TIETO- JA SÄHKÖTEKNIIKAN TIEDEKUNTA

ELEKTRONIIKAN JA TIETOLIIKENNETEKNIIKAN TUTKINTO-OHJELMA

KANDIDAATINTYÖ

Verilog IP-lohkon liittäminen RISC-V Mikrokontrolleriin

Tekijä Nestori Mutanen

Ohjaaja Juha Häkkinen

Helmikuu 2025

Mutanen N. (2025) Verilog IP-lohkon liittäminen RISC-V mikrokontrolleriin. Oulun yliopisto, tieto- ja sähkötekniikan tiedekunta, elektroniikan ja tietoliikennetekniikan tutkinto-ohjelma. Kandidaatintyö, 20 s.

TIIVISTELMÄ

Tässä työssä toteutettiin Verilog-lohkon liittäminen RISC-V mikrokontrolleriin. Liittäminen vaati RISC-V-mikrokontrollerin rakenteeseen tutustumista ja sen rekisteritasoon perehtymistä.

Toteutus tehtiin niin, että RISC-V-mikrokontrolleriin lisättiin PWM-lohko, jota voitiin ohjata ajettavasta koodista Alhambra II -kehitysalustan FPGA-piirissä.

Avainsanat: RISC-V, Verilog, IP-lohko, FPGA.

Mutanen N. (2025) Connecting a Verilog IP block to a RISC-V Microcontroller. University of Oulu, Faculty of Information Technology and Electrical Engineering, Degree Programme in Electronics and Communications Engineering. Bachelor's Thesis, 20 p.

ABSTRACT

In this thesis, a Verilog-block was implemented into a RISC-V microcontroller. This work required researching the structure of the RISC-V microcontroller and its register-level operation.

The implementation involved integrating a PWM block to the RISC-V microcontroller, which could be controlled by executable code in the FPGA of the Alhambra II development board.

Key words: RISC-V, Verilog, IP-block, FPGA.

SISÄLLYSLUETTELO

ALKULAUSE

Kiitos Juha Häkkiselle opastuksesta tämän työn tekemiseen.

Oulussa 16.2.2025

Nestori Mutanen

LYHENTEIDEN JA MERKKIEN SELITYKSET

FPGA	Field-Programmable Gate Array
ISA	Introduction Set Architecture
IP	Intellectual Property
ISA	Introduction Set Architecture
PiM	Processing in Memory
PWM	Pulse Width Modulation
RISC-V	Reduced Introduction Set Computer

1 JOHDANTO

Erilaisten vaihtoehtojen kasvaessa, kuluttaja voi yhä tarkemmin vertailla samankaltaisten tuotteiden ominaisuuksia. Perinteisten mikroprosessoriarkkitehtuurien ARM:n ja x86:n lisäksi on kehitetty useita muitakin mikroprosessoriarkkitehtuureja, mutta avoimeen lähdekoodiin perustuvia on hyvin vähän. Tässä kontekstissa avoimen standardin RISC-V on nostanut houkuttelevuuttaan mikrokontrolleria projekteissaan tarvitsevien kuluttajien keskuudessa, koska se tarjoaa joustavuutta, jota suljetuista järjestelmistä ei löydy.

Ongelmana RISC-V:n käytössä on kuitenkin se, että se ei tarjoa kovin monipuolisia ominaisuuksia sellaisenaan. Vaikka RISC-V on avoimen lähdekoodinsa myötä houkutteleva, sen käyttäminen esimerkiksi infrapunalähettimen ohjaamiseen on vaikeaa, koska RISC-V ei tarjoa kattavia ominaisuuksia oletuksena. Tämä vaatisi sellaisen lohkon lisäämistä RISC-V:hen, joka sisältäisi tarvittavan enkoodauslogiikan. Kuitenkin avoin standardi mahdollistaa muutokset mikrokontrollerissa, mikä antaa käyttäjälle mahdollisuuden lisätä mikrokontrolleriin omia lohkoja. Näin heräsi kysymys, kuinka omien lohkojen lisääminen RISC-V-mikrokontrolleriin onnistuu RISC-V:n ominaisuuksien laajentamiseksi.

Tässä työssä tutustuttiin aluksi RISC-V-mikrokontrolleriin, jonka jälkeen käytiin läpi työssä käytettävän kehitysympäristön ja siihen soveltuvan FPGA-piirin. Sitten työssä käytiin läpi, miten lohko liitetään RISC-V mikrokontrolleriin. Lisäksi kerrotaan, mitä edellä mainituilla laitteilla ja ohjelmilla tehtiin ja lopuksi vielä minkälaisia testituloksia saatiin.

2 RISC-V-ARKKITEHTUURI

2.1 RISC-V

RISC-V on avoimeen lähdekoodiin perustuva käskykanta-arkkitehtuuri (ISA), joka tarjoaa käyttäjille uusia mahdollisuuksia kehittää prosessoreita avoimen standardinsa myötä. RISC-V:n arkkitehtuuri perustuu modulaarisuuteen ja laajennettavuuteen. Toisin kuin suljetut tai patentoidut arkkitehtuurit, avoimen lähdekoodin RISC-V mahdollistaa käyttäjän luoda räätälöityjä prosessoreita tiettyihin käyttötarkoituksiin, toisin kuin useimmat omistusoikeudelliset käskykantaprosessorit. Suljetut arkkitehtuurit kuten x86 ja ARM rajoittavat arkkitehtuuriensa käyttöä korkeilla lisenssimaksuilla. Lisäksi näiden lisenssien käyttöehdoissa on usein määritelty tarkkaa millaisia muutoksia asiakas voi tehdä. Usein pääsy laitteistotason muokkaukselle on estetty. Näistä syistä haluttiin kehittää lisenssivapaa ratkaisu, jonka pohjaksi RISC kehitettiin ja myöhemmin sen viides versio RISC-V. [1] [3]

RISC-V:n kehitys sai alkunsa Berkeleyn yliopistosta Kaliforniassa vuonna 2010. Professori David Patterson ohjasi viisi vuotta kestänyttä rinnakkaislaskemiseen liittyvään projektia, jonka tutkimuksen edistämiseksi kehitettiin RISC-V. RISC-V:n käyttö on yleistynyt viime vuosina. Kiinnostus RISC-V:hen on kasvanut koska se tarjoaa laajan muokattavuuden, sekä sen modulaarisuuden myötä. Odotettavissa on, että RISC-V:n rooli prosessoriteollisuudessa kasvaa merkittävästi seuraavien vuosien aikana. Lisäksi odotettavissa on uusia tutkimuksia RISC-V:n kehitykseen. Uutta kehitystä RISC-V:hen tarvitaan, koska se ei vielä sovellu kaikkiin skenaarioihin. RISC-V on jo saavuttanut suosiota sulautettujen prosessorien alalla, mutta se ei vielä pärjää esimerkiksi pilvilaskennassa tai tietojenkäsittelyssä. RISC-V:n kehitystä tuetaan erityisesti Kiinassa. [1] [2]

Nykyisin RISC-V tukee 32, 64 ja 128 bittisiä toteutuksia. Arkkitehtuuri sisältää perusohjekannat, jotka soveltuvat laajennuksiin avoimen lähdekoodin ansioista. Ohjelmien lisääminen on tehty yksinkertaiseksi, sillä RISC-V:tä voidaan ohjelmoida C-kielen avulla. Tätä varten voidaan hyödyntää olemassa olevia C-kääntäjiä, ja siksi RISC-V:n oman kokoonpanokielen opettelu ei ole välttämätöntä. Laajennuksia varten on tärkeää ottaa huomioon, että RISC-V ei ole valmis prosessori, vaan ennemminkin joukko spesifikaatiota, mitkä määrittelevät sen ohjekannat. Tästä syystä laitteiston kehitys RISC-V:hen onnistuu vain, jos lisätyt ohjelmat eivät ylitä RISC-V:n ohjekannan käskyjoukkoa. [3]

2.2 Avoin standardi

Avoin standardi tarkoittaa systeemiä, jonka tarkoitus on edistää yhteensopivuutta eri toimijoiden välillä. Se eroaa suljetuista standardeista siten, että se on julkisesti saatavilla ja että sen käyttöön ei liity lisenssimaksuja. Julkisen saatavuuden lisäksi avoin standardi mahdollistaa aina käyttäjän tehdä siihen muutoksia, tietyin reunaehdoin. Avoimien standardien lähtökohtana on, että se mahdollistaa ohjelman toimimisen useilla eri alustoilla, mikä vähentää riippuvuutta yksittäisistä toimijoista. [4]

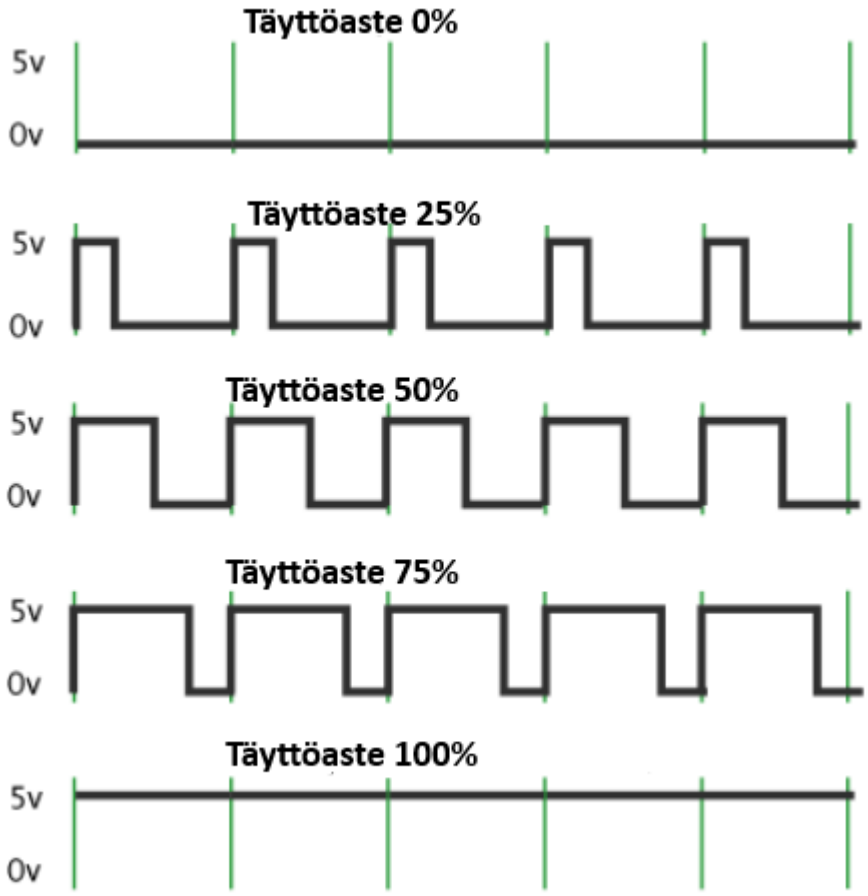
Myös RISC-V-arkkitehtuurin avoin standardi lisää käyttäjän vapautta tehdä siihen muutoksia. Tämä mahdollistaa erilaisten IP-lohkojen integroinnin osaksi arkkitehtuuria. IP-lohkojen lisääminen mahdollistaa käyttäjien tuoda omiin projekteihinsa lisäominaisuuksia lisenssivapaasti. Ongelmana usein on, että käyttäjän pitää itse selvittää, miten lohkon liittäminen tehdään.

Yksi tapa integroida IP-lohko on käyttää RISC-V mikrokontrollerin muistiväylää. Tämä voidaan toteuttaa hyödyntämällä PiM (Processing in Memory) -yksiköitä. Tässä lähestymistavassa PiM-yksiköt voidaan liittää suoraa mikrokontrolleriin ja niitä voidaan ohjata muistiväylän kautta. [5]

Toinen tapa integroida IP-lohko mikrokontrolleriin käsin on käyttää arkkitehtuurissa olevia rekisterikartoituksia. Rekisterissä voidaan määritellä bitit, jotka liittävät ulkoiset lohkot ja mikrokontrollerin toisiinsa. Rekisterin bittien kautta mikrokontrolleri voi ohjata siihen liitettyjä lohkoja, lukea niiden tilatietoja ja käynnistää toimintoja. RISC-V:ssä on ennalta määritelty kuinka monta bittiä sen muisti- ja rekisteriväylät sisältävät. IP-lohkon liitämisessä keskeistä on määritellä mitä rekisteribittejä lohkon ohjaamiseen käytetään. [6]

2.3 PWM

Pulssileveysmodulaatio (PWM) on tekniikka, jossa pulssimuotoisen aaltomuodon jaksoa säädetään toisen signaalin avulla. Jokaisessa jaksossa on päälläolojakso ja pois päältä -jakso. Vertailujännitteen ja kantaja-aallon leikkauskohdat määrittävät avautumis- ja sulkeutumisajat. Täyttösuhde kertoo, kuinka paljon signaali on päällä. PWM-signaalia on havainnollistettu kuvassa 1. [7]



Kuva 1. PWM:n toiminta eri täyttöasteilla.

PWM:ää käytetään laajasti sovelluksissa, kuten moottoreiden nopeuden säädössä, muuntimissa, audio-vahvistimissa ja valojen himmentämisessä. Esimerkiksi PWM:n avulla

voidaan vähentää kuormalle toimitettavaa kokonaistehoa ilman häviöitä, joita normaalisti syntyy, kun tehonlähdettä rajoittaa vastus. PWM:ää käytetään säätämään moottorille syötettävää jännitettä. Kytkinten työjakson muutos vaikuttaa moottorin nopeuteen: mitä pidempi pulssin suljettu aika on suhteessa avoimeen aikaan, sitä suurempi on kuormalle toimitettu teho. Lisäksi PWM-tekniikkaa hyödynnetään LED-valojen kirkkauden säädössä. Siinä valonlähteen virtaa katkaistaan ja kytketään nopeasti tietyllä taajuudella, jolloin valo näyttäytyy himmeämpänä, kuin valo, joka on koko ajan kytkettynä päälle. [7] [8]

PWM-tekniikkaa on kehitetty 1970-luvulta lähtien vastaan mikroprosessorien vaatimuksia. Eri sovelluksiin on kehitetty erilaisia PWM-menetelmiä. PWM:n etu on sen energiatehokkuus. Avoimen ja suljetun tilan vaihtamiseen ei tarvita jatkuvaa virransyöttöä, mikä minimoi energiahäviöt. PWM-tekniikkaa hyödyntävät komponentit ovat myös kooltaan pienempiä kuin vastaavat komponentit, jotka eivät hyödynnä pulsseja. [7]

3 FPGA-KEHITYS

3.1 Alhambra II

Tässä työssä käytetään kehitysalustana Alhambra II FPGA-piiriä. Alhambra II on avoimeen lähdekoodiin perustuva, mikä antaa joustavuutta digitaalisten piirien suunnitteluun. Alhambra II on tähän työhön sopiva alusta, koska Alhambra II sisältää iCE40HX4K FPGA-piirin, joka mahdollistaa alempana kuvatun avoimen lähdekoodin suunnitteluohjelmiston käytön. Alhambra II toimii Linux-, macOS- ja Windows-ympäristöissä. Lisäksi Alhambra II on suunniteltu yhteensopivaksi Icestudio v0.3.3:n [10] ja sitä uudempien versioiden kanssa. Sen ohjelmointi tapahtuu helposti USB-liitännän kautta. Piirissä on useita ohjelmoitavia LED:jä sekä pinnejä, joista voidaan lukea niihin tulevaa signaalia. Alhambra II -piiri näkyy kuvassa 2. [9]

Työssä tarvittiin tällaista kehitysalustaa, jotta RISC-V:tä voitiin toteuttaa käytännössä. FPGA-piiri toimi siis alustana, johon RISC-V ladattiin. Alhambran II:n kautta tapahtui kommunikointi RISC-V:n kanssa. Tässä työssä voitiin Alhambra II:n lähdöistä mitata PWM-signaali, jonka RISC-V-mikrokontrolleriin lisätty lohko tuotti RISC-V:n ohjaamana.

Kuva 2. Alhambra II.

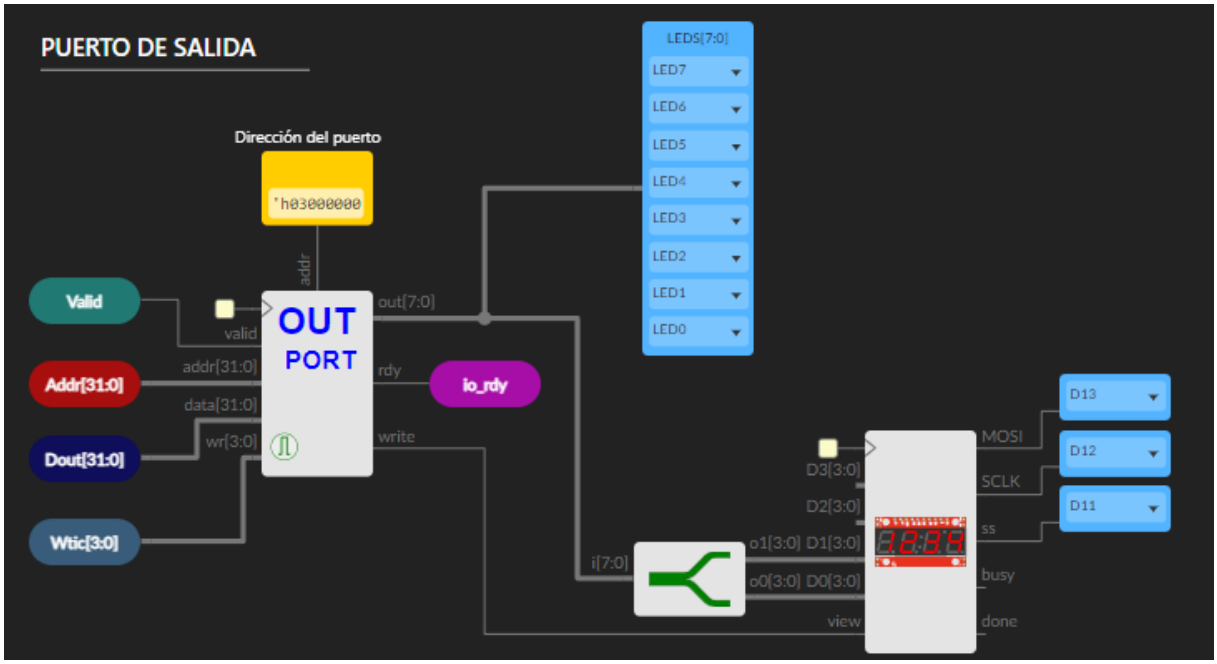
3.2 Icestudio

Icestudio on avoimeen lähdekoodiin perustuva sovellus, joka tarjoaa kehitysympäristön avoimen lähdekoodin FPGA-piireille. Icestudio mahdollistaa käyttäjän syntetisoida suunnitelmansa FPGA-piirille. Sovellus mahdollistaa käyttäjän luoda digitaalisia piirejä ja ohjelmoida niitä, mutta sen käyttäminen ei vaadi vahvaa ohjelmointitaitoa. Laitteistoja luodessa Icestudio luo verilog-koodia, josta Icestudio muodostaa pcf-tiedoston. Icestudiossa laitteiston kehittämiseen käytetään työkalua nimeltä Apio, joka on Pythonilla kirjoitettu avoimeen lähdekoodiin perustuva ohjelma. Apion avulla tapahtuu myös Verilog-suunnitelmien varmentaminen, simuloiminen ja lataaminen fyysiselle alustalle. [10]

Icestudio tarjoaa yksinkertaisen graafisen käyttöliittymän, johon pystyy suunnittelemaan piirejä vedä ja pudota -periaatteella, mikä tekee suunnittelusta yksinkertaista. Käyttäjän on mahdollista luoda ohjelmoitavia lohkoja, joihin voi lisätä vapaasti sisään- ja ulostuloväyliä FPGA-piirin pinneille.

Tässä työssä käytettiin FPGAwarsin [11] RISC-V-pohjaa, joka syntetisoitiin Alhambra II alustalle. Tällä RISC-V-pohjalla suoritettiin myös Verilog-lohkon liittäminen. [10]

FPGAwarsin RISC-V-malli sisältää lähtöportin, joka on osa systeemin tulo- ja lähtöporttirakennetta. Portti kommunikoi ulkoisten laitteiden, kuten LEDien ja digitaalisten näyttöjen kanssa, kuten kuvassa 3 esitetään. Kyseinen ulostuloportti on leveydeltään kahdeksan bittiä, eli sillä voidaan ohjata esimerkiksi kahdeksaa LEDiä yhtä aikaa. Tämän ulostuloportin kautta voidaan ulkoisia komponentteja liittää osaksi RISC-V:tä.

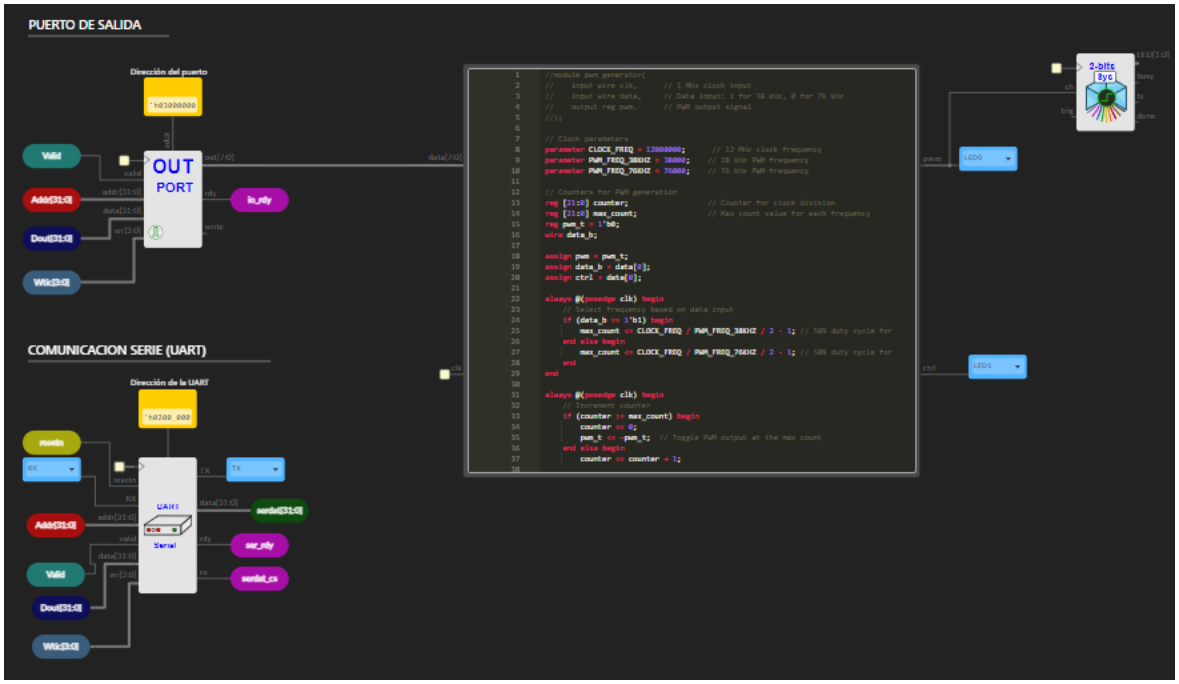


Kuva 3. RISC-V:n ulostuloportti.

4 TOTEUTUS

4.1 Työn suunnitteleminen Icestudiossa

Työn suunnitteleminen tehtiin Icestudiossa. Työssä haluttiin lisätä FPGAWarsin [11] valmiille RISC-V-pohjalle oma räätälöity lohko, sekä ottaa selvää, miten sen lisääminen tähän valmiiseen pohjaan onnistuu. Työssä luotiin graafinen lohkokaavio, joka tuottaa PWM-signaalia, jota oli tarkoitus ohjata RISC-V-mikrokontrollerista käsin. Lohkon liittämisessä hyödynnettiin RISC-V-pohjalla olevaa valmista väylää, jonka kautta informaatio siirtyy RISC-V-mikrokontrollerin ja PWM-lohkon välillä. Tämä väylä on RISC-V-pohjassa oleva kahdeksan bittinen ulostuloportti. FGPAwarsin RISC-V-mallissa tuohon ulostuloporttiin oli valmiiksi liitetty muun muassa LED-valoja, sekä seitsemänsegmenttinäyttö. Tässä työssä nämä komponentit korvattiin kustomoidulla PWM-lohkolla. PWM-lohko kytkettiin rekisterin ensimmäiseen bittiin. PWM signaali kytkettiin Alhambra II:n LEDiin, jonka numero oli nolla. Työhön lisättiin vielä mittapää, josta PWM-signaalin vaihtelua pystyy mittamaan. Liitetty lohko näkyy kuvassa 4.



Kuva 4. PWM-lohko liitettynä RISC-V:hen.

PWM-lohko sisältää ohjelman, jonka toteutukseen käytettiin Verilog-kieltä. Ohjelma tuottaa PWM-signaalia kahdella eri ohjelmoitavalla taajuudella. Valittu taajuus riippuu ohjaussignaalista, joka on yhdistetty lohkon tuloporttiin. Tuotettu signaali päivittyy jokaisella positiivisella kellonreunalla. Koodissa määritellään, kuinka monta kellonjaksoa tarvitaan, jotta signaalin tila vaihtuu. Pulssin leveydeksi määriteltiin 50 %. Ohjelma on esitelty kokonaisuudessaan liitteessä 1.

Kellotaajuus määriteltiin lisätyssä lohkossa 12 MHz:iin. Tämä kellotaajuus määrittelee ulkoisen kellosignaaliin, joka ohjaa koko järjestelmää. Lisäksi lohkossa määriteltiin parametrit kahdelle taajuudelle, joita PWM-signaali käyttää. Lohkossa määriteltiin myös parametrit kolmelle eri rekisterille. Yksi rekistereistä laskee kellopulssit PWM-signaalin tuottamista

varten. Toiseen rekisteriin lasketaan se kellopulssien määrä, joka tarvitaan yhden PWM-jakson tuottamiseen. Tämä kellopulssien määrä riippuu PWM-signaalille valitusta taajuudesta. Viimeinen rekisteri toimii PWM-signaalina, jonka arvo vaihtuu 1:n ja 0:n välillä.

Ohjelmaan tehtiin myös osa, joka aktivoituu aina positiivisella kellonreunalla. Tässä osassa ohjaussignaalin sen hetkinen arvo määrää PWM-signaalin taajuuden. Lisäksi tähän osaan laitettiin kaava, joka laskee PWM-signaalin kummallekin taajuudelle 50 % käyttöasteen. Tämän lisäksi ohjelmaan tehtiin vielä lohko, joka tuottaa PWM-signaalia. Tässä lohkossa määriteltiin laskuri, joka laskee kellopulssien määrää. Lohko käynnistää PWM-signaalin tuoton, kun kellopulssien määrä on riittävän suuri PWM-signaalin tuottamiseen. Tällöin myös laskuri nollataan.

4.2 Kehitysympäristö

Toiminnallisuuden testaamista varten luotiin kehitysympäristö, jossa PWM-lohkon toimintaa pystyttiin havainnoimaan. Kehitysympäristöön rakennettiin hyödyntämällä Linux-pohjaista Ubuntu-käyttöjärjestelmää. Kehitysympäristöön asennettiin Python-työkalut ja Apio. Lisäksi RISC-V-pohja liitetyn lohkon kanssa lähetettiin Icestudiossa Alhambra II -piiriin USB-porttia pitkin.

4.3 C-koodin lähettäminen FPGA-korttiin

Jotta RISC-V-mikrokontrollerille liitettyä lohkoa voitiin ohjata, täytyi kehitysalustalle lähettää ohjelma. Tässä työssä käytettiin C-kielistä ohjelmaa. Tämä tehtiin käyttämällä RISC-V:n kehitystyökaluja, sekä Icestudio-ohjelmistoa. Lähetetty C-kielinen ohjelma on esitetty liitteessä 2.

Koska Alhambra II ei osaa suoraan lukea C-koodia, C-ohjelma täytyi kääntää RISC-V:n ymmärtämään muotoon. Tätä varten C-koodi piti kääntää binäärimuotoon. C-ohjelma käännettiin C-kääntäjällä, jonka tuloksena saatiin suoritettava binääritiedosto. Käännetty C-koodi lähetettiin komentokehotteesta Alhambra II -piirille.

4.4 Testaus

C-kielinen ohjelma lähetettiin syöttämällä komentokehotteeseen komento, joka lähetti binäärimuotoisen tiedoston Alhambraan. Tiedosto lähetettiin komennolla ” apio raw "iceprog -o 1M test.bin"”n joka lähetti ”test.bin” nimisen binääritiedoston. Sitten työssä avattiin Arduinon terminaaliohjelma [12]. Terminaaliohjelmasta seurattiin RISC-V:n ja Alhambra II:n välistä kommunikaatiota. Työssä ulostuloportteina toimivat kaksi LEDiä, joista RISC-V:n tuottamaa PWM-signaalia voitiin havainnoida. Työssä PWM-signaalin taajuudet ohjelmoitiin siten, että niiden vaihtelua pystyttiin havainnoimaan silmämääräisesti. Tämä varmisti, että RISC-V:hen lisätty lohko toimi oikein ja että se oli liitetty RISC-V-mikrokontrolleriin halutulla tavalla.

5 POHDINTA

RISC-V-mikrokontrollerin rekisteritasoinen ohjelmointi on tehty mahdolliseksi hyödyntäen melko yksinkertaisia työkaluja. Icestudio tarjoaa paljon työkaluja RISC-V:n laajentamiseen.

Työssä opittiin RISC-V:n arkkitehtuurista, ja FPGA-alustojen toiminnasta. Työtä voitaisiin kehittää eteenpäin esimerkiksi tutkimalla, miten RISC-V:hen lisätyn lohkon tuottamaa signaalia voidaan mitata FPGA:n pinneistä käyttämällä logiikka-analysaattoria, ja miten signaalia voisi visualisoida. Myös muita liittämistapoja omien lohkojen lisäämiseen olemassa oleviin avoimen lähdekoodin käskykantoihin olisi mahdollista tutkia vielä lisää.

6 YHTEENVETO

Työn tavoitteena oli liittää Verilog-lohko osaksi RISC-V-mikrokontrolleria. Aluksi työssä tutustuttiin RISC-V mikrokontrollerin ominaisuuksiin. Pääpaino oli sen laajentamiseen liittyvissä teemoissa.

Tarvittavat ohjelmistot, sekä oheislaitteet käytiin läpi. Käytetty FPGA, sekä kehitysympäristö FPGA-piirille esitettiin. Pääpaino tässä oli esitellä tarvittavaa työkaluketjua RISC-V:n laajentamiseen.

Laajentamisen toteutus käytiin läpi. Koodi-lohko ja sen liittäminen RISC-V:hen käsiteltiin. RISC-V:n ohjaaminen ja sen lohkon lähettämän signaalin havainnointi visuaalisesti kerrottiin.

7 LÄHDELUETTELO

[1] Kanter, D (2016). RISC-V OFFERS SIMPLE; MODILAR ISA New CPU Instruction Sey Is Open and Extensible.

[2] A Cui, E., Li, T., & Wei, Q. (2023). RISC-V Instruction Set Architecture Extensions: A Survey. *IEEE Access*, *11*, 24696–24711.

[3] Blomkvist, L., Oscarsson, J. I., Nilsson, L., Stenseke, A., & Df, J. W. (n.d.). Implementation of a RISC-V Processor with Hardware Accelerator.

[4] Perens, B., & Sroka, M. (2007). *The Open Source Definition*. http://perens.com/OSD.html

[5] Verma, V., & Stan, M. R. (2022). AI-PiM—Extending the RISC-V processor with Processing-in-Memory functional units for AI inference at the edge of IoT. *Frontiers in Electronics*, *3*.

[6] Patrick Schaumont. (2019, September 12). *Lecture Notes on Memory-Mapped Registers*.

[7] Tran, P. H. (2012). *MATLAB/SIMULINK IMPLEMENTATION AND ANALYSIS OF THREE PULSE-WIDTH-MODULATION (PWM) TECHNIQUES*.

[8] Yifeng W., Xiachen W., Yugi H. (2020). *Full-Range LED Dimming Driver With Ultrahigh Frequency PWM Shunt Dimming Control*.

[9] Alhambra II. Haettu 13.2.2025 osoitteesta https://alhambrabits.com/alhambra/

[10] IceStudio. Haettu 13.2.2025 osoitteesta https://icestudio.io/#lk-overview

[11] FPGAwars. Haettu 16.1.2024 osoitteesta https://fpgawars.github.io/

[12] Arduino serial monitor. Haettu 15.2025 osoitteesta https://docs.arduino.cc/software/ide-v2/tutorials/ide-v2-serial-monitor/

8 LIITELUETTELO

Liite 1 Käytetty ohjelma PWM-modulaatioon
Liite 2 Käytetty C-kielinen ohjelma

Liite 1 Käytetty ohjelma PWM-modulaatioon

```
// Clock parameters
parameter CLOCK_FREQ = 12000000;      // 12 MHz clock frequency
parameter PWM_FREQ_38KHZ = 38000;    // 38 kHz PWM frequency
parameter PWM_FREQ_76KHZ = 76000;    // 76 kHz PWM frequency

// Counters for PWM generation
reg [21:0] counter;                  // Counter for clock division
reg [21:0] max_count;                // Max count value for each frequency
reg pwm_t = 1'b0;
wire data_b;

assign pwm = pwm_t;
assign data_b = data[0];
assign ctrl = data[0];

always @(posedge clk) begin
    // Select frequency based on data input
    if (data_b == 1'b1) begin
        max_count <= CLOCK_FREQ / PWM_FREQ_38KHZ / 2 - 1; // 50% duty cycle for 38 kHz
    end else begin
        max_count <= CLOCK_FREQ / PWM_FREQ_76KHZ / 2 - 1; // 50% duty cycle for 76 kHz
    end
end

always @(posedge clk) begin
    // Increment counter
    if (counter >= max_count) begin
        counter <= 0;
        pwm_t <= ~pwm_t;  // Toggle PWM output at the max count
    end else begin
        counter <= counter + 1;
    end
end
```

Liite 2 Käytetty C-ohjelma

```
#include <stdint.h>
#define reg_uart_data (*(volatile uint32_t*)0x02000008)
#define reg_leds (*(volatile uint32_t*)0x03000000)

// ------------------------------------------------------------



// ------------------------------------------------------------

void main()
{
    int i,j;
    while(1)
    {
        reg_leds = 0x00;
        for(i=0;i<100;i++) for(j=0;j<1000;j++);
        reg_leds = 0xff;
        for(i=0;i<100;i++) for(j=0;j<1000;j++);
    }
}
```