Matias Oikari

Vianaikaisten tietojen tallennus taajuusmuuttajassa

Sähkötekniikan korkeakoulu

Diplomityö, joka on jätetty opinnäytteenä tarkastettavaksi
diplomi-insinöörin tutkintoa varten Espoossa 2.11.2012.

Työn valvoja:

Prof. Pekka Eskelinen

Työn ohjaaja:

TkL Antti Hämäläinen

AALTO-YLIOPISTO
SÄHKÖTEKNIIKAN KORKEAKOULU

DIPLOMITYÖN
TIIVISTELMÄ

Tekijä: Matias Oikari

Työn nimi: Vianaikaisten tietojen tallennus taajuusmuuttajassa

Päivämäärä: 2.11.2012 Kieli: Suomi Sivumäärä: 61 + 9

Elektroniikan laitos

Professuuri: Elektroniikka ja sovellukset Koodi: S-66

Valvoja: Prof. Pekka Eskelinen

Ohjaaja: TkL Antti Hämäläinen

Taajuusmuuttaja on sähkömoottorikäytön osa, jota käytetään muun muassa moottorin portaattomaan pyörimisnopeuden säätöön. Tässä työssä esitetään ratkaisuja taajuusmuuttajan sisäisten mittaus- ja ohjaustietojen tallennusta varten. Tiedonkeruu on tärkeää, koska se helpottaa vikojen diagnosoinnissa. Nykyisillä tuotteilla tiedonkeruun toteuttaminen on kuitenkin haasteellista. Taajuusmuuttajasta saattaa katketa virrat vikatilanteen vuoksi, jonka takia aikaa tietojen tallennukseen on hyvin rajoitetusti. Koska tallennettavissa oleva tietomäärä riippuu väylä- ja muistinopeuksista, tutkitaan työssä erilaisia muisti- ja tiedonsiirtotekniikoita.

Työn lopussa esitetty esimerkkitoteutus perustuu tietojen puskurointiin jatkuvasti taajuusmuuttajan ollessa käynnissä. Kun vikatilanne havaitaan, siirretään puskurin sisältö erilliselle muistipiirille. Rengaspuskuri toteutettiin FPGA:han integroidulla SRAM-muistilla, ja tietojen pysyvään tallennukseen käytettiin EPCS-konfigurointilaitetta.

Esimerkkitoteutuksen tavoin toimivaa ratkaisua voidaan hyödyntää nykyisten tuotteiden kanssa, mutta tulevaisuuden tuotteita suunniteltaessa on suositeltavaa käyttää uusia teknologioita, kuten haihtumatonta RAM-muistia, jotta tietoja voitaisiin tallentaa nykyistä enemmän.

Avainsanat: FPGA, tiedonkeräys, muistipiirit, oheisväylä, SPI, taajuusmuuttaja, rengaspuskuri

AALTO UNIVERSITY
SCHOOL OF ELECTRICAL ENGINEERING

ABSTRACT OF THE
MASTER'S THESIS

Author: Matias Oikari

Title: A data logger for frequency converters

Date: 2.11.2012 Language: Finnish Number of pages: 61 + 9

Department of Electronics

Professorship: Electronics and Applications Code: S-66

Supervisor: Prof. Pekka Eskelinen

Instructor: Lic.Sc. (Tech.) Antti Hämäläinen

A frequency converter is a part of an electric drive used for controlling the speed of a motor. This thesis presents solutions for storing internal measurement data during a possible failure in the converter. Data logging is important, because it helps to diagnose the failure but such features are rare in current products. It is possible that a fault condition causes an outage of the entire drive, in which case there is not lot of time to store the data. Because the amount of storable data depends on bus and memory speed, different memory and data transfer technologies are examined.

In the end of the thesis, an example implementation is presented. It relies on buffering data during the normal use of the drive and on storing the data to a separate memory circuit when a fault is detected. A circular buffer is used on embedded SRAM of a FPGA circuit and an EPCS configuration device is used for storing data.

Solutions similar to the example implementation can be exploited with current products. However, with future products, it is recommendable to consider using new technologies, such as non-volatile random access memory (nvRAM), for increasing storage capacity.

Keywords: FPGA, data logger, memory circuit, peripheral bus, SPI, frequency converter, circular buffer

Esipuhe

Tämä diplomityö on tehty yhteistyössä ABB Oy Drives:n High Power Drives -tulosyksikön kanssa. Haluan kiittää työni ohjaajaa TkL Antti Hämäläistä ja valvojaa Prof. Pekka Eskelistä, joiden neuvot ja kritiikki auttoivat työn onnistumisessa. Tämän lisäksi kiitän myös Ossi Myllyniemeä, Anssi Kämäriä ja kaikkia muita työkavereita, joiden avusta oli työn etenemisen aikana korvaamatonta hyötyä.

Kiitän myös ystäviäni, sisaruksiani ja vanhempiani kannustuksesta ja tuesta työn etenemisen aikana.

Espoo, 2.11.2012

Matias Oikari

Sisällysluettelo

LYHENTEET

ADC	Analog-to-digital converter
AS	Active serial
ASIC	Application specific integrated circuit
CHEI	Channel hot electron injection
CLB	Configurable logic block
CLK	Clock
CS	Chip select
DDCS	Distributed drive control system
DRAM	Dynamic random access memory
DSP	Digital signal processor
EEPROM	Electronically erasable programmable read only memory
EMC	Electromagnetic compatibility
EPROM	Erasable programmable read only memory
FET	Field-effect transistor
FIFO	First in, first out
F-N	Fowler-Nordheim
FPGA	Field programmable gate array
FRAM	Ferroelectric random access memory
HD	High-definition
I^2C	Inter-integrated circuit
IC	Integrated circuit
IO	Input/output
IP	Intellectual property
JTAG	Joint test action group
LAB	Logic array block,
LE	Logic element
LED	Light emitting diode
LUT	Look-up table
MAC	Media access control
MISO	Master in, slave out
MMC	Multimedia card
MOSI	Master out, slave in
MRAM	Magnetoresistive random access memory
MSB	Most significant bit
NRAM	Nano random access memory
NRE	Non-recurring engineering
NRZ	Non-return-to-zero
NRZI	Non-return-to-zero inverted

nvRAM	Non-volatile random access memory
nvSRAM	Non-volatile static random access memory
OTP	One time programmable
PAL	Programmable array logic
PC	Personal computer
PCI	Peripheral component interconnect
PHY	Physical layer
PLC	Programmable logic controller
PLL	Phase locked loop
PRAM	Phase-change random access memory
PROM	Programmable read only memory
PTC	Positive temperature coefficient
RAM	Random access memory
RISC	Reduced instruction set computer
RLE	Run-length encoding
ROM	Read only memory
RRAM	Resistive random access memory
RTC	Real-time clock
RTL	Register transfer level
SATA	Serial AT attachment
SD	Secure digital
SDHC	Secure digital high-capacity
SPI	Serial peripheral interface
SRAM	Static random accesss memory
SSD	Solid state drive
USB	Universal serial bus
UWB	Ultra-wideband
VHDL	VHSIC hardware description language
VHSIC	Very high speed integrated circuit
WLAN	Wireless local area network
WUSB	Wireless universal serial bus
µP	Microprocessor

SYMBOLIT

C	Kapasitanssi	(F)
n	Sivujen määrä	
R	Resistanssi	(Ω)
t_1	Kondensaattorin purkautumiseen kulunut aika	(s)
t_D	Kirjoitettavien tietojen siirtämiseen kuluva aika	(s)
t_K	Tallennukseen kuluva kokonaisaika	(s)
t_{OA}	Operaatiokoodin ja osoitteen siirtoon kuluva aika	(s)
t_{WB}	Yhden sivun tallennukseen kuluva aika	(s)
U_0	Kondensaattoriin varautunut jännite alkutilanteessa	(V)
U_C	Kondensaattoriin varautunut jännite purkautumisen aikana	(V)

1 Johdanto

Kun taajuusmuuttajakäytössä ilmenee vikatilanne, on vian diagnosoinnin kannalta hyödyllistä saada ohjauselektroniikan vianaikaiset tapahtumat kerättyä muistiin. Tekniikka vikojen havainnointia varten on jo olemassa, ja hitaita signaaleja sekä vikakoodeja on jo mahdollista tallentaa [1]. Nykyisillä tuotteilla ei voida kuitenkaan tallentaa tehoasteen liitäntäkortin nopeita, alhaisen tason signaaleja. Tämän työn tarkoitus on selvittää, minkälaisia ratkaisuja ja teknologioita voitaisiin näiden alhaisen tason signaalien tallennukseen käyttää. Tiedonkeräys voidaan toteuttaa joko uudella laitteella tai tekemällä muutoksia jo olemassa olevaan tekniikkaan. Työn lopussa esitellään yksi esimerkkitoteutus. Tavoitteena on suunnitella ratkaisu mahdollisimman paljon nykyisten tuotteiden ominaisuuksia hyödyntäen.

Koska nykyisissä tuotteissa ei ole riittävästi siirto- ja tallennuskapasiteettia, ei kaikkia tietoja pystytä välttämättä siirtämään muistiin, vaan tietoja on mahdollista tallentaa vain vikatilanteiden yhteydessä. On myös huomioitavaa, että piirikorttien virrat voivat katketa vikatilanteen vuoksi, jolloin aikaa tietojen tallennukseen on käytännössä vain niin kauan kuin suurien kondensaattorien varaus riittää ylläpitämään tehoja liitäntäkortilla. Voi myös olla, ettei vianaikaisia tietoja ehditä tallentaa tarpeeksi paljoa, jotta vikatilanteen aiheuttaja voitaisiin päätellä. Jos tietoa esimerkiksi halutaan pidemmältä aikaväliltä kuin mihin kapasiteetti riittää, joutuu ratkaisun käyttäjä valitsemaan, mitä tietoja tallennetaan ja mitä ei. Jättämällä yhden signaalin pois voitaisiin toista signaalia tallentaa enemmän.

Työssä on tarkoitus löytää mahdollisimman yleiskäyttöinen ratkaisu, joka on hyödynnettävissä monien eri tuotteiden kanssa suhteellisen vähällä vaivalla. Vaikka tiedonkeruuyksikkö suunnitellaan lähinnä apuvälineeksi tuotekehitykseen, on toivottavaa, että tätä voidaan käyttää myös valmiissa tuotteissa. Kaikkia tässä työssä esittelemiäni toteutusvaihtoehtoja ei ole mahdollista hyödyntää nykyisten tuotteiden kanssa, mutta osaa ratkaisuista on suositeltavaa harkita tulevia tuotteita suunniteltaessa. On myös huomioitava, etteivät testattavat tuotteet saisi poiketa liikaa markkinoilla olevista, koska muutokset voivat osaltaan vaikuttaa taajuusmuuttajan toimintaan ja vikatilanteiden esiintymiseen. Tämän vuoksi ei ole järkevää lisätä prototyyppeihin ominaisuuksia, joita ei valmiissa tuotteissa ole.

Vianaikaisten tietojen tallennus ei vaadi suurta laskentatehoa, koska kerättyjä tietoja voidaan käsitellä jälkikäteen tietokoneella. Työn kannalta tärkeitä tekijöitä ovat väylän tiedonsiirtonopeus sekä muistin tallennusnopeus ja koko, koska nämä vaikuttavat suoraan tallennettavissa olevaan tietomäärään silloin, kun tallennusta varten käytössä oleva aika on rajallinen.

Tässä työssä otetaan vikatilanteiden havainnointia huomioon vain siltä osin kuin tekniikkaa on sitä varten jo olemassa. On kuitenkin valittava, missä tilanteissa ja mitä tietoja halutaan tallentaa. Ei ole myöskään aiheellista pohtia vikojen diagnosointia kovin tarkasti, vaan työ keskittyy lähinnä diagnosointia helpottavien tietojen tallennustapojen selvittämiseen. Ratkaisun voisi olla hyödyllistä tallentaa esimerkiksi seuraavia ohjaus- ja mittaus-signaaleja:

- Vikasignaalit
- Vaihevirrat
- Tehokytkimien lämpötilat
- Tehokytkimien kytkeytymisohjeet

Työssä käsitellään ensin taajuusmuuttajia sekä niiden ohjausta, mittaustekniikkaa ja vikatilanteita. Tämän jälkeen esitellään vikatietojen siirron ja tallennuksen kannalta oleellista tekniikkaa. Neljännessä luvussa tuodaan esille erilaisia tapoja toteuttaa vianaikaisen tiedonkeräys, ja viidennessä luvussa esitellään työssä toteutettu ratkaisu tietojen tallentamiseksi vikatilanteen esiintyessä.

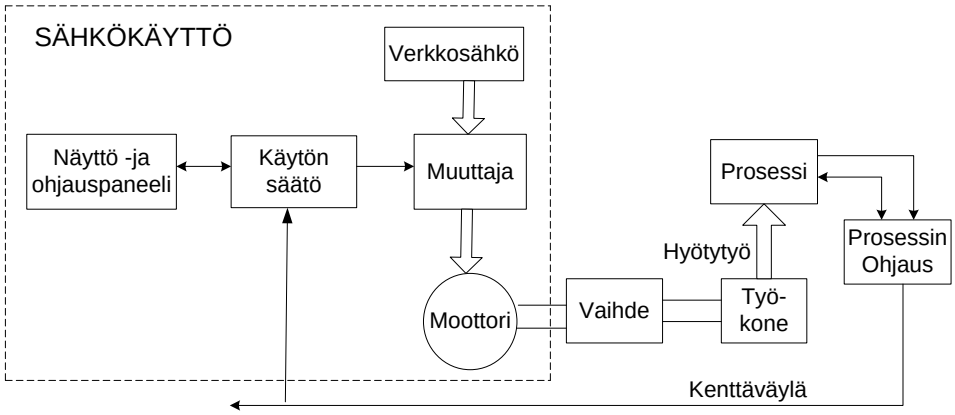
2 Taustaa

2.1 Yritys

ABB on monikansallinen yritys, joka on erikoistunut robotiikkaan, automaatioon ja sähkötekniikkaan. ABB:llä on yli 100 000 työntekijää yli sadassa eri maassa. Helsingin Pitäjänmäessä sijaitseva *ABB Oy Drives* -yksikkö valmistaa taajuusmuuttajakäyttöjä. Pitäjänmäen tehdas kuului alun perin *Oy Strömberg Ab*:lle, mutta ruotsalainen *ASEA* osti sen 1987. *ASEA* ja sveitsiläinen *Brown, Boveri & Cie* yhdistyivät 1988, jolloin ABB syntyi. [2]

2.2 Taajuusmuuttajakäyttö

Sähkökäyttö eli sähkömoottorikäyttö (electric drive) muuntaa sähköverkon energiaa työkoneen kautta hyötytyöksi prosessiin niin, että työkone muuttaa moottorin vääntötyön prosessiin sopivaksi liikkeeksi. Työkoneella voidaan tarkoitta esimerkiksi sähköjunaa tai paperikonetta ja prosessilla junan liikkumista tai tehtaan tuotantolinjan toimintaa. Moottorin ja sähköverkon välillä on muuttaja, joka perustuu puolijohdekytkimillä toteutettuihin tasasuuntaajiin, vaihtosuuntaajiin ja tasasähkökatkojiin. [3] Kuvassa 1 on esitetty sähkökäytön toiminnalliset lohkot.

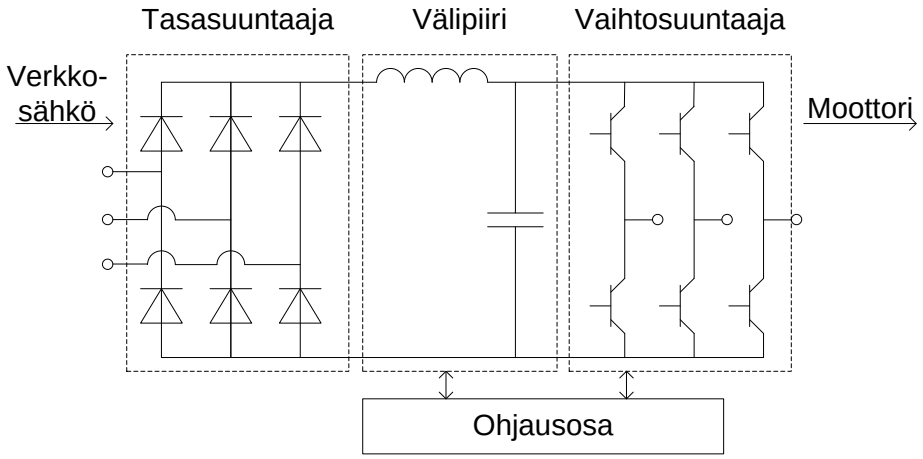


Kuva 1: Sähkökäytön toiminnalliset lohkot. [3]

Taajuusmuuttajaa käytetään moottorin pyörimisnopeuden energiatehokkaassa ja portaattomassa säädössä [4]. Taajuusmuuttajia on sekä välipiirillisiä että välipiirittömiä, joista ensin mainituissa vaihtosähköverkon sähkö muutetaan ensin tasasähköksi ja sitten pilkotaan vaihtosähköksi, kun taas jälkimmäisissä sähkö pilkotaan puolijohdekytkimillä suoraan halutun taajuiseksi ja jännitteiseksi vaihtosähköksi. Suoria taajuusmuuttajatyyppejä

ovat muun muassa matriisimuuttaja ja syklokonvertteri. Tämä työ liittyy erityisesti välipiirillisten taajuusmuuttajien tuotekehitykseen. [3]

Välipiiritaajuusmuuttajia ovat esimerkiksi kuormakommutoitu taajuusmuuttaja, virtavälipiiritaajuusmuuttaja ja jännitevälipiiritaajuusmuuttaja. Välipiirilliset taajuusmuuttajat koostuvat tasasuuntaajasta, tasajännite- tai tasavirtavälipiiristä ja vaihtosuuntaajasta eli invertteristä (kuva 2). Tasasuuntaaja muuttaa verkon vaihtosähköksi tasasähköksi esimerkiksi diodien avulla. Tasavirtavälipiirissä on kuristin, jonka tehtävänä on pienentää tasavirran aaltoisuutta, kun taas tasajännitevälipiirin kondensaattorin tarkoitus on vähentää tasajännitteen aaltoisuutta. Tasajännitevälipiirissä saattaa olla myös tasoituskuristin tasasuuntaajan ja kondensaattorin välissä. Vaihtosuuntaaja muuttaa tasasähkön takaisin vaihtosähköksi tavallisesti kuuden tehokytkimen avulla. [3, 4] Tehokytkiminä käytetään yleisesti IGB-transistoreja (insulated gate bipolar transistor) niiden helpon ohjattavuuden ja korkean kytkentätaajuuden vuoksi. [1]



Kuva 2: Välipiirillinen taajuusmuuttaja koostuu tasasuuntaajasta, välipiiristä ja vaihtosuuntaajasta sekä ohjausosasta.

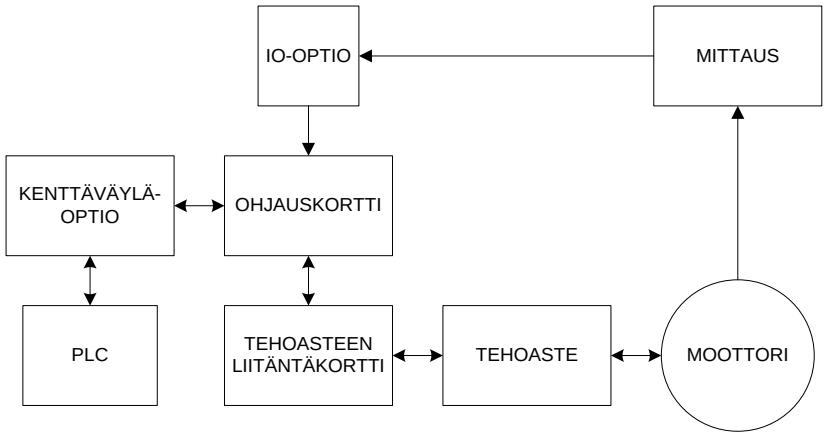
Taajuusmuuttaja voi sisältää useampia tasasuuntaaja- ja vaihtosuuntaaja moduuleja, jolloin on mahdollista saada kytkettyä suurempia virtoja moottorille tai ohjata useampia moottoreita. [4]

2.3 Taajuusmuuttajan mittaustekniikka ja vikojen havainnointi

Mittaustekniikkaa käytetään taajuusmuuttajakäytössä muun muassa vikatilanteiden havainnointiin sekä moottorin pyörimisnopeuden säätöön. Taajuusmuuttajan ohjausosaan kuuluu yleensä ohjauskortti sekä tähän liitetty ohjauspaneeli. Ohjauskorttiin voidaan lisätä liittimiä ja väyliä erilaisten optioiden avulla. Optioiden kautta sähkökäyttöön voidaan liittää

esimerkiksi toimintaa mittaavia antureita tai PLC:n (programmable logic controller). PLC on pieni tietokone, jota käytetään automatisoitujen prosessien ohjaukseen. Liitännät mittaustekniikkaa varten sijaitsevat tehoasteen liitäntäkortilla, ohjauskortilla sekä tälle liitetyillä IO-optioilla (input/output) (kuva 3).

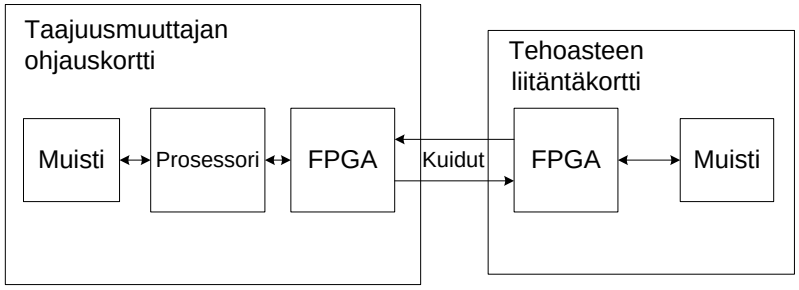
Moottorien toimintaa tarkkaillaan usein mekaanisilla antureilla, kuten kulma- ja kulmanopeusantureilla. Moottorin laakereiden tärinää voidaan mitata kiihtyvyysanturilla. Virranmittaukseen voidaan käyttää esimerkiksi shunt-vastusta tai Hall-elementtiä. Jännitettä voidaan puolestaan selvittää esimerkiksi mittaamalla resistanssiltaan suuren, rinnankytketyn vastuksen yli olevaa jännitettä. Lämpötilan tarkkailussa voidaan käyttää lämpökytkimiä, PTC-termistoria (positive temperature coefficient) tai vastuslämpömittaria (Pt-100). Anturityypistä riippumatta tarvitaan sähköinen eristys, ellei säätölaitteisto ole samassa potentiaalissa anturin kanssa. [3] Moottoria voidaan suojata ylikuumenemiselta mittaamalla moottorin lämpötilaa antureilla. Lämpötilan säädössä voidaan joutua korjaamaan vikapysäytyksen rajalämpötilaa, niin että ympäristön lämpötila otetaan huomioon [1].



Kuva 3: Kuvassa on sähkökäytön ohjauksen kannalta oleelliset moduulit

Tämän työn kannalta tärkein piirikortti on tehoasteen liitäntäkortti, jonka kautta tehoaste on liitetty taajuusmuuttajan ohjauskorttiin valokuiduilla (kuva 4). Valokuitujen ansiosta on suurijännitteisen tehoasteen ja pienijännitteisen ohjausosan välillä galvaaninen erotus. Kuidut ovat myös immuuneja sähkömagneettisten häiriöiden kytkeytymiselle. Mittaustietojen käsittely ja muu laskenta tapahtuu ohjauskortilla ja liitäntäkortilla. Ohjauskortin laskenta perustuu DSP-prosessoriin (digital signal processor), yleisprosessoriin ja FPGA-piiriin (field programmable gate array), kun taas liitäntäkortin toiminta on toteutettu täysin FPGA-pohjaisesti. Suuria määriä valmistettavissa taajuusmuuttajissa saatetaan FPGA-piirin tilalla käyttää myös ASIC-piiriä (application

specific integrated circuit). [4] Yhteen ohjauskorttiin voi olla mahdollista liittää useampia tehoasteita liitäntäkorttien kautta.



Kuva 4: Ohjauskortin ja liitäntäkortin välinen kommunikointi

Liitäntäkortin ominaisuuksina voivat olla esimerkiksi

- Tasajännitteen mittaus
- Lämpötilan mittaus jokaiselle tehomoduulille
- Vahinkokäynnistyksen esto
- Valokuituliitäntä ohjauskortille
- Kolmevaiheinen virranmittaus, jota käytetään myös maasulun osoitukseen
- Tulosilta ohjauskortin on/off -ohjaukseen
- Liitäntä jarrukatkojalle
- IGBT:n hilanohjauksen signaali tasasuuntaajalle

[5]

Sähkökäyttöjen mittauselektroniikkaa voidaan käyttää kolmenlaisten tilanteiden tarkkailuun:

1. Normaali toiminnan monitorointi
2. Poikkeavasta toiminnasta varoittaminen
3. Vikapysäytys

[6]

Normaalilla toiminnan monitoroinnilla tarkkaillaan, että taajuusmuuttajakäyttö toimii annettujen rajojen sisällä. Normaali toiminnan monitorointi on tärkeää myös säädön kannalta. Säädöllä tarkoitetaan sitä, että jos esimerkiksi moottorin havaitaan pyörivän haluttua nopeammin, hidastetaan pyörimisnopeutta automaattisesti vastamaan paremmin asetettua arvoa. Taajuusmuuttajan säädössä hyödynnetään ohjauskortin DSP-prosessoria sekä erilaisia säätöalgoritmeja. [3] Varoitusilmoitus voidaan antaa, mikäli havaitaan jotain normaalista poikkeavaa toimintaa, kuten vain hieman liian korkea lämpötila. Sähkökäytön toiminta saattaa kuitenkin jatkua normaalina varoitusilmoituksesta huolimatta. Mikäli kuitenkin havaitaan tarpeeksi kriittinen vikaindikaattori, liipaistaan vikapysäytys taajuusmuuttajakäytön suojelemiseksi. Jos vikaa ei voida havaita tietyn ajan kuluessa, voi taajuusmuuttajakäyttö rikkoutua, jonka vuoksi on tärkeää havaita kriittiset vikatilanteet

ajoissa. Mittaamisen ansiosta voidaan myös havaita syntymässä olevia vikatilanteita ennen kuin ne aiheuttavat suurempia vaurioita. [6] Vikapysäytys voidaan laukaista myös kenttäväylän kautta. Tyypillisiä taajuusmuuttajakäytön vikatilanteita on esimerkiksi ylivirta, maavuoto, oikosulku, IGBT:n ylikuormitus, välipiirin yli- tai alijännite tai IGBT:n ylilämpö. [1] Lisää taajuusmuuttajakäytössä ilmeneviä vikatilanteita on listattu liitteessä A.

2.4 Taajuusmuuttajien häiriöt

Taajuusmuuttajat ovat häiriömielessä haastava ympäristö suurten virta- ja jännitetasojen vuoksi. Erityisesti häiriöitä aiheuttavat tehokytkimien nopeat virtamuutokset. Taajuusmuuttajien ohjauselektroniikka on puolestaan pienten jännitetasojen vuoksi altis ottamaan häiriöitä ympäristöstä. [4]

Häiriöiden kytkeytymistavat voidaan jakaa johtumiseen, säteilemiseen sekä kapasitiiviseen ja induktiiviseen kytkeytymiseen. Johtumalla kytkeytyvät erityisesti matalilla taajuuksilla esiintyviä häiriöt, kun taas suuritaajuiset häiriöt kytkeytyvät usein säteilyn kautta tai induktiivisesti. Sähkömagneettisesta säteilystä aiheutuvat häiriöt kytkeytyvät usein sähköä johtavien kaapelien, kuten esimerkiksi mittauskaapeleiden, kautta. Induktiivinen kytkentä aiheutuu virtakaapelin muuttuvan magneettikentän indusoimista jännitteistä, kun taas kapasitiivinen kytkentä johtuu hajakapasitanssin kautta kulkevasta virrasta. Jos piiri on maadoitettu useasta kohdasta, voi häiriöitä kytkeytyä myös galvaanisesti maasilmukoiden kautta, jolloin häiriö kytkeytyy johtumalla. Myös tehonsyöttö voi aiheuttaa galvaanista kytkentää, ellei tällä ole omaa paluujohdinta kaapelissa. [3, 7]

Piirikortin ulkopuolelta tulevalta säteilyltä voidaan suojautua esimerkiksi metallikuoren avulla. Johtumiselta suojaudutaan erottamalla toiminnallisia osia galvaanisesti toisistaan. Induktiivista kytkeytymistä voidaan vähentää hidastamalla virtojen muutosnopeuksia ja pienentämällä silmukoita. Kapasitiivistä kytkeytymistä voidaan puolestaan vähentää piirin kokoa pienentämällä tai etäisyyksiä kasvattamalla, jolloin piirin kapasitanssi pienenee. Häiriöiden kytkeytyminen on yksi syy, joiden vuoksi taajuusmuuttajakäytön toiminnassa voi ilmetä vikatilanteita. EMC-ongelmat (electromagnetic compatibility) tulee ottaa huomioon jo taajuusmuuttajaa ja sen ohjauselektroniikkaa suunniteltaessa. Häiriöt aiheuttavat ongelmia etenkin suuria väylänopeuksia käyttäessä. [7]

3 Teknologia

3.1 FPGA-piiri

3.1.1 Ohjelmoitavat piirit

Taajuusmuuttajien ohjauksessa käytetään ASIC-, prosessori- ja FPGA-piirejä. Tässä työssä keskitytään erityisesti FPGA-piirin toimintaan. Mikroprosessorit ovat yleensä varsin yleiskäyttöisiä, mutta niiden suorituskyky ja hyötysuhde jäävät usein pieneksi. Räätälöidyt ASIC-piirit ovat energiatehokkaita, mutta ne sopivat hyvin vain siihen tarkoitukseen, mihin ne on valmistettu. FPGA-piirin voidaan ajatella olevan ominaisuuksiltaan näiden välissä. Taulukossa 1 on vertailtu ASIC:ia, FPGA:ta ja mikroprosessoria (µP).

Taulukko 1: ASIC-, FPGA-, prosessoripiirien vertailu

	Suoritus-kyky	**NRE**	**Yksikkö-kustannus**	**Suunnittelu-aika**
↑	ASIC	ASIC	FPGA	ASIC
	FPGA	FPGA	µP	FPGA
	µP	µP	ASIC	µP

NRE-kustannuksella (non-recurring engineering) tarkoitetaan tuotekehitysjaksosta, kuten esimerkiksi tutkimuksesta, suunnittelusta ja testauksesta, aiheuttamia kustannuksia. ASIC-piireillä on suuri NRE-kustannus, jonka vuoksi yhden ASIC:n hinta jää hyvin suureksi, ellei näitä valmisteta tarpeeksi suurta määrää. FPGA-piireillä NRE-kustannus on puolestaan pienempi, joten FPGA:n käyttäminen pienissä erissä valmistettavissa tuotteissa tulee yleensä halvemmaksi. FPGA-piirin korvaaminen ASIC-piirillä tulee yleensä kannattavaksi, kun tuotantomäärät ovat noin 5000 – 10000 vuodessa [3].

Räätälöityjen IC-piirien (integrated circuit), kuten ASIC-piirien, suunnittelu ja valmistaminen vievät aikaa vähintään muutaman viikon ja usein monia kuukausia, kun taas FPGA:n perustuvan ratkaisun suunnitteluaika on huomattavasti lyhyempi [8]. ASIC-piireillä saavutetaan kuitenkin parempi suorituskyky sekä pienempi koko ja tehonkulutus kuin FPGA:illa. Koska FPGA:t sopivat hyvin ASIC-piirien emulointiin, käytetään FPGA:ta usein ASIC:n tilalla prototyypissä. Suunnittelussa säästytään ylimääräiseltä työltä, mikäli myös valmiissa tuotteessa käytetään FPGA-piiriä. Vikojen korjaaminen on huomattavasti helpompaa FPGA-piiriä käyttäessä, koska muutokset voidaan tehdä pelkästään muokkaamalla FPGA:n logiikkaa PC-ohjelman (personal computer) avulla. Mikäli taas ASIC-piiriin halutaan tehdä muutoksia, täytyy ASIC suunnitella ja valmistaa uudestaan, mikä vie aikaa ja aiheuttaa paljon kustannuksia. [9]

Taajuusmuuttajien tehonkulutus on suuri verrattuna sen ohjauselektroniikan tehonkulutukseen. Tämän vuoksi ei ole kovinkaan merkittävää, kuinka paljon energiaa voitaisiin säästää käyttämällä ASIC-piirejä. Monia taajuusmuuttajia ei myöskään valmisteta esimerkiksi kulutuselektroniikkaan verrattuna kovinkaan suuria eriä, joten ohjauselektroniikan NRE-kustannus kannattaa yleensä pitää mahdollisimman pienenä.

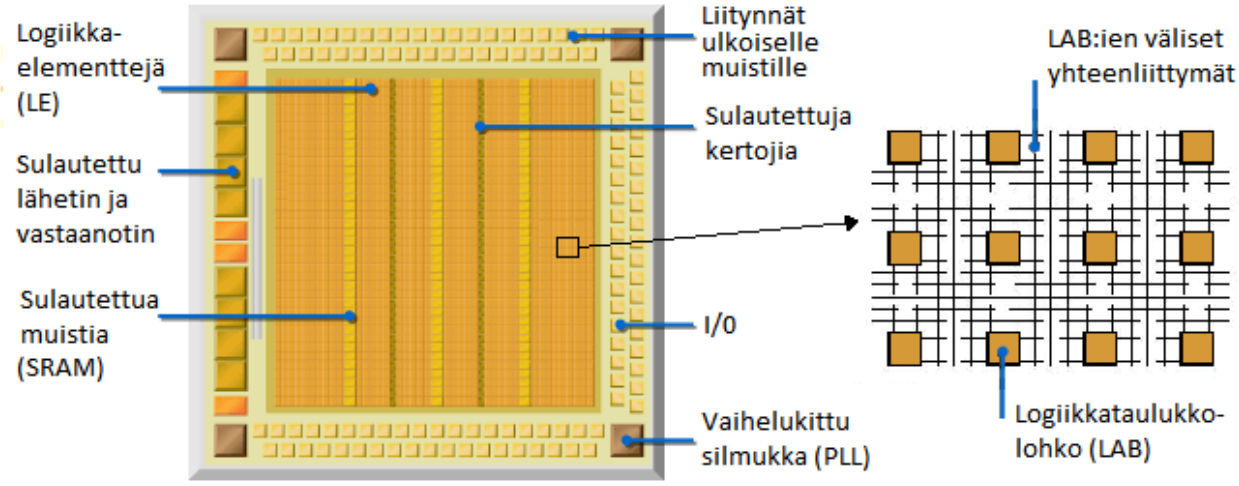
Mikroprosessoripiirien konfiguraatio ei ole juurikaan muokattavissa, mutta mikroprosessorin ohjelmoiminen on nopeaa. Kaikki prosessit tapahtuvat mikro-prosessorissa eri kellojaksoilla, kun taas FPGA tekee monia operaatioita rinnakkain. Tosin esimerkiksi superscalar-prosessori pystyy myös rinnakkaiseen laskentaan. Sulautetuissa järjestelmissä käytetään usein mikro-ohjaimia, joissa on mikroprosessorin lisäksi integroituina yleensä muita ominaisuuksia, kuten esimerkiksi A/D-muunnin (analog-to-digital) ja reaaliaikainen kello (RTC, real-time clock).

3.1.2 FPGA

FPGA on ohjelmoitava porttimatriisipiiri, joka koostuu toisiinsa kytketyistä logiikkaelementeistä (LE) [10]. Logiikkaelementeistä käytetään joissain yhteyksissä myös nimitystä CLB (configurable logic block) eli ohjelmoita logiikkalohko [11]. Logiikkaelementti on FPGA-piirin pienin looginen yksikkö ja se sisältää muun muassa SRAM-pohjaisia (static random access memory) hakutaulukoita (LUT, look-up table) sekä tietoja logiikkaelementtien yhteenliittymistä. Ohjausbittien avulla voidaan ilmaista hakutaulukon alkio, joka halutaan lukea. Logiikkaelementit muodostavat yhdessä suurempia toiminnallisia yksikköjä eli logiikkataulukkolohkoja (LAB, logic array block). Yhdessä logiikkataulukkolohkossa voi olla esimerkiksi 16 logiikkaelementtiä. [10] FPGA:n rakenne on esitetty kuvassa 5.

Logiikkaelementtien määrä kuvaa hyvin, kuinka monimutkaisia laitteistoja FPGA:lla voidaan toteuttaa. Esimerkiksi Alteran Cyclone IV -sarjan FPGA-piirissä on noin 10000 logiikkaelementtiä [12]. Kehittyneimmissä malleissa voi logiikkaelementtejä olla kuitenkin jopa kaksi miljoonaa, kuten Xilinx 7 -sarjan kalleimmissa malleissa [13]. Logiikka-elementtien lisäksi FPGA-piireille voi olla integroituna myös esimerkiksi kertojia, vaihelukittuja silmukoita (PLL, phase-locked loop) ja ylimääräistä SRAM-muistia.

Altera ja Xilinx ovat kaksi suurinta FPGA-piirien valmistajaa. Tässä työssä käytetään Alteran Cyclone IV -sarjan FPGA-piiriä. Cyclone on Alteran matalahintaisin FPGA-tuoteperhe. Hinta on Alteran muihin tuoteperheiden piireihin verrattuna alhaisempi muun muassa vähäisemmän sulautetun muistin ja logiikkaelementtien määrän vuoksi. Cyclone-perheeseen kuuluvien tuotteiden hinnat alkavat kymmenestä dollarista. Kalleimmat FPGA-piirit voivat maksaa kuitenkin jopa tuhansia dollareita. [14]



Kuva 5: FPGA:n toiminta perustuu logiikkaelementtien välisiin yhteenliittymiin, mutta FPGA-piirille voi olla integroituna myös esimerkiksi vaihelukittuja silmukoita.

3.1.3 FPGA:n ohjelmointi

FPGA:n toiminnallisuus voidaan kuvata joko kirjoittamalla ohjelma tai graafisesti lohkokaavioiden avulla. FPGA:n ohjelmointiin käytetään laitteistokuvauskieltä (HDL, hardware description language), joita ovat esimerkiksi VHDL (very high speed integrated circuit HDL) ja Verilog HDL. Eri valmistajilla on omat ohjelmistonsa FPGA:n ohjelmointia varten. Esimerkiksi Alteran FPGA:ita ohjelmoidaan *Quartus II* -ohjelmiston avulla. Laitteistokuvauskielet eroavat ohjelmointikielistä sillä, että koodia ei lueta järjestyksessä, vaan eri koodirivit voidaan lukea ja ajaa samanaikaisesti. Tämän vuoksi FPGA:n konfiguraation kirjoittaminen vaatii hyvin erilaista ajattelutapaa kuin prosessorin ohjelmointi, vaikka laitteistokuvauskielet ja ohjelmointikielet voivatkin näyttää päällepäin hyvin samanlaisilta. Siinä missä ohjelmointikielissä kirjoitetaan algoritmi, joka toteutetaan valmiilla laitteistolla, laitteistonkuvauskielessä kuvataan, minkälainen laitteisto tulee toteuttaa tietyn algoritmin aikaansaamiseksi.

Laitteistonkuvauskieli auttaa kuvaamaan digitaalisia systeemejä, mahdollistaa käyttäytymisen ja operaatioiden simuloinnin sekä systeemien fyysisen syntetisoinnin. Logiikkasynteesi on kriittinen prosessi, jota täytyy ohjata ja optimoida. Valmistajan ohjelma hoitaa yleensä ohjauksen ja optimoinnin automaattisesti. Kun kirjoittua koodia käännetään, täytyy ohjelman tarkastaa muun muassa, sopiiko logiikka käytössä olevaan FPGA:han. [10]

FPGA-piirien ohjelmointi perustuu logiikkaelementtien välisten yhteenliittymien luomiseen sekä hakutaulukoiden sisällön muuttamiseen. Tiedot hakutaulukoiden sisällöstä ja logiikkaelementtien yhteenliittymistä on yleensä tallennettuna FPGA-piirille sulautetulle muistille. [12] Joissain malleissa, kuten esimerkiksi useissa Actelin FPGA-piireissä,

käytetään OTP-muistia (one time programmable), jonka vuoksi konfiguraation voi ladata piirille vain kerran eikä sitä voi jälkeenpäin enää muuttaa. Useissa FPGA-piireissa konfiguraatio ladataan kuitenkin SRAM-muistille, joka sallii lukuisia uudelleen-kirjoituskertoja. SRAM-muistin ongelmana on kuitenkin, ettei sille tallennetut tiedot säily muistissa enää virrankatkaisun jälkeen. Tämän vuoksi SRAM-muistia käyttävien piirien konfiguraatio tallennetaan usein erilliselle flash-muistipiirille, josta konfiguraatio ladataan FPGA:lle aina uudelleenkäynnistyksen yhteydessä. SRAM-pohjaiset FPGA:t ovat yleistyneet laajasti helpon uudelleenohjelmoitavuutensa vuoksi. [3]

Laitteistokuvauskielellä kirjoitetut prosessit jaetaan kombinatorisiin ja sekventiaalisiin. Kombinatorinen logiikka voi toimia ilman kelloa, koska signaalien tilat eivät riipu signaalien aiemmista tiloista. Esimerkiksi summain ja kertoja voidaan lukea kombinatoriseksi logiikaksi, koska ne voidaan toteuttaa ilman kellosignaalia. Sekventiaalinen logiikka vaatii toimiakseen kellosignaalin, koska myös aiemmat arvot vaikuttavat nykyisiin arvoihin. Yksinkertaisin sekventiaalinen logiikkakomponentti on kiikku. Koska FPGA:n signaalien väliset viiveet eivät ole vakioita, on signaalit yleensä aina hyvä tahdistaa kellosignaalin avulla. [3][15]

Quartus II -ohjelmassa on joitain valmiita pohjia (template) tyypillisimpien ominaisuuksien, kuten siirtorekisteri tai muistinhallinta, kirjoitusta varten. Joitain toiminnallisuuksia, kuten esimerkiksi PLL, varten on myös olemassa megafunktioita. Megafunktiot ovat Alteran laitearkkitehtuurille sopivaksi optimoituja IP-lohkoja (intellectual property), joiden parametrit ovat muutettavissa. Megafunktiot on yleensä kirjoitettu AHDL:llä (Altera hardware description language) eli Alteran omalla laitteistonkuvauskielellä. Megafunktioiden määrittämistä varten voidaan käyttää *MegaWizard Plugin Manager* -työkalua. [16]

Konfiguraation lataamiseen FPGA:lle tuotekehitysympäristössä käytetään JTAG-porttia (Joint test action group), joka perustuu IEEE 1149.1 standardiin. JTAG-spesifioinnin kehitys alkoi vuonna 1985 ja siitä tuli standardi vuonna 1990. Standardin ansiosta on integroituja digitaalisia piirejä mahdollista testata tilanteissa, jossa piirit ovat sijoitettu tiheästi painetulle piirikortille. JTAG myös mahdollistaa mikropiirien sisäänrakennettujen testiominaisuuksien hyödyntämisen. Lähes kaikki porttimatriisipiirit sekä mikroprosessorit ja -ohjaimet tukevat JTAG-standardia. [17]

3.1.4 VHDL

VHDL kehitettiin 1980-luvun alussa Yhdysvaltojen puolustusvoimien VHSIC (very high speed integrated circuit) projektin aikana, kun syntyi tarve laitteistokuvauskielelle nopeiden integroitujen piirien kehitysprojektien kuvaamista varten. VHDL:stä tuli standardi (IEEE

1076) vuonna 1987, jonka jälkeen siihen on tullut vielä joitain parannuksia. Ensimmäinen päivitys tuli vuonna 1993. [15] [18]

VHDL on melko riippumaton teknologiasta, joten yhdelle laitteelle kirjoitettu VHDL-koodi voidaan yleensä siirtää toiselle laitteistolle ilman suuria muutoksia. Myös päivitysten tekeminen kehitettäviin systeemeihin onnistuu helposti. VHDL-pohjaisissa toteutuksissa on helppo yhdistää useita valmiita toiminnallisia IP-lohkoja toisiinsa. Valmiita IP-lohkoja on myös mahdollista jakaa ja käyttää uudelleen, jonka ansiosta jo kerran kirjoitettuja toiminnallisuuksia ei ole pakko kirjoittaa uudestaan. [4]

VHDL:n avulla voidaan toiminnallisuuksia kuvata eri abstraktiotasoilla, joihin kuuluu porttitaso, RTL-taso (register transfer level) ja käyttäytymistaso. Porttitaso (gate level) kuvaa toteutusta loogisten lohkojen tasolla ja sen alkeellisin komponentti on logiikkaportti. Porttitasolla kuvataan käytetyt portit ja näiden yhteenliittymät. RTL-tasolla kuvataan systeemi muistin yhteenliittyminä ja loogisina rakenteina. RTL-tason alkeellisimpia komponentteja ovat muun muassa rekisterit, summaimet ja kertojat. Käyttäytymistasolla kuvattua koodia käytetään pääasiassa simuloinnissa, eikä se ole suurimmaksi osaksi syntetisoituvaa. VHDL:llä kirjoitetun ohjelman verifiointi on mahdollista korkeallakin abstraktiotasolla, jonka vuoksi eri toteutusten suoriutumista voidaan vertailla helposti. [19]

Kuten ohjelmointikieliä yleensäkin, myös VHDL:ää voi kommentoida, joten ohjelmatiedosto toimii myös laitteiston toimintaa kuvaavana dokumenttina. VHDL:n ansiosta voidaan systeemin toimintaa kuvata selkeällä ja yksiselitteisellä tavalla. Ohjelmoijan on kuitenkin oltava huolellinen kommentoidessaan ja valitessaan muuttujien nimiä, jotta koodi olisi ymmärrettävää myös muille ohjelmoijille. [20]

Vaikka kaikki VHDL:llä kirjoitetun koodin tiettyyn kellosignaaliin reagoivat tapahtumat toteutuvat samanaikaisesti, voidaan loogisten käskyjen kirjoitusjärjestyksellä kuitenkin vaikuttaa kiikkujen sijoittumiseen lopullisessa toteutuksessa, jonka vuoksi kirjoittaessa on oltava huolellinen, jotta toiminta olisi halutunlaista. [3] Joskus VHDL:llä kuvattu laitteisto on mahdotonta toteuttaa fyysisesti käytössä olevalla teknologialla [21]. Näin voi käydä esimerkiksi, jos FPGA-piirillä ei ole tarpeeksi logiikkaelementtejä tai muistia laitteiston toteuttamiseksi. Näissä tilanteissa on mahdollista esimerkiksi optimoida koodia, karsia ominaisuuksia tai käyttää FPGA-piiriä, jolla on riittävästi logiikkaelementtejä.

VHDL:llä kirjoitettuja toiminnallisuuksia on mahdollista simuloida esimerkiksi *Modelsim-Altera* -ohjelman avulla. Simuloinnin avulla voidaan analysoida laitteistoon toimintaa ennen konfiguraation siirtämistä itse laitteistolle. Simulointia varten tulee testattavalle ohjelmalle luoda testipenkki, jossa määritetään herätteet. Herätteet voidaan esittää myös tuntemattomina. [12]

VHDL-koodi koostuu toiminnallisista yksiköistä. Näistä tärkeimmät ovat entity- ja architecture-yksiköt. Entity-yksikkö kuvaa vain yhteydet ulkoiseen maailmaan eikä kerro operaatioista tai toteutuksista. Architecture-yksikkö kuvaa puolestaan komponenttien sisäiset rakenteet ja toiminnallisuuden, kuten abstrakti kuvaus, logiikka, aritmetiikka sekä moduulien väliset yhteydet.

Sama komponentti voidaan kuvata eri abstraktiotasoilla. Yhdellä komponentilla voi olla myös useita toteutuksia. Esimerkiksi summain voidaan toteuttaa usealla eri tavalla, vaikka toiminnallisuus olisikin sama. Yhdellä entity-yksiköllä voi olla useita architecture-yksiköitä, joiden sisällä voi olla useita prosesseja, joissa varsinainen toiminnallisuus kuvataan. Eri prosesseissa voi olla kuvattuina useita rinnakkaisia tapahtumia. Prosessit kommunikoivat toistensa kanssa signaalien avulla, kun taas prosessien sisällä voidaan kommunikoida sekä signaaleilla että muuttujilla. Muuttujat käyttäytyvät prosessien sisällä sekventiaalisesti, mutta signaalit muuttuvat eri prosesseissa samanaikaisesti. Muita yksiköitä ovat package- ja configuration-yksiköt, joista package-yksikkö sisältää funktioita ja parametreja yleiseen käyttöön, kun taas configuration-yksikkö mahdollistaa eri toteutusten hallinnoimisen. [15]

3.1.5 Ohjelmapohjainen prosessori

Ohjelmapohjaisella eli ”pehmeällä” prosessorilla (soft processor) tarkoitetaan prosessoria, joka on toteutettu ohjelmoitavalla logiikkapiirillä, kuten FPGA:lla, ja tälle integroidulla muistilla. Eri valmistajilla on omat ohjelmapohjaiset prosessorinsa. Esimerkiksi Alteran pehmeä prosessori *NIOS II* toimii kuten 32-bittinen, RISC-tyyppinen (reduced instruction set computer) yleisprosessori. Pehmeät prosessorit mahdollistavat FPGA:n ohjelmoimisen myös C ja C++ -kielillä. Tällöin koodi ajetaan siinä järjestyksessä kuin se on kirjoitettu, kuten myös mikroprosessoreilla yleensäkin. Pehmeää prosessoria käyttäessä voi kuitenkin hyödyntää räätälöityjä käskyjä (custom instruction), jotka sallivat normaalisti useammalla käskyllä toteutettavan toiminnallisuuden suorittamisen yhdellä käskyllä, mikä nopeuttaa aikakriittisten operaatioiden suoritusta [22].

NIOS II -prosessorin luomiseen Alteran FPGA-piireille voidaan käyttää apuna *Quartus II* -ohjelman *SOPC Builder* -työkalua. *SOPC-builderin* avulla voidaan myös määrittää prosessoriin liitettävät oheislaitteet ja -väylät, kuten muistipiirit ja SPI-väylä. [23] Koska tämän työn toteutuksen vaatimat toiminnot ovat hyvin alhaisen tason operaatioita, kuten tiedon tallennusta muistiin, ja koska eri toimintojen halutaan tapahtuvan juuri tietyllä hetkellä, ei sulautettua prosessoria ole syytä käyttää tämän työn toteutuksessa.

3.2 Tiedonsiirto

3.2.1 Väylät

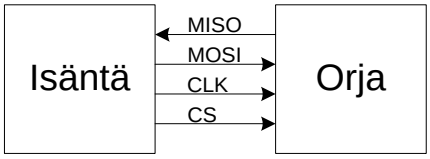
Väyliä pitkin voidaan siirtää tietoa joko piirikortilla olevien mikropiirien välillä tai tietokoneen ja oheislaitteiden välillä. Digitaaliset tiedonsiirtoväylät voidaan jakaa rinnakkais- ja sarjamuotoisiin sekä synkronisiin ja asynkronisiin väyliin. Väylä on synkroninen, mikäli isännällä ja orjalla on yhteinen tiedonsiirtoa tahdistava kellosignaali. Asynkroninen väylä ei puolestaan ole tahdistettu, joten se voi tarvita kättelyprotokollan, jolla isännälle kerrotaan, että siirretty tieto on tullut perille. Rinnakkaisväylässä tiedot kulkevat useammassa johtimessa samanaikaisesti. Väylän nopeutta voidaan kuvata suoritusteholla, joka saadaan, kun kerrotaan väylän kellotaajuus väylän leveydellä. Rinnakkaisväylä voi olla esimerkiksi yhden tavun levyinen eli 8-bittinen. Rinnakkais-väylien ongelmana on, että ne vievät paljon tilaa piirilevyltä, vaativat leveitä liittimiä ja kuluttavat paljon tehoa. Rinnakkaisväyliä ei ole juurikaan standardoitu.

Sarjaväylässä hyötytieto siirretään vain yhdessä johtimessa, mutta sarjaväyliä voidaan ajaa kuitenkin nopeammalla kellosignaalilla kuin rinnakkaisväyliä, joten sarjaväylillä voidaan saavuttaa suurempi tiedonsiirtonopeus. Sarjaväylien standardeja ovat esimerkiksi USB (universal serial bus) ja PCI Express (peripheral component interconnect). Piirikorttiväyliä ovat esimerkiksi SPI (serial peripheral interface) ja I^2C (inter-integrated circuit).

SPI-väylä

SPI on sarjamuotoinen, synkroninen oheislaiteväylä (peripheral bus), jonka avulla suorittimeen tai FPGA:han voidaan liittää oheislaitteita keskinopeudella ja halvalla. SPI-väylää voidaan käyttää esimerkiksi FPGA:n ja konfigurointimuistin välillä tiedon kirjoittamista ja lukemista varten sekä konfiguraation lataamiseen FPGA:n SRAM-muistiin uudelleenkäynnistyksen jälkeen. Vaikka SPI on varsin laajassa käytössä, se ei ole kuitenkaan standardi, joten sen ominaisuudet vaihtelevat eri valmistajien välillä. Tämän vuoksi SPI-väylän ominaisuudet on tarkastettava aina valmistajan esitteestä.

SPI-väylä mahdollistaa kaksisuuntaisen kommunikoinnin isännän ja orjan välillä. Orjia voi olla myös useampi kuin yksi. Isäntänä toimii yleensä FPGA- tai prosessoripiiri ja orjana esimerkiksi muistipiiri tai A/D-muunnin. SPI-väylän signaalit on esitetty kuvassa 6 ja näiden selitykset on taulukossa 2. Tieto kulkee isännältä orjalle MOSI-johtimen (master out slave in) kautta ja orjalta isännälle MISO-johdinta (master in slave out) pitkin. Isännältä orjalle kulkee myös kellosignaali (CLK) ja CS-signaali (chip select). CS-signaalin avulla voidaan valita orja, jonka kanssa isäntä milloinkin kommunikoi. Orja on valittu silloin, kun tälle menevä CS-signaali on alhaalla. SPI-väylän signaaleja voidaan valmistajasta riippuen kutsua hieman eri nimillä.



Kuva 6: Neljäjohtimisen SPI-väylän toiminta perustuu isäntä-orjarakenteeseen.

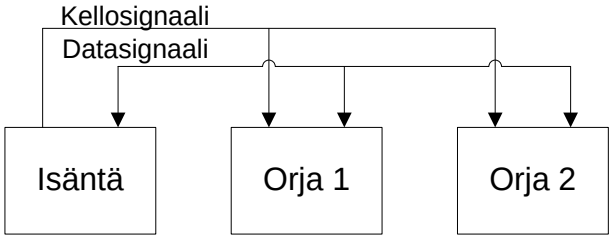
Alun perin SPI-väylän esitteli Motorola, mutta sitä käytetään myös monien muiden valmistajien tuotteissa. National Semiconductorin vastaava väylä tunnetaan nimellä Microwire. SPI-väylän kellotaajuus on tyypillisesti 40 MHz:n luokkaa, mutta voi myös vaihdella paljon käyttötarkoituksesta riippuen.
[24]

Taulukko 2: SPI-väylän signaalit

MOSI	Master Out Slave In	Tiedonsiirto isännältä orjalle
MISO	Master In Slave Out	Tiedonsiirto isännälle orjalta
CLK	Clock	Kellosignaali orjalle
CS	Chip Select	Valitaan eri orjien välillä, aktiivinen alhaalla

I^2C-väylä

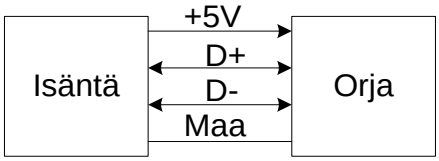
I^2C (kuva 7) Phillipsin 80-luvullla kehittämä sarjamuotoinen väylä. I^2C-väylässä on vain kaksi johdinta, joista toista käytetään kaksisuuntaiseen tiedonsiirtoon ja toinen on kellosignaali. Samaan väylään on mahdollista liittää useampia oheislaitteita, jotka ovat osoitettavissa yksilöitävällä osoitteella. I^2C-väylän nopeus on yleensä välillä 100 kb/s – 3,4 Mb/s. Myös I^2C perustuu isäntä-orja-periaatteeseen.



Kuva 7: I^2C-väylässä on vain kaksi johdinta, joista toista käytetään tiedonsiirtoon ja toinen on kellosignaali.
[25]

3.2.2 USB-väylä

USB-väylää käytetään sarjamuotoiseen tiedonsiirtoon PC:n ja oheislaitteiden välillä. Yhteen USB-väylään voidaan liittää jakajien avulla yhteensä enintään 127 laitetta. USB-väylässä on neljä johdinta, joista toinen pari toimii 5 voltin käyttöjännitteenä ja tämän paluujohtimena, ja toinen pari eromuotoisena siirtokaistana (kuva 8). Eromuotoisuudella tarkoitetaan sitä, että vastaanotin vahvistaa sisään tulevien signaalien välisen jännite-eron. Koska eromuotoisuuden ansiosta saman merkkiset häiriöt kumoutuvat, lähettää USB-väylä vähemmän häiriöitä ympäristöön ja on immuunimpi ulkoa tuleville häiriöille. USB käyttää NRZI-koodausta (non-return-to-zero inverted), jonka ansiosta signaalin tulkittavuuden kannalta liian pitkiä jaksoja, jolloin signaalin tila ei muutu, ei pääse syntymään. [3]



Kuva 8: USB-väylässä tiedot välitetään eromuotoisena D+ ja D- -johtimissa isännän ja orjan välillä. Isännältä on usein myös 5 voltin tehonsyöttö orjalle.

USB-laitteille on olemassa eri luokkia, jotka määrittävät tiedonsiirron yhteysprotokollan. Esimerkiksi tietokoneen näppäimistö välittää lyhyitä pulssimaisia viestejä silloin, kun näppäintä painetaan, kun taas massamuistille täytyy pystyä siirtämään tietoa suurempina paketteina. Tämän hetken yleisin USB-standardi on 2.0, jolla pystytään siirtämään tietoa teoreettisesti 480 Mb/s nopeudella. Todellisuudessa USB 2.0 -väylän hyötytiedon siirron maksiminopeus on noin 300 Mb/s USB:n protokollan vuoksi. [26]

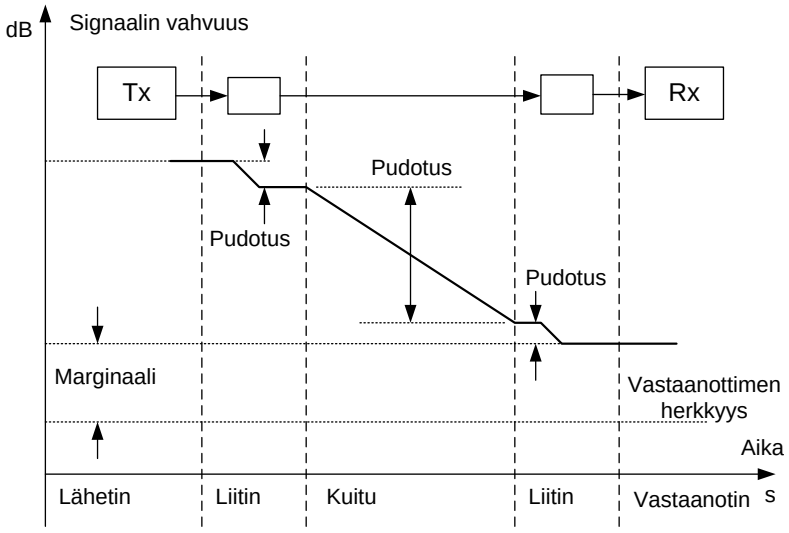
Monissa uusissa laitteissa on jo USB 3.0 -väylää, joka on yli kymmenen kertaa USB 2.0 -väylää nopeampi. [26] Muita nopeita oheislaiteväyliä ovat muun muassa IEEE 1394 -standardin määrittämä FireWire 800 -väylä (800 Mb/s), SATA 3.0 (serial AT attachment, 6 Gb/s) ja Thunderbolt-väylä (10 Gb/s). Todellinen tiedonsiirtonopeus on tosin hieman annettuja nopeuksia pienempi johtuen protokollan aiheuttamista rajoituksista.

3.2.3 Valokuidut

Optisia kuituja käytetään paljon teollisuudessa niiden monien hyötyjen vuoksi. Kuitulähettimien ja -vastaanottimien alhaisen hinnan ansiosta kuitulinkin käyttö on suhteellisen halpaa, mikäli kuituja ei ole paljoa, eikä kovin pitkiä kuituja tarvita. Kuitujen käyttäminen on helppoa ja ne eristävät laitteita toisistaan korkean resistanssinsa vuoksi. Kuitukaapelit ovat myös kevyitä ja pienikokoisia halkaisijaltaan sekä kestävät hyvin korroosiota. Kuidut eivät kipinöi eivätkä aiheuta sähkömagneettisia häiriöitä. Kuitulähettimet ja vastaanottimet ovat kuitenkin elektronisia laitteita, jonka vuoksi nämä

voivat aiheuttaa tai vastaanottaa sähkömagneettisia häiriöitä. Kuiduilla on korkea tiedonsiirtokapasiteetti. Teknologian kehittyessä kuitujen kapasiteetti voi kasvaa teoriassa lähes rajattomasti.

Kuvassa 9 on esitetty signaalin eteneminen lähettimeltä vastaanottimelle. Kuitulähetin perustuu yleensä diodilaseriin tai lediin (light emitting diode) ja vastaanottimena on usein valodiodi. Laser-lähettimillä päästään suurempiin tiedonsiirtonopeuksiin kuin LED-lähettimillä. Kuitulinkki voi sisältää vain yhden kuidun, jossa kulkee yksi signaali, mutta on myös olemassa rinnakkaismuotoisia valokuitukaapeleita. Yhdessä kuidussa voidaan välittää myös useampia, aallonpituudeltaan poikkeavia signaaleja.



Kuva 9: Kuvassa on esitetty tiedonsiirto kuituja pitkin lähettimeltä (Tx) vastaanottimelle (Rx). Signaali menettää osan tehostaan liittimissä sekä kuidussa. Tämän työn mahdollisissa ratkaisuissa kuitujen pituus on todennäköisesti niin lyhyt, ettei signaalien vaimenemisesta kuiduissa tarvitse välittää.
[27]

Tämän hetken taajuusmuuttajissa käytetyillä kuiduilla voidaan siirtää tietoa noin 50 Mb/s nopeudella. Standardin IEEE 802.3-2008 [28] määrittämät Gigabit Ethernet -tyypin kuitujen nopeus voi olla kuitenkin jopa yli 1 Gb/s. On myös olemassa 10 ja 100 Gigabit Ethernet -väylät, joiden nopeus on tätäkin suurempi.

3.2.4 Langaton tiedonsiirto

Langaton tiedonsiirto perustuu radiotaajuisten signaalien siirtämiseen lähettimeltä vastaanottimelle. Langaton signaali koostuu kantoaallosta ja tähän moduloidusta informaatiosta. Langattomilla lähiverkoilla on useita standardeja kuten WLAN (wireless

local area network, IEEE 802.11), Bluetooth (IEEE 802.15.1), UWB (ultra-wideband, IEEE 802.15.3) ja ZigBee (IEEE 802.15.4) [29]. Eri standardeilla on monia eroavaisuuksia, kuten esimerkiksi modulaatio, protokollan monimutkaisuus ja yhteen verkkoon liitettävien laitteiden määrä. Taulukossa 3 on vertailtu standardien kaistanleveyksiä ja kantomatkoja.

Taulukko 3: Langattomien lähiverkkojen vertailu.

	Bluetooth	UWB	ZigBee	WLAN
IEEE-standardi	802.15.1	802.15.3a	802.15.4	802.11a/b/g
Taajuuskaista	2,4 GHz	3,1 - 10,6 GHz	868/915 MHz 2,4 GHz	2,4 GHz 5 GHz
Kaistanleveys	3 Mb/s	110 Mb/s	250 kb/s	54 Mb/s
Kantomatka	10 m	10 m	10 - 100 m	100 m

Bluetooth- ja UWB-teknologioiden kantomatka on 10 metriä, kun taas ZigBee- ja WLAN-teknologioilla voidaan siirtää tietoa 100 metrin päähän. Bluetoothista on myös olemassa eritehoisia luokkia, joiden kantomatkat vaihtelevat. Bluetooth 2.0 -teknologialla päästään 3 Mb/s kaistanleveyteen, mutta hyötytietoa voidaan siirtää maksimissaan noin 2 Mb/s nopeudella.

WLAN-teknologialla (IEEE 802.11 a/g) voidaan päästä 54 Mb/s kaistanleveyteen. Pakettien koko vaikuttaa kuitenkin todelliseen siirtonopeuteen. Esimerkiksi siirrettäessä 60 tavun mittaisia paketteja olisi siirtonopeus vain 2,6 Mb/s, kun taas 1500 tavun paketteja siirrettäessä päästäisiin 30 Mb/s nopeuteen. [40]

Yksi UWB-teknologian tarkoitus on korvata USB 2.0 väylä langattomalla vaihtoehdolla, jonka vuoksi sen on oltava varsin nopea. UWB-teknologialla päästään yli 110 Mb/s nopeuteen. [29] Toinen USB-väylän langaton vastine on WUSB (Wireless USB). WUSB:llä voidaan saavuttaa 110 Mb/s nopeus 10 metrin matkalla ja 480 Mb/s nopeus 3 metrin matkalla. WUSB pystyy hyödyntämään samaa laitteistoarkkitehtuuria kuin langallinen USB. Tästä on paljon hyötyä, koska monilla laitteilla on USB-tuki jo ennestään. [26] Langatonta USB-teknologiaa on vielä varsin vähän markkinoilla.

Langattomat teknologiat eivät ole vielä saavuttaneet suurta suosiota teollisuudessa. Yksi syy tähän on näiden toimivuuden epävarmuus. [30] Toisaalta langallisilla teknologioilla on enemmän taajuuskaistaa. Radiotaajuuksien käyttö on tarkasti säänneltyä ympäri maailmaa, joka voi osaltaan vaikeuttaa monien teknologioiden hyödyntämistä. Langatonta tiedonsiirtoa varten olisi myös mahdollista luoda oma protokolla, mutta ne toimivat huonosti yhteen muiden valmistajien tuotteiden kanssa ja niiden tekeminen vaatisi paljon työtä. [31]

3.2.5 Yhteyskäytännöt

Kahden laitteen välistä kommunikointia varten tarvitaan yhteyskäytäntö eli protokolla, jotta laitteet voisivat kommunikoida keskenään. Protokolla määrittää esimerkiksi viestin kehyksen eli rakenteen ja käytetyn virheenpaljastusmenetelmän. Jotta laitteiden välinen kommunikointi onnistuisi, täytyy laitteelta toiselle lähetetyn viestin kehyksen ja virheentarkistusmenetelmän olla kummallakin laitteella sama. Jos väylässä on useampia laitteita, on lähettäjän tiedettävä myös vastaanottajan osoite. [3]

3.2.6 Virheenpaljastus ja koodaus

Kun häiriöt ylittävät tietyn tason, voi digitaalisen viestin sisältö muuttua niin paljon, ettei viesti ole enää tulkittavissa oikein. Erilaiset virheenpaljastusmenetelmät vähentävät siirtokaistaa hyötytiedolta, mutta näiden käyttö on yleensä kuitenkin tarpeellista digitaalista signaalia siirtäessä, jotta vioittuneet viestit voitaisiin havaita. Mikäli virhe huomataan, voidaan lähettäjää pyytää lähettämään viesti uudelleen.

Virheentarkastuksessa voidaan käyttää esimerkiksi pariteettibittiä. Tällöin esimerkiksi 8 bitin hyötysignaaliin lisätään yhdeksäs bitti, joka on 0 tai 1 sen perusteella, halutaanko ykkösiä olevan parillinen tai pariton määrä. Virhe huomataan, jos yhdeksässä bitissä pitäisi olla parillinen määrä ykkösiä, mutta näitä onkin pariton määrä. Pariteettitarkistuksen lisäksi, voidaan virheenpaljastukseen käyttää myös syklisiä virheenpaljastusmenetelmiä, joista yleisin on CRC (cyclical redundancy check). Siirtorekistereihin perustuvat sykliset virheenpaljastusmenetelmät ovat toteutukseltaan yksinkertaisia ja paljastavat virheitä tehokkaasti. Virheitä on mahdollista paljastaa myös toiston, kaiutuksen ja järkevyys-tarkastuksen avulla. Toistossa sama viesti lähetetään kahteen tai useampaan kertaan ja kaiutuksessa vastaanotettu viesti lähetetään takaisin lähettäjälle. Järkevyystarkastuksessa voidaan katsoa, ovatko siirretyn signaalin arvot järkevällä alueella.

Koodauksen tarkoitus on yleensä estää liian monen peräkkäisen saman bitin esiintyminen tiedonsiirrossa, koska tämä voi vaikeuttaa viestin tulkitsemista oikein. 4B/5B-koodauksessa viesti lähetetään 4 bittiä kerrallaan 5 bitin ryhmissä, jotka on muodostettu tietyn säännön mukaan. Esimerkiksi viides bitti voi olla neljännen bitin komplementti, jolloin yli viiden saman peräkkäisen bitin toistuminen viestissä estyy. Muita samalla periaatteella toimivia koodauksia ovat 5B/6B- ja 8B/10B-koodaukset. Nämä koodausmenetelmät sopivat myös virheenpaljastukseen. Näiden lisäksi on olemassa monia muita koodaustapoja, kuten NRZ-(non-return-to-zero), NRZI-, ja Manchester-koodaukset. [3]

3.3 Muistit

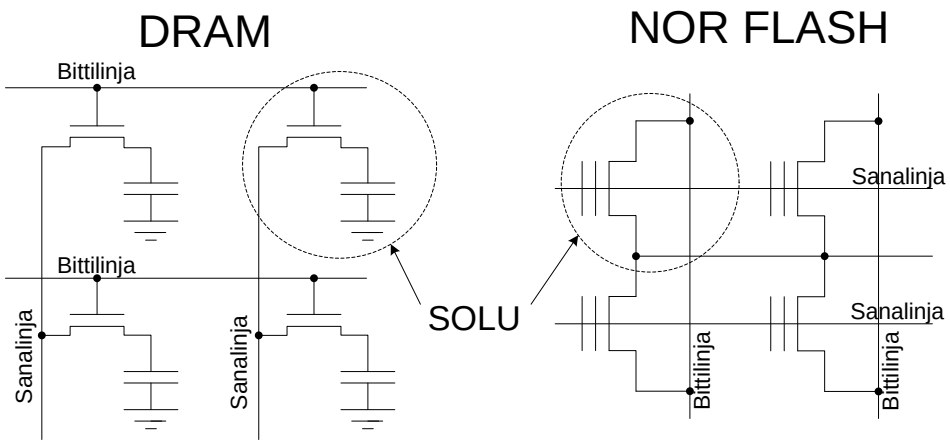
3.3.1 Haihtuvat ja haihtumattomat muistityypit

Muistit voidaan jakaa haihtuviin (volatile) ja haihtumattomiin muisteihin (non-volatile). Kun haihtuvalta RAM-muistilta (random access memory) katkaistaan virrat, muistin sisältö tyhjenee, mutta haihtumattomassa muistissa tiedot säilyvät myös virtojen ollessa poissa. Haihtuvia muisteja ovat esimerkiksi DRAM- (dynamic RAM) ja SRAM-muistit ja haihtumattomia ROM (read only memory)- ja flash-muistit. Haihtuvat muistit ovat usein huomattavasti nopeampia kuin haihtumattomat muistit ja haihtuville muisteille on yleensä mahdollista kirjoittaa lukemattomia kertoja. Haihtumattomat muistit kestävät uudelleen-kirjoitusta rajoitetusti. Siinä missä flash-muistille voi samaan muistiosoitteeseen kirjoittaa tavallisesti noin 100 000 kertaa, sallivat RAM-muistit vähintään 100 000 000 kirjoitus-kertaa. Useimmat haihtumattomat muistit, kuten flash, täytyy tyhjentää ennen uudelleenkirjoitusta, mutta haihtuvissa muisteissa on mahdollista kirjoittaa vanhan tiedon päälle tyhjentämättä piiriä ensin.

ROM-muistit ohjelmoidaan jo valmistaessa, eikä niitä voi enää sen jälkeen tyhjentää, jonka vuoksi ne sopivat lähinnä sarjatuotantona tehdyillä piirikorteilla käytettäviksi. PROM-muisteja (programmable ROM) voi ohjelmoida myös käyttäjä, mutta myöskään PROM-muistia ei ole mahdollista uudelleenohjelmoida. EPROM-muistit (erasable PROM) voidaan tyhjentää altistamalla ne UV-säteilylle kotelossa olevan ikkunan kautta, jolloin muisti tyhjenee kokonaan ja jonka jälkeen niille voi taas kirjoittaa uudestaan. EEPROM-muistit (electrically erasable PROM) on puolestaan mahdollista tyhjentää sähköisesti tavu kerrallaan, joka tekee niistä EPROM-muisteja käytännöllisempiä. [32] EEPROM-muistin käyttämä ”kelluva hila” -toimintaperiaate (floating gate) on määritetty IEEE 1005 -standardissa [33].

3.3.2 Muistin ohjelmointi

Useissa kiinteissä muisteissa (solid state memory), kuten RAM-muistit ja flash-muisti, muistin ohjelmointi perustuu transistorin tilan muuttamiseen joko muuttamalla itse transistorin tai transistoriin liitetyn muistielementin ominaisuuksia. Muistielementti voi olla esimerkiksi resistiivinen tai kapasitiivinen. Tässä työssä tarkoitetaan muistisolulla sitä kokonaista yksikköä, joka määrittä yhden bitin tilan ja muistielementillä tarkoitetaan muistisolun osaa, jonka kynnysjännitteen ja johtamiskyvyn muuttamiseen bittien tallennus perustuu. Esimerkiksi kuvan 10 DRAM-muistin kondensaattori toimii muistielementtinä ja se muodostaan yhden muistisolun transistorin kanssa. NOR-flash-muistisolu sisältää puolestaan vain transistorin.



Kuva 10: Kuvassa on esiteltyä 2x2-kokoiset DRAM- ja NOR-flash-muistit. DRAM-muisti ohjelmoidaan transistoriin liittyneen kondensaattorin ominaisuuksia muuttamalla. Flash-muistien ohjelmoinnissa puolestaan muutetaan FET-transistorin (field-effect transistor) johtavuutta.
[32]

3.3.3 RAM-muistit

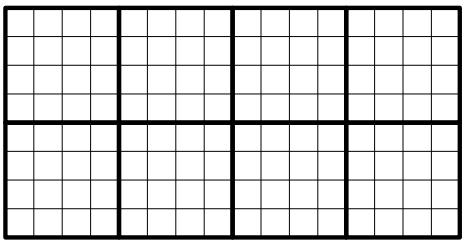
RAM-puskurien käyttö on usein ratkaisu pysyvien muistien hitaaseen kirjoitusnopeuteen. RAM-muistin nimi tulee siitä, että sitä voidaan lukea suoraan halutusta muistiosoitteesta. DRAM-muistissa bitit tallennetaan transistoriin liittyneiden kondensaattorin varauksina. Koska kondensaattorit purkautuvat itsestään, täytyy suorittimen virkistää muistisoluja tietyin väliajoin. Tämä hidastaa DRAM-muistin toimintaa ja lisää tämän energiankulutusta. SRAM-muisti on puolestaan nopeampaa kuin DRAM-muisti, eikä se vaadi jatkuvaa virkistämistä. Yksi SRAM-muistisolu sisältää kuitenkin 4 transistoria, jonka vuoksi SRAM on muistikapasiteettiinsa nähden suhteellisen kallista ja tilaa vievää. On myös olemassa ulkoisesti SRAM-muistin tavoin toimivaa PSRAM-muistia (pseudo static RAM), joka perustuu DRAM-teknologiaan, mutta pystyy itse virkistämään muistisolunsa [11]. SRAM-muistia käytetään nopeutensa vuoksi esimerkiksi prosessorin välimuistina. DRAM-muistia käytetään puolestaan tietokoneen työmuistina.

Koska RAM-muisti ei ole tallennuksen pullonkaula, keskitytään tässä työssä enemmän haihtumattomien muistien vertailuun, joiden nopeuksien ja tallennuskapasiteettien välillä on suuria eroja. Markkinoille on kokoajan tulossa lisää haihtumattomia RAM-muisteja, joilla haihtuvien muistien kirjoituskestävyys ja kirjoitusnopeus, mutta joissa tiedot säilyvät vielä virrankatkaisunkin jälkeen.

3.3.4 Flash-muisti

Haihtumaton flash-muisti perustuu samaan ”kelluva hila” -periaatteeseen kuin EEPROM-muisti, mutta flashin muistialueet ovat huomattavasti suurempia. EEPROM-muistia täytyy tyhjentää tavu kerrallaan, joka tekee EEPROM-muistin tyhjennyksestä hidasta. Koska koko flash-muisti voidaan tyhjentää kerrallaan, voivat tyhjennysajat olla hyvin nopeita. Flash-muistiavaruus on jaettu sektoreihin, jotka on puolestaan jaettu lohkoihin tai sivuihin (kuva 11). Tässä työssä puhutaan selvyyden vuoksi sivuista (page), vaikka jotkin valmistajat käyttävätkin tästä termiä lohko (block). Yhden sektorin koko voi olla esimerkiksi 512 kB, kun taas sivun koko on tavallisesti 256 tavua. Sektori on pienin tyhjennettävissä oleva alue, ja sivu on pienin alue, joka voidaan kirjoittaa kerrallaan. Flash-muistille kirjoittaessa tiedot siirretään sivun kokoisina paketteina. Kun edellinen paketti on tallentunut muistiin, voidaan seuraavan paketin tallennus uuteen osoitteeseen aloittaa.

Flash-muisti sopii nopeutensa vuoksi EEPROM-muistia paremmin suurten tietokappaleiden tallennukseen. Flash-muistia voidaan hyödyntää sulautetuissa järjestelmissä esimerkiksi prosessorin tai FPGA:n konfigurointimuistina. Pienen kokonsa, alhaisen energiantarpeen ja liikkuvien osien puuttumisesta johtuvan kestävyyden vuoksi flash-muistit soveltuvat hyvin myös kannettaviin laitteisiin, kuten USB-tikkuihin, SSD-kiintolevyihin (solid state drive) ja muistikortteihin. [32] Mikäli tiettyä muistialuetta käytetään muistinhallintaan, voi flash-muistin uudelleenkirjoituksen kanssa tulla ongelmia, jos muistinhallintalohkolle tallennetaan niin monta kertaa, ettei piiri kestä enää uudelleenkirjoitusta.[34]



*Kuva 11: Esimerkki flash-muistipiirin järjestäytyminen. Kuvan piirissä on 8 sektoria ja yhdessä sektorissa on 16 sivua. Jos yhden lohkon koko olisi 256 tavua, on koko muistin koko 256 B * 8 * 16 = 32 kB. Todellisuudessa flash-muistit ovat yleensä tätä suurempia ja yhdessä sektorissa on huomattavasti enemmän kuin 16 sivua.*

Flash-muistia ohjelmoidaan muuttamalla transistorien kynnysjännitteitä. Kun muistisolun tila on 1, transistorin kelluvalla hilalla ei ole varausta. Kun muistisolu ohjelmoidaan tilaan 0, syntyy transistorin kelluvalla hilalla varaus, jolloin transistorin kynnysjännite muuttuu ja virran kulkeminen transistorin läpi estyy. Tyhjennyksen jälkeen kaikki alueen muistiolut ovat tilassa 1. Muistialueen täytyy olla tyhjä ennen kuin sille voidaan kirjoittaa.

On olemassa kahta erilaista flash-tyyppiä, NOR- ja NAND-flash-muistit. NOR-flash-muistin tyhjennys perustuu F-N-tunnelointiin (Fowler-Nordheim), kun taas ohjelmoitiin käytetään CHEI-tekniikkaa (channel hot electron injection). NAND -pohjaisella teknologialla sekä ohjelmointi että tyhjennys perustuvat F-N-tunnelointiin. CHEI-tekniikassa elektroneja kiihdytetään sellaiselle tasolle, että ne voivat läpäistä muistisolun lähteen tai nielun ja hilan välisen eristekerroksen. F-N-tunnelointi perustuu puolestaan kvanttimekaniikkaan ja siinä tunneloidaan elektroneja solun kanavasta kelluvalle hilalle ja toisin päin. CHEI-teknologialla tallennetut tiedot pysyvät paremmin muistissa kuin F-N-tunneloinnilla tallennetut, jonka vuoksi NOR-muistit ovat luotettavampia. Yleensä massamuisteissa käytetään kuitenkin NAND-teknologiaa suuremman kirjoitusnopeuden vuoksi. [32][34] SPI-väyläisen flash-muistipiirin ohjelmointia on kuvattu tarkemmin työn toteutuksen yhteydessä.

3.3.5 Haihtumattomat RAM-muistit

Tietojen säilyminen RAM-muistissa virtojen ollessa poissa joudutaan järjestämään yleensä joko erillisen pariston avulla tai siirtämällä RAM-muistin tiedot jollekin haihtumattomalle muistille. Varmistuspariston ongelmana on kuitenkin, että se vie tilaa piirikortilla eikä kestä korkeita lämpötiloja. Haihtumattoman muistipiirin käyttöä puolestaan rajoittaa se, jos virrat katoavat piiriltä niin nopeasti, ettei koko RAM-muistin sisältöä ehditä siirtää pysyvästi toiselle muistipiirille.

Markkinoille on tullut useita uusia nvRAM-muistityyppejä (non-volatile RAM), joilla tiedot säilyvät jopa 10 vuotta ilman varmistusparistoa tai muistisolujen virkistystä. Näitä ovat esimerkiksi nvSRAM (non-volatile static RAM), MRAM (magnetoresistive RAM), FRAM (ferroelectric RAM) ja PRAM (phase-change RAM) -muistit. Näiden lisäksi on vielä kehitteillä muun muassa nanoputkiin perustuva NRAM-muisti (nano-RAM) sekä puolijohteen resistiivisyyden muuttumiseen perustuva RRAM-muisti (resistive RAM). Haihtumattomien RAM-muistien ongelmana on kuitenkin teknologian nuoruus, jonka vuoksi nvRAM-muistipiirit ovat melko kalliita. Vähäisten valmistajien vuoksi voi piireillä ilmetä myös saatavuusongelmia. Useissa tilanteissa nvRAM-piiri voidaan kuitenkin korvata toisella ominaisuuksiltaan vastaavalla nvRAM-piirillä. Kuten muillekin RAM-piireille, myös nvRAM-piirit sallivat huomattavasti useampi uudelleenkirjoituskertoja kuin flash-piirit. NvRAM-piirit ovat kuitenkin vielä varsin pienikokoisia tallennuskapasiteetiltaan. Tulevaisuudessa on tarkoitus pystyä kehittämään jokin muistityyppi, joka sopii hyvin sekä nopeaksi työmuistiksi, että suuren tietomäärän pysyvään tallennukseen. Taulukossa 4 on vertailtu eri muistiteknologioiden tallennusnopeuksia.

FRAM, MRAM ja nvSRAM -muisteille on tiedonsiirto tauotonta, joka tarkoittaa sitä, että tiedot tallentuvat muistiin sitä mukaan kuin niitä siirretään piirille. Tämä tekee tallennuksesta huomattavasti nopeampaa kuin esimerkiksi flash-muistille, joissa kirjoitus

tapahtuu sivu kerrallaan. FRAM, MRAM ja nvSRAM -muisteille voidaan myös tallentaa yhdellä komennolla tietoja niin paljon kuin halutaan. Muistin osoiteavaruuden loppuun tultaessa jatkuu tallennus automaattisesti muistin alusta.
[35]

Taulukko 4: Taulukossa on vertailtuina eri muistiteknologioiden valmistajien esitteissä mainittuja kirjoitusnopeuksia. Helpomman verrattavuuden vuoksi valitsin taulukkoon vain sarjaväylällä toimivia muisteja. Rinnakkaisväyläiset nvRAM-muistit voivat tosin tallentaa nopeammin tietoa muistiin kuin sarjaväyläiset. Esimerkiksi Cypressin 16-bitin rinnakkaisväyläisellä nvSRAM-muistilla voidaan päästä jopa 800 Mb/s kirjoitusnopeuteen (t(Write Cycle) = 20 ns)

** kirjoitusnopeus = sivun koko / sivun kirjoitusaika*
(siirtoaikaa tai tyhjennykseen kuluvaa aikaa ei ole otettu huomioon).

Muisti-tyyppi	**Kirjoitus-nopeus**	**Kirjoitus ilman erasointia?**	**Väylä**	**Malli**
Flash	1,4 Mb/s*	ei	SPI	Altera EPCS16
EEPROM	12,8 kb/s*	ei	I2C	Microchip 24AA02
PRAM	7 Mb/s*	ei	SPI	Micron 128Mb: P5Q Serial PCM
FRAM	40 Mb/s	kyllä	SPI	Ramtron FM25H20
MRAM	40 Mb/s	kyllä	SPI	Everspin MR25H40
nvSRAM	40 Mb/s	kyllä	SPI	Cypress CY14B101P
SDHC-kortti	280 Mb/s	ei	SPI	Kingston UHS-I 233X

nvSRAM

NvSRAM perustuu sekä haihtuvien että haihtumattomien muistisolujen käyttämiseen samassa piirissä. NvSRAM-muisti toimii kuin tavallinen SRAM-muisti, mutta tiedot voidaan tallentaa myös haluttaessa haihtumattomiin muistisoluihin STORE-komennon avulla. AutoStore-ominaisuus mahdollistaa sen, että muistipiirin sisältö tallentuu haihtumattomiin muistisoluihin automaattisesti, kun virrat katkeavat. Tietojen pysyvä tallennus vie valmistajan esitteen mukaan aikaa 8 ms:ia. Markkinoilla on muistipiirejä SPI-väylällä varustettuna 1 Mb:n kokoisena ja rinnakkaisväylällä varustettuja 16 Mb:n kokoisena. SPI-väylällä varustettuja piirejä saa 40 MHz:n kellonopeudella toimivina.
[36]

MRAM

MRAM- eli magnetoresistiivisen RAM-muistin toiminta perustuu transistoriin liittyneisiin magneettisiin muistielementteihin, joissa on kaksi magneettista kerrosta ja näiden välillä oleva ei-magneettinen kerros. Toisen magneettisen kerroksen polarisaation suunta pysyy kokoajan samana, kun taas toisen kerroksen polarisaatiota on mahdollista muuttaa. Polarisaation muutos vaikuttaa muistielementin resistanssiin, joka puolestaan määrä muistisolun tilan. [37] SPI-väylällä varustettua MRAM-piiriä on saatavilla 4 Mb:n kokoisena [38] ja rinnakkaisväyläistä 16 Mb:n kokoisena [39].

FRAM

FRAM:in muistisolun tilan määrittää ferroelektrinen kondensaattori, jonka polarisaatiota on mahdollista muuttaa. FRAM-muisti muistuttaa rakenteeltaan DRAM-muistia, mutta FRAM:n tapauksessa kondensaattori varaus ei purkaudu itsestään, joten muistisoluja ei tarvitse virkistää jatkuvasti. [40] FRAM-muisteja on markkinoilla muun muassa 40 MHz:n SPI-väylällä varustettuna [41] sekä asynkronisella 16-bittisellä rinnakkais-väylällä varustettuina [42].

PRAM

PRAM-muistin eli vaihemuutosmuistin toiminta perustuu muistielementin resistanssin vaihtuvuuteen, mikä johtuu muistielementin vaihemateriaalin ominaisuuksien muuttamisesta. PRAM-muistit ovat toiminnaltaan samantapaisia flash-muistien kanssa. Flash-muistin tavoin PRAM-muistille siirretään tietoa yksi sivu kerrallaan. Ennen kuin uutta sivua voidaan alkaa kirjoittamaan muistiin, täytyy odottaa, että edellinen sivu ehtii tallentua kokonaan. PRAM-muistilla yhden sivun kirjoitusaika on kuitenkin lyhyempi kuin flash-muistilla, joka mahdollistaa nopeamman tallennusnopeuden. PRAM-muistit ovat halvempia suhteessa muistin kokoonsa kuin muut edellä kuvatut uudet nvRAM-teknologiat. Kirjoitusnopeudeltaan PRAM-muistit ovat kuitenkin muita nvRAM-teknologioita hitaampia. [43] [44]

3.3.6 Muistikortti

Flash-tekniikkaan perustuvat SD-muistikortit (secure digital, kuva 12) ovat varsin yleisessä käytössä kuluttajaelektroniikassa. Nykyisin monet laitteet tukevat alkuperäistä SD-korttia pienempikokoista MicroSD- tai MicroSDHC-korttia (secure digital high-capacity). SD-kortteja saa monissa nopeusluokissa, joista nopeimmat riittävät jopa HD-tasoisen (high-definition) videon tallennukseen. Muita muistikorttityyppejä ovat muun muassa CompactFlash- ja MMC-muistikortit (multimedia card). Vaikka myös muistikortit

perustuvat flash-teknologiaan, on muistikortille kuitenkin yleensä nopeampi tallentaa kuin sarjaväyläisille flash-piireille.

Kuva 12: Kuvassa on MicroSDHC-kortti sekä sille tarkoitettu SD-kortin kokoinen sovitinlaite.

SD-kortin suuren kirjoitusnopeuden mahdollistaa se, että useammalle muistialueelle voi kirjoittaa tietoja samanaikaisesti. SD-kortille voidaan kirjoittaa tietoa joko monilohkotallennuksella (multiple-block write) ja yksilohkotallennuksella (single block write). Monilohkotallennus on yksilohkotallennusta selvästi nopeampaa. Eräässä testissä pystyttiin tallentamaan Sandiskin SD-kortille yhdelle 512 tavun kokoiselle muistilohkolle tiedot ajassa 318.88 μs käyttäen yksilohkotallennusta. Kuuden muistilohkon yhtäaikaiseen tallennukseen kului puolestaan 1.02274 ms monilohkotallennustilassa, jonka perusteella on siis noin kaksi kertaa nopeampaa käyttää monilohkotallennusta kuin yksilohkotallennusta. Monilohkotallennuksella voitiin saavuttaa kyseiselle SD-kortille 25 Mb/s:n siirtonopeus. [45] Tämän hetken nopeimmille SD-muisteille voidaan kuitenkin kirjoittaa valmistajan sivujen mukaan jopa 280 Mb/s:n nopeudella [46].

3.3.7 Muita tallennusmuotoja

Nykyiset flash-teknologiaan perustuvat SSD-levyt (solid state drive) kykenevät jopa 1,6 Gb/s kirjoitusnopeuteen, kun taas vanhanaikaisen pyörivään kiekkoon perustuvan kiintolevyn kirjoitusnopeus on ”vain” 500 Mb/s:n luokkaa. Suuren koonsa ansiosta SSD-levyille on mahdollista tallentaa tietoa hyvin pitkältä aikaväliltä, jolloin vikojen diagnosoinnissa voitaisiin tarkastaa, onko esimerkiksi taajuusmuuttajan ohjaussignaaleissa ollut häiriöitä pitkällä aikavälillä. Hinta on luonnollisesti sitä korkeampi mitä suurempaa kiintolevyä halutaan käyttää, ja SSD-levyt ovat selvästi vanhanaikaista kiintolevyä kalliimpia. Liikkumattomien osien vuoksi SSD-levy on kuitenkin kestävämpi kuin pyörivään kiekkoon perustuva kiintolevytekniikka. SSD-tekniikkaa on markkinoilla myös eSSD-siruna, joka on mahdollista integroida suoraan piirikortille [47].

4 Mahdollisia toteutustapoja

4.1 Yleistä

Tehoasteen liitäntäkortin FPGA-piirin käsittelemä tietomäärä on niin suuri, ettei kaikkia tietoja ole käytännössä mahdollista siirtää muistiin, vaikka näistä voisikin olla hyötyä vikatilanteiden diagnosoinnissa. Muistien kirjoitusnopeudet ja väylien siirtonopeudet ovat suurimmat tallennettavissa olevaa tietomäärää rajoittavat tekijät, ja vaikka tietoja pystyttäisiinkin siirtämään ja kirjoittamaan muistiin tarpeeksi nopeasti, tarvittaisiin tallennuskapasiteettia enemmän kuin on järkevää käyttää. Tämän vuoksi on valikoitava, mitä tietoja halutaan tallentaa muistiin ja kuinka paljon. Voidaan ajatella esimerkiksi tilannetta, jossa halutaan tallentaa kolmea 16-bittistä ohjaussignaalia, jotka vaihtelevat 25 MHz:n kellotaajuudella ja kolmea 12-bittistä mittaussignaalia, joiden näytetaajuus on 1 MHz. Jotta nämä signaalit saataisiin tallennettua, tarvittaisiin siirtokaista ja muistipiiri, joiden nopeus on vähintään

$$3 * 16\,\mathrm{b} * 25\,\mathrm{MHz} + 3 * 12\,\mathrm{b} * 1\,\mathrm{MHz} = 1{,}236\,\mathrm{Gb/s}\,.$$

Varsinkaan nykyisten tuotteiden kohdalla ei näin suurta tietomäärää ole mahdollista siirtää muistiin, ja todellisuudessa tietoja saatetaan haluta tallentaa vielä tätäkin enemmän.

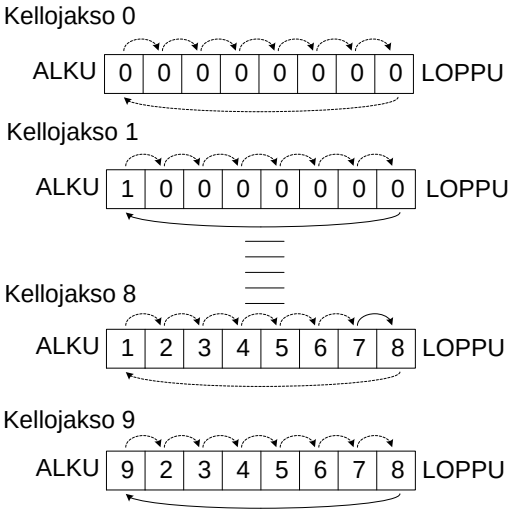
Käytännössä vianaikaisten tietojen tallennus voidaan toteuttaa esimerkiksi seuraavalla periaatteella:

1. Tietoja tallennetaan RAM-muistilla toteutettuun rengaspuskuriin jatkuvasti taajuusmuuttajan ollessa käynnissä.
2. Kun taajuusmuuttajalla havaitaan vikatilanne, aktivoituu liitäntäkortilla liipaisusignaali, jolloin rengaspuskurin sisältö siirretään RAM-muistilta haihtumattomaan muistiin, kuten flash-muistille. Liipaisusignaalin aktivoitumisen jälkeen voidaan vielä tallentaa rengaspuskuriin tietoja halutun verran, jotta myös vian jälkeisiä tapahtumia pystytään näkemään.

Jos taajuusmuuttajalta katkeaa virrat vikatilanteiden aikana, on rengaspuskurin muistiin siirtämiselle tai kirjoittamiselle aikaa hyvin rajoitetusti. Mikäli liipaisusignaali tulee liitäntäkortin ulkopuolelta, voi tästä syntyvä viive myös osaltaan vähentää puskurin pysyvään tallennukseen käytössä olevaa aikaa. Tästä syystä siirron ja tallennuksen nopeudet ovat suurimmat tallennettavissa olevaa tietomäärää rajoittavat tekijät. Tulevissa tuotteissa voisi olla kannattavaa käyttää esimerkiksi nopeampia muisteja ja väyliä, jotta tietoja ehdittäisiin tallentaa nykyistä enemmän. Huomioitavaa on kuitenkin, että samalla, kun uuden teknologian siirtonopeudet ja tallennuskapasiteetit kasvavat, saattavat myös tulevien taajuusmuuttajien suorittimien nopeudet ja käsiteltävän tiedon määrä kasvaa.

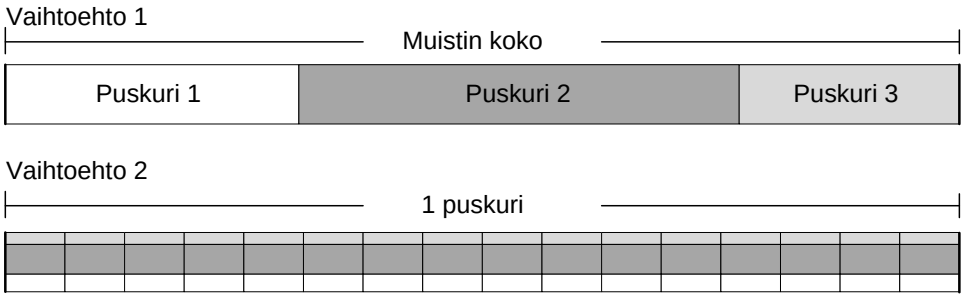
4.2 Puskuri

Rengaspuskuri toimii siten, että puskuriin tallennetaan näytteitä esimerkiksi kellosignaalin määrittämällä tahdilla, jolloin tallennusosoite kasvaa aina yhdellä jokaista kellojaksoa kohden. Kun saavutaan puskurin viimeiseen osoitteeseen, aloitetaan tallennus taas puskurin ensimmäisestä osoitteesta. Puskurin leveys kertoo yhden näytteen bittisyyden ja puskurin pituus tarkoittaa, kuinka monta näytettä puskuriin mahtuu. Puskurin koko saadaan, kun leveys kerrotaan pituudella. Rengaspuskuri voidaan toteuttaa esimerkiksi käyttäen FPGA:lle integroitua SRAM:ia tai muuta nopeaa muistia. Kuvassa 13 on havainnollistettu kahdeksan näytteen pituisen 4-bittisen rengaspuskurin toimintaa.



Kuva 13: Kuvan esimerkissä on neljän bitin, eli yhden heksadesimaalin, levyinen ja kahdeksan näytettä pitkä rengaspuskuri. Yksi ruutu kuvaa yhtä näytettä. Kellojaksolla 0 puskuri on tyhjä eli kaikki sen arvot ovat nollia. Kellojaksoilla 1-8 puskuriin tallennetaan uusia näytteitä jatkuvasti kasvavaan muistiosoitteeseen, kunnes saavutaan puskurin loppuun. Kellojaksolla 9 aloitetaan tallentaminen taas puskurin alusta, jolloin puskurin ensimmäiseen osoitteeseen kirjoitettu arvo 1 ylikirjoitetaan arvolla 9. Kuvan nuolet viittaavat puskurin osoitteen muuttumiseen jokaisen kellojakson aikana. Puskuriin kirjoitetut arvot: 1234 5678 9 (eniten merkitsevä bitti ensin).

Puskuroitavia signaaleja voidaan tallentaa muistiin ainakin kahdella eri tavalla. Kuvassa 14 on havainnollistettu tilannetta, jossa halutaan tallentaa kolmea eribittistä signaalia, joilla on sama näytetaajuus. Vaihtoehdossa 1 tallennetaan jokaista signaalia omiin puskureihinsa. Vaihtoehdossa 2 signaalit ensin yhdistetään vektoreiksi, joita tallennetaan kaikkia samaan puskuriin. Vaihtoehdon 2 ratkaisu on vaivattomampi toteuttaa kuin vaihtoehto 1, mutta se sopii lähinnä tilanteisiin, joissa kaikilla signaaleilla on sama näytetaajuus. Luettaessa vaihtoehdon 2 tavalla tallennettuja tietoja täytyy lukuohjelman tai jälkikäsittelyohjelman osattava palauttaa yhdistetyt vektorit takaisin erillisiksi signaaleiksi.



Kuva 14: Kaksi erilaista tapaa tallentaa tietoja puskuriin.

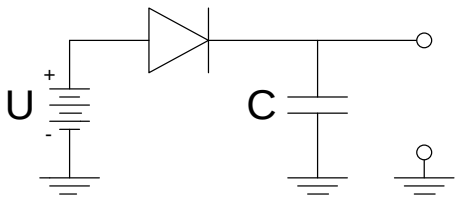
4.3 Tapoja lisätä tallennettavaa tietomäärää

Tallennettavissa olevaa tietomäärää voidaan kasvattaa sekä käyttämällä nopeampia muistipiirejä ja väyliä tai pidentämällä tallennusta varten olevaa aikaa esimerkiksi pariston tai kondensaattorin avulla. Monissa tiedonkeruuratkaisuissa käytetään esimerkiksi paristolla varustettua RAM-piiriä, jossa tiedot säilyvät muistissa myös virtojen katketessa. Myös karsimalla epäoleellisia tietoja tai käyttämällä pakkausalgoritmejä on mahdollista saada suurempi määrä hyötyinformaatiota muistiin.

Varmistusparisto

Ainakin tuotekehitykseen liittyvien testien aikana on mahdollista liittää varmistusparisto teholähteen rinnalle, jotta aikaa rengaspuskurin siirtämiseen pysyvään muistiin olisi enemmän. Toisaalta, koska varmistuspariston ansiosta SRAM-muistin rengaspuskuri ei tyhjenisi, ei erillistä pysyvää muistia ole pakko käyttää, vaan rengaspuskurin sisällön voi lukea suoraan SRAM-muistilta. Pelkkään haihtuvaan SRAM-muistiin luottaessa voidaan rengaspuskurin sisältö kuitenkin menettää, mikäli esimerkiksi paristo on tyhjentynyt.

Kuvan 15 paristokytkentä voitaisiin liittää esimerkiksi liitäntäkortin hakkuriteholähteen tulopuolelle. Jos pariston jännite on esimerkiksi 9 V, päästää paristokytkennän diodi jännitettä läpi vasta, kun hakkurille tuleva jännite laskee alle 9 voltin.



Kuva 15: Kuvassa on varateholähde. Diodi estää virrankulkemisen normaalin käytön aikana.

Elektrolyyttikondensaattori varausvarastona

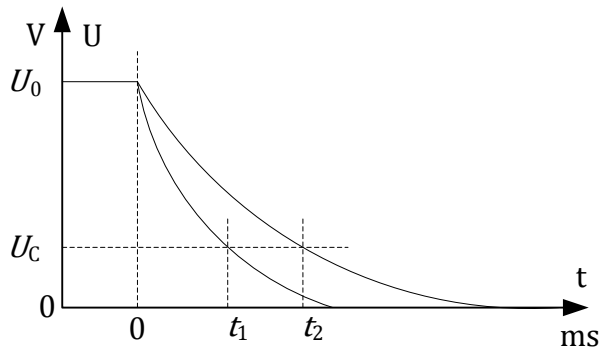
Ajatellaan liitäntäkortin jännitetuloa RC-piirinä. Jos RC-piirin resistanssi ja kapasitanssi tunnetaan, voidaan kondensaattorin varauksen purkautumiseen kuluva aika laskea yhtälöllä (1).

$$U_C = U_0 e^{-t/RC}, \tag{1}$$

jossa U_C on jännite ajan funktiona, U_0 on jännite alkutilanteessa, R on resistanssi ja C on kapasitanssi. Valitaan, että U_C on pienin jännite, jolla liitäntäkortin FPGA- ja muistipiirit pysyvät toiminnassa, jolloin voidaan laskea kapasitanssin vaikutus varauksen purkautumiseen kuluvaan aikaan:

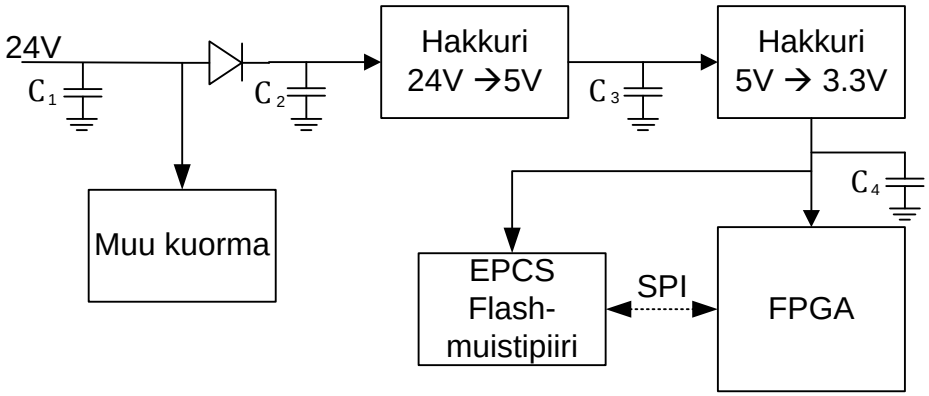
$$U_{C1} = U_{C2}$$
$$U_0 e^{-t_1/RC_1} = U_0 e^{-t_2/RC_2}$$
$$\frac{t_1}{RC_1} = \frac{t_2}{RC_2}$$
$$t_2 = t_1 \frac{C_2}{C_1} \tag{2}$$

Yhtälöstä (2) nähdään, että kondensaattorin purkautumiseen kuluva aika on suoraan verrannollinen kapasitanssiin, jonka perusteella tallennukseen käytössä olevaa aikaa voidaan lisätä kasvattamalla hakkuriteholähteen tulon kapasitanssia. Kapasitanssi voidaan kasvattaa esimerkiksi lisäämällä nykyisten kondensaattorien rinnalle lisää suuria kondensaattoreita. Rinnankytkettyjen kondensaattorien yhteinen kapasitanssi saadaan laskettua summaamalla. Kapasitanssin vaikutusta purkautumiseen kuluvaan aikaan on havainnollistettu kuvassa 16.



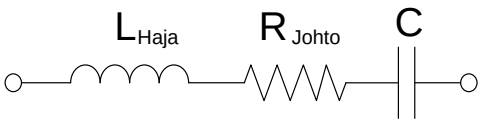
Kuva 16: Jännitteen purkautuminen kondensaattoreissa C_1 ja C_2. Purkautuminen alkaa hetkellä 0. Kun jännite alittaa tietyn arvon (U_C), piirikomponentit eivät enää toimi.

Ylimääräinen kondensaattori kannattaisi lisätä 5 voltin hakkurin tulopuolelle eli kuvassa 17 esitetyn kytkennän kondensaattori C_2:n rinnalle. Kytkennän diodi estää virran kulkeutumisen muulle kuormalle, jotta suurempi osuus kondensaattoriin varautuneesta energiasta riittäisi ylläpitämään tehoja FPGA- ja EPCS-piireillä.



Kuva 17: Tehonsyöttö FPGA- ja EPCS-piirille liitäntäkortilla. Lisäkondensaattori kannattaa lisätä kondensaattorin C_2 rinnalle.

Elektrolyyttikondensaattori sopii hyvin lyhytaikaiseksi varateholähteeksi suhteellisen korkean kapasitanssinsa vuoksi. Elektrolyyttikondensaattoreilla on kuitenkin melko korkeat häviöt ja induktanssi. Koska kondensaattoria käytetään tässä tapauksessa tasajännitteen ylläpitämiseksi, ei impedanssilla ole juurikaan vaikutusta muulloin kuin kondensaattorin varauksen purkautuessa tai latautuessa. Kuvassa 18 on esitetty epäideaalisen kondensaattorin malli. Voi myös olla, ettei lisäkondensaattoria saataisi mahtumaan liitäntäkortin koteloon. Mikäli metallikotelo poistettaisiin, kasvaa liitäntäkortin herkkyys häiriöille.



Kuva 18: Kondensaattorissa, on kapasitanssin lisäksi resistanssia ja induktanssia.
[7]

Pakkaus ja muu tiedon optimointi

Yksi tapa lisätä tallennettavaa hyötyinformaatiota on pakata tietoja ennen tallennusta. Pakkaustavat jaetaan häviöllisiin ja häviöttömiin. Häviöttömässä pakkauksessa kaikki signaalin informaatio säilyy, joten se voidaan palauttaa alkuperäiseksi, mutta häviöllisessä

pakkauksessa signaalista poistetaan epäoleellista informaatiota. RLE (run-length encoding) eli jakson pituuden koodaus on yksi yksinkertaisimmista häviöttömistä pakkaustavoista. RLE sopii tilanteisiin, jossa on paljon toistuvia peräkkäisiä arvoja. RLE:n lisäksi voitaisiin tietoja pakata häviöttömästi myös esimerkiksi merkitsemällä useimmiten esiintyviä bittijaksoja jollain lyhyemmällä tavalla (Huffmanin pakkaus).[48] Näiden pakkaus-algoritmien tehokkuutta saattaa vähentää se, että vikatilanteiden yhteydessä signaalien arvot voivat vaihdella suuresti tai poiketa muuten paljon normaalista.

Myös alinäytteistämällä tallennettavia signaaleja voidaan saada tietoa muistiin pidemmältä aikaväliltä. Tällöin kuitenkin saatetaan menettää hyödyllistäkin informaatiota. Toinen vaihtoehto on pienentää signaalin bittileveyttä. Joltain kanavalta voi olla järkevää tallentaa vain maksimiarvoja, kuten maksimilämpötila. Voi olla myös välttämätöntä jättää jotain tietoja tallentamatta, jotta tärkeämpiä tietoja pystyttäisiin tallentamaan enemmän.

4.4 Kaikki toiminnallisuus liitäntäkortilla

Käytännössä liitäntäkortille on vaikea lisätä uusia komponentteja, mikä rajoittaa monien tiedonkeruuratkaisujen käyttämistä. Helpoin tapa toteuttaa vianaikaisen tiedon tallennus nykyisten tuotteiden kanssa on käyttää rengaspuskurina liitäntäkortin FPGA-piirin SRAM-muistia, josta tiedot voidaan siirtää esimerkiksi EPCS-laitteelle, kun vikatilanne on havaittu. Tämä ratkaisu kuvataan tarkemmin seuraavassa luvussa. Tulevaisuuden taajuusmuuttajia suunniteltaessa on suositeltavaa käyttää liitäntäkortilla komponentteja, jotka mahdollistavat nykyistä suuremman tietomäärän tallennuksen vikatilanteen yhteydessä.

Muistikorttipaikan lisääminen liitäntäkortille

Nykyisissä tehoasteen liitäntäkorteissa ei ole muistikorttipaikkaa, jonka vuoksi muistikortin käyttäminen vianaikaisen tiedon tallennukseen on varsin haasteellista. Mikäli jokin liitäntäkortin digitaaliliitäntä voitaisiin jättää vapaaksi, olisi mahdollista suunnitella tähän sopiva muistikorttipaikan sisältävä moduuli. Ongelmana tässä on kuitenkin se, että liitäntäkortin portit ovat käytössä jo muihin tarkoituksiin. Muistikortin käyttäminen onnistuu parhaiten, mikäli muistikorttipaikka lisättäisiin piirikortille jo suunnittelu-vaiheessa. Muistikortin käytön etuna liitäntäkortille kiinteästi asennettuihin muistipiireihin verrattuna olisi se, että jos liitäntäkortti rikkoutuu vikatilanteen yhteydessä, saataisiin vikatilanteen aikaiset tiedot vielä pelastettua suhteellisen helposti, mikäli muistikortti pysyy kuitenkin ehjänä.

Yleisin käytössä oleva muistikortti tyyppi on SD-kortti. SD-kortti vaatii toimiakseen varsin monimutkaiset ajurit, jonka vuoksi FPGA:n kommunikointi SD-kortin kanssa kannattaa toteuttaa SD-ohjainpiirin avulla. SD-ohjain voisi kommunikoida FPGA-piirin kanssa esimerkiksi SPI-väylän kautta. SD-kortteja on markkinoilla varsin suurikokoisina [46], mikä mahdollistaa sen, että vanhoja vikatietoja ei ole välttämätöntä poistaa kortilta uusien tieltä, vaan uudet tiedot voidaan kirjoittaa aina uuteen muistiosoitteeseen. Samoin kuin mille tahansa flash-muistille, myös SD-kortille on mahdollista uudelleenkirjoittaa vähintään 100 000 kertaa yhdelle muistialueelle. Useiden SD-korttien kirjoitusnopeus on suurempi kuin flash-muistipiirien, jonka ansiosta tietoja ehditään tallentaa varsin paljon myös silloin, kun tallennukseen käytössä oleva aika on rajallinen.

RAM-muistipiirin lisääminen

Tämän hetken tuotteissa FPGA:n sulautettu SRAM-muisti riittää myös rengaspuskurin tarpeisiin. Jos tulevaisuuden tuotteissa SRAM-muistia tarvitaan enemmän muihin toiminnallisuuksiin, joudutaan puskuria varten lisäämään erillinen muistipiiri. Mikäli rengaspuskuri toteutetaan nvRAM:lla, ei tietoja tarvitse siirtää erikseen puskurista erilliselle haihtumattomalle muistipiirille, vaan virtojen lähtiessä puskurin tiedot jäisivät automaattisesti muistiin. NvRAM-pohjainen ratkaisu olisi suhteellisen helppo ja nopea toteuttaa, mikäli se otetaan jo piirikortin suunnitteluvaiheessa huomioon. Jotta tietoja ehdittäisiin tallentaa nvRAM-muistille tarpeeksi, täytyisi käyttää yhtä tai useampaa 16-bittisellä rinnakkaisväylällä varustettua nvRAM-piiriä. Rinnakkaisväyläisten piirien suurin ongelma on, että ne vievät paljon pinta-alaa piirikortilta.

4.5 Ulkoinen laite vianaikaisen tiedon tallennusta varten

Vianaikaisten tietojen tallennusta varten on mahdollista suunnitella ulkoinen laite, joka kommunikoi liitäntäkortin kanssa esimerkiksi kuitujen tai langattoman yhteyden kautta. Tässä ratkaisussa rengaspuskuri toteutetaan FPGA:n SRAM-muistilla, ja vikatilanteen yhteydessä puskurin sisältö siirretään ulkoiselle laitteelle. Ratkaisun etuna on se, että vaikka liitäntäkortti rikkoutuisi, voidaan vianaikaisia tietoja ehtiä siirtämään ulkoisen laitteen muistiin, josta tiedot ovat luettavissa jälkikäteen. Toisin kuin tehoasteen liitäntäkortti, galvaanisesti taajuusmuuttajasta erotettu ulkoinen laite pysyy todennäköisesti ehjänä kriittisissäkin vikatilanteissa. Taajuusmuuttajasta erillisen teholähteen ansiosta kaikki ulkoiselle laitteelle siirretyt tiedot ehtisivät tallentua silloinkin, kun liitäntäkortin virrat katkeavat. Ongelmana on kuitenkin, että liitäntäkortissa ei ole nopeita digitaaliliittimiä tiedonkeruuyksikköä varten, ellei joitain portteja jätetä tarkoituksella vapaaksi. Ulkoisen laitteen käyttö onnistuu parhaiten, mikäli liitäntäkortille lisätään jo suunniteltaessa tarpeeksi nopeat portit ulkoiselle tiedonkeruuyksikölle.

Tietojen siirtäminen erilliselle, omalla teholähteellä varustetulle laitteelle on kannattavaa, koska tällöin lähinnä vain siirtonopeus rajoittaisi tallennettavissa olevien tietojen määrää eikä tietojen muistipiirille kirjoitukseen kuluvaa aika aiheuta rajoituksia. Ulkoiseen laitteeseen tallennettavissa oleva tietomäärä riippuu lähinnä siirtokaistan nopeudesta ja koodaustavasta sekä rengaspuskurin koosta. Esimerkiksi, jos tietoja siirrettäisiin 25 Mb/s nopeudella 4B/5B-koodausta käyttäen, kestäisi 400 kb:n kokoisen puskurin siirtämiseen ulkoiselle laitteelle

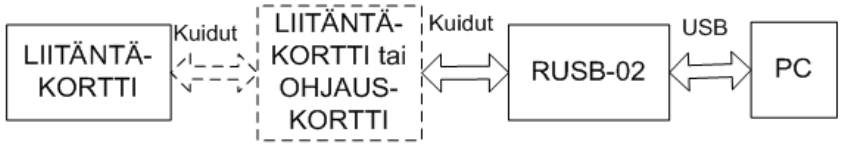
$$\frac{400\ \text{kb}}{{}^{4}/_{5} * \ 25\ 000\ \text{kb/s}} = 20\ \text{ms} \quad .$$

Testeissä osoitettiin, että liitäntäkortin suurien kondensaattorien varaus riittää ylläpitämään tiedonsiirtoon tarvittavia tehoja ainakin 20 ms:n ajan. Jotta ratkaisu toimii, pitäisi myös vastaanottavalla laitteella olla 400 kb:n kokoinen puskuri, ellei muistin kirjoitusnopeus ole saamaa suuruusluokkaa siirtonopeuden kanssa. Protokollasta riippuen tietopaketit voivat sisältää koodauksen lisäksi myös alku- ja lopputunnisteita tai CRC-tarkisteavaimia, jotka osaltaan vievät kapasiteettia hyötytiedon siirtämiseltä. Ulkoisen laitteen ja liitäntäkortin välisessä kommunikaatiossa voitaisiin käyttää esimerkiksi ABB:n omaa DDCS-protokollaa (distributed drive control system).

Nykyisen teknologian käyttäminen

Ulkoiseksi tiedonkeruuyksiköksi ei ole välttämätöntä suunnitella uutta laitetta, vaan myös nykyistä teknologiaa (kuva 19), kuten toista liitäntäkorttia tai RUSB-02-laitetta, voidaan käyttää vianaikaisen tiedon tallennukseen. RUSB-02:ta käytetään normaalisti sovitin-laitteena DDCS-kuituväylän ja USB-väylän välissä. RUSB-02:n kautta voidaan esimerkiksi lukea PC:lle tietoja liitäntäkortin muistipiireiltä tai A/D-muuntimelta.

RUSB-02:n käyttäminen tiedonkeruuyksikkönä onnistuu pelkästään modifioimalla tämän FPGA:n konfiguraatiota. Tietojen tallennukseen voidaan käyttää FPGA:n EPCS-konfigurointipiiriä. RUSB-02 saa tehonsa USB:n kautta, joten laitteen pitäisi olla kiinni PC:ssä aina, kun sitä käytetään. On kuitenkin suhteellisen yksinkertaista suunnitella USB-porttiin liitettävä 5 voltin teholähde, jolloin RUSB:ia olisi mahdollista käyttää tiedonkeruuyksikkönä myös ilman PC:tä. Erityisesti muistien koot ja kuituväylien nopeudet rajoittavat RUSB-02:n käyttömahdollisuuksia.



Kuva 19: Kuvassa on esitelty, mitä nykyistä teknologiaa voidaan käyttää tiedonkeruuyksikkönä. ”Liitäntäkortti tai ohjauskortti” -lohko on esitetty katkoviivalla, koska tiedot voidaan tallentaa myös suoraan liitäntäkortilta RUSB-02-sovitinlaitteen konfigurointimuistille.

Ulkoisena tiedonkeruuyksikkönä voidaan käyttää myös toista liitäntäkorttia. Liitäntäkortissa ei ole kuitenkaan USB-tukea, jonka vuoksi tietojen siirtämiseen PC:lle täytyy joka tapauksessa käyttää kuitu-USB-sovitinta. Liitäntäkortti on myös fyysisesti ison kokoinen ja se sisältää monia tarpeettomia ominaisuuksia. Toisen liitäntäkortin käyttäminen sopii paremmin tiedonkeruuyksikön prototyypiksi tai FPGA:n logiikan suunnittelualustaksi kuin varsinaiseksi tiedonkeruuyksiköksi. Liitäntäkortilla on kuitenkin suuremmat SRAM- ja flash-muistit kuin RUSB-02:lla.

Tiedonkeruuyksikkö on mahdollista integroida myös ohjauskorttiin, johon liitäntäkortti on muutenkin yhteydessä. Jo nykyisissä taajuusmuuttajissa tallennetaan joitain vikatietoja ohjauskortin muistille. Jotta mahdollisimman paljon tietoja ehdittäisiin siirtää, kannattaa vian liipaisun jälkeen käyttää koko siirtokaista vain rengaspuskurin sisällön siirtämiseen. Ei voida olla kuitenkaan varmoja, säilyykö ohjauskortilla virrat vikatilanteen aikana tarpeeksi kauan, jotta tallennus onnistuu varmasti.

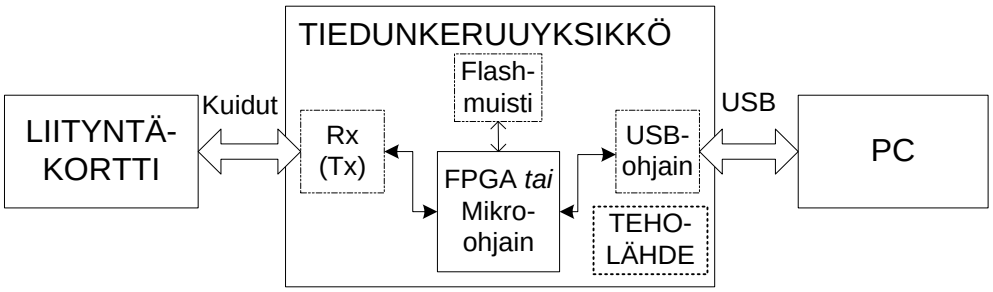
Uuden tiedonkeruuyksikön suunnittelu

Uusi tiedonkeruuyksikkö voidaan suunnitella joko mikro-ohjain- tai FPGA-pohjaisena. Mikro-ohjaimeen perustuva ratkaisu on FPGA:han perustuvaa ratkaisua vaivattomampi toteuttaa helpomman ohjelmoitavuuden vuoksi. Markkinoilla on myös useita mikro-ohjaimia, joille on valmiiksi integroituna monia tiedonkeruuyksikön tarvitsemia ominaisuuksia, mikä osaltaan helpottaa suunnittelua. Toisaalta laitteen yksinkertaisuuden vuoksi myös FPGA-pohjainen ratkaisu voidaan toteuttaa suhteellisen vähällä työllä. Kuvassa 20 on esitetty ulkoisen tiedonkeruuyksikön välttämättömimmät komponentit.

Laite voisi kommunikoida liitäntäkortin kanssa kuituvastaanotin (Rx) - ja kuitulähetin (Tx) -porttien kautta. Kuitulähtöportti ei ole tosin välttämätön riippuen, mitä toiminnallisuuksia laitteelle halutaan. Tiedon tallennukseen riittäisi hyvin pelkkä kuitutuloportti, mikäli liitäntäkortin ja tiedonkeruuyksikön välillä ei tarvita kaksisuuntaista kommunikaatiota. Monet protokollat vaativat kuitenkin toimiakseen kaksisuuntaisen siirtokaistan. Langatonta

tiedonsiirtoa ei nykyisissä tuotteissa vielä käytetä, mutta tätä voi olla tulevaisuudessa aiheellista harkita.

Koska lähes jokaisesta PC:ssä on USB-portti, kannattaa tiedonkeruuyksikön kommunikoida PC:n kanssa USB-väylän kautta. Markkinoilla on mikro-ohjaimia, joihin USB-tuki on integroituna, jolloin USB-portti voidaan liittää suoraan mikro-ohjaimeen. Mikäli tiedonkeruuyksikkö toteutetaan FPGA-pohjaisesti, joudutaan FPGA:n ja USB-portin välillä käyttämään USB-ohjainta, joka kommunikoi FPGA:n kanssa esimerkiksi SPI-väylän kautta.



Kuva 20: Kuvassa on esitetty lohkokaavio mahdolliselle suunniteltavalle tiedonkeruu-yksikölle. Osa ominaisuuksista voi olla integroituina mikro-ohjaimelle.

Tiedonsiirron lisäksi USB-standardi mahdollistaa myös tehonsyötön. USB 2.0 -standardin mukaiset laitteet antavat maksimissa 2,5 wattia tehoa, joka riittää hyvin tiedonkeruu-yksikön käyttöön. Koska USB-väylä antaa 5 voltin jännitettä, mutta FPGA-piiri toimii usein 3,3 voltin jännitteellä, täytyy jännite laskea oikealle tasolle hakkuriteholähteen avulla. Pelkän USB-teholähteen käytössä on ongelmana kuitenkin se, jos tiedonkeruuyksikkö halutaan käyttää myös ilman, että se on kytkettynä tietokoneeseen. Tämän vuoksi, USB-teholähteen lisäksi, on hyvä käyttää myös paristoa tai muuta teholähdettä. Suunnittelussa kannattaa ottaa myös huomioon, ettei pariston vaihtaminen saa olla liian vaikeaa. Pariston tilalla voidaan käyttää myös esimerkiksi USB-väylän kautta latautuvaa akkua.

Mikäli tiedonkeruulaite suunnitellaan mikro-ohjainpohjaiseksi, kannattaa valita sellainen piiri, jossa on riittävästi omaa flash-muistia tiedonkeruulaitteen tietojen tallennusta varten, jolloin ulkoista flash-piiriä ei tarvita. Integroidulla flash-muistilla varustettuja FPGA-piirejä on markkinoilla varsin vähän. Mikäli FPGA:lle valittaisiin riittävän iso konfigurointimuisti, ei toista flash-piiriä tarvita, vaan konfigurointimuistia voidaan käyttää myös vikatietojen tallennukseen. Flash-muistipiirin sijasta tiedonkeruuyksikkö voisi sisältää myös SD-korttipaikan.

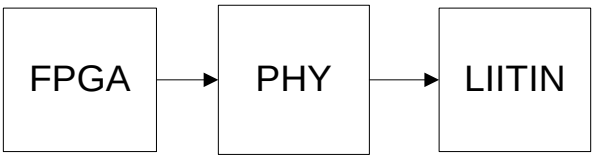
Sekä mikro-ohjain tai FPGA -piiriä valitessa on hyvä myös huomioida, että piiri sisältää riittävästi integroitua RAM-muistia, jotta laitteelle siirrettyjä tietoja voitaisiin puskuroida tarpeeksi ilman, että ulkoista RAM-piiriä tarvitsee käyttää.

4.6 Tiedonkeräys kokonaan ulkoisella laitteella

Tulevaisuuden taajuusmuuttajia suunniteltaessa on syytä arvioida, kannattaako kerättävät tiedot siirtää ulkoiselle laitteelle jatkuvasti taajuusmuuttajan ollessa käynnissä. Teoriassa tarvittavan nopea tiedonsiirto on toteutettavissa monilla uusilla siirtoteknologioilla, kuten SATA-väylällä, joka mahdollistaa tietojen siirtämisen myös suoraan kiintolevylle. Nopeat väylät voivat kuitenkin aiheuttaa paljon häiriöitä ja niiden häiriöherkkyys on suuri, joten ei ole varmaa, kuinka hyvin nopea tiedonsiirto onnistuu suurihäiriöisessä taajuusmuuttaja-ympäristössä. Galvaanisen erotuksen saamiseksi sopisivat ratkaisuun lähinnä kuituteknologiat, koska langattomat lähiverkot eivät ole riittävän nopeita kaiken halutun tiedon lähetystä varten. Gigabit Ethernet -teknologiaa on sekä optisena että sähköisesti tietoa kuljettavina.

Suurinopeuksisen teknologian suunnittelu on huomattavasti haastavampaa kuin hitaamman teknologian, joten se vaatii yleensä suurta ammattitaitoa. Kun johtimen pituus on selvästi signaalin aallonpituutta lyhyempi, ei johtimessa tapahtuvista heijastuksista tarvitse välittää. Suurilla siirtonopeuksilla, jolloin aallonpituudet ovat hyvin lyhyitä, täytyy johdinta kuitenkin tarkastella siirtolinjana. Jos siirtolinjan ominaisimpedanssi poikkeaa kuorman impedanssista, voi signaalista heijastua osa takaisin. Heijastumisen vuoksi signaali saattaa heikentyä tai vääristyä vastaanottimella. Heijastuksista voidaan päästä eroon sovittamalla siirtolinja kuormaan päätteen avulla. [7] Vaikka kuituja ei tarvitsekaan tarkastella siirtolinjana, voi suurista nopeuksista aiheutua kuitenkin ongelmia piirikortin väylillä ja liittimissä.[27]

Monet FPGA-perheet, kuten Alteran Cyclone IV GX, sisältävät tuen monille nopeille siirtotekniikalle – myös Gigabit Ethernetille [12]. Alteran kotisivuilla on myös valmiita IP-lohkoja protokollien toteuttamiseksi. FPGA-piirillä voidaan esimerkiksi toteuttaa MAC-protokolla (media access control), jota tarvitaan esimerkiksi kehysten lähettämistä ja vastaanottamista varten. Ethernet-liitin liitetään FPGA-piiriin PHY-piirin (physical layer) kautta (kuva 21). PHY-piiri huolehtii muun muassa tietojen enkoodaamisesta ennen lähetystä ja dekoodaamisesta tietoa vastaanottaessa.



Kuva 21: Yksinkertaistettu lohkokaavio FPGA:n liittämisestä Gigabit Ethernet -väylään.

4.7 Ratkaisujen vertailu

Helpoiten toteutettavissa olisi ratkaisu, jossa tietojen pysyvään tallennukseen käytetään liitäntäkortin FPGA:n konfigurointilaitetta. Tämän ratkaisun suurin ongelma on rajallinen tallennettavissa oleva tiedon määrä ja se, ettei tietoja saada välttämättä luettua, mikäli liitäntäkortti rikkoutuu vikatilanteessa. Myös ratkaisu, jossa tietoja siirretään ulkoiselle laitteelle pysyvää tallennusta varten, on mahdollista toteuttaa nykyisten tuotteiden kanssa, mikäli liitäntäkortilla on vapaita liittimiä. Eniten tietoja saataisiin muistiin, mikäli kaikki haluttu tieto siirretään ulkoiselle laitteelle jatkuvasti taajuusmuuttajan ollessa käynnissä esimerkiksi Gigabit Ethernet -väylän kautta. Tällaista ratkaisua ei olisi kuitenkaan mahdollista käyttää nykyisten tuotteiden kanssa ja ratkaisun suunnittelu olisi hyvin haastavaa. Eri toteutuksia on vertailtu taulukossa 5.

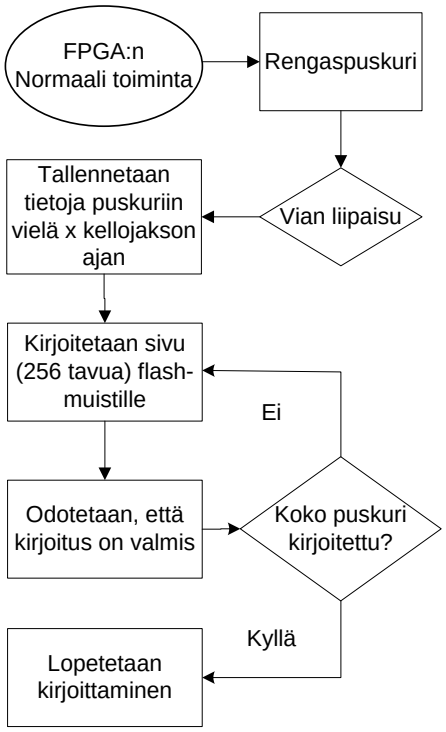
Taulukko 5: Eri ratkaisujen vertailu. Taulukossa on otettu huomioon vain kunkin kategorian helpoiten toteutettavat vaihtoehdot.

	Kaikki toiminnallisuus liitäntäkortilla	**Vain puskuri ja lähetys liitäntäkortilla**	**Kaikki tiedot lähetetään ulkoiselle laitteelle**
Käyttö nykyisten tuotteiden kanssa	Kyllä	Ehkä	Ei
Toteuttaminen	Helppo	Helppo	Vaikea
Uusi laite suunniteltava	Ei	Suositeltavaa	Kyllä
Tietoa mahdollista tallentaa	Vähiten	-	Eniten
Hinta	Halpa	-	Kallis
Tiedot muistista liitäntäkortin rikkoutuessa	Ehkä	Kyllä	Kyllä

5 Esimerkkitoteutus

5.1 Toimintaperiaate

Työssä toteutettiin ratkaisu, jossa kaikki vianaikaisen tiedonkeräykseen liittyvä toiminnallisuus on liitäntäkortilla. Ratkaisun toiminta (kuva 22) perustuu siihen, että tietoja tallennetaan jatkuvasti FPGA-piirin SRAM-muistilla toimivaan rengaspuskuriin taajuusmuuttajan ollessa käynnissä. Kun vikatilanne havaitaan, siirretään rengaspuskurin sisältö SPI-väylää pitkin FPGA:n konfiguirointipiirille.

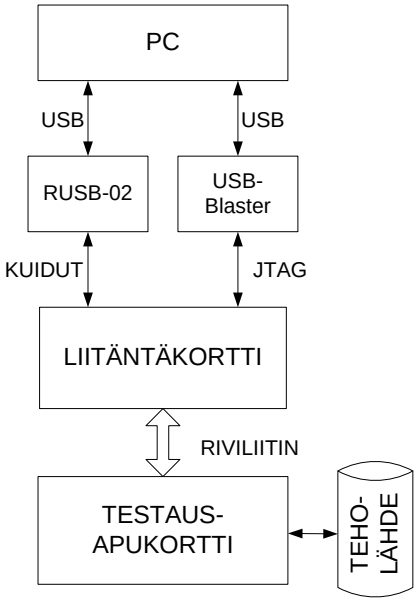


Kuva 22: Taajuusmuuttajan normaalin toiminnan aikana tallennetaan tietoja jatkuvasti rengaspuskuriin. Kun vikatilanne havaitaan, siirretään puskurin sisältö konfigurointi-laitteelle pysyvää tallennusta varten. Konfigurointilaitteelle on mahdollista kirjoittaa tietoa vain 256 tavua kerrallaan, joten tätä suuremmat puskurit täytyy kirjoittaa useammassa osassa.

5.2 Kehitysympäristö

Kehitystyötä varten rakennettu testijärjestelmä (kuva 23) sisälsi tietokoneen ja liitäntäkortin sekä näiden välillä olevat USB-Blaster ja RUSB-02 -sovitinlaitteet. Näiden lisäksi testijärjestelmään kuului vielä teholähde, joka liitettiin liitäntäkorttiin

testausapukortin kautta. USB-Blasteria käytettiin konfiguraation lataamiseen liitäntäkortin FPGA:lle tai EPCS-laitteelle JTAG-liittimen kautta. RUSB-02:ta tarvittiin puolestaan kaksisuuntaista kuituliikennettä varten PC:n ja liitäntäkortin välillä. Testausapukortin testipisteiden kautta pystyi syöttämään liitäntäkortin FPGA:lle signaaleja tai lukemaan signaaleja FPGA:n pinneiltä. Koska liitäntäkorttia voitiin ohjata PC:n kautta, ei ohjauskorttia tarvittu järjestelmän toiminnan testaamiseen.

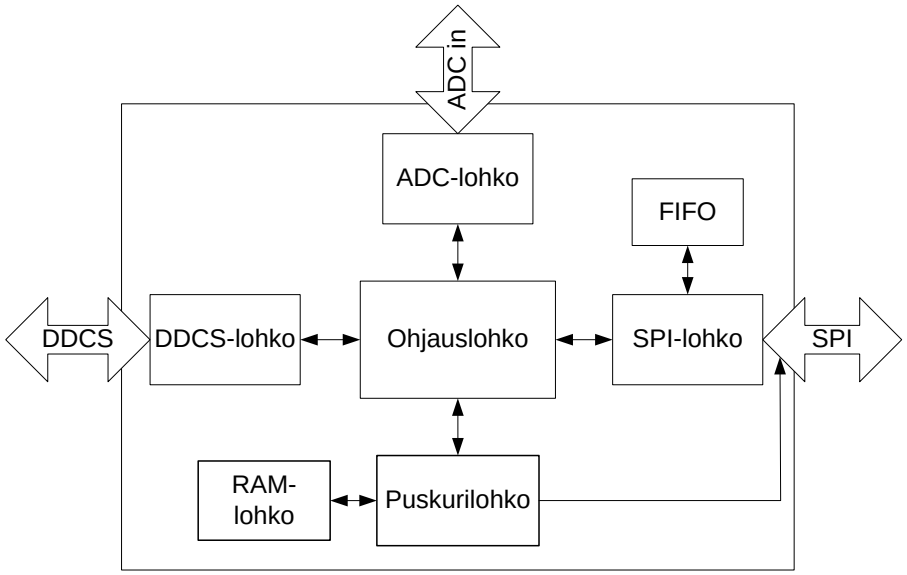


Kuva 23: Kuvassa on esitettynä kehitysympäristö, jolle tiedonkeräyslohko toteutettiin.

Kehitystyötä varten liitäntäkortilta oli otettu käyttöön A/D-muunnin kaksi lediä sekä valokuituliitännät lähetystä (Tx) ja vastaanottoa (Rx) varten.

5.3 Logiikka

Logiikan kirjoittamiseen ja kääntämiseen käytettiin *Quartus II* -ohjelmaa. Rengaspuskurin toiminta ja tiedonsiirto EPCS-laitteelle toteutettiin puskurilohkon avulla. RAM-lohkoa käytettiin puskurilohkon apuna integroidun SRAM-muistin hyödyntämistä varten. Puskurilohkolla luotiin signaalit SPI-kommunikaatiota varten ja määritettiin RAM-lohkon luku- ja kirjoitusosoitteet. DDCS- ja SPI-lohko mahdollistivat yhdessä EPCS-laitteen lukemisen ja tyhjentämisen PC:n kautta. ADC-lohkon (analog-to-digital converter) vastasi puolestaan FPGA:n kommunikoinnista A/D-muuntimen kanssa.



Kuva 24: Kehitysympäristön VHDL-ohjelman toiminnalliset lohkot. Ohjauslohkon kautta hallinnoitiin muita lohkoja, ja siinä oli määritetty FPGA:lle sisääntulevat ja lähtevät signaalit. DDCS-lohko vastasi kommunikaatiosta PC:n ja liitäntäkortin välillä RUSB-02:n kautta. ADC-lohko huolehti kommunikaatiosta A/D-muuntimen ja FPGA:n välillä. SPI-lohkon kautta voitiin lukea ja poistaa tietoja flash-muistilta. Puskurilohko vastasi rengaspuskurista ja tietojen tallennuksesta SPI-väylää pitkin flash-muistille. RAM-lohko huolehti rengaspuskurin lukemisesta ja kirjoittamisesta tiettyyn ositteeseen FPGA:n SRAM-muistille.

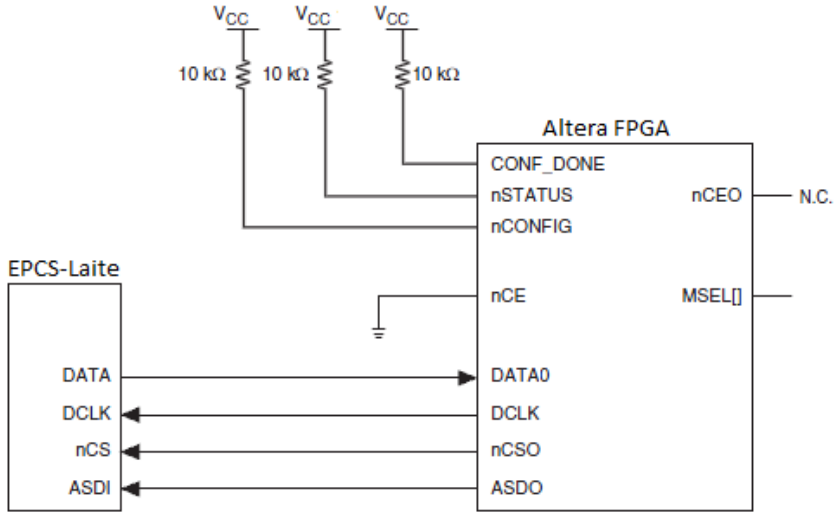
5.4 EPCS-konfigurointilaite

Flash-tekniikkaan perustuvaa EPCS-laitetta käytetään Alteran FPGA-piirien konfigurointiin käynnistyksen yhteydessä. EPCS-laitetta voi käyttää kaikkien FPGA-piirien kanssa, jotka tukevat AS-konfiguraatiota (active serial). Yhdellä EPCS-laitteella on mahdollista konfiguroida useita FPGA-piirejä, mutta yhden FPGA-piiriin ei voi kuitenkaan kytkeä kuin yhden EPCS-laitteen.

EPCS-laitetta on myynnissä 1, 4, 16, 64 ja 128 megabitin kokoisina. Tässä työssä käytettiin 16 Mb:n EPCS16-piiriä. EPCS16-piirin muistikonfiguraatiota rakentuu 32 sektorista, jotka on jaettu 256 sivuun. Yhden sivun koko on 256 tavua. EPCS16-muistilta voidaan tyhjentää vähintään sektori kerrallaan. Muistille kirjoitetaan puolestaan aina sivu kerrallaan.

EPCS:n käyttäminen muistipiirinä

Koska liitäntäkortin FPGA:n konfiguraatio ei täytä koko EPCS16-laitteen muistiavaruutta, voidaan käyttämättömille muistialueille tallentaa tietoa, kuten mille tahansa flash-muistille. EPCS-laitteen käyttäminen muistipiirinä onnistuu samalla periaatteella kuin SPI-väylällä varustettujen flash-muistipiirien käyttäminen yleensäkin (kuva 25). Koska SPI-väylä ei ole standardi, on väylän spesifikaatioissa kuitenkin eroja eri valmistajien välillä.



Kuva 25: EPCS-laitteen kommunikaatio FPGA-piirin kanssa.

EPCS-laitteen SPI-portit ovat DATA, DCLK, ASDI ja nCS, joita FPGA:lla vastaavat portit DATA0, DCLK, ASDO ja nCS0 (taulukko 6). FPGA:n portit nCONFIG, nSTATUS ja CONF_DONE on asetettu käyttöjännitteeseen.

Taulukko 6: EPCS-laitteen ja FPGA:n välisen SPI-väylän signaalit.

DATA	Signaali sarjamuotoisena EPCS:ltä FPGA:lle lukuoperaation ja konfiguroinnin aikana. Lukuoperaation aikana nCS:n on oltava alhaalla. DATA-signaali tahdistetaan laskevalla kellonreunalla.	DATA0
DCLK	FPGA:lla luotu kellosignaali, joka tahdistaa tiedonsiirtoa	DCLK
ASDI	Signaali sarjamuotoisena FPGA:lta EPCS:lle. Tahdistetaan nousevalla kellonreunalla.	ASDO
nCS	Kontrollisignaali, jolla muistipiiri aktivoidaan. nCS on aktiivinen alhaalla. nCS-signaalin on noustava ylös, kun tallennettava data on siirretty EPCS:n välimuistiin	nCS0

Muistioperaatiot

Muistioperaatiota suorittaessa siirretään EPCS-laitteen ASDI-pinnille ensin operaatiota vastaava kahdeksan bitin pituinen operaatiokoodi niin, että eniten merkitsevä bitti (MSB, most significant bit) siirretään ensimmäisenä. Operaatiokoodia ajettaessa täytyy nCS-signaalin mennä alas. Operaatiokoodin lähetyksen jälkeen lähetetään luku-, kirjoitus- ja tyhjentämisoperaatioissa aloitusosoite. EPCS16-laitteen muistiosoitteet ovat 24 bitin mittaisia. Kun tietoja siirretään FPGA:lta EPCS:lle, tahdistetaan signaali nousevalla kellonreunalla. Kirjoitus- ja tyhjennysoperaatioissa nCS-signaalin tulee nostaa kahdeksalla jaollisen kellojaksomäärän jälkeen ylös tai muuten operaatio hylätään. Lukuoperaation aikana tiedot siirretään EPCS:ltä FPGA:lle DATA-johdinta pitkin laskevalla kellonreunalla. nCS-signaali nostetaan ylös, kun tietojen lukeminen halutaan lopettaa.

Tämän työn kannalta oleellisia operaatioita ovat *Write enable-*, *Read status-*, *Read bytes-*, *Erase sector* ja *Write bytes* -operaatiot (Taulukko 7). Puskurilohkolla käytettiin työssä kaikkiin operaatioihin 20 MHz:n kelloa, vaikka myös 25 MHz:n kellotaajuuden käyttäminen olisi joidenkin operaatioiden kohdalla ollut mahdollista. Muistioperaatioita on kuvattu tarkemmin liitteessä B.

Taulukko 7: Työn kannalta oleelliset operaatiot sekä näiden operaatiokoodit ja maksimi-kellotaajuudet.

Operaatio	**Operaatiokoodi**	**DCLK f_{MAX} (MHz)**
Write enable	0000 0110	25
Read status	0000 0101	25
Read bytes	0000 0011	20
Write bytes	0000 0010	25
Erase sector	1101 1000	25

[49]

Kirjoitusnopeus EPCS-laitteelle

Tietyn tietomäärän kirjoitukseen kuluva aika voidaan arvioida kaavalla (3).

$$t_K = n\,(t_{OA} + t_D + t_{WB}), \tag{3}$$

jossa t_{OA} on aika, joka kuluu operaatiokoodien (*Write enable* ja *Write bytes*) ja osoitteen siirtämiseen SPI-väylää pitkin, t_D on aika, joka kuluu muistiin tallennettavan tiedon siirtämiseen muistille, t_{WB} kertoo, kuinka kauan yhden sivun tallentuminen muistille kestää ja n on tallennettavien sivujen määrä. t_{WB} voidaan selvittää joko kokeellisesti tai katsomalla valmistajan esitelehdestä. Yhden sivun koko EPCS16-piirillä on 256 tavua. Koska t_{WB} on näistä selvästi suurin aikakomponentti, voidaan kaava yksinkertaistaa muotoon

$$t_K \approx n\, t_{WB}\,. \tag{4}$$

5.5 Vikatilanteiden emulointi

Kehitysympäristössä voitiin emuloida vikatilanteita liitäntäkortin jännitteentarkkailupiirin avulla. Kun jännitteentarkkailupiirillä havaittiin jännitteen pudonneen alle 18 volttiin, lähetti tämä FPGA:lle liipaisusignaalin, joka laukaisi vianaikaisten tietojen siirron. Liipaisu saatiin aikaiseksi esimerkiksi laskemalla jännitettä säädettävästä teholähteestä alle liipaisurajan tai kytkemällä jännite apukortin kautta yhdelle liitäntäkortin FPGA:n pinnille. Näiden liipaisutapojen etuna oli se, että liitäntäkortti ei sammunut ”vikatilanteen” yhteydessä. Myös silloin, kun virrat katkesivat liitäntäkortista kokonaan, käynnistyi tallennus tarkkailupiirin ansiosta automaattisesti.

5.6 Tallennettavissa olevan tietomäärän testaus

Vikatilanteen yhteydessä tallennettavissa olevaa tietomäärää voitiin testata katkaisemalla virrat liitäntäkortilta, koska tämä liipaisi rengaspuskurin siirtämisen flash-muistille. Jälkeenpäin voitiin lukea EPCS:ltä, kuinka paljon tietoja oli ehditty tallentaa. Tietojen lukeminen PC:n komentoriviltä on kuvattu liitteessä C.

Virtoja ei katkaistu teholähteen virtakytkimestä, vaan irrottamalla virtajohdot liitäntäkortilta, koska teholähteen lähdön oman kapasitanssin vuoksi liitäntäkortin tulon jännite ei laskisi nollaan välittömästi, minkä takia testit antaisivat todellista tilannetta optimistisemman käsityksen tallennusta varten käytössä olevasta ajasta. Testeissä huomattiin, että lisäämällä kondensaattoreita liitäntäkortin jännitetulon kondensaattorien rinnalle saatiin tallennusta varten käytettävissä olevaa aikaa kasvatettua huomattavasti. Ilman lisäkondensaattoreita ei aikaa tarvittavan tietomäärän siirtämiseen ole kuitenkaan välttämättä tarpeeksi.

Todellisissa tilanteissa on teholähteen ja liitäntäkortin välissä enemmän kapasitanssia kuin testitilanteessa oli (kuva 26). Kapasitanssia lisäävät muun muassa teholähteen omat kondensaattorit sekä mahdollisten teholähteen ja liitäntäkortin välisten korttien, kuten hila-ajurikortin, kondensaattorit. On kuitenkin huomioitavaa, että liitäntäkortti ei ole ainoa kuorma, johon näiden kondensaattorien varaus purkautuu, vaan taajuusmuuttajan tehoelektroniikkakomponentit vievät huomattavasti pienitehoista digitaalielektroniikkaa enemmän energiaa. Voidaan kuitenkin olettaa, että taajuusmuuttajakäytössä liitäntäkortin tehot pysyvät riittävän korkealla kauemmin kuin koetilanteessa, jossa tehot irrotetaan suoraan liitäntäkortin jännitetulolta.

Kuva 26: Tehonsyöttö liitäntäkortille kulkee hila-ajurikortin kautta

Yhden sivun tallennus

Yhden sivun (256 tavua) kirjoitukseen EPCS-laitteelle kuluva saatiin selville laskemalla, kuinka monta kellojaksoa tilarekisterin 0-bitti oli ylhäällä kirjoitusoperaation aikana. Tätä työtä varten tehdyssä testissä kului yhden sivun kirjoitukseen keskimäärin 690 µs. Myllyniemen diplomityötään [4] varten tekemässä vastaavassa testissä saatiin lyhimmäksi sivun kirjoitukseen kuluvaksi ajaksi puolestaan 110 µs ja pisimmäksi 870 µs. Testeissä saadut tulokset ovat huomattavasti pienempiä kuin valmistajan esitteessä annettu tyypillinen aika 1,5 ms tai maksimiaika 5,0 ms [49]. Piirien valmistuserissä voi kuitenkin olla eroja, joten suunnittelun lähtökohtana on suositeltavaa käyttää valmistajan antamia arvoja.

6 Johtopäätökset ja suositukset

6.1 Nykyiset tuotteet

Nykyisten tuotteiden vianaikainen tiedonkeräys on haasteellista varsinkin, jos tietoja halutaan tallentaa paljon. Työssä esitetyn esimerkkiratkaisun kaltaista järjestelmää on mahdollista käyttää vikojen diagnosoinnissa apuvälineenä, mikäli tarvittava tietomäärä ei ole erityisen suuri. Ratkaisussa hyödynnettiin liitäntäkortin ylimääräisiä resursseja, jonka ansiosta siitä ei aiheudu lisää kustannuksia. Esimerkkitoteutuksesta jätettiin monia ominaisuuksia pois, koska sen tarkoitus oli lähinnä osoittaa ratkaisun toimivuus ja testata, kuinka paljon tällä tavalla saataisiin tallennettua raakatietoja muistiin. Ratkaisussa tallennettiin vain kahta 16-bittistä signaalia. Todellisuudessa ratkaisulla on kuitenkin tarkoitus tallentaa huomattavasti useampia signaaleja.

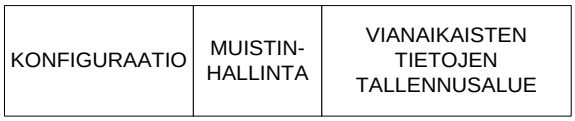
Mikäli kaikkia liitäntäkortin liittimiä ei tarvitse muuhun tarkoitukseen, on myös vianaikaisten tietojen siirtäminen ulkoiselle laitteelle mahdollista. Ulkoisen laitteen käyttäminen on suositeltavaa varsinkin, jos liitäntäkortilla on suuri riski rikkoutua vikatilanteiden aikana. Mikäli halutaan suunnitella uusi laite, on tiedonkeruuyksikkö helpoin toteuttaa mikro-ohjaimen avulla, koska mikro-ohjainten ohjelmointi on suhteellisen helppoa ja koska useissa eri mikro-ohjaimissa on integroituna monia tiedonkeruuyksikön tarvitsemia ominaisuuksia. Uutta laitetta ei ole kuitenkaan pakko suunnitella, koska myös RUSB-02:ta voidaan hieman modifioituna käyttää vianaikaisten tietojen tallennukseen. RUSB-02:n käyttämisen suurin rajoite on kuitenkin SRAM-muistin vähyys.

6.2 Lisäkehitettävää esimerkkiratkaisuun

Muistinhallinta

Ratkaisussa ei käytetty vianaikaisten tietojen tallennuksessa erillistä muistinhallintalohkoa EPCS-laitteella, vaan tiedot tallennettiin aina samaan ennalta määrättyyn osoitteeseen, joka täytyi tyhjentää ennen kuin sille alettiin tallentaa tietoa. Muistialueen tyhjennyksen jälkeen oli mahdollista tallentaa tietoja vain yhden vikatilanteen aikana, koska muuten tiedot olisivat korruptoituneet. Muistinhallintalohkon (kuva 27) avulla tiedonkeruuratkaisun olisi kuitenkin voinut toteuttaa esimerkiksi niin, että tiedot tallennetaan aina ensimmäiseen vapaaseen muistiosoitteeseen, jolloin tietojen tallennus onnistuisi, vaikka edellisen vikatilanteen tiedot olisivat vielä tallella muistipiirillä, kunhan muistissa on tilaa. Muistinhallintalohko mahdollistaisi myös aikaleiman jättämisen kunkin vikatilanteen yhteyteen, mikä helpottaisi liitäntäkortin tietojen yhdistämistä muihin vianaikaisiin

tietoihin. Muistinhallintalohko ei ole kuitenkaan välttämätön aikaleiman jättämisen kannalta.



Kuva 27: Pientä osaa EPCS:n muistiavaruudesta on mahdollista käyttää vianaikaisten tietojen tallennuksen hallinnoimiseen.

Useamman muistipiirin käyttäminen

Tässä työssä esitetyssä ratkaisussa tietoja tallennettiin vain liitäntäkortin FPGA:n konfigurointilaitteelle. Liitäntäkortilla on kuitenkin myös toinen flash-muistipiiri. Tallennukseen käytössä olevan ajan ollessa rajallinen ehdittäisiin tietoja kirjoittaa muistiin kaksinkertainen määrä, mikäli näille kummallekin muistipiirille tallennettaisiin samanaikaisesti.

Tallennettavan tiedon optimointi

Koska liipaisusignaali saattaa jo itsessään kertoa, mistä vikatilanne todennäköisesti johtuu, voitaisiin vianaikana tallennettavaa tietoa optimoida valikoimalla tallennettavia signaaleja liipaisusignaalin perusteella. Tietojen tallentamiseen käytettävissä oleva aika saattaa myös vaihdella vikatilanteen mukaan. Ratkaisu voidaan toteuttaa esimerkiksi niin, että rengaspuskuri käyttäisi kaiken vapaan tilan FPGA:n SRAM-muistilla, mutta rengaspuskurin sisällöstä valitaan EPCS:lle siirrettävä osa liipaisusignaalin perusteella. Toisaalta tallennettavissa olevaa informaatiota voitaisiin optimoida myös pakkaus-algoritmien avulla. Tietoja ei ole kuitenkaan välttämätöntä optimoida, mikäli puskurimuistin koko on riittävä ja liitäntäkortilla käytettäisiin jotain varateholähdettä, jolloin koko puskurin sisältö ehdittäisiin siirtää muistiin kaikissa tunnetuissa vikatilanteissa, kunhan liitäntäkortti ei rikkoudu.

Esimerkkitoteutuksessa tietoja tallennettiin pysyvään muistiin puskurin alusta lähtien. Tämän ongelmana on kuitenkin se, että mikäli tietoja ei ehditäkään siirtää muistiin, voivat vikahetkellä tapahtuvat signaalimuutokset jäädä tallentamatta. Toinen vaihtoehto on tallentaa tiedot puskuriin viimeinen näyte ensimmäisenä, jolloin ainakin FPGA:n viimeisimmät tapahtumat ehditään tallentaa.

Ohjauskortti

Työssä ei otettu huomioon taajuusmuuttajan ohjauskortin ominaisuuksia, vaan ohjauskortin toimintaa emuloitiin PC:llä. Liitäntäkortin vianaikaiset tapahtumat voisi olla kuitenkin

järkevää siirtää automaattisesti ohjauskortin muistiin, jonka jälkeen tietoja voitaisiin myös tarkastella ohjauspaneelilta ilman, että tietoja on pakko lukea PC:lle. Jos vianaikaisia tietoja tallennetaan useammalle alueelle EPCS-laitteelle, auttaisi EPCS:n muistinhallinta-lohko vikatietojen lukemisessa liitäntäkortilta ohjauskortille.

6.3 Tulevaisuuden tuotteet

Vianaikaisten tietojen tallennus kannattaa ottaa huomioon jo uusia tuotteita suunniteltaessa, koska tällöin on mahdollista käyttää useampia ratkaisuvaihtoehtoja kuin nykyisten tuotteiden kohdalla. Riippuen siitä, halutaanko tiedonkeräysyksikkö integroida liitäntäkorttiin tai ohjauselektroniikkaan vai halutaanko tiedonkeräystä varten käyttää erillistä laitetta, voivat toteutustavat ja käytettävät teknologiat poiketa paljonkin. Optimaalisimpaan toteutustapaan vaikuttaa myös, kuinka paljon tietoa halutaan tallentaa. Mikäli kaistanleveydet riittävät, kannattaa pyrkiä käyttämään vianaikaisten tietojen siirtämiseen myös muuhun tarkoitukseen käytössä olevia väyliä, jolloin piirkorttipinta-alaa vieviltä, ylimääräisiltä liittimiltä vältytään.

Jos tiedonkeruun halutaan toimivan pelkästään liitäntäkortin omilla komponenteilla, kannattaa rengaspuskuriin perustuvassa toteutuksessa käyttää haihtumatonta nvRAM-muistipiiriä. Jotta tietoa pystyttäisiin siirtämään tarpeeksi puskurimuistin ja FPGA:n välillä, joudutaan käyttämään yhtä tai useampaa rinnakkaisväyläistä nvRAM-piiriä, mikä vie paljon piiripinta-alaa. Ratkaisun suurin hyöty on kuitenkin se, että tietoja ei tarvitse siirtää vikatilanteen aikana erilliselle muistipiirille, vaan nvRAM-muistin sisältö on luettavissa myös virtojen katkeamisen jälkeen.

Erillistä laitetta käytettäessä sopivat tiedonsiirtoon suurihäiriöisessä taajuusmuuttaja-ympäristössä parhaiten nopeat valokuidut niiden korkean häiriösietoisuuden vuoksi. Gigabit Ethernet -tekniikalla päästään suuriin siirtonopeuksiin, joten sen toimivuutta taajuusmuuttajaympäristössä kannattaa tutkia. Langattomat lähiverkkotekniikat eivät pysty yhtä suuriin tiedonsiirtonopeuksiin kuin nopeimmat valokuitutekniikat, joten langattomien tekniikkojen käytöllä ei saavutettaisi muihin tekniikoihin verrattuna huomattavia etuja. Langattoman tiedonsiirron hyödyntäminen olisi järkevää lähinnä, mikäli sitä käytettäisiin myös muihin tarkoituksiin.

7 Yhteenveto

Työssä tutkittiin erilaisia ratkaisuja ja teknologioita vianaikaisten tietojen tallentamiseksi taajuusmuuttajan tehoasteen liitäntäkortilta. Osaa työssä käsitellyistä ratkaisuista ei ole mahdollista hyödyntää nykyisten tuotteiden kanssa, mutta nämä ratkaisut voi olla kuitenkin hyvä ottaa huomioon uusia tuotteita suunniteltaessa. Nykyisten tuotteiden kanssa on mahdollista hyödyntää lähinnä ratkaisuja, joissa tietoja tallennetaan jatkuvasti rengaspuskuriin taajuusmuuttajan ollessa käynnissä, ja kun vikatilanne havaitaan, siirretään puskurin sisältö joko ulkoiselle laitteelle tai liitäntäkortin omalle muistipiirille. Väylien siirtonopeudet ja muistien kirjoitusnopeudet aiheuttivat suurimmat rajoitukset vianaikaisten tietojen tallennukseen.

Markkinoille on tullut useita uusia teknologioita, jotka sopisivat vianaikaisen tiedon tallennukseen huomattavasti tämän työn toteutuksessa käytettyä teknologiaa paremmin. Erityisesti haihtumattomat RAM-muistit kannattaisi ottaa huomioon tulevaisuudessa vianaikaista tiedonkeruuratkaisua suunniteltaessa. Voidaan olettaa, että taajuusmuuttajissa käytetään tulevaisuudessa nopeampia valokuitulinkkejä, kuten esimerkiksi Gigabit Ethernet -kuitulinkkiä, jota käyttämällä olisi monissa tilanteissa mahdollista siirtää kaikki vian diagnosoinnin kannalta oleelliset signaalit ulkoiselle tiedonkeruuyksikölle.

Työssä toteutetussa esimerkkiratkaisussa käytettiin tietojen puskurointiin FPGA-piirin SRAM-muistia. Kun vikatilanne havaittiin, siirrettiin puskurin sisältö SPI-väylää pitkin EPCS-laitteelle. Ratkaisun vahvuutena oli, että se voitiin toteuttaa tekemällä muutoksia FPGA:n logiikkaan, eikä erillistä tiedonkeruulaitetta tai vapaita liittimiä tarvittu. Tietoja ei kuitenkaan ehditä tallentaa välttämättä vikadiagnoosin kannalta tarpeeksi, ja liitäntäkortin rikkoutuessa ei vianaikaisia tietoja ole enää kaikissa tilanteissa mahdollista lukea. Vaikka esimerkkiratkaisu suunniteltiin ensisijaisesti käytettäväksi tuotekehityksessä, voi samalla periaatteella toimivaa ratkaisua käyttää myös valmiissa tuotteissa.

Lähdeluettelo

[1] ABB. *ACS880 Firmware Manual*. 2011. [Viitattu 15.10.2012] Saatavissa: http://www05.abb.com/global/scot/scot201.nsf/veritydisplay/02398d86da12ba9dc12579a3004c2dc1/$file/FI_ACS880_primary_ctr_prg_FW_B_screen_res.pdf

[2] Salminen, J. *Prototyping platform for a general-purpose fieldbus adapter*. Diplomityö. Aalto-yliopisto. Sähkötekniikan korkeakoulu. Espoo. 2011. 80 s.

[3] Niiranen, J. *Sähkömoottorikäytön digitaalinen ohjaus.* Oy Ylioppilaskustannus/Otatieto, 1999. 379s. ISBN 951-672-270-9

[4] Myllyniemi, O. *Ohjelmoitavan porttimatriisipiirin päivittäminen taajuusmuuttajasovelluksessa*. Teknillinen Korkeakoulu. Sähkö- ja Tietoliikennetekniikan osasto. Espoo. 2009. 97 s.

[5] Viitanen, E. 2006. *Taajuusmuuttajien vikapuuanalyysi*.[Verkkodokumentti] Insinöörityö. Satakunnan ammattikorkeakoulu. Tekniikka. Pori. 2006. 59 s. [Viitattu 12.10.2012] Saatavissa: https://publications.theseus.fi/bitstream/handle/10024/1051/Viitanen_Elina.pdf

[6] Kral, C & Habetler, T. G., Zhang, W. (editor). *Fault Detection, Chapter 7: Condition Monitoring and Fault Detection of Electric Drive*. InTech. 2010. ISBN 978-953-307-037-7. DOI: 10.5772/213

[7] Sepponen, R. *EMC – periaatteet ja käytännöt*. Opetusmoniste. Aalto-Yliopisto. Sähkötekniikan korkeakoulu.

[8] Vahif, F. *Digital Design*. Wiley, 2007. 540 s. ISBN 0-470-04437-3

[9] Amara A., Frédéric A., Thomas E. *FPGA vs. ASIC for low power applications,* Microelectronics Journal, Volume 37, Issue 8, August 2006, Pages 669-677, ISSN 0026-2692, 10.1016/j.mejo.2005.11.003.

[10] Jenkins, J. H. *Designing With FPGAs and CPLDs*. PTR Prentice Hall, 1994. 273 s. ISBN 0-13-721549-5

[11] Chen, W-K. *Memory, Microprocessor and ASIC.*CRC Press, 2003. 384 s. ISBN 0-8493-1739-8

[12] Altera. *Cyclone IV Device Handbook, vol. 1. 2011.*[Viitattu 15.10.2012] Saatavissa: http://www.altera.com/literature/lit-index.html

[13] Xilinx. *7 Series Product Brief.* 2012. [Viitattu 15.10.2012] Saatavissa: www.xilinx.com/publications/prod_mktg/7-Series-Product-Brief.pdf

[14] Altera. Kotisivujen verkkokauppa. [Viitattu 15.10.2012] Saatavissa: http://www.altera.com/buy/devices/buy-devices.html

[15] Rushton, A. *VHDL for Logic Synthesis*. Wiley, 1998 (reprinted 2001). 375 s. ISBN 0-471-98325-X

[16] Altera. *Quartus II handbook.* 2011. [Viitattu 15.10.2012] Saatavissa: http://www.altera.com/literature/lit-index.html

[17] IEEE 1149.1-2001. *IEEE Standard Test Access Port and Boundary-Scan Architecture*. vol., no., pp.i-200, 2001, doi: 10.1109/IEEESTD.2001.92950

[18] IEEE 1076-2008. *IEEE Standard VHDL Language Reference Manual* (Revision of IEEE Std 1076-2002).vol., no., pp.c1-626, Jan. 26 2009, doi: 10.1109/IEEESTD.2009.4772740

[19] Sjoholm, S.; Lindh, L. *VHDL for Designers, Pearson Prentice Hall*. 1997, 473s. ISBN 0-13-473414-9

[20] Zwolinki, M. *Digital System Design With VHDL*. Pearson Prentice Hall, 2000. 323 s. ISBN 0-202-36073-2

[21] Naylor D.; Jones S. *VHDL: A Logic Synthesis Approch*. Chapman & Hall, 1997. 339 s. ISBN 0-412-61650-5

[22] Altera. *Nios II Custom Instruction User Guide.* 2011. [Viitattu 15.10.2012] Saatavissa: http://www.altera.com/literature/lit-index.html

[23] Altera. *Nios II Processor Reference Handbook.* 2011. [Viitattu 15.10.2012] Saatavissa: http://www.altera.com/literature/lit-index.html

[24] Introduction to SPI and I²C protocols. [Verkkomateriaali, viitattu 15.10.2012] Saatavilla: http://www.byteparadigm.com/kb/article/AA-00255/22/Introduction-to-SPI-and-IC-protocols.html

[25] I^2C. Kotisivu. [Viitattu 15.10.2012] http://www.i2c-bus.org/

[26] USB. Kotisivu. [Viitattu 15.10.2012] http://www.usb.org

[27] Azadeh, M. *Fiber Optic communication*. Springer, 2009. ISBN 978-1-4419-0303-7

[28] IEEE Std 802.3bd-2011. *IEEE Standard for Information technology--Telecommunications and information exchange between systems--Local and metropolitan area networks--Specific requirements Part 3: Carrier Sense Multiple Access with Collision Detection (CSMA/CD) Access Method and Physical Layer Specifications Amendment 8: MAC Control Frame for Priority-based Flow Control (Amendment to IEEE Std 802.3-2008).* vol., no., pp.1-31, Aug. 10 2011, doi: 10.1109/IEEESTD.2011.5982058

[29] Jin-Shyan L.; Yu-Wei S.; Chung-Chou S.A *Comparative Study of Wireless Protocols: Bluetooth, UWB, ZigBee, and Wi-Fi*. Industrial Electronics Society, 2007. IECON 2007. 33rd Annual Conference of the IEEE , vol., no., pp.46-51, 5-8 Nov. 2007, doi: 10.1109/IECON.2007.4460126

[30] Willig, A.; *Matheus, K.; Wolisz, A. Wireless Technology in Industrial Networks.* Proceedings of the IEEE , vol.93, no.6, pp.1130-1151, June 2005, doi: 10.1109/JPROC.2005.849717

[31] Cederström, J. *Wireless local area communication of electric drives.* Aalto-Yliopisto, Sähkötekniikan Korkeakoulu. Espoo. 2010. 86 s.

[32] Cappelletti, P., Goll, C., Piero, O., Zanoni, E. *Flash Memories.* Kluwer Academic Publishers, 1999. ISBN 0-7923-8487-3

[33] IEEE Std 1005-1998. *IEEE Standard Definitions and Characterization of Floating Gate Semiconductor Arrays*. vol., no., pp.i, 1998, doi: 10.1109/IEEESTD.1998.89425

[34] Mäntymaa, J. *Haihtumattomat tallennusmenetelmät sulautetussa anturijärjestelmässä.* [Verkkodokumentti] Diplomityö. Jyväskylän yliopisto. Tietotekniikan laitos. Jyväskylä. 2006. 71 s. [Viitattu 15.10.2012] Saatavissa: https://jyx.jyu.fi/dspace/bitstream/handle/123456789/12484/URN_NBN_fi_jyu-200746.pdf

[35] Caulfield, A.M.; Coburn, J.; Mollov, T.; De, A.; Akel, A.; Jiahua He; Jagatheesan, A.; Gupta, R.K.; Snavely, A.; Swanson, S. *Understanding the Impact of Emerging Non-Volatile Memories on High-Performance, IO-Intensive Computing*. High Performance Computing, Networking, Storage and Analysis (SC), 2010 International Conference for , vol., no., pp.1-11, 13-19 Nov. 2010, doi: 10.1109/SC.2010.56

[36] Cypress Perform. *CY14B101P - 1-Mbit (128 K × 8) Serial SPI nvSRAM with Real Time Clock Datasheet*. [Viitattu 15.10.2012] Saatavissa: http://www.cypress.com/?docID=39586

[37] Tehrani, S.; Slaughter, J.M.; Chen, E.; Durlam, M.; Shi, J.; DeHerren, M. *Progress and outlook for MRAM technology*. Magnetics, IEEE Transactions on , vol.35, no.5, pp.2814-2819, Sep 1999, doi: 10.1109/20.800991

[38] Everspin. *MR25H40 - 4Mb Serial SPI MRAM Datasheet*. 2011. [Viitattu 15.10.2012] Saatavissa: http://everspin.com/PDF/EST_MR2xH40_prod.pdf

[39] Everspin. *MR4A08B - 2M x 8 MRAM Datasheet*. 2011. [Viitattu 15.10.2012] Saatavissa: http://www.everspin.com/PDF/EST_MR4A08B_prod.pdf

[40] Kim, K. *The Future Prospect of Semiconductor Nonvolatile Memory*. Magnetics Conference, 2006. INTERMAG 2006. IEEE International , vol., no., pp.102, 8-12 May 2006, doi: 10.1109/INTMAG.2006.37560

[41] Ramtron. *FM25H20 - 2Mb Serial 3V F-RAM Memory Datasheet*. [Viitattu 15.10.2012] Saatavissa: http://www.ramtron.com/files/datasheets/FM25H20_ds.pdf

[42] Fujitsu. *MB85R4002A - FRAM 4 M Bit (256 K × 16) Memory Datasheet*. [Viitattu 15.10.2012] Saatavissa: http://www.fujitsu.com/downloads/MICRO/fme/fram/datasheet-MB85R4002A.pdf

[43] Wong, H-S. P.; Raoux, S.; Kim, S. B.; Liang, J. *Phase Change Memory*. [Verkkodokumentti] Proceedings of the IEEE. Vol. 98, No. 12, December 2010. [Viitattu 15.10.2012] Saatavissa: http://www-tecc.stanford.edu/pubs/A126_5528.pdf

[44] Micron, *128Mb: P5Q Serial PCM Datasheet*. [Viitattu 15.10.2012] Saatavissa: http://www.micron.com/~/media/Documents/Products/Data%20Sheet/PCM/p5q_32_64_128Mb_serial_pcm_ds.pdf

[45] Elkeelany, O.; Todakar, V.S. *Data concentration and archival to SD card via hardware description language*. GLOBECOM Workshops (GC Wkshps), 2011 IEEE , vol., no., pp.625-630, 5-9 Dec. 2011, doi: 10.1109/GLOCOMW.2011.6162527

[46] Kingston. Kotisivut. [Viitattu 15.10.2012] Saatavissa: http://www.kingston.com/us/flash/sd_cards#sdha1

[47] TDK. *Serial ATA 3Gbps Compatible One Package Solid State Drive eSSD Series 1GB to 4GB.* [Viitattu 15.10.2012] Saatavilla: http://www.tdk.co.jp/tefe02/ew_019_essd.pdf

[48] Sayood, K. *Introduction to Data Compression*. Morgan Kaufmann, 2006. 680 s. ISBN 978-0-12-620867-7

[49] Altera. *Serial Configuration Devices (EPCS1, EPCS4, EPCS16, EPCS64, and EPCS128) Datasheet*. 2012. [Viitattu 15.10.2012] Saatavissa: http://www.altera.com/literature/lit-index.html

LIITE A

Vikatilanteita

Koodi (heksa)	Vika	Syy	Korjaustoimet
2281	Kalibrointi	Mitattu lähtövaiheen virranmittauksen poikkeama tai lähtövaiheiden U2 ja W2 välisen virtamittauksen ero on liian suuri (arvot päivitetään virran kalibroinnin aikana).	Kokeile virran kalibrointia uudelleen (valitse *Virtamittauksen kalibrointi* parametrilla *99.13*). Jos vika jatkuu, ota yhteys ABB:n paikalliseen edustajaan.
2310	Ylivirta	Lähtövirta on ylittänyt sisäisen vikarajan.	Tarkista moottorin kuormitus. Tarkista kiihdytysajat parametriryhmästä *23 Nopeusohjeen ramppi*. Tarkista moottori ja moottorikaapeli (mukaan lukien vaiheistus ja kolmio-/tähtikytkentä). Tarkista, että parametriryhmän 99 käyttöönottotiedot vastaavat moottorin arvokilven arvoja. Tarkista, että moottorikaapelissa ei ole tehokertoimen korjauskondensaattoreita tai ylijännitesuojia. Tarkista pulssianturin kaapeli (mukaan lukien vaiheistus).
2330	Maavuoto Ohjelmoitava vikatoiminto: *31.20 Maasulku*	Taajuusmuuttaja on havainnut kuormituksen epätasapainon, joka johtuu tavallisesti moottorissa tai moottorikaapelissa olevasta maasulusta.	Tarkista, että moottorikaapelissa ei ole tehokertoimen korjauskondensaattoreita tai ylijännitesuojia. Tarkista, että moottorissa tai moottorikaapelissa ei ole maasulkua: Mittaa moottorikaapelin ja moottorin eristysvastukset. Jos maasulkua ei löydy, ota yhteys ABB:n paikalliseen edustajaan.
2340	Oikosulku	Moottorikaapelissa tai moottorissa on oikosulku.	Tarkista moottori ja moottorikaapeli. Tarkista, että moottorikaapelissa ei ole tehokertoimen korjauskondensaattoreita tai ylijännitesuojia.
2381	IGBT:n ylikuormitus	IGBT:n liitoslämpötila on liian korkea. Tämä vikailmoitus suojaa IGBT:tä, ja se voi aktivoitua moottorikaapelin oikosulun seurauksena.	Tarkista moottorikaapeli.
3130	Syötön vaihekatkos Ohjelmoitava vikatoiminto: *31.21 Syötön vaihekatkos*	Välipiirin tasajännite vaihtelee. Syynä voi olla verkkojännitevaiheen puuttuminen tai sulakkeen palaminen.	Tarkista verkkosulakkeet. Tarkista syöttövirran symmetria.
3180	Latausrele ei toimi	Latausreleestä ei ole vastaanotettu tilatietoa.	Ota yhteys ABB:n paikalliseen edustajaan.
3181	Syöttö/moott.k. ristissä Ohjelmoitava vikatoiminto: *31.23 Syöttö/moott.k. ristissä*	Väärä verkkokaapelin ja moottorikaapelin kytkentä (verkkokaapeli on ehkä kytketty taajuusmuuttajan moottoriliitäntään).	Tarkista verkkokytkennät.

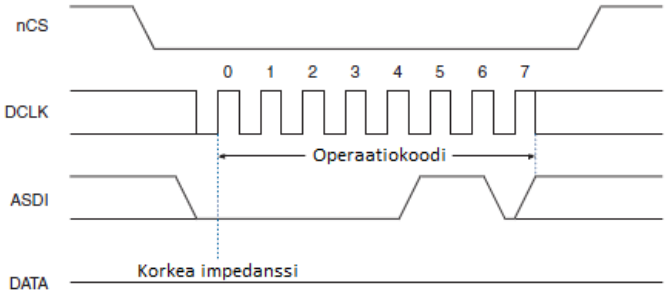
Koodi (heksa)	Vika	Syy	Korjaustoimet
3210	Välipiirin ylijännite	Välipiirin tasajännite on liian suuri.	Tarkista, että ylijännitesäätö on toiminnassa (parametri *30.30 Ylijännitesäätö*). Tarkista, että verkkojännite vastaa taajuusmuuttajan nimellistä tulojännitettä. Tarkista, esiintyykö verkkojännitteessä piikkejä. Tarkista jarrukatkoja ja -vastus (jos käytössä). Tarkista hidastusaika. Käytä vapaasti hidastuvaa pysäytystä (jos käytettävissä). Asenna taajuusmuuttajaan jarrukatkoja ja -vastus.
3220	Välipiirin alijännite	Välipiirin tasajännite ei ole riittävä, koska syöttövaihe puuttuu, sulake on palanut tai tasasuuntaussillassa on vika.	Tarkista syöttökaapelit, sulakkeet ja kytkinlaitteet.
3291	DC-jännite-ero	Tasajännite-ero rinnan kytkettyjen vaihtosuuntaajamoduulien välillä.	Ota yhteys ABB:n paikalliseen edustajaan.
3381	Lähdön vaihekatkos Ohjelmoitava vikatoiminto: *31.19 Moottorin vaihekatkos*	Moottoripiirin vika, jonka syynä on moottorin vaiheen puuttuminen (kaikkia kolmea vaihetta ei ole kytketty).	Kytke moottorikaapeli.
3385	Automaattinen vaiheistus	Automaattinen vaiheistus (katso kohta *Automaattinen vaiheistus* sivulla *34*) on epäonnistunut.	Kokeile muita automaattisen vaiheistuksen tapoja (katso parametri *21.13 Automaattinen vaiheistustila*), jos mahdollista.
4210	IGBT:n ylilämpö	Taajuusmuuttajan IGBT:n lämpötila on liian korkea.	Tarkista käyttöolosuhteet. Tarkista ilman virtaus ja puhaltimien toiminta. Tarkista, etteivät jäähdytyselementin rivat ole pölyyntyneet. Tarkista, että moottorin teho vastaa taajuusmuuttajan tehoa.
4290	Jäähdytys	Taajuusmuuttajamoduulin lämpötila on liian korkea.	Tarkista ympäristön lämpötila. Jos lämpötila ylittää 40 °C, varmista, että kuormitusvirta ei ylitä taajuusmuuttajan alennettua kuormituskapasiteettia. Lisätietoja on vastaavassa *laiteoppaassa*. Tarkista taajuusmuuttajamoduulin jäähdytysilmavirta ja puhaltimen toiminta. Tarkista, onko laitekaapin sisälle ja taajuusmuuttajamoduulin jäähdytyselementtiin kertynyt pölyä. Puhdista tarvittaessa.

LIITE B

EPCS-konfigurointilaitteen operaatioita

Write enable

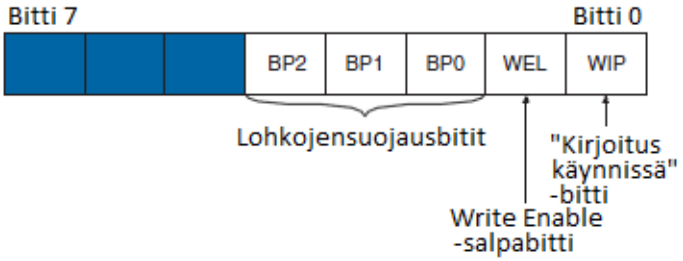
Jotta muistiin ei kirjoitettaisi tarkoituksettomasti, ovat kirjoitus- ja tyhjennysoperaatiot käynnistyksen sekä joidenkin muiden operaatioiden jälkeen estettyjä. Muuttamalla tilarekisterin bitti 1:stä eli salpabittiä *Write enable* -operaation avulla voidaan kirjoitus- ja tyhjennysoperaatiot kuitenkin sallia. Kuvassa B1 on esitetty *Write enable* -operaation ajoituskaavio.



Kuva B1: Write enable -operaatiossa ajetaan EPCS-laitteen ASDI-pinnille operaatiokoodi nCS-signaalin ollessa alhaalla. Kun operaatiokoodi on ajettu, on nCS-signaalin noustava ylös.

Read status

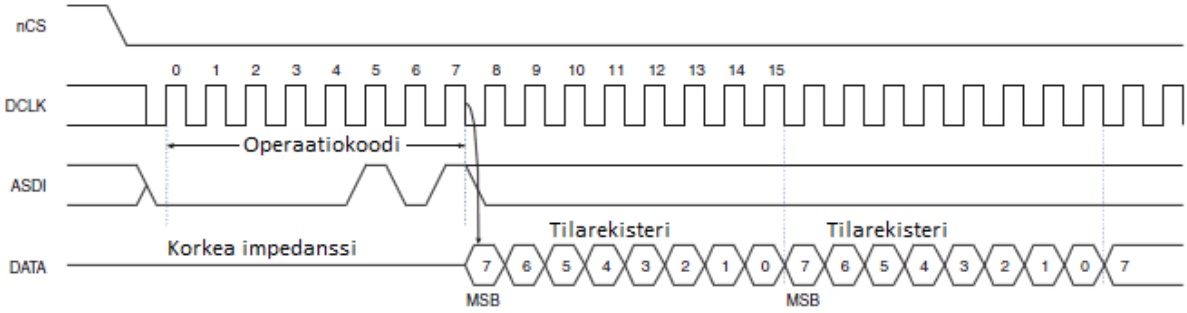
Read status -operaation avulla voidaan nähdä, missä tilassa tilarekisterin (kuva B2) bitit ovat.



Kuva B2: Tilarekisteristä voidaan tarkastaa kirjoitussuojatut lohkot, Write enable -salpabitin tila ja onko kirjoitus käynnissä.

Kun tilarekisterin 0-bitin tila on 1, on kirjoitus käynnissä, eikä uuden sivun kirjoittamista voi vielä aloittaa. *Read status* -operaatio on mahdollista suorittaa tietojen tallennuksen

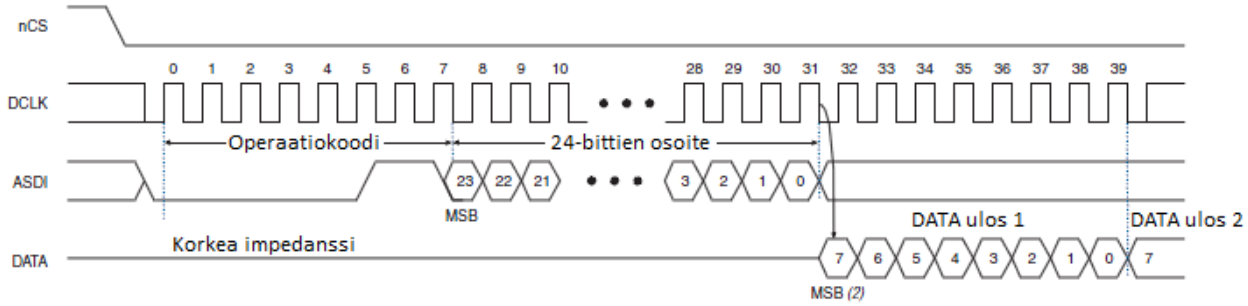
ollessa vielä käynnissä. Laskemalla, kuinka monta kellojaksoa ”kirjoitus käynnissä” -bitti on ylhäällä, voidaan selvittää yhden sivun kirjoittamiseen kuluva aika. Tilarekisterin lohkojensuojausbittien avulla voidaan nähdä, mitkä alueet muistissa on suojattuina. *Read status* -operaatio on esitetty kuvassa B3.



Kuva B3:Read status -operaatiossa ensin nCS-signaali menee alas ja operaatiokoodi ajetaan ASDI-pinnille. Operaatiokoodin ajamisen jälkeen FPGA:n DATA0-pinnille syötetään tilarekisterin tietoja niin kauan kuin nCS-signaali pysyy alhaalla.

Read bytes

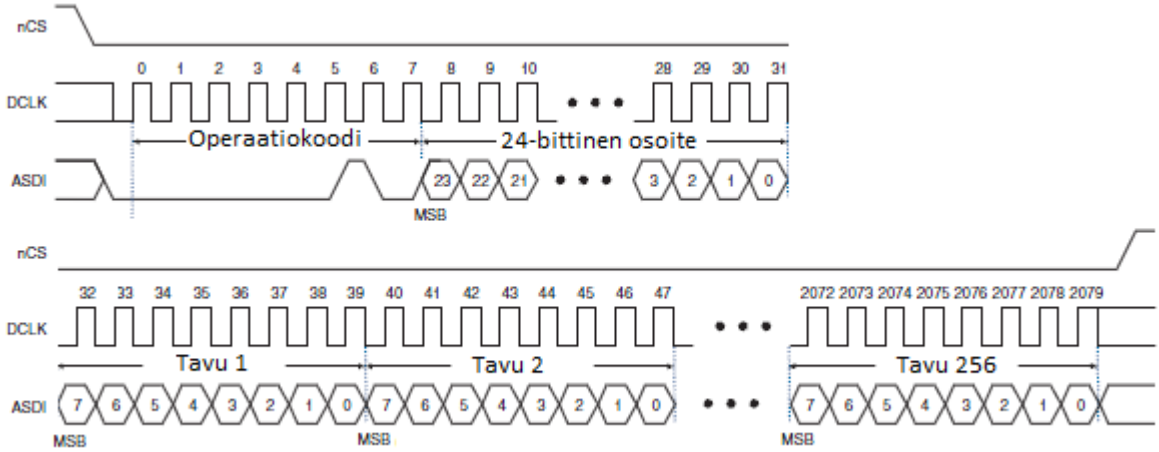
Read bytes -operaation (kuva B4) avulla voidaan lukea haluttu määrä tietoa annetusta muistiosoitteesta alkaen. Tiedot luetaan aina laskevalla kellonreunalla. Yhdellä lukuoperaatiolla on mahdollista lukea katkeamatta tietoa muistipiirin osoiteavaruuden loppuun asti.



Kuva B4: Read bytes -operaatiossa nCS-signaalien menee alas ja EPCS-laitteen ASDI-pinnille siirretään operaatiokoodi sekä osoite, josta lukeminen aloitetaan. Tämän jälkeen DATA-johdinta pitkin siirtyy tietoa FPGA:n suuntaan niin kauan kuin nCS-signaali pysyy alhaalla.

Write bytes

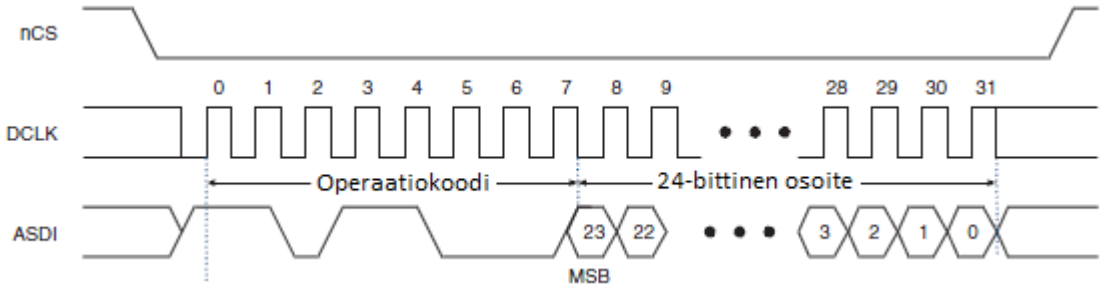
Write bytes -operaation avulla voidaan kirjoittaa sivun verran tietoa valittuun muistiosoitteeseen. Tiedot kirjoitetaan aina nousevalla kellonreunalla. Kirjoituksen aikana tilarekisterin 1. bitti saa arvon 1, jolloin ei ole mahdollista aloittaa uutta kirjoitusjaksoa. Mikäli yhden kirjoitusjakson aikana yritetään kirjoittaa enemmän kuin 256 tavua kerrallaan, niin vain viimeiset 256 tavua tallennetaan muistiin. Jos halutaan tallentaa yhtä sivua enemmän tietoa, täytyy tämä tehdä useassa erässä. Aina ennen seuraavan sivun tallennusta täytyy ajaa *Write enable* -operaatio. nCS-signaali on alhaalla, kun tietoja siirretään EPCS-piirille. Kun kaikki tallennettavat tiedot on siirretty muistipiirille, on nCS-signaalin noustava ylös, jotta tiedot alkaisivat tallentua muistipiirille pysyvästi. Vasta sitten, kun edellinen sivu saatu tallennettua, voidaan seuraavan sivun tallennus aloittaa. Yhden sivun kirjoittamiseen kuluu valmistajan esitteen mukaan tyypillisesti 1,5 ms.



Kuva B5: Write bytes -operaatiossa nCS-signaalien menee alas ja EPCS-laitteen ASDI-pinnille siirretään operaatiokoodi sekä osoite, johon halutaan kirjoittaa. Tämän jälkeen ASDI-pinnille siirretään tietoa niin kauan kuin nCS-signaali pysyy alhaalla mutta korkeintaan 256 tavua. Tietojen tallentuminen alkaa nCS-signaalin noustua ylös.

Erase sector

Erase sector -operaation avulla voidaan tyhjentää haluttu sektori, kun taas *Erase bulk* -operaatiolla voidaan tyhjentää tätä suurempia alueita kerrallaan. Tyhjennysoperaatio muuttaa kaikki tyhjennetyn sektorin bitit arvoon 1. Operaatiokoodin jälkeen EPCS16-piirille syötetään mikä tahansa tyhjennettävälle muistisektorille kuuluva osoite. Yhden sektorin tyhjennys kestää tyypillisesti 2 – 3 sekuntia. Koko EPCS16-piirin tyhjennys *Erase bulk* -operaatiolla kestää puolestaan 17 – 40 sekuntia. Muistintyhjennyksen hitauden vuoksi täytyy tallennukseen käytettävän muistialueen olla tyhjennettynä ennen vian liipaisua.

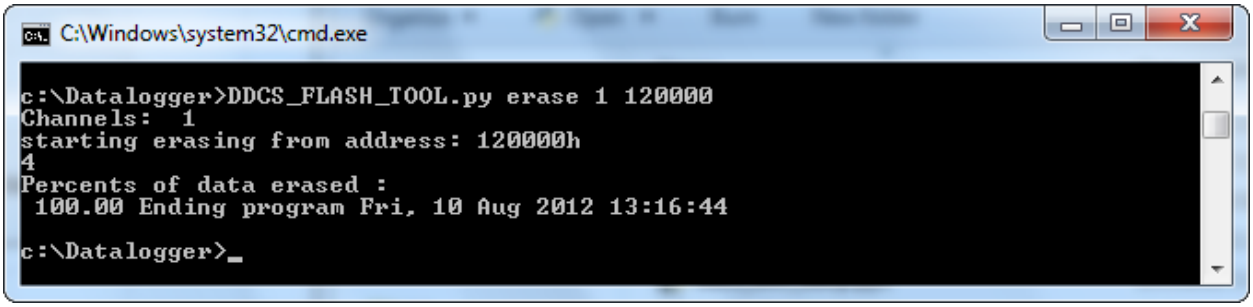


Kuva B6: Erase sector -operaatiossa nCS-signaalien menee alas ja EPCS-laitteen ASDI-pinnille siirretään operaatiokoodi sekä jokin tyhjennettävän muistisektorin osoite. nCS-signaali nousee ylös, kun osoite on syötetty.
[17]

LIITE C

Muistipiirin lukeminen ja tyhjentäminen

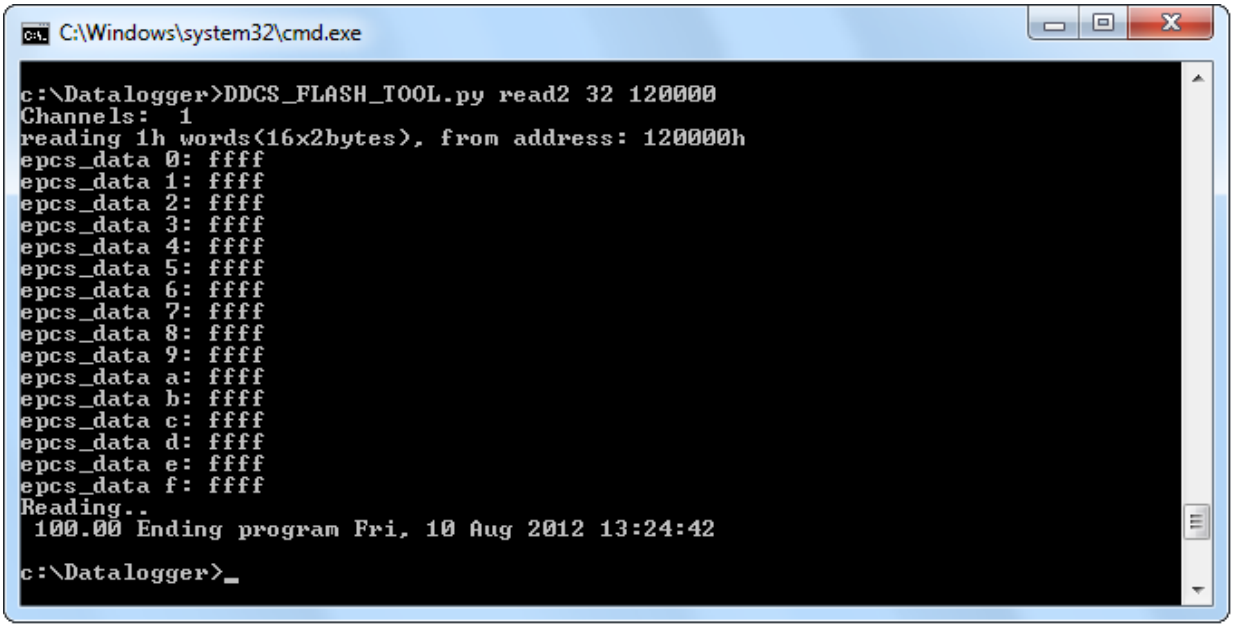
EPCS-piirin tyhjentämiseen ja lukemiseen PC:n kautta käytettiin Python-ohjelmaa DDCS_FLASH_TOOL.py. Ohjelman avulla voitiin kommunikoida liitäntäkortin kanssa komentoriviltä (kuva C1).



Kuva C1: Kuvassa yhden muistisektorintyhjentäminen komentoriviltä DDCS-linkin kautta. Tyhjennettävän muistialueen ensimmäinen osoite on 120000h.

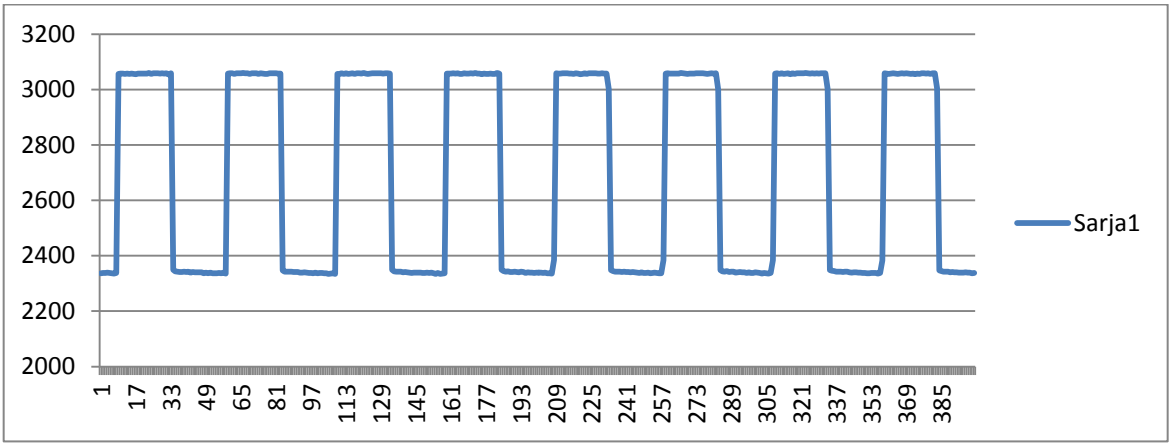
Komentoriviargumentti *erase* tyhjentää halutun määrän muistialueita annetusta osoitteesta alkaen. *Erase*-argumentin jälkeen annetaan tyhjennettävien sektoreiden määrä ja tämän jälkeen jokin ensimmäisenä tyhjennettävän muistisektorin osoite. Muistisektorit sijaitsevat osoiteavaruudessa 10000h:n välein. Tyhjennettäessä suurta muistialuetta, jos ensimmäisenä tyhjennettävä muistialueen osoite on 120000h, on seuraavan osoite 130000h.

Komentoriviargumentti *read2* lukee valitun tietomäärän EPCS:n muistista *read_wanted.csv*-tiedostoon (kuva C2). *Read2*-argumentin jälkeen annetaan seuraavana argumenttina luettavan tietomäärän koko tavuina ja tämän jälkeen osoite, josta tietoa aletaan lukea.



Kuva C2: Kuvassa luetaan 32 tavua tietoa EPCS:n muistista komentoriviltä DDCS-linkin kautta. Koska muisti oli tyhjä lukuhetkellä, ovat kaikki luetut arvot heksadesimaalina ilmaistuna f:iä.

Kuvassa C3 on esitetty Excel-ohjeman avulla mittaustuloksista piirretty kuvaaja.



Kuva C3: Liitäntäkortille A/D-muuntimelle syötettiin testausta varten suorakaideaaltoa. Selkeyden vuoksi kuvan signaalissa on vain 400 näytettä (800 tavua).