TAMPEREEN AMMATTIKORKEAKOULU
Tietotekniikan koulutusohjelma
Tietokonetekniikan suuntautumisvaihtoehto

Kivelä Anssi

DIGITAALIPIIRISUUNNITTELU ALTERA QUARTUS II 5.0 OHJELMISTOLLA

Insinöörityö, joka on jätetty opinnäytteenä tarkastettavaksi insinöörin tutkintoa varten Tampereella 08.10.2007

Työn valvoja: Yliopettaja Mauri Inha
Työn ohjaaja: Yliopettaja Mauri Inha

Tekijä: Kivelä Anssi
Työn nimi: Digitaalipiirisuunnittelu Altera Quartus II 5.0-ohjelmistolla
Päivämäärä:
Sivumäärä:
Hakusanat: Altera-ohjelmisto, laitteiston kuvauskieli, AHDL, Ohjelmoitavat logiikat
Koulutusohjelma: Tietotekniikka
Suuntautumisvaihtoehto: Tietokonetekniikka

Työn valvoja: Yliopettaja Mauri Inha
Työn ohjaaja: Yliopettaja Mauri Inha
Tampereen ammattikorkeakoulu, Tampere

Nykypäivän elektroniikkalaitteiden suunnittelussa on jouduttu siirtymään komponenttipohjaisesta suunnittelusta tietokoneella tehtävään ohjelmistopohjaiseen suunnitteluun. Suunnitteluun on tarjolla monia eri kieliä, joista ehkä tunnetuimpia ovat AHDL, VHDL ja Verilog. Pääsyy tähän siirtymiseen on se, että komponenttipohjalta tehdyt logiikkapiirit kasvavat kooltaan suhteettoman suuriksi ja fyysiset kytkennät saattaisivat olla todella massiivisia. Tähän tarkoitukseen on tässä tutkintotyössä käytetty Altera Quartus II 5.0-ohjelmistoa. Tällä ohjelmistolla voidaan suunnitella joko graafisesti logiikkapiireillä tai jollakin edellä mainituista komponentin toimintaa kuvaavalla AHDL-, VHDL- tai Verilog-kielellä.

Tämän opinnäytetyön tarkoituksena on suunnitella ja toteuttaa kaksi eri tavoilla tehtyä projektia. Toinen projekteista suunniteltiin graafisesti komponentti komponentilta ja toinen AHDL-kielellä. Graafisesti suunniteltiin 8 x 8 bittinen kertojapiiri, joka kertoo kahdeksanbittisen binääriluvun toisella kahdeksanbittisellä binääriluvulla; ja AHDL-kielellä suunniteltiin arvauspeli, jossa arvataan neljästä valosta tai ledistä, mikä niistä seuraavaksi syttyy.

Tutkintotyön tarkoituksena on luoda pohja Altera Quartus II 5.0-ohjelman käytölle toteuttamalla kaksi esimerkkiprojektia ja käyttämällä saatuja kokemuksia opetustarkoitukseen Tampereen ammattikorkeakoulussa. Tutkintotyö käsittelee Altera Quartus II 5.0 ohjelmiston perusominaisuuksia ja niiden toimintoja ja ei näin ollen selvittele kaikkia ohjelmiston tarjoamia hienouksia. Tutkintotyön tarkoituksena on valaista ohjelmiston perusominaisuuksien toimintoja.

Author: Kivelä Anssi
Title: Digital circuit designing with Altera Quartus II 5.0 software
Date:
Number of pages:
Key words: Altera-software, Altera Hardware Description Language , AHDL, Programmable Logic
Program: Information Technology
Specialisation: Computer Technology

Supervisor: Senior Lecturer Mauri Inha
Intructor: Senior Lecturer Mauri Inha
Tampere Polytechnic University, Tampere

Present-day design for electronic devices has come to the point were it is necessary to move from basic component based design to software based design for electonic circuits. Designing with software you have many lanquage choices and best known of these are AHDL, VHDL and Verilog. Main reason for this transformation of design is that design with basic component based style is going to end because circuits has come to more complicated and physical circuits very are massive. For this purpose in this thesis I used Altera Quartus II 5.0-software. With this software you can design your circuit with component based design tool or with one of these lanquages, AHDL, VHDL or Verilog.

This thesis purpose is to design two circuits using different styles of design methods. First design was made with blockdiagram tool grafically and second with AHDL design lanquage. Circuit designed with blockdiagram tool was 8 times 8 multiplier circuit, that multiplies eight bit binary number with an other eight bit binary number. The other circuit was guessing state machine were one have to guess witch led will light up next.

This thesis purpose is to create base for Altera Quartus II 5.0-software usage to desining two example projects and use the knowledge of the experience to educational purpoces in Tampere polytechnic. Thesis is dealing Altera Quartus II 5.0-softwares basic properties and does not deal with the most complicated properties what the software contains. Thesis purpose is to present softwares basic qualities and properties for the user.

ALKUSANAT

Tämä tutkintotyö on tehty Tampereen ammattikorkeakoulun tietotekniikan linjan insinöörityönä.

Suuret kiitokset tutkintotyön valvojalle Mauri Inhalle hyvästä ja mielenkiintoisesta tutkintotyön aiheesta sekä pitkäjänteisyydestä työn valmistumisen aikana.

Tampereella 8. lokakuuta 2007

_____

Anssi Kivelä

SISÄLLYSLUETTELO

KÄYTETYT MERKINNÄT JA TERMIT

CIN	carry in, muistibitti edellisestä loogisesta operaatiosta. Carry in bitti on yleensä edellisen operaation carry out bitti
COUT	carry out, loogisesta operaatiosta muodostuva muistibitti joka viedään yleensä seuraavan loogisen operaation carry in sisääntuloon
AND	Looginen operaatio JA, myös komponentti joka tekee JA operaation
OR	Looginen operaatio TAI, myös komponentti joka tekee TAI operaation
NAND	Looginen operaatio käänteinen JA, myös komponentti joka tekee käänteisen JA operaation
AHDL	Altera Hardware Description Language, laitteistonkuvauskieli
VHDL	Very High Integrated Circuits Hardware Description Language, hyvin korkeantason laitteistonkuvauskieli
MAX7000S	Suunnitteluun käytetty piiriperhe
Invertteri	Komponentti joka kääntää ykkösen nollaksi tai vastaavasti nollan ykköseksi
State Machine	Tilakone
VCC	Käyttöjännite
GND	Maapotentiaali
NOT	Kääntää ykkösen nollaksi, looginen EI-operaatio
NOR	Looginen TAI-EI-operaatio, eli käänteinen TAI-operaatio
XOR	Poissulkeva TAI -peraatio
XNOR	Poissulkeva käänteinen TAI-operaatio

1 JOHDANTO

Nykyään elektroniikkasuunnittelua tehdään yleensä joitakin markkinoilla olevia logiikkakuvauskieliä käyttämällä. Tässä tutkintotyössä perehdytään yhteen tällaiseen ohjelmistoon eli Altera Quartus II 5.0:aan. Työssä perehdytään siihen, miten ohjelmistolla toteutetaan projekti joko graafisesti komponenteilla tai vaihtoehtoisesti AHDL-kielellä ohjelmoimalla. Kummassakin tapauksessa suunnitellaan ohjelmistolla projekti siten, että tuloksena on loogisesti oikein toimiva lopputulos. Näiden kahden projektin ero on toteutus tapa, kun toinen suunnitellaan komponenteilla, niin toisessa projekti ohjelmoidaan AHDL-kielellä.

Työn lähtökohtana oli suunnitella Altera quartus II 5.0-ohjelmalla kaksi erityyppistä piiriä. Ensimmäinen suunniteltava piiri oli kertojapiiri, joka kertoo kaksi kahdeksan-bittistä binäärilukua keskenään ja antaa tuloksena binäärisen kuusitoistabittisen luvun. Toinen suunniteltava piiri oli asynkroninen tilakone. Tilakone on arvauspeli, jossa pitää arvata, mikä lamppu tai vaihtoehtoisesti ledi syttyy. Pelaajalla on neljä nappia, joista jokainen kuvaa sen vieressä olevaa lamppua. Pelaaja yrittää arvata, mikä lamppu syttyy seuraavaksi, ja painaa arvaamansa lampun nappia. Jos arvaus menee oikein arvattu lamppu syttyy ja peli jatkuu. Jos taas arvaus menee väärin niin syttyy virhe lamppu ja peli loppuu.

2 AHDL KIELIOPPIA

AHDL-kieli eli Altera Hardware Description Language on yksi käytettävistä laitteistonkuvaskielistä, jota Altera Quartus II 5.0-ohjelmisto tukee. Muita ohjelmiston laitteistonkuvauskieliä ovat VHDL ja Verilog HDL. AHDL on Alteran oma laitteistonkuvauskieli, ja toisessa projektissa suunnittelu toteutettiin tällä kielellä.

2.1 AHDL-kielen rakenne

AHDL-kielinen tiedosto tehdään ascii-tiedostoksi ja tiedostopääte on ”tdf” eli lyhenne sanoista Text Design File. Tällainen tiedosto voidaan tehdä Altera Quartus II 5.0:aan sisäänrakennetulla tekstieditorilla tai ihan millä tahansa tekstieditorilla. Subdesign section- eli alisuunnittelu- osiossa määritellään käyttöliittymä tämän osion ja muiden suunnittelussa käytettävien moduulien välillä. Tässä osassa ohjelmaa määritellään myös sisääntulot eli INPUTit ja ulostulot eli OUTPUTit. Nämä määritellyt nimet sisääntuloille ja ulostuloille pätevät vain tämän suunniteltavan tiedoston sisällä. Sisääntulot ja ulostulot voidaan myös ryhmittää yhteen käskyyn, esimerkiksi jos on monta samannimistä sisääntuloa X, niin kaikki voidaan esitellä yhdellä käskyllä. Näin kirjoitettavan koodin pituus lyhenee huomattavasti isoja ohjelmia tehtäessä.

2.2 AHDL-kielen muuttujaosio

AHDL-kielessä Variables section eli muuttuja osiossa esitellään ohjelmalle logiikka osiossa käytettävät muuttujat. Tässä kohdassa määritellään muuttujien tyypit, ”buffers” eli puskurit, ”flip-flops” eli kiikut, ”I/O” eli sisääntulot/ulostulot ja ”gates” eli portit. Tässä kohdassa ohjelmaa määritellään myös ohjelmassa käytettävät tilakoneet.

2.3 Looginen osio ja boolean yhtälöt

Looginen osio tulee heti muuttujaosion jälkeen, ja se alkaa käskyllä BEGIN ja loppuu käskyyn END. Näiden käskyjen väliin kirjoitetaan loogiset toiminnot ja boolean funktiot. Boolean funktiossa käytetään perusdigitaalitekniikasta tuttuja AND-, OR-, NOT-, XOR-, NAND-, NOR-, XNOR- ja negate-operaattoreita. Pääoperaattorit tässä ovat seuraavat.

!	NOT
&	AND
#	OR
$	XOR
!&	NAND
!#	NOR
!$	XNOR
-	negate

Taulukko 1. Pääoperaattorit.

Boolean funktioden lisäksi loogisessa osiossa voidaan käyttää ehtolauseita, totuustauluja ja toistolauseita. Totuustauluissa käytetään ykkösiä ja nollia, ja jos ”don´t care” eli tila jolla ei ole väliä onko se ykkönen vai nolla, siitä käytetään kirjainta X. Käyttöjännitteelle käytettän ilmaisua ”VCC” ja maapotentiaalille ”GND”. Totuustaulussa sallittuja ovat myös ”nodes”, ”groups” ja ”constant”.

3 ALTERA QUARTUS II 5.0-OHJELMISTON KUVAUS

Altera Quartus II 5.0-ohjelmisto on tarkoitettu digitaalipiirien suunnitteluun. Suunnittelu voidaan tehdä joko graafisesti piirtämällä kytkentä portti/komponentti kerrallaan tai vaihtoehtoisesti ohjelmoimalla jollakin ohjelmiston tukemalla laitteistonkuvauskielellä. Ohjelmisto tukee seuraavia suunnittelutiedostoja: AHDL, EDIF, SOPC, Verilog HDL, Block diagram/Schematic ja VHDL. Tässä tutkintotyössä on käytetty kahta näistä mahdollisuuksista: Block digram/Schematic eli kytkentäkaaviotiedostoa ja sitten AHDL-tiedostoa, joka on Alteran laitteistonkuvauskielitiedosto. Ensimmäinen projekti suunniteltiin ja toteutettiin kytkentäkaaviotiedostona, joka käännettiin myös AHDL-kieliseksi tiedostoksi ohjelmistolla. Toinen projekti suunniteltiin AHDL-kielellä ja siitä käännettiin kytkentäkaavioon piiri, joka sisältää AHDL-kielellä suunnitellun piirin toiminnan.

3.1 Altera Quartus II 5.0-ohjelmistolla suunnittelu

Altera Quartus II 5.0-ohjelmistolla suunnittelu aloitetaan valitsemalla uusi projekti ja määrittelemällä aluksi, millä tyylillä suunnittelu toteutetaan eli valitaan joko laitteistonkuvauskieli tai sitten vaihtoehtoisesti kytkentäkaaviototeutus. Kun suunnittelu aloitetaan, valitaan ensin ”File” valikosta ”New Project Wizard” ja tässä määritellään, mihin hakemistoon projekti tallennetaan. Projektille annetaan myös nimi, joka kopioituu myös seuraavalle riville ”top-entity”-nimeksi. Tässä on oltava tarkkana, koska ”top-entity” nimen on täsmättävä projektissa suunnittelutiedoston (design file) kanssa, muuten suunniteltua tiedostoa ei voida simuloida, kun tiedoston syntaksin tarkastukset eivät mene kääntäjästä läpi, vaikka tiedostossa ei olisi muuta vikaa.

Suunniteltaessa subdesign-tiedostoa niin jos esimerkiksi AHDL-tiedoston subdesign-rivillä nimi ei täsmää ”top level entity”-nimen kanssa, suunniteltu tiedosto ei toimi. Tästä syystä projektin nimi kannattaa pitää samana kuin ”top-level design entity”-nimi. Seuraavaksi painetaan ”next”, joka avaa ikkunan, jossa voidaan suunniteltavaan projektiin lisätä jo tehtyjä tiedostoja tai omia kirjastoja. Painettaessa taas ”next” voidaan valita projektissa käytettävä laiteperhe, jolla suunnittelu halutaan tehdä. Tässä kohdassa voidaan myös valita suoraan ”finish”, jolloin ohjelmisto käyttää vakio parametreja projektin luontiin. Painettaessa taas ”next” avautuu seuraava ikkuna, jossa voidaan lisätä EDA- eli ”Electronic Design Automation”-työkaluja, joita halutaan käyttää suunnitteluprojektissa. Painettaessa taas ”next” avautuu ikkuna, jossa näkyy, mitä olet valinnut projektisi piiriksi ja työkaluiksi. Kun tässä valikossa painetaan ”finish”, voidaan aloittaa itse suunnittelu.

3.2 Suunnitteluprojektin teko piirikaavioeditorilla

Piirikaavioeditorilla aloitat projektin kohdassa 3.1 mainitulla tavalla. Seuraavaksi valitaan ruudun oikeassa ylänurkassa olevasta ”MAX+PLUS II”-valikosta ”Gaphics editor”, jolloin ruutuun avautuu ikkuna, johon voidaan suunnitella piirikaaviona kytkentä. Piirtäminen tapahtuu ruudun keskellä olevan palkin työkaluja käyttämällä. Piirin kuvaa painamalla löytyvät kytkettävät piirit. Piirien välille tehdään johdotukset ”orthogonal node tool”-työkalulla tai tarvittaessa väyliä käyttäen ”orthogonal bus tool”-työkalua.

Johdotuksessa voidaan käyttää myös ominaisuutta ”use rubberbanding”, jolloin piirretyt kytkentäjohdot venyvät ja kutistuvat piirikaaviossa piirejä siirrettäessä. Tämä on käytännöllinen työkalu, kun aletaan muokata piirikaaviota selkeään lopulliseen muotoon. Kun kytkentä on valmis, projekti tallennetaan ja kytkentä tarkastetaan painamalla ”check & save”-nappia ikkunan yläreunassa olevasta kuvakkeesta. Tällöin ohjelma tallentaa suunnitellun kytkennän ja alkaa tarkistaa sitä. Jos kytkennässä on virheitä ohjelma ilmoittaa ne ruudun alareunassa olevassa ikkunassa varoitukset ”warning”, kriittiset varoitukset ”critical warning” ja virheet ”error”. Tästä kohdasta voidaan tarkastella, mikä kytkennässä on vikana. Jos taas kytkennästä ei löydy virheitä, niin tarkastuksen loputtua ruudulle aukeaa ilmoitus läpäistystä testistä, jossa ilmoitetaan virheiden määräksi nolla ja varoitusten määräksi nolla. On mahdollista, että tarkastus menee kääntäjästä läpi vaikka kytkennästä tulee varoituksia. Tällöin kun simuloidaan kytkentää, voi varoitus kadota, jos se ei johdu kytkentävirheestä. Varoitus voi tulla kelluvasta eli

kytkemättömästä piirin jalasta. Kun kytkennän testaus on suoritettu, mutta kytkennässä on virheitä, avautuu ikkuna, josta voi valita erilaisia testituloksia. Kun testin jälkeen avautuu ikkuna, jossa kerrotaan, kuinka monta virhettä tai varoitusta kytkennässä oli, niin siitä jatketaan painamalla ”ok”. Tämän jälkeen voidaan valita ”report”, jolloin saadaan esiin testitulokset. Testituloksia on viisi, ja näistä voidaan valita se mitä halutaan tarkastella.

Kun kytkennän toimivuus on testattu kytkentä pitää simuloida. Simulointi tapahtuu siten, että valitaan ”MAX+PLUS II” ja sieltä ”Waveform editor”, jolloin avautuu ikkuna, johon voidaan lisätä input- ja output-portit. Ikkunassa vasemmanpuoleisinta saraketta tuplaklikkaamalla avautuu ikkuna, jossa voidaan määritellä input- ja output-signaalit, jotka halutaan näkyviin Waveform-taulukkoon. Tässä kohdassa voidaan myös tuoda useamman signaalin samanaikaisesti taulukkoon painamalla ”Node Finder”-nappia. Tämä avaa ikkunan, jossa voidaan painamalla ”list”-nappia nähdä kaikki signaalit, joista voidaan valita halutut Waveform taulukkoon.

3.3 Suunnitteluprojektin teko AHDL-kielellä

Projekti aloitetaan samalla tavalla kuin piirikaavioeditorillakin, mutta siinä vaiheessa kun itse projektia aletaan tehdä, mennään valikkoon ”File”, joka on kuvaruudun vasemmassa yläreunassa. Täältä valitaan ”New” ja ruutuun aukeaa ikkuna, josta voidaan valita metodi, jolla suunnittelu tehdään. Tässä valitaan ”AHDL File” ja painetaan ”ok”. Ruutuun avautuu tekstieditori-ikkuna, johon AHDL-kielinen ohjelma kirjoitetaan. Kun ohjelma on valmis, se tallennetaan. Niin kuin piirikaavioeditorillakin tehdyn suunnittelutiedoston toiminta testataan ”save & check”-työkalulla ja testaustulokset ilmoitetaan samalla tavalla kuin piirikaavioeditorillakin tehdyssä projektissa. Ohjelma ilmoittaa tässä vaiheessa mahdolliset virheet ja varoitukset. Kun ne on saatu korjattua ja testi toteutettua voidaan suunniteltu AHDL-tiedosto simuloida. Simulointi tapahtuu samalla tavalla kuin piirikaavioeditorillakin tehdyn kytkennän simulointi. Valitaan ”MAX+PLUS II” ja sieltä ”Waveform editor”, ja jolloin avautuu ikkuna, johon voidaan hakea halutut signaalit painamalla ”Node Finder”-nappia.

3.4 Projektin simulointi

Projektin valmistuttua pitää suunniteltu projekti simuloida, jolloin voidaan todentaa, että kytkentä tai ohjelma toimii oikein. Kun ”Waveform”-taulukkoon on saatu halutut signaalit, löytyy siitä nuoli, jolla signaalit siirretään oikeanpuoleiseen ikkunaan ja valitaan ”ok”. Tällöin ”Waveform”-editoriin ilmaantuvat valitut signaalit. Nyt voidaan määritellä kullekin signaalille sen toiminta eli syötöt aikatasossa. Input-signaalit laitetaan haluttuihin tiloihin. Työkalut signaalien muunteluun löytyvät kuvaruudun vasemmasta reunasta. Ulostulot eli output-signaalit saavat jäädä määrittelemättömiksi, mikä ilmaistaan näytöllä ”XXX...XX” kirjainjonona. Kun signaaleille on saatu halutut toiminnat, pitää ne tallentaa ja kääntää ”save & compile”-työkalulla. Tätä nappia painamalla saadaan ohjelma tallentamaan tiedosto ja alkamaan automaattisesti kääntään projektia. Ruutuun avautuu ”Compiler Tool”-ikkuna jossa, kääntämisen eri vaiheet näkyvät. Kun kääntäminen on suoritettu ohjelma ilmoittaa mahdollisista virheistä ja varoituksista. Kun painat ”report” nappia saat näkyviin käännöstulokset. Kun käännös on saatu menemään kääntäjästä läpi ilman virheitä

ja varoituksia, siirrytään itse simulointivaiheeseen. Simulointeja on kahta tyyppiä ”Timing” ja ”Functional” eli ajoitus- ja toiminnallinen simulointi. Simulointityypin voi valita, kun painaa nappia ”save & simulate”. Toinen tapa aloittaa simulointi on valita ”processing” ja ”start simulation”. Kummallakin tavalla avautuu ikkuna, jossa voidaan valita simulointityyppi eli ”Timing” tai ”Functional”. Timing simulointi testaa kytkennän toiminnan piirien viiveiden mukaan, kun taas functional simulointi testaa kytkennän toiminnan ilman viiveitä. Kun simulointi on valmis, ohjelma kertoo, kuinka paljon on virheitä ja varoituksia.

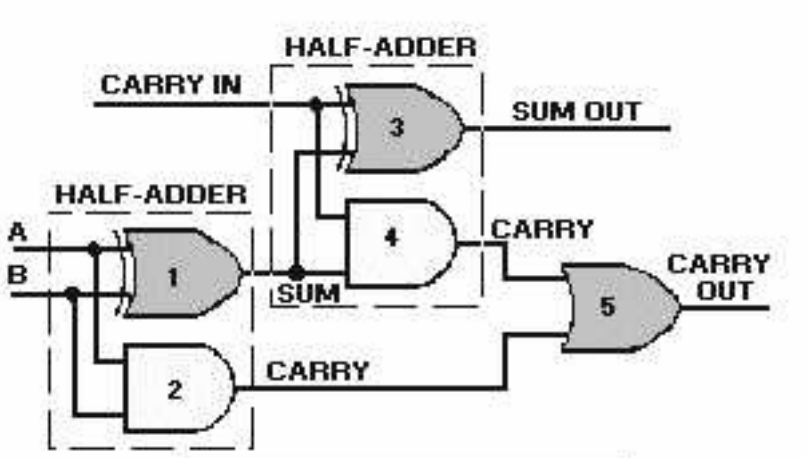
Painamalla ”report” nappia saadaan tässä ikkunassa auki, simuloinnin tulokset ikkuna. Tässä ikkunassa vasemmanpuoleisimmasta ikkunasta voi avata ikkunoita eri tulosten näkemiseksi. Keskellä näkyvät signaalit ja niiden nimet ja oikealla näkyy waveform-kuva, jos vasemmanpuoleisimmasta ikkunasta on valittu ”simulation waveforms”. Tätä waveform-kuvan näyttöaluetta voi muuttaa suuremmaksi tai pienemmäksi ”zoom”-työkalulla. Toinen tapa nähdä kytkentä ”waveform” ikkunassa on painaa ”open”-nappia ”report”-napin sijaan, tällöin avautuu ”Waveform”-editori, jossa on päivitetyt signaalit. Toinen simulointi on ”functional”-simulointi. Esimerkiksi AHDL-kielisen suunnittelutiedoston simuloiminen toiminnallisella eli functional simuloinnilla tapahtuu seuraavasti: Valitse ensin ”save & simulate” ja sitten kohtaan ”simulation mode” ”Timing” toiminnan asemessa ”Functional” paina tämän jälkeen ”Generate Functional Simulation Netlist”-nappia. Muuten simulointi tehdään samalla tavalla kuin ”Timing” simulointikin.

4. 8 X 8 BITTIÄ KERTOJAN SUUNNITTELU

Työn lähtökohtana oli suunnitella Altera quartus II-ohjelmalla kaksi eri tyyppistä piiriä. Ensimmäinen suunniteltava piiri oli kertojapiiri, joka kertoo kaksi kahdeksanbittistä binäärilukua keskenään ja antaa tuloksena binäärisen kuusitoistabittisen luvun. Tämä suunnittelu tuli tehdä piirikaavioeditorilla. Kertojapiiri perustuu kokosummainpiirien käyttöön.

4.1 Kokosummainpiiri

Kokosummain muodostuu kahdesta puolisummainpiiristä, jotka on kytketty yhteen ”OR” portilla. Kuvassa 1 (s.6) näkyy kokosummainrakenne. Ylempään puolisummaimeen kytketään ”CIN”-eli ”carry in”-signaali ja kuvassa olevan alemman puolisummainpiirin summa. Laskettavat bitit A ja B summataan ensimmäisessä puolisummainpiirissä ja kummankin puolisummainpiirin carry-biteille tehdään ”OR”-operaatio, josta muodostuu kokosummainpiirin ”COUT”-eli ”carry out”-bitti. Ylemmän puolisummainpiirin summabitti on suoraan kokosummainkytkennän summabitti. Kuvassa 1 (s.6) näkyy myös kokosummainpiirin toteuttama totuustaulu (taulukko 2).



Kuva 1. Kokosummainpiiri./5/

A	B	CARRY IN	SUM OUT	CARRY OUT
0	0	0	0	0
0	1	0	1	0
1	0	0	1	0
1	1	0	0	1
0	0	1	1	0
0	1	1	0	1
1	0	1	0	1
1	1	1	1	1

Taulukko 2. Kokosummainpiirin totuustaulu./5/

Kokosummainpiirin toteuttamat boolean-funktiot ovat seuraavat:/2/

$$S = X \oplus Y \oplus CIN$$
$$= X \cdot Y' \cdot CIN' + X' \cdot Y \cdot CIN' + X' \cdot Y' \cdot CIN + X \cdot Y \cdot CIN$$
$$COUT = X \cdot Y + X \cdot CIN + Y \cdot CIN$$

4.2 8 bittiä x 8 bittiä-kertoja

Kertojapiirin suunnittelussa valittiin Altera quartus II-ohjelmassa piiriperheeksi MAX7000S-piiriperhe, jolla kytkentä tuli suunnitella. Kertojapiiristä piti tehdä käytännön piirikaavio ja AHDL-kielinen tiedosto. Piirikaaviossa käytettiin 56 kokosummainpiiriä, 64 NAND-piiriä ja 64 invertter-piiriä. Suunnittelussa jouduttiin käyttämään 64 inverter-piiriä, koska MAX7000S-piiriperheestä ei löytynyt sopivaa AND-piiriä, jolla oltaisiin voitu korvata NAND-piirit. Ongelma syntyi, kun kytkennässä X- ja Y-parametreista piti muodostaa AND-operaatio ja sopivaa piiriä ei piiriperheestä löytynyt. Silloin kyseeseen tuli NAND-piiri, jonka perään sijoitetaan invertteri ja saadaan näin aikaiseksi AND-operaatio. Tämä muutos vaikuttaa piirin käytännön toteuttamiseen niin, että datan siirtymisaika sisääntulosta ulostuloon hidastuu ja saattaa aiheuttaa vääriä tuloksia, jos viiveet kasvavat liian suuriksi. Kuitenkaan simuloinnissa datan kulkuajasta ei tullut ongelmia. Simulointi tehtiin kahdella eri

tavalla: Ensimmäisessä simuloinnissa testattiin piirin looginen toiminta ja toisessa simuloinnissa piirin viiveet, eli se voisiko tästä piiristä valmistaa fyysisessä maailmassa toimiva kytkentä. Kumpikin simulaatio toimi odotetusti eli looginen toiminta oli niin kuin pitikin ja datan viiveet olivat sallituissa rajoissa.

4.3 8 bittiä x 8 bittiä kertojan rakenne

Kertoja koostuu seitsemästä kahdeksan kokosummainpiirin ryhmästä, joilla jokaiselle ryhmälle lasketan yksi vaihe. Ensimmäisessä vaiheessa, kun sisään menevä data, 8 bittinen X-data ja 8 bittinen Y-data saavat jonkun arvon eli binäärisen luvun, suoritetaan AND-operaatio vähiten merkitsevistä biteistä alkaen seuraavasti: X0+Y0. Tämä menee suoraan ulostuloon vähiten merkitseväksi bitiksi P0. Seuraavana suoritetaan AND-operaatio X1+Y0 ja sitten X0+Y1. Näiden tulos summataan ensimmäisessä kokosummainpiirissä, josta saatava summa menee ulostuloon P1 ja kokosummainpiirin ”COUT” eli ”carry out” menee seuraavan kokosummainpiirin ”CIN”-eli ”carry in”-sisääntuloksi. Seuraavana AND-operaationa ovat X2+Y0 ja X1+Y1, ja näiden summa menee seuraavan kokosummainvaiheen summan toiseksi komponentiksi. Näin edetään ensimmäisen vaiheen loppuun, kunnes kaikki X:n ja Y:n bittien AND-operaatiot on saatu suoritettua ja summa laskettua. Seuraavaan vaiheeseen siirryttäessä ensimmäisen vaiheen summat siirtyvät toisen vaiheen summan toisiksi tekijöiksi ensimmäistä summaa lukuun ottamatta, koska se ohjattiin jo toiseksi vähiten merkitseväksi bitiksi ulostuloon P1. Toisessa vaiheessa lasketaan ensimmäisen vaiheen summien ja X0+Y2, X1+Y2, …. X7+Y2 saakka AND-operaatioiden kesken ja toisen vaiheen ensimmäinen summa ohjataan ulostuloon P2. Sama kaava jatkuu niin kauan vaihe vaiheelta, että päästään Y:n bitti seitsemään kokosummainrakenteessa. Seitsemännessä vaiheessa muodostuvat summabitit ohjataan kaikki ulostuloihin ja eniten merkitseväksi bitiksi tulee viimeisen kokosummainpiirin ”COUT”-eli ”carry out”-bitti eli P15. Kertojanrakenne näkyy tarkemmin liitteessä 1. Liitteessä 2 kertojan piirikaavio kokonaisuudessaan.

4.4 8 bittiä x 8 bittiä kertojan suunnittelu

Kertojapiirin suunnittelun alkuvaiheessa piti etsiä tarkoitukseen sopivimmat piirit. Päädyin piirivalinnoissa seuraaviin piireihin: NAND-piiriksi valittiin ”quad nand 7437”-piiri, kokosummainpiiriksi valittiin ”full adder 8fadd” ja invertteriksi yksittäiset invertterit. 8fadd paljastui tarkemmin tarkasteltaessa Altera quartus II ohjelman makrofunktioksi. Jos haluttaisiin rakentaa oikea toimiva laite, olisi tämä ”full adder 8fadd” korvattava piiriperheestä löytyvällä 4-bittisellä ”74HC283”-piirillä. Näitä piirejä käytettäisiin siten, että 8 bittisen kokosummainoperaation aikaansaamiseksi piirejä olisi laitettava kaksi peräkkäin, jolloin saataisiin sama looginen operaatio aikaan. Ensin tehtiin liitteen 2 esittämän kertojan piirikaavio Altera quartus II ohjelmalla ja käännettiin siitä ohjelmassa olevalla megafunktiolla AHDL-kieliseksi ”8x8ahdl.tdf”-tiedostoksi ”text design file”.

4.5 AHDL-kielinenkäännöstiedosto

Työn AHDL-kielinen tiedosto saatiin kääntämällä valmiista toimivasta piirikaaviosta Altera quartus II-ohjelmasta löytyvällä megafunktiolla. Käännöstuloksena saatiin seuraava AHDL-kielinen tiedosto.

```
-- megafunction wizard: %LPM_MULT%
-- GENERATION: STANDARD
-- VERSION: WM1.0
-- MODULE: lpm_mult

-- ============================================================
-- File Name: 8x8ahdl.tdf
-- Megafunction Name(s):
--                              lpm_mult
-- ============================================================
-- ************************************************************
-- THIS IS A WIZARD-GENERATED FILE. DO NOT EDIT THIS FILE!
--
-- 5.0 Build 148 04/26/2005 SJ Full Version
-- ************************************************************


--Copyright (C) 1991-2005 Altera Corporation
--Your use of Altera Corporation's design tools, logic functions
--and other software and tools, and its AMPP partner logic
--functions, and any output files any of the foregoing
--(including device programming or simulation files), and any
--associated documentation or information are expressly subject
--to the terms and conditions of the Altera Program License
--Subscription Agreement, Altera MegaCore Function License
--Agreement, or other applicable license agreement, including,
--without limitation, that your use is for the sole purpose of
--programming logic devices manufactured by Altera and sold by
--Altera or its authorized distributors.  Please refer to the
--applicable agreement for further details.

INCLUDE "lpm_mult.inc";



SUBDESIGN 8x8ahdl
(
       dataa[7..0]  : INPUT;
       datab[7..0]  : INPUT;
       result[15..0]            : OUTPUT;
)

VARIABLE

       lpm_mult_component : lpm_mult WITH (
                                LPM_WIDTHA = 8,
                                LPM_WIDTHB = 8,
                                LPM_WIDTHP = 16,
                                LPM_WIDTHS = 1,
                                LPM_TYPE = "LPM_MULT",
                                LPM_REPRESENTATION = "UNSIGNED",
                                LPM_HINT = "MAXIMIZE_SPEED=5"
                                );

BEGIN
```

```
        result[15..0] = lpm_mult_component.result[15..0];
        lpm_mult_component.dataa[7..0] = dataa[7..0];
        lpm_mult_component.datab[7..0] = datab[7..0];
END;
```

```
-- ============================================================
-- CNX file retrieval info
-- ============================================================
-- Retrieval info: PRIVATE: USE_MULT NUMERIC "1"
-- Retrieval info: PRIVATE: WidthA NUMERIC "8"
-- Retrieval info: PRIVATE: WidthB NUMERIC "8"
-- Retrieval info: PRIVATE: WidthS NUMERIC "1"
-- Retrieval info: PRIVATE: WidthP NUMERIC "16"
-- Retrieval info: PRIVATE: OptionalSum NUMERIC "0"
-- Retrieval info: PRIVATE: AutoSizeResult NUMERIC "1"
-- Retrieval info: PRIVATE: B_isConstant NUMERIC "0"
-- Retrieval info: PRIVATE: SignedMult NUMERIC "0"
-- Retrieval info: PRIVATE: ConstantB NUMERIC "0"
-- Retrieval info: PRIVATE: ValidConstant NUMERIC "0"
-- Retrieval info: PRIVATE: Latency NUMERIC "0"
-- Retrieval info: PRIVATE: aclr NUMERIC "0"
-- Retrieval info: PRIVATE: clken NUMERIC "0"
-- Retrieval info: PRIVATE: LPM_PIPELINE NUMERIC "0"
-- Retrieval info: PRIVATE: optimize NUMERIC "0"
-- Retrieval info: CONSTANT: LPM_WIDTHA NUMERIC "8"
-- Retrieval info: CONSTANT: LPM_WIDTHB NUMERIC "8"
-- Retrieval info: CONSTANT: LPM_WIDTHP NUMERIC "16"
-- Retrieval info: CONSTANT: LPM_WIDTHS NUMERIC "1"
-- Retrieval info: CONSTANT: LPM_TYPE STRING "LPM_MULT"
-- Retrieval info: CONSTANT: LPM_REPRESENTATION STRING "UNSIGNED"
-- Retrieval info: CONSTANT: LPM_HINT STRING "MAXIMIZE_SPEED=5"
-- Retrieval info: USED_PORT: dataa 0 0 8 0 INPUT NODEFVAL dataa[7..0]
-- Retrieval info: USED_PORT: result 0 0 16 0 OUTPUT NODEFVAL result[15..0]
-- Retrieval info: USED_PORT: datab 0 0 8 0 INPUT NODEFVAL datab[7..0]
-- Retrieval info: CONNECT: @dataa 0 0 8 0 dataa 0 0 8 0
-- Retrieval info: CONNECT: result 0 0 16 0 @result 0 0 16 0
-- Retrieval info: CONNECT: @datab 0 0 8 0 datab 0 0 8 0
-- Retrieval info: LIBRARY: lpm lpm.lpm_components.all
-- Retrieval info: GEN_FILE: TYPE_NORMAL 8x8ahdl.tdf TRUE
-- Retrieval info: GEN_FILE: TYPE_NORMAL 8x8ahdl.inc TRUE
-- Retrieval info: GEN_FILE: TYPE_NORMAL 8x8ahdl.cmp FALSE
-- Retrieval info: GEN_FILE: TYPE_NORMAL 8x8ahdl.bsf FALSE
-- Retrieval info: GEN_FILE: TYPE_NORMAL 8x8ahdl_inst.tdf FALSE
-- Retrieval info: GEN_FILE: TYPE_NORMAL 8x8ahdl_waveforms.html TRUE
-- Retrieval info: GEN_FILE: TYPE_NORMAL 8x8ahdl_wave*.jpg FALSE
```

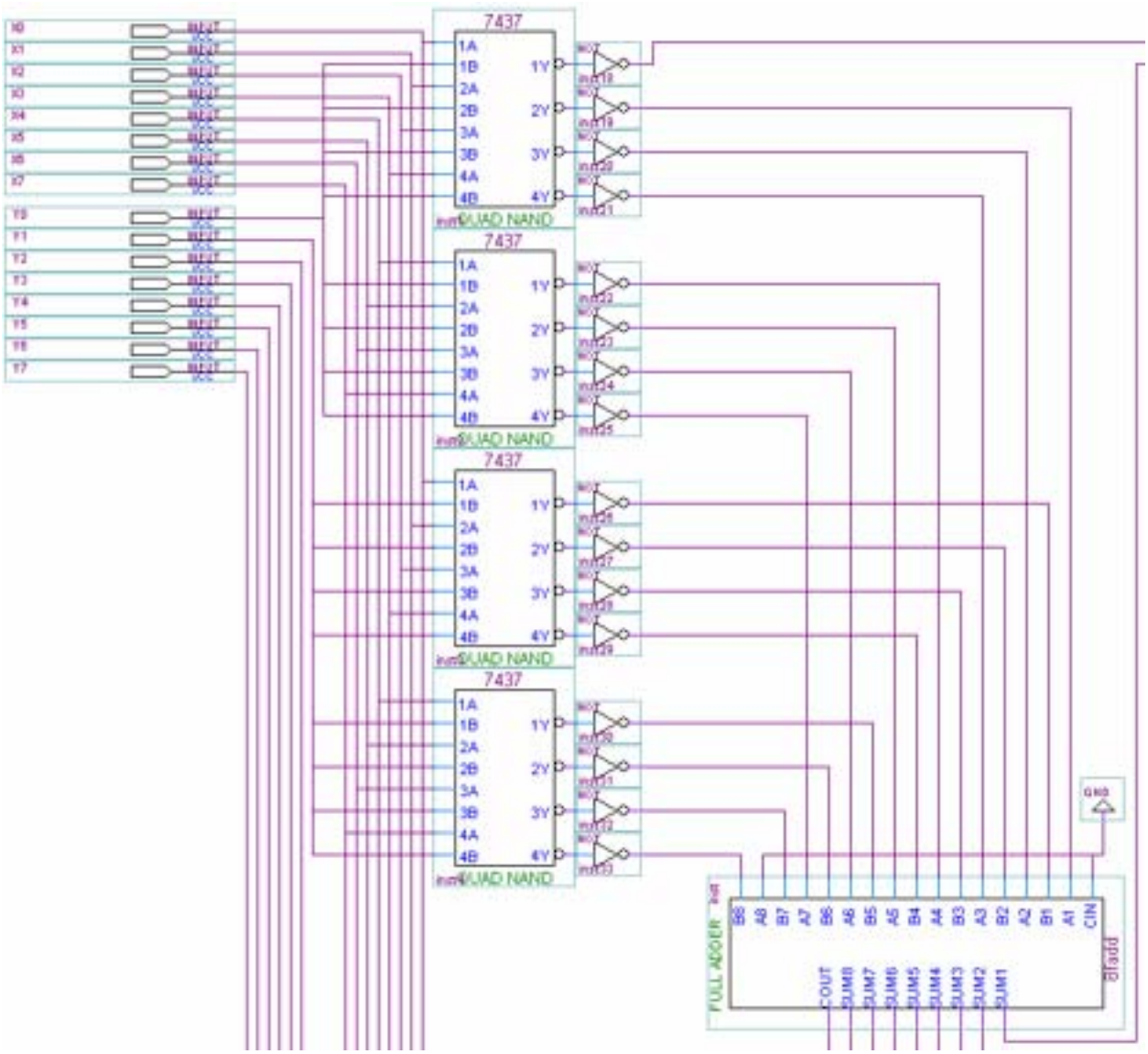
4.6 AHDL-kielisen tiedoston käskyjen selitys

Ensimmäisellä rivillä käskyllä ”INCLUDE "lpm_mult.inc";” lisätään tdf-tiedostoon lpm_mult.inc-kirjasto. Ohjelma alkaa ”SUBDESIGN 8x8ahdl”-käskyllä, jonka sulkeiden sisään tulee ”dataa[7..0]”, ”datab[7..0]”-sisääntulot ja ”result[15..0]”-ulostulo. Dataa- ja datab-sisääntulot ovat 8-bittisiä ja result-ulostulo on 16-bittinen. Muuttujat määritellään ”VARIABLE”-kohdassa. Siinä mainitaan A- ja B-datan bittimäärät ja ulostulon P bittimäärä. Siellä mainitaan myös suurin nopeus, jolla piiriä voidaan käyttää.

Ohjelmassa ”BEGIN” aloittaa ohjelman ja tämän jälkeen suoritetaan kertolasku dataa:n ja datab:n kesken ja tulos sijoitetaan result-ulostuloon ja lopuksi käsky ”END” lopettaa ohjelman. Käskyä ”INCLUDE” ennen olevat rivit ovat megafunktion itsensä kirjoittamia otsikkotietoja, jotka eivät vaikuta ohjelman toimintaan. Samoin myös käskyn ”END” jälkeen tulevat rivit ovat itse ohjemaan kuulumattomia rivejä. Rivit sisältää tietoja kytkennästä.

4.7 8 x 8-bittisen kertojan suunniteltu piirikaavio

Kytkennän ensimmäinen vaihe, jossa lasketaan ensimmäinen kokosumman vaihe ja josta tule toiseksi vähiten merkitsevä bitti ulostuloon P1. P0 eli vähiten merkitsevä bitti saatiin pelkällä AND-operaatiolla. Seuraavassa kuva ensimmäisestä kokosummain kytkennästä.

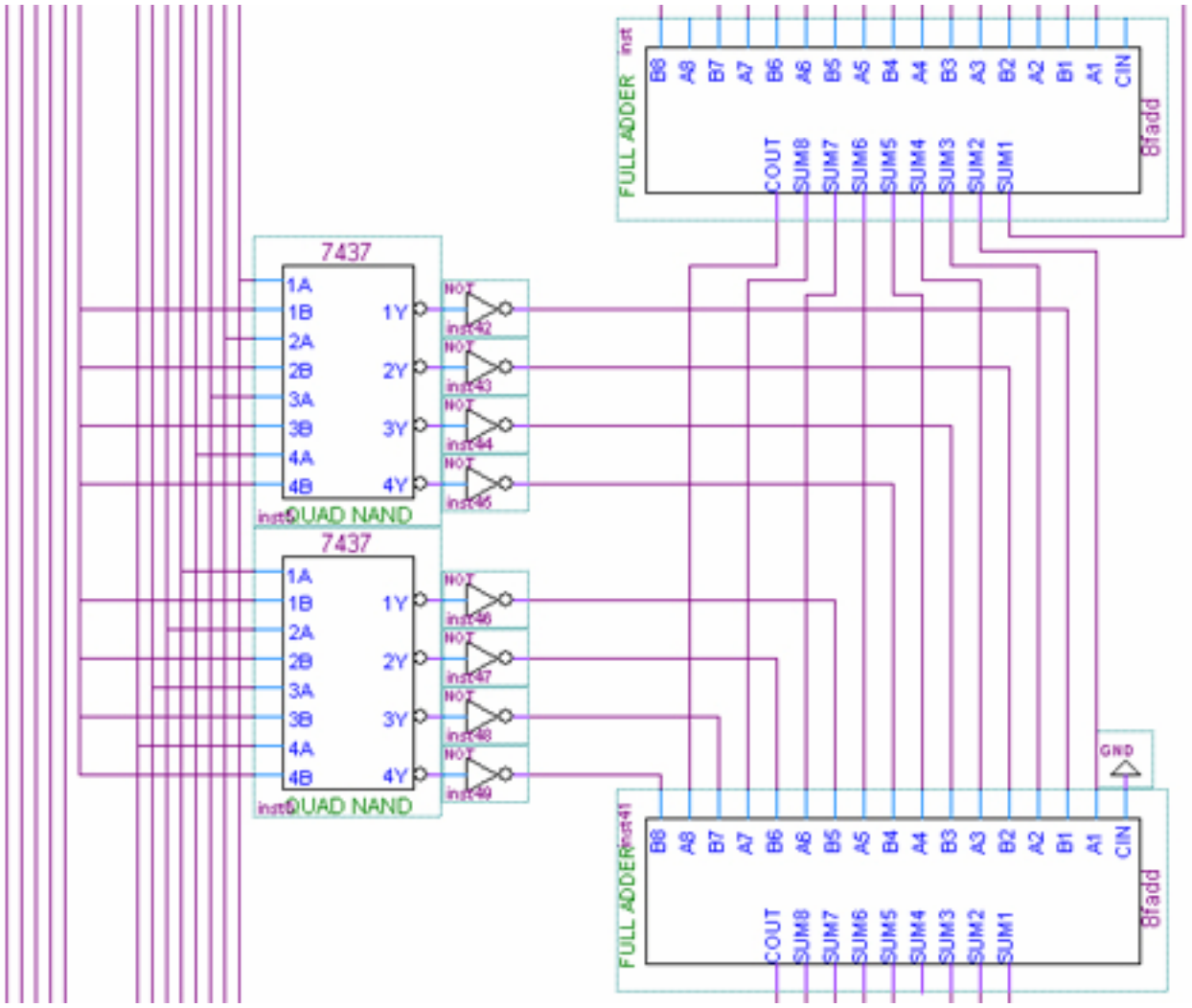


Kuva 2. Kertojapiirin ensimmäinen kokosummain osio.

Kuvassa 2 näkyy vasemmalla ylhäällä X0-X7-sisääntulot ja Y0-Y7-sisääntulot. Kuvan keskellä on neljä 7437-piiriä, joiden jokaisessa ulostulossa on invertteri. Jokaisessa 7437-piirissä on neljä NAND-piiriä. Kun piirin ulostuloon laitetaan invertteri, saadaan haluttu AND-operaatio. Kun AND-operaatiot on suoritettu, tulokset viedään kokosummaimeen ”FULL ADDER”. Kokosummain piirissä tai oikeastaan tässä tapauksessa kokosummain makrofunktiossa on kahdeksan kokosummainta. Kokosummaimessa on ”carry in”-eli ”CIN”-sisäänmeno johon tässä tilanteessa on liitetty maapotentiaali ”0V”, koska muistibittejä ei ole. Tässä kokosummaimessa syntyvät muistibitit siirtyvät kokosummaimen sisällä eivätkä näy siitä syystä ulkopuolelle. Siis jokainen ”carry out”-eli ”COUT”-bitti, jonka kahdeksan kokosummaimen kytkennässä muodostuu, on seuraavan kokosummaimen ”carry in”-eli ”CIN”-bitti. Tässä kahdeksan kokosummainoperaation jälkeen piirissä ulospäin näkyvä ”carry out”-eli ”COUT”-bitti tulee seuraavan kokosummainvaiheen eniten merkitsevän kokosumman operaation toiseksi sisääntuloksi. Tästä ensimmäisestä vaiheesta summa 1 eli ”SUM1” menee ulostuloon P1 ja summasta 2 eteenpäin kaikki summat menevät seuraavaan kokosummainvaiheeseen.

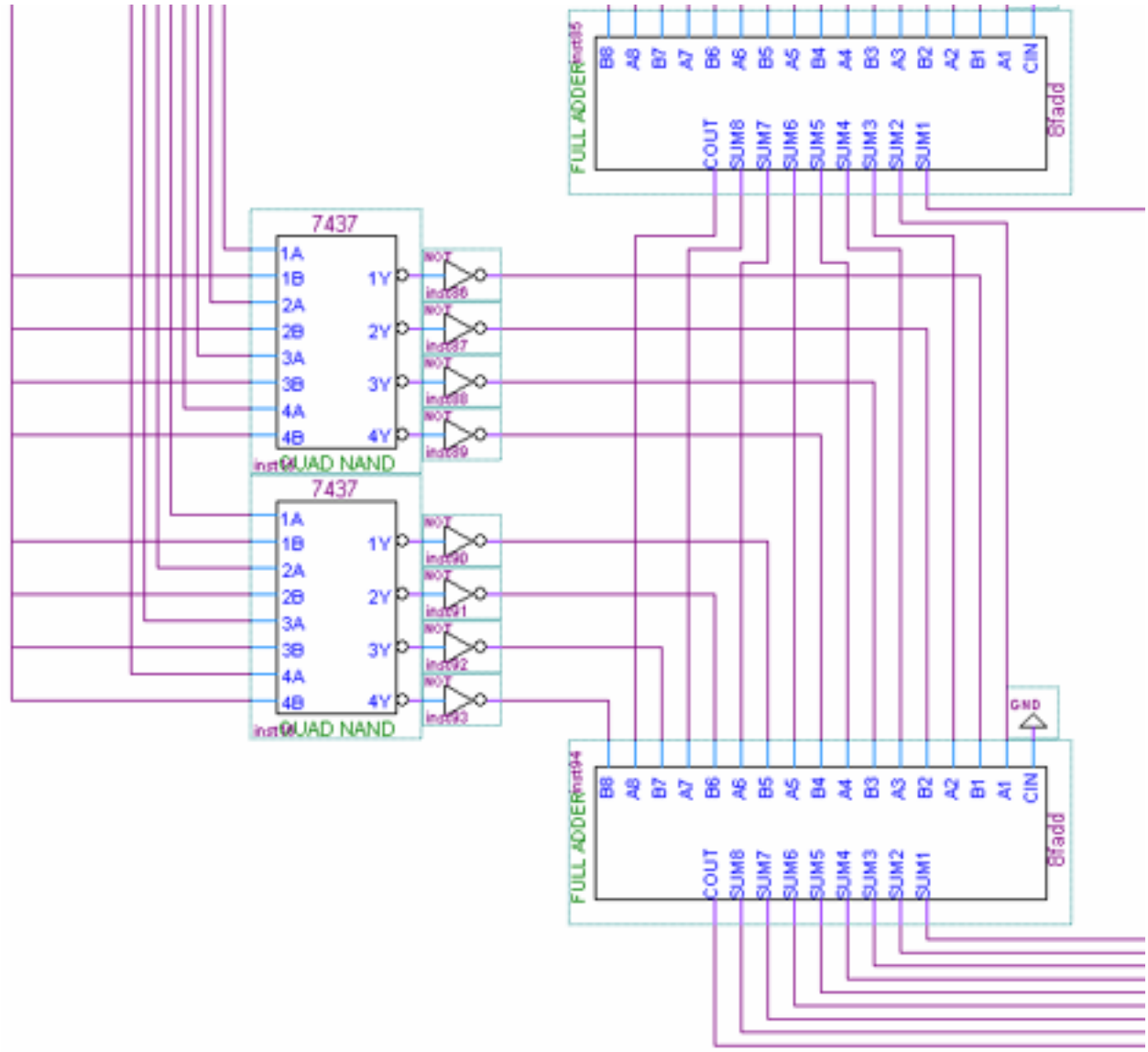
4.8 Piirikaavion toinen kokosummaus vaihe

Kuvassa 3 nähdään, miten ensimmäinen ja toinen kokosummainvaihe on kytketty toisiinsa. Eli ensimmäisen vaiheen summa 1 menee P1-ulostuloon ja toisen kokosummainvaiheen summa 1 eli alemman piirin summa 1 menee P2-ulostuloon. Tässä vaiheessa tarvitaan enää kaksi 7437-piiriä ensimmäiseksi kokosummain tekijäksi ja toiseksi tekijäksi tulevat edellisen vaiheen summat 2 - 8 ja vielä ”carry out”-eli ”COUT”-bitti. Tällä tavalla edetään aina viimeiseen vaiheeseen asti, jossa sitten kaikki loput summat siirretään ulostuloon järjestyksessä. Eli seitsemäs kokosummain on viimeinen (Kuva 4).



Kuva 3. Kytkentä ensimmäisen ja toisen kokosummausvaiheen välillä.

Kuvassa 4 näkyy toiseksi viimeinen eli ”kuudes” ja viimeinen eli ”seitsemäs” kokosummainvaihe. Tässä viimeisessä vaiheessa kaikki summat P7 - P15 tulevat järjestyksessä summa 1 ”SUM1” on P7, summa 2 on P8... ja lopuksi ”carry out”-eli ”COUT” on ulostulo P15. Liitteessä 3 vielä selventävä kuva viimeisestä seitsemännestä kokosummanvaiheesta.

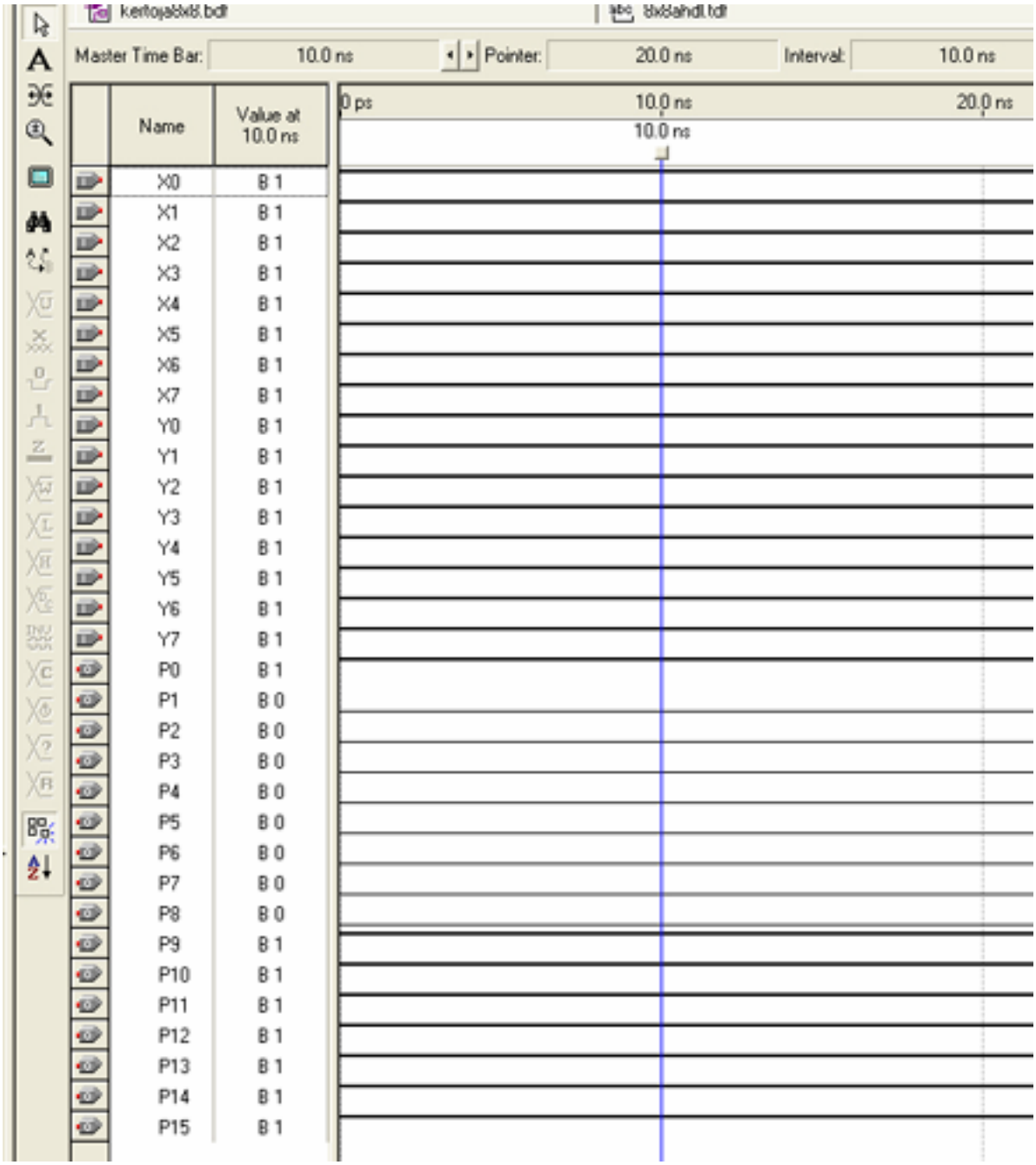


Kuva 4. Kuudennen ja seitsemännen kokosummainvaiheen välinen kytkentä ja seitsemännestä vaiheesta ulostuloihin lähtevät summat.

4.9 Kertojapiirin simulointi

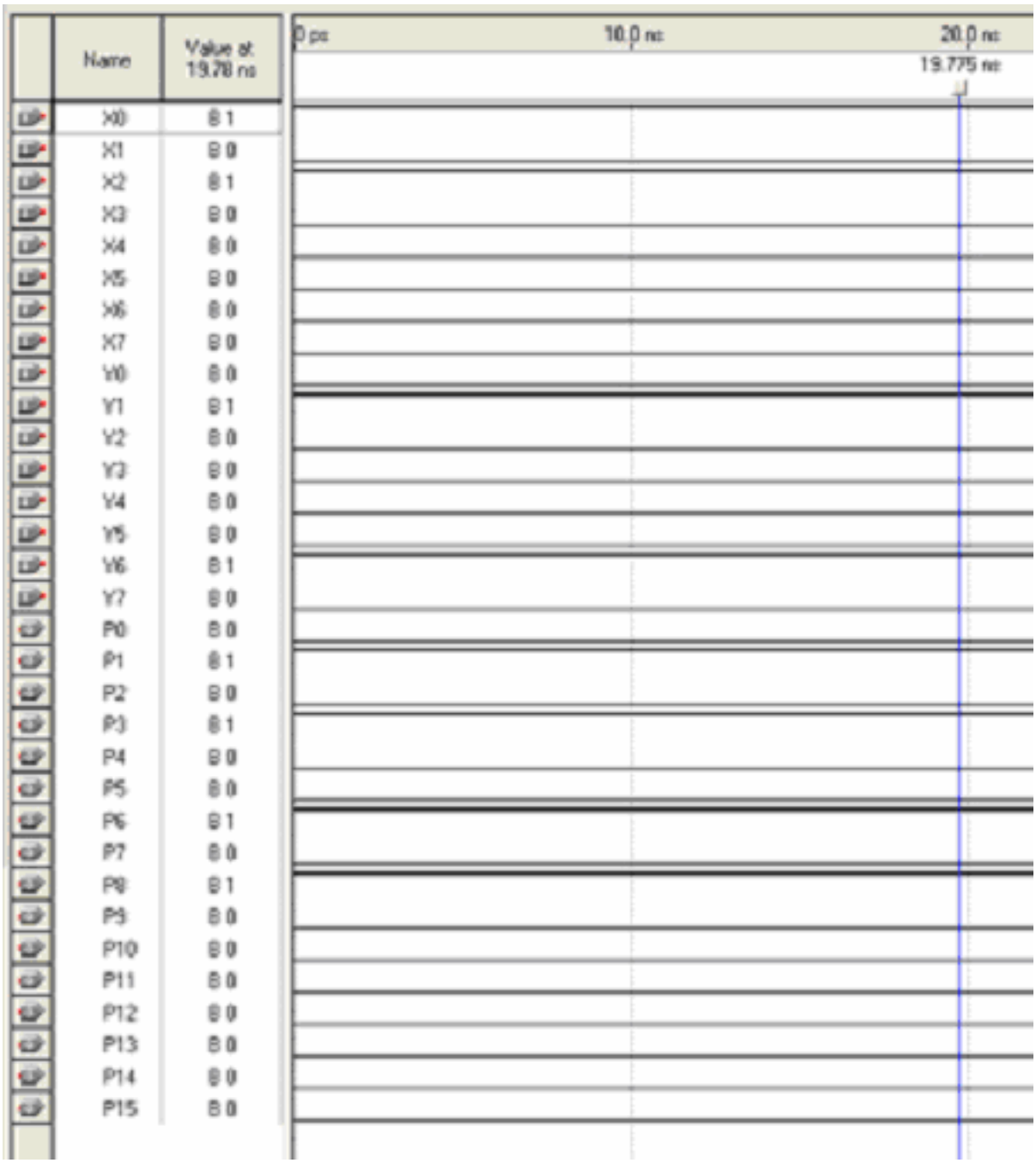
Kertojapiiristä tehdyn piirikaavion valmistumisen jälkeen piti piiri testata ja Altera quartus II-ohjelmasta löytyvällä kahdella erityyppisellä testausmenetelmällä. Ensimmäinen näistä testaa piirin loogisetominaisuudet, että toimiiko piiri loogisesti oikein ja toinen testaus menetelmä testaa, että onko piirin data-viivet sallituissa rajoissa, jotta piiri toimii myös fyysisessä maailmassa. Kummatkin näistä testeistä meni läpi eli piiri toimi toivotulla tavalla. Kun testaukset oli mennyt läpi piti piiri simuloida ja tarkastaa, että kertojapiiri kertoo kaksi kahdeksanbittistä lukua keskenään oikein. Seuraavassa kuvassa on laskettu seuraavat kahdeksanbittiset luvut yhteen. Eli X-sisääntulobitit oli 11111111 eli luku

kymmenkantaisessa lukujärjestelmässä on 255 ja Y-sisääntulobitit oli samat 11111111 eli sama 255 kymmenkantaisessa järjestelmässä. Tuloksena saatiin seuraava 16-bittinen luku 1111111000000001, joka on kymmenkantaisessa järjestelmässä luku 65025, joten laskutulos oli oikea. Kuvassa 5 näkyy simulointitulos.



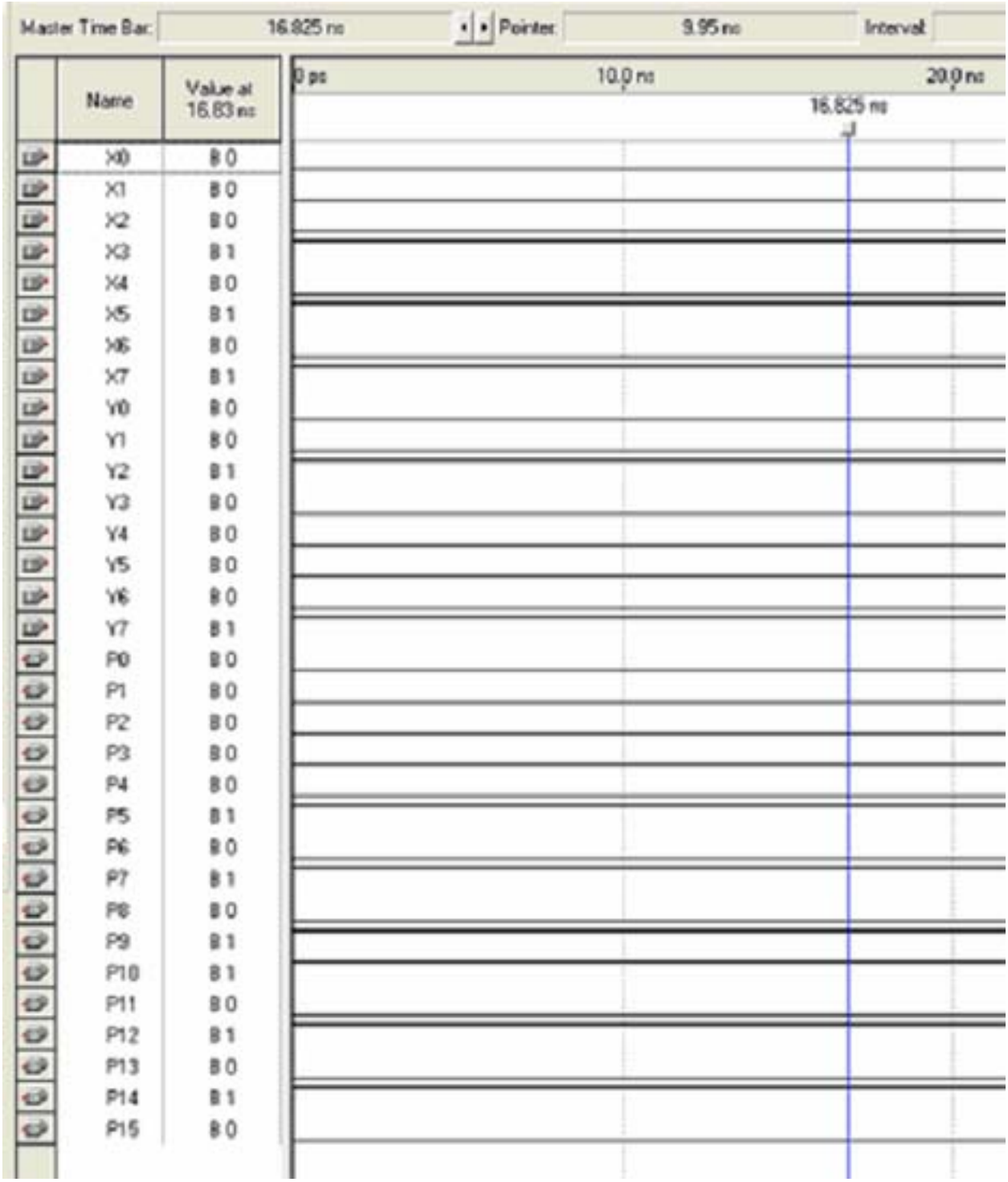
Kuva 5. Simulointitulos, jossa näkyy binäärilukujen X ja Y tulo P.

Seuraavassa kuvassa toinen simulointitulos. Kuvassa 6 kerrotaan keskenään binääriluvut 00000101 ja 01000010.Vastaukseksi saadaan binääriluku 101001010. Eli kymmenkantaisessa järjestelmässä luvut 5 ja 66 kerrotaan keskenään ja vastaukseksi saadaan 330. Eli tässäkin tapauksessa kytkentä toimii oikein.



Kuva 6. Kuvassa toinen simulointi tulos, binäärilukujen X ja Y tulo P.

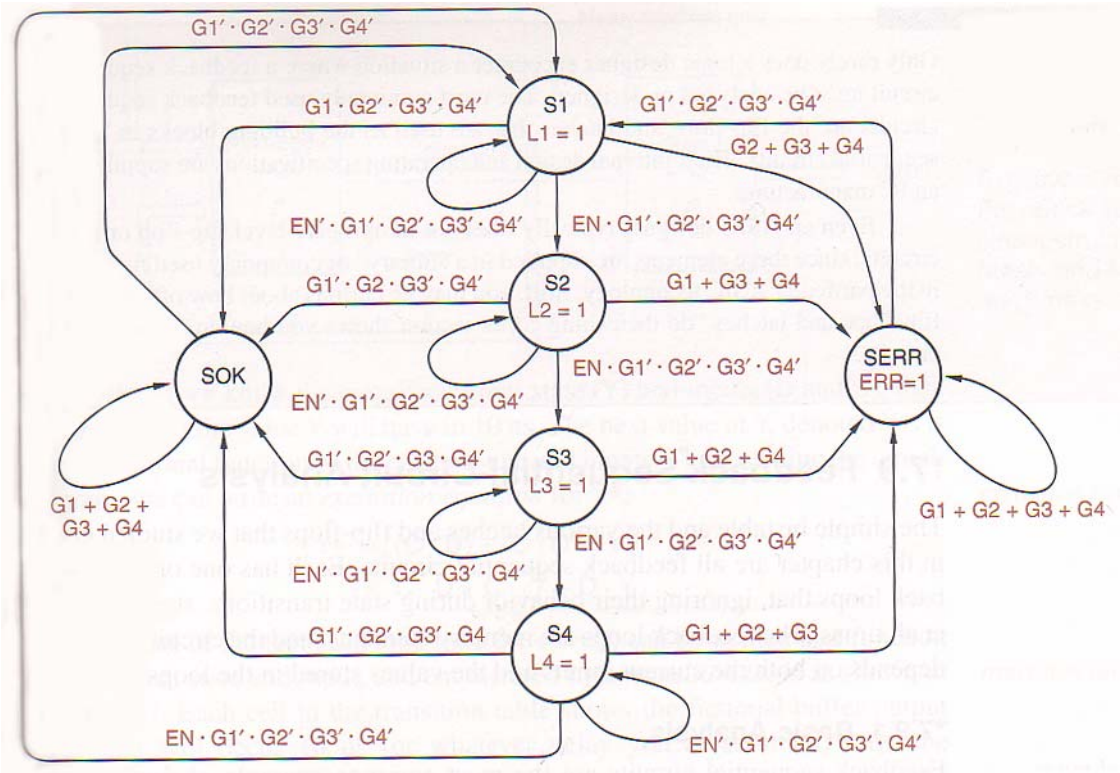
Kolmannessa simuloinnissa, kuvassa 7, kerrottiin keskenään binääriluvut 10101000 ja 10000100. Tuloksena saatiin binääriluku 0101011010100000. Kymmenkantaisella järjestelmällä laskettuna tämä on seuraavaa 168 kertaa 132 ja tulos on 22176. Tämäkin laskutoimitus oli oikein eli kytkentä toimii odotetusti.



Kuva 7. Kuvassa kolmas simulointi tulos.

5. ARVAUSPELI

Arvauspeli suunniteltiin Altera quartus II-ohjelmalla. Arvauspeli toteutettiin AHDL-kielellä tilakonesuunnitteluna ”quessing state machine”. Tilakoneessa on kuusi tilaa ja tilat ovat S1, S2, S3, S4, SOK ja SERR. Tila S1 kuvaa lamppua L1, S2 lamppua L2 jne. L4:jään asti. SERR tila kuvaa lamppu L5 ja tilalla SOK ei ole lamppua lainkaan, koska lamput L1-L4 kuvaavat oikein mennyttä arvausta. Tilakone toimii 4 Hz taajuudella, joten pelaajalla on juuri aikaa arvata lamppu, joka syttyy. Alkuperäistä suunnitelmaa muutin neljän lampun pelistä viiden lampun ratkaisuun sen takia, että pelaajalla on mahdollisuus nähdä milloin hän tekee virheen. Tähän tilakoneeseen olisi vielä mahdollista tehdä ”reset” toiminto, joka toimisi aina, kun pelaaja arvaa väärin, jolloin voitaisiin nollata tilanne aina jokaisen virheen jälkeen ja aloittaa peli alusta. Virheen merkkinä voidaan käyttää äänimerkkiä tai lamppua. Pelissä virheellinen arvaus johtaa tilaan SERR, jos väärää nappia painetaan eli G:n arvo ei vastaa syttyvän lampun numeroa seuraavalla kellopulssilla niin virhevalo syttyy. Kun arvaus on tehty virhevalo palaa eli virhetila pitää arvonsa yhden tai useamman kellopulssinajan, kunnes G:n arvo on muuttunut päinvastaiseksi. Tämän jälkeen peli jatkuu. Kuvassa 8 nähdään tilakonekaavio ja G:n arvot millä siirrytään tilasta toiseen ja millä pysytään samassa tilassa.



Kuva 8. Kuvassa arvauspelin tilakonekaavio./3/

5.1 AHDL-kielinen suunniteltu tiedosto

Seuraavassa näkyy suunniteltu AHDL-kielinen tiedosto arvauspelitilakoneelle.

```
SUBDESIGN Game6
(
	clk			:input;
	reset			:input;
	G1			:input;
	G2			:input;
	G3			:input;
	G4			:input;
	L[5..1]			:output;
)
VARIABLE
	ss: machine of bits (L[5..1])
	with states(
	s1 = B"10000",
	s2 = B"01000",
	s3 = B"00100",
	s4 = B"00010",
	sok = B"00000",
	serr = B"00001");
BEGIN
	L1= !L1 & !L2 & !L3 &  L4 & !L5 & (!G1 & !G2 & !G3 & !G4)
	 # !L1 & !L2 & !L3 & !L4 & !L5 & (!G1 & !G2 & !G3 & !G4)
	 # !L1 & !L2 & !L3 & !L4 &  L5 & (!G1 & !G2 & !G3 & !G4);
	L2=  L1 & !L2 & !L3 & !L4 & !L5 & (!G1 & !G2 & !G3 & !G4);
	L3= !L1 &  L2 & !L3 & !L4 & !L5 & (!G1 & !G2 & !G3 & !G4);
	L4= !L1 & !L2 &  L3 & !L4 & !L5 & (!G1 & !G2 & !G3 & !G4);
	L5= L1 & (G2 # G3 # G4)
	 # !L1 &  L2 & (G1 # G3 # G4)
	 # !L1 & !L2 &  L3 & (G1 # G2 # G4)
	 # !L1 & !L2 & !L3 &  L4 & (G1 # G2 # G3)
	 # !L1 & !L2 & !L3 & !L4 &  L5 & (G1 & G2 & G3 & G4);


ss.clk = clk;
ss.reset = reset;

TABLE
	ss,		G1,G2,G3,G4		=>ss;
	s1,		0, 0, 0, 0		=>s2;
	s1,		1, 0, 0, 0		=>sok;
	s1,		X, 1, 1, 1		=>serr;
	s2,		0, 0, 0, 0		=>s3;
	s2,		0, 1, 0, 0		=>sok;
	s2,		1, X, 1, 1		=>serr;
	s3,		0, 0, 0, 0		=>s4;
	s3,		0, 0, 1, 0		=>sok;
	s3,		1, 1, X, 1		=>serr;
	s4,		0, 0, 0, 0		=>s1;
	s4,		0, 0, 0, 1		=>sok;
	s4,		1, 1, 1, X		=>serr;
	sok,		1, 1, 1, 1		=>sok;
```

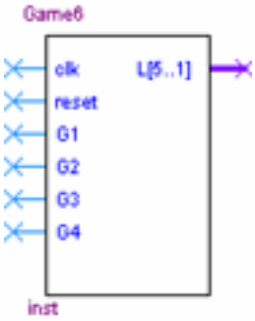
```
        sok,        0, 0, 0, 0          =>s1;
        serr,       1, 1, 1, 1          =>serr;
        serr,       0, 0, 0, 0          =>s1;


END TABLE;

END;
```

5.2 AHDL-kielisentiedoston käskyjen selitykset

Ohjelma alkaa käskyllä ” SUBDESIGN Game6 ” Game6 on projektin nimi ja sen pitää olla sama, mikä projektillakin muussa tapauksessa kääntäjä ilmoittaa virhettä ”top level entity”. Sulkeiden sisään merkitään ohjelmassa tarvittavat sisääntulot ja ulostulot (input / output). Seuraavassa kohdassa ”VARIABLE” merkitään tilakoneen bitit ”L1-L5” ja tilat ”s1, s2, s3, s4, sok ja serr” käskyllä ”ss: machine of bits (L[5..1]) with states(s1 = B"10000", s2 = B"01000", s3 = B"00100", s4 = B"00010", sok = B"00000", serr = B"00001");” Tässä on myös annettu tiloille binääriset vastaavuusarvot eli tilalle ”s1” on annettu binääriarvo ”10000” ja samoin muille tiloille. Käsky, jolla binääriarvo annetaan on ” s1=B”10000” ”. Käskyn ”BEGIN” jälkeen ilmoitetaan tilakoneessa tarvittavat booleanfunktiot ulostuloille ”L1-L5”, tilakoneen kello ”ss.clk = clk;” ja tilakoneen reset ”ss.reset = reset;”. Seuraavaksi tilakoneelle kerrotaan sen siirtymätaulu käskyllä ”TABLE” ja ”END TABLE;”. Tässä kohdassa kerrotaan millä sisääntulojen G1-G4 arvoilla siirrytään kustakin tilasta seuraavaan. Arvo 1 taulussa merkitsee, että kyseinen nappi on painettuna ja arvolla 0 nappia ei paineta. Merkintä X taas tarkoittaa, että kyseisellä G:n arvolla ei tässä tapauksessa ole merkitystä. Ohjelma päättyy käskyyn ”END;”.

5.3 Arvauspelin kytkentä

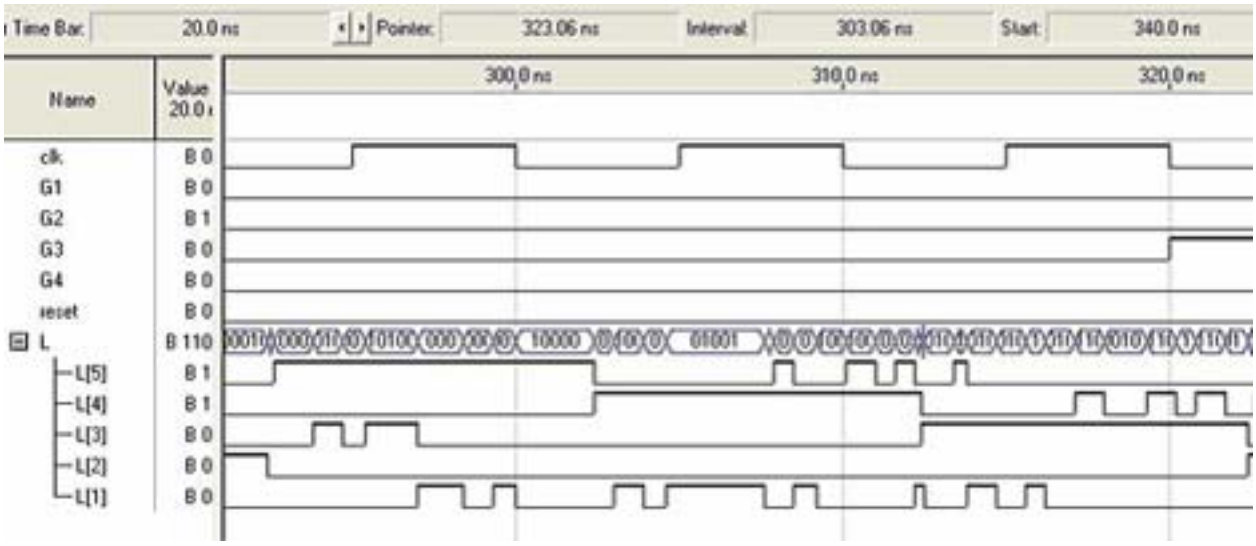
Arvauspelin AHDL-kielellä suunniteltu tiedosto käännettiin Altera Quartus II 5.0-ohjelmistolla kytkentäkaavioon sopivaksi piiriksi. Käytännössä ohjelma tekee suunnitellusta ”tdf”-tiedostosta graafisen piirin, joka sisältää suunnitellun ohjelman loogiset ominaisuudet. Tiedoston muoto on ”bsf”-eli ”Quartus II block symbol file”. Tämä megafunktiolla komponentiksi käännetty ohjelma voidaan tämän jälkeen liittää vaikka osaksi jotain suurempaa kytkentää. Seuraavassa kuva käännetystä piiristä, kuva 9.



Kuva 9. AHDL-kielisestä tiedostosta käännetty komponentti.

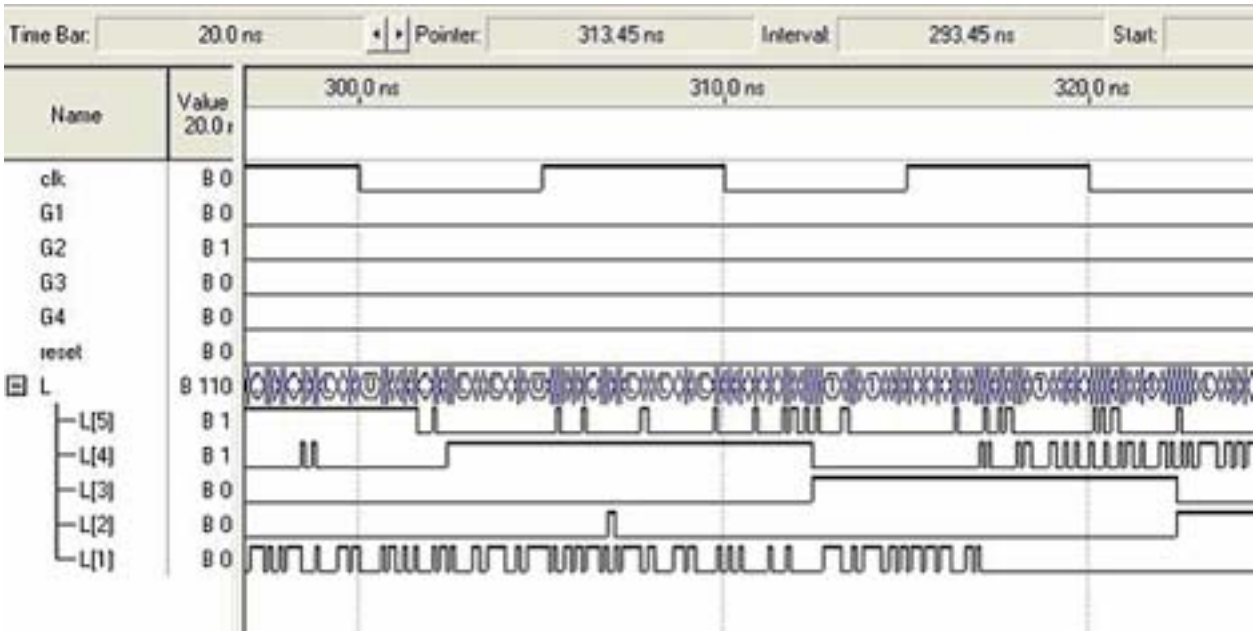
5.4 Arvauspelin simulointi

Arvauspelin tilakoneen simulointi oli ensinnäkin hankalaa siinä suhteessa, että peli toimii näennäisesti satunnaisella tavalla. Tilakone on suunniteltu sellaiseksi, että käytännössä ihmisen pitää arvata, mikä lamppu syttyy seuraavana ja painaa sitä lamppua kuvaavaa nappia. Kun arvaus menee oikein, arvattu lamppu syttyy ja pelaajan pitää arvata seuraava lamppu. Kun arvaus menee väärin, syttyy error-valo virheen merkiksi. Simulointi oli tässä tapauksessa vaikeaa, kun simuloinnissa ei voinut järjestää varmasti oikein menevää arvausta ja varmasti väärin arvausta. Simuloinnissa kävi kuitenkin ilmi se, että mikäli mitään ei arvata eli mitään nappia ei paineta, lamppujen syttymis- ja sammumissekvenssi tihenee eli palamis- ja sammumisaika lampulla lyhenee. Seuraavissa kuvissa pari simulointi tulosta.



Kuva 10. Ensimmäinen simulointitulos.

Toinen simulointitulos, jossa sisääntulojen G1-G4 tilanvaihteluita on jatkettu pidemmälle, kuin ensimmäisessä simuloinnissa.



Kuva 11. Toinen simulointitulos.

Kun verrataan simuloinnin 1 (kuva 10) ja 2 (kuva 11) lamppujen L1-L5 tilanvaihteluita samalta kohdalta ja samalla aikaväliltä nähdään, että ensimmäisessä simuloinnissa lamppujen tilanvaihtelu on huomattavasti tiheämpää, kuin toisessa simulointi tuloksessa.

6 YHTEENVETO

Nykyään logiikkapiirien suunnittelussa käytetään yhä useammin, jotain laitteistonkuvauskieltä. Syynä tähän on se, että perus logiikkakomponenteilla suunnittelu on tullut tiensä päähän ja logiikkapiirit ovat monimutkaistuneet. Kooltaan piirit ovat kasvaneet sellaisiksi, että ei ole enää käytännöllistä suunnitella komponentti komponentilta. Laitteiston kuvauskielet helpottavat ja nopeuttavat suunnitteluprosessia. Toisaalta nykysuunnittelijan pitää opetella ja hallita erilaisia laitteistonkuvauskieliä. Tyypillisiä nykyään käytettyjä laitteistonkuvauskieliä ovat AHDL, VHDL ja Verilog.

Altera Quartus II 5.0-ohjelmisto on monipuolinen ohjelmisto laitteistonkuvauskielillä ja piirikaavioeditorilla suunnitteluun. Ohjelmistolla suunnittelu voidaan toteuttaa ohjelmoimalla, jollakin laitteistonkuvauskielellä tai vaihtoehtoisesti graafisesti piirikaavioeditorilla. Alkukankeuden jälkeen suunnitteluohjelmiston toiminnat on helppo omaksua. Ohjelmisto on looginen ja soveltuu myös hyvin vasta-alkajan työkaluksi. Ohjelmiston käyttö on helppoa, kunhan tutustuu muutamaan malliprojektiin. Ohjelmiston toiminnan opetteluun löytyy apua ohjelmiston omasta ”help”-tietokannasta ja myös ohjelmiston valmistajan internet-sivuilta.

Esimerkkiprojekteja suunniteltaessa suurimmaksi vaikeudeksi osoittautui AHDL-kielellä suunniteltaessa ohjelmointikielen syntaksin opettelu. Ohjekirja, joka oli tarkoitettu ohjelmistolle oli käytössä kulunut siihen kuntoon, että tärkeimmät sivut olivat siitä hukkuneet, joten jouduin ensialkuun etsimään AHDL-kielen syntaksiin perehtyvää materiaalia internetistä. Kun tämä asia oli saatu kuntoon ohjelmoiminen oli kohtalaisen helppoa.

Piirikaavioeditorilla tehty suunnittelu onnistui helpommin. Tässä suunnittelumuodossa tarvitsi vain valita piiriperhe, jolla suunnittelu tehdään ja alkaa rakentamaan kytkentää piireistä. Piirikaavioeditorilla tehty kytkentä muodostui erittäin suureksi komponenttien määrän takia, mutta ohjelmallisesti toteutettuna kytkentä on vain muutaman rivin mittainen. Tässä voi huomata käytännössä, kuinka suuri apu on suunnitella digitaalipiiri ohjelmoimalla.

Kaiken kaikkiaan Altera Quartus II 5.0-ohjelmisto on erittäin hyvä työkalu digitaalipiirien suunnitteluun ja voin lämpimästi suositella sitä myös haastavampiin projekteihin.

LÄHDELUETTELO

Painetut lähteet

1. John F. Wakerly, Digital Design Principles and Practices Fourth Edition

2. Funktiot lähteestä John F. Wakerly; Digital Design Principles and Practices Fourth Edition 2006.sivulta 475.

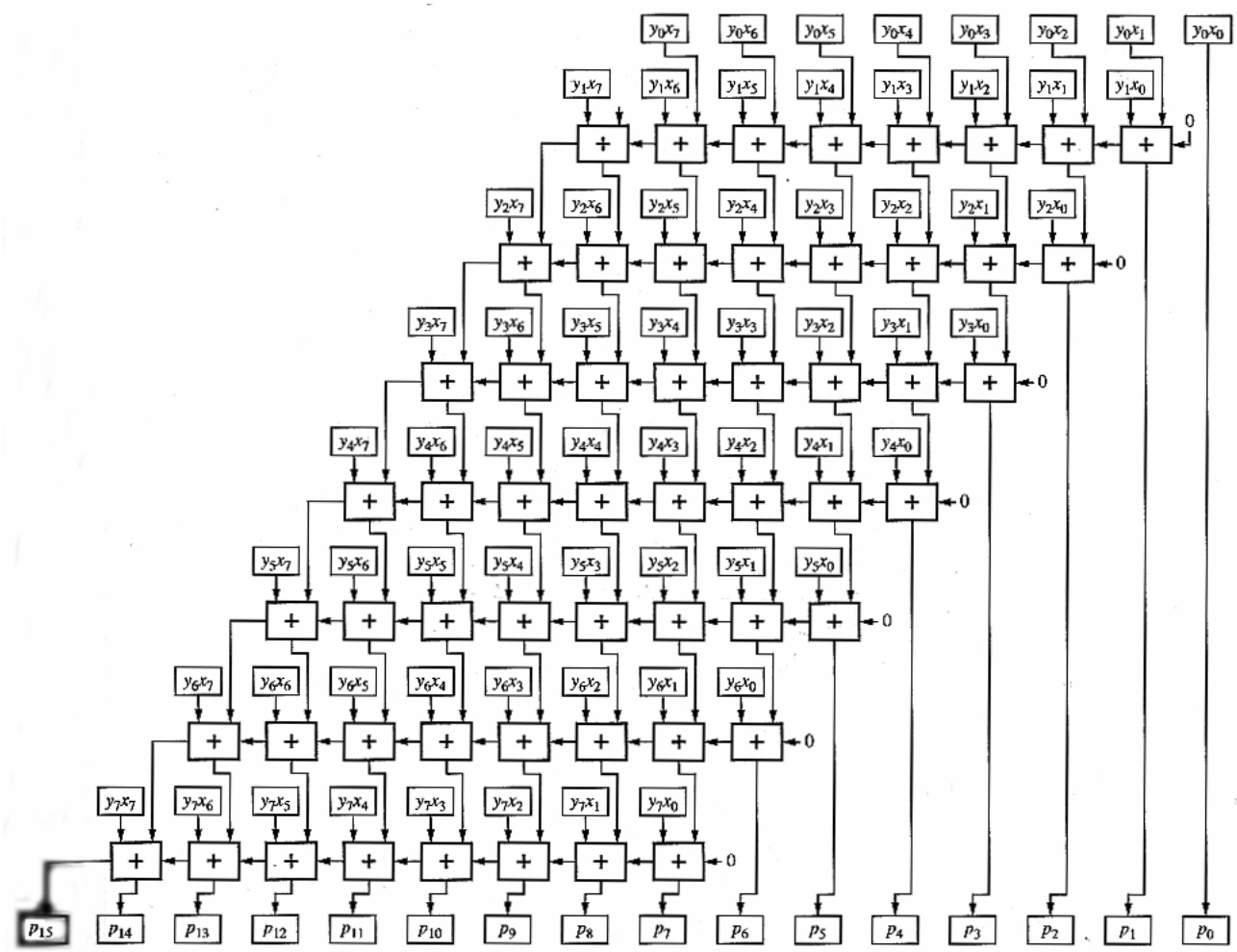
3. Kuva lähteestä John F. Wakerly; Digital Design Principles and Practices Fourth Edition 2006.sivulta 589.

4. Kuva lähteestä John F. Wakerly; Digital Design Principles and Practices Fourth Edition 2006.sivulta 496.

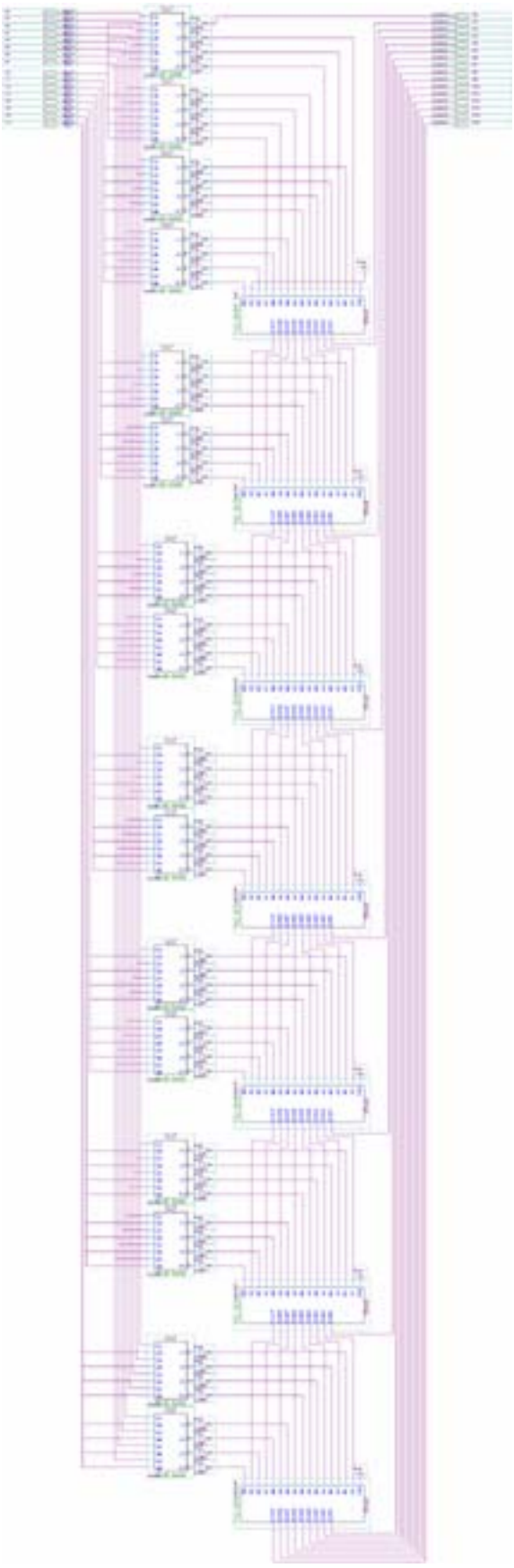
Painamattomat lähteet

5. http://www.tpub.com/content/neets/14185/css/14185_125.htm
kuva: 14185_125_1, (27.08.2007)

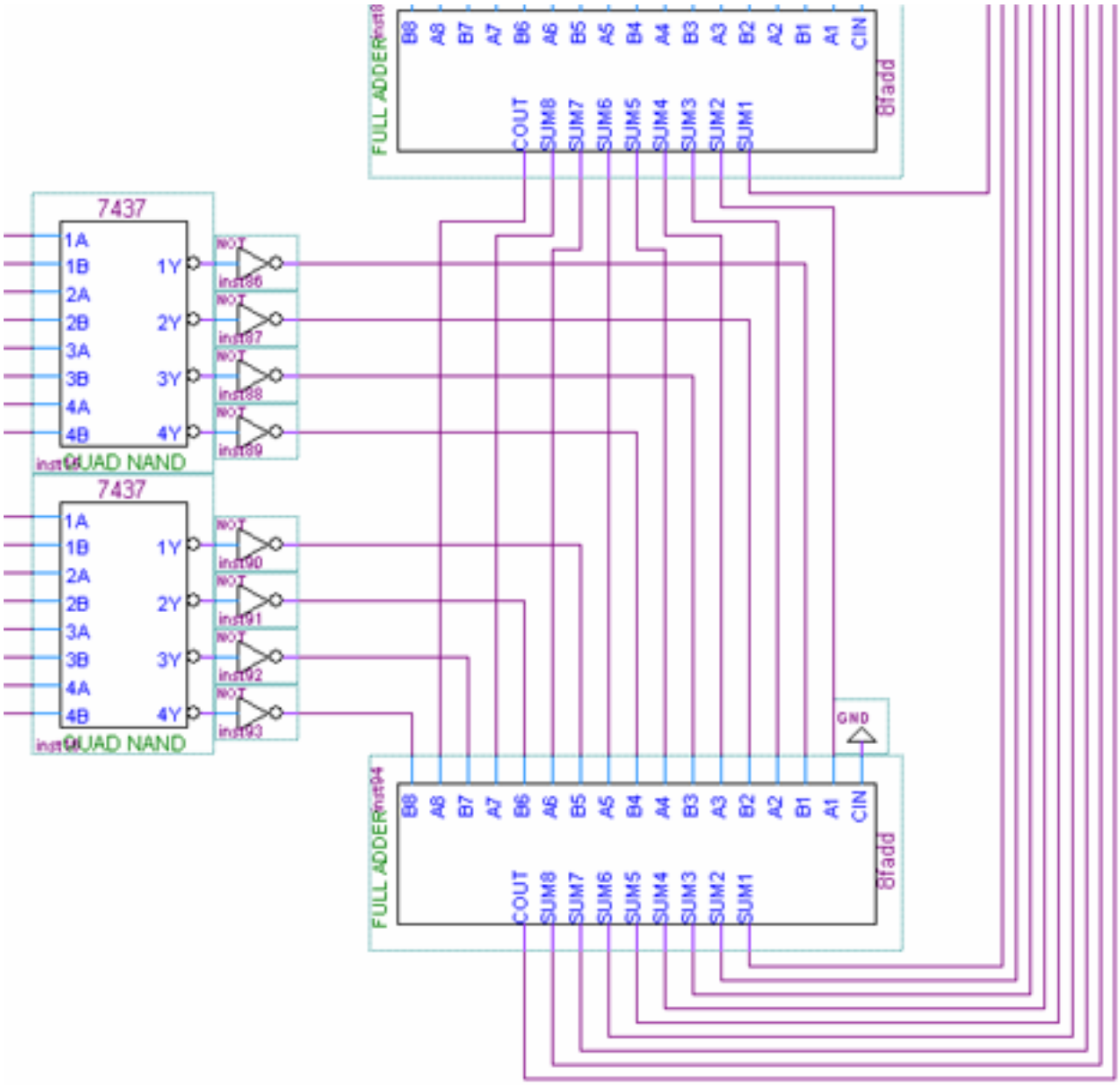
LIITELUETTELO



Kertojanrakenne./4/



8 bittiä x 8 bittiä, kokonainen kytkentäkaavio.



Seitsemännen eli viimeisen vaiheen summat menevät ulostuloihin. ”SUM1” menee ulostuloon P7, ”SUM2” ulostuloon P8 ja niin edelleen aina summaan ”SUM8” asti ja ”SUM8”on P14. Viimeisenä ”COUT”-eli viimeisen kokosummainpiirin muistibitti ”carry out”-bitti viedään ulostuloksi P15.