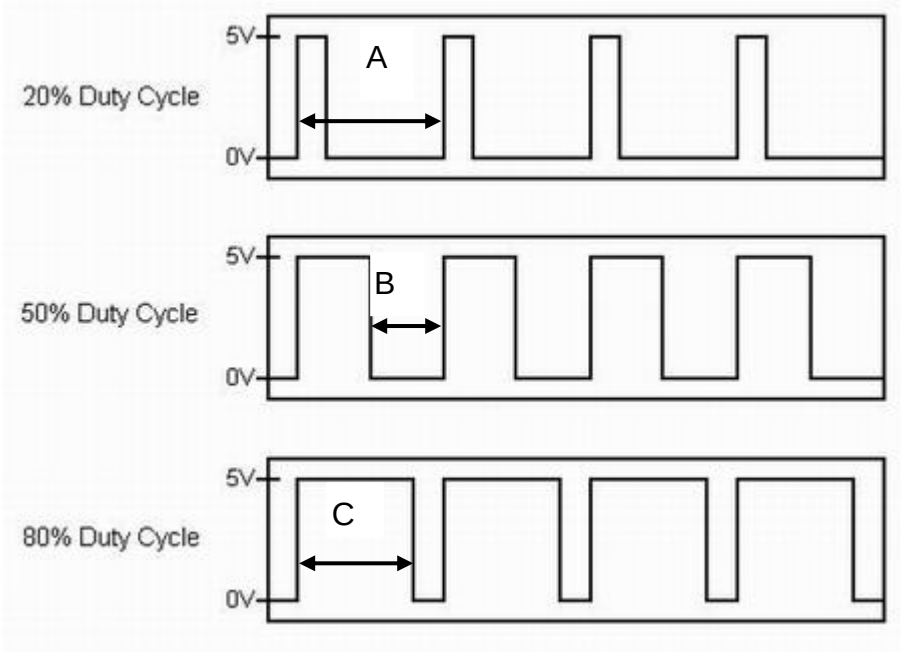
- valourkujen toteutusta suunniteltiin 2-3 harjoituskerralla
- jokainen ryhmä (1-3 henkilöä) kirjoitti alkuraportin harjoitustyöstä kurssin opettajalle, jossa tuli selittää valourkujen toiminta lohkokaavion avulla
- harjoitustyön ohjauskerroilla aloitettiin VHDL-kielisen PWM-moduulin toteuttaminen ja liittäminen valmiiksi annettuun projektipohjaan, lisäksi koekytkennästä tuli tehdä suunnitelma
- viimeisellä ohjauskerralla rakennettiin ledimatriisi koekytkentälevylle ja testattiin koko toteutuksen toiminta, samalla suoritettiin palautekysely käyttäen liitteen 1 kyselylomaketta
- lopuksi tuli tehdä kurssin opettajalle palautettava harjoitustyöraportti.



Kuva 24. Valourut. Koekytkentälevylle asetellaan ledit ja vastukset (16 kpl), jotka kytketään kytkinjohdolla DE2-kehitysalustan GPIO:hon. Kuulokemikrofoni kytkettiin kehitysalustalla olevan audio codeciin, joka muuttaa äänisignaalin arvot digitaaliseksi. FPGA-piiri ohjaa ledien valaistusta.

4.2.1. Pulssinleveysmodulaatio

Pulssinleveysmodulaation (PWM) avulla voidaan rajoittaa jännitteen kulkua siten, että se pääsee kulkemaan tietyn ajan yhdessä ennalta määritellyssä jaksossa (Kuva 25). Jännite vaihtaa arvoaan 5 V:sta 0 V:iin hyvin nopeasti, jolloin sillä voidaan rajoittaa jännitteen määrää tiettyyn osaan maksimijännitteestä alipäästö suodattamalla. Digitaalisesti jännitetasot voidaan merkitä siten, että looginen 1 vastaa 5 V:a ja 0 V:a looginen 0.



Kuva 25. Pulssinleveysmodulaatio kanttiaaltona. Nuoli kohdassa A kuvaa yhden jakson pituutta eli periodia, nuoli kohdassa C jännitteen toiminta-aikaa (duty cycle) ja nuoli kohdassa B aikaa jolloin jännitteen arvo on nolla (off cycle). Digitaalisesti jännitteen tasoa 5 V vastaa looginen 1 ja 0 V looginen 0. Kuvassa esitetään 20 %, 50 % ja 80 %:n pulssinleveydet. (National Instruments 2010).

Haluttu pulssinleveys (duty cycle) voidaan laske kaavalla

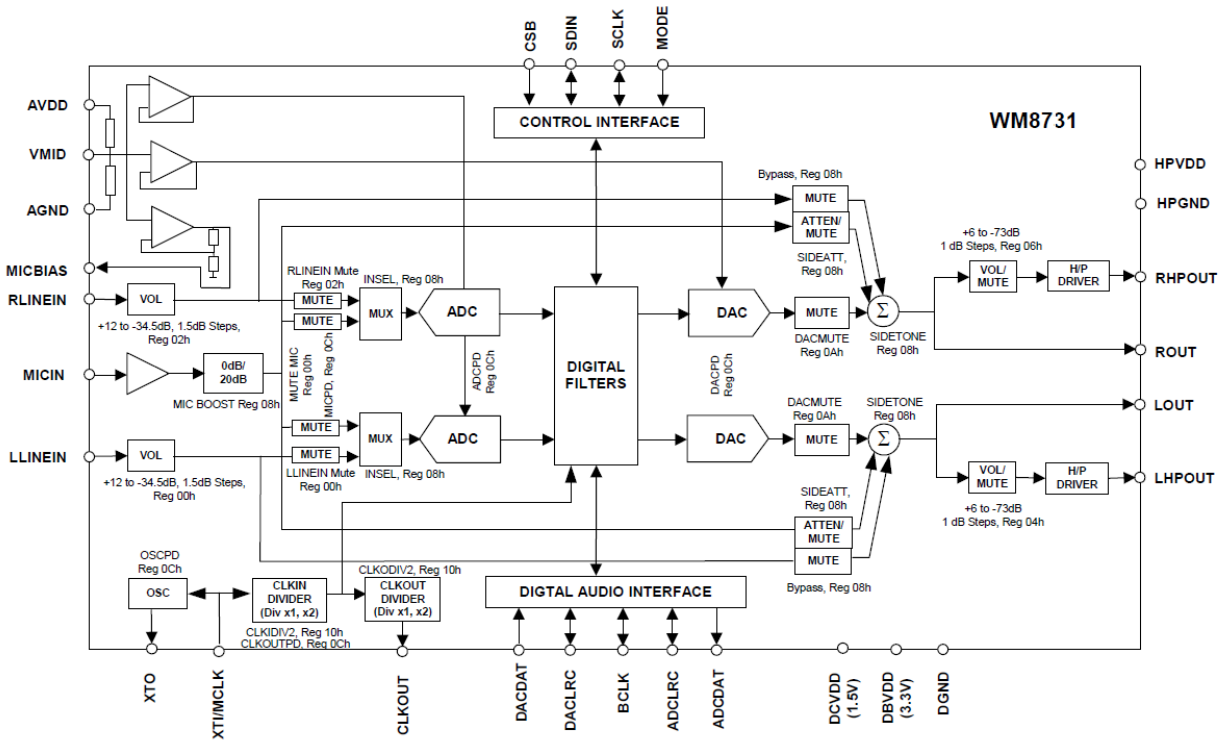
$$D = \frac{\text{PL}}{\text{Periodi}} * 100\% \qquad (1)$$

missä *D* on pulssinleveys (duty cycle), PL on toiminta-aika ja Periodi jakson pituus. Esimerkiksi jos periodin pituus on 100 ms (0.1 s) ja haluttu pulssinleveys 80 %:a (0.08 ms) periodin pituudesta *D* = (80/100) x 100 %, josta saadaan arvoksi 80 %:ia. Harjoitustyössä opiskelijoiden tarkoituksena on PWM toiminto omaksi moduulikseen VHDL-kielellä osaksi valourkuja käyttäen VHDL:n rakennemallia (structural). PWM moduulin oli tarkoitus toimia päämoduulina rakennemallissa.

4.2.2. WM8731 Audio CODEC

Altera DE2-kehitysalusta sisältää pienitehoisen WM8731 stereokoodekin (Kuva 26) integroidulla kuulokeajurilla, joka on erityisesti suunniteltu kannettaviin audio laitteisiin. Se sisältää stereo- ja monoäänisisääntulot, sekä vaimennusominaisuudet ja ohjelmoitavan sisääntulon äänenvoimakkuuden säädön. Linja- ja mikrofonisisääntulot menevät koodekilla olevalle ADC-muuntimelle sekä linja- ja mikrofoniulostulot DAC-muuntimen kautta. Lisäksi koodekki sisältää kello-oskillaattorin, konfiguroitavan digitaalisen ääniliittymän sekä 2-3 linjaisen mikroprosessoriohjainliittymän.

Koodekin sisääntulona on monomikrofoni ja kaksi stereolinjaa. Linjasisääntuloilla on säädettävä välillä (+12, -34)dB äänitaso ja vaimennus. Mikrofonilla on välillä (-6, +34)dB säädettävä äänentaso. Koodekin sisältämä stereo-ADC-muunnin käyttää monibittistä ylinäytteistystä. ADC-muuntimen ulostulo on saatavilla digitaalisessa ääni- liittymäkohdassa. Muunnin sisältää lisäksi valinnaisen digitaalisen ylipäästösuotimen, jonka avulla päästään eroon ei-toivotusta äänisignaalin tasakomponentista (DC). Piirin sisältämä DAC-muunnin ottaa vastaan digitaalista dataa digitaalisesta ääni- liittymäkohdasta. DAC-muunnin voidaan ohjelmoida 32, 44.1 tai 48 kHz:ksi, ja se käyttää monibittistä ylinäytteistystä. Koodekissa on mahdollisuus käyttää eri näytteistystaajuuksia 8 kHz:stä aina 96 kHz:iin asti, lisäksi ADC ja DAC-muuntimet voivat toimia eri näytteistyksellä. WM8731 on suunniteltu vähän tehoa kuluttavaksi haittaamatta kuitenkaan sen suorituskykyä. Siinä on mahdollisuus sammuttaa tietyt osat piiristä ohjelmallisesti. Piirissä on mahdollista valita ohjelmallisesti yhdeksän erilaista tehonsäästötapaa. Altera DE2 kehitysalustalla olevan FPGA-piirin (Cyclone II EP2C35F672C6) kanssa kommunikointi tapahtuu I^2C-väylän kautta. (Altera 2014: 41; Wolfson microelectronics 2012: 1, 20.) Harjoitustyössä opiskelijat saavat valmiin projektipohjan, jossa WM8731 on valmiiksi alustettu. Tähän pohjaan heidän tulee integroida aiemmin mainittu PWM-moduuli.

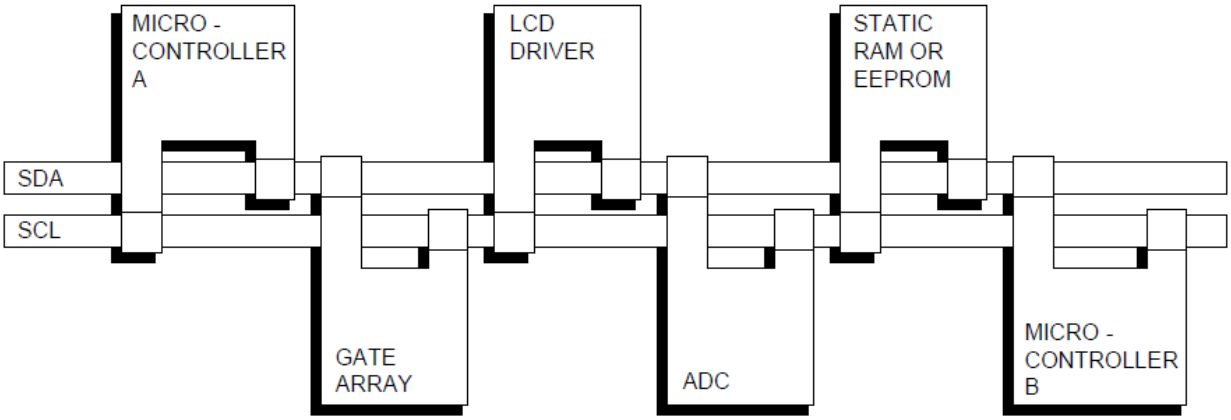


Kuva 26. Funktionaalinen lohkokaavio WM8731 Audio CODEC:sta. Alareunassa (digital audio interface) olevat sisääntulot DACDAT, DACLRC, BCLK, ADCLRC ja ADCDAT ovat kello ja datasignaaleja, joilla FPGA kommunikoi codecin kanssa. Yläreunassa (control interface) olevat SDIN ja SCLK ovat I^2C-väylän data ja kellosignaalit. (Wolfson microelectronics 2012: 21; Altera 2014: 41).

4.2.3. I^2C-väylä

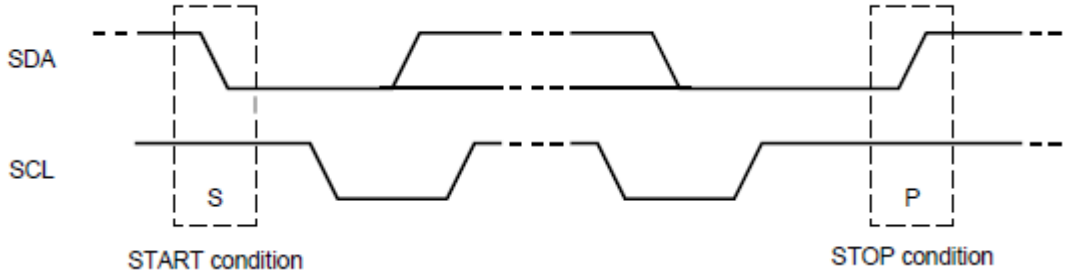
I^2C-väylä on NXP Semiconductorsin kehittämä kaksisuuntainen ja -linjainen väylä (Kuva 27). Väylät ovat sarjallinen dataväylä (SDA) ja sarjallinen kelloväylä (SCL). Jokaisella väylään kytketyllä laitteella on oma uniikki osoite. Väylällä on voimassa isäntä/orja-periaate, jossa isäntälaite voi toimia lähettäjänä ja vastaanottajana, muut laitteet toimivat lähettäjänä tai vastaanottajana riippuen niiden toimintatarkoituksesta. Väylällä on datan yhteentörmäysten ja siirron kontrolli, joten siinä voi samanaikaisesti toimia useampi isäntälaite. Datan siirto toimii kaksisuuntaisena 8-bittisenä tiedonsiirtona,

jota on mahdollista kasvattaa 100 kbit/s normaalitilassa, nopeassa toimintatilassa 400 kbit/s ja maksimisssaan 3.4 Mbit/s asti.

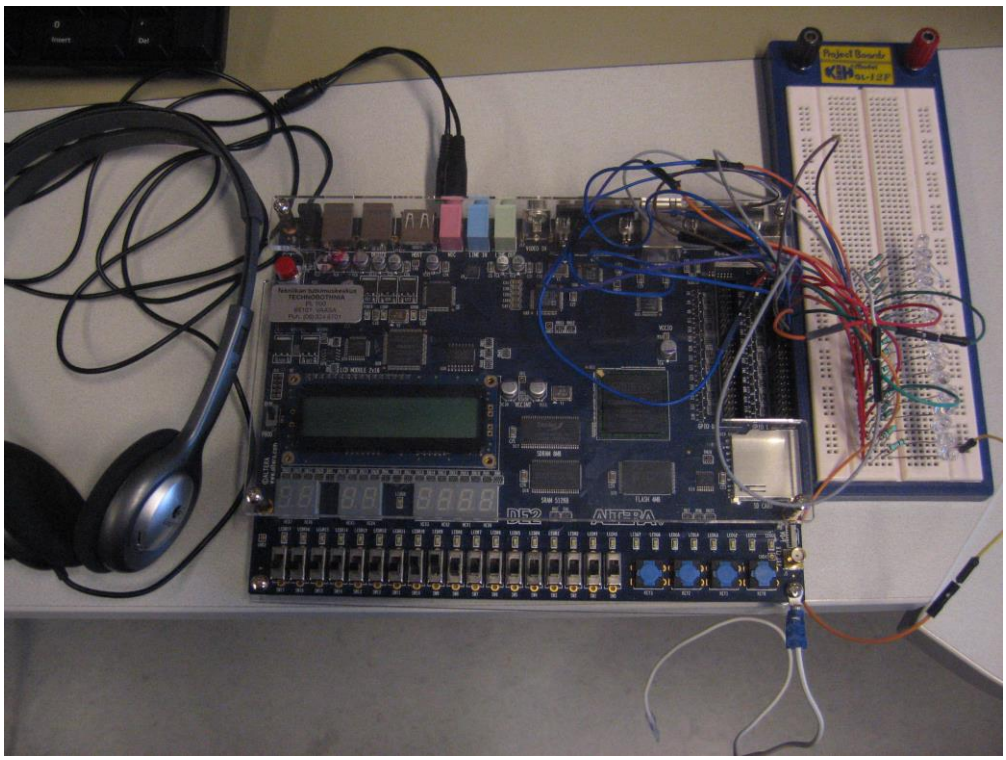


Kuva 27. I^2C-väylä. Mikrokontrolleri A toimii tässä isäntälaitteena ja muut komponentit orjalaitteina. Mikrokontrolleri A pystyy lähettämään halutun määrän dataa mikrokontrolleri B:lle. Mikrokontrolleri A:n halutessa dataa mikrokontrolleri B:ltä se osoittaa sen osoitetta ja pyytää sitä lähettämään. Molemmissa tapauksissa mikrokontrolleri A pystyy halutessaan lopettamaan datan siirron. Kommunikointiin tarvitaan kaksi väylää SDA ja SCL. Mikrokontrolleri A huolehtii SCL-väylän generoinnista. (UM10204 2014: 7).

Data- ja kellolinja ovat molemmat kytketty positiiviseen käyttöjännitteeseen ylösvetovastuksen tai virtalähteen kautta. Väylän ollessa vapaa molemmat linjat ovat ylätilassa. Väylän dataliikenne alkaa aina aloitustilasta ja loppuu lopetustilasta (Kuva 28). Aloitustila tapahtuu datalinjan muuttuessa korkeasta matalaan tilaan kellolinjan pysyessä korkeassa tilassa. Lopetustila tapahtuu datalinjan muuttuessa matalasta korkeaan tilaan kellolinjan ollessa ylhäällä. Isäntälaite generoi aina aloitus- ja lopetustilan. Aloitustilan jälkeen väylä on ruuhkainen ja lopetustilan jälkeen avoin. Väylällä olevien laitteiden tulee sisältää liityntälaitteisto, jolla ne huomaavat aloitus- ja lopetustilat. (UM10204 2014: 3, 6–10.)



Kuva 28. I2C-väylän aloitus- ja lopetustilat. (UM10204 2014: 9).



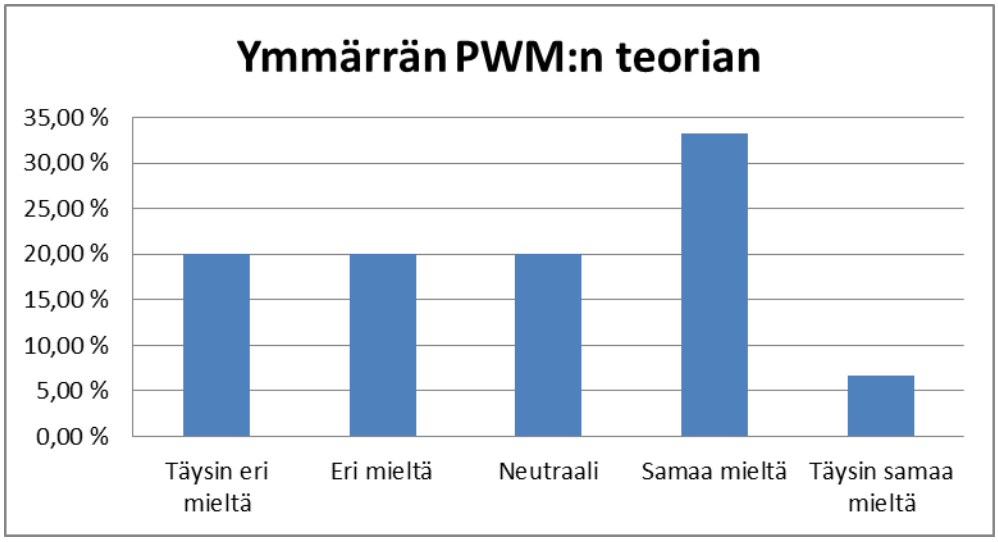
Kuva 29. Valourut. Oikealle koekytkentälevylle on aseteltu ledit ja vastukset. Kehitysalustan (DE2) GPIO:sta menee kytkentäjohdot vastuksille ja yksi maadoitusjohto. Kuulokemikrofoni on kytketty suoraan alustan mikrofonipaikkaan.

5. KYSELY

Tässä kappaleessa käydään läpi AUTO1010 Digitaalitekniikan perusteet-kurssin harjoitustyöstä tehty kysely (Liite 1) ja sen tulokset. Palautekysely laadittiin yhteistyössä TkT Birgitta Martinkaupin kanssa. Kyselyssä oli 7 monivalintakysymystä ja 4 sanallista kysymystä. Vastaukset saatiin 15 opiskelijalta. Kyselyn tulokset esitellään myös konferenssiartikkelissa (Karhu, Alander & Nurmi 2015; Liite 2), joka julkaistaan CSEDU-konferenssissa Lissabonissa.

PWM

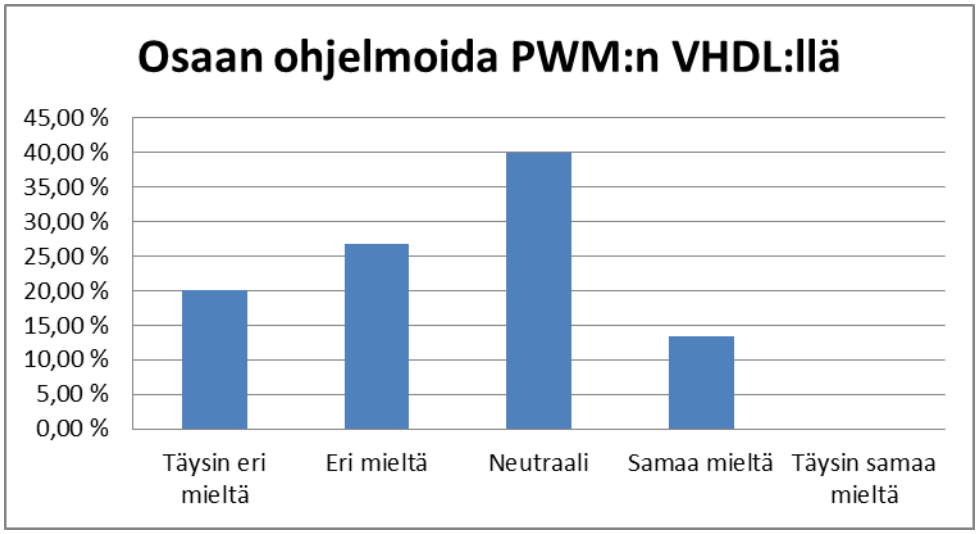
Ensimmäinen monivalintakysymys (Kuva 30) koski PWM:n teorian ymmärtämistä. Ainoastaan yksi henkilö (7 %) sanoi ymmärtävänsä täysin teorian ja 33 % (5 henkilöä) ymmärtävänsä sen hyvin. Yhdellä luentokerralla asiaa oli käsitelty ja yksi harjoituskerta sisälsi tehtävän toteuttaa 4 bittisen PWM:n. Asian ei pitäisi olla monimutkainen, ja syynä ymmärtämättömyyteen voi olla luennolta/laskuharjoituskerralta poissaolo.



Kuva 30. Vastausprosentit kysymykseen: ”*Ymmärrän PWM:n teorian*”.

Toisena samaan aiheeseen liittyvänä monivalintakysymyksenä (Kuva 31) oli kyky ohjelmoida VHDL-kielinen PWM-toteutus. Kaksi henkilöä (13 %) sanoi osaavansa ohjelmoida VHDL-kielisen PWM-toteutuksen. Niillä joilla oli ongelmia ymmärtää sen

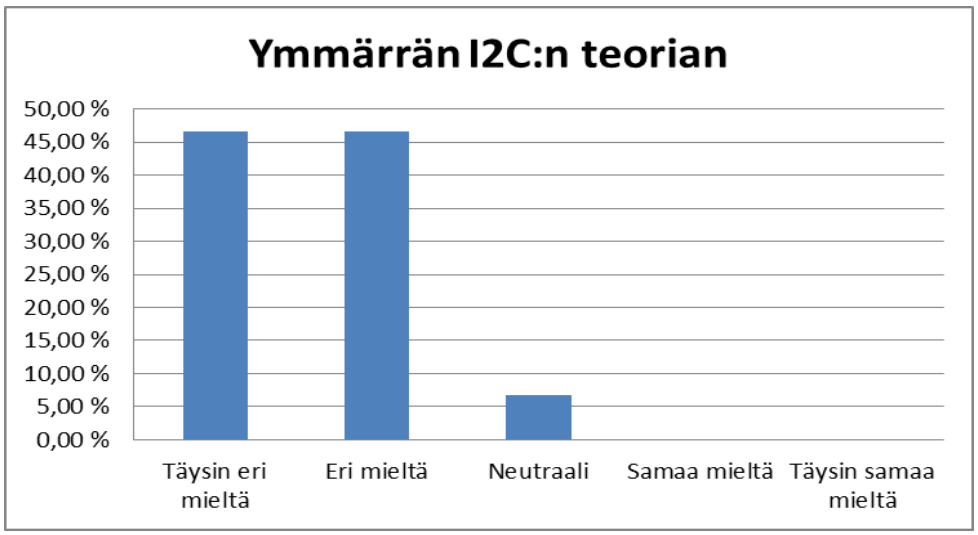
toiminta, oli vaikeuksia toteuttaa se. Ohjelmointitaitojen puute (VHDL) ja ongelmat ohjelman käytössä saattoivat olla esteenä toteutuksen tekemisessä.



Kuva 31. Vastausprosentit kysymykseen: ”*Osaan ohjelmoida PWM:n VHDL:llä”*.

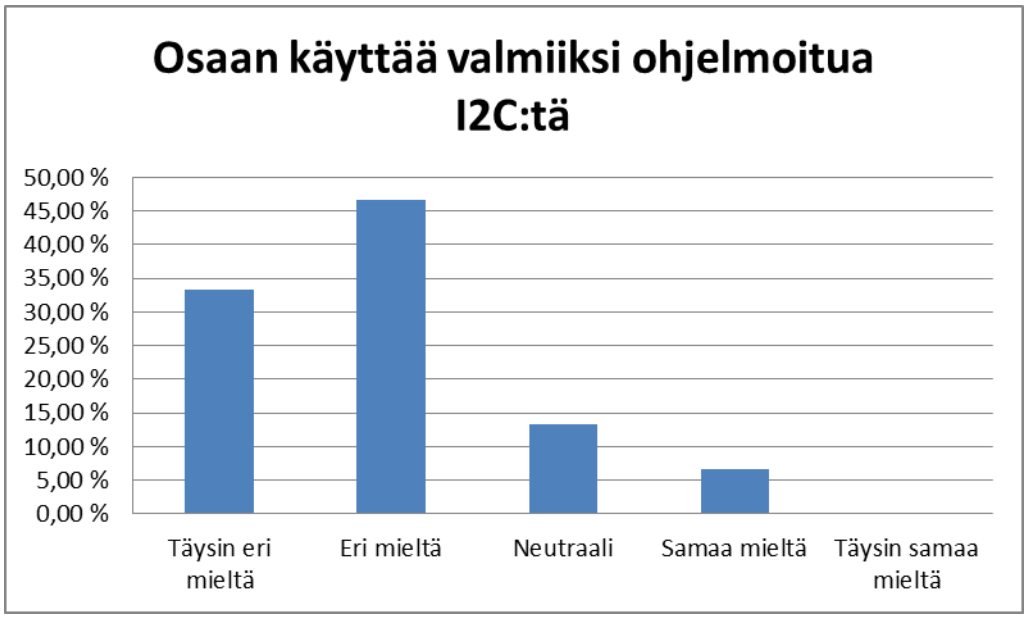
I^2C

Kolmas monivalintakysymys (Kuva 32) koski I^2C-väylää, jolla hoidetaan tiedonsiirto FPGA:n ja koodekin välillä. Luennoilla asiasta ei ollut puhuttu, eikä laskuharjoituksissakaan ollut tehtäviä siihen liittyen. Muillakaan kursseilla aihetta ei ollut käsitelty. Tämän vuoksi suurimmalle osalle opiskelijoista asia oli tuntematon.



Kuva 32. Vastausprosentit kysymykseen: ”*Ymmärrän I^2C:n teorian”*.

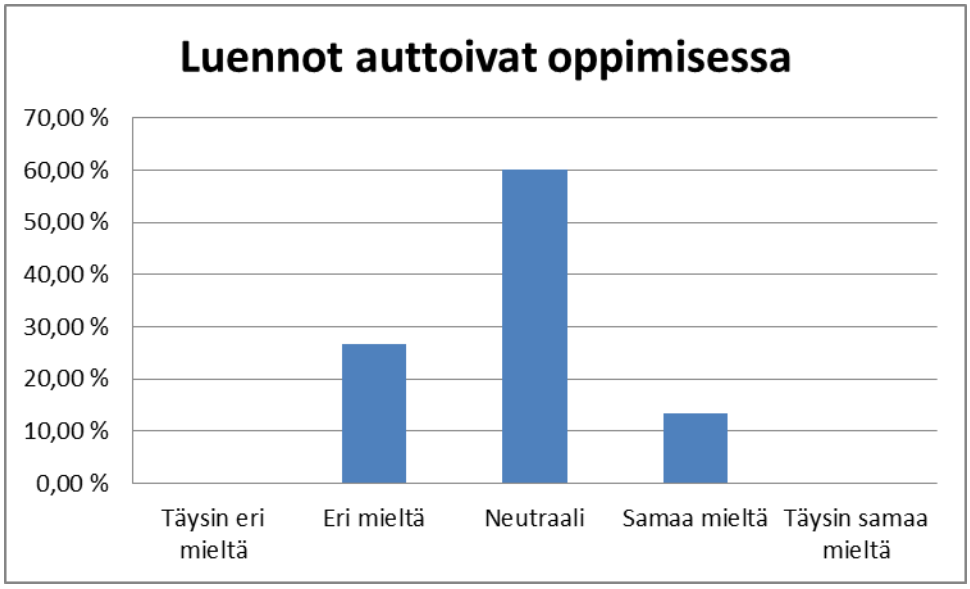
Neljäs monivalintakysymys (Kuva 33) koski valmiiksi annetun projektipohjan käyttöä, vaikka kysymys koski valmiiksi ohjelmoitua I^2C-väylää. Valmis projektipohja oli tarkoitus muuttaa omaksi projektiksi käyttäen Altera Quartus-ohjelmaa. Projektipohja, joka sisälsi viisi VHDL-moduulia, oli toteutettu VHDL:n rakennemallina. Luennoilla ja kurssin oppikirjassa (Grout 2008) käsiteltiin rakennemalli, mutta harjoituksissa ei ollut tehtävää tähän liittyen. Osa opiskelijoista ei ollut ennen toteuttanut rakennemallia. Opiskelijat eivät tunteneet audiokoodekin toimintaa, joka oli määritelty projektipohjassa. Valmiin projektipohjan käytössä oli ongelmia, koska opiskelijat eivät tunteneet koodekin (ja väylän) ominaisuuksia.



Kuva 33. Vastausprosentit kysymykseen: *”Osaan käyttää valmiiksi ohjelmoitua I^2C:tä”*.

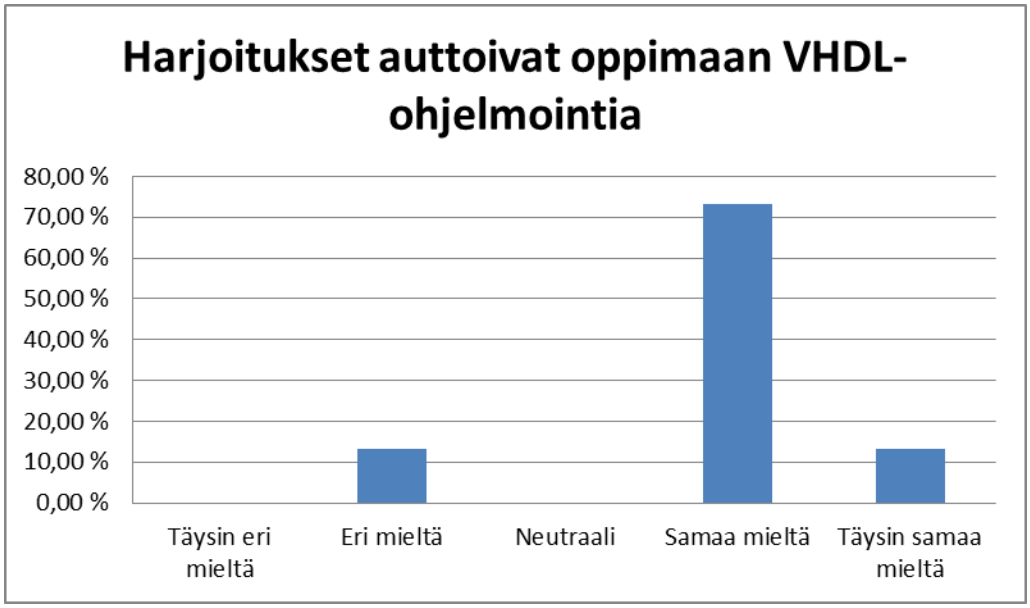
Luennot ja harjoitukset

Viides monivalintakysymys (Kuva 34) koski luentoja. Suurin osa (60 %) vastanneista antoi neutraalin vastauksen. Luennoilla asiat käytiin lävitse perusteista alkaen ja osa luentokerroista käsitteli pelkästään VHDL-kieltä. Opiskelijat, jotka kävivät aktiivisesti luennoilla, sekä kertasivat opetetut asiat, saivat enemmän hyötyä niistä.



Kuva 34. Vastausprosentit kysymykseen: *”Luennot auttoivat oppimisessa”*.

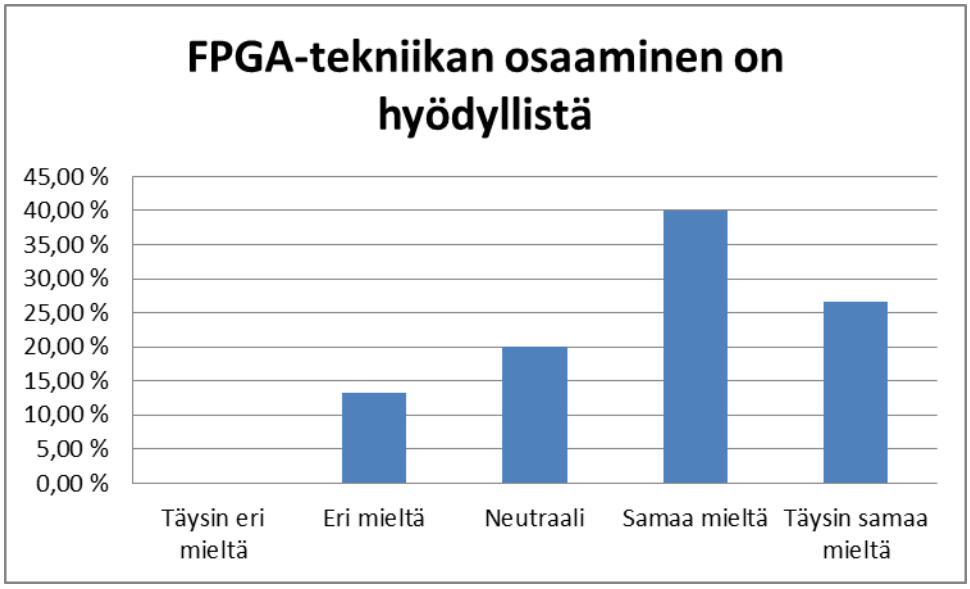
Kuudes monivalintakysymys (Kuva 35) koski laskuharjoituksia ja niiden ohjelmointitehtäviä. Enemmistö vastanneista (74 %) oli sitä mieltä, että harjoituksista oli hyötyä VHDL-kielen oppimiseen. Ohjelmointitehtävät alkoivat opastamalla opiskelijoita Alteran Quartus ohjelman käytössä. Opastus alkoi uuden projektin luonnista, tallennuksesta, uuden ohjelmointitiedoston (VHDL) luomisesta, kääntämisestä sekä toteutuksen simuloinnista käyttäen ohjelman Waveform-editoria. Jokaisessa laskuharjoituksessa toteutettiin edellä mainittu opastus ensimmäistä ohjelmointitehtävää suoritettaessa, koska suurin osa opiskelijoista halusi sitä. Harjoituksissa käytiin tarpeen vaatiessa jokainen ohjelmointitehtävän koodi lävitse ja tarvittaessa opastettiin tarkemmin eri kohdat. Osa opiskelijoista oli tehnyt ohjelmointitehtävät etukäteen, mikä auttoi heitä sisäistämään asiat jotka neuvottiin. Osalla opiskelijoista oli ongelmia ohjelman käytössä, jolloin he eivät saaneet tehtäviä suoritettua.



Kuva 35. Vastausprosentit kysymykseen: *”Harjoitukset auttoivat oppimaan VHDL-ohjelmointia”*.

FPGA-tekniikka

Viimeinen monivalintakysymys (Kuva 36) koski FPGA-tekniikan hyödyllisyyttä. Suurin osa (40 %) koki sen osaamisen hyödylliseksi. Luentojen ja laskuharjoitusten perusteella opiskelijoilla on käsitys hyödyntämismahdollisuuksista teknisissä sovelluksissa. Vastauksista voi päätellä, että he ymmärtävät digitaalitekniikan (FPGA) hyödyntämismahdollisuudet eri sovelluksissa.



Kuva 36. Vastausprosentit kysymykseen: *”FPGA-tekniikan osaaminen on hyödyllistä”*.

Sanalliset kysymykset

Ensimmäiseen sanalliseen kysymykseen (Liite 1) opiskelijat vastasivat osaavansa/ymmärtävänsä ohjelmoinnin perusteita, perustiedot VHDL:stä ja FPGA-piireistä, numerojärjestelmistä sekä karkean periaatteen valourkujen toiminnasta. Toisessa kysymyksessä kaikki vastaajat sanoivat oppineensa VHDL-kielen ohjelmointia, kuten Kuva 34 osoittaa. Vastauksissa kolmanteen kysymykseen opiskelijat toivoivat oppivansa enemmän VHDL-kieltä ja sen soveltamista.

Harjoitustyön suoritus

Harjoitustyön ohjauskerroilla, sekä laskuharjoituksissa osalla opiskelijoista oli ongelmia Quartus-ohjelman käytön kanssa uuden projektin luomisessa, tallentamisessa, kääntämisessä, simuloinnissa ja toteutuksen asentamisessa kehitysalustalle. Ongelmia oli lisäksi koekytkentälevylle tehdyissä kytkennöissä, joita olivat ledien ja vastusten asettaminen, kytkentäjohdon käytössä ja kytkemisessä kehitysalustalle. Suurimmalla osalla opiskelijoista ei ollut aikaisempaa kokemusta elektronisten kytkentöjen tekemisestä, mikä todennäköisesti selitti ongelmat niiden teossa. Muutama opiskelija olisi halunnut, että

ohjauskertoja olisi ollut enemmän samanaikaisesti luentojen ja laskuharjoitusten kanssa. Suurin osa ryhmistä sai kuitenkin toteutettua toimivan sovelluksen valouruista.

6. POHDINTA

Palautekyselyssä vastaukset saatiin 15 opiskelijalta (~50 % kurssille osallistuneilta), vastausprosentti oli samansuuruinen Kumarin, Fernandon ja Panickerin (2013: 407) tekemässä samantapaisessa kyselyssä. Pienestä vastausprosentista johtuen opiskelijoille olisi varmasti kannattanut tehdä tämäntapaisia kyselyitä useammalla laskuharjoituskerralla.

Kyselytulosten perusteella voidaan sanoa, että AUTO1010 Digitaalitekniikan perusteet-kurssin laskuharjoitukset ja harjoitustyö ovat auttaneet opiskelijoita oppimaan VHDL-kieltä ja FPGA-tekniikkaa (Kuvat 35 & 36). Osalla opiskelijoista oli kuitenkin ongelmia laskuharjoituksissa ja harjoitustyössä VHDL toteutuksen tekemisessä. Aikaisempaa käyttökokemusta Alteran Quartus-ohjelmasta ei suurimmalla osalla ollut, jonka käyttö tuotti vaikeuksia, vaikka VHDL-kieli olisikin ollut tuttu. Luennoilla ei käyty lävitse ohjelman toimintaa, ainoastaan laskuharjoituksissa ja harjoitustyön ohjauskerroilla opastettiin sen käytössä.

Kurssin harjoitustyön aiheena ollut valourut oli laaja toteutus, joka piti sisällään kehitysalustan, kehitysalustalla olevan audiokoodekin (sisältäen I^2C-väylän), koekytkentälevyllä olevat ledit ja vastukset sekä kuulokemikrofonin. Tarkoituksena oli toteuttaa PWM-moduuli, joka liitettiin valmiiksi annettuun projektipohjaan. Moduulilla ohjattiin koekytkentälevyllä olevien ledien valaistuksen intensiteettiä. Kuvan 29 perusteella suurin osa opiskelijoista ymmärsi mistä PWM:ssä on kysymys. Kuitenkaan kovin moni ei osannut toteuttaa sitä VHDL:llä (Kuva 30), vaikka luennolla ja yhdellä laskuharjoituskerralla sitä oli käyty lävitse. Audiokoodekin toimintaan tarvittiin I^2C-väylä, joka hoiti kommunikoinnin kehitysalustan FPGA-piirin kanssa. Suurelle osalle opiskelijoista väylä oli tuntematon (Kuva 31), koska sitä ei käsitelty luennoilla, laskuharjoituksissa eikä muillakaan kursseilla. Harjoitustyön ensimmäisellä ohjauskerralla väylän ominaisuudet käytiin lävitse. Väylä oli valmiiksi ohjelmoitu opiskelijoille annettuun projektipohjaan. Koska väylän (ja audiokoodekin) ominaisuudet eivät olleet ennestään selviä oli heillä vaikeuksia ymmärtää miten se oli toteutettu projektipohjaan ja käyttää sitä (Kuva 32). Suurin osa opiskelijoista kuitenkin uskoi, että FPGA-tekniikan osaamisesta on hyötyä

tulevaisuudessa (Kuva 35). Koekytkentälevylle tehtävien elektronisten kytkentöjen tekeminen ei ollut suurimmalle osalle opiskelijoista ennestään tuttua, ja laskuharjoituksissa monet tekivätkin ensimmäistä kertaa niitä.

Kyselyvastausten perusteella voidaan sanoa, että erilliselle etälaboratoriolle/laboratoriotilalle ei ole tarvetta. Olisi myös mahdollista käyttää kolmansien osapuolten ohjelmistoja VHDL/Verilog toteutuksiin esimerkiksi Matlabin Simulink:a, kuten Athar, Siddiqi ja Masud (2012: 2766–2767) artikkelissaan esittelevät. O'Connorin (2008: 1, 18) lopputyössä esiteltiin FPGA laboratoriotila, jossa Simulink mallista voitiin Xilinxin liitännäistyökalulla tuottaa tarvittavat VHDL-tiedostot. Simulink mallin avulla käyttäjät voivat tuottaa VHDL koodia liitännäistyökalun avulla ilman ohjelmointikokemusta, sekä asentaa sen kehitysalustalle. Tällainen systeemi ei kuitenkaan opeta opiskelijoita ohjelmoimaan laitteistokuvauskieltä, mutta he joutuvat opettelemaan Matlabin ja sen eri osien käytön koodin tuottamiseen, vaikka saman ajan he voisivat käyttää itse kielen (VHDL/Verilog) opiskeluun. Kyselyvastausten perusteella parhaiten oppii itse tekemällä ohjelmointitehtäviä, esimerkiksi laskuharjoituksissa. Itse tehdylle opetusalustalle, kuten Donzellinin ja Pontan (2012: 1–4) kaltaiselle alustalle ei ole tarvetta, koska opiskelijoiden on hyvä oppia käyttämään laitevalmistajien ohjelmistoja, koska niitä käytetään myös teollisuudessa.

Kappaleessa kolme esitellyt FPGA-etälaboratoriot sopivat joissakin tapauksissa fyysisten laboratoriotilojen korvikkeeksi. Kuitenkin niiden rakentaminen, ylläpitäminen ja käyttö vaativat resursseja. Etälaboratoriot soveltuvat tapauksesta riippuen erilaisten mittausten tekemiseen ja alustan ohjelmoimiseen internetin välityksellä, kuten Drutarovsky ym. (2009: 55–56) esittelivät. Peruskurssien FPGA toteutukset lähtevät liikkeelle melko yksinkertaisista sovelluksista, joten erillisen etälaboratoriotilan rakentaminen ei tässä mielessä ole järkevää. Etälaboratorioissa kehitysalusta on erillisessä tilassa, jolloin opiskelijat eivät itse pääse liittämään siihen erillisiä komponentteja, eivätkä käyttämään siinä olevia kytkimiä. Harjoitustyön tekeminen osoitti opiskelijoiden oppivan, jos he pääsevät itse käyttämään kehitysalustaa sekä pystyvät tekemään siihen kytkentöjä ja näkevä toteutuksen heti, eikä esimerkiksi tietokoneen näytön välityksellä.

Harjoitustyössä ja osassa laskuharjoituksissa käytetty Altera DE2-kehitysalusta soveltuu hyvin peruskurssin opetusalustaksi, eikä erilliselle laboratoriotilalle ole tarvetta. Kehitysalustaa (hinta <100$) käyttääkseen opiskelija tarvitsee tietokoneen, sekä tarvittavat ohjelmat. Peruskurssien FPGA-toteutukset eivät ole sellaisia, että ne vaatisivat erillisiä tiloja esimerkiksi turvallisuuden takia. Kuten Vainio ym. (2010: 137–138, 140) kertoivat artikkelissaan, kannattaisi varmasti muissakin automaatiotekniikan kursseissa tulevaisuudessa käyttää kyseistä alustaa, koska opiskelijoille sen käyttö on peruskurssin jälkeen tuttua. Kellet (2012: 379) käyttää kaikissa sulautettujen järjestelmien kursseissa DE2-alustaa ja siinä olevaa NIOS2-prosessoria. Kyselytulosten perusteella opiskelijat oppivat VHDL-kieltä ja uskoivat, että FPGA:n osaaminen on hyödyllistä, johon varmasti kehitysalustan käyttö laskuharjoituksissa ja harjoitustyössä vaikutti positiivisesti. Erillisen laboratoriotilan rakentaminen voisi tulla kysymykseen silloin kun käsitellään suurempia jännitteitä, jolloin turvallisuuskin tulee ottaa huomioon. Esimerkiksi sähkömoottorin säätösovelluksiin voisi kehittää oman FPGA-pohjaisen alustan, kuten Laakkonen ym. (2006: 1–4) ovat toteuttaneet. FPGA-tekniikkaa voitaisiin käyttää digitaalisten säätimien toteutukseen, kuten Artigas, Barragan, Isidro, Navarro ja Lucia (2011: 55–60), jotka käyttivät FPGA-pohjaista alustaa tehomuuntimien digitaaliseen säädön opetukseen. Alusta sisälsi laitteiston ja ohjelmiston (SoC). Araujo ja Alves (2008: 42) painottavat artikkelissaan laboratoriolaitteiden (FPGA-kehitysalustat) käyttöä onnistuneeseen oppimiseen. Tällä tavalla heidän mukaan opiskelijat voivat kokeilla ratkaisujaan ja soveltaa taitojaan tehokkaalla tavalla. Näin ollen kehitysalustojen käytöstä on hyötyä tulevaisuudessakin laskuharjoitusten ja harjoitustyön yhteydessä.

Harjoitukset auttoivat kyselyn mukaan sisäistämään VHDL-kielen. Ellervee ym. (2008: 39) ehdottivat, että kielen opetus kannattaisi aloittaa rakenteellisella kuvaustavalla. Kelletin (2012: 379) mukaan laitteistokuvauskielen oppimisen vaikeutta ei pidä aliarvioida, koska rinnakkainen ohjelmointiparadigma vaatii opiskelijoilta uudenlaista ajattelua. Laitteistokuvaskielen (VHDL/Verilog) valinta riippuu henkilökohtaisesta mieltymyksestä, koska molemmat kielet ovat teollisuusstandardoituja. Kellet (2012: 379) käyttää opetuksessa VHDL:ää, koska siinä on selvä ero esimerkiksi C-kieleen, joka helpottaa erottamaan ohjelmointiparadigmojen erot rinnakkaisessa ja peräkkäisessä ohjelmointikielessä. Työelämässä tarvitaan usein molempia kieliä, jonka vuoksi Kellet käyttää

kurssikirjana teosta jossa ne molemmat esitellään rinnakkain, joka opiskelijoiden mukaan on selkeä käsittelytapa. VHDL:n käyttö kurssilla on hyödyllistä, koska se on yleisin laitteistokuvauskieli Euroopassa ja ilmeisesti myös Suomessa.

Rakenteellinen VHDL-kuvaus ei ole yhtä suoraviivainen kuin esimerkiksi käyttäytymiskuvaus. Rakenteellisessa kuvaustavassa jokainen komponentti tulee määritellä erikseen kytkentäkaavion tapaan. Kurssin laskuharjoituksissa esimerkit näytettiin käyttäytymiskuvauksena. Harjoitustyön suorituksessa olisi ollut hyötyä, jos opiskelijat olisivat tunteneet rakennemallin (valmis projektipohja), sekä sen toiminnan. Rodriguez-Ponce ym. (2013: 172–177) suosittelivat TTL-piirien korvaamista FPGA-piireillä ja perusteellisemmalla VHDL:n käsittelyllä. Kurssin laskuharjoituksissa käytettiin TTL-piirejä FPGA-piirien kanssa. Opiskelijat saivat näissä harjoituksissa rakennella itse niistä kytkentöjä koekytkentälevyille, sekä joutuivat itse etsimään piirien tietoja niiden datalehdistä. TTL-piirien ja FPGA:n yhteiskäytöllä opiskelijat saivat konkreettisen esimerkin piirien ohjaamisesta FPGA:lla ja saivat itse rakennella koekytkentälevylle kytkennät. Tämän perusteella TTL-piirien käytöstä laskuharjoituksissa on hyötyä. Rodriguez-Poncen ym. (2013: 172–177) tekemän kyselyn tulokset ovat samankaltaisia tämän työn kyselytulosten kanssa, suurin osa heidän opiskelijoistaan oli sitä mieltä, että FPGA-VHDL-osaamisesta on hyötyä työelämässä.

Harjoitustyöhön kuului osia, joita ei ollut käyty lävitse luennoilla ja laskuharjoituksissa, kuten audiokoodekki väylineen. Valmis projektipohja sisälsi koodekin alustuksen käyttäen VHDL:n rakennemallia. Monilla oli kuitenkin vaikeuksia käyttää tätä pohjaa, eivätkä koodekin ominaisuudet olleet tuttuja, kuten kyselytuloksista on nähtävissä. Tulevaisuudessa harjoitustöissä/laskuharjoituksissa, tai myöhemmillä kursseilla voitaisiin käyttää laitteisto/ohjelmisto yhteistoteutusta käyttäen laitteistolle sulautettavia prosessoreja, jolloin opiskelijat saisivat kokemusta ohjelmisto/laitteisto yhteistoteutuksesta. Esimerkiksi Olivares, Gomez, Palomares ja Montijano (2008: 250–251) käyttivät opetuksessaan alustaa johon oli integroitu useita prosessoreja. He käyttivät graafista ohjelmistoa laitteiston ja ohjelmiston tekemiseen. Heidän mukaan se yksinkertaistaa käyttäjän suunnittelua antaen heille mahdollisuuden valita, yhdistää ja konfiguroida komponentit halutun tuloksen saavuttamiseksi.

7. YHTEENVETO

Tässä diplomityössä tutkittiin FPGA-tekniikan opetuksen kehittämistä Vaasan yliopiston automaatiotekniikan kursseissa. Teoriaosuudessa kuvattiin kaksi FPGA-liityntäarkkitehtuuria, hierarkkinen ja saareketyyppinen. Laitteistokuvauskielistä esiteltiin VHDL, Verilog, SystemVerilog sekä selvitettiin niiden opetusta muissa korkeakouluissa. Esiteltiin katsaus etälaboratorioista ja tavanomaisista laboratorioista. Etälaboratoriot toimivat internetin kautta ja niitä käytetään kirjallisuuden mukaan melko monessa ulkomaisessa yliopistossa. Tavanomaisissa laboratoriossa käytetään yleisesti eri valmistajien kehitysalustoja, lisäksi osa käyttää omia suunnitteluohjelmia laitevalmistajien ohjelmien lisäksi.

Valourut toteutettiin AUTO1010 digitaalitekniikan perusteet-kurssin harjoitustyönä keväällä 2014. Harjoitustyö toteutettiin kolmen hengen ryhmissä. Harjoitustyö oli laaja kokonaisuus, johon kuului kytkentöjä, ohjelmointia sekä FPGA-kehitysalustan eri toimintojen käyttöä, kuten analogia-digitaalimuunnosta (ADC) ja I^2C-väylää. Opiskelijoille valmiiksi annettu projektipohja käsitti tarvittavien toimintojen alustuksen, ja heille jäi toteutettavaksi pulssinleveysmoduuli (PWM) ja sen integrointi projektipohjaan, sekä tarvittavien kytkentöjen tekeminen koekytkentälevylle. Projektipohja sisälsi viisi VHDL-moduulia ja se oli toteutettu rakennemallina.

Laskuharjoitusten (VHDL) ohjelmointitehtävien tarkoituksena oli opettaa opiskelijoille laitteistokuvauskielen ja suunnitteluohjelman käyttöä, joita he tarvitsivat harjoitustyön suorituksessa. Laskuharjoituksissa opastettiin suunnitteluohjelman käytössä uuden projektin luomisessa, tallentamisessa, simuloinnissa ja FPGA-piirille lataamisessa. Laskuharjoituksissa käytettiin myös TTL-piirejä koekytkentälevyllä, joita ohjattiin FPGA:lla. Koekytkentälevyjä käytettiin useammassa laskuharjoituksessa, joissa niihin kytkettiin ledejä ja vastuksia, joita ohjattiin FPGA-piirillä. Nämä tehtävät olivat yksinkertaisia sovelluksia, kuten binäärilaskuri.

Opiskelijoilla ei ollut aikaisempaa käyttökokemusta kehitysalustalla olevien ja harjoitustyössä käytettyjen komponenttien ominaisuuksista ja toiminnoista, jolloin heillä oli vaikeuksia ymmärtää projektipohjan toiminta. Projektipohjassa oli alustettu WM8731

audio koodekin toiminta ja se sisälsi viisi VHDL-moduulia. Harjoitustyön ensimmäisellä ohjauskerralla esiteltiin audiokoodekin toiminta ja I^2C-väylä. Harjoitustyön suorituksesta suoritettiin palautekysely, johon vastaukset saatiin 15 opiskelijalta. Palautekyselyn toteutuksessa auttoi TkT Birgitta Martinkauppi, jonka vastuulla oli keväällä 2014 kurssin laskuharjoitusten pitäminen. Suurin osa kyselyyn vastanneista sanoi oppineensa laskuharjoituksissa VHDL-ohjelmointia. Kyselyn tulokset julkaistiin myös konferenssiartikkelissa (Karhu ym. 2015). Osalla oli ohjelmiston käytössä teknisiä ongelmia, joten aina opiskelijat eivät saaneet suoritettua kaikkia tehtäviä, vaikka he olisivat osanneet ohjelmoida tehtävän. Laitevalmistajien ohjelmistoja kannattaa kuitenkin käyttää laskuharjoitusten suorituksessa kolmansien osapuolien suunnitteluohjelmien sijasta, koska niitä käytetään myös teollisuudessa. Laitteistolle sulautettuja prosessoreita (SoC) voitaisiin käsitellä laskuharjoituksissa/harjoitustyössä tai muilla automaatiotekniikan kursseilla, koska digitaalitekniikan ratkaisut kehittyvät kohti laitteisto/ohjelmisto yhteistoteutuksia. Altera on toteuttanut piirin joka sisältää ARM-prosessorin sekä FPGA-logiikan, lisäksi systeemin ohjaukseen on mahdollisuus käyttää Linuxia. Laitteisto/ohjelmisto yhteistoteutus on kuitenkin teknisesti monimutkaisempi kuin pelkkä laitteiston toteuttaminen.

Kurssin (AUTO1010) toteutukseen ei tarvita kirjallisuudessa esiteltyjä etälaboratorioita tai erillisiä laboratoriotiloja. Kurssin laskuharjoitukset/harjoitustyö eivät vaadi erityisjärjestelyitä esimerkiksi turvallisuuteen liittyen. Tarvittavat tehtävät voidaan toteuttaa käyttäen kehitysalustaa ja ohjelmistoa (simulointi ym), myöhemmillä kursseilla tiettyyn sovellukseen tarkoitettu erillinen laboratoriotila voisi tulla kysymykseen. Luentojen toteutuksissa tuskin on mitään suurempaa muutettavaa, koska suurin osa kyselyyn vastanneista antoi niille neutraalin vastauksen. Vastauksien mukaan FPGA-tekniikan osaaminen on enemmistön mielestä hyödyllistä, johon on varmasti vaikuttanut luennoilla ja laskuharjoituksissa käsitellyt asiat.

LÄHTEET

Alander, Jarmo T (2015). AUTO1010 Digitaalitekniikan perusteet/Introduction to digital electronics [online]. [Viitattu 12.1.2015]. Saatavana World Wide Webistä: <URL: http://lipas.uwasa.fi/~TAU/AUTO1010/>

Altera (2014). DE2 Development and Education Board User Manual [online]. [Viitattu 15.10.2014]. Saatavana World Wide Webistä: <URL: ftp://ftp.altera.com/up/pub/Webdocs/DE2_UserManual.pdf>

Altera (2014). Processors from Altera and embedded alliance partners [online]. [Viitattu 4.12.2014]. Saatavana World Wide Webistä: <URL: http://www.altera.com/devices/processor/emb-index.html >

Araujo, Antonio J & Alves Jose C (2008). A project driven digital design course using FPGAs. *EAEEIE Annual Conference, 2008 19th*. 42–47.

Artigas, Jose I, Barragan, Luis A, Isidro, Urriza, Navarro, Dennis & Lucia Oscar (2011). FPGA-based digital control implementation of a power converter for teaching purposes. *e-Learning in Industrial Electronics (ICELIE), 2011 5th IEEE International Conference on.* 55–60.

Ashenden, Peter J (2008). Digital Design An Embedded Systems Approach Using Verilog. Elsevier Ltd. ISBN 978-0-12-369527-7.

Athar, Shahrukh, Siddiqi, Muhammad Ali & Masud Shahid (2012). Teaching and research in FPGA based digital signal processing using Xilinx system generator. *Acoustics, Speech and Signal Processing (ICASSAP), 2012 IEEE International Conference on.* 2765–2768. ISSN 1520-6149.

Balid, Walid & Abdulwahed Mahmoud (2013). A novel FPGA educational paradigm using the next generation programming languages case of an embedded FGPA sys-

tem course. *Global Engineering Education Conference (EDUCON), 2013 IEEE*. 23–31.

Brekken, Ted K.A & Mohan Ned (2006). A flexible and inexpensive FPGA-based power electronics and drives laboratory. *Power Electronics Specialists Conference, 2006. PESC '06. 37th IEEE*. 1–4. ISSN 0275-9306.

Datta, Kushal & Sass Ron (2007). RBoot: Software infrastructure for a remote FPGA laboratory. *Field-Programmable Custom Computing Machines FCCM 2007*. 343–344.

Donzellini, G & Ponta D (2012). A novel tool to introduce FPGA in digital design laboratory. *Remote Engineering and Virtual Instrumentation (REV), 2012 9th International Conference on*. 1–8.

Donzellini, Giuliano & Ponta Domenico (2013). From gates to FPGA: Learning digital design with Deeds. *Interdisciplinary Engineering Design Education Conference (IEDEC), 2013 3rd*. 41–48.

Drutarovsky, Milos, Saliga, Jan, Michaeli, Linus, Hroncova, Ingrid (2009). Remote laboratory for FPGA based reconfigurable systems testing [online]. Kosice: Technical University of Kosice, 2009 [Viitattu 4.11.2014]. Saatavana World Wide Webistä: <URL:http://pdf.aminer.org/000/270/369/a_remote_laboratory_for_debugging_fpga_based_microprocessor_prototypes.pdf>

Duckworth, James R (2005). Embedded system design with FPGAs using HDLs (Lessons learned and pitfalls to be avoided). *Microelectronic Systems Education, 2005. (MSE '05). Proceedings. 2005 IEEE International Conference on*. 35–36.

Ellervee, Peeter, Reinsalu, Uljana, Arhipov, Anton, Ivask, Eero, Tammemäe, Kalle, Evartson, Teet & Sudnitson Aleksander (2008). HDL-s and FPGA-s in digital design education. *EAEEIE Annual Conference, 2008 19th*. 37–41.

El-Medany, Wael M (2008). FPGA remote laboratory for hardware e-learning courses. *Computational Technologies in Electrical and Electronics Engineering SIBIRCON 2008 IEEE Region 8 International Conference on*. 106–109.

Grout, Ian (2008). Digital Systems Design with FPGAs and CPLDs. Elsevier Ltd. ISBN-13: 978-0-7506-8397-5.

Haba, Christian-Gyözö (2014). Using FPGA development boards for multi-course laboratory support. *Global Engineering Education Conference (EDUCON), 2014 IEEE*. 794–797.

Hocenski, Zeljko, Aleksi, Ivan & Sruk Vlado (2013). Adaptive Virtual Devices Platform for verification of FPGA modules in student courses on Digital Design. *e-Learning in Industrial Electronics (ICELIE), 2013 7th International Conference on*. 22–27.

IEEE Std 1364-2001 (2001). IEEE Standard Verilog® Hardware Description Language [online]. New York, NY, USA: The Institute of Electrical and Electronics Engineers Inc [siteerattu 3.2.2014]. Saatavana World Wide Webistä: <URL: http:// http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=954909>

IEEE Std 1076-2008 (2009). IEEE Standard VHDL Language Reference Manual [online]. New York, NY, USA: The Institute of Electrical and Electronics Engineers Inc [siteerattu 28.11.2014]. Saatavana World Wide Webistä: <URL: http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=4772740>

IEEE Std 1800-2005 (2005). IEEE Standard for SystemVerilog–Unified Hardware Design, Specification, and Verification Language [online]. New York, NY, USA: The Institute of Electrical and Electronics Engineers Inc [siteerattu 1.12.2014]. Saatavana World Wide Webistä: <URL: http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=1560791>

Karhu, Suvi, Alander, Jarmo & Nurmi Otto (2015). Some tools for aiding teaching the basics of digital electronics and signal processing. *To appear: CSEDU2015 7th International Conference on Computer Supported Education.* Lisbon, Portugal, May 2015.

Karyano, Wicaksana, Arya (2013). Teaching microprocessors and microcontroller fundamental using FPGA. *New Media Studies (CoNMedia), 2013 Conference on*. 1–5.

Kastelan, Ivan, Majstorovic, Dusan, Nikolic, Milos, Eremic, Jelena & Katona Mihajlo (2012). Laboratory exercises for embedded engineering learning platform. *MIPRO, 2012 Proceedings of the 35th International Convention*. 1113–1117.

Kellet, Christopher M (2012). A project-based learning approach to programmable logic design and computer architecture. *Education, IEEE Transactions on (Volume 55, Issue: 3)*. 378–383. ISSN 0018-9359.

Kiray, Vedat, Demir, Salih & Zhaparov Meirambek (2013). Improving digital electronics education with FPGA technology, PBL and Micro Learning Methods: A case study from FPGA based digital electronics education project (FPGA_b_DEEP). *Teaching, Assessment and Learning for Engineering (TALE), 2013 IEEE International Conference on*. 445–448.

Knodel, Oliver, Zabel, Martin, Lehmann, Patrick & Spallek Rainer G (2014). Educating hardware design – from Boolean equations to massively parallel computing systems. *Programmable Logic (SPL), 2014 IX Southern Conference on.* 1–6.

Kumar, Akash, Fernando, Shakith & Panicker Rajesh C (2013). Project-based learning in embedded systems education using an FPGA platform. *Education, IEEE Transactions on (Volume:56, Issue: 4).* 407–415. ISSN 0018-9359.

Kuon, Ian, Tessier, Russel & Rose Jonathan (2007). FPGA architecture: survey and challenges. *Foundations and Trends in Electronic Design Automation* 2, [Viitattu 14.12014], 135–253.

Laakkonen, Olli, Rauma, Kimmo, Ikonen, Mika & Pyrhönen Olli (2006). Reconfigurable platform for teaching motor control algorithms. *Power Electronics Specialist Conference, 2006. PESC '06. 37th IEEE.* 1–4.

Lötjönen, Lauri (2012). FPGA Implementation of the Stepwise Shutdown System [online]. Espoo: Valtion Teknillinen Tutkimuskeskus, 2014 [siteerattu 14.1.2014]. Saatavana World Wide Webistä: <URL: http://www.vtt.fi/inf/julkaisut/muut/2012/VTT-R-06053-12.pdf >

Lötjönen, Lauri (2013). Field-Programmable Gate Arrays in Nuclear power Plant Safety Automation [online]. Espoo: Diplomityö, Aalto-yliopisto: Sähkötekniikan korkeakoulu, 2013 [siteerattu 26.11.2014]. Saatavana World Wide Webistä: <URL: https://aaltodoc.aalto.fi/bitstream/handle/123456789/10921/master_L%C3%B6tj%C3%B6nen_Lauri_2013.pdf?sequence=1>

Mahmoodi, Hamid, Montoya, Arturo, Franco, Joie du, Rodriguez, Chris, Carrillo, Jose, Goel, Ankita, Chen, Cheng, Enriquez, Amelito G, Jiang, Hao, Pong, Wenshen & Shanasser Hamid (2012). Hands-on teaching of embedded systems design using FPGA-Based tPad development kit. *Interdisciplinary Engineering Design Education Conference (IEDEC), 2012 2nd*. 1–6.

Matilainen, Lauri, Salminen, Erno & Hämäläinen Timo D (2014). Experiences from System-on-Chip design courses. *Microelectronics Education (EWME), 10th European Workshop on*. 37–42.

Meyer-Baese, Uwe (2007). Digital Signal Processing with Field Programmable Gate Arrays. 3. painos. Springer. ISBN 978-3-540-72612-8.

Morgan, Fearghal & Cawley Seamus (2011). Enhancing learning of digital systems using a remote FPGA lab. *Reconfigurable Communication-centric Systems-on-Chip (ReCoSoC)* 6. 1–8.

Morgan, Fearghal, Cawley, Seamus, Callaly, Frank, Agnew, Shane, Rocke, Patrick, O'Halloran, Martin, Drozd, Nina, Krzysztof, Kepa & McGinley Brian (2011). Remote FPGA with interactive control and visualisation interface. *Field Programmable Logic and Applications (FPL)*. 496–499.

Naser, J (2009). Guidelines on the use of Field Programmable Gate Arrays (FPGAs) nuclear power plant I&C systems [online]. Palo Alto, California, USA: Electric Power Research Institute (EPRI), 2009 [siteerattu 11.3.2015]. Saatavana World Wide Webistä: <URL: http://www.epri.com/abstracts/Pages/ProductAbstract.aspx?ProductId=000000000001019181&Mode=download>

National Instruments (2012). Introduction to FPGA technology: top 5 benefits [online]. [Viitattu 28.11.2014]. Saatavana World Wide Webistä: <URL: http://www.ni.com/white-paper/6984/en/#top>

National Instruments (2010). Pulse Width Modulation (PWM) Using NI-DAQmx and LabVIEW [online]. [Viitattu 7.1.2015]. Saatavana World Wide Webistä: <URL: http://www.ni.com/white-paper/2991/en/>

Navabi, Zainalabedin (2006). Verilog Digital System Design. 2. painos. McGraw-Hill Publishing Companies.

O'Connor, Joseph E (2008). Field programmable gate array control of power systems in graduate student laboratories [online]. Monterey, California: Naval postgraduate school, 2008 [siteerattu 9.2.2015]. Saatavana World Wide Webistä: <URL: http://edocs.nps.edu/npspubs/scholarly/theses/2008/mar/08mar_oconnor.pdf>

Olivares, Joaquin, Gomez, Juan, Palomares, Jose M & Montijano Miguel A (2008). Biprocessor SoC in an FPGA for teaching purposes. *Advanced Learning Technologies, 2008. ICALT '08. Eighth IEEE International Conference on.* 250–251.

Oriti, Giovanna, Julian, Alexander L, Zulaica, Dan & Cristi Roberto (2010). Hardware laboratories for power electronics and motor drives distance learning courses. *Energy Conversion Congress and Exposition (ECCE), 2010 IEEE.* 2229–2236.

P.Chu, Pong (2008). FPGA prototyping by Verilog examples Xilinx Spartan™ -3 Version. John Wiley & Sons, Inc. ISBN 978-0-470-18532-2.

Pedroni, Volnei (2008). Digital Electronics and Design with VHDL. Elsevier Inc. ISBN: 978-0-12–374270-4.

Penttinen, Aki (2005). FPGA:lle sulautetulla mikroprosessorilla toteutettu sähkönkäytön säätöjärjestelmä [online]. Lappeenranta: Diplomityö, Lappeenrannan teknillinen yliopisto, 2005 [siteerattu 26.11.2014]. Saatavana World Wide Webistä: <URL: https://www.doria.fi/bitstream/handle/10024/34566/nbnfi-fe20051588.pdf?sequence=1>

Persiano, G.V, Rapuano, S, Zoino, F, Morganella, A & Chiusolo G (2007). Distance learning in digital electronics: laboratory practice on FPGA. *Instrumentation and Measurement Technology Conference Proceedings IMTC 2007*. 1–6. ISSN 1091-5281.

Qasim, Syed Manzoor, Abbasi, Shuja Ahmad & Almashary Bandar (2009). A review of FPGA-based design methodology and optimization techniques for efficient hardware realization of computation intensive algorithms. *Multimedia, Signal Processing and Communication Technologies, 2009. IMPACT '09. International*. 313–316.

Quintas, Camilo, Valdes, M.Dolores, Moure, M Jose, Fernandez-Ferreira, Luis & Mandado Enrique (2005). Digital electronics learning system based on FPGA applications. *Frontiers in Education, 2005. FIFE '05. Proceedings 35th Annual Conference*. ISSN 0190-5848.

Ranta, Jukka (2012). The current state of FPGA technology in the nuclear domain [online]. [Viitattu 28.11.2014]. Saatavana World Wide Webistä: <URL: http://www.vtt.fi/inf/pdf/technology/2012/T10.pdf>

Rajasekhar, Yamuna, Kritikos, William V, Schmidt, Andrew G & Sass Ron (2008). Teaching FPGA system design via a remote laboratory facility. *Field Programmable Logic and Applications FPL2008*. 687–690.

Reinsalu, Uljana, Arhipov, Anton, Evartson, Teet & Ellervee Peeter (2007). HDL-s for students with different background. *Microelectronic Systems Education, 2007. MSE '07. IEEE International Conference on*. 69–70.

Rodriguez-Ponce, Rafael & Rodriguez-Resendiz Juvenal (2013). Integrating VHDL into an undergraduate digital design course. *Teaching, Assessment and Learning for Engineering (TALE), 2013 IEEE International Conference on*. 172–177.

Rose, Jonathan (1991). Flexibility of interconnection structures for field-programmable gate arrays. *Solid-State Circuits, IEEE Journal (Volume: 26, Issue: 3).* 277–282.

Schneider, Joseph, Bezdek, Mikel, Zhang, Ziyu, Zhang, Zhao & Rover Diane T (2005). A platform FPGA-based hardware-software undergraduate laboratory. *Microelectronic Systems Education, 2005. (MSE '05). Proceedings. 2005 IEEE International Conference on*. 53–54.

Skliarova, Iouliia, Sklyarov, Valery, Sudnitson, Alexander & Kruus Margus (2014). Teaching FPGA-based systems. *Global Engineering Education Conference (EDUCON), 2014 IEEE*. 460–469.

Skliarova, Iouliia, Sklyarov, Valery, Sudnitson, Alexander & Kruus Margus (2013). Using mobile technology to enhance teaching reconfigurable systems. *Teaching, Assessment and Learning for Engineering (TALE), 2013 IEEE International Conference on*. 478–483.

Smith, Gina R (2010). FPGAs 101. 1 .painos. Elsevier Inc. ISBN 978-1-85617-706-1.

Soares, João & Lobo Jorge (2011). A remote FPGA laboratory for digital design students [online]. Coimbra: University of Coimbra, 2011 [siteerattu 4.11.2014]. Saatavana World Wide Webistä: <URL:http://ap.isr.uc.pt/archive/jlobo-REC2011.PDF>. ISBN 978-989-97151-0-3.

Stastny, Jakub & Ruckay Lukas (2006). Teaching field programmable gate array design of digital signal processing systems. *Applied Electronics, 2006. AE 2006. International Conference on.* 201–204.

Sutherland, Stuart, Davidmann, Simon & Flake Peter (2006). SystemVerilog for design. 2 painos. Springer Science+Business Media, LLC. ISBN-10 0-387-33399-1.

UM10204 (2014). I2C-bus specification and user manual [online]. [Viitattu 8.1.2015]. Saatavana World Wide Webistä: <URL: http://www.nxp.com/documents/user_manual/UM10204.pdf>

U.S.NRC (2009). Review guidelines for field-programmable gate arrays in nuclear power plant safety systems [online]. [Viitattu 28.11.2014]. Saatavana World Wide Webistä: <URL: http://pbadupws.nrc.gov/docs/ML1008/ML100880142.pdf>

Vainio, Olli, Salminen, Erno & Takala Jarmo (2010). Teaching digital systems using a unified FPGA platform. *Electronics Conference (BEC), 2010 12th Biennial Baltic*. 137-140. ISSN 1736-3705.

Vinay, Krishna Chandran, Shyam, H N, Rishi, S & Moorthi S (2011). FPGA based implementation of variable-voltage variable-frequency controller for a three base induction motor. *Process automation, Control and Computing (PACC), 2011 International Conference on.* 1–6.

Wolfson microelectronics (2012). Portable internet Audio CODEC with headphone driver and programmable sample rates [online]. [Viitattu 8.1.2015]. Saatavana

World Wide Webistä: <URL: http://www.wolfsonmicro.com/documents/uploads/data_sheets/en/WM8731.pdf>

Xilinx (2013). What is an FPGA?[online]. [Viitattu 14.1.2014]. Saatavana World Wide Webistä:<URL:http://www.xilinx.com/fpga/>

Xilinx (2014). Embedded processing peripheral IP cores [online]. [Viitattu 4.12.2014]. Saatavana World Wide Webistä: <URL: http://www.xilinx.com/ise/embedded/edk_ip.htm >

Yuxi, Zhang, Li, Kang, Jun, Wang, Jinping, Sun & Zulin Wang (2010). Methods and experience using Matlab and FPGA for teaching practice in digital signal processing. *Education and Management Technology (ICEMT), 2010 International Conference on*. 414–417.

Zwolinski, Mark (2004). *Digital System Design with VHDL*. 2. painos. Pearson Education. ISBN 0 130 39985 X.

LIITTEET

LIITE 1. Kyselylomake

AUTO1010 – harjoitustyö

Harjoitustyötä on tarkoitus tukea lisätunteina. Tämän vuoksi opiskelijoilta halutaan palautetta siitä, missä he haluavat lisäopetusta. Lisäopetuksen tarkoituksena on saattaa harjoitustyöt valmiiksi touko-kesäkuun aikana. Palautteen saa antaa myös nimellisenä, jos haluaa.

1. Mitä osasit, kun tulit harjoituksiin? / What did you already know before coming to exercise?

2. Mitä opit harjoituksissa? / What did you learn in exercise?

3. Mitä haluaisit vielä oppia? / What do you want to learn?

	Täysin eri mieltä /disagree strongly	Eri mieltä /disagree	Neutraali Neutral	Samaa mieltä / agree	Täysin samaa mieltä / agree strongly
Ymmärrän PWM:n teorian					
Osaan ohjelmoida PWM:n VHDL:llä					
Ymmärrän I2C:n teorian					
Osaan käyttää valmiiksi ohjelmoitua I2C:tä					
Luennot auttoivat oppimisessa					
Harjoitukset auttoivat oppimaan VHDL-ohjelmointia					
FPGA-tekniikan osaaminen on hyödyllistä					

Muuta?

LIITE 2. Konferenssiartikkeli

SOME TOOLS FOR AIDING TEACHING THE BASICS OF DIGITAL ELECTRONICS AND SIGNAL PROCESSING

Suvi Karhu[1], Jarmo T. Alander[1] and Otto Nurmi[1]
Department of Electrical Engineering and Energy Technology, University of Vaasa, Yliopistonranta 10, Vaasa, Finland
jarmo.alander@uva.fi, {suvi.karhu, otto.nurmi}@student.uva.fi

Keywords: Animation, Digital Electronics, Digital Signal Processing (DSP), Finite Impulse Response Filter (FIR), Engineering Studies, Field Programmable Gate Array (FPGA), Hardware Design Languages (HDL), VHDL

Abstract: In this paper we describe some computing tools designed for aiding teaching of the basics of digital electronics and its applications mainly in signal processing for university studies of engineering. In this study we have developed two types of teaching tools: firstly, several small JavaScript-based simulation tools for visualizing the basic functions of digital circuits and their hardware design language models, and secondly, an FPGA-based FIR filter system for showing how to perform simple digital signal processing tasks with FPGAs.

1 INTRODUCTION

One of the authors (J.A.) has been teaching digital electronics in several courses in Finnish universities for over thirty years. For these courses the author has implemented some simulators using such programming languages as Extended Algol, Simula, VisualBasic, and C++. The idea has been to teach the basics of functioning of digital circuits with computers which allow detailed monitoring of the events in digital circuits while eliminating totally the many practical problems of physical circuits and their monitoring. For most engineers and scientists it is important to know the principles of digital electronics, how typical circuits function and how they can be combined together to create more complex devices. If real circuit implementations are needed, they are typically designed by experts of digital electronics to whom the physical aspects of circuits are naturally very well known. They also use highly complex software not suitable for learning the very basics of digital electronics.

Today the computer aided programming approach itself is more or less enough when the system or circuit is usually implemented on a microcontroller, Programmable Digital Signal Processor (PDSP), or Field Programmable Gate Array (FPGA) circuits. Only a minimum amount of knowledge of the physical aspects is needed, since most of the design work is done using software. Usually the hardware is ready to run without any soldering or similar physical steps. FPGAs are not a curiosity of digital electronics, but a class of devices for the most challenging computational problems encountered by as well engineers in industry as scientists in university and research laboratories. Therefore it is important that the future experts who might need FPGAs in their projects get some basic knowledge of this technology.

In our university we have concentrated on FPGAs as the hardware realization of digital devices. All the graduating students of engineering get at least basic knowledge of FPGAs, while there is also the possibility to concentrate on FPGAs especially in signal processing applications at Master's level. Therefore it is important to give students some very easy hands-on experimentations during lectures and laboratory works of several courses related to digital electronics and digital signal processing (DSP).

In the field of signal processing, PDSPs dominate due to their low price and suitability for performing complicated algorithms (Meyer-Baese 2007: 12-13). However, nowadays some signal processing algorithms are increasingly

often run on FPGAs. FPGAs are particularly good in front-end applications, like Finite Impulse Response (FIR) filters and Fast Fourier Transforms (FFTs) (Donovan, 2002). Due to this trend, it is useful to teach FPGA-based signal processing to the students of basic DSP course.

Programming of FPGAs is considered challenging. This is because of several reasons:

- FPGAs are digital hardware; therefore basic knowledge of digital circuits is needed to understand and use them.
- FPGAs are used by programming; basic knowledge of programming is needed but is not enough simply because of the parallel nature of the circuits and their modeling languages. This parallelism is usually totally absent from the general-purpose programming languages like Java.
- The efficient use of parallel processing by FPGAs presupposes understanding of pipelining, which is usually not known by ordinary programmers.

In this study we have developed two types of teaching tools. Firstly, several small web-based JavaScript simulation tools were created for visualizing the function of digital circuits directly on lecture slides. The idea has been to aid learning of basics by providing very simple animations without the need of more sophisticated simulation software. Secondly, an FPGA-based FIR filter system was developed for showing how to perform a simple digital signal processing task with an FPGA device and with the help of Matlab HDL Coder. Finally, as an application of the audio system, a light organ was designed as a student project using FPGA and a Light Emitting Diode (LED) display made of discrete components.

1.1 Related work

Several software for simulating electronic circuits exist. Those include i.a. SPICE for analog circuits, Logisim for digital circuits and GNU Circuit Analysis Package for mixed-signal (analog and digital) circuits. These software are suitable when doing homework and project work, but they are usually not used during the lectures. Our idea is to bring some visual aspect and interactivity to the lectures by integrating some simulations to the lecture slides. The hypothesis is that slides with animated graphics improve learning when compared to conventional, static lecture slides.

Since the learning of FPGA is time-consuming, it is good if the student can study the subject also by distance-learning. The idea of learning FPGA programming in web environment has been utilized in virtual and remote laboratories. A remote laboratory is based on real hardware, while virtual laboratories are based on software models. Our simulation tools presented in this paper are neither remote nor virtual laboratories, but suitable for distance-learning because they are available via Internet.

A proposal for remote laboratory is presented by Drutarovský et al. (2009). Their system includes a LabVIEW based measurement server, FPGA development boards, logic analyzer, digital storage oscilloscope and signal generator. Persiano et al. (2007) propose a system through which users can remotely control FPGA applications without installing any development software to the computer. The user can follow teaching and control several measuring devices through a LabVIEW based user interface. According to Soares et al. (2011) in remote laboratories the challenge is the interaction between the user and the hardware as a realistic Internet-based interface. They implemented an application using Hypertext Preprocessor (PHP) which makes it possible to use Altera® DE2 board and camera remotely. Multimedia and interactivity have also been utilized by Quintas et al. (2005), who have implemented a hypermedia application for self-learning.

In the field of DSP, Hall et al. (2002) have developed a framework which allows students to combine their knowledge about DSP theory and digital hardware design. They developed a Matlab toolbox for sending and receiving data to and from an FPGA device. The functions are useful in testing, debugging and verifying the design. In addition, they introduced a VHDL template, which contains all needed testing infrastructure, and the students only need to develop their own DSP module. Their pedagogical principle was to start from small filters and proceed to more complex DSP systems after the students have understood the flow of the design process. Later Hall et al. (2007) have also emphasized the need of teaching fixed-point signal processing, as the fixed-point solutions are often faster, smaller, cheaper, and consume less power than floating-point solutions. Haba (2014) suggests that

FPGA circuits can replace microcontrollers, processors and other similar devices in the field of digital signal processing. Reyes et al. (2009) developed a FPGA-based DSP trainer for learning Fourier transform, FIR filters, correlation and linear convolution. The trainer was also able to generate and acquire input signals and display the results on a monitor.

2 WEB-BASED SIMULATIONS

The idea of our approach is to provide students with study material that contains simple visual animations on the basic components of digital electronics. The hypothesis is that simple animations aid learning by giving clear visual representation of the functions. There are known problems with complex animations that may even disturb learning by overloading perception (Ainsworth, 2008). Here we assume that simple animations have more positive than negative effects on learning.

In order to keep things as simple as possible our tools are separate and available via Internet not only for the students of our university, but also for anyone interested in digital electronics simulations. Our simulations only demonstrate certain aspects of digital electronics and implementation of hardware using HDLs. They are not meant to replace any more complex simulation software but to provide a replacement for illustrations and show how the function can be realized in various Hardware Description Languages (HDLs) and styles (VHDL architectures).

All lecture pages can contain emphasized words, keywords, that are not only highlighted using html <b>-tags but also used in a simple game, in which the words are guessed like in the hangman game. The material used in this game is also used in weekly brief exams we call microexams so that the student can use this feature for preparing to the exams by drill learning key concepts of the lecture material (Delazer et al., 2005).

Our simulations are implemented using JavaScript, and they have been designed and tested only with Mozilla Firefox. However, most of the demos seem to work quite nicely at least also with Google Chrome, perhaps because parts of them are implemented with jQuery, a JavaScript library which helps to avoid most browser incompatibilities.

Because the simulations are implemented in JavaScript which is run on the client (student PC) side and not on the server (university) side, there can be in principle a nearly unlimited number of concurrent users (students) without much computing resource problems. This means that in future we can easily increase the amount of similarly implemented animations and other interactive material to our lecture slides.

2.1 DigiAnilator

Our first tool presented here is called DigiAnilator. To make the tool to be easily found on the Internet, it was given a googleunique name, which means that according to Google, it is not used on any other webpage.

DigiAnilator[1] is able to simulate simple digital circuits having up to 5 inputs and outputs. Thus it is able to simulate gates or other simple circuits used in digital circuits such as multiplexers and decoders. The simulation is based on the truth table of the circuit. The user can freely edit the truth table with "*Invert State*" button to model any function of up to 5 variables he or she wants to simulate. The other output of DigiAnilator gives the textual description (program code) of the function in several HDLs and variants of implementations, which in VHDL are called architectures. When the user changes the function to be simulated, the descriptions are immediately updated. DigiAnilator uses a function colour coding (the truth table) which was originally shown in our earlier Java- and Visual Basic- based digital circuit simulators (Alander et al., 1998).

2.2 AutoMagic

A human being is good at understanding and remembering (learning) visual information. The obvious problem in this respect when using any modeling language, not only HDL, is that they almost totally lack any graphical illustrations. This is a big problem especially in teaching of digital electronics basics, while the circuits are most naturally represented by well established circuit diagrams.

---

1 http://lipas.uwasa.fi/~TAU/AUTO1060/slides.php?File=3000Computer.txt&Page=27

In order to show how the VHDL model is running in parallel, another JavaScript-based web application called AutoMagic[2] was implemented to illustrate both the running of the HDL program and the graphical presentation of the circuit and other possible illustrations such as the truth tables. AutoMagic provides a step-by-step simulation of the model according to Gray-coded input sequence. In addition the speed of the simulation can be interactively controlled with arrow keys.

2.3 DSPAutoMagic

For demonstrations of DSP basics within web-based lecture slides a simple DSP system called DSPAutoMagic[3] was implemented. The idea is to combine all multimedia modes (graphics, image both 2D and 3D, and audio) to interactively illustrate typical signals, filters and their applications. The system can be used both as a standalone webpage, and as a part of another webpage. A simple event-recording and playback system was implemented to make it easy to automatically start a given demonstration on a lecture slide without the need for manual interactive setting of parameters.

3 FIR FILTER DEMONSTRATIONS

Yuxi et al. (2010) suggest that with FPGA circuits students can implement practical applications from theoretical subjects along with design software like Matlab. In addition to developing the web-based simulation tools, we wanted to demonstrate FPGA-based digital signal processing for students of a bachelor level DSP course, and to introduce the use of Matlab HDL Coder to facilitate the programming process.

For this purpose we created an FPGA-based FIR filter system, which was designed for filtering audio signals. The students of the bachelor level DSP course have no previous experience on FIR filters, but they have studied one course of digital electronics, which includes basic FPGA programming with VHDL. The aim of this demonstration is to present the students how to perform a basic signal processing task with an FPGA device.

The system included two filters, which we call LargeFilter and SmallFilter. The first filter, which we call LargeFilter, is designed to filter integer signals with integer coefficients. The input is convolved with 31 coefficients, and the result is divided by 256. This filter was designed by modifying the code from Meyer-Baese (2007: 179-180). The input and output signals are sent to the filter from Nios II processor, but the coefficients are not programmable. The input and output are 16-bit and 32-bit, respectively. Depending on the choice of coefficients, the filter can be either symmetric or asymmetric. In this study we used a symmetric filter.

The second filter, which we call SmallFilter is designed to filter fixed-point signals with fixed-point coefficients. The input and output are both 14 bits wide. The filter is symmetric and contains 8 programmable coefficients. The filter was generated with Matlab HDL Coder.

3.1 Tools

The system was designed with the Development and Education board DE2 of Altera®. The system consisted of Nios II processor, audio codec, SDRAM memory, switches and buttons. Nios II is a soft processor which can be instantiated on an Altera® FPGA board. Nios processor was programmed using SOPC Builder of the Quartus II 10.1sp1 software and Nios Software Build Tools (SBT) 10.1sp1 which is an Eclipse-based integrated development environment (IDE) for Nios-specific C/C++ code. The latter is an Eclipse-based integrated development environment (IDE) for Nios-specific C/C++ code. The former is the built-in SOPC tool of Quartus, but is nowadays considered a legacy tool because in newer Quartus versions it is replaced with a more high-performance tool, Qsys.

---

[2] *AND gate example. AutoMagic*. Jarmo Alander. *AUTO1060 course.* http://lipas.uwasa.fi/~TAU/AUTO1060/slides.php?File=3000Computer.txt&Page=27

[3] *DSP / Measurements. DSPAutoMagic*. Jarmo Alander. *AUTO1060 course.* http://lipas.uwasa.fi/~TAU/AUTO1060/slides.php?File=0000Intro.txt&Page=35

Matlab® HDL Coder is a tool which is able to create VHDL and Verilog code from Matlab functions, Simulink models and Stateflow charts. HDL Coder generates both synthesizable code for programming FPGAs and testbenches for simulation. Matlab code generation process is executed with the help of Workflow advisor, which offers the following actions (MathWorks®, 2014):

- Verify floating point design. Floating-point design is the original Matlab code written by the user. In this step the software simulates that code and collects the minimum and maximum values for all the variables. These results can later be compared to the results of the simulation of the fixed-point design.
- Run floating-to-fixed conversion. In this step the original Matlab code is transformed into different kind of Matlab code, which uses only fixed-point numbers.
- Verify the fixed-point design. In this step we can compare the outputs of floating- and fixed-point codes.
- Generate HDL code.
- Simulate the generated HDL code. Here the code can be simulated with ModelSim or ISIM simulator, and we see if the results match with the results of the fixed-point design.
- Synthesis and project creation. HDL Coder also helps create an Altera® Quartus™ or Xilinx® ISE™ project with the selected options.

3.2. The system

The FPGA system was built around Altera®'s Nios II processor. The Nios II system was connected to one user peripheral - the audio codec Wolfson WM8731 controller - and to some pre-designed peripherals: parallel I/O (PIO) modules for the buttons, switches and light-emitting diodes (LEDs), Liquid Crystal Display (LCD) controller, on-chip memory controller and an SDRAM controller. All these modules used the Avalon Memory-Mapped (MM) interface to communicate with the processor. PIO modules had also the Interrupt Sender interface and signals from the audio codec were exported to the top level using Avalon conduit interface.

3.2.1 Audio codec

WM8731 is a stereo codec device that has two pairs of ADA (analog-digital-analog) converters: one for both channels, left and right. The audio codec controller VHDL files and the controller driver C file were obtained from the textbook of Chu (2011). WM8731 was first configured through I2C bus so that the left and right headphone volumes were set to 0 dB and analog path was set to "microphone in" with no boost. Default 48K sampling rate with 16-bit samples and 12.288 MHz master clock was used.

3.2.2 Memories

The application used two memories: the on-chip memory of the FGPA device and an external SDRAM memory module. The on-chip memory of the EP2C35 FPGA is organized in 105 M4K blocks. However, in this work we needed only a small on-chip memory, and thus its size was set to 16 Kbytes. Communication was performed using 32-bit words, which is the recommended wordlength when connecting the on-chip memory to the data master port of the processor (Altera®, 2010).

The SDRAM memory module located on the DE2 board can store 8 MiB of data. It is meant for storing large blocks of data, for example audio data (Altera®, 2010). The communication between the system and memory module was performed using the pre-designed SDRAM controller provided by the SOPC Builder. In addition to this, a phase-locked loop (PLL) was needed to clock the SDRAM module. The purpose of the PLL circuit is to generate a clock signal which is suitable for the SDRAM. That clock must be phase-shifted by -3ns from the original clock source, as described in Altera®'s (2008) tutorial. The ALTPLL module of the SOPC Builder software did not work, but the problem was solved by using an ALTPLL megafunction instead of ALTPLL Nios module. The megafunction was inserted to the top-level schematic diagram and the inputs and outputs were connected to the appropriate pins.

3.2.3 Instantation of the Nios system

VHDL files of the SOPC system were generated and a block diagram file (BDF) was generated from the VHDL files. The BDF file was then used to create the top-level schematic diagram file (Fig. 1). A PLL megafunction block was inserted to the top-level diagram file, as well as the power source for the LCD. The pins were assigned by using the Comma-Separated Values (CSV) file from the DE2 system CD. The

maximum operating frequency of the system was set to 50 MHz, which was the frequency of the system clock signal.

3.2.4 Filters

The filters could have been integrated inside the Nios system in the same way as the audio codec controller: by writing an Avalon-compatible wrapping circuit around the filter component. However, we used a less time-consuming approach and placed the FIR filters outside the Nios system and connected them to the Nios system with PIO modules. The input data could then be sent to the filter by writing it to the appropriate PIO register, and similarly the filtered data was acquired by reading the PIO register.

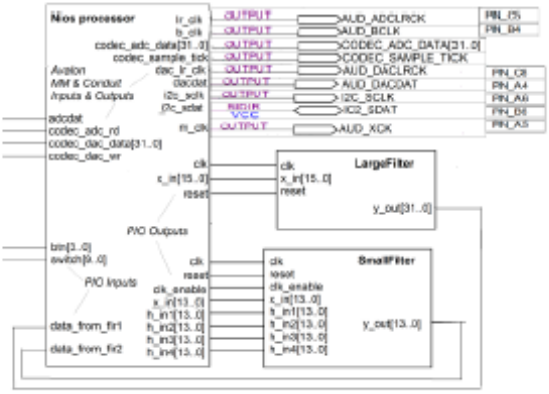


Figure 1: Part of the top-level schematic diagram of the FIR audio system.

3.2.6 Testing

The filters were first tested with an impulse signal, which consisted of 199 zeroes and a one in the middle. This simple test reveals if the filter has some serious problem in it. For this type of input signal the filter should return its own coefficients. In our first test, the filter returned the sum of its coefficients. This error was due to problems in clock rate. After fixing the clock rate, the response was correct.

Next, FIR filtering of a real audio signal was tested. The filters were designed to filter away all frequencies above 1000 Hz. We used an input signal which was a sum of 1000 Hz and 2000 Hz frequencies (Fig. 2 and Fig. 3). The test was performed on both filters. The result of the LargeFilter showed that the frequency of 2000 Hz was clearly attenuated (Fig. 2 and Fig. 3). The response of the SmallFilter was also correct, but the change in the signal was only small due to the low order of the filter, and thus the signal is not shown here.

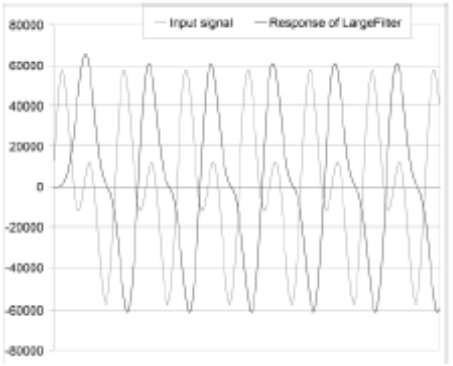


Figure 2: The harmonic input signal (grey) consisting of sine waves at 1000 Hz and 2000 Hz and the response of LargeFilter (black) to the input signal.

Figure 3: The frequency spectra of the input signal (grey squares) and the LargeFilter's response (black diamonds).

However, this signal consisted only of harmonic frequencies. Next, a signal consisting of non-harmonic frequencies was tested. We used a signal which was a sum of 1000 Hz and 5678 Hz frequencies. The generation of the signal was complicated, and thus not exactly the correct shape for the signal was obtained, but an approximation accurate enough was created (Fig. 4 and Fig. 5).

Both LargeFilter and SmallFilter were tested. The results showed that the LargeFilter removed the frequencies above 1000 Hz completely (Fig. 4 and Fig. 5), whereas the SmallFilter removed them only partially due to the low order of the filter (Fig. 4 and Fig. 6). Both results were proved correct when verified with equivalent Matlab filters.

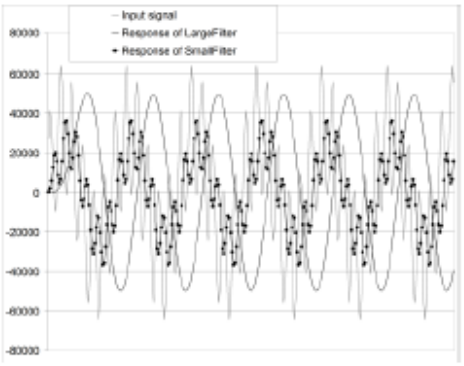


Figure 4: The non-harmonic input signal (grey) consisting of frequencies of 1000 Hz and 5678 Hz, the LargeFilter's response (black) and the SmallFilter's response (black circles).

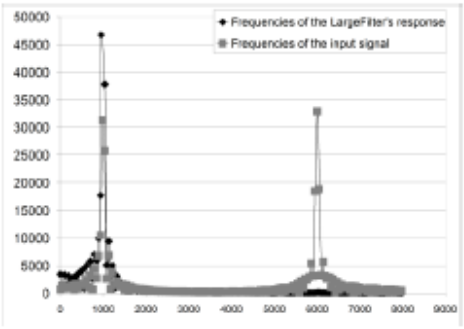


Figure 5: The frequency spectra of the non-harmonic input signal (grey squares) and the LargeFilter's response (black diamonds).

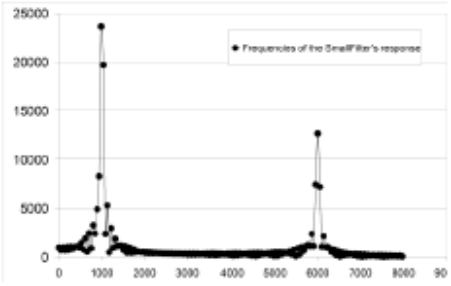


Figure 6: The frequency spectrum of the SmallFilter's response (black circles).

3.3 Experiments with students

The filter system was demonstrated for the students of our bachelor level DSP course. The frequencies of 1000 Hz and 5678 Hz were used. The change of the signal was clearly audible, and the differences between the filters were understood. The benefits of using Matlab HDL Coder for facilitating FPGA development were presented. To further improve the demonstration, the generation of the plots should be made more automatized.

Later in the course the students were given a task to implement a light organ as a project work of the course (Fig. 9). The students had to build a column of LEDs and resistors on a breadboard, and to develop and include a Pulse Width Modulation (PWM) module to a predesigned Quartus project. Although we used a different design than the system presented in this paper, the contents of it were similar excluding the Nios processor and the FIR filters. Thus the demonstration described here was probably helpful in implementing the task. The predesigned project included a functionality to adjust the number of LEDs shining according to the volume, and the students' task was to control the brightness of the LEDs with the PWM module.

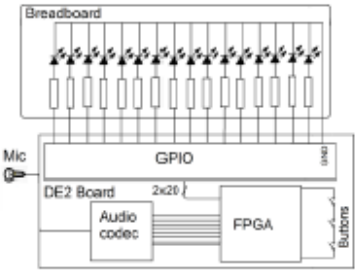


Figure 9: The circuit diagram of the light organ project work.

Some students had problems modifying the project and adding the module to the system. This suggests that a Nios-based approach might be more suitable for similar tasks because connecting modules through PIO is easy with it. Also using a block diagram file (BDF) instead of a textual, i.e. HDL-based, description on the top level of the design would help the students to understand the structure of the design and to make modifications to it.

A feedback survey was carried out in the last project work tutorial. 15 students (~50 % of the participants) answered to the survey. The feedback survey consisted of seven multiple choice questions and three written questions. The answers of the multiple choice questions showed that learning the programming of FPGA is considered useful but difficult (Table 1).

Seemingly the simulations and animations have not yet influenced much on the students' opinion on the usefulness of the lectures, but better results can be expected later when the amount of simulations increase, students get more familiar with them and all of their advantages will be fully exploited.

4 DISCUSSION

Visual perception and feedback is most natural to humans. This also applies in engineering studies including digital electronics and systems studies.

Interaction is definitely increasing in computer aided teaching in many ways. The contemporary computer game student generation will probably welcome also more entertaining computer game approaches within more traditional study material. In this work we have implemented a very simple keyword drilling game approach originally designed for weekly exams that we call microexams. This could be made not only more vivid and entertaining but more fit to learning by drill.

In our university engineering studies have the following master level main subjects: automation (A), computer science (C), electrical engineering (EE), energy technology (ET), and telecommunications (T), which are grouped in to two programs for bachelor level studies: ACT and EE&ET. However, the curricula are planned so that the students have as much basic studies in common as possible. Thus the contents of the basic courses should be such that they support the studies of all these branches of engineering. Therefore the simulations also contain topics more or less relevant not only to digital electronics and its application but also electrical engineering, energy technology, computing in general and data transmission devices.

5 CONCLUSIONS

In this work we have described some simple multimedia and hardware tools designed for aiding the teaching of basics of modern digital electronics circuits and implementation by programming FPGA circuits. Many of our simulation tools are based on web technology and are available via Internet. The FIR system, on the contrary, runs on an FPGA board along with a PC, and is aimed for demonstrating FPGA-hardware-based digital signal processing and introducing the code development process with a HDL coder.

5.1 Future

Our examples are related to understanding the basics of digital electronics and signal processing for the design (engineering) of applications. In practise it is important not only to understand the intended correct behaviour but also verify that the implemented designs fulfil specifications, which is a challenging task for any more complex designs. Our future plans include tools for this verification and test phase using soft computing and especially evolutionary algorithms based test automation (Mantere et al., 2005).

It seems that pipelining, which is essential when implementing powerful computation systems on FPGAs, is not fully understood by programmers having pure software development background. Therefore our plans for future simulation tools include also presentation and modeling of basic pipelining architecture for efficient use of digital logic.

Table 1: The results of the survey. Notations: *SD*=Strongly Disagree, *D*=Disagree, *N*=Neutral, *A*=Agree, *SA*=Strongly Disagree.

Statements	Percentage of students				
	SD	*D*	*N*	*A*	*SA*
I understand the theory of PWM	20	20	20	33	7
I can program PWM with VHDL	20	27	40	13	0
I understand the theory of I2C bus	46	47	7	0	0
I can use the ready programmed I2C code	33	47	13	7	0
Lectures helped me to learn digital technology	0	27	60	13	0
Exercises helped me to learn VHDL programming	0	13	0	74	13
Understanding FPGA technology is useful	0	13	20	40	27

For embedded systems and SoC we plan to replace Nios by ARM IP core, which gives potential to use a large variety and volume of software together with reconfigurable hardware intensive computing.

ACKNOWLEDGEMENTS

The authors acknowledge the Regional Council of Ostrobothnia (project Teho-FPGA-I) which funded the project. Anonymous referees are acknowledged because their critique helped to clarify the pedagogical motivation and ideas of the paper.

REFERENCES

Ainsworth, Shaaron, 2008. How do animations influence learning? In *Recent Innovations in Educational Technology that Facilitate Student Learning*, Daniel H. Robinson (ed.), Information Age Publishing Inc.

Alander, J.T. & Salo, M., 1998. Digitaalisten piirien yksinkertainen graafinen simulaattori DIGI [DIGI: a simple graphical simulator of digital circuits]. In: *Tuntematon tietoyhteiskunta? Interaktiivinen teknologia koulutuksessa* [Proceedings of the Finnish Conference on Interactive Computing in Pedagogics], p. 106, Varpu Kuuliala & Elina Suojoki (eds.), Hämeen kesäyliopisto, Hämeenlinna, 17.-18.4. 1998.

Altera (2010). *Embedded Design Handbook*. http://www.altera.com/literature/hb/nios2/edh_ed_handbook.pdf

Altera (2008). *Using the SDRAM Memory on Altera's DE2 Board with VHDL Design*. http://www.cs.columbia.edu/~sedwards/classes/2010/4840/tut_DE2_sdram_vhdl.pdf

Chu, Pong P., 2011. *Embedded SoPC Design with Nios II Processor and VHDL Examples*. John Wiley & Sons. New Jersey.

Delazer, M., Ischebeck, A., Domahs, F., Zamarian, L., Koppelstaetter, F., Siedentopf, C.M.L. Kaufmann, L., Benke, T., Felber, S., 2005. Learning by strategies and learning by drill. *NeuroImage* 25:3, 838–849.

Donovan, J., 2002. *The truth about 300mm*. http://www.eetimes.com/document.asp?doc_id=1145296.

Drutarovský, M., Šaliga, J., Michaeli, L., Hroncová, L., 2009. Remote laboratory for FPGA based reconfigurable systems testing. *XIX IMEKO World Congress Fundamental and Applied Metrology*, pp. 54-58

Duckworth, R. J., 2005. Embedded system design with FPGA using HDL (lessons learned and pitfalls to be avoided). In *MSE '05, Microelectronic Systems Education, 2005. Proceedings. 2005 IEEE International Conference on*, pp.35-36.

Haba, C.-G., 2014. Using FPGA development boards for multi-course laboratory support. In *EDUCON, Global Engineering Education Conference, April 2014*. pp. 794-797. IEEE.

Hall, T.S., Anderson, D.V., 2002. From algorithms to gates: developing a pedagogical framework for DSP hardware design. *Digital Signal Processing Workshop, 2002 and the 2nd Signal Processing Education Workshop. Proceedings of 2002 IEEE 10th*, 157:161, 13-16.

Hall, T.S., Anderson, D.V., 2007. Teaching Hardware Design of Fixed-Point Digital Signal Processing Systems. *Proceedings of the 2007 American Society for Engineering Education Annual Conference & Exposition*. pp. ASEE.

Mantere, T., Alander, J.T., 2005. Evolutionary software engineering, a review. *Applied Soft Computing* 5:3, 315-331.

MathWorks®, 2014. *Basic HDL Code Generation with the Workflow Advisor*. http://www.mathworks.se/help/hdlcoder/examples/basic-hdl-code-generation-with-the-workflow-advisor.html.

Meyer-Baese, U., 2007. *Digital Signal Processing with Field Programmable Gate Arrays*. Springer. Berlin, 3rd edition.

Reyes, R.S., Oppus, C.M., Monje, J.C.N., Patron, N.S., Gonzales, R.A., Fajardo, J.T.B., 2009. FPGA-Based Digital Signal Processing Trainer, *Computer Science and Information Engineering, 2009 WRI World Congress on* , 343-347,

Soares, J., Lobo, J. A Remote FPGA Laboratory for Digital Design Students. In *7th Portuguese Meeting on Reconfigurable Systems (REC 2011), Feb 2011*, pp 95-98.

Yuxi, Z., Li, K. Jun, W., Jinping S., Zulin, W., 2010. Methods and experience of using Matlab and FPGA for teaching practice in digital signal processing. In *International Conference on Education and Management Technology (ICEMT), Nov 2010*.

Quintans, C., Valdes, M.D., Moure, M.J., Fernandez-Ferreira, L., Mandado, E., 2005. Digital electronics learning system based on FPGA applications. In *FIE '05, Frontiers in Education, 2005. Proceedings 35th Annual Conference* , pp. S2G-7.