

Signaalinkäsittelytekniikan laboratorio

Digitaalitekniikan perusteet - luento 1

Tieto ja sen esitystavat

Tieto on yleensä tietoliikenne- tai laitteistotekniikasta puhuttaessa dataa. Tiedolla on esitysmuoto, sitä voidaan tallettaa, toistaa, siirtää ja muokata. Perinteisiä tiedon lajeja ovat esimerkiksi teksti, ääni, kuva, liikkuva kuva, numeerinen data tai multimedia (yhdistelmä, joka sisältää monta edelläkuvattua mediatyyppiä.)

Tietoon liittyvän tekniikan - **tietotekniikan** - kehitys on räjähdysmäistä. Puhumme informaatioyhteiskunnasta ja tiedon merkityksen kasvamisesta. Tiedon ilmaisemiseen käytetään kahta esitystapaa: analogista ja digitaalista.

Peruskäsitteitä:

Analoginen data

Data on analogista, jos sen arvo voi vaihdella portaattomasti tietyllä välillä. Analoginen data on olemassa kaikkina ajanhetkinä ja se voi saada minkä tahansa arvon vaihteluvälillään. Esimerkkejä ovat vaikkapa, ääni tai lämpötila. Puhutaan analogisista lämpömittareista. (esim. nestelämpömittari)

Analogiselle datalle vastakohtana voidaan pitää diskreettiä dataa, joka voi saada vain tiettyjä, toisistaan erillisiä arvoja. Näitä arvoja on aina äärellinen määrä. Analogisesta datasta saadaan diskreettiä näytteistämällä. Diskreetti data voidaan koodata edelleen numeeriseksi eli digitaalseksi dataksi.

Digitaalinen (eli numeerinen) data

Digitaalinen data muodostuu joukosta lukuja. Jokainen luku on matemaattisen lukuavaruuden piste, joka kuvaa olemassa olevan datan tiettyä arvoa. Digitaalista dataa voidaan siirtää lukujärjestelmästä toiseen. Yleensä digitaallilaitteissa käytetään ns. binääri- eli kaksikantajärjestelmää. Ihmiselle havainnollisempi on tavallinen kymmenkantajärjestelmä, jollaista näkee esimerkiksi digitaalisissa lämpömittareissa, kelloissa, jne..

Digitaalinen data eroaa analogisesta siinä, että sillä pystytään ilmaisemaan lukuarvoja vain äärellisellä tarkkuudella. Esimerkiksi digitaalinen lämpömittari voi ilmaista lämpötilan vaikkapa 0,1 ° C tarkkuudella, jolloin tätä pienemmät erot jäävät havaitsematta.

Tiedon tallennus ja siirto

Jotta tiedolla olisi käyttöä, täytyy sitä voida tarvittaessa **tallentaa, toistaa** tai **siirtää**. Dataa voidaan tallentaa, toistaa ja siirtää niin analogisena kuin digitaalisena. Yleensä dataa täytyy muokata tai muuntaa sopivampaan muotoon ennen muita toimenpiteitä.

Tieto vääristyy ja kadottaa pienen osan informaatiostaan aina, kun sitä muokataan. Erilaisia vääristymiä ovat esimerkiksi särö, kohina ja impulssihäiriöt. Tällöin tiedon tarkkuus huononee ja voi syntyä virheinformaatiota.

Tallennus

Tieto saadaan **tietolähteestä**, jonka jälkeen se muutetaan sopivaksi signaaliksi tallennusta varten. Talletettaessa tietoa **tietovälineelle** sitä vielä muokataan sopivampaan muotoon. Toistettaessa tieto puretaan tietovälineeltä, minkä jälkeen se muokataan ja muunnetaan takaisin käyttäjälle sopivaan muotoon.

Esimerkki tiedon tallennus- ja toistojärjestelmästä

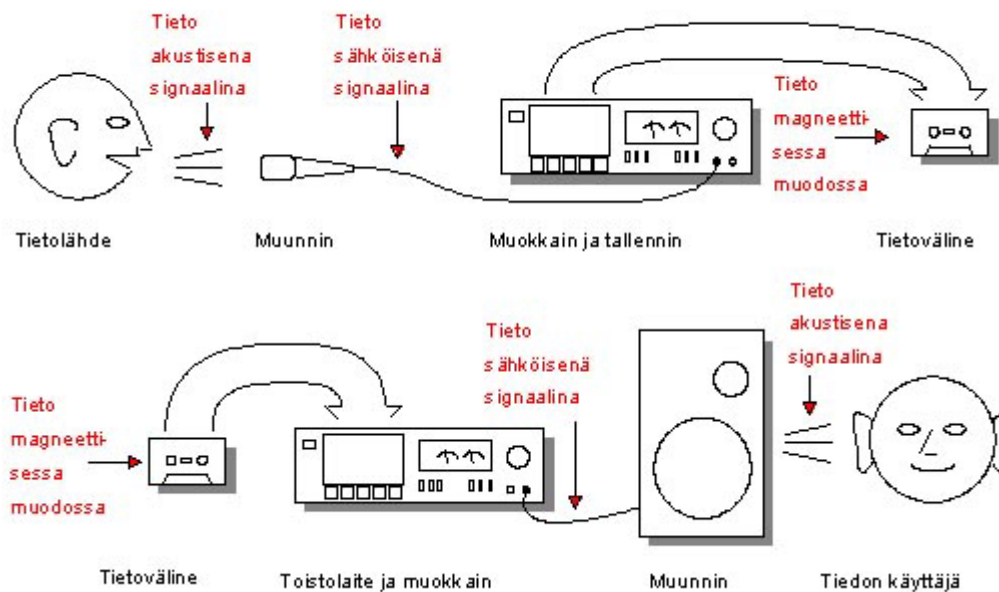
Tallennetaan puhetta tavalliselle c-kasetille. Tämä siksi, että näin saamme esimerkin vanhantyyppisestä analogisesta tiedon tallennuksesta: Tietolähde on siis ihminen, ja puhe on talletettava tieto. Puhe on akustinen signaali (ilman pitkittäistä aaltoliikettä). Se on myös hyvä esimerkki analogisesta signaalista.

Puhe muunnetaan ensin mikrofoniin sähköiseen muotoon. Sen jälkeen mikrofoniin saatavaa sähköistä signaalia muokataan tallennukseen sopivaan muotoon. Muokkauksen jälkeen se tallennetaan kasetille. (Sanotaan, että puhetta nauhoitetaan.)

Tiedon uudelleenkäyttö on toistoa. Toistossa tietoa muokataan uudestaan, ensin kaiuttimille sopivaan sähköiseen muotoon. Sitten kaiutin muokkaa saamansa sähköisen signaalin akustiseksi, jollaisena tiedon käyttäjä ottaa sen vastaan. (Eli kuuntelee kasettia.)

Vastaavassa digitaalisessa tallennuksessa muunnettaisiin signaali ensin digitaalseksi ja tallennettaisiin bitteinä tietovälineelle (esim. CD- tai audioCD-levy, DAT-nauha) Virheidenkorjaavaa koodausta käyttäen saadaan alkuperäinen signaali lähes vääristymättömänä talteen. Digitaalisen tallennuksen etu on, että ajan kulumisen tai toisto ei yleensä aiheuta lisävääristymää tietoon.

Esimerkki: puheen tallennus magneettinauhalle



Siirto

Tiedon siirto on pitkälti vastaava sen tallennuksen kanssa:

Tieto saadaan tietolähteeltä, muutetaan sopivaksi signaaliksi, muokataan ja lähetetään siirtotielle. Siirtotien varrella signaalia tarvittaessa vahvistetaan ja toistetaan. Siirtotien toisessa päässä signaali muokataan ja muunnetaan takaisin alkuperäiseen muotoonsa.

Esimerkki tiedonsiirtojärjestelmästä

Esimerkkinä siirretään puhetta puhelinverkossa:

Puhe muunnetaan ensin luurin mikrofonilla sähköiseen muotoon.

Mikrofonista saatavaa signaalia muokataan edelleen siirtoon soveltuvaan muotoon ja lähetetään siirtotielle. Siirtotie voi sisältää käytännössä esimerkiksi puhelinkaapelia, puhelinkeskuksia, mahdollisesti optista kuitua.

Siirtotien toisessa päässä signaalia muokataan uudelleen luurin kuulokkeelle sopivaan muotoon. Kuuloke taas muuttaa sähköisen signaalin akustiseksi vastaanottajaa varten.

Esimerkki: puheen siirto puhelinverkossa



Signaali voidaan siirtää analogisessa, digitaalisessa tai osittain kummassakin muodossa. Analogia-digitaali- (AD-) ja digitaali-analogia- (DA-) muunnos on päivän sana.

Signaalin käsite

Signaali on datan fysikaalinen (käytännössä usein sähköinen) olomuoto, jonka avulla dataa siirretään paikasta toiseen.

Analoginen vs. digitaalinen tiedonsiirto

Analogisessa siirrossa signaali vastaa arvoiltaan suoraan dataa, joka siirretään. Digitaalisessa siirrossa tehdään ensin analogia-digitaali (eli AD) -muunnos (käsitellään seuraavalla sivulla tarkemmin) ja varsinainen signaali koostuu dataa kuvaavista numeerisista arvoista. Tällä menettelyllä mahdollistetaan pienempi häiriötodennäköisyys (virhe signaalissa ei automaattisesti merkitse virhettä signaalin tietosisällössä) ja parempi virheenkorjaus.

Analogisen tiedonsiirron ominaisuuksia

- Signaali vaimenee ja vääristyy siirrettäessä. Lisäksi siirtotiellä signaaliin summautuu kohinaa
- Signaalia vahvistettaessa vahvistetaan myös häiriötä samalla.
- Vääristymiä voidaan korjata hyvin rajoitetusti, koska häiriötä ei voida erottaa varsinaisesta signaalista
- Analogisen signaalin laatua mitataan signaali-kohina-suhteella (Signal-to-Noise Ratio, SNR)

Tämä luku kertoo hyötysignaalin määrän taustakohinaan nähden. Mitä suurempi SNR, sitä parempilaatuinen signaali on kyseessä. Esim SNR = 20 dB tarkoittaisi, että signaali on 20dB eli 100 kertaa voimakkaampi kuin kohina.


Digitaalisen tiedonsiirron ominaisuuksia

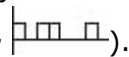
- Osa informaatiosta kadotetaan ensin tehtävässä analogia-digitaalimuunnoksessa.
- Signaali vääristyy ja vaimenee, mutta se voidaan regeneroida eli uusia toistimissa.
- Useimmat vääristymät voidaan korjata, koska signaalilla on vain harvoja sallittuja arvoja.
- Silloin tällöin voi esiintyä esim. niin suuri jännitepiikki, että jokin bitti voi "kääntyä ympäri" ts.
0 --> 1 tai toisinpäin (1 --> 0) . Tällöin on tapahtunut bittivirhe.
- Digitaalisen signaalin laatua mitataan bittivirhesuhteella (Bit Error Ratio, BER). Tämä luku kertoo virheellisten bittien määrän verrattuna johonkin referenssimäärään esim. $BER = 5 \cdot 10^{-6}$, tarkoittaisi, että 5 bittiä miljoonasta on virheellisiä. Mitä pienempi BER, sitä parempi signaali on kyseessä.

Yleisesti ottaen digitaalinen tiedonsiirto ja -tallennus vaativat analogista monimutkaisemman laitteiston, mikä viivytti aikoinaan niiden käyttöönottoa. Nykyinen elektroniikan ja mikropiirien kehittyneisyys kuitenkin takaavat digitaalisen siirron kannattavuuden.

Muunnettaessa dataa analogisesta digitaalseksi (analogia-digitaali eli AD -muunnos) hukataan tietoisesti dataa. Tämä kannattaa, kun mitoitetaan virhe tarvittavan tarkkuustason ulkopuolelle. (Esimerkiksi sopivalla näytteenottotaajuuden ja kvantisointitasojen lukumäärän valinnalla.) Tällöin päästään analogista siirtoa huomattavasti parempaan virheensietokykyyn varsinaisessa siirrosta.

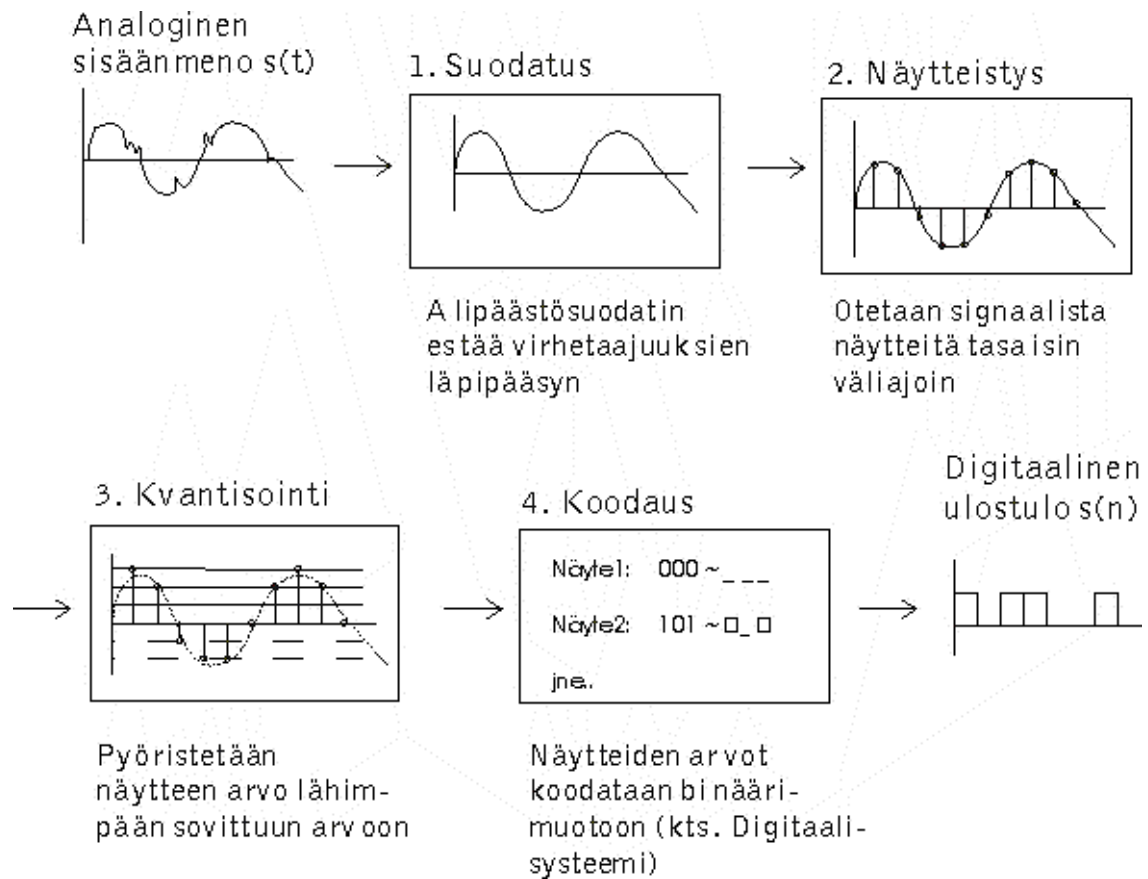
Analogia-digitaalimuunnos

Analoginen signaali () muunnetaan digitaalseksi (koodiksi, joka voidaan esittää esim. kahden jännitetason muutoksina.

Vaikkapa: 1011001 ~ .

AD-muunnos sisältää neljä vaihetta:

1. Suodatus: Analogisesta signaalista suodatetaan pois suurtaajuiset häiriöt, jotka kuvassa näkyvät piikkeinä. (Suodatuksesta ja suodattimista puhutaan enemmän kurseissa Ele 2, signaalit ja järjestelmät jne..)
2. Näytteistys: signaalista otetaan määrävälein näytteitä. Esim 1 ms välein, jolloin katsotaan mikä arvo signaalilla juuri silloin on.
3. Kvantisointi: näytteet kvantisoidaan eli pyöristetään vastaamaan lähintä taulukoitua arvoa. Esim 2,7 V --> 3 V
4. Koodaus: saadut näytearvot koodataan tallennusta tai siirtoa varten sopivaan muotoon. Esim arvoa 3 V vastaa koodi 011 (tässä esimerkissä)



AD-muunnoksen epäideaalisuuksia

AD-muunnoksessa menetetään aina osa informaatiosta. Osa häviöistä on toivottua kuten suodatuksessa poistuvat ylimääräiset häiriöt. Näytteenotossa ja kvantisoinnissa voidaan vahingossa hukata ns. tarpeellista informaatiota. Pääsääntö on tarkistaa kaksi perusasiaa:

- Näytevälin täytyy olla tarpeeksi pieni. (Eli olennaista tietoa ei saa jäädä näytteiden välistä huomaamatta.)
- Kvantisointitasoja täytyy olla tarpeeksi eli numeeriselle koodaukselle on varattava tarpeeksi tilaa.

Näytevälin sijasta puhutaan yleensä näytetaajuudesta, joka on näytevälin pituuden käänteisarvo.

Digitaalialogiamuunnos

Siirron (tai talletuksen) jälkeen koodattu tieto muunnetaan yleensä taas analogiseksi. Tähän tarkoitukseen käytetään digitaalialogiamuunninta (eli D/A-muunninta). D/A-muunnoksen vaiheet ovat seuraavat:

1. koodattu tieto dekodataan digitaalisiksi arvoiksi (luvuiksi)
2. muodostetaan lukuja vastaavat analogiset signaaliarvot (esim. jännitteet)
3. sijoitetaan analogiset signaaliarvot peräkkäin s.e. niiden aikaväli on yhtä pitkä kuin aikaisemmin A/D-muunnoksessa käytetty näytteenottoväli
4. venytetään signaaliarvoa pitopiirillä täyttämään väli seuraavaan arvoon asti
5. poistetaan saadun signaalin kulmikkaus suodattamalla

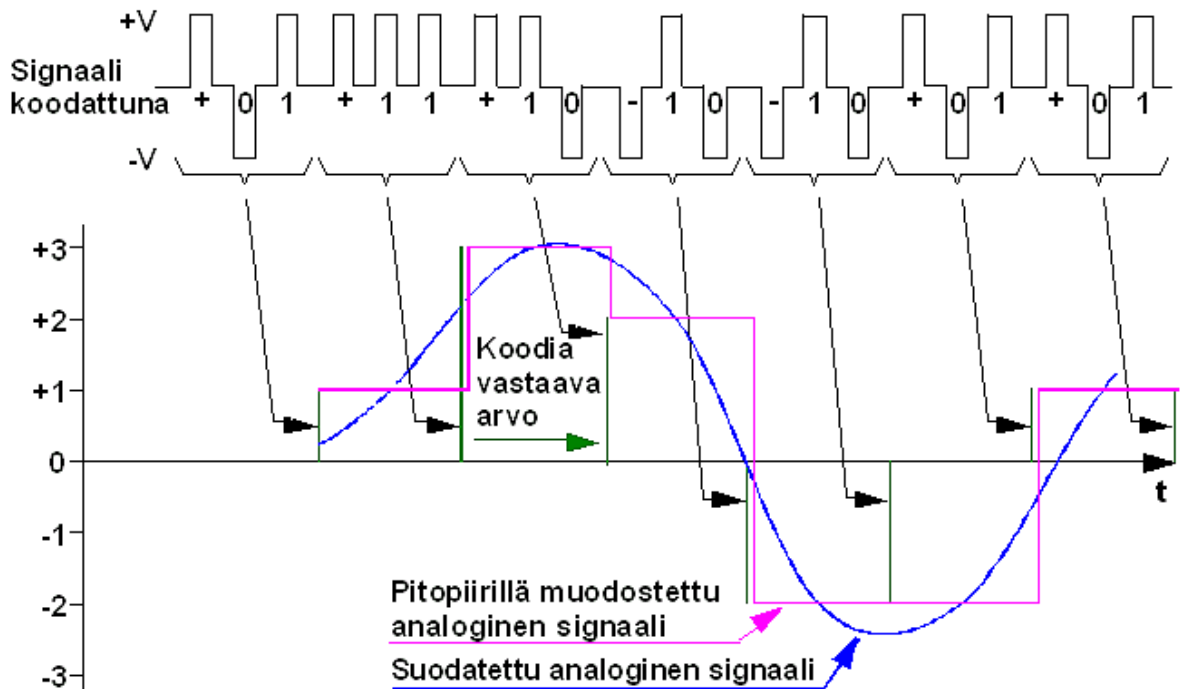
D/A-muunnoksen vaiheet, esimerkkikuva

D/A-muunnos

Huom: Tässä kuvassa D/A-muunnos on esitetty hyvin eri tyyppisesti kuin A/D-muunnos aiemmillä sivuilla. (Näin teoreettisella tasolla esitysmahdollisuuksia on useita.)

Esimerkkikuvassa käytetyt koodit ovat:

+01 = +1,
+10 = +2,
+11 = +3



Digitaalisysteemi

Kurssilla pyritään antamaan valmiudet yksinkertaisten digitaallilaitteiden ymmärtämiseen ja suunnitteluun. Suuremman mittakaavan digitaalisuunnittelun tuotteita ovat esimerkiksi tietokoneet ja niihin liitettävät lisäkortit, 'hardware'. Arkielämän tuotteista digitaalipiirejä voi löytää lähes kaikkialta: kännyköistä, videoista, pesukoneista, CD / minidisc soittimista, autoista, lentokoneista jne.. Lista on loppumaton.

Perinteisen jaon mukaan digitaalisysteemejä on kahdenlaisia:

- koneiden toimintaa ohjaavia (automaatiota)
- laskusuorituksia suorittavia

Digitaalisysteemille ominaista on, että se toimii binäärilogiikalla. Binäärilogiikka tarkoittaa logiikan muotoa, jossa kaikki laskutoimitukset suoritetaan 2-kantajärjestelmässä. Tällöin pienin käytetty tiedon yksikkö on bitti. Yhdellä bitillä on kaksi mahdollista tilaa, 0 (epätosi) ja 1 (tosi). Nämä tilat esitetään usein jännitteinä. Bitin arvo 1 voi vastata esimerkiksi jännitettä 5 voltia ja tila 0 jännitettä 0 voltia. Jännitteitä kutsutaan loogisiksi signaaleiksi.

Koska yleensä tarvitaan enemmän kuin kaksi vaihtoehtoa, yhdistetään monta bittiä peräkkäin ja annetaan näille jonona uusi merkitys. Sanotaan, että tieto esitetään koodattuna.

Signaaleja voidaan kuvata numeroarvoilla, jännitteillä sekä toisinaan myös kytkimillä. Näiden välinen analogia voi toimia esimerkiksi seuraavasti:

Looginen arvo	looginen jännite	kytkin
False (=epätosi) 0	0 V	auki
True(=tosi) 1	5 V	kiinni

Logiikan toteutus on aina sovittu - tapauskohtaisesti tai standardien mukaan. (Se ei ole mikään fysikaalinen totuus.) Edelläkuvattu analogia pohjautuu positiiviseen logiikkaan. Vastaavasti negatiivisessa logiikassa korkeampi jännitetaso vastaisi loogista arvoa 0 (epätosi) ja matalampi jännitetaso loogista arvoa 1 (tosi).

Digitaalitekniikan historiaa (hyvin lyhyesti, yleissivistystä varten)

Mekaanisen laskentatekniikan kausi

- Noin vuosituhannelta 3000 eaa aina nykypäiviin saakka käytössä on ollut helmitaulu (eli abacus).
- 800-luvulla (jaa) Al-Khuwarizmi, bagdadilainen aikansa suurin matemaatikko, kehitti tunnetulle 9-alkioiselle lukujärjestelmälle yhteenlasku-, vähennys-, kerto- ja jakolaskuoperaatiot. (Huomattakoon, että samainen sivistys levisi Eurooppaan vasta 1200 -luvulla, kun Khuwarizmin kirjoituksia alettiin julkaista meillä päin.)
- 1600 -luvulla kehiteltiin ensimmäisiä mekaanisen laskimen prototyypppejä, nämä jäivät kukin tosin vain tekijänsä käyttöön ja suhteellisen epäluottoettavia. (mm. Pascal, Schickard, Leibniz)
- 1800-luvulla saatiin markkinoille ensimmäiset kaupalliset mekaaniset laskukoneet
- 1800-luvulla Charles Babbage kehitti tietokoneen esiäidin: koneen, joka osasi suorittaa lukuisia yksinkertaisia laskutoimituksia peräjäkeen virheettä.

Tietokoneiden aikakausi

- Sähkömekaaniset eli reletietokoneet
 - Ensimmäinen Bellin laboratoriossa USA:ssa vuonna 1939, kiinteäohjelmainen relaelaskukone. Hallitsi neljä aritmeettista operaatiota (+ - * :)
- Puolijohdetietokoneet 1956-63
 - Transistori keksittiin jo vuonna 1948 (John Baden, Walter Brattain, William Shockley)
 - Ensimmäinen transistoritietokone Tx-0 vuonna 1956 (Massachusetts Institute of Technology)
 - Ensimmäinen sukupolvi ohjelmointikieliä, kääntäjiä jne..
- Mikropiireihin (IC, integrated Circuit) perustuvat tietokoneet 1964-71
 - Mikropiiri keksittiin vuonna 1959 (Jack Kilby, Texas Instruments)
 - Teknologinen vallankumous mikropiirien ja printattavien piirilevyjen osalta
 - Kehittyneitä sovelluksia, monisuoritteisia (multitasking) operaatiosysteemejä, ohjelmakirjastoja
 - Intelin ensimmäinen mikroprosessori Intel 4004 markkinoille vuonna 71
- Neljännen sukupolven tietokoneet 1971-
 - Integroidut piirit (IC: Integrated Circuits) laajenevat (LSI: Large Scale Integrated Circuits, VLSI: Very Large Scaled Integrated Circuits,).
 - Nykyään yhdellä chipillä voi olla miljoonia komponentteja
 - Tietoliikennetekniikan kehitys mahdollistaa monelle koneelle hajautetut prosessit
- Viides sukupolvi 1980-
 - Keinoälyn kehittäminen laitteistoon (puheen- ja hahmontunnistus jne..) - koneita myös kotikäyttöön
- 2000- - Multimedian aikakautta, ohjelmoitavat piirielementit kasvaneet laskentatehoiltaan valtaviksi
 - CPLD (Complex Programmable Logic Device) ja FPGA (Field-Programmable Gate Array) johtavat ohjelmoitavat piirityypit digitaalitekniikassa.

- Digitaalitekniikan suunnittelun pääalat: FPGA, ASIC (Application Specific Integrated Circuit)- ja SOC (System on Chip).

Digitaalitekniikan käyttöalueita

Ensialkuun digitaalitekniikkaa käytettiin ensisijaisesti tieteen ja liike-elämän laskentatehtäviin sekä tietokoneisiin. Viimeisten parin vuosikymmenen aikana mikropiirien, puolijohdeteknologian ja elektroniikan yleinen kehitys ovat kuitenkin mahdollistanut digitaalitekniikan käyttöönoton hyvin monilla aloilla. Näitä ovat esimerkiksi:

- tietoliikenne ja teleliikenne
- automaatio, mittaus- ja säätölaitteet
- viihde-elektroniikka, lelut, pelit
- liikenne (esim. vaihtuvat nopeusrajoitukset, liikennevalojen ohjaus, autoissa paljon digitaalitekniikkaa)
- joukkoviestintä (paljon keskustelua viime aikoina on herättänyt esim. digitaalinen televisio)
- DVD:t, CD:t, audioCD:t,
- monet multimedialaitteet
- Tietotekniset lisälaitteet ja laajennukset: digitaalikamerat, kommunikaattorit, videoneuvottelulaitteistot, videotykit
- Erilaiset puolustusteollisuuden systeemit

Digitaalitekniikka on siis osa jo lähes kaikkia elämänalueita. Ennustetaan lisäksi, että tulevaisuudessa digitaalitekniikan merkitys tulee entisestään kasvamaan. Esimerkiksi tietoliikenneala tulee tarvitsemaan huomasti lisää digitaalista laskentatehoa tulevaisuuden mediapuhelinten ja niiden tukiverkkojen muodossa. Toisaalta myös tietotekniikan jatkuva nopea kehitys vaatii koko ajan lisää digitaaliselta suunnittelulta. (Tämän hetken trendit näyttävät olevan multimediatekniikan ja erilaisien arkielämän eriytettyjen ratkaisujen kehittämisessä.)

Piirien koko ja monimutkaisuusaste kasvavat räjähdysmäisesti. Toisaalta uusia sovellusalueita löydetään yhä kiihtyvällä tahdilla. Myös monia, perinteisesti analogisia sovellutuksia suunnitellaan uudestaan digitaaliseen muotoon. Digitaalisuunnittelu -nimitys on melko yleistävä. Nykyään alalla työskentelevät ihmiset puhuvat suunnittelusta mieluummin piirin valmistustekniikoiden mukaisin nimityksin. (Esim FPGA-suunnittelusta.) Yksi tämän hetken suuntaus on SOC eli System on Chip suunnittelu. Tämän ideana on rakentaa yhdelle piirille monenlaista logiikkaa, esimerkiksi prosessori, kiinteätä logiikkaa ja muisteja, sekä koota näistä toimiva kokonaisuus.

Tästä kaikesta voidaan vetää johtopäätös, että työllisyys on ja tulee olemaan erittäin hyvä. Ihan peruskurssin tiedoilla ei digitaalilogiikkaa kaupallisessa mielessä lähdetä työstämään, mutta tämän kurssin perusteisiin kaikki yllämainittu pohjaa.

Tällä kurssilla käsitellään digitaalisuunnittelun perusasioita, jotka jokaisen suunnittelijan tulee hallita hyvin. Nykyään suunnittelu tapahtuu yleensä joko laitteistokuvauskielillä ohjelmoiden tai systeemitason kieltä käyttäen. Esimerkkejä laitteistokuvauskielistä ovat VHDL ja Verilog. Systeemitason kieli on esimerkiksi SystemC. Täytyy kuitenkin muistaa, että vaikka

digitaalisuunnittelussa ohjelmoidaan, on ohjelmoinnin pohjana digitaalisuunnittelun säännöt. Näin ollen tavanomaisten ohjelmointikurssien opit eivät digitaalisuunnittelussa päde.

Signaalinkäsittelytekniikan laboratorio

Digitaalitekniikan perusteet - luento 2

Tietokoneen rakenne

Tietokone on laajasti käytetty apuväline yhteiskunnan eri aloilla. Sen vahvuus on monikäyttöisyys. (Samalla fyysisellä laitteella pystytään suorittamaan eri tyyppisiä tehtäviä, laitetta voidaan myös jatkuvasti päivittämään.) Toisaalta suuri osa tietokoneista on erikoistunut suorittamaan tietyn tyyppisiä tehtäviä. Tietokone on siis digitaalilaitte. Sen toiminnan pääpiirteet (huomattavan yksinkertaistetusti) esitetään alla:

Tietokoneen ydin on integroitu piirielementti, jota kutsutaan **prosessoriksi**. Nykyään prosessorit ovat erittäin monimutkaisia ja sisältävät miljoonia transistoreita. Prosessori suorittaa kaikki koneen toiminnot, jotka ovat kaikki käytännössä erilaisia laskuja, hyppyjä tai muistihakuja. Sen toiminnan määräävät ohjelmat. Prosessori jakaantuu neljään toiminnalliseen yksikköön:

- **CPU, Central Processing Unit**
 - sisältää ohjaimen (Control Unit) ja suorittimen (Datapath). Ohjain hakee muistista käskyjä peräjälkeen ja ohjaa suorittimen toimintaa niiden mukaisesti.
- **FPU: Floating-Point Unit**
 - toimii samantyyppisesti kuin CPU, mutta on erikoistunut ns. liukulukujen laskutoimituksiin
- **MMU: Memory Management Unit**
 - ohjaa muistin toimintaa
- **Internal Cache** (suomennos olisi suunnilleen sisäinen lähimuisti)
 - Nopeimmin haettava muistiaines sijaitsee täällä. MMU:n ohjaama Internal Cache sisältää aktiivisimmin tarvittavaa muistiainesta. (Tulevaisuuden multimediatietokoneissa tämä hierarkinen muistirakenne ei tule toimimaan. Yksi mahdollisuus on silloin liittää suuri määrä nopeaa DRAM -muistia suoraan prosessorin yhteyteen.)

Prossessorin lisäksi tietokone sisältää lisää muistia, joka voidaan jakaa kahteen luokkaan:

- **ROM, Read Only Memory** (suomeksi kiintomuisti): Tämän muistin muistiaines on pysyvää. Muistista voi lukea, mutta sinne ei voi kirjoittaa. Käytetään laitteen ohjelman säilytykseen, voi sisältää joitakin vakioarvoja.
- **RAM, Random Access Memory** (suomeksi vaihtomuisti): Muistista voi lukea ja sinne voi myös kirjoittaa. Tähän talletetaan ohjelman väli- ja lopputuloksia sekä erilaista tarvittavaa dataa. RAM-muisti voi olla joko staattista (SRAM) tai dynaamista (DRAM). RAM-muisti tyhjenee kun tietokoneesta katkaistaan virta. Lisäksi DRAM vaatii jatkuvaa uudelleen virkistystä, eli muistin sisältö täytyy määräjain kirjoittaa uudelleen.

Kaikki muut tietokoneen yksiköt (prossessorin ja muistin lisäksi) voidaan laskea syöttö- tai tulostuslaitteistoon. (I/O Devices, Input/Output). Näitä ovat esim. näyttö, näppäimistö, hiiri, erilaiset lisäkortit, tulostin jne..

Kiintolevy voidaan luokitella sekä I/O-välineeksi että osaksi muistia. Syöttö- ja tulostuslaitteisto on kommunikointirajapinta ulkomaailman kanssa.

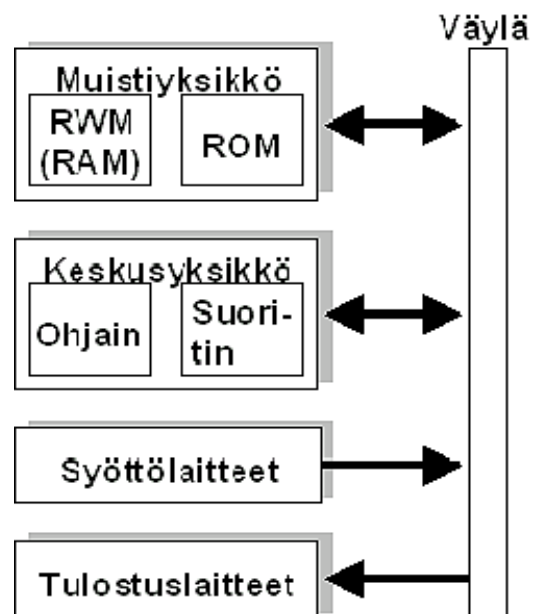
(Kansankielellä sanotaan, että kone ottaa tietoja syöttölaitteen välityksellä ja antaa palautteen tulostuslaitteen kautta. Yleisesti digitaallilaitteessa yhteys ulkomaailmaan hoidetaan otto- ja antopiirien kautta.)

Tietokoneen eri osia yhdistää väylä (Bus). Väylää pitkin siirretään tiedot yksiköltä toiselle.

Kaavakuva tietokoneen rakenteesta:

Tässä on yksi vaihtoehto, jolla tietokonetta voidaan kuvata. Kannattaa muistaa, että kuva ei ole missään suhteessa kattava tai edes ylimalkainen. Lähinnä pyrimme esittelemään yhden hyvin yksinkertaisen mahdollisuuden jakaa kone toiminnallisiin kokonaisuuksiin. (Lukija voi halutessaan piirrellä kotonaan yksityiskohtaisemman kaaviokuvan.)

- Muistiyksikössä on ohjelma, lähtötiedot, tulostiedot ja työtiedot
- Keskusyksikkö hakee muistista käskyn ja suorittaa sen, hakee seuraavan käskyn, suorittaa sen jne..
- Syöttö- ja tulostuslaitteet ovat yhteys ulkomaailmaan: ihmiseen ja ohjattaviin tai tutkittaviin prosesseihin.
- Väylä on ohjelman (käskyjen) ja tietojen siirtotie tietokoneessa
- Tietokoneen toiminta on pääasiassa sarjamuotoista mutta nopeaa.



Lisää tietokoneen rakenteesta ja toimintamalleista kerrotaan kurssilla Tietokoneen arkkitehtuuri.

Lukujärjestelmistä

Nykyinen kymmenjärjestelmämme on perua hedelmällisen puolikuun alueelta ja se on syntynyt jonkin verran ennen ajanlaskumme alkua. Kymmenjärjestelmä perustuu nimensä mukaisesti luvun kymmenen eri potensseihin. Käytetyt numeroarvot ovat 0,1,2,3,4,5,6,7, 8, 9.

Kymmenjärjestelmä voidaan tulkita muodossa:
$$A = \sum_i a_i \cdot 10^i,$$

missä a_i on jokin kerroin välillä $[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]$

Esimerkiksi luku 3025 tarkoittaa oikeastaan lauseketta $3 \times 10^3 + 0 \times 10^2 + 2 \times 10^1 + 5 \times 10^0$.

Toisena esimerkkinä luku 1978, joka tarkoittaa samaa kuin lauseke $1 \times 10^3 + 9 \times 10^2 + 7 \times 10^1 + 8 \times 10^0$.

Eli ensin tuhannet, sitten sadat, kymmenet ja lopuksi ykköset.

Edellä kuvatun ymmärtäminen on tärkeää, jotta voidaan laskea yhteyksiä lukujärjestelmistä toisiin. 10 (eli desimaali) -järjestelmä ei ole läheskään ainoa mahdollinen. Muita paljon käytettyjä ovat erityisesti digitaalilogiikassa käytetty 2-kantajärjestelmä (binääri), 8-kantajärjestelmä (oktaali) ja 16-kantajärjestelmä (heksadesimaali).

Alla olevasta taulukosta nähdään eri lukujärjestelmien vastaavuuksia:

Heksadesimaali	Desimaali	Oktaali	Binääri
0	0	0	0
1	1	1	1
2	2	2	10
3	3	3	11
4	4	4	100
5	5	5	101
6	6	6	110
7	7	7	111
8	8	10	1000
9	9	11	1001
A	10	12	1010
B	11	13	1011
C	12	14	1100
D	13	15	1101
E	14	16	1110
F	15	17	1111

Taulukosta nähdään, että numeroita minimoidakseen kannattaa käyttää mahdollisimman suurikantaista järjestelmää: siinä missä heksadesimaalijärjestelmässä selvittää yhdellä numerolla joudutaan binäärijärjestelmässä pahimmillaan käyttämään neljää numeroa ilmaisemaan samaa lukua. Binäärijärjestelmän laskutoimitukset ovat kuitenkin niin yksinkertaisia, että se on monissa tilanteissa kannattavin vaihtoehto.

Ihmiselle luontevin näyttää olevan desimaalijärjestelmä. Laitteissa taas tarkoituksesta riippuen yleensä kannattaa käyttää jotain muuta järjestelmää.

(Lähes aina binäärijärjestelmää tai sen monikertaa oktaali- tai heksadesimaalijärjestelmää.) Silloin tulostus- ja syöttölaitteita varten tarvitaan muunnoksia lukujärjestelmästä toiseen.

Hyvän digitaalisuunnittelijan tulisi osata perusteellisesti binääri-, oktaali- ja heksadesimaalijärjestelmien väliset suhteet. Samoin tulisi osata nopeasti ratkaista, mitä tietty kymmenjärjestelmän luku on näissä kolmessa järjestelmässä. Lukujärjestelmämuunnoksia esitellään tarkemmin seuraavilla sivuilla.

Lukujärjestelmämuunnokset

Muusta järjestelmästä desimaalijärjestelmään

Lukujärjestelmästä toiseen siirtyminen vaatii laskemista. Jos otetaan perusjärjestelmäksi tuttu desimaalijärjestelmä, muista siihen siirtyminen on helppoa. Lausekemuotoinen määrittely mille tahansa järjestelmälle suhteessa kymmenjärjestelmään voitaisiin ilmaista vaikka seuraavasti:

$$\text{Luku} = \text{Summa}_i (\text{kerroin}_i \times \text{kantaluku}_i)$$

Kun siirrytään jostain muusta järjestelmästä kymmenjärjestelmään, voidaan soveltaa kaavaa suoraan.

Esim. luku $(2634)_7$ (eli luku 2634 7-kantajärjestelmässä) voidaan laskea suoraan auki seuraavasti:

$$(2634)_7 = 2 \times 7^3 + 6 \times 7^2 + 3 \times 7^1 + 4 \times 7^0 = 686 + 294 + 21 + 4 = (1005)_{10}$$

Tässä kantaluku on tietystikin 7, sillä muunnettavaluku on 7-kantajärjestelmässä.

Kaavassa potenssit tulevat sen mukaan millä kohtaa kyseinen numero on alkuperäisessä luvussa $(2634)_7$ eli tuhannet saavat potenssiluvun 3, sadat saavat potenssiluvun 2 jne.

Desimaalijärjestelmästä muuhun järjestelmään

Kun halutaan siirtää luku (n) toiseen suuntaan, etsitään suurinta halutun kantaluvin (k) potenssia (i) , joka 'mahtuu' siirrettävään lukuun. (Eli $n \div k^i$ yhtäsuuri tai suurempi kuin 1) Kun suurin mahdollinen potenssi löydetään, suoritetaan jakolasku $N \div k^i$ ja siirrytään tarkastelemaan mahdollista jakojäännöstä.

Eli:

1. Etsitään uuden kantaluvin (k) suurin potenssi (i) , joka mahtuu 10 -järjestelmän lukuun.
2. Lasketaan, montako kertaa kyseinen potenssi mahtuu alkuperäiseen lukuun, kerroin on uuden luvun eniten merkitsevä numero.
3. Lasketaan jakojäännös yhtälöstä vanhaluku - $(\text{kerroin} \times \text{kantaluku}^{\text{potenssi}})$ ja tarkastellaan sitä.
4. Etsitään uuden kantaluvin potenssi, joka mahtuu jakojäännökseen.

5. Lasketaan, montako kertaa tämä potenssi mahtuu jakojäännökseen, saatava kerroin on uuden luvun seuraavaksi eniten merkitsevä numero.
6. Lasketaan jälleen uusi osamäärä, tarkastellaan sitä jne.. kunnes koko luku on käyty läpi.
7. Uusi luku muodostuu kertoimista.

Esimerkki lukujärjestelmämuunnoksesta:

Muunnetaan luku **(3718)₁₀** 7-kantajärjestelmään.

1. Etsitään iteroimalla/kokeilemalla 7:n (haluttu kantaluku) suurin potenssi, joka mahtuu kyseiseen vanhaan 10-järjestelmän lukuun:

$$7^4 = 2401 < \mathbf{3718} < 7^5 = 16807$$

2. 7^4 mahtuu lukuun 3718 yhden kerran ($3718 \div 7^4 = \mathbf{1,548521}$), joten uuden luvun eniten merkitseväksi kertoimeksi tulee **1**
3. Tarkastellaan jakolaskun jäännöstä ($= 3718 - 1 \times 7^4 = \mathbf{1317}$):
4. $7^3 = 343 < \mathbf{1317} < 7^4 = 2401$
5. 7^3 mahtuu jakojäännökseen 3 kertaa ($1317 \div 7^3 = \mathbf{3,83965}$) joten uuden luvun toiseksi suurimmaksi kertoimeksi tulee **3**
6. lasketaan seuraava jakojäännös: $1317 - 3 \times 7^3 = \mathbf{288}$

$$7^2 = 49 < \mathbf{288} < 7^3 = 343$$

7^2 mahtuu 288:n 5 kertaa ($288 \div 7^2 = \mathbf{5,87755}$), joten uuden luvun seuraavaksi kertoimeksi tulee **5**

lasketaan seuraava jakojäännös: $288 - 5 \times 7^2 = \mathbf{43}$

$$7^1 = 7 < \mathbf{43} < 7^2 = 49$$

7^1 mahtuu 43:een 6 kertaa ($43 \div 7^1 = \mathbf{6,142857}$), joten uuden luvun seuraavaksi kertoimeksi tulee **6**

lasketaan seuraava jakojäännös: $43 - 6 \times 7^1 = \mathbf{1}$

osamääräksi jää **1**, joka on suoraan uuden luvun viimeinen numero

7. uusi luku on siis **(13561)₇**

Esimerkki 2.

Muunnetaan luku **(1241)₁₀** 5-kantajärjestelmään.

1. Etsitään iteroimalla/kokeilemalla 5:n (haluttu kantaluku) suurin potenssi, joka mahtuu kyseiseen vanhaan 10-järjestelmän lukuun:

$$5^4 = 625 < \mathbf{1241} < 5^5 = 3125$$

2. 5^4 mahtuu lukuun 1241 yhden kerran ($1241 \div 5^4 = 1,9856$), joten uuden luvun eniten merkitseväksi kertoimeksi tulee **1**

3. Tarkastellaan jakolaskun jäännöstä ($= 1241 - 1 \times 5^4 = \mathbf{616}$):

4. $5^3 = 125 < \mathbf{616} < 5^4 = 625$

5. 5^3 mahtuu jakojäännökseen 4 kertaa ($616 \div 5^3 = 4,928$) joten uuden luvun toiseksi suurimmaksi kertoimeksi tulee **4**

6. lasketaan seuraava jakojäännös: $616 - 4 \times 5^3 = \mathbf{116}$

$5^2 = 25 < \mathbf{116} < 5^3 = 125$

5^2 mahtuu 116:n 4 kertaa ($116 \div 5^2 = 4,64$), joten uuden luvun seuraavaksi kertoimeksi tulee **4**

lasketaan seuraava jakojäännös: $116 - 4 \times 5^2 = \mathbf{16}$

$5^1 = 5 < \mathbf{16} < 5^2 = 25$

5^1 mahtuu 16:een 3 kertaa ($16 \div 5^1 = 3,2$), joten uuden luvun seuraavaksi kertoimeksi tulee **3**

lasketaan seuraava jakojäännös: $16 - 3 \times 5^1 = \mathbf{1}$

osamääräksi jää **1**, joka on suoraan uuden luvun viimeinen numero

7. uusi luku on siis **(14431)₅**

Tulos voidaan vielä varmistaa jo aiemmin opitun muunnoksen avulla (5-järj. -- > 10-järj.) Saadaan seuraavaa:

$$(14431)_5 = 1 \times 5^4 + 4 \times 5^3 + 4 \times 5^2 + 3 \times 5^1 + 1 \times 5^0$$

$$= 625 + 500 + 100 + 15 + 1 = (1241)_{10}$$

Huom: Ylläolevat periaatteet soveltuvat **etumerkki-itseisarvomuotoisten** (=jo peruskoulussa opittu 'tavallinen' tapa esittää positiivisia ja negatiivisia lukuja) kokonaislukujen muuntamiseen järjestelmästä toiseen. Oma lukunsa on desimaalilukujen tai komplementoitujen (toinen tapa esittää etumerkillisiä lukuja) lukujen muuntaminen.

Binääriluvuista

Digitaalilogiikassa yleisimmin käytetään binääri- eli kaksikantajärjestelmää. Järjestelmän pienin erillinen yksikkö on bitti. Bitillä voi olla kaksi arvoa, 1 tai 0. Binäärilukujen vahvuuksia ovat laskutoimitusten yksinkertaisuudet ja lukujen fysikaalisen esittämisen helppous. (On vain kaksi numeroa.) Heikkouksia ovat lukujen pituudet, epähavainnollisuus ja laskutoimituksien suuri määrä. Digitaalilaitteissa binäärilukuja säilytetään rekistereissä. Luvun esittämiseen on käytössä tietty vakiomäärä tai sen monikerta bittejä. Esimerkiksi 8, 16 tai 32.

Esimerkki 2-kantajärjestelmän luvusta: $011010_2 = 1 \times 2^4 + 1 \times 2^3 + 1 \times 2^1 = 26_{10}$.

2. Esimerkki 2-kantajärjestelmän luvusta: $101001_2 = 1 \times 2^5 + 1 \times 2^3 + 1 \times 2^0 = 41_{10}$.

Ylläolevan esimerkin lukua sanotaan kiinteään pilkun luvuksi. Kiinteään pilkun luvut ovat aina kokonaislukuja. Ne voivat olla joko positiivisia lukuja, jolloin luku tulkitaan sellaisenaan tai etumerkillisiä. Tällöin luvun eniten merkitsevä bitti on merkkibitti. Yleensä 1 tarkoittaa miinusta ja 0 plussaa.

Etumerkillisillä luvuilla on erilaisia esitystapoja:

- Etumerkki-itseisarvoesitys: esitys toimii kuten desimaaliluvuissa ollaan totuttu. Tämä esitys hankaloittaa yhteen- ja vähennyslaskua, kertolasku on yksinkertainen. Etumerkki-itseisarvoesityksessä eniten merkitsevä bitti on siis etumerkkibitti, loput bitit muodostavat varsinaisen luvun. Ensimmäistä bittiä ei oteta huomioon luvun arvoa laskettaessa, muut bitit tulkitaan normaalisti.

Esimerkiksi 4-bittinen luku 1010_2 . Tässä ensimmäinen bitti on merkkibitti, joka kertoo luvun olevan negatiivinen, koska merkkibitti on **1**. Loput bitit (010) kertovat, että luvun itseisarvo on 2 ($1 \times 2^1 = 2$). Näin $1010_2 = -(0 \times 2^2 + 1 \times 2^1 + 0 \times 2^0) = -2_{10}$

2. Esimerkki 5-bittinen luku 11001_2 . Tässä ensimmäinen bitti on jälleen merkkibitti, joka kertoo luvun olevan negatiivinen, koska merkkibitti on **1**. Loput bitit (1001) kertovat, että luvun itseisarvo on 9 ($1 \times 2^3 + 1 \times 2^0 = 9$). Näin $11001_2 = -(1 \times 2^3 + 0 \times 2^2 + 0 \times 2^1 + 1 \times 2^0) = -9_{10}$

- BCD (Binary Coded Decimal) -luvut ovat 10-järjestelmän lukuja joiden jokainen numero on muutettu erikseen binääriseksi

Esimerkiksi 10-järjestelmän luku 24_{10} saadaan BCD-muotoon muuttamalla erikseen binääriseksi luvut 2 ja 4. Muuntaminen binääriseksi tapahtuu samalla tavalla kuin mihin tahansa kantaan (binääriluvuthan ovat 2-kantajärjestelmän lukuja)

$$\begin{aligned} 2_{10} &\rightarrow 10_2 \\ 4_{10} &\rightarrow 100_2 \end{aligned}$$

Kaikki BCD luvut esitetään lopulta neljällä bitillä, koska suurin numero joka voidaan joutua muuttamaan on 9_{10} , joka on binäärijärjestelmässä 1001 ts.

$$24_{10} \rightarrow 0010\ 0100 \text{ (BCD)}$$

2. Esimerkki 10-järjestelmän luku 165_{10} esitettynä BCD-muodossa saadaan kun kaikki kolme numeroa muutetaan erikseen binääriseksi. $1_{10} \rightarrow 1_2$
 $6_{10} \rightarrow 110_2$
 $5_{10} \rightarrow 101_2$

$165_{10} \rightarrow 0001\ 0110\ 0101$ (BCD)

- Alla olevasta taulukosta binääriluvun esittäminen eri esitystavoissa:

Desimaali luku	Etumerkki -itseisarvo	1:n komplementti	2:n komplementti
+7	0111	0111	0111
+6	0110	0110	0110
+5	0101	0101	0101
+4	0100	0100	0100
+3	0011	0011	0011
+2	0010	0010	0010
+1	0001	0001	0001
+0	0000	0000	0000
-0	1000	1111	--
-1	1001	1110	1111
-2	1010	1101	1110
-3	1011	1100	1101
-4	1100	1011	1100
-5	1101	1010	1011
-6	1110	1001	1010
-7	1111	1000	1001
-8	--	--	1000

Esim seuraavasti (kyseessä on 32-bittinen liukuvan pilkun luku):

[illegible]

s = merkkibitti (aina 1 kpl)
e = eksponentin bitit (tässä tapauksessa 8 kpl)
f = mantissan bitit (tässä tapauksessa 23 kpl)

On siis sovittu, että kyseisessä bittijonossa ensimmäinen bitti kertoo luvun etumerkin. Siitä tietty määrä seuraavia bittejä ilmaisee luvun kantaluvun eksponentin arvon. Loput bitit tulkitaan mantissana. Liukuluvuista on olemassa muutama eri standardi. Tässä materiaalissa esitetään niistä yksi. Liukuluvun laskentakaava on esitetty alla:

$$\text{Luku} = (-1)^s \times 2^e \times 1.f$$

(Liukuluvuista lisää myöhemmin.)

Binäärijärjestelmän komplementeista

Yleisesti missä tahansa lukujärjestelmässä (kantaluku r), on kaksi erityyppistä komplementtia: kantaluvun komplementti (*radix complement*) ja kantaluvun heikennetty komplementti (*diminished radix complement*). Näistä edellistä kutsutaan r :n (eli kantaluvun) komplementiksi ja jälkimmäistä $(r - 1)$:n komplementiksi. Nämä komplementit binäärijärjestelmän tapauksessa ovat siis 2:n komplementti ja 1:n komplementti, sillä binäärijärjestelmän kantalukuhan on 2.

Komplementtia tarvitaan digitaalijärjestelmissä kuvaamaan negatiivisia ja positiivisia lukuja. Komplementointi on eräänlaista lukualueen peilaamista tietyn peilauskohdan suhteen. Kyseessä on taas sopimus: on sovittu, että tiettyä rajapyykkiä itseisarvoltaan suuremmat binäärisarjat tulkitaan negatiivisina ja tätä samaa rajapyykkiä pienemmät positiivisina. Rajapyykki sijoitetaan mahdollisimman tarkasti lukualueen puoliväliin. Käytäntö on, että jos komplementtiluvun eniten merkitsevä bitti on 0, luku on positiivinen. Jos komplementtiluvun eniten merkitsevä bitti on 1, on luku negatiivinen.

Jos luku on positiivinen, se tulkitaan normaalisti sijoittamalla lausekkeeseen tarvittavat kakkosen kantaluvut, kuten on aiemminkin jo tehty.

(Esim $0101 = 1 \times 2^2 + 1 \times 2^0 = 4 + 1 = 5$).

Jos luku on negatiivinen, täytyy tehdä peilausoperaatio. Peilauksessa (riippuen komplementin tyypistä) suhteutetaan loput bittisarjan bitit ensimmäiseen lukuun.

Esimerkiksi luku 10101. Eniten merkitsevä bitti on 1, joten luvun muut bitit peilataan sen suhteen. 2-komplementin tapauksessa peilaus tapahtuu näin: $10101 = -16 + 4 + 1 = -11$.

(Jos komplementin idea ei heti aukea, ei kannata masentua. Normaalielämässä riittää pitkälle, jos muistaa, että komplementointi tarkoittaa lukualueen peilausta. Digitaalisuunnittelussa työkalut yleensä huolehtivat komplementoinnista. Siihen törmätään vain erityisen haastavissa tapauksissa tai ongelmatilanteissa. Tentissä tosin asiaa kysytään usein jonkin verran.)

1-komplementti

Olkoon B positiivinen kokonaisluku, jonka suuruusosan esityspituus on n bittiä. Tällöin B:n 1:n komplementti on

$$C_{1,n}(B) = 2^n - 1 - B$$

1-komplementin muuntaminen desimaalimuotoon kannattaa tehdä siten, että muunnetaan luku ensin 2-komplementtiin. Komplementtien välillä vallitsee yhteys:

$$C_{2,n}(B) = C_{1,n}(B) + 1$$

2-komplementti

Olkoon B positiivinen kokonaisluku, jonka suuruusosan esityspituus on n bittiä. Tällöin B:n 2:n komplementti on

$$C_{2,n}(B) = 2^n - B$$

Käytännössä helppona nyrkkisääntönä voidaan muistaa seuraavaa: Jos luku on positiivinen, se tulkitaan normaalisti. Tällöin muunnos 10-järjestelmään tapahtuu aivan samalla tavalla kuin on aiemmin esitetty.

Jos luku on negatiivinen, tulkitaan luvun eniten merkitsevä bitti (merkkibitti) aivan kuin se olisi tavallinen kerroin, mutta **negatiivisena**. Tämän jälkeen tulkitaan muut kertoimet positiivisina.

Esimerkki 1

Luku 100110_2 (luku on siis 2-komplementtimuodossa, merkkibitti on 1 eli luku on negatiivinen) tulkitaan seuraavasti:

$$100110_2 = -1 \times 2^5 + 1 \times 2^2 + 1 \times 2^1 = -32_{10} + 4_{10} + 2_{10} = -26_{10}$$

Esimerkki 2

Luku 101101_2 (luku on siis 2-komplementtimuodossa, merkkibitti on 1 eli luku on negatiivinen) tulkitaan seuraavasti:

$$101101_2 = -1 \times 2^5 + 1 \times 2^3 + 1 \times 2^2 + 1 \times 2^0 = -32_{10} + 8_{10} + 4_{10} + 1_{10} = -19_{10}$$

Esimerkki 3

Luku 011001_2 (luku on siis 2-komplementtimuodossa, merkkibitti on 0 eli luku on positiivinen) tulkitaan seuraavasti:

$$011001_2 = 1 \times 2^4 + 1 \times 2^3 + 1 \times 2^0 = 16_{10} + 8_{10} + 1_{10} = 25_{10}$$

Binääriluvun komplementin muodostaminen

1-komplementti on helppo muodostaa:

Kaavaa ($C_{1,n}(B) = 2^n - 1 - B$) tulkittaessa huomataan, että 1-komplementin luvusta saa suoraan kääntämällä kaikki bitit nurin (eli ykkösestä tulee nolla, nollasta ykkönen). Jos luku on lyhyempi kuin käytetty bittien määrä, lisätään alkuun lisää nollia ja käännetään ne muunnoksessa ykkösiksi. Eli esim. jos on valittu 4 bittinen esitys tapa ja käsiteltävä luku on $5_{10} = 101_2$. Nyt pitää lisätä yksi nolla luvun alkuun jolloin luvuksi tulee 0101_2

HUOM ! Mikäli luku on aluksi itseisarvo-etumerkki muodossa, jätetään etumerkki rauhaan. Siis etumerkkiä ei käännetä edellisen ohjeen mukaisesti vaan se pysyy samana.

Esimerkki 1

Suuruusosan esityspituus 7 bittiä, luku $B=11001$ ($=25_{10}$)

- Lisätään tarvittavat nollat: $B=0011001$
- Käännetään bitit: $C_{1,7}(B) = 1100110$, mikä on haluttu tulos

Tuloksen voi tarkistaa muuntamalla sen edelleen 2-komplementiksi ja sitä kautta kymmenjärjestelmään:

2-komplementiksi eli lisätään 1: $C_{2,7}(B)=1100111=$
Muuntaminen 2-komplementista 10-järjestelmään tehdään kuten on esitetty edellä:

$$C_{2,7}(B)=1100111 = -2^6 + 2^5 + 2^2 + 2^1 + 2^0 = -25_{10}$$

Esimerkki 2

0110011 Suuruusosan esityspituus 7 bittiä, luku $B=1001101$ ($=77_{10}$)

- Nyt ei tarvitse lisätä numeroita luvun eteen, sillä sen pituus on jo 7 bittiä.
- Käännetään bitit: $C_{1,7}(B) = 0110010$, mikä on haluttu tulos

Tuloksen voi tarkistaa muuntamalla sen edelleen 2-komplementiksi ja sitä kautta kymmenjärjestelmään:

2-komplementiksi eli lisätään 1: $C_{2,7}(B)=0110011$.

HUOM ! Tämä vaikuttaisi olevan positiivinen luku, mutta sehän ei voi olla sitä, koska edellä suoritimme komplementtia muodostaessamme peilauksen. Tämän takia tulee nyt lisätä yksi ykkönen koko luvun eteen, jotta tulos olis negatiivinen kuten pitääkin.

$$C_{2,7}(B)=0110011 = -2^7 + 2^5 + 2^4 + 2^1 + 2^0 = -77_{10}$$

Esimerkki 3

Suuruusosan esityspituus 7 bittiä, luku (itseisarvo-etumerkki muodossa) $B=10011010$ ($=-26_{10}$). Ensimmäinen bitti kertoo etumerkin (1 = -) ja sitten tulee luvun suuruusosa (7 bittiä).

- Nyt ei kosketa lainkaan etumerkkiin. Lisäksi huomataan, että suuruusosa on jo 7 bittiä pitkä, joten ei tarvetta toimenpiteisiin.
- Käännetään bitit: $C_{1,7}(B) = 11100101$, mikä on haluttu tulos

Tuloksen voi tarkistaa muuntamalla sen edelleen 2-komplementiksi ja sitä kautta kymmenjärjestelmään:

$$\begin{aligned} \text{2-komplementiksi eli lisätään 1: } C2,7(B) &= 11100110 \\ &= -2^7 + 2^6 + 2^5 + 2^2 + 2^1 = -26_{10} \end{aligned}$$

Esimerkki 4

Suuruusosan esityspituus 7 bittiä, luku (itseisarvo-etumerkki muotoinen) $B=11001 (= -9_{10})$. Ensimmäinen bitti kertoo etumerkin (1 = -) ja sitten tulee luvun suuruusosa (4 bittiä).

- Nyt ei taaskaan kosketa lainkaan etumerkkiin. Huomataan kuitenkin, että suuruusosa on vain 4 bittiä pitkä, joten siihen lisätään nollia. Näin saadaan: $B=1\mathbf{000}1001$, missä lisätyt nollat on lihavoitu.
- Lopuksi käännetään bitit: $C1,7(B) = 11110110$, mikä on haluttu tulos

Tuloksen voi tarkistaa muuntamalla sen edelleen 2-komplementiksi ja sitä kautta kymmenjärjestelmään:

$$\begin{aligned} \text{2-komplementiksi eli lisätään 1: } C2,7(B) &= 11110111 \\ &= -2^7 + 2^6 + 2^5 + 2^4 + 2^2 + 2^1 + 2^0 = -9_{10} \end{aligned}$$

Esimerkki 5

Suuruusosan esityspituus 7 bittiä, luku (itseisarvo-etumerkki muotoinen) $B=01100 (= 12_{10})$. Ensimmäinen bitti kertoo etumerkin (0 = +) ja sitten tulee luvun suuruusosa (4 bittiä).

- Nyt ei taaskaan kosketa lainkaan etumerkkiin. Huomataan kuitenkin, että suuruusosa on vain 4 bittiä pitkä, joten siihen lisätään nollia. Näin saadaan: $B=0\mathbf{000}1100$, missä lisätyt nollat on lihavoitu.
- Lopuksi käännetään bitit: $C1,7(B) = 00001100$, mikä on haluttu tulos

HUOM! Tässä oli siis **positiivisen** luvun muuntaminen 1-komplementtiin. Tarkkasilmäisimmät voivat huomata, että luku ei muuttunut lainkaan (mitä nyt nollia tuli lisää..) Jos tämä tuntuu hämärältä ks. selitystä aiemmin komplementin johdannon kohdalta.

Tuloksen voi tarkistaa muuntamalla sen edelleen 2-komplementiksi ja sitä kautta kymmenjärjestelmään:

Muuntaminen 2-komplementiksi on helppoa **positiivisen** luvun tapauksessa. Ei tehdä mitään muutoksia eli: $C2,7(B)=00001100= 2^3 + 2^2 = 12_{10}$

2-komplementti muodostetaan joko 1-komplementin kautta lisäämällä siihen 1 (kuten edellisen esimerkin tarkistusosassa) tai suoraan seuraavalla algoritmilla:

- aloitetaan vähiten merkitsevistä bitistä
- jos bitti on 0 sitä ei muuteta, vaan siirrytään seuraavaan
- ensimmäinen vastaan tuleva 1-bitti säilytetään ennallaan
- loput bitit käännetään (kaikki)

Esimerkki

Suuruusosan esityspituus 7 bittiä, luku $B=10100= 0010100 (= 10_{10})$
 $C2,7(B)=1101100 (= -10_{10})$

Komplementin komplementti

$$C2,n(C2,n(B)) = 2^n - (2^n - B) = B$$

$$C1,n(C1,n(B)) = 2^n - 1 - (2^n - 1 - B) = B$$

Komplementoimalla komplementti saadaan siis alkuperäinen luku uudelleen.

Pari esimerkkimuunnosta

	Etumerkki-itseisarvoinen luku	1:n komplementti	2:n komplementti
A	01100011	01100011	01100011
B	10110010	11001101	11001110

Positiiviset luvut ovat siis joka muodossa samanlaisia. Negatiivisissa suuruusosasta tehdään komplementti. Etumerkki (jos sellainen on) jätetään komplementoitaessa rauhaan.

Muiden lukujärjestelmien komplementeista

Komplementti voidaan ottaa myös muissa lukujärjestelmissä kuin binäärijärjestelmässä. Tässä on esitetty esimerkinomaisesti miten tapahtuu muuntaminen muissa lukujärjestelmissä. Mikäli etumerkki sisältyy ensimmäiseen lukuun, se on komplementtimuodossa.

Esimerkki 1

Luku $+37_8 (=31_{10})$ on itseisarvo-etumerkki muodossa.

- Muistetaan, että komplementoitaessa positiivista lukua ei luvulle tapahdu mitään. Näin ollen:

$+37_8$ komplementoituna on 37_8

Tuloksen voi tarkistaa muuntamalla sen takaisin 10-järjestelmään:

$$37_8 = 3 \times 8^1 + 7 \times 8^0 = 24 + 7 = 31_{10}$$

Esimerkki 2

Luku $-21_4 (= -9_{10})$ on itseisarvo-etumerkki muodossa.

- Nyt negatiivisen luvun komplementoinnissa on syytä olla tarkkana !
- Ensinnäkin huomataan, että suurin mahdollinen luku 4-järjestelmässä kahdella numerolla esitettynä on 33_4 . Tämä vastaa lukua -1_{10} komplementoinnin takia.

Kun luvun ensimmäinen numero on 0 tai 1 = (+) on kyseessä positiivinen luku ja kun ensimmäinen numero on 2 tai 3 = (-), on kyseessä negatiivinen luku. Nyt tiedetään, että tulokseksi on tulossa negatiivinen luku ts. komplementoidun luvun ensimmäinen numero on 2 tai 3 Näin ollen:

Lasketaan $33_4 - 21_4 + 1_4 = 13_4$ **Tämä on väärin! Ensimmäinen numero osoittaisi luvun olevan positiivinen, mitä se ei ole!!**

Tarvitsemme siis esitystä kolmella numerolla jolloin luku $333_4 = -1_{10}$

Lasketaan $333_4 - 21_4 + 1_4 = 313_4$

Tämä on haluttu vastaus.

Hieman lisäselvitystä

1. Suurin luku 4-järjestelmässä on kahdella numerolla 33 tai kolmella 333. Vastaavasti 9-järjestelmässä 88 tai 888.

Tämä suurin luku vastaa lukua -1_{10} , mikäli ei olla itseisarvo-etumerkki-esityksessä. (Tällöinhän luvun etumerkin ilmoittaa suoraan + tai -)

2. Pienin negatiivinen luku olisi esimerkin tapauksessa $200_4 = -32_{10}$

3. Se miksi laskutoimituksissa on mukana +1 johtuu siitä, että aloitamme vähennyslaskun laskemisen luvusta -1_{10} , emmekä luvusta 0_{10}

Toisin sanoen, mikäli emme lisääisi lukua +1, olisi edellinen muunnos ollut luvusta -22_4

4. Luku +1 voidaan myös jättää lisäämättä, mutta silloin täytyy muistaa tietenkin vähentää vain -20_4 (Esimerkkimme tapauksessa)

Tuloksen voi tarkistaa muuntamalla sen takaisin 10-järjestelmään:

$$-(333_4 - 313_4 + 1_4) = -(21)_4 = -(2 \times 4^1 + 1 \times 4^0) = -(8 + 1) = -9_{10}$$

Esimerkki 3

Muunnettava luku $1A_{11}$ takaisin 10-järjestelmään.

- Luvussa ei näy etumerkkiä, joten tulkitaan se positiiviseksi luvuksi ensimmäisen numeron perusteella. ($0 \rightarrow 5 = +$ ja $6 \rightarrow A = -$)
- Nyt muuntaminen käy helposti, koska luku on positiivinen. Muunnetaan se aivan normaalisti (Komplementtiesityksessä positiiviset luvut säilyvät muuttumattomina)
- $1 \times 11^1 + 10 \times 11^0 = (11 + 10) = 21_{10}$

Hieman lisäselvitystä

1. Luku A_{11} vastaa lukua 10_{10} . Muita kirjaimia kuin A ? \rightarrow Ks. taulukko aiemmin, jossa oli esitelty mm. heksadesimaaliluvut, oktaaliluvut ja desimaaliluvut. Taulukon löydät lukujärjestelmä osion alusta.

Esimerkki 4

Muunnettava 9-järjestelmän 9-komplementin luku 82_9 takaisin 10-järjestelmään.

- Luku on negatiivinen ensimmäisen numeron perusteella ($0 \rightarrow 4 = +$ ja $5 \rightarrow 8 = -$)
- Nyt mietitään mikä on suurin mahdollinen luku (kahdella numerolla, koska tehtävänantokin oli kahdella numerolla) 9-järjestelmässä.
- Suurin mahdollinen luku on $88_9 = -1_{10}$
- Nyt tehdään laskutoimitus: $88_9 - 82_9 = 6_9$
 $6_9 + 1_9 = 7_9$
- Lopuksi muunnetaan 7_9 10-järjestelmään kuten on aiemmin opittu ja muistetaan vielä, että käsitellään negatiivista lukua. Tulokseksi saadaan : -7_{10}

Hieman lisäselvitystä

- Miksi laskutoimituksessa lisätään yksi "ylimääräinen" ykkönen?

Tämä tehdään siksi, että jos olisi seuraavanlainen tilanne: Pitäisi muuntaa luku 88_9 takaisin 10-järjestelmään. Tällöin muunnos $88_9 - 88_9 = 0_{10}$. Tämä antaisi tulokseksi luvun nolla. Kuitenkin on sovittu, että 88_9 vastaa lukua -1_{10} . Tämän takia vähennetään yksi "ylimääräinen" ykkönen. Tämä tapahtuu laskemalla $0_9 + 1_9 = 1_9$, joka muunnetaan 10-järjestelmään muistaen kuitenkin, että kyse on neg.luvusta $1_9 \Rightarrow -1_{10}$. Tämä on haluttu (ja oikea) lopputulos

Kiinteän pilkun lukujen laskutoimituksia

Etumerkki-itseisarvomuotoisten binäärilukujen yhteenlasku (ja vähennyslasku)

Algoritmi:

- jos merkkibitit ovat samat
 - lasketaan suuruusosat yhteen
 - merkkibitti säilyy samana
- jos merkkibitit ovat erilaisia
 - vähennetään suuruusosaltaan pienempi suuruusosaltaan suuremmasta
 - merkkibitti on sama kuin suuruusosaltaan suuremmalla

Esimerkkejä: (Ensimmäinen bitti on merkkibitti 0='+' 1='-')

A	B	C	D
0110 1001 + 0001 0110	1000 1111 + 0001 0110	0100 0110 + 0011 0011	0010 1100 + 1000 0110
0111 1111	0000 0111	0111 1001	0010 0110
OK	Merkkibitit erilaisia!	Muista! $01+01=10$	Merkkibitit erilaisia!

B-kohdassa merkkibitit ovat erilaisia, joten vähennetään suuruusosaltaan pienempi (ylempi luku) suuruusosaltaan suuremmasta: $22_{10} - 15_{10} = 7_{10}$

Vastaava tapaus myös D-kohdassa: $44_{10} - 6_{10} = 38_{10}$

Jokainen voi itse tarkistaa laskut, esim. muuntamalla ne 10-järjestelmään ja sieltä tulos takaisin binääriseksi.

Binäärilukujen yhteenlaskua varten käytetään kokosummain rakennetta, joka tulee kurssilla myöhemmin esille (luennossa 6).

1-komplementtimuotoisten binäärilukujen yhteenlasku

Algoritmi:

- lasketaan luvut **merkkibitteineen** yhteen
- jos merkkibiteistä muodostuu ylimääräinen summabitti, lisätään se summan vähiten merkitsevään bittiin.

Esimerkkejä:

A	B	C	D
0011 0101 +0010 0010	1000 1111 + 0001 0110	1011 0010 +0111 0001	1111 0111 +1000 0110
0101 0111	1010 0101	1 0010 0011 '----->1 0010 0100	10111 1101
Muista! 1+1=10	Muista! 1+1+1=11	OK	Ylivuoto

A ja B kohdassa bitit on yksinkertaisesti laskettu yhteen.

C-kohdassa ylimääräinen merkkibitti on lisätty vähiten merkitsevään bittiin.

D-kohdassa tapahtuu ylivuoto (merkkibitti on vaihtunut, luku on liian suuri lukualueeseen. Ks. 2-komplementtien yhteenlaskua).

2-komplementtimuotoisten binäärilukujen yhteenlasku

Algoritmi:

- lasketaan luvut **merkkibitteineen** yhteen
- poistetaan** tarvittaessa merkkibiteistä muodostunut 'ylimääräinen bitti', niin sanottu ylivuoto

Esimerkkejä:

A	B	C	D
0011 0101 +0010 0010	1000 1111 + 0001 0110	1011 0010 +0111 0001	1111 0111 +1000 0110
0101 0111	1010 0101	1 0010 0011 => 0010 0011	1 0111 1101 =>0111 1101
OK	OK	Poistetaan 'ylimääräinen' bitti	Ylivuoto

A ja B kohdassa bitit on yksinkertaisesti laskettu yhteen.

C-kohdassa laskuun tulisi ylimääräinen merkkibitti, mutta sitä ei oteta huomioon vaan tiputetaan pois. (Se siis ei näy lopullisessa tuloksessa.)

D-kohdassa merkkibitti on muuttunut positiiviseksi, vaikka molemmat luvut ovat negatiivisia. Kyseessä on **ylivuoto**: tulos on itseisarvoltaan liian suuri esitettäväksi 8 bitillä ja näin ollen **tulos on siis virheellinen**. Tämä on tyypillinen vikatilanne, joka voidaan tavanomaisissa tapauksissa helposti korjata kasvattamalla sanapituutta eli lisäämällä lukujen esittämiseen käytettävien bittien määrää.

Komplementtimuotoisten lukujen vähennyslasku

- Tehdään vastaluvun yhteenlaskuna: $A - B = A + (-B)$
- $-B = \text{kompl}(B)$

Komplementtimuotoisten binäärilukujen vähennyslaskuesimerkkejä

A, B, C ja D ovat 2-komplementtimuotoisia binäärilukuja.

A=01001110

B=00110111

C=11111000

D=10001001

Laske A - B, A - C, A - D ja C - D

Muodostetaan ensin komplementoimalla luvut -B, -C ja -D:

B=00110111 ; -B=11001001

C=11111000 ; -C=00001000

D=10001001 ; -D=01110111

Ja varsinaiset laskut:

A-B	A-C	A-D	C-D
01001110 +11001001	01001110 +00001000	01001110 +01110111	11111000 +01110111
00010111	01010110	11000101 1 00100100	01101111
OK	OK	Ylivuoto	OK

Liukuvan pilkun luvuista

Liukuvan pilkun lukujen käytön perusteita:

- kiinteän pilkun esityksessä käytössä suppea lukualue. Esim. 7:llä bitillä 0-128.
(Esityspituuden lisääminen epäkannattavaa: yhden bitin lisääminen vain kaksinkertaistaa lukualueen)
- Lisäksi jos kiinteän pilkun esityksessä kasvatetaan lukualuetta, kasvattaa se myös esitystarkkuutta, mikä on usein tarpeetonta
- liukuvan pilkun esitystavalla saadaan yleensä riittävän suuria lukuja 32 bitillä

- esityspituuden lisääminen yhdellä bitillä kasvattaa lukualuetta eksponentiaalisesti

Liukuvan pilkun luku vastaa siis desimaalijärjestelmän ilmaisua 10 eksponentteina.

(Esim. $0,5235 \times 10^{12}$, $2,24 \times 10^{-7}$, -1.3×10^{30})

Liukuvan pilkun luku jakaantuu kolmeen osaan:

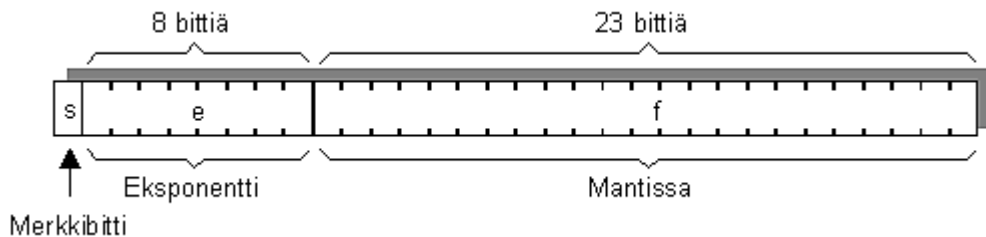
Eniten merkitsevä bitti on merkkibitti (sign, s).

Tämän jälkeen seuraa joukko eksponenttibittejä (exponent, e).

Lopuksi tulee mantissa. (fraction, f)

Lisäksi liukulukuun liittyy siirre (bias, b)

Esimerkki IEEE-standardin mukaisesta liukuluvusta:



$$\text{Luku} = (-1)^s \times 2^{e-b} \times 1.f$$

(mantissa on siis ykköstä pienempi luku, bitit tulkitaan kakkosen negatiivisina potensseina s.e. eniten merkitsevän kerroin on 2^{-1} , seuraavan 2^{-2} jne..) Muistettavaa:

- Eksponentti esitetään aina siirrettyssä eli biasoidussa muodossa. Tässä on kyse siitä, ettei eksponentilla ole omaa merkkibittiä, joten se on periaatteessa aina positiivinen. Koska kuitenkin halutaan pystyä esittämään sekä negatiivisia että positiivisiä eksponentteja, siirretään lukualuetta vasemmalle. Lukualue siirretään siten että sen keskipiste on nollassa (tai sen vieressä, tasan siirto ei mene). (Biasointi tulee esiin 2. laskuharjoituksessa. Ks. Myös esimerkit)
- Mantissa esitetään yleensä itseisarvomudossa.
- Jollei ole erityistä syytä, luvut esitetään aina normaalimuodossa.
- Tarvittaessa luku skaalataan normaalimuotoon eksponenttia muuttamalla.
- Normaalimuodossa luvulla on suurin mahdollinen esitystarkkuus.
- Mantissa on normaalimuotoinen, mikäli mantissa alkaa ykkösellä.

Esimerkki 1 Tulkitse binääriluku $(1001\ 1000\ 0101\ 0100)_2$ liukuvan pilkun lukuna.

Tässä tehtävässä ensimmäinen bitti on merkkibitti, eksponentti esitetään viidellä bitillä ja loput 10 kuuluvat mantissaan, joten esitys **ei ole** IEEE-standardin mukainen

Edetään tehtävässä seuraavasti:

- Katsotaan ensin mikä on tulevan luvun merkki (onko luku positiivinen vai negatiivinen).
Koska merkkibitti (s) on 1, luku on negatiivinen
- Tämän jälkeen tulkitaan eksponentti, eli seuraavat 5 bittiä muutetaan 10-järjestelmään, kuten on opittu aiemmin.

$$00110_2 = 6_{10}$$

- Koska halutaan sekä positiiviset, että negatiiviset *eksponentit* käyttämme siirrettä (bias).
Siirteen suuruus on puolet koko lukualueen määrästä, joka eksponentilla voidaan esittää. Tässä tapauksessa eksponentti on 5 bittiä pitkä eli sillä voidaan esittää $2^5 = 32$ erilaista lukua. Siirteen keskipiste on nollan vieressä (koska lukualue ei mene tasan) jolloin voidaan esittää eksponentit välillä -15 --> 16. Ts. siirre on **15**
- Eksponentin (e) ollessa 6_{10} ja siirteen (b) 15_{10} on $e-b=-9_{10}$
- Loput 10 bittiä kuuluvat mantissaan ja niiden muuntaminen 10-järjestelmään tapahtuu helposti

$$0001010100_2 = 0 \times 2^{-1} + 0 \times 2^{-2} + 0 \times 2^{-3} + 1 \times 2^{-4} + 0 \times 2^{-5} + 1 \times 2^{-6} + 0 \times 2^{-7} + 1 \times 2^{-8} = 0,0625 + 0,015625 + 0,00390625 = 0,08203125$$
- Lopuksi saamme siis liukuluvuksi: $(-1)^1 \times 2^{6-15} \times 1.0820312$
= 0,002113342₁₀

Esimerkki 2 Tulkitse binääriluku $(0101\ 1000\ 0111\ 1000)_2$ liukuvan pilkun lukuna.

Ensimmäinen bitti on merkkibitti, eksponentti esitetään viidellä bitillä ja loput 10 kuuluvat mantissaan

Edetään tehtävässä seuraavasti:

- Katsotaan ensin mikä on tulevan luvun merkki (onko luku positiivinen vai negatiivinen).
Koska merkkibitti (s) on 0, luku on positiivinen
- Tämän jälkeen tulkitaan eksponentti, eli seuraavat 5 bittiä muutetaan 10-järjestelmään, kuten on opittu aiemmin.

$$10110_2 = 22_{10}$$

- Koska halutaan sekä positiiviset, että negatiiviset *eksponentit* käyttämme siirrettä (bias).
Siirteen suuruus on puolet koko lukualueen määrästä, joka eksponentilla voidaan esittää. Tässä tapauksessa eksponentti on 5 bittiä pitkä eli sillä voidaan esittää $2^5 = 32$ erilaista lukua. Siirteen keskipiste on nollan vieressä (koska lukualue ei mene tasan) jolloin voidaan esittää eksponentit välillä -15 --> 16. Ts. siirre on **15**
- Eksponentin (e) ollessa 22_{10} ja siirteen (b) 15_{10} on $e-b=7_{10}$
- Loput 10 bittiä kuuluvat mantissaan ja niiden muuntaminen 10-järjestelmään tapahtuu helposti

$$0001111000_2 = 1 \times 2^{-4} + 1 \times 2^{-5} + 1 \times 2^{-6} + 1 \times 2^{-7} = 0,0625 + 0,03125 + 0,015625 + 0,0078125 = 0,1171875$$
- Lopuksi saamme siis liukuluvuksi: $(-1)^0 \times 2^{22-15} \times 1.1171875$
= 143₁₀

Lyhyesti liukuvan pilkun luvun laskutoimituksista (esitetään yleissivistyksen vuoksi)

Yhteen- ja vähennyslasku

- lukuista pienempi muutetaan normaalimuodosta muotoon, jossa lukujen eksponentit ovat samat
- mantissoilla tehdään tarvittava laskutoimitus

- eksponentiksi tulee yhteinen eksponentti
- lopuksi normalisoidaan, mikäli se on tarpeen

Kerto- ja jakolasku

- mantissoilla tehdään tarvittava laskutoimitus
- kertolaskussa eksponentit lasketaan yhteen ja summasta vähennetään siirre
- jakolaskussa jaettavan eksponentista vähennetään jakajan eksponentti ja erotukseen lisätään siirre
- lopuksi normalisoidaan, mikäli se on tarpeen

Liukuvan pilkun lukuja käsittelee ANSI/IEEE:n standardi 754-1985. Standardi on laaja ja yksityiskohtainen ja se määrittelee esitystavat ja laskutoimitukset. Standardi on laadittu mikrotietokoneita varten. Se sisältää kaksi perusesitystapaa:

- yksinkertainen esitystarkkuus (single precision): 32 bittiä
 - kaksinkertainen esitystarkkuus (double precision): 64 bittiä
-

Signaalinkäsittelytekniikan laboratorio

Digitaalitekniikan perusteet - luento 3

Merkkikoodista

Digitaallilaitteissa esitetään erilaisia merkkejä. Merkit voivat olla esimerkiksi kirjaimia, numeroita, erikoismerkkejä (esim. | « © ß), puoligraafisia merkkejä tai ohjauskoodia (esim. CR, LF, FF, SYN). Merkit esitetään **merkkikoodien** avulla:

- jokaista merkkiä vastaa tietty koodi eli bittijono
- merkkikoodin bittien lukumäärä riippuu esitettävien merkkien määrästä: n :llä bitillä voidaan esittää 2^n merkkiä
- merkkikoodin bittimäärä on yleensä sama kaikille merkeille

Merkkikoodia:

ASCII-koodi

Aakkosellisten merkkien keskeinen binäärikoodi on ASCII (American Standard Code For Information Interchange). ASCII koodi käyttää seitsemää bittiä esittämään 128 erilaista merkkiä ($2^7=128$). Merkistö sisältää 94 tulostettavaa kirjainta sekä 32 ohjauskoodia. Tulostettavat kirjaimet ovat englanninkielisen aakkoston 26 merkkiä sekä isoina että pieninä kirjaimina, 10 numeroa sekä 32 erikoismerkkiä. (Ohjausmerkit eivät tulostu.)

Huomattavaa on siis, että ASCII ei sisällä ns. 'ääkkösiä'. ASCII-koodista on olemassa alueellisia versioita, joissa jotain erikoismerkkejä on korvattu kansallisilla kirjaimilla.

ASCII-koodi taulukon löydät esim. kirjoista
Logic and Computer Design Fundamentals (Mano & Kime),
The Essence of Digital Design (Wilkinson),
Principles of Digital Design (Gajski)
Fundamentals of Digital Logic with VHDL Design (Brown & Vrasenich)

ANSI-koodi

ANSI-koodauksessa (American National Standards Institute) käytetään kahdeksaa bittiä, joilla voidaan kuvata yhteensä 256 merkkiä ($2^8=256$). Numerot ja kirjaimet ovat vastaavat kuin ASCIIssa, mutta lisäksi koodauksessa on valmiina kansallisia kirjaimia sekä runsaasti lisää erikoismerkkejä.

Muita merkkikodeja (Yllämainitut ovat ylivoimaisesti tärkeimmät.)

DOS-merkistöt

- kuusi vähän toisistaan poikkeavaa merkistöä, yksi näistä pohjoismainen (sis. ääkköset)
- 8 bittiä, 256 merkkiä
- numerot, kirjaimet ja erikoismerkit kuten ASCIIssa
- lisäksi kansallisia kirjaimia, lisää erikoismerkkejä, puoligraafisia merkkejä sekä matemaattisia symboleja
- käytössä DOS -ympäristössä

Baudot -koodi

- käytössä esim. joissakin Telex -laitteissa
- 5 bittiä, 32 merkkiä
- merkistön laajennus tasonvaihtomerkeillä

Virheen havaitseminen ja korjaus

Kun tietoa muutetaan muodosta toiseen, talletetaan tai erityisesti, kun sitä siirretään, syntyy virheitä. Voidaan asettaa erilaisia rajoja sille, missä määrin esiintyessään virheet ovat häiritseviä. Virheiden häiritsevyys vaihtelee siirrettävän datan, sen pakkauksen ja tarkoituksen mukaan. (Esim. tavallisessa äänen siirrossa voi esiintyä paljonkin virheitä, ennen kuin käyttäjä havaitsee ne häiritsevänsä naksahduslauna. Toisaalta taas jpeg -kuvaformaattissa yksikin virheellinen merkkikoodi voi pilata koko kuvan.)

Tiedonsiirrossa on tärkeää huomata virheet vastaanottopäässä. Kun virhe huomataan, voidaan pyytää virheellisen kohdan uudelleenlähettämistä. On myös olemassa virhettä korjaavia koodaustapoja. Virheen havaitseminen perustuu **redundanssin** lisäämiseen. Tämä tarkoittaa sitä, että vain osa biteistä käytetään varsinaisen datan siirtoon. Loppuosa biteistä on joko toistoa (esim. sama bitti lähetetään kahteen kertaan) tai se tulkitaan virhekoodiksi. Jos sanaan tulee virhe matkalla, tulkitaan se osaksi virhekoodia ja joko pyydetään lähettämään uudelleen tai jätetään huomioimatta sanomaa tulkittaessa.

Erilaisia virheentarkistus tai korjausmenetelmiä:

Pariteettitarkastus:

Pariteettibitin käyttö

Pariteettibitin käyttö perustuu siihen, että pakotetaan jokainen koodaussana (joka on siis tietyn mittainen bittijono) pariteetiltaan samanlaiseksi:

- Jos koodaussanassa on parillinen määrä ykkösiä, on käytetty pariteetti parillinen (even).
- Jos koodaussanassa on pariton määrä ykkösiä, on käytössä luonnollisesti pariton pariteetti (odd).

Pariteetin muodostus perustuu siihen, että jokaiseen sanaan lisätään ylimääräinen bitti (pariteettibitti), jonka arvo laitetaan lähetyspäässä joko ykköseksi tai nolaksi siten, että saadaan kaikkiin sanoihin sama pariteetti (siis pariton tai parillinen pariteetti). Vastaanottopäässä tarkastetaan, kunkin sanan pariteetti ja jos löydetään pariteetiltaan muista poikkeava sana, voidaan päätellä virheen tapahtuneen.

Pariteettitarkastuksessa virhe huomataan vastaanottavassa päässä, mutta sitä ei voida korjata paitsi pyytämällä virheellisen kohdan uudelleenlähetystä. Toinen menetelmän heikkous on, että se ei huomaa parillista määrää virheitä samassa sanassa. (Jos siis kaksi bittiä kääntyy nurin päin, säilyy pariteetti.) Tarkastus sopii kätevästi esim. ASCII -merkistöön. Tällöin pidennetään jokainen 7-bittinen sana, lisäämällä niihin pariteettibitti. (Mikä nostaa sanapituuden 8-bittiin eli yhteen tavuun. 8-bittiä on sikäli hyvä sanapituus, että tietokoneen sanat ovat 8-bittisiä tai 8-bitin monikertoja.)

Esimerkki pariteettitarkastuksesta löytyy alla olevan esimerkin lisäksi toisen kierroksen harjoituksista.

Esimerkki 1

Lähetetään sana '**Digis**' ASCII-koodattuna. (ASCII-koodi saadaan esim. Internetistä tai kirjallisuudesta, kuten edellä oli todettu)
Sana ASCII-koodattuna on : 100 0100 110 1001 110 0111 110 1001 111 0011

Käytetään paritonta pariteettia eli otetaan 7 bitin ryhmiä ja lisätään joko 0 tai 1 ryhmän eteen, jotta saadaan pariton määrä ykkösiä 8 bitin sarjaan.
Saadaan: 1100 0100 1110 1001 0110 0111 1110 1001 0111 0011
(ykkösiä) |--3 kpl--| |-5 kpl--| |--5 kpl--| |-5 kpl--| |-5 kpl---| eli jokaisen kirjaimen kohdalla pariton määrä.

Vastaanottopäässä tiedetään, että on käytetty paritonta pariteettia.
Tarkastetaan siis jokaisen vastaanotetun kirjaimen kohdalla pariteetti.
Vastaanotettu bittijono on: 1100 0100 1111 1001 0110 0111 1110 1001 1111 0111
(ykkösiä tässä bittijonossa) |-3 kpl --| |-6 kpl--| |--5 kpl--| |-5 kpl--| |--7 kpl--|

Kirjaimessa numero 2 on siis tullut virhe, koska ykkösten määrä on parillinen vaikka sen piti olla pariton. Tästä kirjaimesta pyydetään uudelleenlähetys ja tulkitaan vastaanotettu ASCII-koodi

Vastaanotettu ASCII-koodi on: 100 0100 110 1001 110 0111 110 1001 111 0111
ja tämä tulkittuna on: '**Digiw**' !! Missä meni vikaan ??

Pariteettibitti on siis yksinkertainen tapa tarkastaa data, mutta siinä on omat puutteensa

Hamming -koodaus

Hamming -koodauksessa käytetään montaa pariteettibittiä yhtäaikaan, jolloin virheellinen bitti voidaan myös paikallistaa ja korjata. (Eli kyseessä on virheenkorjaava koodaus.) Koodauksesta havaitaan kaikki yhden bitin virheet, muut virheet eivät tule ilmi.

- n:lle databitille tarvitaan k pariteettibittiä siten, että $n + k + 1 = < 2^k$
- bittipaikat numeroidaan 1 ... n + k
- pariteettibitit sijoitetaan kakkosen potenssin mukaisille bittipaikoille: 1, 2, 4, 8, jne
- Esim: **b₁**, **b₂**, b₃, **b₄**, b₅, b₆, b₇, **b₈**, ..
- Kukin pariteettibitti muodostetaan osalle databiteistä käyttäen parillista pariteettia.
 - **b₁** muodostetaan databiteille, joiden paikkanumeron binääriesityksen vähiten merkitsevä bitti on 1 (b₃, b₅, b₇, b₉..)
 - **b₂** muodostetaan biteille, joiden paikkanumeron binääriesityksen toiseksi vähiten merkitsevä bitti on 1 (b₃, b₆, b₇, b₁₀..)
- Koodia tulkittaessa muodostetaan pariteettibittejä vastaavat tarkastusbitit C₁, C₂, C₄.. kustakin pariteettibitistä ja sen muodostamiseen käytetyistä databiteistä.
- Tarkistusbitti muodostetaan kuin se olisi pariteettibitti.
- Jos kaikki tarkistusbitit ovat nolliä, koodi on virheetön. Jos eivät, voidaan tarkistusbittien arvoista päätellä virheen paikka.

Monimutkaista siis. Kannatti lukea yleissivistyksen vuoksi.. Jos aihe kiinnostaa enemmän, löytyy esimerkiksi Internetistä paljonkin asiaa Hamming koodauksesta. Todella innokkaille mainittakoon, että käytännön harjoituksena Hamming-kooderin ohjelmointi esimerkiksi C-kieltä käyttäen ei ole mikään mahdoton urakka.

Esimerkki 1 Lähetetään saman bittijonon alku kuin pariteettiesimerkissämme eli sanan 'Digis' ensimmäinen kirjain 'D' ASCII-muodossa. Lähetetään se kahteen eri vastaanottopaikkaan, joista toiseen on hyvä ja toiseen huono yhteys. Saadaan: 100 0100

Nyt meillä on 7 databitin ryhmä. Tarvitaan siis k pariteettibittiä siten, että $n + k + 1 = < 2^k$

Mikäli k = 3. Tällöin $n + k + 1 = 7 + 3 + 1 = 11$ ja $2^k = 2^3 = 8$. Todetaan, että $11 < 8$ mikä ei pidä paikkaansa!

Mikäli k = 4. Tällöin $n + k + 1 = 7 + 4 + 1 = 12$ ja $2^k = 2^4 = 16$. Todetaan, että $12 < 16$ mikä on totta.

Numeroidaan bittipaikat 1 --> 11, jolloin saadaan : b₁, b₂, b₃, b₄, b₅, b₆, b₇, b₈, b₉, b₁₀, b₁₁.

Sijoitetaan pariteettibitit kakkosen potenssin mukaisille paikoille eli 1,2,4, ja 8
Tällöin saadaan seuraava bittijono: b₁, b₂, **1**, b₄, **000**, b₈, **100** (Tummennetut bitit ovat alkuperäiset ASCII-koodista saadut bitit järjestyksessä)

Parillista pariteettia käyttäen

b₁ muodostetaan databiteille (b₃, b₅, b₇, b₉, b₁₁). Kyseiset bitit ovat 10010.

Parillista pariteettia käyttäen saadaan b₁=0

Vastaavasti

b₂ muodostetaan databiteille (b₃, b₆, b₇, b₁₀, b₁₁). Kyseiset bitit ovat 10000.

Parillista pariteettia käyttäen saadaan b₂=1

Edelleen

b₄ muodostetaan databiteille (b₅, b₆, b₇). Kyseiset bitit ovat 000.

Parillista pariteettia käyttäen saadaan b₄=0

Lopuksi

b_8 muodostetaan databiteille (b_9, b_{10}, b_{11}). Kyseiset bitit ovat 100.

Parillista pariteettia käyttäen saadaan $b_1=1$

Lopullinen Hamming-koodattu bittijono on siis: 01100001100

Vastaanotto 1: Vastaanotetaan bittijono 01100001100

Tarkastetaan pariteettibittit:

b_1 muodostetaan databiteille ($b_3, b_5, b_7, b_9, b_{11}$). Kyseiset bitit ovat 10010.

Parillista pariteettia käyttäen saadaan $b_1=0$

b_2 muodostetaan databiteille ($b_3, b_6, b_7, b_{10}, b_{11}$). Kyseiset bitit ovat 10000.

Parillista pariteettia käyttäen saadaan $b_1=1$

jne. jne. Lopuksi saadaan seuraavat pariteettibittit:

- $b_1 = 0$
- $b_2 = 1$
- $b_4 = 0$
- $b_8 = 1$

Nämä vastaavat täysin niitä pariteettibittejä jotka oli muodostettukin lähetyspäässä. Siis vastaanotto on onnistunut.

Vastaanotto 2: Vastaanotetaan bittijono 01100101100

Tarkastetaan pariteettibittit:

b_1 muodostetaan databiteille ($b_3, b_5, b_7, b_9, b_{11}$). Kyseiset bitit ovat 10010.

Parillista pariteettia käyttäen saadaan $b_1=0$

b_2 muodostetaan databiteille ($b_3, b_6, b_7, b_{10}, b_{11}$). Kyseiset bitit ovat 11000.

Parillista pariteettia käyttäen saadaan $b_1=0$

b_4 muodostetaan databiteille (b_5, b_6, b_7). Kyseiset bitit ovat 010.

Parillista pariteettia käyttäen saadaan $b_1=1$

b_8 muodostetaan databiteille (b_9, b_{10}, b_{11}). Kyseiset bitit ovat 100.

Parillista pariteettia käyttäen saadaan $b_1=1$

Lopuksi saadaan seuraavat pariteettibittit:

- $b_1 = 0$
- $b_2 = 0$
- $b_4 = 1$
- $b_8 = 1$

Huomataan, että b_2 ja b_4 ovat väärin!. Seuraavaksi tarkastellaan mikä bitti on kääntynyt:

- b_2 huomioi databittit ($b_3, b_6, b_7, b_{10}, b_{11}$). Joku näistä on mennyt väärin.
- b_4 huomioi databittit (b_5, b_6, b_7). Joku näistä on mennyt väärin. Kun huomioidaan myös b_2 :n bitit voidaan sanoa, että jompikumpi biteistä (b_6, b_7) on väärin.
- b_1 huomioi databittit ($b_3, b_5, b_7, b_9, b_{11}$). Tämä pariteettibitti oli oikein, joten myös bitti b_7 on oikein.

- Voidaan lopuksi päätellä siis, että bitti b_6 on kääntynyt ympäri ja tämä voidaan nyt korjata kun tiedetään missä kohtaa virhe sijaitsee!!

KytKentäalgebra ja logiikkapiiri

Digitaalilaitteet suorittavat erittäin monimutkaisia usein laskennallisia tehtäviä. Pohjimmaiset operaatiot ovat kuitenkin hyvin yksinkertaisia. Digitaalipiirien suunnittelu perustuu ns. Boolean algebraan, joka muistuttaa matemaattista logiikkaa, joskaan ei vastaa sitä kaikilta ominaisuuksiltaan.

Digitaalilaitteet toteutetaan kytKentäalgebralla. KytKentäalgebra pyrkii toimimaan Boolean algebran mukaan. (Nopeissa ja monimutkaisissa sovelluksissa esim. porttien hitaus aiheuttaa rajoituksia.) KytKentäfunktiot eli loogiset funktiot muodostetaan logiikkapiireillä, joita on hyvin eri tyyppisiä. Esimerkiksi porttipiirit, joiden käsittely aloitetaan tällä luennolla ovat loogisia piirejä.

Digitaalisella signaalilla on kaksi arvoa (0 ja 1). Signaaleista muodostetaan uusia signaaleja tietyin perustein. (Käytännössä tämä tarkoittaa sitä, että yhtä tai useampaa signaalia kuljetetaan erilaisten piirielementtien läpi, jolloin muodostuu erilainen uusi signaali. Ehkäpä esimerkki selvittää.)

Esimerkki signaaleista

Esimerkkimme on elävästä elämästä tilanne kun ylitetään ajorataa liikennevalojen kohdalta.

Meillä on kaksi ottosignaalia (nämä siis tulevat laitteeseen sisään), joiden perusteella teemme tienylitys päätöksen:

- EI_AUTO (autoa ei näy): signaalin arvo on 1, kun autoja ei näy; signaalin arvo on 0, kun autoja on tulossa
- V_VALO (vihreä valo palaa): signaalin arvo on 1, kun valo palaa; signaalin arvo on 0, kun valo ei pala

Sovitetaan, että on turvallista ylittää tie, mikäli autoja ei näy **ja** vihreä valo palaa. Halutaan antosignaali (signaali, jonka laite antaa ulos), joka kertoo milloin on turvallista ylittää tie. Annetaan antosignaalille nimeksi TURVA. Tällöin voidaan muodostaa TURVA signaalille looginen ehto: $TURVA = EI_AUTO \text{ ja } V_VALO$. (Mikä siis tulkitaan, että TURVA signaali saa arvon 1, jos sekä $EI_AUTO = 1$ että $V_VALO = 1$)

Tässä esimerkissä kaikki signaalit ovat muuttujasignaaleita. Niiden arvo siis voi vaihdella ajanhetkestä toiseen. (Vakiosignaali olisi aina joko 0 tai 1.)

Loogiset perusfunktiot

Loogiset perusfunktiot muodostavat digitaalilogiikan perustan. Näistä funktioista voidaan muodostaa kaikki muut loogiset funktiot. Loogisia muuttujia merkitään isoilla kirjaimilla (esim. A, B, C, D, ...). Loogiset perusfunktiot ovat EI, TAI ja JA. (NOT, OR, AND).

EI (NOT) -funktio

EI on negaatiofunktio. Jos sisääntulevan muuttujan (ottosignaalin) arvo on 1, tulee sen lähtöarvoksi 0. Jos muuttujan arvo on 0, saa funktio arvon 1. (EI-funktio kääntää siis bitin nurinpäin.) EI -funktio merkitään lausekkeissa pilkulla A' tai viivalla muuttujan päällä: \overline{A}

Totuustaulu on yksiselitteinen tapa esittää funktio. Totuustaulussa esitetään funktion sisäänottamat signaalit ja ulostulevat signaalit siten, että kaikki mahdolliset tapaukset käsitellään. EI-funktion totuustaulu näyttää tältä:

Input A	Output F
0	1
1	0

Eli $F = A'$ tai toisella tapaa merkittynä $F = \overline{A}$

TAI (OR) -funktio

EI -funktio oli yhden muuttujan funktio. TAI funktio voi käsitellä kahta tai useampaa ottosignaalia yhtä aikaa. (Puhutaan 2-otto TAI -porteista, 3-otto TAI -porteista jne..) TAI -funktio saa arvon yksi aina, kun yksikin ottosignaaleista on arvoltaan 1. Jos kaikki ottosignaalit ovat nollia, saa TAI -funktio arvon nolla.

Kahden muuttujan TAI-funktio merkitään seuraavasti: $F = A + B$. Tässä on huomattava, että TAI - funktio ei noudata matematiikan vaan Boolean logiikan laskusääntöjä, mikä nähdään totuustaulusta:

Input A	Input B	Output F
0	0	0
0	1	1
1	0	1
1	1	1

(Siis + ei tarkoita tässä yhteenlaskua, vaan operaatiota TAI. Tämä selittää, miksi voidaan väittää, että $A + B = 1$.)

JA (AND) -funktio

JA -funktio muistuttaa kertolaskua. Myös se voi käsitellä kahta tai useampaa ottosignaalia yhtä aikaa. JA -funktio saa arvon nolla, jos yksikin sen ottosignaaleista on nolla. Vain, jos kaikkien ottosignaalien arvo on yksi, saa myös funktio arvon yksi. Lausekkeissa JA -funktio merkitään kertolaskunotaatiolla: $F = A \cdot B$. Usein kertomerkki jätetään välistä ja kirjoitetaan yksinkertaisesti $F = AB$. Tätä käytetään erityisesti lausekkeissa, joissa on sekä JA että TAI operaatiota. (Jolloin voidaan ilmaista esim: $F = AB + AC + BDE$.)

JA -funktion totuustaulu näyttää siis seuraavalta:

Input A	Input B	Output F
0	0	0
0	1	0
1	0	0
1	1	1

KytKentäalgebran teoreemoja eli Boolean Algebra

Loogiset funktiot toimivat osittain suoraan binääriaritmetiikan mukaan. Eroakin löytyy, yleisin lienee TAI -funktiossa oleva $1 + 1 = 1$. KytKentäalgebrassa JA -funktiota sanotaan loogiseksi tuloksi ja TAI -funktiota loogiseksi summaksi. KytKentäalgebran (eli ns. Boolean algebran) tärkeimpiä teoreemoja on listattu alle:

Yhden muuttujan teoreemoja:

$$\begin{array}{ll} X + 0 = X & X \times 1 = X \\ X + 1 = 1 & X \times 0 = 0 \\ X + X = X & X \times X = X \\ X + X' = 1 & X \times X' = 0 \end{array}$$

Monen muuttujan teoreemoja:

$$\begin{array}{lll} X + Y = Y + X & XY = YX & \text{kommutatiivisuus} \\ X + (Y + Z) = (X + Y) + Z & X(YZ) = (XY)Z & \text{assosiatiivisuus} \\ X(Y + Z) = XY + XZ & X + YZ = (X + Y)(X + Z) & \text{distributiivisuus} \\ \overline{X+Y} = \overline{X} \cdot \overline{Y} & \overline{X \cdot Y} = \overline{X} + \overline{Y} & \text{De Morgan} \end{array}$$

Nämä siis ovat suoraan laskusääntöjä, joita saa (ja pitää) käyttää digitaalisuunnittelussa. Näiden sääntöjen mukaan toimivat myös digitaalisuunnittelussa käytettävät portit, jotka esitellään myöhemmin. Monen muuttujan teoreemien viimeisellä rivillä esitetään ns. De Morganin kaavat. Ne käsittelevät siis komplementtia sellaisessa tapauksessa, jossa komplementoidaan kerralla enemmän kuin yksi muuttuja (yleensä lauseke tai osa siitä). De Morganin teoreema on tärkeä suunnittelussa, koska sen avulla voidaan saada erilaisia toteutuksia samalle loogiselle lausekkeelle. De Morganin teoreema voidaan yleistää $n \geq 2$ muuttujalle. (Jätetään lukijan pohdittavaksi, miten.)

Teoreemoista puhutaan paljon lisää myöhemmin, kun käsitellään kytKentäfunktioita. (KytKentäfunktio on siis lauseke, joka sisältää edellä esitettyjä funktioita ja joka voidaan toteuttaa lopulta esim. porttipiireillä.)

Porttipiirit

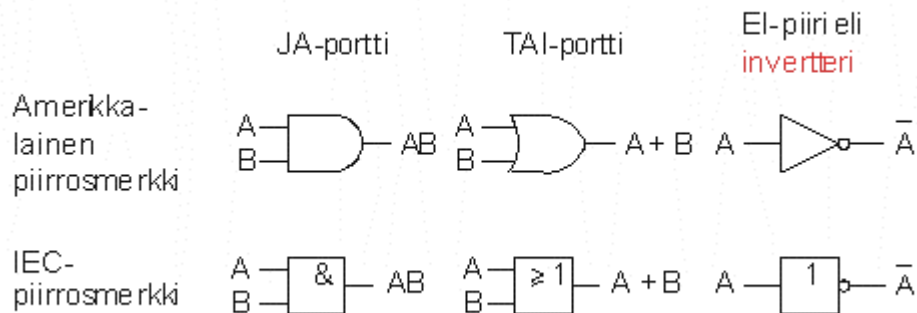
Loogiset funktiot ja lausekkeet toteutetaan yleensä sähköisillä piireillä. Tällä kurssilla tarkastellaan porttipiiritoteutuksia. Sähköisissä piireissä tietty jännitetaso vastaa tiettyä loogista tilaa.

Esimerkkinä CMOS-logiikkapiirin vastaavuudet:



Siis kaikki eri jännitteet 3,15 voltin ja 5 voltin välillä tulkitaan loogiseksi ykköseksi, kaikki alle 1,35 voltin jännitteet tulkitaan loogiseksi nollaksi. Jännitteellä on tietty sallittu vaihteluväli, koska olisi hankalaa ja kallista vaatia tasan tiettyä jännitettä. Virheiden eliminoimiseksi välillä on myös kielletty alue, joka pyrkii estämään vahingossa tilasta toiseen lipsuvan jännitteen.

Perusfunktioita (JA, TAI, EI) vastaavat omat porttipiirinsä, joiden piirrosmerkit on taulukoitu alle. Kurssilla pyritään käyttämään IEC standardin mukaisia merkkejä, mutta käytännön elämässä näkee paljon myös amerikkalaisen standardin mukaisia merkkejä.



Kun piirretään yksittäisiä portteja tai piirikaavioita laitetaan se, mitä portti ottaa sisäänsä (ottosignaali/-signaalit) vasemmalle puolelle ja antosignaali oikealle puolelle. Jos piirretään aikakaavioita, kulkee aika vasemmalta oikealle.

KytKentäfunktion kolme esitystapaa

Looginen funktio, totuustaulu ja piirikaavio ovat elementtejä, jotka kaikki vastaavat toisiaan ja tarvittaessa yhdestä esitystavasta voidaan johtaa muut kaksi. Looginen funktio lienee tässä vaiheessa tutuin.

Looginen funktio voi olla esimerkiksi $F = X + Y'Z$ tai $G = (D + E)(MKAY + PVALO) + B'$. Looginen funktio voidaan kytkentäalgebran säännösten mukaan muuntaa muodosta toiseen. Usein lausekkeen muokkaaminen ja yksinkertaistaminen kannattaa, koska näin saadaan aikaan yksinkertaisempi piiritoteutus.

Totuustaulu taas käy taulukkomuodossa läpi kaikki mahdolliset kombinaatiot muuttujista (input) ja osoittaa, mikä on funktion arvo (output) kullakin muuttujien kombinaatiolla. Alla on esimerkkinä loogisen funktion $F = X + Y'Z$ totuustaulu:

Input X	Input Y	Input Z	Output F
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	1

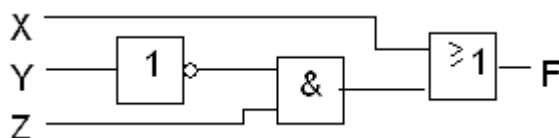
Ylläoleva totuustaulu on muodostettu loogisesta funktiosta seuraavasti:

Merkitään ensin sarakkeisiin X, Y ja Z kaikki mahdolliset kombinaatiot, joita on siis 8 erilaista (000 -> 111)

Tämän jälkeen merkitään F sarakkeeseen ykkönen kaikille riveille joissa toteutuu $Y=0$ **ja** $Z=1$. Tämä vastaa loogisen funktion lausekkeesta osaa $Y'Z$

Lopuksi merkitään vielä ykköset F sarakkeeseen kaikille riveille joissa toteutuu $X=1$. Näin on muodostettu loogista funktiota $X+Y'Z$ vastaava totuustaulu.

Piirikaavio on taas havainnollinen kuvaus funktion toteuttavasta piiristä. Piirikaaviota piirrettäessä muistetaan, että ottosignaalit tulevat yleensä vasemmalta ja anto laitetaan oikealle. (Monimutkaisissa piireissä tämä sääntö tosin voidaan unohtaa.) Piirikaavio voidaan piirtää suoraan funktion lausekkeesta. Esimerkkinä on piirretty lausekkeen $F = X + Y'Z$ piirikaavio. (Ensin siis suoritetaan komplementaatio Y:lle käyttäen EI-porttia, sitten Y':n ja Z:n JA-operaatio ($Y'Z$) ja lopuksi TAI-operaatio JA:n tuloksen ja X:n välillä ($X + Y'Z$)).



Signaalinkäsittelytekniikan laboratorio

Digitaalitekniikan perusteet - luento 4

Funktion sieventäminen ja käsittely

Mahdollisimman kannattavan piiritoteutuksen aikaansaamiseksi pyritään sieventämään looginen funktio yksinkertaisimpaan olemassa olevaan muotoonsa. Funktion voi sieventää joko totuustaulusta tai lausekkeesta. Lausekkeesta sieventäminen on suoraviivaisempaa. Siihen käytetään suoraan Boolean algebraa (eli kytkentäalgebraa). Tietokone apuvälineenä on suotavaa, mutta pienet lausekkeet voidaan hyvin sieventää käsin.

Sievennyksesimerkki $F = X'YX' + XZ'Y' + Z'XY$:

$$F = X'YX' + XZ'Y' + Z'XY$$

$$F = \mathbf{X'YX'} + XZ'Y' + Z'XY \\ = \mathbf{X'Y} + XZ'Y' + Z'XY \quad | \text{ perustelu: } AA=A$$

$$= X'Y + XZ'Y' + \mathbf{Z'XY} \\ = X'Y + XZ'Y' + \mathbf{XZ'Y} \quad | \text{ perustelu: } AB = BA \text{ (kommutatiivisuus sääntö)}$$

$$= X'Y + \mathbf{XZ'Y'} + \mathbf{XZ'Y} \\ = X'Y + \mathbf{XZ'(Y + Y')} \quad | \text{ perustelu: } AB + AC = A(B+C) \\ \text{(distributiivisuus sääntö)}$$

$$= X'Y + XZ'(\mathbf{Y + Y'}) \\ = X'Y + XZ' \quad | \text{ koska } A + A' = 1$$

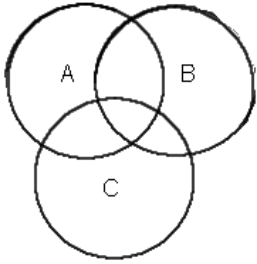
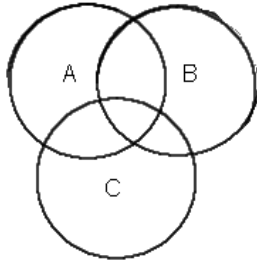
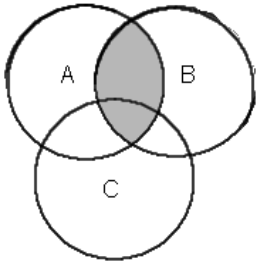
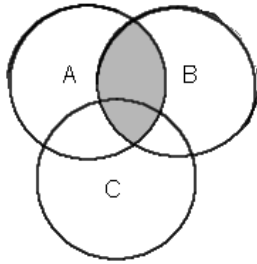
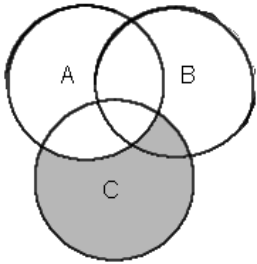
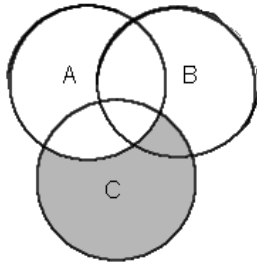
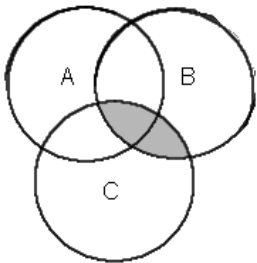
Kaikki sieventäminen perustuu suoraan "Kytkeäälgebran teoreemoja"-sivulla esitettyihin sääntöihin. Alle on koottu pari vinkkiä, joita tarvitaan usein:

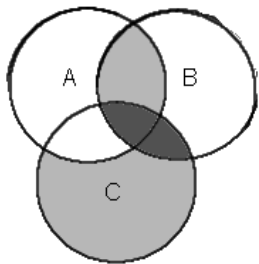
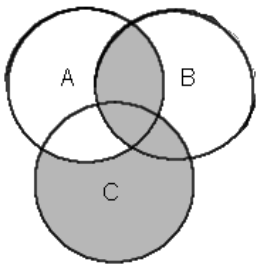
- sulut saa kertoa auki ja yhdistellä samoin kuin tavallisessa algebrassa
- $AA = A$ samoin kuin $AAAA = A$ (eli saman muuttujan monikerrat voi poistaa)
- $A + A'$ saa aina arvon yksi (koska $A + A'$ on aina $1 + 0$ tai $0 + 1$)
- AA' saa aina arvon nolla (koska AA' on aina $1*0$ tai $0*1$)

Konsensusteoreema

Konsensusteoreema on hyödyllinen lausekkeitä yksinkertaistettaessa. Se sanoo seuraavaa: $AB + A'C + BC = AB + A'C$. Eli vasemmanpuoleisen lausekkeen viimeinen termi on tarpeeton. Tämä siksi, että termi B aiheuttaa funktion ulostuloksi ykkösen silloin kuin A on yksi. Toisaalta termi C pakottaa funktion arvon ykköseksi silloin kun A on nolla. Tällöin nämä kaksi termiä sisältävät tapauksen BC .

Teoreema voidaan todistaa helposti joukko-opilla. Kuten joukko-opista on jo lukiossa opittu, vastaa kahden funktion JA-operaatio kahden joukon leikkausta (eli niiden yhteistä aluetta), sekä kahden joukon TAI-operaatio vastaa kahden joukon unionia (eli niiden molempien täyttämää aluetta yhdessä) Esimerkki selventää asiaa:

yhtälön vasen puoli				yhtälön oikea puoli	
joukko	kuva			joukko	kuva
lähtötilanne A, B, C				lähtötilanne A, B, C	
(leikkaus) AB				(leikkaus) AB	
(leikkaus) A'C				(leikkaus) A'C	
(leikkaus) BC				Tämä termi (BC) jää pois turhana, sillä se vain värjää toiseen kertaan jo värjätyn alueen.	

(unioni edellisistä leikkauksista) $AB + A'C + BC$ Huomataan, että tummennettu osa kuvassa on värjätty kaksi kertaa eli turhaan.		(unioni edellisistä leikkauksista) $AB + A'C$	
--	---	--	---

Näin ollen termin BC voi siis jättää pois, koska olemme saaneet määritellyä saman alueen ilmankin sitä.

Lausekkeen komplementointi

Toisinaan tarvitaan jonkin kokonaisen funktion komplementtia. Tällöin käytetään De Morganin teoreemaa yleisessä muodossa:

- komplementoidaan kaikki muuttujat ja
- vaihdetaan TAI operaatiot JA operaatioiksi ja päinvastoin.

Esimerkkinä komplementoidaan lauseke $F = AB' + C(A + B')$.

$$F = AB' + C(A + B')$$

$$F' = (A' + B)(C' + A'B) \text{ - eli TAI-operaatiot ovat muuttuneet JA-operatioiksi.}$$

Lisäksi kaikki muuttujat on komplementoitu (Mikäli muuttuja on valmiiksi komplementoitu muuttuu se normaaliksi uudelleen komplementoitaessa esim. $A' \rightarrow (komplementointi) \rightarrow A'' = A$)

Tämä voidaan johtaa myös peruskaavojen perusteella:

$$\begin{aligned}
 F' &= (AB' + C(A + B'))' \\
 &= (AB')'(C(A + B'))' \quad | \text{ perustelu: } (X+Y)' = X'Y' \\
 &= (A' + B'')(C' + (A + B')') \quad | \text{ perustelu: } (XY)' = X' + Y' \\
 &= (A' + B)(C' + A'B) \quad | \text{ perustelu: } X'' = X
 \end{aligned}$$

KytKentäfunktion perusmuodot

KytKentäfunktiolla on kaksi perusmuotoa: Tulojen summamuoto (Sum Of Products, SOP) ja summien tulomuoto (Product Of Sums, POS).

SOP esimerkki: $F(A,B,C) = A + A'B + BC'A$ Tässä yksittäinen termi on tulotermi.

POS esimerkki: $G(X, Y, Z) = (Y + X')(X + Z + Y')Z$ Tässä yksittäinen termi on summatermi.

Tulojen summamuodossa lasketaan siis tuloja yhteen kun taas summien tulomuodossa kerrotaan summia keskenään.

Käsitteitä:

- **Tulotermi**, jossa on mukana kaikki muuttujat (esim A, B ja C), joista kukin esiintyy vain kerran, on minimitermi (esim ABC).

- **Summatermi**, jossa on mukana kaikki muuttujat (esim A, B ja C), joista kukin esiintyy vain kerran, on maksimitermi (esim. $A+B+C$).
- N:llä muuttujalla on 2^N minimi- ja maksimitermiä.
- Jokainen totuustaulun rivi voidaan siis ilmoittaa minimi- tai maksimiterminä.

Taulukoidaan kolmen muuttujan totuustaulun minimi- ja maksimitermit:

Minimitermit		X	Y	Z	Maksimitermit	
tulotermi	symboli				symboli	summatermi
$X'Y'Z'$	m0	0	0	0	M0	$X+Y+Z$
$X'Y'Z$	m1	0	0	1	M1	$X+Y+Z'$
$X'YZ'$	m2	0	1	0	M2	$X+Y'+Z$
$X'YZ$	m3	0	1	1	M3	$X+Y'+Z'$
$XY'Z'$	m4	1	0	0	M4	$X'+Y+Z$
$XY'Z$	m5	1	0	1	M5	$X'+Y+Z'$
XYZ'	m6	1	1	0	M6	$X'+Y'+Z$
XYZ	m7	1	1	1	M7	$X'+Y'+Z'$

Minimi- ja maksimitermien idea

- Yksi tietty minimitermi vastaa tarkalleen yhtä (ja vain yhtä) binäärimuuttujien kombinaatiota totuustaulussa. Esim. minimitermin $X'YZ$ binäärikombinaatio on 011
- Minimitermi saa arvon 1 *kyseisellä* binäärimuuttujien kombinaatiolla ja kaikilla muilla kombinaatioilla sen arvo on aina nolla.
(Tämä perustuu JA-operaatioon, eli esim. voidaan kysyä milloin minimitermi $X'YZ = 1$. Nähdään, että se on 1 jos ja vain jos $X=0$, $Y=1$ ja $Z=1$ ---> Siis kombinaatiolla 011. Lisäksi tämä on ainoa kombinaatio, jolla $X'YZ=1$.)
- Maksimitermi taas saa aina arvokseen ykkösen, paitsi sillä rivillä, joka vastaa sen muuttujien binäärikombinaatiota. Tällöin se saa arvon nolla.
Esim. voidaan kysyä milloin maksimitermi $X+Y+Z'$ saa arvon 0. Tämä toteutuu binäärikombinaatiolla 001, ja vain tällä binäärikombinaatiolla.
(Tämä perustuu puolestaa TAI-operaatioon. Mikäli binäärikombinaatio on mikä tahansa muu kuin 001, saa maksimitermi arvokseen 1)

Minimi- tai maksimitermein määritelty funktio on aina oikein. Se on vain yleensä turhan monimutkainen sellaisenaan. Voidaan huomata, että minimi- ja maksimitermit vastaavat siis totuustaulun rivejä. Funktio voidaan esittää minimitermeillä ilmoittamalla termit, jotka vastaavat ykkösrivejä. (Rivejä, jolloin funktion arvo on yksi.) Vastaavasti voidaan funktio esittää maksimitermeillä, jolloin termit vastaavat nollarivejä.

Esimerkki minimi- ja maksimitermien käytöstä

Esitellään funktio F , jonka totuustaulu näyttää seuraavalta:

Minimitermit		X	Y	Z	F	Maksimitermit	
tulotermi	symboli					symboli	summatermi
$X'Y'Z'$	m0	0	0	0	1	M0	$X+Y+Z$
$X'Y'Z$	m1	0	0	1	0	M1	$X+Y+Z'$
$X'YZ'$	m2	0	1	0	0	M2	$X+Y'+Z$
$X'YZ$	m3	0	1	1	1	M3	$X+Y'+Z$
$XY'Z'$	m4	1	0	0	1	M4	$X'+Y+Z$
$XY'Z$	m5	1	0	1	0	M5	$X'+Y+Z'$
XYZ'	m6	1	1	0	1	M6	$X'+Y'+Z$
XYZ	m7	1	1	1	1	M7	$X'+Y'+Z'$

Tässä siis 'F' sarakkeen arvot ovat vain keksitty opettajan päästä ja niitä ei siis ole johdettu mistään.

Nyt totuustaulun mukainen funktio F voidaan esitellä minimitermien summana:

$F = m_0 + m_3 + m_4 + m_6 + m_7$ (Tässä on siis etsitty ne rivit, joilla funktio F saa arvon 1)
 $= X'Y'Z' + X'YZ + XY'Z' + XYZ' + XYZ$ Tästä taas jatketaan funktion sieventämistä normaalisti Boolean algebran mukaan.

Maksimitermien tulona sama funktio F ilmaistaisiin:

$F = M_1 M_2 M_5$ (Tässä on siis etsitty ne rivit, joilla funktio F saa arvon 0)
 $= (X + Y + Z')(X + Y' + Z)(X' + Y + Z')$ Tästä taas jatketaan funktion sieventämistä normaalisti Boolean algebran mukaan.

Ylläolevista kahdesta eri muodosta voidaan huomata, että sama funktio voi olla yksinkertaisempi jommallakummalla tavalla esitettynä. Tässä tapauksessa se on yksinkertaisempi maksimitermien tulona esitettynä, sillä siinä on vain kolme eri summatermiä kun minimitermien summamuodossa tulotermejä on 5 kappaletta.

Jos funktio toteutetaan perusmuodossaan (SOP tai POS), saadaan sille aina kahden tason piiritoteutus. Tämä tarkoittaa, että portteja löytyy vain kahdessa rivissä. Esimerkiksi SOP toteutuksessa s.e. ensin tehdään kaikki JA -operaatiot ja sitten näiden tuloksille tehdään yhteinen TAI -operaatio. Standardimuoto ei kuitenkaan aina tuota optimaalisia ratkaisuja. Vähemmälläkin porteilla voidaan selvittää. Tällöin ratkaisusta tulee useampi tasoinen. Muutama ylimääräinen taso ei haittaa, mutta tietyt ongelmat kasaantuvat, jos portteja sijoitetaan paljon peräkkäin (saman kellojakson sisään):

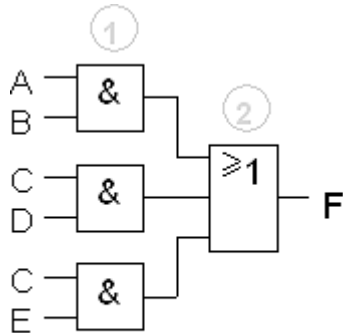
- etenemisviive kasvaa
- virhepulssiriskit kasvavat
- hankalia suunnitella, hankalia toteuttaa

Esimerkki 2- ja 3-tasoisesta piiritoteutuksesta samalle funktiolle

Esitellään funktio F , jota määrittävät muuttujat A , B , C ja D :

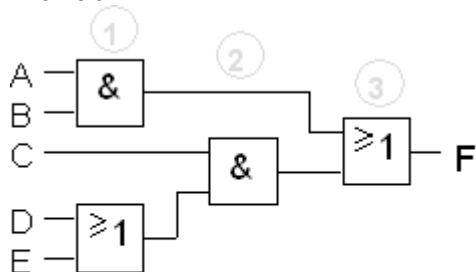
$F(A,B,C,D) = AB + CD + CE = AB + C(D + E)$ Lausekkeet ovat vastaavia, jälkimmäisessä yhdistetty kaksi viimeistä termiä.

2-tasoinen piiritoteutus saadaan vasemmanpuoleisen lausekkeen mukaan:



$$F = AB + CD + CE$$

3-tasoinen piiritoteutus saadaan oikeanpuoleisen lausekkeen mukaan:



$$F = AB + C(D + E)$$

Tässä tapauksessa ero ei ole merkittävä. Käytännössä kannattaa yleensä tarkistaa, että saman kellojakson sisään ei tule yli kuutta tai seitsemää porttia peräkkäin. (Tai montako sitten piirivalmistaja lupaakaan mahdollistaa samaan kellojaksoon.)

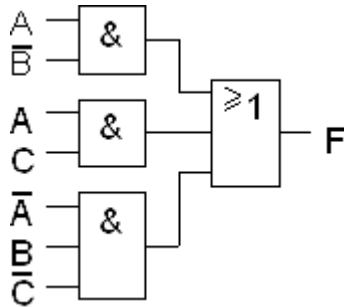
Esimerkki SOP- ja POS-muotoisesta toteutuksesta samalle funktiolle

Kertauksena: SOP = Sum Of Products = Tulojen summa
POS = Product Of Sums = Summien tulo

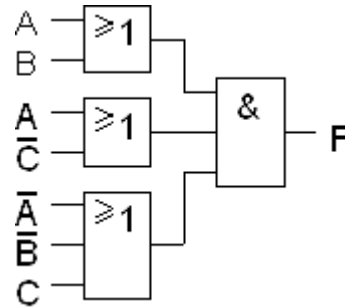
Otetaan esimerkiksi funktio, joka on määritelty: $F(A,B,C) = AB' + AC + A'BC'$
 $= (A + B)(A + C')(A' + B' + C)$

Vasemmanpuoleinen lauseke on F esitettyä SOP -muodossa, oikeanpuoleinen on sama funktio POS -muodossa. (Johto esitetään sivun alalaidassa.)

SOP -toteutus SOP -lausekkeen mukaan: ensin JA-operaatiot, sitten TAI-operaatiot



POS toteutus POS -lausekkeen mukaan: ensin TAI-operaatiot, sitten JA -operaatiot



Tässä tapauksessa toteutukset ovat yhtä yksinkertaisia

Bonusena lausekkeen johto SOP -muodosta POS -muotoon. Yleissivistystä

- yksinkertaistetaan SOP lauseketta, jos mahdollista
- komplementoidaan koko lauseke
- kerrotaan sulut auki
- yksinkertaistetaan lauseke, jos mahdollista
- komplementoidaan koko lauseke uudelleen

Esimerkkitapauksessa muunnoksen voi tehdä näin:

$$F(A,B,C) = AB' + AC + A'BC' \quad \text{komplementoidaan } F, \text{ jolloin saadaan } F'$$

$$F' = (AB' + AC + A'BC')' \quad \text{Käytetään De Morganin sääntöjä.}$$

$$F' = (A' + B)(A' + C')(A + B' + C) \quad \text{kerrotaan sulut auki}$$

$$= A'A + A'B' + A'C + BC'A + BCB' + BC'C \quad \begin{array}{l} \text{sievennetään lauseke} \\ \text{kytkentäalgebran säännöillä} \end{array}$$

$$= A'B' + A'C + BC'A \quad \begin{array}{l} \text{komplementoidaan } F' \text{ takaisin } F\text{:ksi De Morganin} \\ \text{sääntöjen mukaan} \end{array}$$

$$F = (A + B)(A + C')(A' + B' + C) \quad \text{Haluttu POS -lauseke}$$

Bonus 2: lausekkeen johto POS -muodosta SOP -muotoon. (Yksinkertaisempaa kuin edellinen)

- yksinkertaistetaan POS -lauseke, jos mahdollista
- kerrotaan sulut auki
- yksinkertaistetaan saatu SOP -lauseke, jos mahdollista

Karnaugh'n kartta

Eräs tehokas tapa sieventää funktioita on Karnaugh'n kartta. Se perustuu samaan ajatukseen, kuin sieventäminen suoraan totuustaulusta, mutta on helpommin hahmotettavissa. Kartta on joukko ruutuja, jotka vastaavat täsmälleen yhtä riviä totuustaulussa. Kunkin ruudun arvoksi tulee totuustaulun kyseisellä rivillä olevan funktion F arvo. Karnaugh'n kartan käyttöalue on 3-6 muuttujaa. 5-6 muuttujan kartat ovat jo suhteellisen hankalia. Tätä suurempiin muuttujamääriin suosittelemme tietokoneavusteista sieventämistä.

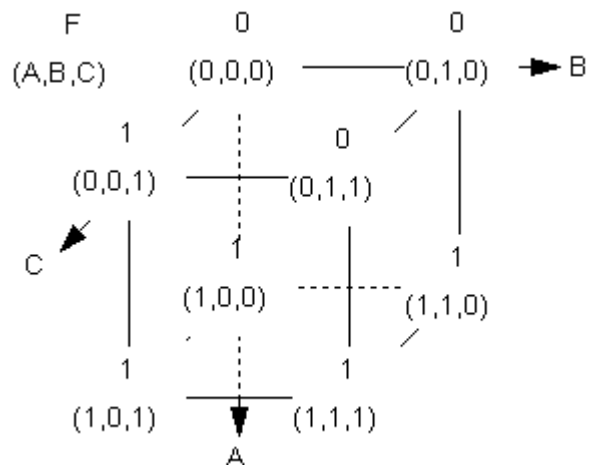
Tutkitaan aluksi kolmen muuttujan totuustaulua:

rivi-nro	Input X	Input Y	Input Z	Output F
0.	0	0	0	0
1.	0	0	1	1
2.	0	1	0	0
3.	0	1	1	0
4.	1	0	0	1
5.	1	0	1	1
6.	1	1	0	1
7.	1	1	1	1

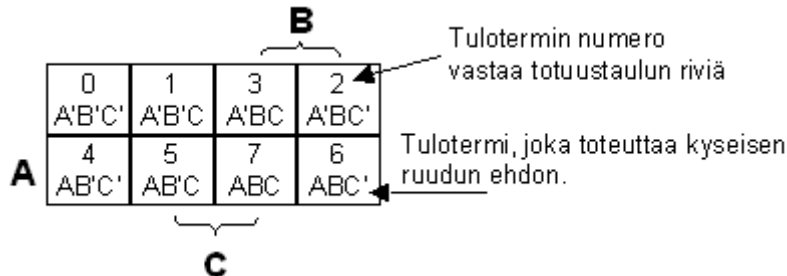
Minimitermien perusteella lausekkeeksi saataisiin $F = X'Y'Z + XY'Z' + XY'Z + XYZ' + XYZ$. Oikea muoto, mutta myös valitettavan hankala ja kaipaa paljon sievennystä. Totuustaulusta voidaan nähdä, että jos funktio saa arvon 1 kahdella sellaisella rivillä, joiden suhteen eroa on vain yhden muuttujan osalta, voidaan tämä muuttuja poistaa termistä. Esimerkiksi rivit 6. ja 7.: Vain Z muuttuu eli se voidaan poistaa: $XYZ' + XYZ = XY(Z' + Z) = XY$.

Itseasiassa lauseke voidaan sieventää muotoon $F = X + Y'Z$. Tämä olisi kuitenkin suhteellisen hankalaa taulusta lukemalla tai laskusääntöjä käyttäen.

Karnaugh'n kartta on silmämääräisesti havainnollisempi. Se on kehitetty siitä ajatuksesta, että jos yksi muuttuja vaihtelee funktion arvon sekä muiden pysyessä vakiona, voidaan kyseinen yksi muuttuja jättää huomiotta. Tätä on helppo tarkastella sijoittamalla muuttujat moniulotteiseen koordinaatistoon, jossa jokainen muuttuja pysyy vakiona omalla sivullaan. Särmää pitkin kuljettaessa taas vain yksi muuttuja muuttuu kerrallaan. Esimerkiksi kuten kuvassa oikealla (Funktion arvo kulloisessakin kuution kulmassa on merkattu kulman päälle.):



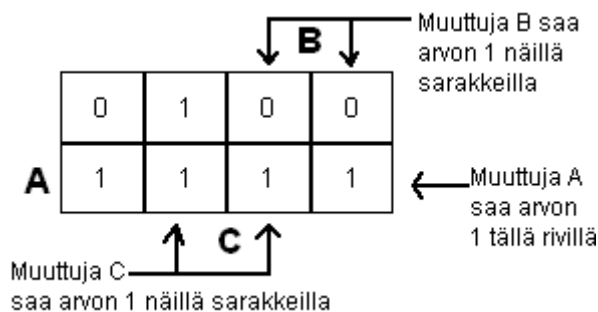
Havainnollisempi esitystapa, mutta silti suhteellisen hidas ja suuritöinen. Nopeampaan ja yksinkertaisempaan päästään, kun vielä leikellään kuutio ja litistetään se 2-ulotteiseksi. Tällöin täytyy vain muistaa, että **reunapalat ovat edelleen vierekkäisiä**. Lopputulos on ruudukko, jota sanotaan Karnaugh'n kartaksi:



Karttaa tulkitaan siten, että jokainen ruutu vastaa yhtä totuustaulun riviä, eli kyseisen rivin toteuttavaa minimitermiä. Ylläolevassa kuvassa termit on kirjoitettu näkyviin. Huomata kannattaa, että termit 2 ja 3 sekä 6 ja 7 ovat vaihtaneet paikkaa. Tämä on tarpeellista siksi, että sillä saadaan eri muuttujille yhtenäiset alueet, jossa ne ovat vakioita. Siis kun Karnaugh'n kartalla liikutaan ruudusta viereiseen, vain yksi muuttuja vaihtaa arvoaan.

Taulussa A -muuttuja saa arvon 1 koko alarivillä. B-muuttuja taas kahdella oikeanpuoleisella sarakkeella. C:n ykkösalue on kaksi keskimmäistä saraketta. Ruutuihin merkataan funktion arvo, joka vastaa siis funktion arvoa samannumeroisella rivillä totuustaulussa. Viereisessä pienessä kuvassa on vielä kartan ympärille piirretty muuttujien arvot. (A:n arvot vaihtelevat vaakariveittäin, B:n ja C:n pystyriveittäin.)

A \ BC				
	00	01	11	10
0	F0	F1	F3	F2
1	F4	F5	F7	F6



On jo huomattavasti helpompi havaita, että funktio saa arvon 1 aina, kun muuttuja A saa arvon 1. Tämän lisäksi taulusta nähdään, että funktio saa arvon 1, kun $C = 1$ ja $B' = 1$.
 $F = A + B'C$

Esimerkkitaulumme kartta näyttäisi siis seuraavalta:

3, 4 tai 5 muuttujaa

Edellisellä sivulla esitelty kartta on tarkoitettu kolmelle muuttujalle. Neljän muuttujan kartta on kokonaan neliö, jossa on 16 ruutua. Ruudut vastaavat neljän muuttujan totuustaulun rivejä. (Neljällä muuttujalla saadaan aikaan 16 eri kombinaatiota.) Kannattaa muistaa, että rivi-, ruutu- tai terminumerointi aloitetaan aina nollasta. Ohessa neljän muuttujan karnaugh'n kartta:

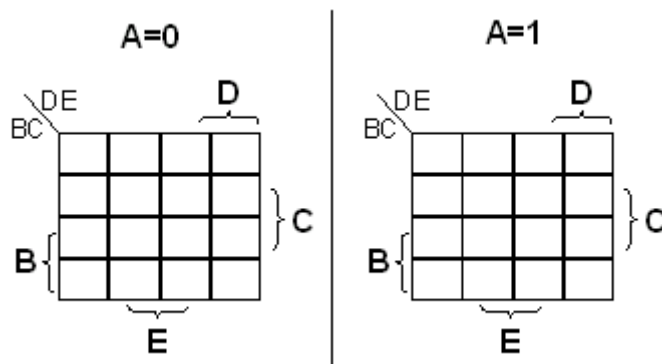
		C				
		00 01 11 10				
A	B	00	0	1	3	2
		01	4	5	7	6
	11	12	13	15	14	
	10	8	9	11	10	
		D				

Kartan vasemmassa yläreunassa esitellään kartan muuttujat A, B, C, D. Nähdään, että A ja B hallitsevat rivejä: A saa arvon yksi kahdella alimmaisella rivillä, B kahdella keskimmaisella. C ja D hallitsevat sarakkeita: C = 1 kahdella oikeanpuoleisista sarakkeista, D kahdella keskimmaisella. Muuttujan nimi merkataan niiden rivien tai sarakkeiden viereen, joilla se saa arvon 1.

Ruutujen numerointi vastaa totuustaulun rivejä (= minimitermien järjestysnumero). Nähdään, että rivien numeroiden paikkaa on vaihdettu (2 oikeanpuoleista saraketta keskenään sekä 2 alinta riviä keskenään). Tämä tehdään, jotta muuttujille saataisiin mahdollisimman suuret vakioalueet. (Alue, jolla muuttujan arvo pysyy vakiona.)

Kun vakioalueet ovat suuret, on helpompi silmämääräisesti erottaa kartasta alueita, joissa tietyllä muuttujalla on vakioarvo. Kun ruudut sisältävät funktion arvon, voidaan nähdä, mistä muuttujista mikin funktion arvo määräytyy. (Esim. jos kartan kaksi alinta riviä olisivat ykkösiä ja muut nollija, määräytyisi funktion arvo suoraan muuttujan A mukaan. $F = A$)

Viisi muuttujaa Viisi muuttujaa vaatii lisää ulottuvuuksia. Asia hoidetaan siten, että tarkastellaan kahta taulua. Jos käytetään muuttujia A, B, C, D, E, asetetaan näistä eniten merkitsevä (A) s.e. se saa arvon 0 ensimmäisessä taulussa ja arvon 1 toisessa taulussa. Kun tutkitaan funktion käyttäytymistä, kuvitellaan taulut päällekkäin:



Viidellä muuttujalla saadaan 2^5 kombinaatiota. Tämä vastaa 32 totuustaulun riviä. Rivit numeroidaan s.e. vasemmanpuoleinen kartta vastaa rivejä 0-15 ja oikeanpuoleinen 16-31.

Karnaugh'n kartan käyttö esimerkin avulla

Kartalla pyritään täsmälleen samaan kuin totuustaulun suhteen: muodostamaan mahdollisimman yksinkertainen lauseke, jolla pystytään määrittelemään funktion arvo. Kartasta katsomalla siis haetaan yksinkertaisia ehtoja muuttujille, jotta funktion arvo on haluttu kaikilla mahdollisilla muuttujakombinaatioilla. Kartta toimii siis oikopolkuna totuustaulusta yksinkertaiseen funktion määrittelevään lausekkeeseen.

Käytännössä tämä tapahtuu kalastamalla kartasta alueita, jotka voidaan määritellä yhdellä tai useammalla muuttujalla tai näiden komplementeilla.

Tutkitaan kartan käyttöä neljällä muuttujalla. Otetaan esimerkkifunktio F, joka riippuu neljästä muuttujasta A,B,C ja D ja jolle on määritetty seuraavanlainen totuustaulu:

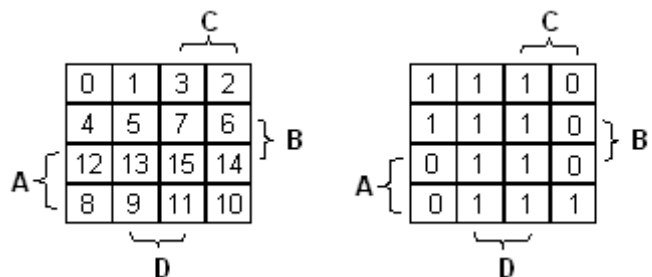
rivi-nro	Input A	Input B	Input C	Input D	Output F
0.	0	0	0	0	1
1.	0	0	0	1	1
2.	0	0	1	0	0
3.	0	0	1	1	1
4.	0	1	0	0	1
5.	0	1	0	1	1
6.	0	1	1	0	0
7.	0	1	1	1	1
8.	1	0	0	0	0
9.	1	0	0	1	1
10.	1	0	1	0	1
11.	1	0	1	1	1
12.	1	1	0	0	0
13.	1	1	0	1	1
14.	1	1	1	0	0
15.	1	1	1	1	1

Ensimmäiseksi sijoitetaan funktion arvot Karnaugh'n karttaan. Muistin virkistämiseksi vasemalla alla esitetään mikä totuustaulun rivi tulee mihinkin ruutuun. (Sääntö: muuten vasemmalta oikealle, ylhäältä alas; paitsi kaksi oikeanpuoleista saraketta sekä kaksi alinta riviä vaihtavat keskenään.

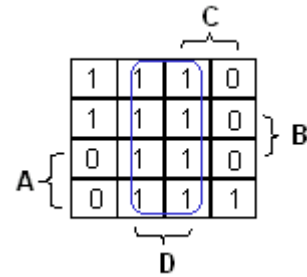
! HUOM ! Edellä mainittu sääntö rivien sijoittamisesta ruutuihin pätee ainoastaan, mikäli muuttujat (nyt A,B,C ja D) määrittävät juuri ne alueet kuin esimerkeissä on näytetty.

Toisinsanoen, mikäli menet vaihtamaan Karnaugh'n kartassa esim. B:n ja C:n alueiden paikkoja, ei edellämainittu sijoittelusääntö enää päde.)

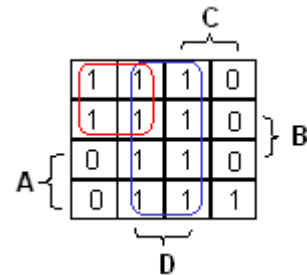
Oikeanpuolimmaisessa kuvassa funktion arvot on sijoitettu paikoilleen.



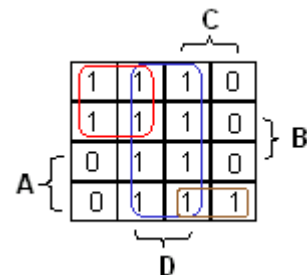
Tarkastellaan karttaa ja etsitään lauseketta funktiolle. Perussääntö on, että mitä suurempi yhtenäinen alue voidaan saavuttaa, sen parempi, sillä sitä enemmän lauseke yksinkertaistuu. Nähdään, että funktio saa arvon 1 kahdella pystyrivillä. Näillä muuttujia D saa myös arvon 1. Voidaan valita rivit suoraan, jolloin saadaan funktion ensimmäinen termi: $F = D + ..$



Jatketaan vaikka vasemman yläkulman ykkösillä. Kaksi ykköstä voisi valita yksinään, mutta se ei ole kannattavaa: **mitä suurempi alue saadaan valittua, sitä vähemmän muuttujia tarvitaan termiin.** (Mitä vähemmän muuttujia sitä yksinkertaisempi toteutus.) Toisaalta **samoja ruutuja voi valita uudelleen.** (Ei siis haittaa, jos funktio saa arvon yksi useammalla kuin yhdellä perusteella.) Näillä perustein voidaan ryhmitellä neljä vasemman yläreunan ruutua alueeksi ja antaa näille peruste $A'C'$.



Jäljelle jää enää siis yksi kiusallinen ykkösrutu oikeaan alanurkkaan. Jos tähän viitataan yksinään, vaatii se kaikkien neljän muuttujan käytön: $AB'CD'$. (Voi helposti todeta, että termi on juuri tuo esim. rengastamalla alueet A, B', C, D' ja katsomalla, että ruutu jäi ulkopuolelle.) Kannattaa yhdistää ruutu viereisen kanssa, jolloin saadaan kolmen muuttujan termi: $AB'C$.



Näin ollen funktion lausekkeeksi saadaan $F = D + A'C' + AB'C$.

Karnaugh'n kartan alueiden valitseminen

Sääntöjä

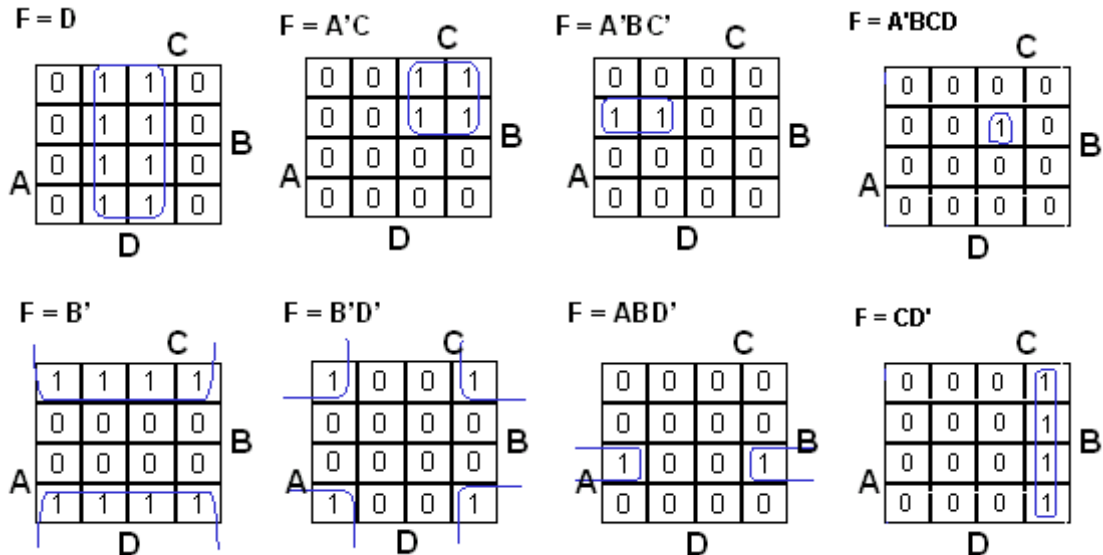
- Valittu alue voi olla vain suorakaide, neliö tai jopa yksittäinen ruutu jos sitä ei saa yhdistettyä esim. reunan yli.
- Kartan reunaruudut tulkitaan vastakkaistensa kanssa vierekkäisiksi. (Alueen voi muodostaa s.e. se menee yhden reunan yli ja jatkuu toisella.)
- Alueen koon täytyy olla joku kakkosen potenssi (1, 2, 4, 8, 16..)
- Alue ei voi saada mutkaa tai kulkea viistoon.

Muita huomattavia seikkoja:

- Mitä suurempi alue, sitä vähemmän muuttujia tarvitaan kuvaamaan sitä. --> Pyritään siis suuriin alueisiin.
- Samoja ruutuja **voi** (ja usein kannattaakin) valita moneen kertaan.
- Ei kuitenkaan valita ruutuja uudelleen, jollei sillä saada kalastettua yhtään kokonaan uutta ruutua.
- Jos ryhmä sisältää ruutuja, joista osa kuuluu vain tähän ryhmään, sanotaan ryhmän muodostamaa termiä **olennaiseksi perustermiksi**.

- Jos ryhmä sisältää vain ruutuja, jotka kuuluvat muihinkin ryhmiin, sanotaan ryhmän muodostamaa termiä **perustermiksi**.
- Kannattaa muodostaa vain ryhmiä, joista saadaan olennaisia perustermejä.

Esimerkkejä ryhmistä ja niiden muodostamista termeistä:



Epätäydellinen kytkentäfunktio

Toisinaan kytkentäfunktion arvolla ei ole merkitystyä tietyllä muuttujakombinaatiolla. Tällainen tapaus on laite, jossa ei koskaan voi esiintyä tiettyä kombinaatiota tai laite, jonka toimintaan tietyt kombinaatiot eivät vaikuta. Kohtaa vastaavaa minimitermiä kutsutaan hälläväliä-termiksi (don't care term).

Jos jonkin kombinaation arvo on jätetty määrittelemättä, sanotaan funktiota epätäydelliseksi. Funktion arvoksi asetetaan totuustaulun kyseiseen kohtaan X. Jos kytkentäfunktio määritellään minimitermien avulla, täytyy erikseen ilmaista X -termit.

Karnaugh'n karttaan kyseiset kohdat merkitään X:llä. Ruudut voi tulkita joko ykköseksi tai nolllaksi riippuen, kumpi tapaus on edullisempi. **HUOM: Sama ruutu voi olla vain ykkönen tai nolla. Ei siis voida rengastaa kahta aluetta puoliksi päällekkäin ja tulkita toisessa ruutu ykköseksi ja toisessa nolllaksi.** (Tämä siksi, että lausekkeena määritelty kytkentäfunktio on yksikäsitteinen: funktiolla on aina jokin arvo.)

Esimerkki kolmen muuttujan epätäydellisestä määrittämisestä ja sen johtamisesta lausekkeeksi:

Input A	Input B	Input C	Output F
0	0	0	1
0	0	1	X
0	1	0	1
0	1	1	1
1	0	0	0
1	0	1	X
1	1	0	1
1	1	1	0

Piirretään Karnaugh'n kartta:

	B			
	1	X	1	1
A	0	X	0	1
	C			

Ryhmitellään ja muodostetaan lauseke

Toteutuu ykkösenä

	B			
	1	X	1	1
A	0	X	0	1
	C			

Toteutuu nollana

$F = A' + BC'$

Karnaugh'n kartan käyttö, kun halutaan SOP -lauseke

(SOP lauseke on siis Sum Of Products eli tulojen summa.)

Säännöt, joiden mukaan edetään:

1. Laaditaan totuustaulu speksien mukaan, jollei totuustaulua ole annettu jo valmiiksi.
2. Piirretään totuustaulua vastaava Karnaugh'n kartta eli tarkistetaan muuttujien määrä ja tehdään sen mukaan oikean kokoinen kartta
3. Siirretään totuustaulusta ykköset karttaan rivejä vastaaviin ruutuihin
4. Vierekkäisistä ykkösistä muodostetaan mahdollisimman suuria ryhmiä (joiden koot ovat kakkosen potensseja), kunnes jokainen ykköseen kuuluu johonkin ryhmään. (Saa kuulua moneen.)
5. Muodostetaan ryhmiä vastaavien tulotermien looginen summa. Se on yksinkertaisin SOP toteutus käytetylle totuustaululle. (Mikäli ruudut on ryhmitelty oikein.)

Kannattaa muistaa, että saatu esitys ei välttämättä ole ainoa oikea.

1. Laaditaan totuustaulu speksien mukaan, jollei totuustaulua ole annettu jo valmiiksi.
2. Piirretään totuustaulua vastaava Karnaugh'n kartta eli tarkistetaan muuttujien määrä ja tehdään sen mukaan oikean kokoinen kartta
3. Siirretään totuustaulusta ykköset karttaan rivejä vastaaviin ruutuihin ja tarkastellaan kartan nollia.
4. Muodostetaan vierekkäisistä **nolla-alueista** mahdollisimman suuria alueita, samaan tapaan kuin SOP-tapauksessa haettiin **ykkösiä**
5. Muodostetaan lauseke kuten SOP-tapauksessakin. **Huom ! Tämä muodostettu lauseke on nyt funktion komplementoitu lauseke siis F'**

6. Muodostetaan summatermien looginen tulo: Tämä saadaan muodostettua komplementoimalla F' De Morganin säännöillä. Tällöin saadaan POS-muotoinen lauseke

Kannattaa muistaa, että saatu esitys ei välttämättä ole yksikäsitteinen.

Esimerkkinä POS lausekkeen muodostaminen neljällä muuttujalla.

Käytetään samaa totuustaulua kuin POS -tapauksessa.

rivi-nro	Input A	Input B	Input C	Input D	Output F
0.	0	0	0	0	1
1.	0	0	0	1	0
2.	0	0	1	0	1
3.	0	0	1	1	1
4.	0	1	0	0	1
5.	0	1	0	1	1
6.	0	1	1	0	0
7.	0	1	1	1	1
8.	1	0	0	0	1
9.	1	0	0	1	0
10.	1	0	1	0	1
11.	1	0	1	1	1
12.	1	1	0	0	0
13.	1	1	0	1	1
14.	1	1	1	0	0
15.	1	1	1	1	1

				C
	1	0	1	1
	1	1	1	0
A	0	1	1	0
	1	0	1	1
				D

Muodostetaan Karnaugh'n kartta: (Tämä osuus on siis täysin sama kuin SOP toteutuksessa.)

				C
	1	0	1	1
	1	1	1	0
	0	1	1	0
A	1	0	1	1
				D

$$F' = ABD' + B'C'D + BCD'$$

$$F = (A' + B' + D)(B + C + D')(B' + C' + D)$$

Muodostetaan nolla-alueet ja niiden perusteella funktion lauseke (De Morganoimalla nolliä ympäröimällä saatu F' :n lauseke):

Logiikkapiirin suunnittelu

Loppukevennyksenä pientä ohjeistusta käytännön suunnitteluun. Lista ei suinkaan ole kattava, mutta jotain sinne päin.

1. Määritellään tehtävä ja mitä halutaan.
2. Päätetään, mitä ja miten paljon muuttujia käytetään. (Ja miten niiden tulee vaikuttaa piirin antoon.) **Nämä ovat piirin speksit**
3. Laaditaan totuustaulu tai -taulut
4. Muodostetaan sievennetyt kytkentäfunctiot. (Karnaugh'ta apuna käyttäen tai ilman.)
5. Suunnitellaan piiri käytettävissä olevista peruspiireistä.
6. Testataan suunnitellun piirin toiminta.
 - o jos piiri toimii, se on valmis
 - o jos se ei toimi, etsitään virhe ja rakennetaan piiri uudestaan ja testataan. Käydään näitä vaiheita läpi yhä uudelleen, kunnes piiri toimii tai projekti lopetetaan tuloksettomana.

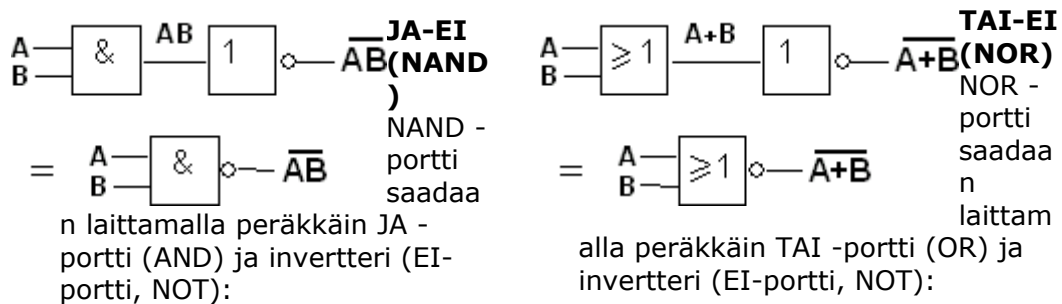
Signaalinkäsittelytekniikan laboratorio

Digitaalitekniikan perusteet - luento 5

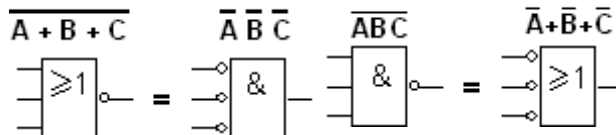
JA-EI (NAND), TAI-EI (NOR) portit

Aikaisemmat piirikomponentit JA, TAI, EI (AND, OR, NOT) kannattaa tarvittaessa kerrata [Loogiset perusfunktiot -sivulta](#). Tällä sivulla esitellään kaksi uutta porttia, jotka ovat hyvinkin käteviä piirisuunnittelussa. Uudet portit eivät kuulu perusporttien luokkaan siksi, että ne voidaan muodostaa yhdistelemällä perusportteja keskenään. Portteja käytetään esimerkiksi koska pelkkiä perusportteja monipuolisemmalla porttivalikoimalla on helpompi suunnitella. Myös komponenttien hinnassa on huomattavaa, että perusporteista johdetut portit voivat toisinaan olla alkuperäisiä perusportteja halvempia valmistustekniikoista riippuen.

NAND ja NOR -portit ovat siitä erityisiä, että niillä pystytään yksinään rakentamaan mikä tahansa logiikkaportteilla toteutettavissa oleva digitaalisysteemi. (De Morganin sääntöjä käyttäen lausekkeet voidaan aina muokata muotoon, jossa on joko pelkkiä NAND -operaatioita tai pelkkiä NOR -operaatioita.)

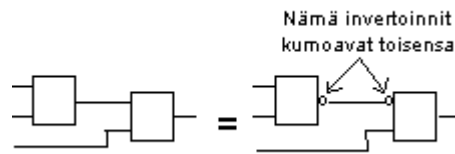


De Morganin kaavojen graafisia vastineita:



NAND- ja NOR -porteilla voidaan siis 'oikaista' kaavoja ja yksinkertaistaa funktioita. Muuta mielenkiintoista kyseisissä porteissa on, että niillä voidaan havainnollistaa myös kytkentäalgebran teoreemoja. Pari esimerkkiä seuraavalla sivulla:

$$\overline{\overline{A}} = A: \quad A \rightarrow \boxed{1} \rightarrow \overline{A} \rightarrow \boxed{1} \rightarrow \overline{\overline{A}} = A$$



Esimerkki NAND toteutuksesta loogiselle funktiolle

Annettuna on SOP -muotoinen funktio $F = B'C + CD + AB'D$

$F = B'C + CD + AB'D$ Kytkeäalgebran perusteella voidaan sanoa, että $F'' = F$. Tästä saadaan:

$$F'' = (B'C + CD + AB'D)''$$

$$F = (B'C + CD + AB'D)''$$

Edelleen voidaan sieventää De Morganin avulla, koska $(A + B + C + \dots + K)' = A'B'C'..K'$

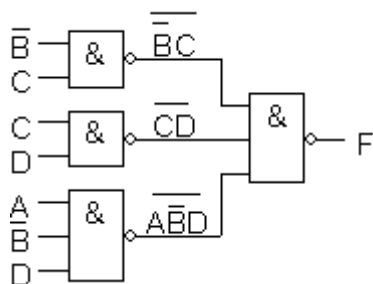
$F = ((B'C)'(CD)'(AB'D)')'$ Tämä lauseke voidaan toteuttaa pelkillä NAND -porteilla.

NAND -toteutus muokatulle lausekkeelle: $F = ((B'C)'(CD)'(AB'D)')'$

Kannattaa huomata, että jos komplementit ilmaistaan viivalla, eikä pilkulla, näyttää lauseke tältä:

$$F = \overline{\overline{B}C} \cdot \overline{CD} \cdot \overline{ABD}$$

(Kyse on kuitenkin täsmälleen samasta funktiosta.)



Esimerkki NOR toteutuksesta loogiselle funktiolle

Annettuna on POS -muotoinen funktio $F = C(A' + B')(A + B)$

$F = C(A' + B')(A + B)$ Kytkeäalgebran perusteella voidaan sanoa, että $F'' = F$. Tästä saadaan:

$$F'' = (C(A' + B')(A + B))''$$

$$F = (C(A' + B')(A + B))''$$

Edelleen voidaan sieventää De Morganin avulla, koska $(A + B + C + \dots + K)' = A'B'C'..K'$

$F = (C' + (A' + B')' + (A + B)')'$ Tämä lauseke voidaan toteuttaa pelkillä NOR-porteilla.

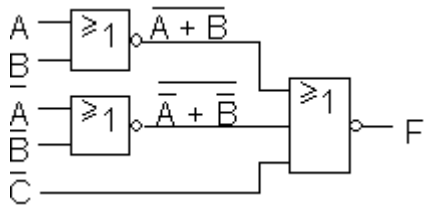
NOR -toteutus muokatulle lausekkeelle:

$$F = (C' + (A' + B')' + (A + B)')'$$

Kannattaa huomata, että jos komplementit ilmaistaan viivalla, eikä pilkulla, näyttää lauseke tältä:

$$F = \overline{\overline{C} + (\overline{A} + \overline{B}) + (A + B)}$$

(Kyse on kuitenkin täsmälleen samasta funktiosta.)



EHDOTON TAI -portti (EXCLUSIVE OR)

EHDOTON TAI -portti saa arvon 1 silloin kun täsmälleen yksi sen ottosignaaleista saa arvon yksi. Loogisen funktion lauseke on $F = A'B + AB'$, joka merkitään

$$F = A \oplus B \quad \text{ja piirrosmerkki on} \quad \begin{array}{c} A \\ B \end{array} \begin{array}{|c|} \hline =1 \\ \hline \end{array} F$$

EHDOTON TAI -piirin selvä sukulainen on ekvivalenssiipiiri 'EHDOTON-EI-TAI' (EXCLUSIVE NOR), joka saa arvon 1, jos sen molemmat ottosignaalit ovat samat. Eli $F = AB + A'B'$. (Piirille ei ole olemassa suomenkielistä nimeä.)

$$F = \overline{A \oplus B} \quad \text{ja piirrosmerkki on} \quad \begin{array}{c} A \\ B \end{array} \begin{array}{|c|} \hline =1 \\ \hline \end{array} F$$

EHDOTON TAI -funktiolla on tiettyjä ominaisuuksia, jotka kannattaa ymmärtää:

$$\begin{aligned} A \oplus 0 &= A & A \oplus 1 &= \overline{A} \\ A \oplus A &= 0 & A \oplus \overline{A} &= 1 \\ A \oplus B &= B \oplus A \\ A \oplus \overline{B} &= \overline{A \oplus B} & \overline{A} \oplus B &= \overline{A \oplus B} \\ (A \oplus B) \oplus C &= A \oplus (B \oplus C) = A \oplus B \oplus C \end{aligned}$$

EHDOTON-TAI (XOR)-portin totuustaulu näyttää siis seuraavalta:

Input A	Input B	Output F
0	0	0
0	1	1
1	0	1
1	1	0

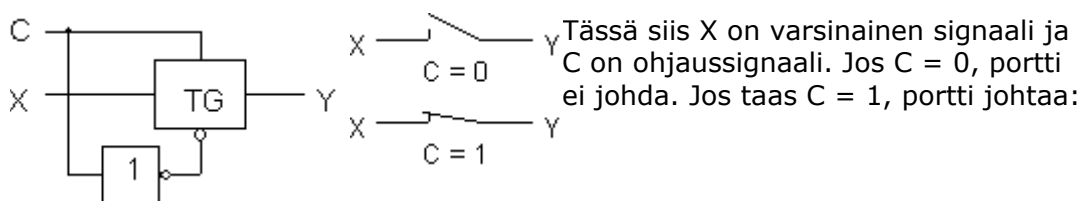
XNOR-portin totuustaulu on tietystikin XOR invertoituna.

EHDOTON TAI -portin sovelluksia:

- ohjattava invertteri
- digitaalinen yhtäsuuruuden vertailupiiri
- aritmeettiset piirit: summabitin muodostus
- PARITON -funktio
- pariteetin muodostus ja tarkastus
- vaiheilmaisin
- lineaariset digitaalipiirit
- valesatunnaissignaalien generointi

Siirtoportti (Transmission Gate, TG)

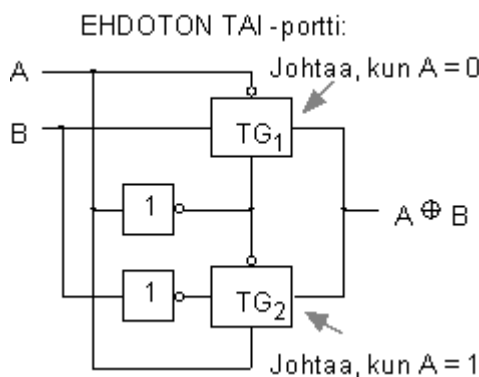
Siirtoportti on CMOS-tekniikalla tehty portti, joka vastaa kytkintä. Päinvastoin kuin porteissa yleensä, siirtoportissa signaali voi kulkea kumpaankin suuntaan tahansa. Itse kytkimen lisäksi siirtoportissa on ohjaussignaali, jonka arvo määrää, onko kytkin auki vai kiinni. Siirtoportti ei sellaisenaan muodosta mitään tiettyä loogista funktiota, mutta sitä voidaan käyttää osana CMOS-tekniikalla tehdyissä porteissa ja muissa piireissä.



Selventävä kuva:

EHDOTON-TAI-funktio siirtoporteilla

Siirtoporttien käyttö perustuu EHDOTON-TAI -funktion siihen ominaisuuteen, että funktion arvo on toisen muuttujan (esim A) arvo sellaisenaan, jos toinen muuttuja (esim B) saa arvon 0 ja toisen (A) arvon inversio, jos toinen (B) saa arvon 1. Allaolevassa piirissä otto-signaalin A arvo 0 avaa siirtoportin TG1, jolloin signaali B pääsee läpi, ja arvo 1 siirtoportin TG2, jolloin B' pääsee läpi. Saadaan siis aikaan haluttu toiminta.



Integroidut piirit

Digitaalipiirit toteutetaan integroiduilla piireillä (Integrated Circuit, IC). Integroitu piiri on pieni puolijohteesta (yleensä piistä) valmistettu hyvin pieni kappale, kutsumanimeltään siru (engl. chip). Sirulla on n kpl toisiinsa yhdistettyjä portteja, jotka yhdessä muodostavat integroidun piirin. Piiri pakataan muovi- tai keraamiseen koteloon, ja se yhdistyy muuhun laitteeseen pinnien (joskus sanotaan myös jalkojen, englanniksi pins) avulla. Pinnien määrä voi tyypillisesti vaihdella 14:stä moneen sataan kappaleeseen. IC piirin kotelon kannessa ilmoitetaan aina piirin numero. Valmistajat taas julkaisevat luetteloita (data book), joka sisältää kaikkien valmistajan tuottamien piirien toiminnan kuvaukset ja muut piirisuunnittelijalle tarpeelliset asiat.

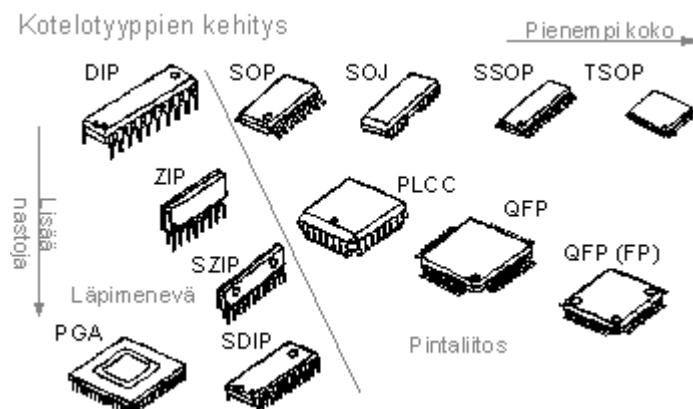
Piirien integrointiaste

IC -piirien koko kasvaa huimasti sitä mukaa, kun kehitetään keinoja pakata samalle sirulle yhä enemmän portteja yhä pienempään tilaan. Alla on esitelty tärkeimmät kokoluokat:

- Small-scale integrated (SSI) devices: yksi paketti sisältää muutaman portin. Yleensä alle 10. Porttien määrää rajoittaa sirun pinnien määrä
- Medium-scale integrated (MSI) devices: yksi paketti sisältää kymmeniä portteja (10-100). MSI piirit suorittavat yleensä yksinkertaisia (matemaattisia) funktioita. Esimerkiksi yksi MSI -piirikomponentti voi suorittaa neljän bitin yhteenlaskuoperaation.
- Large-Scaled integrated (LSI) devices: portteja mahtuu pakettiin sadasta muutamaan tuhanteen kappaletta. LSI -piirit voivat olla esim. yksinkertaisia prosessoreita tai pieniä muisteja.
- Very large-scale integrated (VLSI) devices: portteja on muutamasta tuhannesta useisiin miljooniin. Suuren kokonsa ja halvan valmistustapansa ansiosta VLSI-piirit ovat mullistaneet tietokonesuunnittelun kehityksen. Niitä käytetään esim. monimutkaisina prosessoreina ja laajoina muistitaulukkoina.

Miltä mikropiiri näyttää?

Esimerkiksi tällaiselta (kuva tosin on muutaman vuoden takaa):



Logiikkaperheet

Integroidut piirit eroavat paitsi integrointiasteeltaan myös toteutus-tekniologialtaan. Erilaisten rakennustyyppien ja piirin sisäisten perusporttien rakenteen mukaan piirit erotellaan logiikkaperheiksi (digital logic family). Tietyn logiikkaperheen piirit ovat keskenään yhteensopivia ja ne voidaan suoraan kytkeä toisiinsa, kun taas eri perheiden välillä tarvitaan sovituspiirejä.

Tärkeimpiä perheitä:

- RTL (Resistor-transistor logic) ja DTL (Diode-transistor logic) olivat ensimmäisiä logiikkaperheitä ja nyt jo vanhentuneita
- TTL (Transistor-transistor logic): Vuosikymmenien ajan tärkein ja laajimmalle levinnein logiikkaperhe, mutta sen käyttö on vähentynyt paljon viime aikoina
- ECL (Emitter-Coupled Logic): hyvin suuri toimintanopeus, käytetty paljon supertietokoneissa, nykyään väistymässä CMOS:in tieltä
- MOS (Metal-oxide semiconductor): hyvä valinta, kun tarvitaan suurta komponenttitiheyttä
- CMOS (Complementary metal-oxide semiconductor): tämän perheen piireillä on erittäin vähäinen tehonkulutus. Tehonkulutus on noussut niin määrääväksi seikaksi suurten piirikokonaisuuksien myötä, että CMOS on noussut ehdottomasti tärkeimmäksi piirityypiksi. CMOS piireistä on saatavilla kaikkia kokoluokkia SSI:stä VLSI -piireihin. Lisäksi CMOS piirejä on saatavilla yhteensopivina TTL piirien kanssa. (Joita käytetään apuna silloin, kun CMOS ei pysty tuottamaan tarpeeksi suurta virtaa tai toimimaan tarpeeksi nopeasti.)

Piiriparametrit

Piirejä vertaillaan toisiinsa tiettyjä parametrejä käyttäen. Parametrit kuvaavat piirin erityyppisiä ominaisuuksia ja ne ovat erilaisia eri logiikkaperheillä. Alla on listattu parametreista tärkeimpiä:

- Fan-in: kuvaa portin ottojen (input) määrää.
- Fan-out: kuvaa portin kuormitettavuutta. (Käytännössä parametrilla ilmaistaan se määrä seuraavan saman logiikkaperheen ottoja, jotka voidaan kytkeä yhteen kyseisen piirin antoon ilman, että piirin toiminta häiriintyy.)
- Noise-margin: kuvaa kuinka suuren virheimpulssin ottoon voi saattaa ilman, että piiri toimii ei-toivotusti. Käytännössä siis tämä parametri kuvaa sitä, minkä kokoinen impulssi kääntää bitin ympäri.
- Power dissipation (tehonkulutus) kertoo paljonko portti kuluttaa tehoa. Koska tehosta suuri osa haihtuu lämmöksi, voidaan tästä vertailla myös piirin kriittistä lämpenevyyttä.
- Propagation delay (etenemisviive) kertoo kauanko signaalilla kestää kulkea piirin läpi.

Positiivinen ja negatiivinen logiikka

Tässä kohtaa kannattaa taas muistaa, että logiikkapiireissä ykköset ja nollat ovat jännitteitä. Piirivalmistajat määrittelevät piirien toiminnot fyysisillä tasoilla L ja H siten, että signaalijännite $U_H > U_L$. Loogisten signaalien arvot 0 ja 1 voidaan sitoa fyysisiin tasoihin kahdella tavalla: joko suoraan $0=L$ ja $1=H$, jolloin puhutaan positiivisesta logiikasta tai ristiin $0=H$ ja $1=L$, jolloin kyseessä on negatiivinen logiikka.

Fyysinen taso	Tasoa vastaava jännitealue (esim. HC-CMOS)	Loogisen signaalin arvo	
		Positiivinen logiikka	Negatiivinen logiikka
H	3,15 - 5 V	1	0
L	0 - 1,35 V	0	1

Signaalinkäsittelytekniikan laboratorio

Digitaalitekniikan perusteet - luento 6

Kombinaatiopiireistä

Kombinaatiopiirissä antosignaalien arvot riippuvat vain sen hetkisten ottosignaalien arvoista: Piirissä on n kappaletta ottosignaaleja, jotka kulkevat digitaaliporttien läpi (näiden digitaaliporttien muodostama kokonaisuus on siis kombinaatiopiiri). Tämän lisäksi piirissä on m kappaletta antosignaaleja. Kombinaatiopiirit voidaan toteuttaa esimerkiksi porttipiireillä.

(Erotuksena on kombinaatiopiireille on olemassa sekvenssiipiirejä, joissa antosignaalien arvot voivat riippua joko piirin aikaisemmista tiloista tai piirin aikaisemmista tiloista ja nykyisistä ottosignaaleista. Sekvenssiipiirejä käsitellään myöhemmin tällä kurssilla.)

Alla on kuva kombinaatiopiirin **lohkokaaviosta**. Lohkokaaviossa kuvataan yksittäisillä laatikoilla piirin toiminnallisia osia. (Kombinaatiopiirissä toiminnallisia osia on yksi kappale.)



Kombinaatiopiirin analyysi

Kun kombinaatiopiiri on toteutettu porteilla, voidaan piiri analysoida seuraavasti:

- Kytkentäfunktioiden selvitys:
 - nimetään jokaisen portin antosignaali (esim. F, G, VALO, tms.)
 - aloitetaan koko piirin antosignaaleista ja muodostetaan niille lauseketta portti kerrallaan kohti ottosignaaleja.
 - jokaisen yksittäisen portin antosignaali voidaan merkitä portin ottosignaalien operaationa
 - Kun päästään tilanteeseen, jossa antosignaali on määritelty pelkästään koko piirin ottosignaaleilla, on saatu haluttu kytkentäfunktio
- Totuustaulujen laadinta
 - kun kytkentäfunktiot on selvitetty, muodostetaan taulu kaikista ottosignaalien kytkentäkombinaatioista
 - tarkastetaan kytkentäfunktion arvot kaikilla kombinaatioilla kytkentäfunktion lausekkeen avulla (Sijoittamalla funktioihin kaikki eri kombinaatiot yksi kerrallaan.)
 - Piirretään tulosten perusteella totuustaulut

=>Piiri on analysoitu.

Kombinaatiopiirin suunnittelu

(Tätähän on käyty läpi aikaisemminkin.) Suunnitteluprosessi on käytännössä seuraavanlainen:

- Määritellään piirin toiminta (spesifikaatio)
- Nimetään piirin otot ja annot
- Laaditaan totuustaulu tai -taulut
- Sievennetään funktiot esim. Karnaugh'n kartan avulla, kuten on opittu. (Mikäli mahdollista kannattaa jo tässä vaiheessa ottaa huomioon, mitä komponentteja on käytössä)
- Suunnitellaan piiri käytettävissä olevista komponenteista
- Testataan suunnitellun piirin toiminta ja korjataan sitä haluttuun suuntaan

Proffamuistutin kombinaatiopiiritoteutuksella

Tässä esimerkissä käydään käytännönläheinen kombinaatiopiirin suunnittelutapaus läpi.

Toiminnan määrittely

Jotta säästettäisiin proffan koko ajatuskapasiteetti opetuskäyttöön, suunnitellaan muistutin, joka mittaa ilman lämpötilaa ja kosteutta ja neuvoo sen mukaan professoria joko pukemaan päällensä turkin tai ottamaan sateenvarjon mukaansa.

Muistuttimeen tulee siis sadeanturi ja pakkasanturi. Kun sadeanturi mittaa ilmankosteuden suureksi (ottosignaali RAIN=1), syttyy sateenvarjolamppu (antosignaali UMBR=1) palamaan. Kun taas lämpömittarin lukema laskee alle nollan celsiusasteen (ottosignaali COLD =1), syttyy turkkilamppu (antosignaali COAT=1) palamaan.

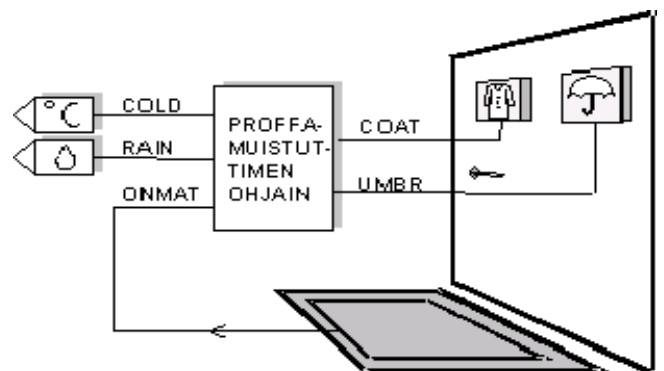
Lisäksi laitteessa on mattoanturi, joka ilmoittaa, milloin proffa seisoo matolla (ottosignaali ONMAT =1). (Lamppujahan ei kannata polttaa turhaan.) Lisäksi täytyy ottaa huomioon vielä tilanne 'lumisade': sateenvarjoa ei tarvita, jos on kylmä ja sataa, koska silloin sataa lunta. Tällöin muistutetaan proffaa vain takista.

Piirin otto- ja antosignaalit

Piiri tarvitsee siis toimiakseen seuraavat signaalit:

Ottosignaalit:

- RAIN (sadeanturin lukema: 1, jos sataa; 0, jos ei sada)
- COLD (lämpömittarin lukema: 1, jos pakkasta; 0, jos ei)
- ONMAT (mattoanturin lukema: 1, jos matolla seistään, 0, jos ei seistä)



Antosignaalit:

- UMBR (ohjaa sateenvarjolamppua: 1, lamppu palaa; 0, lamppu ei pala)
- COAT (ohjaa turkkilamppua: 1, lamppu palaa; 0, lamppu ei pala)

Totuustaulut

Esitetään kaikkien muuttujien (input) kaikki mahdolliset kombinaatiot:

Input COLD	Input RAIN	Input ONMAT	Output COAT	Output UMBR
0	0	0	0	0
0	0	1	0	0
0	1	0	0	0
0	1	1	0	1
1	0	0	0	0
1	0	1	1	0
1	1	0	0	0
1	1	1	1	0

(Tässä tapauksessa tehtävänanto on sen verran yksinkertainen, että funktiot olisi voinut muodostaa myös suoraan.)

Kytkeäntäfunktiot

$$\begin{aligned}\text{COAT} &= \text{COLD} \times \text{RAIN}' \times \text{ONMAT} + \text{COLD} \times \text{RAIN} \times \text{ONMAT} \\ &= \text{COLD} \times \text{ONMAT}\end{aligned}$$

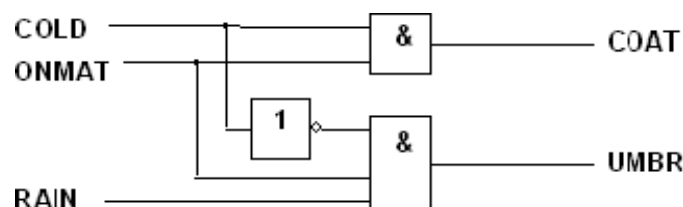
Tämä sievennys tulee suoraan Boolean algebran avulla: $XYZ' + XYZ = XY(Z' + Z) = XY$

(Eli on pakkasta ja proffa seisoo matolla. Sateesta ei välitetä.)

$$\text{UMBR} = \text{COLD}' \times \text{RAIN} \times \text{ONMAT}$$

(Eli ei ole pakkasta, sataa ja proffa seisoo matolla.)

Piirikaavio



Testaus ja korjaukset

Jätetään esittämättä, koska piiri on toiminut edellisinäkin vuosina.

Aritmeettisista piireistä

Aritmeettinen piiri on kombinaatiopiiri, joka suorittaa matemaattisia laskuoperaatioita. (Esimerkiksi +, -, ×, ÷) Logiikka perustuu siihen, että loogisia tiloja '0' ja '1' asetetaan vastaamaan binääriaritmetiikan numeroita '0' ja '1'. (Täytyy siis muistaa, että matemaattinen $1 + 1$ tarkoittaa eri asiaa kuin kytkentäalgebran $1 + 1$.)

Kahden bitin summa on aina minimissään 0 ja maksimissaan 10 ($=2_{10}$). Tämän vuoksi jokaisen bitin kohdalla tulokseen on varattava **kaksi bittiä**: varsinainen summabitti sekä siirtobitti (tai muistibitti, engl. carry). Tämä muistibitti siirretään seuraavaksi enemmän merkitsevän bitin laskutoimitukseen. (Ylimääräistä ongelmaa tilan kanssa ei tule, koska $1 + 1 + 1 = 11 = (3_{10})$.)

Puolisummain (Half Adder, HA)

Puolisummain muodostaa kahden yksibittisen luvun (X ja Y) summabitin (S) ja siirtobitin (C = carry):

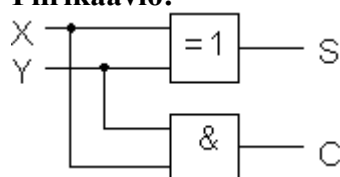
Input X	Input Y	Output C	Output S
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

Puolisummainen kytkentäfunktiot ovat seuraavanlaisia:

$$S = X'Y + XY' = X \oplus Y$$

$$C = XY$$

Piirikaavio:



Kokosummain (Full Adder, FA)

Kokosummain on puolisummaina kehittyneempi siinä mielessä, että se ottaa sisäänsä myös tulevan muistibitin (CI = carry in) esim. edellisestä yhteenlaskutoimituksesta. Kokosummain laskee siis kolme bittiä yhteen ja palauttaa summan (S) sekä uuden siirtobitin (CO = carry out).

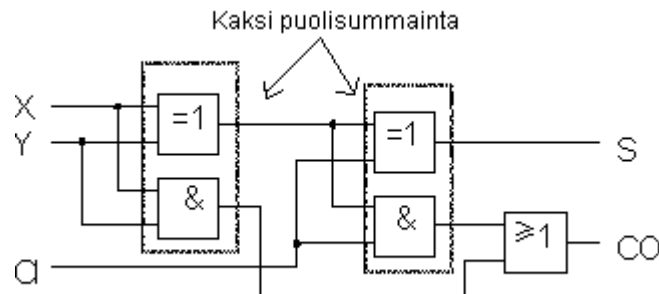
Input X	Input Y	Input CI	Output C	Output S
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

Kokosummainen kytkentäfunktiot:

$$S = X'Y'CI + X'YCI' + XY'CI' + XYCI = X \oplus Y \oplus CI$$

$$C = X'YCI + XY'CI + XYCI' + XYCI = XY + CI(X \oplus Y)$$

Piirikaavio:

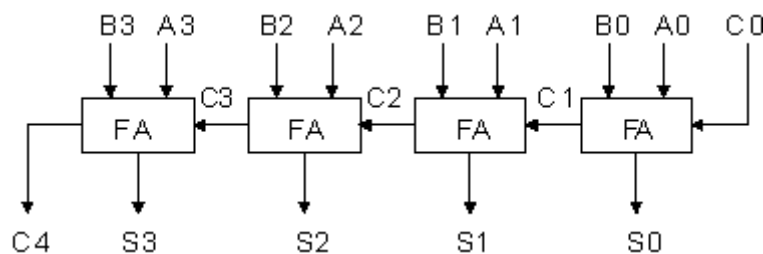


(Kun ollaan ovelia, nähdään, että kokosummain muodostuu kahdesta puolisummaimesta, joita yhdistää TAI -portti.)

Esimerkkejä aritmeettisista piireistä

Summaimista on hyötyä vasta sitten, kun niillä voidaan käsitellä isoja lukuja. Isojen lukujen käsittelyä varten yksinkertaisesti liitetään monia summaimia joko sarja- tai rinnakkaismuotoisesti yhteen.

4-bittinen rinnakkaismuotoinen summain



Summainen lisäksi tärkeä operaattori on 'summain - vähennin'. Tämä on piiri, joka pystyy laskemaan sekä yhteen- että vähennyslaskua. Tämä toteutetaan siten, että käytetään sopivaa merkkibittä, jonka mukaan päätellään, kumpi laskutoimitus on kyseessä: yhteenlasku suorittaa normaalin yhteenlaskuoperaation, vähennyslaskussa täytyy muuttaa ensin vähentäjä kahden komplementtiinsa.

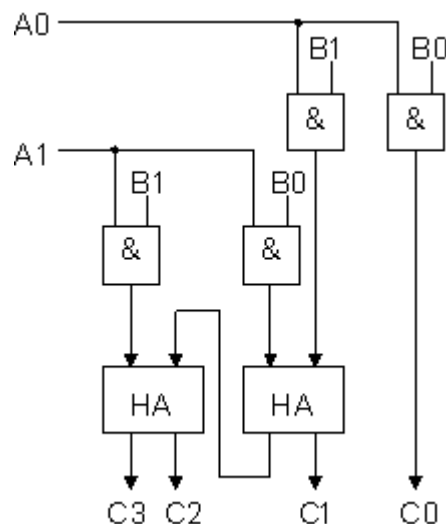
Binäärikertoja

Binäärikertojassa yksittäisten bittien tulot lasketaan JA -piireillä. Osatulot lasketaan yhteen puolisummaimilla (Half-adder, HA). Esimerkiksi näin:

Suoritetaan yksinkertainen 2 -bittisten lukujen kertolasku:

		B1	B0	
		A1	A0	
		A0B1	A0B0	
	A1B1	A1B0		
C3	C2	C1	C0	

Piiritoteutus:



Dekooderit

Dekooderi on kombinaatiopiiri, joka koodaa binäärilukuja yksikäsitteiseksi binäärikoodiksi. Käytännössä tämä tarkoittaa sitä, että dekodeeri ottaa sisään n kpl ottosignaaleja ja sillä on 2^n kappaletta antosignaaleja. (Näistä ei välttämättä käytetä kaikkia, joten asianmukaisempaa on puhua n :stä m :ään dekodeereista, missä $m < 2^n$.) Antosignaalit ovat ottosignaalien muodostamia minimitermejä. Siis siten, että kun ottosignaali muodostaa kombinaation 000..00, saa vastaava antosignaali D_0 arvon yksi muiden antosignaalien saadessa arvon nolla. Edelleen kombinaatiolla 000..001, saa antosignaali D_1 arvon yksi muiden ollessa nolliä jne. Eli ottosignaalien (n kappaletta) kombinaatio määrää **yksikäsitteisesti** mikä antosignaaleista (m kappaletta) aktivoituu. Aktivoituvia antosignaaleja on vain ja ainoastaan 1 kappale jokaista binäärilukua kohti.

Esimerkki dekooderista

Selvennetään asiaa esittelemällä yksinkertainen kahdesta neljään (2-4) -dekooderi.

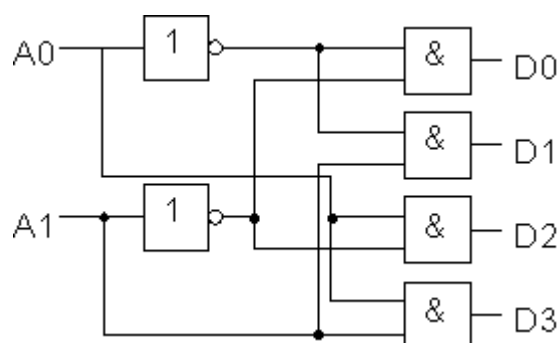
Dekooderilla on siis kaksi ottosignaalia, joiden perusteella sillä on neljä antosignaalia.

(Ottosignaalien muodostamat minimitermit.)

Kyseisen dekooderin totuustaulu näyttää seuraavalta:

Input A0	Input A1	Output D3	Output D2	Output D1	Output D0
0	0	0	0	0	1
0	1	0	0	1	0
1	0	0	1	0	0
1	1	1	0	0	0

Piirikaavio dekooderille saadaan totuustaulun perusteella:



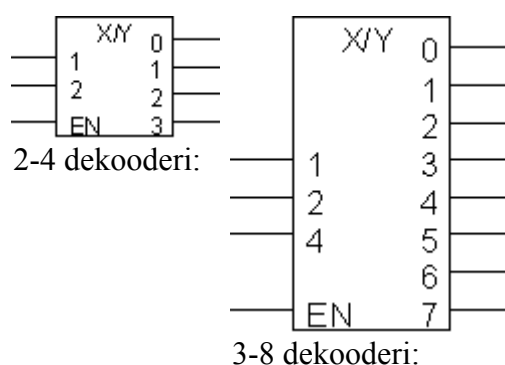
Usein käytetään myös invertoitua dekooderia, joka on piirikaavioltaan edellisen kaltainen, mutta siinä on JA -porttien sijaan JAEI -portit (NAND). Tällöin totuustaulu muuttuu myös päinvastaiseksi: anto saa arvon yksi kaikkialla muualla paitsi oman järjestysnumeronsa ilmoittavan minimitermin kohdalla, jolloin anto saa arvon nolla. Invertoituantoisesta dekooderista on esimerkki jäljempänä.

Dekooderin piirrosmerkki:

Yleistunnus: X/Y

Otot merkataan binääriluvun kertoimen mukaan eli 1,2,4,8...

Annot kuvataan minimitermien numeroilla eli järjestyksessä 0 --> 2^n



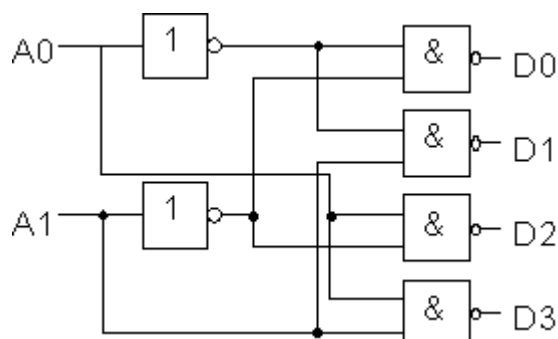
EN -otto on sallintaotto (enable). Tyypillisesti arvolla EN=0, piiri ei toimi, ja arvolla EN=1 piiri toimii normaalisti.

Esimerkki invertoituantoisesta 2-4 dekooderista

Totuustaulu on muuten vastaava kuin perusdekooderilla, paitsi ettäannot on invertoitu ts. ottosignaalit määrittävät mikä antosignaaleista on looginen nolla, muiden ollessa ykkösiä:

Input A0	Input A1	Output D3	Output D2	Output D1	Output D0
0	0	1	1	1	0
0	1	1	1	0	1
1	0	1	0	1	1
1	1	0	1	1	1

Piirissä käytetään JA -porttien sijaan JAEI -portteja:



Kooderit

Kooderi on kombinaatiopiiri, joka muuntaa yksikäsitteistä binäärikoodia yksikäsitteiseksi binääriluvuksi eli unaarimuodosta binaarimuotoon. Käytännössä tämä tarkoittaa sitä, että kooderin sisään tulossa on 2^n kappaletta ottosignaaleja, jotka koodataan n kappaleeksi antosignaaleja. Kooderin toiminta on siis päinvastaista aiemmin esitetyn dekooderin toimintaan nähden.

Esimerkki kaksibittisestä binaarikooderista

Selvennetään asiaa esittelemällä yksinkertainen kaksibittisen binaarikooderin (neljästä kahteen kooderin) toiminta. Kooderilla on siis neljä ottosignaalia, joiden perusteella sillä on kaksi antosignaalia. Kyseisen kooderin totuustaulu näyttää seuraavalta:

Input D3	Input D2	Input D1	Input D0	Output A1	Output A0
0	0	0	1	0	0
0	0	1	0	0	1
0	1	0	0	1	0
1	0	0	0	1	1

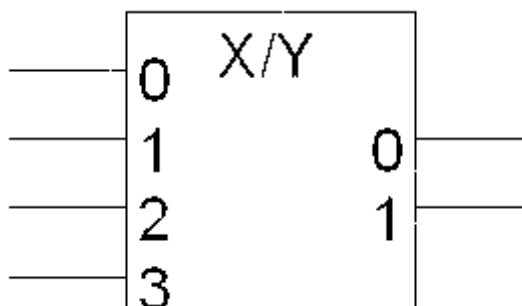
Binaarikooderin piirrosmerkki:

Yleistunnus: X/Y

Otot kuvataan minimitermien numeroilla eli järjestyksessä $0 \rightarrow 2^n$

Annot merkataan binääriluvun kertoimen mukaan eli 0,1,2,4...

4-2 kooderi:



EN -otto on sallintaotto (enable). Tyypillisesti arvolla EN=0, piiri ei toimi, ja arvolla EN=1 piiri toimii normaalisti.

Prioriteettikooderi

Edellä esitetyn kooderin toiminta ei ole määritelty siinä tapauksessa mikäli ottosignaaleista D3 - D0 useampi kuin yksi saa arvon 1 samanaikaisesti. Tätä varten on prioriteetti kooderi, jossa ottosignaaleiden eniten merkitsevä ykkönen määrää kooderin antokoodin. Prioriteettikooderin toiminta on kuvattu alla olevassa esimerkissä.

Esimerkki kaksibittisestä binaarikooderista

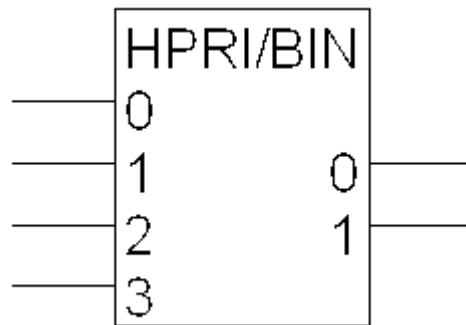
Selvennetään asiaa esittelemällä kaksibittisen prioriteettikooderin toiminta. Kooderissa siis ottosignaaleiden D3-D0 eniten merkitsevä ykkönen määrää kooderin antokoodin, jolloin kyseisen kooderin totuustaulu näyttää seuraavalta:

Input D3	Input D2	Input D1	Input D0	Output A1	Output A0
0	0	0	1	0	0
0	0	1	X	0	1
0	1	X	X	1	0
1	X	X	X	1	1

Ylläolevaa totuustaulua tarkastelemalla nähdään, että toisin kuin kooderin tapauksessa, prioriteettikooderin toiminta on määritelty riippumatta ottosignaalien ykkösten määrästä.

Prioriteettikooderin piirrosmerkki:

4-2 prioriteettikooderi:



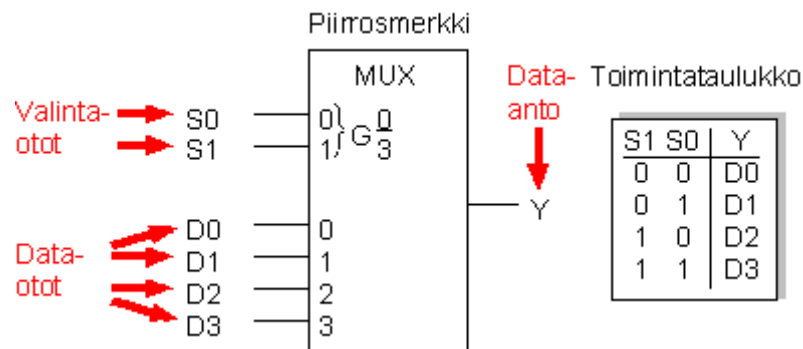
Signaalinkäsittelytekniikan laboratorio

Digitaalitekniikan perusteet - luento 7

Ottovalitsin eli multiplexeri (MUX)

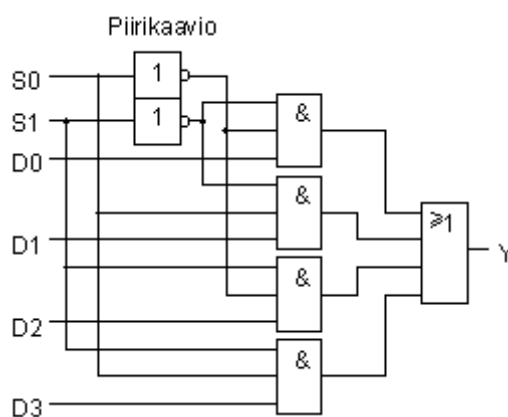
Multiplexeri on kombinaatiopiiri, joka tiettyjen valintaehtojen mukaan yhdistää monesta binääridataisesta otosta yhden kerrallaan antoon. Valintaa ohjaa joukko valintaottoja (selection inputs). Tyypillisesti piirissä on n kpl valintaottoja, jotka ohjaavat, mikä 2^n stä dataotosta kulloinkin pääsee piirin antoon.

Alla esitellään yksinkertainen $4 \rightarrow 1$ ottovalitsin. Merkintä tarkoittaa, että piirissä on neljä dataottoa, joista yksi kerrallaan pääsee antoon. Koska dataottoja on neljä, tarvitaan 2 valintaottoa (koska $4 = 2^2$, joiden kombinaatio siis määrää, mikä dataotoista kulloinkin pääsee piirin antoon.

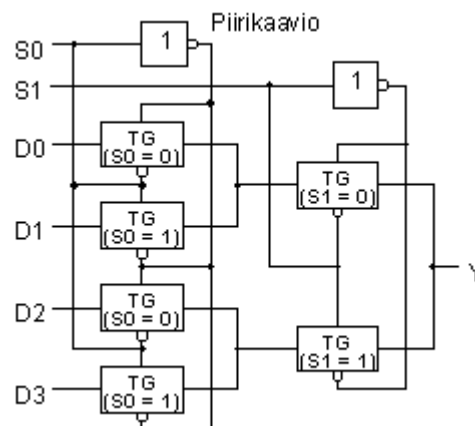


Ottovalitsimen toteutus

Tässä esitellään kaksi tapaa toteuttaa ottovalitsin. Ensimmäisessä käytetään peruspiirejä (AND ja OR -portteja sekä inverttereitä eli NOT-portteja), jälkimmäisessä siirtoportteja (TG). Molemmissa tapauksissa perusidea on, että kunkin valintakombinaation aikana vain sitä vastaava dataotto pääsee data-antoon ja muiden yhteys estetään:



MUX perusporteilla



MUX siirtoporteilla

Sallintaotto (enable, EN) ottovalitsimessa

Monimutkaisempiin logiikkapiireihin lisätään usein otto, joka kontrolloi piirin toimintaa. Tätä ottoa sanotaan sallintaotoksi ja sitä merkitään yleensä lyhenteellä EN. Sallintaoton toiminta on yksinkertainen: otto joko estää tai sallii piirin varsinaisen toiminnan. Periaatteessa voidaan esittää, että jos $EN=0$, piiri ei tee mitään ja toisaalta jos $EN=1$, piiri toimii normaalisti.

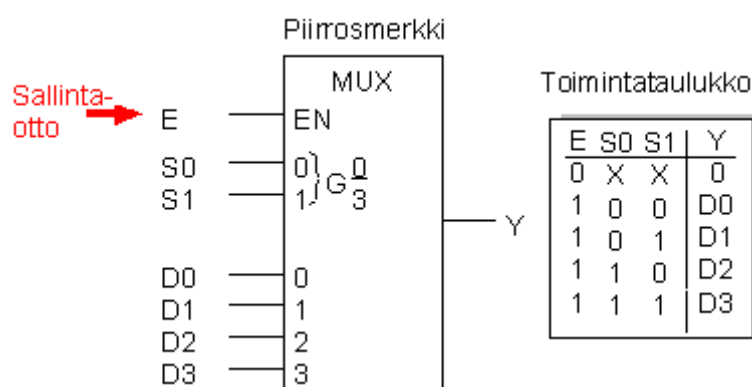
Kaavamuodossa siis näin:

$EN=0 \rightarrow Y=0$

$EN=1 \rightarrow Y=D_i$

Otto voi kuitenkin toimia negatiolla, joten sen toiminta kannattaa tarkistaa jokaisessa tapauksessa erikseen.

Tavallisen ottovalitsimen tapauksessa sallintaotto merkitään allaolevan kaavion mukaan. Vieressä oleva totuustaulu esittää, miten otto vaikuttaa piirin toimintaan:



KytKentäfunktion toteutus ottovalitsimella

Ottovalitsinta käytetään kytkentäfunktion toteuttamiseen silloin, kun kytkentäfunktio on monimutkainen ja sievenee huonosti tai kun halutaan välttää turhaa itse-rakentamista ja valmiskomponenttien määrää. Ottovalitsin on yleislogiikkapiiri. Nyrkkisääntönä voidaan käyttää sitä, että N:llä valintaotolla varustetulla MUXilla voidaan toteuttaa N+1 muuttujan funktio.

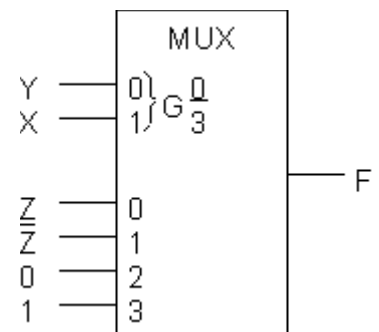
N+1 muuttujan totuustaulun mukaan N suurinta, tai eniten merkitsevää muuttujaa (eli totuustaulussa N kpl muuttujia vasemmalta päin luettuna) kytketään multiplekserin valintaottoihin. Dataottoihin kytketään joko **1, 0, vähiten merkitsevä muuttuja tai sen komplementti**. Tällä menettelyllä säästetään ottovalitsimen koossa yhden kakkosen potenssin verran. Esimerkki selventää asiaa:

Esimerkki kytkentäfunktioista MUXilla toteutettuna

Käsitellään kolmen muuttujan funktiota $F(X,Y,Z) = \text{minimien summa}(1,2,6,7)$. Koska **muuttujia on kolme, tarvitaan kaksi valintaottoa** (koska $2^2 = 4 > 3$). **Valintaottoihin menevät eniten merkitsevät muuttujat** eli X ja Y. Kahdella valintaotolla taas voidaan muodostaa neljä eri kombinaatiota. Tämä tarkoittaa, että MUXin koko on $4 \rightarrow 1$. Piirretään funktion totuustaulu ja **jaotellaan totuustaulu kahden rivin pätkiin**: (Tämä siksi, että näillä kahdella rivillä on viimeinen muuttuja (vähiten merkitsevä muuttuja) aina sama)

X	Y	Z	F	data-otto	selitys
0	0	0	0	Z	koska $F=Z$
0	0	1	1		
0	1	0	1	Z'	koska $F=Z'$
0	1	1	0		
1	0	0	0	0	koska $F=0$ riippumatta Z:sta
1	0	1	0		
1	1	0	1	1	koska $F=1$ riippumatta Z:sta
1	1	1	1		

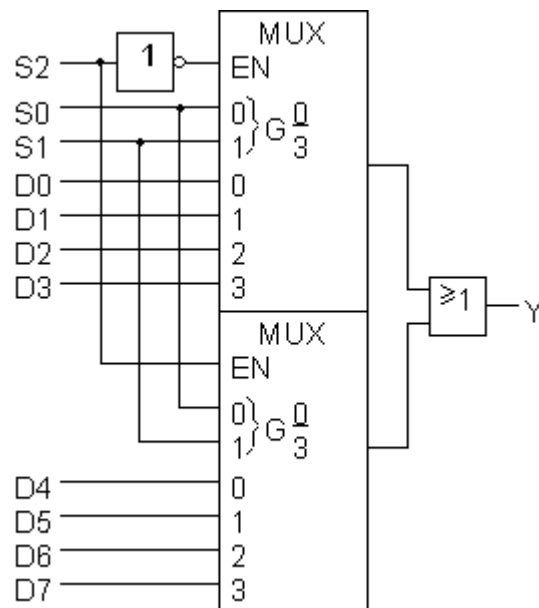
Totuustaulusta nähdään, että funktion arvo jokaisella kahden rivin kokonaisuudella on ilmaistavissa joko kiinteänä ykkösenä tai nollana tai Z-muuttujan avulla (Z suoraan tai komplementoituna). Viimeinen sarake taulussa (dataotto - sarake) kertoo dataottoon kulloinkin tulevan signaalin. Funktion varsinainen toteutus näyttää siis seuraavalta (kuvassa kytkentäkaavio):



Ottovalitsimen laajennus

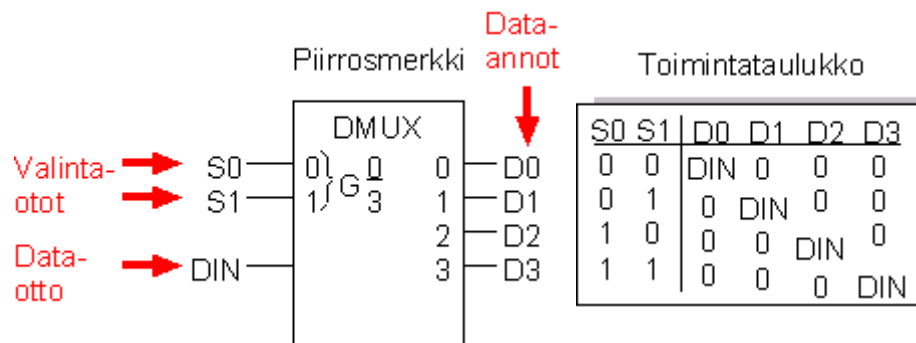
Yleislogiikkapiireistä voidaan usein räätälöidä alkuperäisiä suurempia kokonaisuuksia tarpeen mukaan. Oikealla on esimerkki ottovalitsimen laajennuksesta. Laajennussäännöt ovat yleensä yksinkertaista kytkentäalgebraa, ajattelemisen auttaa.

Kuvassa kahdesta 4 -> 1 MUXista on muodostettu yksi 8 -> 1 MUX. Ottosignaali S2 määrittää onko käytössä ylempi vai alempi MUX, sillä se on kytketty molempien MUXien enable-ottoon.



Antovalitsin

Ottovalitsimen vastakohtana voidaan pitää antovalitsinta. Se on yleislogiikkapiiri, joka yhdistää oton yhteen monesta annosta. Muut annot ovat vakioitilassa. (Sovittu joko 0 tai 1.) Antovalitsin on toimintansa puolesta sama piiri kuin sallintaotolla varustettu dekooderi. Alla on yksinkertainen esimerkki 1 -> 4 antovalitsimesta.



Logiikkapiirien piirrosmerkeistä

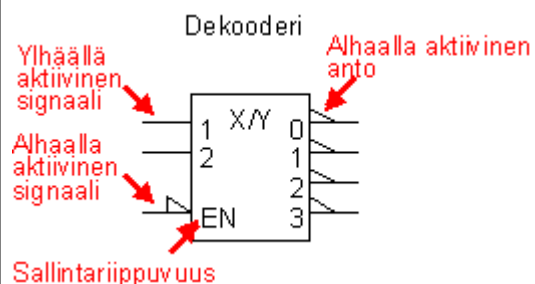
Logiikkapiirien piirrosmerkeistä on kaksi koulukuntaa. Tällä kurssilla käytetään IEC:n kansainvälisiä suorakulmaisia merkkejä. Toinen vaihtoehto on perinteiset amerikkalaiset merkit, joissa portit erotellaan toisistaan lähinnä merkin muodon mukaan. IEC:n merkit ovat aina suorakulmaisia, sivujen suhde on vapaasti määriteltävissä. Otot ovat vasemmalla,annot oikealla. Perusporttipiirien piirrosmerkinnät on esitetty aikaisemmin. Tässä käsitellään yleisiä sääntöjä sekä hieman monimutkaisempia komponentteja. Alla on esimerkkejä merkinnöistä ja merkkien yhdistämissäännöistä:

Ottoihin ja antoihin liittyvät merkinnät

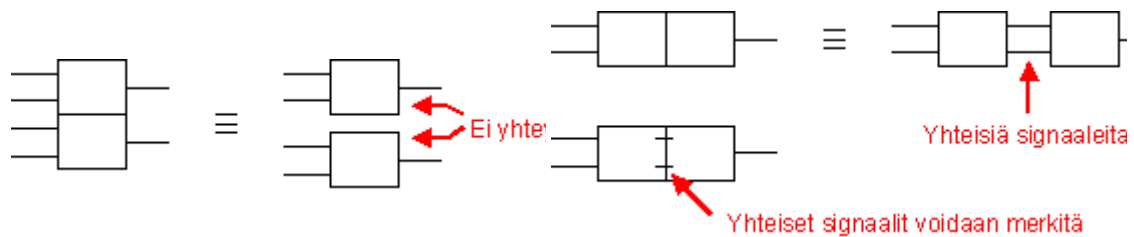
(Jos merkintä vaikuttaa oudolta, se liittyy todennäköisesti aikariippuvaan komponenttiin, joita käsitellään myöhemmin kurssilla..)

Merkintä	Käyttötarkoitus ja merkitys
	Alhaalla aktiivinen otto ja anto
	Looginen negaatio otossa ja annossa (alhaalla aktiivinen otto ja anto)
	Sallintaotto: sallii kaikki piirinannot.
	Dynaaminen otto (reunaliipaistava kiikku)
	Puskurianto

Dekooderiesimerkki



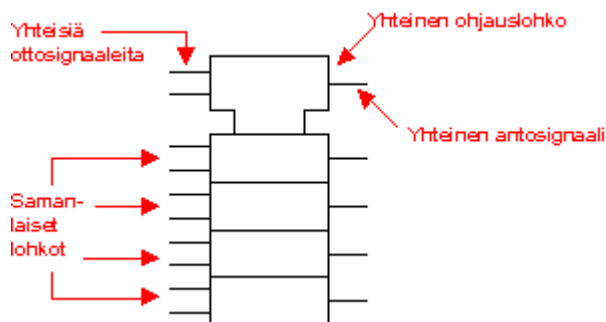
Merkkien yhdistämissäännöt:



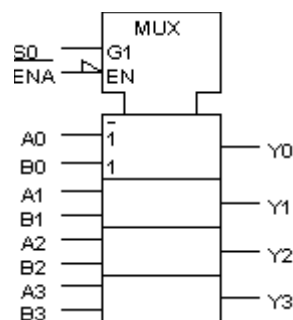
Esimerkkejä yleislogiikkapiirien piirrosmerkeistä ja niitä koskevista säännöistä

Yhteinen ohjauslohko

On joukko yleislogiikkapiirejä, joissa on samassa piirissä useita samanlaisia osia (lohkoja) ja näillä yhteisiä signaaleita. Tällöin yhteiset signaalit merkitään erikseen yhteiseen ohjauslohkoon ja kaikki erilliset osat kasataan tämän lohkon alle. Yhteisen ohjauslohkon omaavia komponentteja käsitellään kurssin loppupuolella.

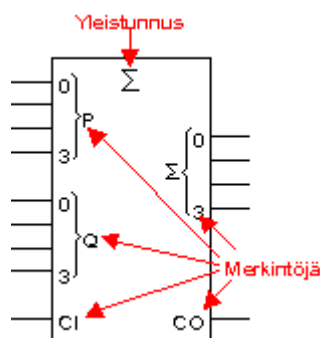


Nelikko yksi kahdeksasta valikko (eräs ottovalitsintyyppi)

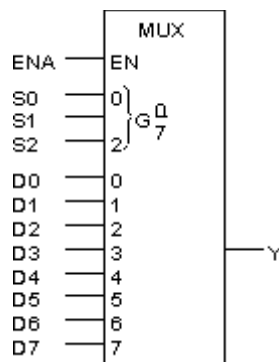


Muita piirrosesimerkkejä

4-bittisen binäärisummaimen piirrosmerkki



1->8 MUXin piirrosmerkki



Riippuvuusmerkinnöistä

Samoissa komponenteissa on usein eriarvoisia ottoja ja antoja. Sanotaan, että otto tai anto on riippuvainen jostain. Riippuvuus esitetään yleensä kirjainlyhenteellä ja numerolla siten, että hallitsevassa otossa (annossa) on sama merkki kuin hallinnan alaisessa otossa (annossa). Merkintä on keskeinen logiikkapiirien piirrosmerkintäjärjestelmässä. Ehkä yksinkertaisin tapaus on riippuvuus kello-otosta, joita käsitellään aikariippuvaisissa piireissä.

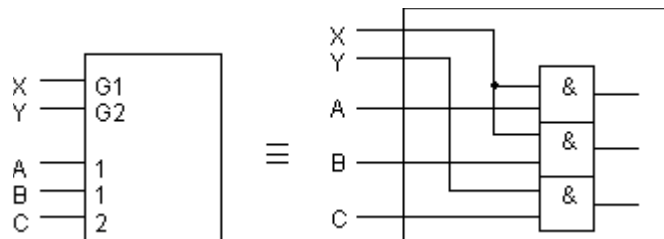
Alla esitellään yleisimmin käytetyt riippuvuusmerkinnät. Näiden osaamista tärkeämpää on ymmärtää riippuvuuden periaate. Riippuvuus on tavallaan kyseisen oton enableointi: otto voi toimia vain, jos sitä hallitseva otto on suotuisassa tilassa.

Riippuvuusmerkinnät

Merkintä	Riippuvuus	Käyttö
A	Osoite	Muistipiirien osoiteotoissa
C	Ohjaus	Sekvenssipiirin kello-otoissa
EN	Sallinta	Signaalin aktiivisuuden sallinta
G	JA	JA-funktio muun signaalin kanssa
M	Toimintatapa	Valitsee toimitavan useista erilaisista
N	Negaatio	Invertoi signaalin (EHDOTON TAI)
R	Nollaus	Nollaa signaalin (sekvenssipiireissä)
S	Asetus	Asettaa signaalin 1-tilaan (sekvenssipiireissä)
V	TAI	TAI-funktio muun signaalin kanssa
X	Kytkenä	Siirtoportin kytkennän sallinta
Z	Siirto	Piirin sisäisissä kytkennöissä

Riippuvuusmerkintäesimerkki (JA-riippuvuus)

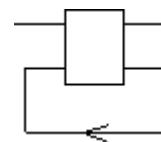
Kuvassa esitetään komponentti, jossa on kaksi hallitsevaa ottoa (X, Y) ja kolme näistä hallitsevista ottoista riippuvaa hallinnan alaista ottoa. Hallitsevat otot numeroidaan 1 (X) ja 2 (Y). Vastaava numero merkitään alisteisten ottojen kohdalle ilmaisemaan, mille hallitsevalle otolle toinen otto (A, B tai C) on alisteinen. Tässä esimerkissä otot A ja B ovat alisteisia hallintaotolle X, kun taas otto C on alisteinen hallintaotolle Y.



Sekvenssiipiireistä

Tähän mennessä kurssilla on käsitelty kombinaatiopiirejä. Kombinaatiopiireissä antosignaalien arvot riippuvat vain kytkentäfunktion määräämällä tavalla ottosignaaleista. Sekvenssiipiireissä antosignaalien arvot riippuvat myös piirin tilasta, johon vaikuttavat myös piirin aikaisemmat ottosignaalit ja tilat. Puhutaan, että piiri on aikariippuvainen. (Sekvenssiipiirin toinen nimitys on aikariippuvainen piiri.)

Sekvenssiipiirin erityisyys on, että se kykenee tallettamaan ja muistamaan oman tilansa. Tästä siis johtuu aikariippuvuus. Sekvenssiipiireissä on sisäisiä takaisinkytkentöjä. Näillä kytkennöillä säilytetään piirin tila erityisissä muistikomponenteissa. Sekvenssiipiirin yksinkertainen yleismerkki on kuvattu oikealla. Ylimääräinen silmukka tarkoittaa sisäistä takaisinkytkentää. Sillä siirretään piirin tila uudestaan takaisin piiriin.



Sekvenssiipiirejä on kahden tyyppisiä: asynkronisia ja synkronisia. Näistä synkroniset ovat yleisempiä paremman luotettavuutensa vuoksi. Asynkronisessa piirissä piirin tila vaihtuu ottosignaalin vaihtuessa. (Eli heti, kun piiriin tulee uutta dataa, päivittää piiri tilansa.) Synkronisessa sekvenssiipiirissä piiri vaihtaa tilaansa erityisen kellosignaalin tahdissa, eli aina tietyin tasaisin väliajoin.

Sekvenssiipiirit sisältävät aina tavallisen kombinaatiopiirin. Asynkronisessa sekvenssiipiirissä on tämän lisäksi takaisinkytkentä. Synkronisessa sekvenssiipiirissä on takaisinkytkennän lisäksi vielä tilarekisteri. Tilarekisteri on piirin osa, jonka tehtävänä on säilyttää sisään saamaansa dataa, kunnes se kellosignaalin tahdissa antaa tallettamansa datan muun piirin käyttöön. Alla on esitetty rakennekaaviot tavallisesta kombinaatiopiiristä, asynkronisesta sekvenssiipiiristä sekä synkronisesta sekvenssiipiiristä.



Salvat ja kiikut

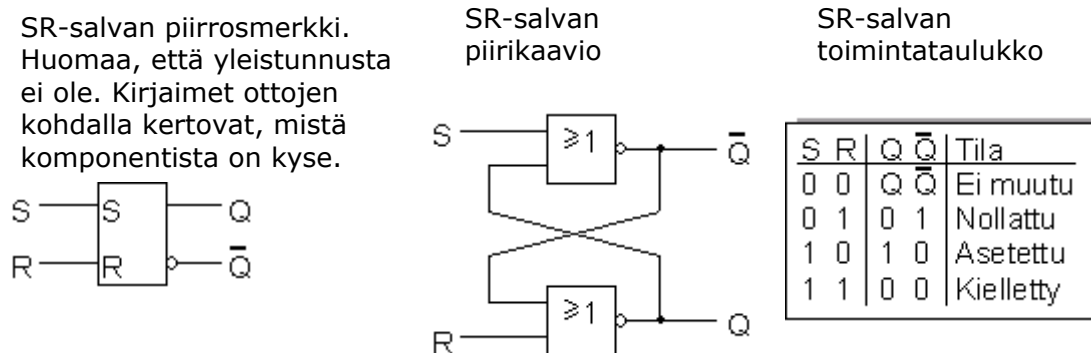
Salvat ja kiikut ovat yksinkertaisia sekvenssiipiirejä, jotka pystyvät muistamaan niihin talletetun tilan. Salvat ovat näistä yksinkertaisempia ja niiden toiminta on suhteellisen rajoittunutta. Kiikut ovat salpoja monimutkaisempia, mutta luotettavampia. (Suosittelemme kiikkujen käyttöä piirisuunnittelussa..) Kiikkuja käytetään synkronisten sekvenssiipiirien tilarekistereinä. Ne tallettavat piirin tilan kellojakson ajaksi. Kiikkujen etu salpoihin nähden on se, että ne muuttavat tilaansa vain kellojakson tahdissa. Salvat tarkastelevat ehtoinaan vain tavallisia ottosignaaleja ja saattavat muuttaa tilaansa myös kesken kellojaksoa ts. asynkronisesti.

Salpojen ja kiikkujen toiminta on suoraviivaista: ne muuttavat antosignaaliaan sen mukaan, minkälaisia kombinaatioita ottosignaaleista ne saavat sisäänsä. (Tosin kiikut siis vain kellosignaalin tahdissa.) Jokaisella salpa- (kiikku) tyyppillä on toimintakaavio, jonka mukaan salpa (kiikku) käyttäytyy. Salvoilla (kiikuilla) on joukko ottoja, joiden nimet ilmenevät tyyppillisesti salvan (kiikun) nimestä. Antoa merkitään Q kirjaimella. Yleensä ilmoitetaan sekä Q että Q'. Tässä yhteydessä esitellään lyhyesti 2 yksinkertaista salpaa. Kiikkuja käsitellään seuraavalla luennolla.

SR-salpa

SR-salpa on sekvenssiipiireistä yksinkertaisin. Se voidaan toteuttaa joko TAI-EI (NOR) tai JA-EI (NAND) porteilla. SR-salvalla on kaksi ottoa: S (Set) ja R (Reset). Kun ottosignaali on $S=1$ ja $R=0$, salpa asettaa tilansa ykköseksi (siis $Q \rightarrow 1$ ja $Q' \rightarrow 0$). Päinvastaisessa ottosignaalien tapauksessa ($S=0$, $R=1$) salpa nolaa tilansa (siis $Q \rightarrow 0$ ja $Q' \rightarrow 1$). Tilassa 00 anto pysyy vakiona eli piiri ei näytä tekevän mitään.. Tila 11 on ns. kielletty tila. Sitä ei ole määritelty ja piirin toiminta häiriintyy, jos salpa joutuu kyseiseen tilaan (suunnittelijan pitää siis huolehtia, että näin ei pääse tapahtumaan). Kyseisessä tilassa voi antosignaali (Q) saada kummantahansa arvon (0 tai 1) ja kumman arvon se saa, sitä ei voi etukäteen tietää!

Alla on esitetty SR-salvan piirrosmerkki, piirikaavio (eli kuinka salpa on mahdollista toteuttaa peruspiireillä) sekä toimintakaavio:

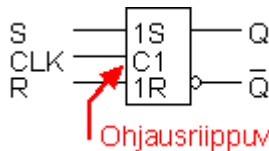


SR-salpa kello-otolla varustettuna

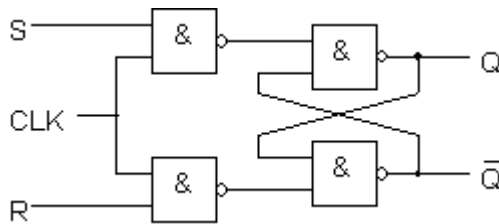
Kello-oton lisääminen salpaan tarkoittaa, että salpa toimii normaalisti vain silloin, kun kellosignaali on aktiivinen ($CLK = 1$). Tällöin piiri nukkuu kellosignaalin passiivisen ajan, eikä tuolloin reagoi ottosignaalien muutoksiin. Alla on kuvattu kello-otollisen SR-salvan piirrosmerkki, piirikaavio ja toimintakaavio. Kannattaa huomata piirrosmerkissä S ja R ottojen riippuvuus kellosignaalista. Riippuvuus on esitetty numerovastaavuudella:

Kellosignaalihan on luonteeltaan jaksollista vaihtelevaa signaalia (signaali on vuorotellen 1 ja 0). Siinä on aktiivinen ($CLK=1$) ja passiivinen ($CLK=0$) osuus. Jaksonaika on vakio (mikä se sitten kyseisellä kellolla onkaan)

Kello-otolla varustetun SR-salvan piirrosmerkki



Kello-otolla varustetun SR-salvan piirikaavio



Kello-otolla varustetun SR-salvan toimintataulukko

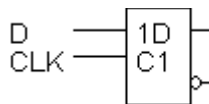
CLK	S	R	Q	Q̄	Tila
0	X	X	Q	Q̄	Ei muutu
1	0	0	Q	Q̄	Ei muutu
1	0	1	0	1	Nollattu
1	1	0	1	0	Asetettu
1	1	1	1	1	Kielletty

Kello-otolla varustettu D-salpa

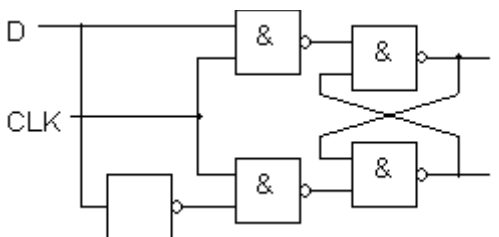
Äskeisessä SR-salvassa esiintyi ongelmatila ottosignaalien kombinaation ollessa $S=1$, $R=1$. Jos piiri jostain syystä joutuu kiellettyyn tilaan, menee piirin toiminta sekaisin. D-salvassa tilanne on korjattu estämällä pääsy kiellettyyn tilaan asettamalla $D=S=R'$. D-salpa siis toimii samoin kuin SR salpa paitsi että virhetoiminta kielletyn tilan suhteen on estetty. Tämä toisaalta tekee piiristä monimutkaisemman. Alla on esitetty D-salvan piirrosmerkki, piirikaavio ja toimintakaavio.

Siis kun kello-signaali on alhaalla ($CLK=0$), ei piiri reagoi ottosignaali D:n muutoksiin. Kun taas kello-signaali on ylhäällä ($CLK = 1$), määräytyvät antosignaalien Q ja Q' arvot ottosignaali D:n mukaan seuraavasti. Kun $D=0$ (ja $CLK=1$) salpa nollautuu eli $Q \rightarrow 0$ ja $Q' \rightarrow 1$. Mikäli taas $D=1$ (myös $CLK=1$ samalla), piiri asettaa tilansa eli $Q \rightarrow 1$ ja $Q' \rightarrow 0$.

D-salvan piirrosmerkki



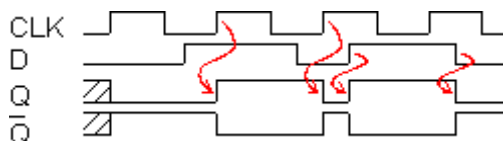
D-salvan piirikaavio



D-salvan toimintataulukko

CLK	D	Q	Q̄	Tila
0	X	Q	Q̄	Ei muutu
1	0	0	1	Nollattu
1	1	1	0	Asetettu

Kello-otolla varustetun salvan tapuksessa on muistettava, että kello-otto hallitsee salvan muita ottoja. Salvan muut otot siis toimivat vain kello-pulssin mukana. Tämä näkyy piirin toiminnassa siten, että piiri reagoi ottosignaalin muutoksiin vain kellon "aktivoitessa" salvan ($CLK=1$). Tässä on esimerkkinä kuvattu kello-otolla varustetun D-salvan aikakaavio **HUOM! Mikäli $CLK=0$, ei piirissä tapahdu mitään eli antosignaalit eivät vaihda arvoaan tällöin:**



Signaalinkäsittelytekniikan laboratorio

Digitaalitekniikan perusteet - luento 8

Kiikut (engl. flip-flops)

Kiikkujen funktio on säilyttää niihin tuotu data kellojakson yli ja välittää se tämän jälkeen eteenpäin. Kiikut ovat olennainen osa aikariippuvaisia piirejä, käytännössä tarkoitamme näillä pelkästään synkronisia sekvenssipiirejä. (Asynkroniset sekvenssipiirit ovat erikoistapaus. Niitä ei käsitellä tällä kurssilla sen enempää.) Synkronisen sekvenssipiirin kiikut muodostavat yhdessä rekisterin, jossa säilytetään piirin tilaa. Salvat eivät sovellu tähän tarkoitukseen, koska niiden tila voi muuttua kesken kellojakson (kuten jo aiemmin opittiin). Synkronisessa sekvenssipiirissä tilamuutokset tapahtuvat vain kellosignaalin tahdissa.

Kiikku muuttaa tilaansa vain kellosignaalin muuttuessa: tällöin se tarkastelee piirin nykytilaa ja sen hetkisiä ottosignaaleita ja tallettaa näiden aiheuttaman tuloksen seuraavaan kellojaksoon saakka. Sanotaan, että kellosignaali liipaisee kiikun. Tällä tarkoitetaan sitä ominaisuutta, että kiikku tarkastelee ympäristöänsä vain kellosignaalin tietyllä reunalla. Kiikut luokitellaan liipaisutavan perusteella. Kiikkutyypit:

- reunaliipaistava eli dynaaminen kiikku (edge-triggered flip-flop)
- vastakkaisreunoin liipaistava kiikku (data lock-out flip-flop)

Pulssiliipaistava kiikku ottaa dataa sisään kellopulssin nousevalla reunalla ja antaa sitä ulos kellopulssin laskevalla reunalla.

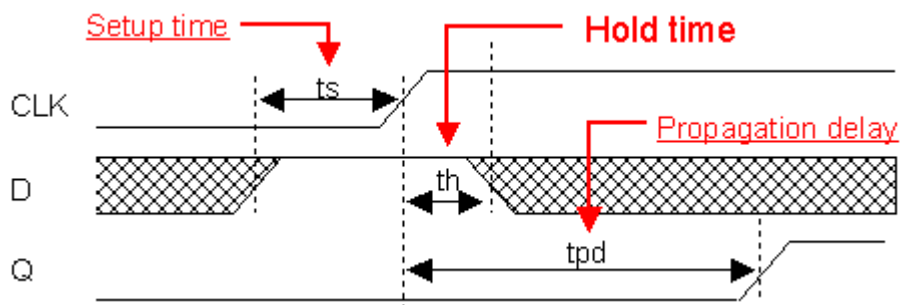
Reunaliipaistava kiikku vaihtaa tilaansa kellopulssin aktiivisella reunalla, joka on kellosignaalin joko nouseva tai laskeva reuna. On siis olemassa erikseen kellopulssin nousevalla ja laskevalla reunalla liipaistavia kiikkuja. Tilanmuutos näkyy kiikun annossa joko heti tai kellosignaalin toisensuuntaisen muutoksen jälkeen. Jälkimmäisessä tapauksessa puhutaan viivästetystä annosta.

Käytännön digitaalisuunnittelussa käytetään lähinnä pelkästään reunaliipaistavia kiikkuja. Eri kiikkutyypit on kuitenkin hyvä tuntea, erikoistilanteiden varalta.

Kiikkujen ajoituksesta

Kiikkujen yhteydessä on otettava huomioon kaksi eri varoaikaa sekä kiikusta itsestään aiheutuva viive. Varoajat ovat niitä ajanjaksoja, jolloin kiikku ei huomaa muutoksia piirissä. Käytännössä varoajat ovat kiikulle merkityksellisen kellopulssin reunan molemmin puolin. (Setup time eli asetus aika ennen kellopulssin nousua sekä Hold time eli pitoaika nousun aikana sekä vähän sen jälkeen.) Tämä tarkoittaa sitä, että **ottosignaalit eivät saa muuttua varoaikana tai piirin toiminta häiriintyy.**

Toinen ajoituksessa huomioitava seikka on kiikusta aiheutuva etenemisviive (propagation delay): signaalilla kestää vähän aikaa kulkea kiikun läpi (tyypillisesti muutaman kymmenen nanosekunnin luokkaa). Allaolevassa kuvassa esitetään reunaliipaistavan D-kiikun kriittiset aikajaksot:



Huomattavaa:

Tästä eteenpäin esitellään eri kiikkutyyppejä niiden toiminnan mukaan. Tärkeimmät loogiset kiikkutyypit ovat D, JK, T ja SR. Näistä esitellään pulssi- ja reunaliipaistavia versioita. (Kannattaa siis huomata, että jokaisesta loogisesta päätyypistä (D, JK, T, SR) voidaan tuottaa sekä reuna-, että pulssiliipaistavia kiikkuja.)

Pulssilla määräytyviä salpoja

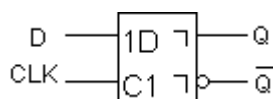
Pulssilla määräytyminen tarkoittaa sitä, että salpa lukee piirin tilan kellopulssin nousevalla reunalla ja antaa datan ulos kellopulssin laskevalla reunalla. Tässä esitellään esimerkkeinä pulssilla määräytyvistä salvoista D- ja JK -salvat. **HUOM! Salpojen totuustauluissa olevista merkinnöistä:**

Q(t) tarkoittaa salvan tilaa ennen kellopulssia.

Merkintä Q(t+1) tarkoittaa salvan seuraavaa tilaa ts. tilaa kellopulssin jälkeen (riippuu ottosignaaleista ja salpatyypistä)

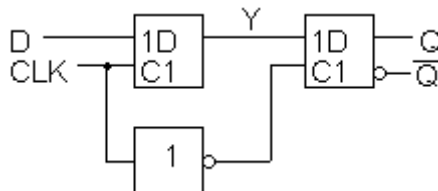
Pulssilla määräytyvä D-salpa

Pulssilla määräytyvän D-salvan piirrosmerkki ja totuustaulu.

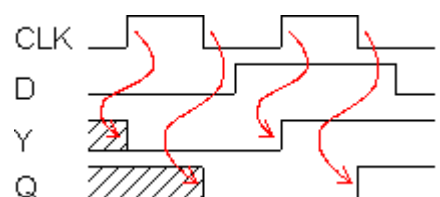


D	Q(t+1)	Tila
0	0	Nollautuu
1	1	Asettuu

Pulssilla määräytyvän D-salvan rakenne. (Voidaan muodostaa kahdesta D-salvasta, joiden rakenne oli selvitetty jo aiemmin.)



Pulssin aikakaavio

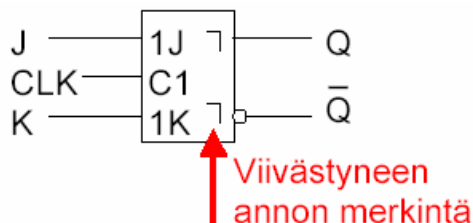


Pulssilla määräytyvä JK -salpa

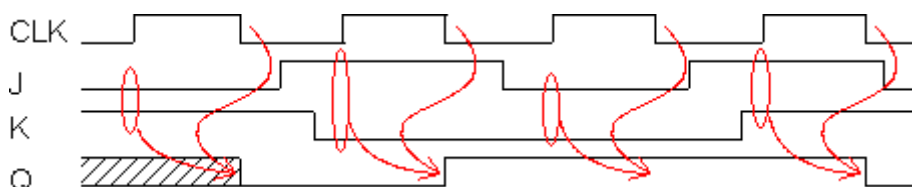
Pulssilla määräytyvän JK-salvan piirrosmerkki. Käytännössä salpa toimii kuten SR-salpa, mutta vaihtaa tilaansa aina edellisen tilan komplementtiin kun molemmat otot (J- ja K-otto) ovat ykkösiä.

JK-salvan totuustaulu

J	K	Q(t+1)	Tila
0	0	Q(t)	Ei muutu
0	1	0	Nollautuu
1	0	1	Asettuu
1	1	$\bar{Q}(t)$	Vaihtuu



Pulssin aikakaavio



Huomaa, että J ja K-ottojen tulee pysyä stabiilina koko CLK-pulssin ajan

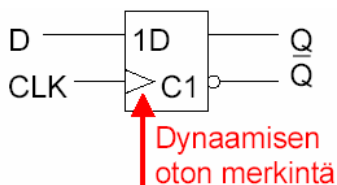
Reunaliipaistavia kiikkuja

Kiikun tilanmuutos tapahtuu, kun kellosignaali muuttaa tilaansa (kellosignaalin aktiivinen reuna). Vaihtoehdot ovat käytännössä, joko nousevalla tai laskevalla reunalla liipaisu tai vastakkaisreunoin liipaisu. Kuvissa kannattaa huomata viive. Esimerkkinä esitellään D- ja JK- kiikut.

Nousevalla reunalla liipaistava D-kiikku

Tilanmuutos tapahtuu, kun kellosignaali muuttuu nolasta ykköseen.

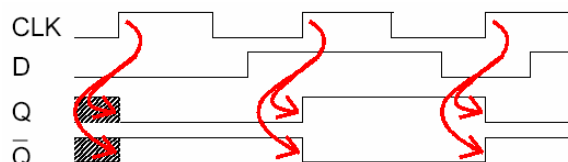
Piirrosmerkki



Totuustaulu

D	Q(t+1)	Tila
0	0	Nollautuu
1	1	Asettuu

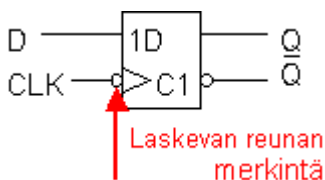
aikakaavio



Laskevalla reunalla liipaistava D-kiikku

Tilanmuutos tapahtuu, kun kello signaali muuttuu ykkösestä nollaan.

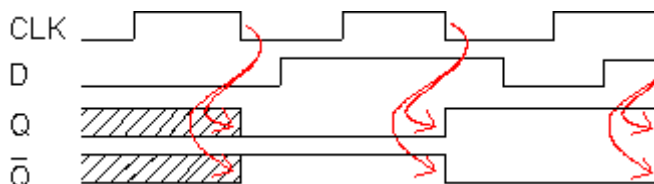
Piirrosmerkki



Totuustaulu

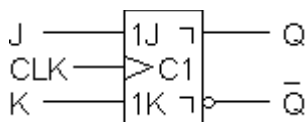
D	Q(t+1)	Tila
0	0	Nollautuu
1	1	Asettuu

aikakaavio



Vastakkaisreunoin liipaistava JK -kiikku (joka on siis käytännössä toiminnaltaan sama kuin pulssiliipaistava JK-kiikku)

Piirrosmerkki

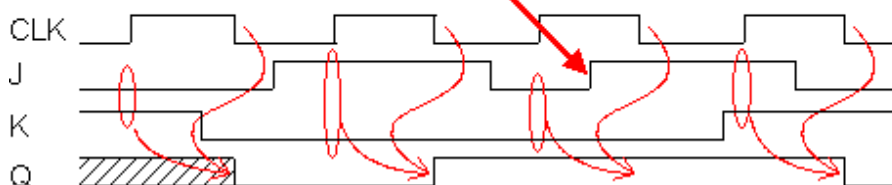


Aikakaavio

Otto saa muuttua, kun CLK = 1, kiikku ei huomaa muutosta muualla kuin liipaisukohdassa.

Totuustaulu

J	K	Q(t+1)	Tila
0	0	Q(t)	Ei muutu
0	1	0	Nollautuu
1	0	1	Asettuu
1	1	$\bar{Q}(t)$	Vaihtuu



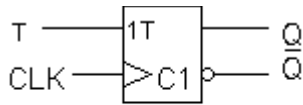
Kiikkuyhteenveto

Tässä kerrataan vielä lyhyesti kaikki peruskiikut. Mukaan on liitetty yksi aiemmin esittelemätön kiikkutyyppi: T-kiikku. Kaikkia allaolevia kiikkuja käytetään kurssilla. Käytännön digitaalisuunnittelussa yleisin suunnittelijan näkemä kiikkutyyppi on D-kiikku. On kuitenkin huomattava, että suunnittelussa on tiettyjä perustilanteita, esimerkiksi niin sanottua 'edge detectoria' suunnitellessa, joissa on ehdottomasti käytettävä muita kiikkutyppejä. Tämän vuoksi kaikkien kiikkutyyppien tuntemus on tärkeää. Samoin on osattava ja ymmärrettävä kiikkutyyppien toiminta.

T-kiikun toiminta on hyvin yksinkertainen. Kiikulla on kaksi ottoa: T ja CLK (kello). Jos T-ottoon tulee nollaa, säilyttää kiikku tilansa. Jos T-ottoon tulee ykköstä, komplementoi kiikku tilansa. Alla on esitetty nousevan reunan T-kiikun piirrosmerkki, sen totuustaulu ja aikakaavio.

Tilanmuutos tapahtuu, kun kello-signaali muuttuu nolasta ykköseen.

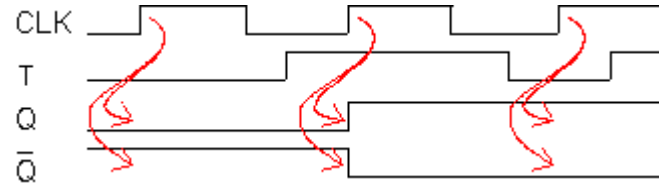
Piirrosmerkki



Totuustaulu

T	Q(t+1)	Tila
0	Q(t)	Ei muutu
1	$\overline{Q(t)}$	Vaihtuu

aikakaavio



Tällä kurssilla on osattava peruskiikut: D, JK ja T. Näistä D ja JK -kiikut ovat tärkeimmät. Alla on esitetty kaikkien peruskiikkujen totuustaulut:

D-kiikku

D	Q(t+1)	Tila
0	0	Nollautuu
1	1	Asettuu

JK-kiikku

J	K	Q(t+1)	Tila
0	0	Q(t)	Ei muutu
0	1	0	Nollautuu
1	0	1	Asettuu
1	1	$\overline{Q(t)}$	Vaihtuu

T-kiikku

T	Q(t+1)	Tila
0	Q(t)	Ei muutu
1	$\overline{Q(t)}$	Vaihtuu

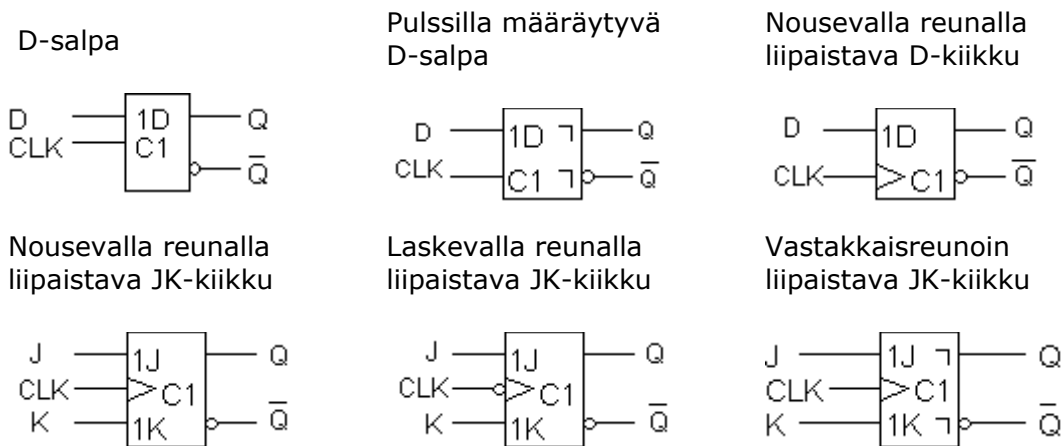
SR-salpa

SR	Q(t+1)	Tila
00	Q(t)	Ei muutu
01	0	Nollautuu
10	1	Asettuu
11	X	Kielletty

Jos halutaan etsiä yhtäläisyyksiä ja eroavaisuuksia perheen kiikkujen välillä, voidaan huomiota kiinnittää mm. seuraaviin seikkoihin:

- D-kiikku on viivästetty seuraaja: se siirtää oton signaalin antoon. Tästä syystä D-kiikku on yksinkertaisin käyttää synkronisen sekvenssi-piirin suunnittelussa.
- D-kiikku voidaan muodostaa JK-kiikusta yhdistämällä J ja K otto s.e. K otton edelle laitetaan invertteri.
- T-kiikku voidaan muodostaa JK -kiikusta yhdistämällä J ja K otot suoraan
- JK-kiikku on parannettu versio SR-kiikusta. (JK kiikun tapauksessa ei ole kiellettyä tilaa.)

Esimerkkejä kiikkujen piirrosmerkeistä



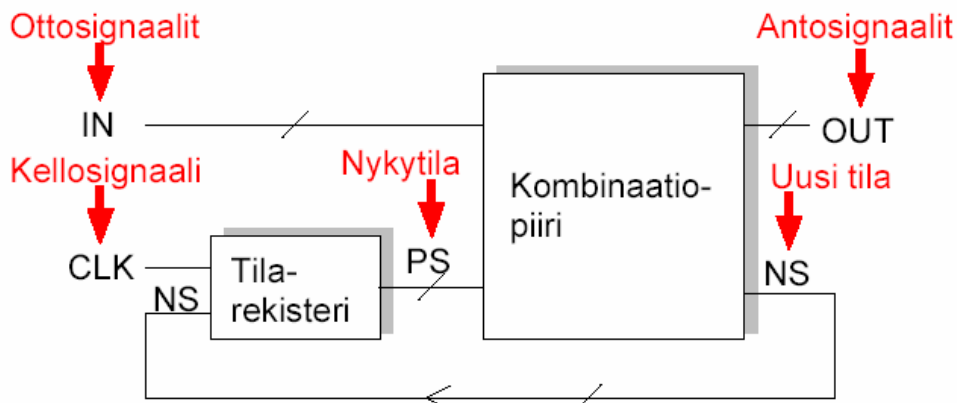
Synkroniset sekvenssipiirit

Vanhan kertauksena:

Digitaaliset piirit jaetaan kahteen pääluokkaan: kombinaatiopiireihin ja sekvenssipiireihin. Kombinaatiopiireissä antosignaalien arvot riippuvat vain ottosignaalien samanhetkisistä arvoista. Sekvenssipiirit ovat aikariippuvaisia eli arvojenannot riippuvat piirin tilasta, joka on talletettuna piiriin ja mahdollisesti myös ottosignaaleista. Käytännössä tämä tarkoittaa sitä, että sekvenssipiiri sisältää muistavia joitain komponentteja, yleensä kiikkuja. Sekvenssipiirit jaetaan edelleen kahteen alaluokkaan: asynkronisiin ja synkronisiin sekvenssipiireihin. Asynkroninen piiri vaihtaa tilaansa, kun jokin sen ottosignaaleista muuttuu. Synkronisessa sekvenssipiirissä on yksi hallitseva otto: kello-otto. Synkroninen sekvenssipiiri vaihtaa tilaansa vain tämän oton tahdissa. Asynkronisten piirien suunnittelu ja analyysi on työlästä ja hankalaa. Tällä kurssilla käsitellään vain synkronisia sekvenssipiirejä. Näiden analyysi ja suunnittelu on suhteellisen suoraviivaista ja opettaa digitaalilaitteiden toiminnan perusperiaatteita.

Synkronisen sekvenssipiirin rakenne ja toiminta

Synkroninen sekvenssipiiri koostuu kombinaatiopiiristä ja tilarekisteristä. Piirin ottosignaalit tulevat kombinaatiopiiriosaan ja antosignaalit lähtevät siitä. Tilarekisteri on erillinen rivi kiikkuja. Tilarekisteriin menee kombinaatiopiiristä piirin uusi tila (new state, NS) ja siitä palaa kombinaatiopiiriin piirin nykytila (present state, PS):



Yhteen kiikkuun voidaan tallettaa yksibittinen tieto, joko 0 tai 1. Näin ollen kullakin kiikulla on kaksi erilaista tilaa. Tarvittava kiikkumäärä riippuu piirin tilojen määrästä: **k:lla kiikulla voidaan toteuttaa 2^k tilaa**. Näin ollen kiikkujen ja tilojen välillä täytyy vallita seuraavanlainen suhde:

$$2^{k-1} < s \leq 2^k$$

missä s on piirin tilojen ja k tilarekisterin kiikkujen määrä.

Helpoin tapa muodostaa tilarekisteri on käyttää D-kiikkuja. Tällöin piirin uusi tila tuodaan suoraan D-kiikun ottoihin. Joskus käytetään myös JK -kiikkuja.

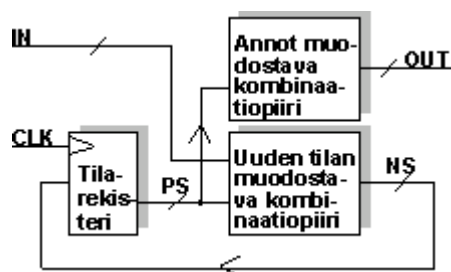
Periaatteessa kiikut uudistavat tilansa jokaisen kellopulssin aktiivisella reunalla: uusi tila menee kiikkuun ja sieltä saadaan nykytila. Käytännössä tässä täytyy kuitenkin muistaa kiikuista syntyvä viive. (Tilarekisterin annot muuttuvat vasta viiveen jälkeen.) Piirin ulkopuolella viiveet näkyvät tilasignaaleista riippuvien antosignaalien muutoksen viivästymisenä.

Synkronisten sekvenssipiirien pääluokat

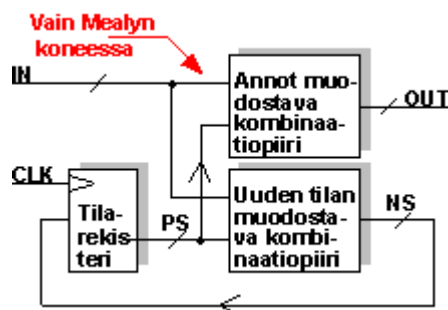
Synkronisessa sekvenssipiirissä piirin antosignaalit voivat riippua pelkästään tilasta, jossa piiri on (MOORE). Toinen vaihtoehto on, että annot riippuvat sekä piirin tilasta, että ottosignaaleista (MEALY). Synkronista sekvenssipiiriä, jonka antosignaali ei suoraan riipu mistään ottosignaalista vaan pelkästään piirin tilasta sanotaan Mooren koneeksi. Mikäli yksikin antosignaali riippuu jostakin ottosignaalista, on kyseessä Mealyn kone. Mealyn koneen sellaista antosignaalia, joka riippuu suoraan yhdestä tai useammasta ottosignaalista sanotaan ehdolliseksi annoksi.

Yleensä tietyn määrittelyn mukainen piiri voidaan toteuttaa joko Mooren koneena tai Mealyn koneena. Tällöin **Mooren kone -toteutuksessa on yleensä enemmän tiloja, mutta kombinaatiopiiri saattaa olla Mealyn kone -toteutusta yksinkertaisempi**. Alla Mooren ja Mealyn koneen rakenteet:

Mooren kone:
Vain piirin nykytila vaikuttaa antoihin. Antosignaalit muuttuvat kellon tahdissa.



Mealyn kone:
Nykytila ja ottosignaalit vaikuttavat antosignaaleihin. Antosignaalit voivat muuttua kesken kellojakson.



Graafinen esimerkki synkronisesta sekvenssipiiristä

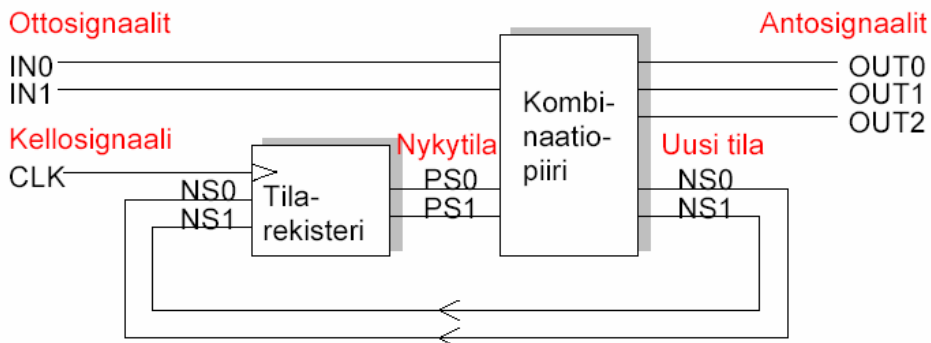
Tässä esitellään 4-tilaisen (=kaksi tilasignaalia) sekvenssipiirin lohkokaavio ja signaalien aikakaavio. Esitykset ovat suuntaa-antavia. Näiden kahden esityksen perusteella piirin toimintaa ei voida analysoida kokonaisvaltaisesti, koska molemmat jättävät liian paljon asioita avoimeksi. (Aikakaavio ei kuvaa signaalien kaikkia kombinaatioita, joten piirin toimintaan jää aukkoja.)

Esimerkin on tarkoitus tuoda aika- ja lohkokaavioita tutummiksi. Aikakaaviosta pitäisi pystyä ymmärtämään signaalien välisiä riippuvuussuhteita sekä viiveitä. Kummastakin tapauksesta on näkyvissä ainakin yksi selkeä esimerkki.

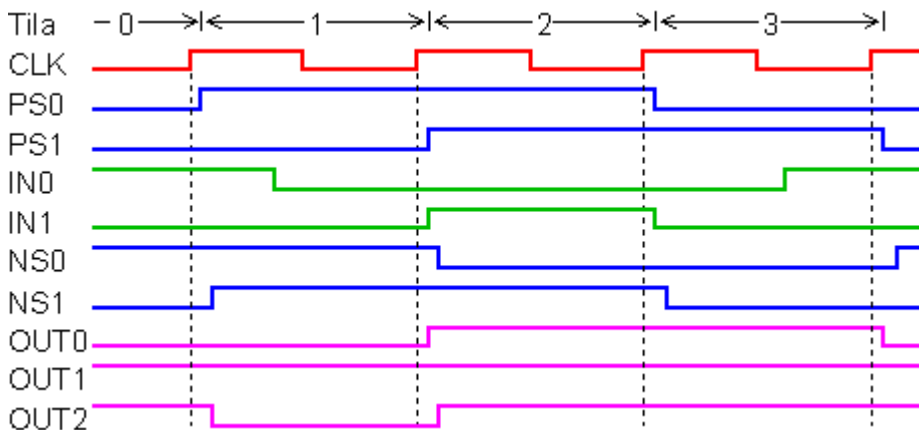
Viive näkyy aikakaaviossa tilasignaalien osalta parhaiten. Nykytilan signaalit (PS0 ja PS1) ovat viivästyneet hieman kun taas uuden tilan signaalit (NS0 ja NS1) ovat viivästyneet enemmän. Mikäli viivettä ei olisi lainkaan tapahtuisi muutos näissä signaaleissa *täsmälleen* samaan aikaan kellopulsin (CLK) muutoksen kanssa. Nyt ei kuitenkaan ole näin.

Signaalien välinen riippuvuussuhde näkyy kun tarkastellaan aikakaaviota seuraavasti: Nousevalla kellonreunalla (CLK) piirin "tarkastaa" ottosignaalit (IN0 ja IN1), sekä piirin nykytilan (PS0 ja PS1). Näiden perusteella piiri muodostaa uuden tilan (NS0 ja NS1) sekä antosignaalit (OUT0, OUT1 ja OUT2) (Lisäksi voidaan havaita, että OUT0 on täsmälleen sama signaali kuin PS0. Se on vain viivästynyt hieman)

Lohkokaavio



Aikakaavio



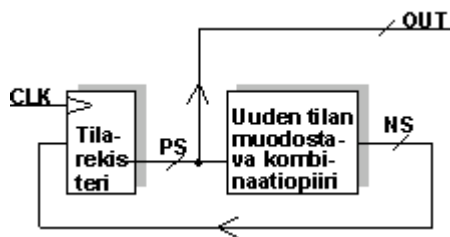
Synkronisten sekvenssipiirien erikoistapauksia

Mealyn kone on kaikkein yleisin synkroninen sekvenssipiiri. Jopa toinen pääluokka, Mooren kone, voidaan esittää Mealyn koneen erikoistapauksissa. Tällä sivulla käsitellään (kytkentämalleiltaan) erityyppisiä sekvenssipiirejä. Näistä ehdottomasti tärkein ryhmä on erilaiset laskurit (counter). Muut voi lukeista yleissivistyksen vuoksi.

Peruslaskuri ja ohjattava laskuri

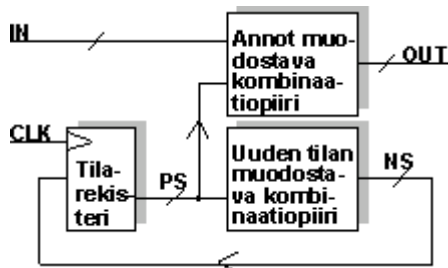
Peruslaskuri

Piirillä ei ole ottosignaaleja, se kiertää tilaketjua. Jokainen ketjun tila toistuu määrävlein esim. 0,1,2,0,1,2,jne.. Tilaketjun yksi kierros on laskentajakso, tilojen lukumäärä = laskentajakson pituus. Anto = piirin tila.



Ohjattava laskuri

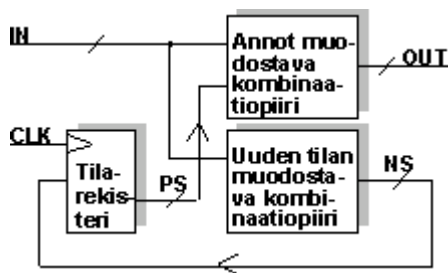
Tilakierto on edelleen itsenäinen, ottosignaalit eivät vaikuta siihen. Antoon taas vaikuttaa sekä senhetkinen tila että ottosignaali. (Tällöin antosignaali eivät välttämättä toistu samanlaisina, vaikka piirin tilakierto on vakio kuten peruslaskurissa.)



Muita erikoistapauksia

Sekvenssipiirin rajatapaus

on piiri, jossa uusi tila riippuu vain ottosignaalien arvosta eikä nykytilasta. On epäselvää, onko tämä enää oikeaoppinen sekvenssipiiri. Antosignaaliin vaikuttavat sekä nykytila että otot. Erikoistapaus tästä piiristä on synkronoiva rekisteri.

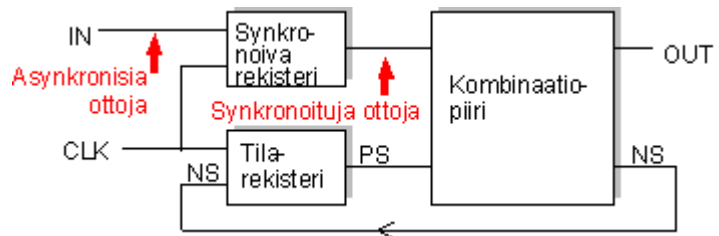


Synkronoiva rekisteri:



Asynkronisia ottoja

Edellisen piirityypin erikoistapaus, synkronoiva rekisteri, lisätään usein piiriin, jossa on kellosignaalista riippumattomia eli asynkronisia ottoja. Tällöin synkronoiva rekisteri tahdistaa asynkroniset ottoa muuhun piiriin s.e. synkronoitavat signaalit tuodaan kiikkujen ottoihin ja niiden annoista saadaan synkronoidut signaalit. Siis asynkroniset signaalit muutetaan synkronisiksi signaaleiksi.



Synkronisen sekvenssipiirin toteutusvaiheet

Sekvenssipiirin toteutus jakaantuu vaiheisiin. Työn kuluessa voidaan joutua palaamaan uudelleen aikaisempiin vaiheisiin.

1. Määrittely

Aluksi määritellään piirin toiminta. Määrittely on yleensä sanallinen kuvaus piirin toiminnasta. Piiriä tarkastellaan ulkoa päin, eikä sen otto- ja antosignaaleja vielä määritellä tarkasti. Sanallinen määrittely voi olla yksinkertaisille piireille pari lausetta, mutkikkaille hyvinkin laaja. Määrittelyyn voidaan liittää vuokaavio, jos se auttaa asiaa.

2. Lohkokaavion piirtäminen

Määrittelyn pohjalta laaditaan piirin lohkokaavio. Lohkokaavio kuvaa piirin toiminnalliset osat, ei varsinaista toimintaa. Monimutkainen piiri jaetaan mielekkäisiin lohkoihin, koska monen pienen lohkon suunnitteleminen on helpompaa kuin yhden suuren. Lohkokaavioon merkitään näkyviin lohkojen väliset signaalit sekä piirin ulkoiset signaalit.

3. Lohkojen suunnittelu

Määritellään lohko ja piirretään siitä tilakaavio joko tavallisena tilakoneesityksenä tai ASM -kaaviona (opetetaan myöhemmin). Tämän jälkeen toteutetaan lohkot käytettävissä olevilla piireillä ja piirretään piirikaaviot. (Nykyään on käytössä myös laitteiston korkean tason kuvauskieliä. Tällainen on esimerkiksi VHDL -kieli. Mikäli lohko kuvataan VHDL:ää käyttäen, ei välttämättä tarvita ollenkaan tilakone- tai ASM -mallia, vaan tietokone laatii tarvittavan mallin kuvauksen perusteella.)

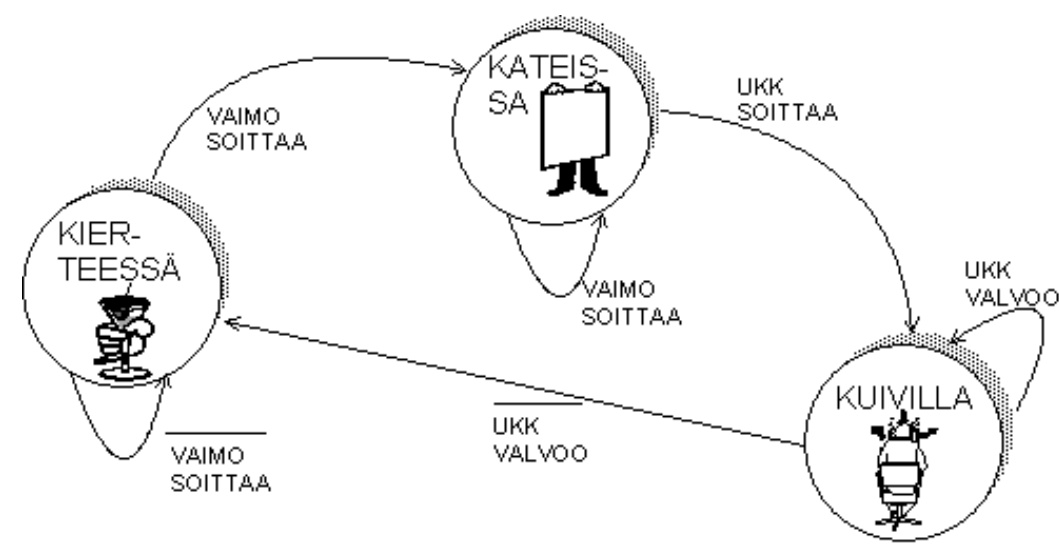
4. Piirin simulointi ja rakentaminen

Simuloidaan piiri käytettävissä olevilla simulaattorihjelmilla. Ohjelma matkii piirin toimintaa ja tutkii sen toimintaa erilaisilla ottosignaalisekvensseillä. Näin pyritään selvittämään, toimiiko piiri määrittelyn edellyttämällä tavalla. Virheiden löytyessä palataan takaisinpäin toteutusvaiheissa ja korjataan virheet. Tämän jälkeen rakennetaan prototyyppi ja tutkitaan sen toimintaa mittalaitteilla. Sekä palataan taas taaksepäin korjaamaan lisää ilmenneitä virheitä. Nykyisin pyritään havaitsemaan virheet simulointivaiheessa, koska se on huomattavasti halvempaa ja nopeampaa. Simulointi- ja rakennusvaiheissa virheitä löytyy usein moneen kertaan, joten korjaus ja uudelleen testaus -vaiheita täytyy uusia.

Loppukevennyksenä poliitikon tilakone

Sekvenssiipiirejä mallinnetaan siis tilakoneita piirtämällä. Näissä piirroksissa pyritään graafisesti esittämään, miten piirin toiminnan historia vaikuttaa piirin toimintaan tällä hetkellä tai tulevaisuudessa. Kansanomaisemmin ilmaistuna tilakaaviossa tarkastellaan kysymystä: Jos piiri aiemmin oli tietyssä tilanteessa (piirin nykytila) ja sai tietyn syötteen (ottosignaalit), niin minkälaiseen tilanteeseen piiri joutuu seuraavaksi (uusi tila) ja minne se tulee menemään, jollain tietyllä syötteellä?

Esimerkkinä tästä mallinnuksesta on alla poliitikon tilakone. Tässä tilanteessa tutkittavaa piiriä vastaa poliitikko. Poliitikko voi olla kolmenlaisessa eri tilanteessa (kierteessä, kateissa tai kuivilla). Piirin syötteitä (ottosignaalit) vastaavat vaimo ja UKK. Riippuen syötteistä poliitikko voi joko pysyä tilanteessa, jossa hän valmiiksi on tai siirtyä seuraavaan.



Signaalinkäsittelytekniikan laboratorio

Digitaalitekniikan perusteet - luento 9

Tilakaavio (state diagram) ja sen käyttö

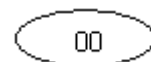
Synkronisella piirillä on joukko eri tiloja. Nämä tilat muodostavat takaisinkytkentöjä ja sijoittuvat piiri-/lohkokaaviossa seuraavasti: **'uusi tila' on se tilasignaalien joukko, joka menee tilarekisterin kiikkuihin sisään. 'Nykytila' on se tilasignaalien joukko, joka saadaan kiikuista ulos.** Synkroniset sekvenssiipiirithän koostuivat tilarekisteristä (= joukko kiikkuja) sekä kombinaatiopiiriosasta. Synkroninen sekvenssiipiiri on siis tilakone.

Tila kuvaa aikariippuvuutta ja sillä on erilaisia ominaisuuksia joita ovat mm. tilakoodi sekä tilaan liittyvät annot. Tilakoodi vastaa suoraan niitä nollia ja ykkösiä, joita saadaan kiikuista piiriin, kun ollaan kyseisessä tilassa. (Esim. 4 kiikkua, tilakoodi 0110. Kyseisessä tilassa kiikkujen annot ovat 0, 1, 1, ja 0.) Jos tilaan liittyy anto, joka on riippuvainen vain tilasta (ei lainkaan ottosignaaleista, esim. Mooren koneessa on vain tilaan liittyviä antoja), kirjoitetaan anto kauttaviivan toiselle puolelle muodossa: tilakoodi / tilaan suoraan liittyvä anto. (Ks. myös graafinen esimerkki alemmalla)

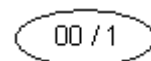
Tilasta voidaan liikkua toiseen tilaan tai pysyä samassa tilassa. Tilasta toiseen liikkuminen tapahtuu tiettyjen tilaehtojen avulla. Käytännössä uusi tila saavutetaan aina seuraavan kellojakson alussa. (Synkronisen sekvenssiipiirin ominaisuuksiinhan kuuluu, että piiri vaihtaa tilaansa vain kellopulssein tahdissa.) Tilasiirtymä kuvataan tilakaaviossa nuolella. Nuolen yläpuolelle kirjoitetaan sen ottosignaalin kyseinen arvo, joka mahdollistaa siirtymän tilasta toiseen. Jos lähtötilaan liittyy ottosignaalista riippuvia antoja, ilmoitetaan annot kauttaviivan toisella puolella muodossa: ottosignaali / signaalista riippuva anto.

... ja sama graafisin esimerkein

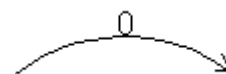
Tilakaaviossa jokaista piirin tilaa kuvaa soikio. Soikion sisällä on kyseisen tilan tilakoodi (Esim. tässä tapauksessa piiri on tilassa 00)



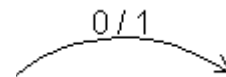
Jos tilaan liittyy anto, **joka riippuu vain tilasta**, ilmaistaan se tilan yhteydessä kauttaviivan toisella puolella seuraavasti: (esim. tässä tapauksessa kun piiri on tilassa 00, on piirin antosignaali 1)



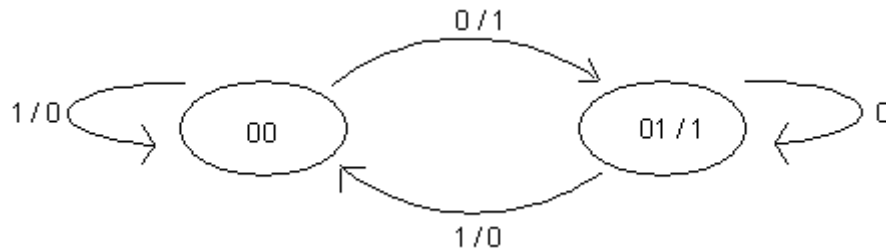
Tilasiirtymät tilasta toiseen tilaan esitetään nuolilla. Nuolen yläpuolelle merkitään sen ottosignaalin kyseinen arvo, joka mahdollistaa tilasiirtymän (esim. tässä tapauksessa ottosignaalin tulee olla 0):



Jos lähtötilassa on ottosignaalista riippuva anto, merkataan kyseinen anto ottosignaalin seuraksi tilasiirtymään kauttaviivan toiselle puolelle (esim. tässä tapauksessa mikäli ottosignaali saa arvon 0 ja suoritetaan tilasiirtymä, antaa piiri antosignaalin arvoksi 1:n)



Yksinkertainen tilakaavio voisi näyttää vaikka tältä: (Esimerkki olisi tosielämässä hyvin kömpelö. Siihen on lähinnä pyritty mahdutettamaan kaikki pääasiat tilakaaviosta.)



Tätä ylläolevaa tilakaaviota voidaan tulkita seuraavalla tavalla:

- Kun piiri on tilassa 00 (tilakoodi=00) ei näytä tapahtuvan mitään.
- Kun tilassa 00, ottosignaali on 1. Tällöin piirin "ulosanti" eli antosignaali saa arvon 0.
- Kun tilassa 00, ottosignaali onkin 0, antaa piiri antosignaalina 1:en ja siirtyy samalla tilaan 01
- Tilassa 01 antosignaali on koko ajan 1
- Kun tilassa 01 ottosignaali on 0, piiri vain pysyy samassa tilassa kuin ennenkin (siis tilassa 01)
- Kun taas tilassa 01 ottosignaali onkin 1, vaihtaa piiri tilaansa takaisin tilaan 00 ja samalla antosignaalin arvoksi tulee 0.

ASM (Algorithmic State Machine)

ASM -kaavio on perinteisen tilakaavion graafisesti havainnollisempi muoto. ASM -kaavio ei ole yhtä yleismaailmallinen kuin perinteinen tilakaavio, mutta se on toisaalta huomattavasti selkeämpi ja siten takaisinpäin synkronisten sekvenssiipiirien suunnittelussa.

ASM kaavio piirretään ASM:n toiminnallisen määrittelyn ja lohkokaaavion perusteella. Se kuvaa yksikäsitteisesti synkronisen sekvenssiipiirin toiminnan ja sisältää kaiken piirin toteuttamisen kannalta tarpeellisen informaation. ASM -kaaviossa on symboleja (tila, anto, päätöslohko sekä ehdollinen anto) ja niihin liittyviä merkintöjä. Symbolit kuvaavat siis ASM:n tiloja, tilasiirtymiä ja niihin liittyviä ehtoja. Merkinnät antavat lisätietoja.

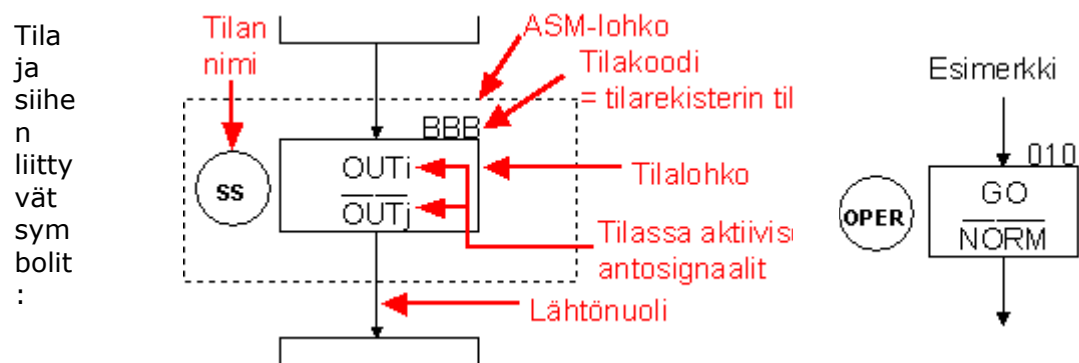
ASM -kaavion pääkohdat

Myöhemmissä esimerkeissä (luennot 9 ja 10) esitellään ASM-kaavion käyttöä osana synkronisen sekvenssipiirin suunnittelua. Tässä käydään kuitenkin lyhyesti yhteenvedona läpi tärkeimmät ASM -kaavion merkinnät:

- tila = suorakaide, tilakoodi ilmoitetaan suorakaiteen oikeassa yläkulmassa
- tilan nimi on ilmoitettu ympyrän sisällä, tilaa kuvaavan suorakaiteen vieressä
- päätöslohko = ehto tilasiirtymälle (ottosignaalin nimi) = salmiakki, jossa on ehtona päätökseen vaikuttavan ottosignaalin nimi. Vaihtoehtojen (ottosignaalin arvo 1 tai 0) mukaan lähtee nuolet salmiakin kulmista uusiin (päättöstä seuraaviin) tiloihin
- tilaan liittyvä antosignaali = antosignaalin nimi, joka on kirjoitettu tilalohkoon sisälle. Tämä antosignaali toteutuu aina ko. tilassa
- anto, joka riippuu suoraan ottosignaalista = ehdollinen anto = pyöristetty suorakaide. Tämä seuraa päätöslohkon jälkeen, jolloin ottosignaali on siis vaikuttanut antosignaaliin (= ehdollisuus). Tätä käytetään Mealyn-koneissa. Antosignaali saa ko. arvon ainoastaan mikäli ottosignaali toteutuu (vrt. normaali antosignaali)
- tilasiirtymä = jos tilasta A voi siirtyä tilaan B, on niiden välillä nuoli. Nuoli voi kulkea erilaisten päätöslohkojen kautta.

ASM:n toiminnalliset osat (Tärkeää osata tulevia harjoituksia varten.)

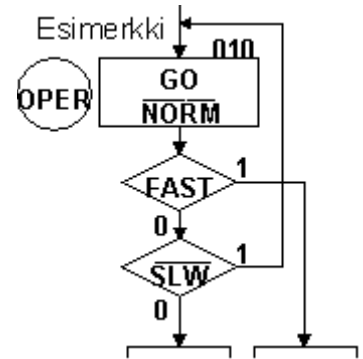
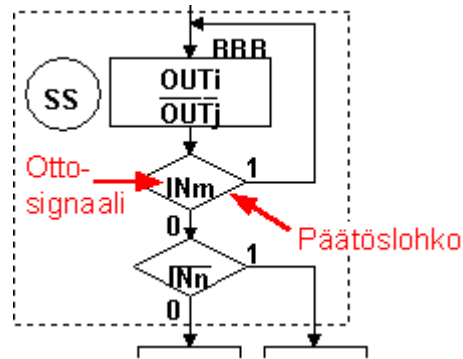
ASM (Algorithmic State Machine) -kaavion osat



Esimerkin tulkinta:

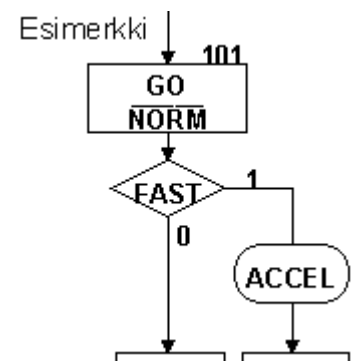
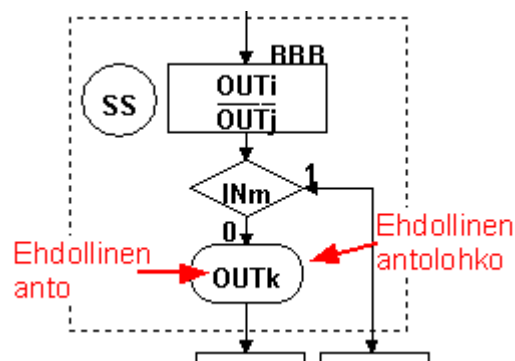
- Kyseisen tilan tilakoodi (tilan oik. yläkulmassa) on 010 (Tätä tarvitaan myöhemmin, kun kytketään kiikkuja yhteen)
- Tilan nimi on OPER
- Tilan annot ovat: GO sekä (NORM)

Päätöslohko - huomattavaa: ottosignaalin vaikutus kuvataan päätöslohkolla. Jokaista tilanmuutokseen vaikuttavaa ottosignaalia varten on oma päätöslohko. Lohkot ovat peräkkäin, eivät rinnakkain.



- Tilan tilakoodi on 010
- Tilan nimi on OPER
- Tilan annot ovat: GO sekä (NORM)'
- Ottosignaaleina ovat FAST sekä (SLW)'. Näistä ottosignaaleilla FAST on korkeampi prioriteetti, joten se "tarkastetaan" ensiksi.
- Mikäli otto FAST = 1, etenee laite uuteen tilaan (uuden tilan tilakoodia ei kuvassa näy). Tässä tapauksessa on merkityksetöntä, mikä on ottosignaali (SLW)':n arvo.
- Mikäli otto FAST = 0, siirrytään tarkastelemaan ottoa (SLW)'.
- Mikäli (SLW)' = 0, edetään uuteen tilaan (jonka tilakoodia ei kuvassa näy). (Edellyttäen, että FAST=0)
- Mikäli (SLW)' = 1, edetään takaisin vanhaan tilaan (eli pysytään koko ajan samassa tilassa OPER, jonka tilakoodi on 010)

Ehdollinen anto: anto, joka riippuu suoraan jostain ottosignaalista

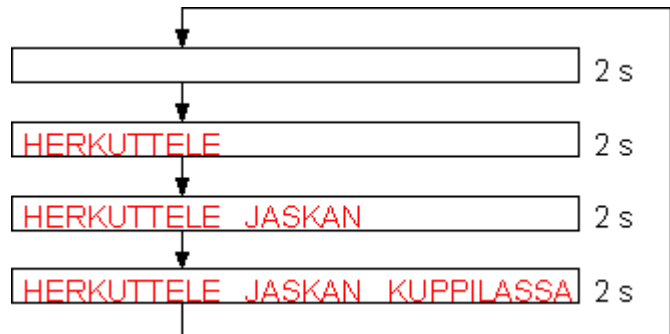


- Tilan tilakoodi on 101
- Tilan annot ovat: GO sekä (NORM)'
- Ainoana ottosignaalina on FAST.
- Mikäli otto FAST = 0, etenee laite uuteen tilaan (uuden tilan tilakoodia ei kuvassa näy).
- Mikäli otto FAST = 1, etenee laite uuteen tilaan (uuden tilan tilakoodia ei kuvassa näy). Samalla laite antaa myös antosignaalin ACCEL (eli ACCEL=1, muutenhan sitä ei merkitä ASM-kaavioon lainkaan)
- **Huom! Antosignaali ACCEL on siis ehdollinen anto, jonka laite antaa vain jos otto FAST=1**

ASM -esim 1: yksinkertainen tilaketju

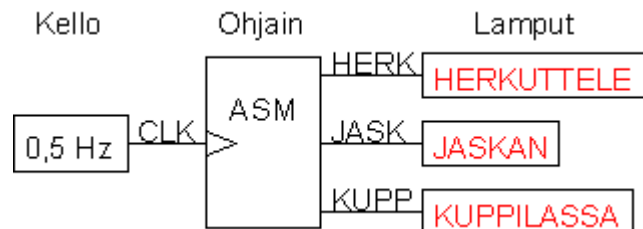
Toiminnan määrittely

Suunnitellaan valomainos, jossa teksti vilkkuu oheisen kaavion mukaan. Valopaneeli on ensiksi tyhjä, sitten paneeliin ilmestyy yksi kerrallaan sanat HERKUTTELE, JASKAN ja KUPPILASSA. Tämän jälkeen näyttö tyhjenee jälleen. Jokainen erilainen valonäyttö on yksi tila. Sanat ovat laitteen antosignaaleja. Kellojakso on 2 sekuntia:



Lohkokaavio

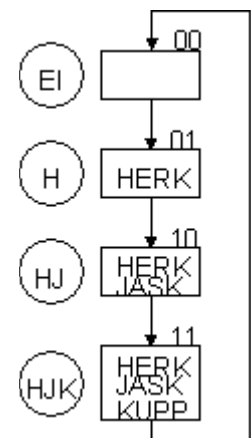
Lohkokaavio määrittelee piirin toiminnalliset osat. Laativinen ei ole välttämätöntä, mutta voi selkeyttää suunnittelua isompien piirien ollessa kyseessä. Tässä esimerkissä rakennetaan neljätilainen sekvenssiipiiri. Tilat toistuvat peräkkäin aina samanlaisina ja samassa järjestyksessä (piiri on siis teoreettisesti laskuri, joka laskee $0 \rightarrow 3$), joten ottosignaaleja ei kellon lisäksi ole:



ASM -kaavio

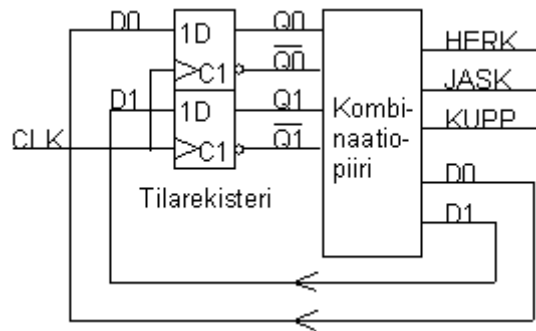
Jokaiselle tilalle täytyy määritellä tilalohko, nimi ja tilakoodi. Tilakoodit voidaan valita halutusti, kunhan se on **jokaiselle tilalle yksilöllinen**. Yleensä pyritään nimeämään tilat jossakin loogisessa järjestyksessä. (Esim. 4 tilaa: 00, 01, 10, 11). Tilakoodin koko pyritään pidetään mahdollisimman pienenä, koska mitä suurempi tilakoodi on, sitä enemmän tarvitaan kiikkuja. Käytännössä bittien määrä juontuu 2-kantajärjestelmän mukaisesti. Mikäli piirissä on s tilaa, tarvitaan bittejä $k = \log_2 s$ kokonaisluvuksi pyöristettynä. Ylimääräisiä bittikombinaatioita kutsutaan käyttämättömiksi tiloiksi. Tilakoodien valinta vaikuttaa siis piirin monimutkaisuuteen. Suoraan ei kuitenkaan voida päätellä, mikä koodivalinta olisi optimaalinen.

Tilat voidaan nimetä mielivaltaisella tavalla. Järkevää on kuitenkin nimetä ne nimillä, jotka jotenkin kuvaavat ko. tilaa. Esimerkissämme tilat on nimetty: EI (näytöllä ei ole mitään, *tilakoodi*=00), H (näytöllä lukee HERKUTTELE, *tilakoodi*=01), HJ (näytöllä lukee HERKUTTELE JASKAN, *tilakoodi*=10), sekä HJK (näytöllä lukee HERKUTTELE JASKAN KUPPILASSA, *tilakoodi*=11)



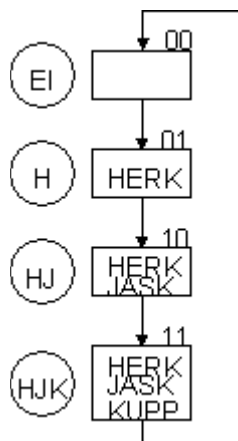
Jos kaivataan täsmällisempää lohkokaaaviota, voidaan se tehdä esim. ASM-kaavion perusteella:

D-kiikkujen määrä (2 kpl) saadaan suoraan tilakoodin suuruudesta (4 tilaa = 2 numeroa). Se, että kiikut ovat juuri D-kiikkuja on vedetty hatusta ja kiikkujen tyyppi otetaan huomioon kunnolla vasta kombinaatiopiiriä suunnitellessa.



Tässä kuvassa Q0 ja Q1 (sekä niiden komplementit) ovat nykytilan signaaleja ja D0 sekä D1 ovat uuden tilan signaaleja. Kombinaatiopiiri on esitetty vasta lohkokaaaviotasolla.

Kombinaatiopiirin toteutuksen suunnittelu



ASM -kaaviosta saadaan tilataulukko:

nykytila = tila, joka tulee kiikuista

uusi tila = tila, joka menee kiikkuihin

Nykytila		Uusi tila		Annot		
Q1	Q0	D1	D0	HERK	JASK	KUPP
0	0	0	1	0	0	0
0	1	1	0	1	0	0
1	0	1	1	1	1	0
1	1	0	0	1	1	1

Tilataulu (joka siis muistuttaa totuustaulua) muodostetaan seuraavasti:

- Aloitetaan ASM-kaavion ensimmäisestä tilasta nimeltä "EI" (tila 00), joka merkitään nykytilaksi.
- Merkitään samalle riville uudeksi tilaksi se tilakoodi, johon ASM:n nuoli osoittaa vanhasta tilasta siis tila "H" (eli tilakoodi = 01).
- "Annot"-kohtaan kirjoitetaan mitkä ovat piirin antosignaalit tilassa 00, ja niitä ei ole (valomainos on siis tyhjä), joten kaikkiannot on merkattu nolllaksi.

Käydään vielä tilataulun toinen rivi läpi...

- Merkataan nykytilaksi tila 01 (siis tila "H"). Tästä tilasta lähtee nuoli seuraavaan tilaan, jonka tilakoodi on 10.
- Uusi tila on siis 10 eli tila "HJ"
- Annot tilassa 01 ovat: HERK=1, JASK=0 ja KUPP=0. Tämä tarkoittaa sitä, että kyseisessä tilassa valotaululla palaa ainoastaan sana "Herkuttele"

Samalla tavalla käydään läpi kaikki tilat ja lopuksi saadaan muodostettua tilataulu, joka yllä oli esitetty

Tilataulukon perusteella saadaan lausekkeet tilakiikkujen D-otoille. Koska tilasiirtymään vaikuttaa vain edellinen tila, tutkitaan vain taulun osia nykytila ja uusi tila.

Tässä kohtaa tulee vaikuttamaan se, että on valittu D-kiikku toteutus. Toteutus on nyt helppo ja menee seuraavalla tavalla:

Tehdään taulukko, joka kertoo meille D-kiikkujen ottosignaalit.

Nyt tuijotetaan ensin tilataulukon **sarakkeita Q1 ja D1**. Tämä kertoo meille miten kiikku nro1 muuttaa tilaansa. Siis kysymys kuuluu: Mikä on D-kiikun oton (eli D:n) arvo kun tila muuttuu

Q1 --> D1
0 --> 0
0 --> 1
1 --> 1
1 --> 0

(eli tässä on sarakkeet Q1 ja D1 kopioitu tilataulusta)

Muistellaan D-kiikun muutostaulua (ks. edellinen luento "kiikkuyhteen veto"), ja sehän oli seuraavanlainen.

D	Q(t+1)	Tila
0	0	Nollautuu
1	1	Asettuu

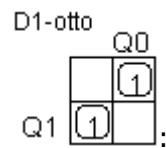
Ja tästä saadaan siis D-kiikku nro1:en D-oton arvoiksi:

Q1 --> D1 ==> **D1-otto**
0 --> 0 ==> **0**
0 --> 1 ==> **1**
1 --> 1 ==> **1**
1 --> 0 ==> **0**

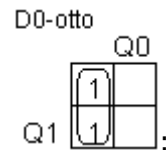
Ja tästä saadaan siis D-kiikku nro0:n D-oton arvoiksi:

Q0 --> D0 ==> **D0-otto**
0 --> 1 ==> **1**
1 --> 0 ==> **0**
0 --> 1 ==> **1**
1 --> 0 ==> **0**

Tästä saadaan sitten Karnaugh'n avulla varsinaiset lausekkeet kiikkujen otoile (sijoitellaan 1:set jo opittuun tapaan kartalle) :

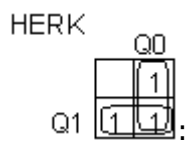


$$D1 = Q0Q1' + Q0'Q1$$

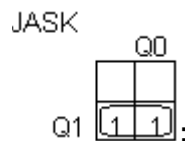


$$D0 = Q0'$$

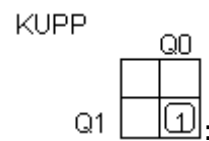
Seuraavaksi määritellään lausekkeet annoille (sijoitellaan 1:set Karnaugh'n kartalle):



$$HERK = Q0 + Q1$$

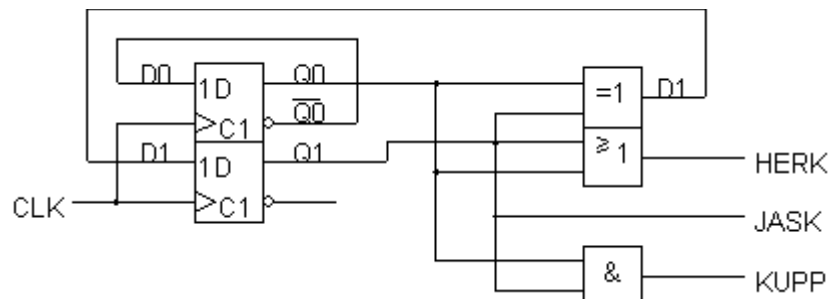


$$JASK = Q1$$



$$KUPP = Q0Q1$$

Tämän jälkeen toteutetaan lausekkeet porttipiireillä (ja muistetaan, että kiikuista lähtee **ulos nykytila**):



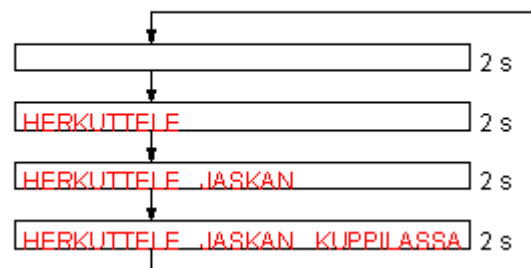
ASM esim 2: Piiri, jossa on tilanmuutoksiin vaikuttavia ottosignaaleja

Toiminnan määrittely

Monimutkaistetaan edellistä mainosta hieman. Lisätään siihen kytkin, joka pitää piirin joko vakio-tilassa, jolloin koko teksti on näkyvissä tai sitten aiheuttaa normaalin tilakierron. Piiri toimii siis oheisen kaavion mukaan:

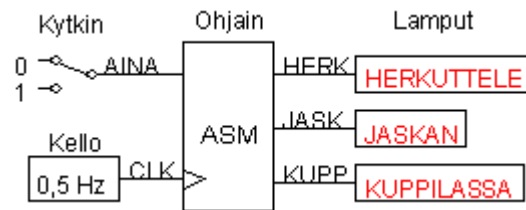
AINA = ON HERKUTTELE JASKAN KUPPILASSA

AINA = EI



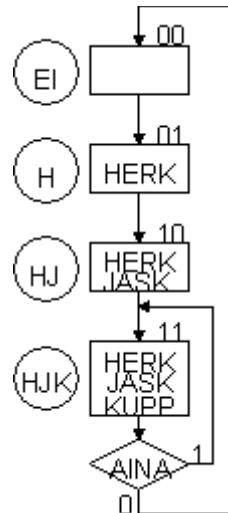
Lohkokaavio

Määrittelyn mukaan piirin toimintaan vaikuttaa yksi ottosignaali: "AINA". Tämän vaikutus täytyy näkyä lohkokaaviossa. Muuten lohkokaavio pysyy edellisen esimerkin mukaisena.

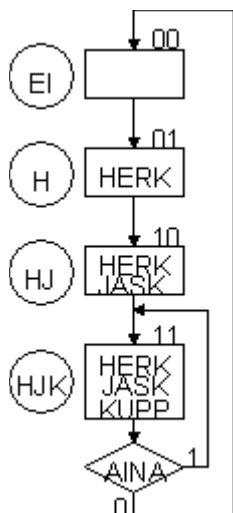


ASM -kaavio

Ottosignaalin vaikutus näkyy ASM -kaaviossa päätöslohkona (salmiakki): jos AINA = 1, palataan aina takaisin HJK -tilaan, jossa kaikki sanat siis palavat yhtäaikaan. Jos AINA = 0, kierretään tavallista tilaketjua. ASM kaavio voitaisiin myös piirtää siten, että ottosignaali AINA tarkistettaisiin jokaisessa tilassa. (Nythän, jos AINA muuttuu ykköseksi jossain tiloista EI, H, HJ, huomataan muutos vasta tilassa HJK.) Jos tilanmuutosta tarkkailtaisiin useammin, toimisi laite tarkemmin speksien mukaan. Valomainoksen tapauksessa asia ei kuitenkaan ole oleellinen.



ASM esim 2, piirin toteutuksen suunnittelu



ASM -kaaviosta saadaan tilataulukko:
 nykytila = tila, joka tulee kiikuista
 uusi tila = tila, joka menee kiikkuihin
 X-merkintä tilataulukossa tarkoittaa sitä, että kyseisellä ottosignaaliilla ei ole vaikutusta ts. aivan sama onko se 0 vai 1.

Koska X-merkintä vastaa 0 tai 1:stä niin rivi 00X onkin kaksi riviä (000 ja 001) Jos kaikki rivit kirjoittaisi tällä tavalla "auki" saataisiin normaalin kokoinen tilataulukko (siis 8 rivinen 3:lla muuttujalla, koska $2^3=8$)

Nykytila		Ottosignaali	Uusi tila		Annot		
Q1	Q0		D1	D0	HERK	JASK	KUPP
0	0	X	0	1	0	0	0
0	1	X	1	0	1	0	0
1	0	X	1	1	1	1	0
1	1	0	0	0	1	1	1
1	1	1	1	1	1	1	1

Tilataulukon perusteella saadaan lausekkeet tilakiikkujen D-otoille. Kiikkuja on edelleen kaksi, mutta niihin vaikuttaa yhteensä kolme signaalia:

molemmat nykytilat sekä otto AINA.

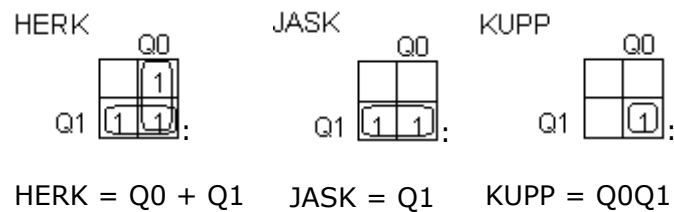
Tässäkin tapauksessa molempien kiikkujen D-otot vastaavat suoraan D1 ja D0 sarakkeita

Karnaugh'n avulla saadaan lausekkeet kiikkujen otille:

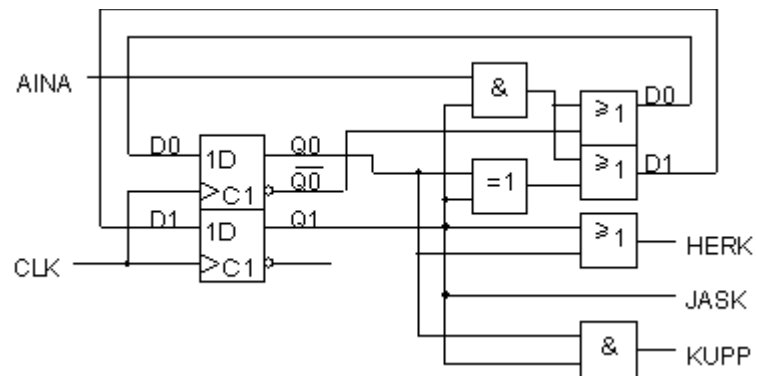


$$D1 = Q0Q1' + Q0'Q1 + Q1AINA \quad D0 = Q0' + Q1AINA$$

Seuraavaksi määritellään lausekkeet annoille. Kannattaa huomata, että annot ovat riippuvaisia vain tiloista, ei ottosignaaleista:



Tämän jälkeen toteutetaan lausekkeet porttipiireillä:



Signaalinkäsittelytekniikan laboratorio

Digitaalitekniikan perusteet - luento 10

Mooren vs. Mealyn kone -toteutus sekvenssipiirille

Käsitellään esimerkin valossa kahta eri toteutustapaa Mooren konetta ja Mealyn konetta

Muistutukseksi / kertauksena:

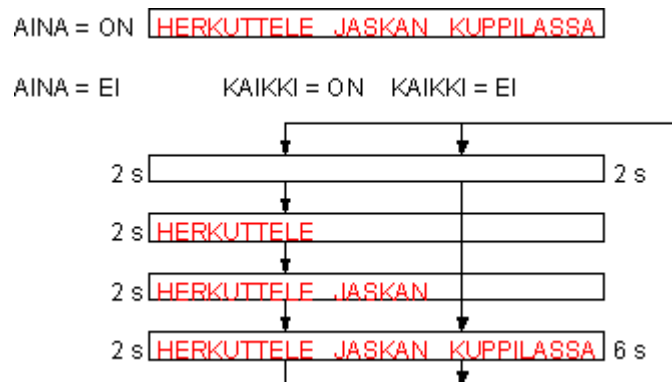
Mooren koneessa ovat kaikki antosignaalit riippuvat **vain** piirin tilasta. Kaikki annot on siis ASM-kaaviossa merkitty suorakulmioiden sisään.

Mealyn koneessa esiintyy myös ehdollisia antoja eli antosignaali riippuu sekä piirin tilasta *että* ottosignaaleista. Tämähän merkittiin ASM-kaaviossa päätöslohkon jälkeen pyöristetyn suorakulmion sisään.

Kun valitaan Mooren ja Mealyn kone- toteutuksen välillä muistetaan vielä, että **Mooren kone -toteutuksessa on yleensä enemmän tiloja, mutta kombinaatiopiiri saattaa olla Mealyn kone -toteutusta yksinkertaisempi**

Tehtävä

Tehtävän anto on monimutkaistaa aikaisemmin esitettyä valomainosta siten, että siihen tulee yksi toiminto lisää. (Tähän mennessähän vaihtoehtoja oli kaksi: joko kaikki sanat palavat koko ajan tai mainos kiertää vanhaa tilaketjua normaalisti). Lisätään näihin vaihtoehtoihin vielä yksi vaihtoehto: jos ottosignaali KAIKKI = 0, kierretään pelkästään tilaketjua tyhjä taulu --> kaikki sanat taululllla --> tyhjä taulu.
Jos KAIKKI = 1, kierretään normaalia tilaketjua. AINA -signaalin toiminta pysyy ennallaan. Kaavio selventänee.

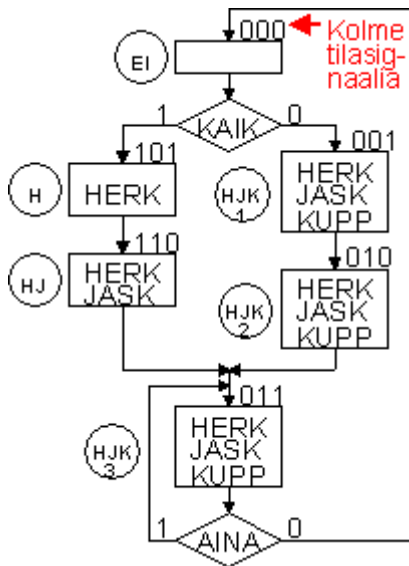


Piirin toteutus

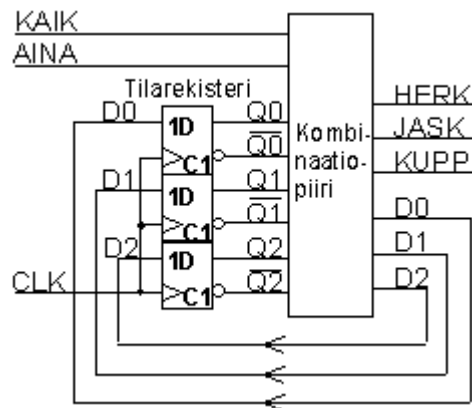
Mooren kone -toteutus monimutkaiselle valomainokselle

Piirin ASM-kaavion teko lähtee liikkeelle valomainoksen tyhjän taulun tilasta (= EI -tila) seuraavasti:

EI-tilassa pitää haarautua sen mukaan, pysytäänkö entisessä tilasekvenssissä (tilakierrossa, jossa kaikki valot syttyvät erikseen 2 sekunnin välein) vai mennäänkö tilaan, jossa kaikki valot palavat 6 sekuntia. Näin ollen EI ja HJK (= kaikki sanat palavat) välille tulee kaksi erilaista tilakiertomahdollisuutta. Molemmat tilakierto mahdollisuudet tarvitsevat kaksi tilaa. Lisäksi tilakierroille on yhteistä ensimmäinen (EI-tila) ja viimeinen (HJK-tila) tila. Tästä johtuen => Tilakoneeseen tarvitaan yhteensä kuusi tilaa (ks. ASM-kaavio alla). Tästä johtuen tilakoodin pituudeksi tulee kolme bittiä (joilla saadaan toteutetuksi $2^3=8$ tilaa):



KAIK-signaali määrää EI -tilasta lähdetessä, kumpaan haaraan mennään. Sekvenssin vaihto yhden tietyn haaran aikana ei ole mahdollista. (Muitakin toteutusmahdollisuuksia olisi). Piirin lohkokaavio on esitetty alla. Kiihkuja tarvitaan kolme, koska tilabittejä on 3 (ja tiloja siis 6). Tässä on valittu D-kiihku toteutus, mutta se ei suinkaan ole ainoa vaihtoehto



Tilataulukko

Nykytila	Otot	Uusi tila	Annot
Q2 Q1 Q0	AINA KAIK	D2 D1 D0	HERK JASK KUPP
0 0 0	X 0	0 0 1	0 0 0
0 0 0	X 1	1 0 1	0 0 0
0 0 1	X X	0 1 0	1 1 1
0 1 0	X X	0 1 1	1 1 1
0 1 1	0 X	0 0 0	1 1 1
0 1 1	1 X	0 1 1	1 1 1
1 0 0	X X	0 X X	X X X
1 0 1	X X	1 1 0	1 0 0
1 1 0	X X	0 1 1	1 1 0
1 1 1	X X	0 X X	X X X

Seuraavaksi muodostetaan ASM-kaaviosta tilataulukko, joka on esitetty vasemmalla. Tilataulukon muodostettiin seuraavasti:

Lähdetään ASM-kaavion ensimmäisestä tilasta (tilakoodi = 000), jonka tilakoodiin tulee tilataulukon nykytilaksi (Q2, Q1 ja Q0).

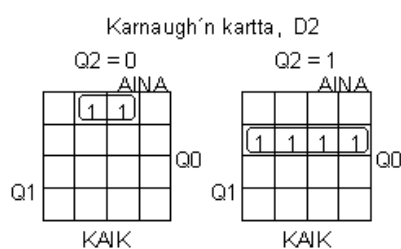
Sitten merkataan uudeksi tilaksi (D2, D1 ja D0) sen tilan tilakoodi johon siirrytään (tilakoodi 001), sekä lisäksiannot, jotka vaikuttavat mihin tilaan siirrytään (AINA = X, KAIK=0).

Mikäli otto-signaalilla ei ole merkitystä tilasiirtymään, merkataan sen kohdalle X. (Näin saadaan tilataulukko lyhyemmäksi ja saadaan sievennysvaiheessa mahdollisesti sievempiä lausekkeita --> kuluu vähemmän portteja.)

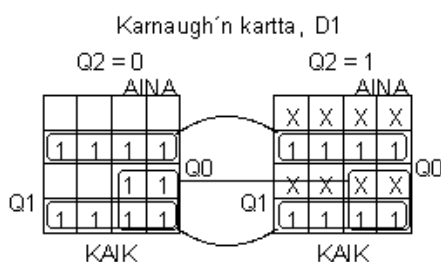
Tässä tapauksessa jää kaksi tilaa jää käyttämättömiksi (100, 111), sillä 3:lla tilabitillä saatiin 8 tilaa ja niistä käytetään vain 6. Nämä tilat otetaan kuitenkin mukaan tilataulukkoon, jotta tilataulukko kuvaisi yksiselitteisesti piirin toiminnan. Lisäksi laitteen täytyy pystyä poistumaan käyttämättömästä tilasta jos se syystä tai toisesta siihen joutuu. Uusi tila määritellään siis siten, että käyttämättömistä tiloista päästään pois ja palataan johonkin käytössä olevaan tilaan. Tässä tapauksessa siirto on hoidettu s.e. palataan johonkin tilaan, jonka ensimmäinen bitti on nolla. Muuten ehtoja ei ole asetettu, jotta toiminta ei mutkistaisi piirin toteutusta. Tilataulunannot, ovat antosignaali, jotka liittyvät **nykytilaan**. (Ne saadaan siis ASM-kaavion neliöiden sisältä)

Kiikkujen otto-signaalit ja antojen lausekkeet

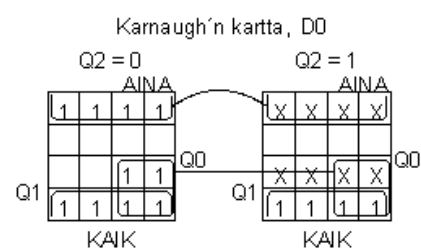
Määritellään Karnaugh'n kartan avulla tilataulukosta lausekkeet D-kiikkujen ottoille (jotka ovat siis samat kuin D2, D1 ja D0). Tässä tarvitaan kolme kappaletta 5:en muuttujan Karnaugh'n karttoja, sillä D-ottoja on 3 (D2, D1 ja D0) ja muuttujia on 5 (Q2, Q1, Q0, AINA ja KAIK):



$$D2 = KAIKQ0'Q1'Q2' + Q0Q1'Q2$$

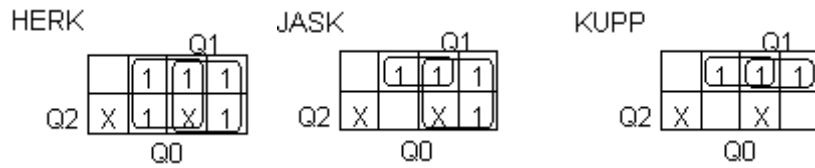


$$D1 = Q0Q1' + Q0'Q1 + AINAQ1$$



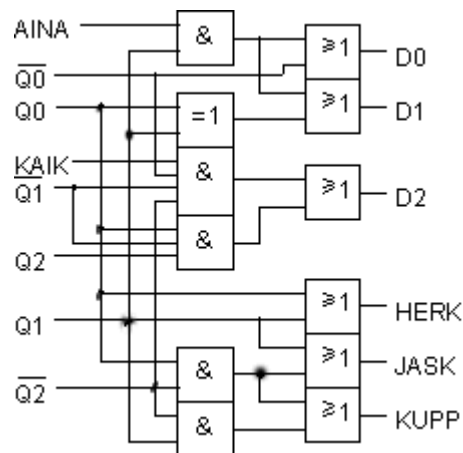
$$D0 = Q0' + AINAQ1$$

Lausekkeet annoille:



$$\text{HERK} = Q_0 + Q_1 \quad \text{JASK} = Q_0Q_2' + Q_1 \quad \text{KUPP} = Q_0Q_2' + Q_1Q_2'$$

Kombinaatiopiiri



Mealyn kone -toteutus monimutkaiselle valomainokselle

Kun valomainos toteutetaan Mealyn koneella, tarvitaan tiloja vain neljä (ks. ASM-kaavio). Tämä johtuu siitä, että antamalla ottosignaalin KAİK vaikuttaa suoraan antosignaaleihin voidaan haluttu toiminta saada aikaan yhdellä ja samalla tilasekvenssillä (tilakierrolla). Antosignaalia, johon ottosignaali vaikuttaa suoraan sanotaan ehdolliseksi antosignaaliksi. (Kuvataan kulmistaan pyöristetyllä suorakaiteella.)

ASM-kaaviota tulkitaan nyt seuraavasti:

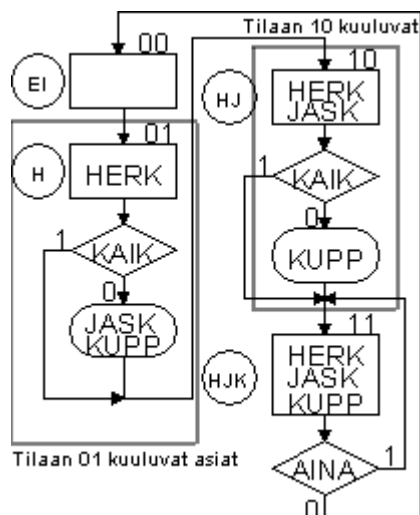
Tila 00 - mitään ei näy taululla

Tila 01 - Taululla näkyy HERK. **Samalla** katsotaan ottosignaalin KAİK arvo.

Mikäli tämä on 0 lukee taululla **myös** JASK KUPP. Muutoin taululla ei lue kuin HERK. ASM-kaavioon on nyt esimerkinomaisesti piirretty neliö, joka näyttää, mitkä kaikki asiat kuuluvat samaan tilaan (eli tapahtuvat samalla ajanhetkellä).

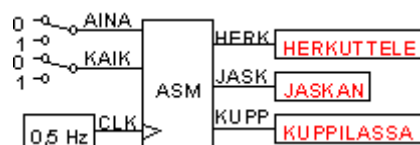
Tila 10 - Taululla lukee HERK JASK. **Samalla** katsotaan ottosignaalin KAİK arvo. Mikäli tämä on 0 lukee taululla **myös** KUPP. Muutoin taululla lukee vain HERK JASK.

Tila 11 - Taululla lukee HERK JASK KUPP.



Siis H ja HJ tiloissa ottosignaali KAIK vaikuttaa siihen, ovatko muutkin annot kuin tilassa aina aktiiviset annot päällä vai poissa.

Koska tiloja on vain neljä, pärjätään kahdella kiikulla ($2^2=4$)



Tilataulukko

Nykytila		Otot		Uusi tila		Annot		
						HERK	JASK	KUPP
Q1	Q0	AINA	KAIK	D1	D0			
0	0	X	X	0	1	0	0	0
0	1	X	0	1	0	1	1	1
0	1	X	1	1	0	1	0	0
1	0	X	0	1	1	1	1	1
1	0	X	1	1	1	1	1	0
1	1	0	X	0	0	1	1	1
1	1	1	X	1	1	1	1	1

Kiikkujen ottosignaalit ja antojen lausekkeet totuustaulun perusteella:

D1

			AINA	
1	1	1	1	1
			1	1
1	1	1	1	1

Q1

KAIK

Q0

$$D1 = Q0Q1' + Q0'Q1 + Q1AINA$$

D0

			AINA	
1	1	1	1	1
			1	1
1	1	1	1	1

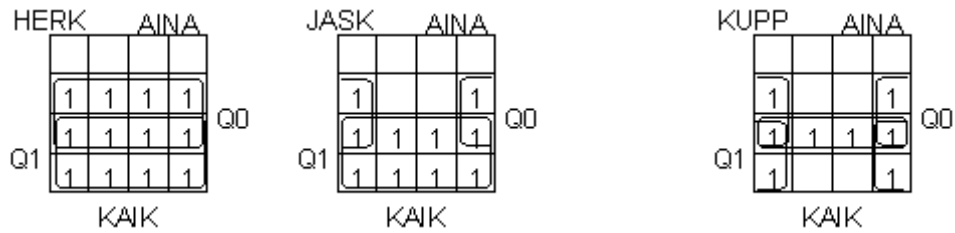
Q1

KAIK

Q0

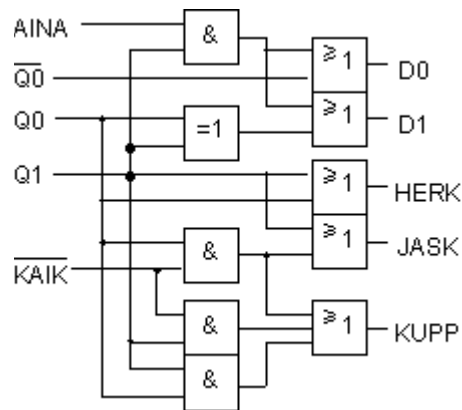
$$D0 = Q0' + Q1AINA$$

Lausekkeet annoille:



$$\text{HERK} = Q0 + Q1 \quad \text{JASK} = \text{KAIAK}'Q0 + Q1 \quad \text{KUPP} = \text{KAIAK}'Q0 + \text{KAIAK}'Q1 + Q0Q1$$

Kombinaatiopiiri



Erot toteutusten välillä

Toteutus	Kiikkuja	Portteja	Porttien ottoja
Moore	3 - D	12	27
Mealy	2 - D	10	21
Moore JK	2 - JK	8	17

(JK -kiikkutoteutusta käsitellään vasta seuraavalla sivulla, mutta nähdään, että se säästää huomattavasti piirin porttien määrässä D -kiikkutoteutukseen verrattuna.) Samoilla kiikuilla toteutettujen Mooren ja Mealyn toteutuksen välillä nähdään, että Mealy on taloudellisempi. Mooren kone on kuitenkin luotettavampi ottoosignaali virheiden suhteen. Valinta kahden tavan välillä pitää tehdä tapauskohtaisesti.

JK-kiikkujen käyttö sykkronisten sekvenssiipiirien suunnittelussa

D-kiikussa D-ottoon tuleva signaali määrää suoraan seuraavan tilan (ks. D-kiikun tilataulu aiempaa "kiikku yhteenveto"-kohdasta). Näin ollen tilataulukon sarakke 'Uusi tila' riittää sellaisenaan määrittelemään tilakiikkujen D-ottojen lausekkeet. JK-kiikkuja käytettäessä tilanne on mutkikkaampi. Jokaisen JK-kiikun J- ja K- otoille on muodostettava oma lauseke (eli nyt kussakin kiikussa on kaksi eri ottoa, joiden kombinaatio vaikuttaa uuteen tilaan). Tehtävän helpottamiseksi kannattaa laatia JK-kiikun tilataulusta toinen versio eli muutostaulu. Tämä kuvaa sen, miten missäkin halutussa tilanmuutoksessa ottojen arvot on valittava, jotta haluttu muutos saadaan aikaan. JK -kiikun muutostaulu esitetään alla. (Vertailun vuoksi oikealla D-kiikun muutostaulu.)

Muutostaulua siis luetaan seuraavasti: Tiedetään nykytila $Q(t)$ ja uusi tila $Q(t+1)$. Millä J:n ja K:n arvoilla saadaan muutos tästä tiedetystä nykytilasta uuteen tilaan aikaiseksi??

Tilataulu			Muutostaulu			
J	K	$Q(t+1)$	$Q(t)$	$Q(t+1)$	J	K
0	0	$Q(t)$	0	0	0	X
0	1	0	0	1	1	X
1	0	1	1	0	X	1
1	1	$\overline{Q(t)}$	1	1	X	0

Tilataulu		Muutostaulu		
D	$Q(t+1)$	$Q(t)$	$Q(t+1)$	D
0	0	0	0	0
1	1	0	1	1
		1	0	0
		1	1	1

Muutostaulusta nähdään suoraan, mitkä arvot J- ja K- otoille on annettava mitään tilanmuutosta haluttaessa. Muutostaulu havainnollistaa myös sen, miksi usein JK-kiikkutoteutus johtaa yksinkertaisempaan lopputulokseen kuin D-kiikkutoteutus: Kutakin tilanmuutosta varten on vain joko J- tai K- otton saatava tietty arvo. Toinen otto voi saada kumman arvons tahansa. Tämä näkyy muutostaulussa X:nä.

Seuraavaksi toteutetaan monimutkaisen valomainoksen Mealyn kone -versio JK -kiikkutoteutuksena. **JK-kiikku - toteutus ei vaikuta mitenkään määrittelyyn, lohkokaaevioon tai tilataulukkoon. Se tulee esille ainoastaan, kun määritellään kiikkujen ottojen lausekkeet ja kun piirretään piirikaavio.**

Perustilataulukko on sama kuin aikaisemmin esitellyssä Mealy - toteutuksessa. JK -toteutuksessa tarvitaan lisäksi sarakkeet kiikkujen J- ja K- otoille. Alla esitetty sarakkeiden johtaminen muutostaulun avulla:

Nykytila		Otto		Uusi tila		JK-kiikkujen otto			
Q1	Q0	AINA	KAIK	Q1	Q0	J1	K1	J0	K0
0	0	X	X	0	1	0	X	1	X
0	1	X	0	1	0	1	X	X	1
0	1	X	1	1	0	1	X	X	1
1	0	X	0	1	1	X	0	1	X
1	0	X	1	1	1	X	0	1	X
1	1	0	X	0	0	X	1	X	1
1	1	1	X	1	1	X	0	X	0

$Q(t)$	$Q(t+1)$	J	K
0	0	0	X
0	1	1	X
1	0	X	1
1	1	X	0

Tarkastellaan vaikkapa tilataulukon ylintä riviä. Siinä tila muuttuu 00:sta 01:ksi. Q1 ei muutu, vaan pysyy nollana. Tämä saadaan aikaan muutostaulun mukaisesti asettamalla $J1=0$ ja $K1=X$. Q0 muuttuu nolasta ykköseksi. Tämä taas edellyttää muutostaulun mukaan, että $J=1$ ja $K=X$.

Tarkastellaan vielä harjoituksen vuoksi tilataulukon toinen rivi, jossa tapahtuu tilanmuutos tilasta 01 tilaan 10.

Ensin Q1 muuttuu 0 --> 1. Tämä muutos saadaan JK-kiikulla tehtyä kun $J=1$ ja $K=X$

Lisäksi Q0 muuttuu 1 --> 0. Tämä muutos puolestaan saadaan tehtyä kun $J=X$ ja $K=0$

Tilataulukko käytetään joka rivin osalta vastaavasti.

JK-kiikut sekvenssipiireissä jatkuu..

JK -kiikkujen lausekkeet saadaan edellisen sivun tilataulukosta normaalisti Karnaugh'n karttojen avulla:

Nykytila		Otot		JK-kiikkujen otot			
Q1	Q0	AINA	KAIK	J1	K1	J0	K0
0	0	X	X	0	X	1	X
0	1	X	0	1	X	X	1
0	1	X	1	1	X	X	1
1	0	X	0	X	0	1	X
1	0	X	1	X	0	1	X
1	1	0	X	X	1	X	1
1	1	1	X	X	0	X	0

J1

1	1	1	1
X	X	X	X
X	X	X	X

Q0

KAIK

$$J1 = Q0$$

K1

X	X	X	X
X	X	X	X
1	1		

Q0

KAIK

$$K1 = Q0AINA'$$

K0

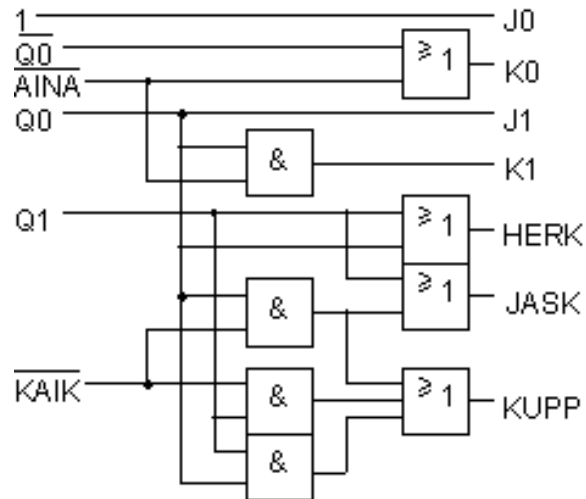
X	X	X	X
1	1	1	1
1	1		
X	X	X	X

Q0

KAIK

$$K0 = Q1' + AINA'$$

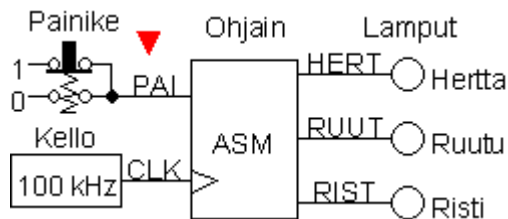
Sekä $J0 = 1$. Nähdään jo tässä, että JK -kiikkutoteutus on yksinkertaisempi kuin vastaava D-kiikuin toteutettu. Antojen lausekkeet ovat samat kuin D-kiikuilla. Lopullinen kombinaatiopiiriosan kaavio näyttää tältä:



Maija-pelin valtinavontakone

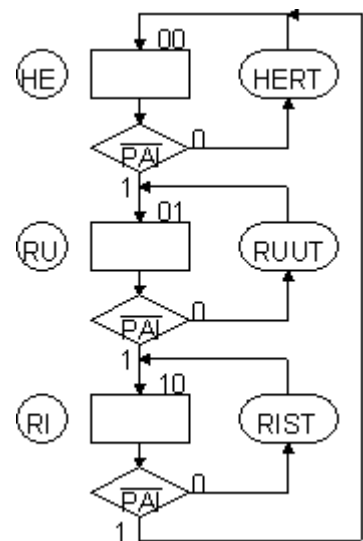
Maija-pelissä pata ei voi olla valttia, mutta muuten valtti valitaan pelikohtaisesti. Tässä esimerkissä suunnitellaan valtinavontakone, joka antaa satunnaisesti valttiksi joko hertan, ruudun tai ristin. Arvontakone toteutetaan tilakoneella, jossa on kolme tilaa, yksi kutakin valttikandidaattimaata (hertta, ruutu, risti) kohden. Sekvenssiipiirin kelloaajuudeksi valitaan ihmisen reagoitokykyyn nähden suuri arvo, esimerkiksi 100 kHz. Piiri kiertää kolmessa eri tilassaan 10 mikrosekunnin välein. Piiriin liitetään painike, jonka avulla tilakierto saadaan pysähtymään siihen tilaan, jossa piiri painamishetkellä oli. Tämä ilmaisee, mikä maa on valttia. Lisäksi määritellään, että tilaa ilmaiseva lamppu palaa vain nappia painettaessa.

ASM -ja lohkokaavio



Esimerkin omaisesti tehdään painikesignaalista (PAI) nollassa aktiivinen: haluttu toiminto tapahtuu silloin, kun PAI saa arvon nolla (ks. lohkokaaviota yllä, painike (PAI)=0 kun sitä painetaan). Valtinavontakone voidaan toteuttaa kolmitilaisena, kun se tehdään Mealyn koneena. Mooren kone -toteutuskin voitaisiin tehdä kolmitilaisena. Tällöin kuitenkin lamput vilkkuisivat (ainakin teoriassa) koko kierron ajan, mikä ei juurikaan olisi järkevää. Toteutetaan valtinavontakone erikseen sekä D- että JK -kiikuilla.

Koska koneeseen tulee kolme tilaa, tarvitaan kaksi kiikkua ja yksi tila jää käyttämättömäksi. (Kahdella kiikulla saadaan $2^2=4$ tilaa)



D-kiikkutoteutus

Kiikkujen otot:

Nykytila		Otot	Uusi tila		Annot		
Q1	Q0	PAI	D1	D0	HERT	RUUT	RIST
0	0	0	0	0	1	0	0
0	0	1	0	1	0	0	0
0	1	0	0	1	0	1	0
0	1	1	1	0	0	0	0
1	0	0	1	0	0	0	1
1	0	1	0	0	0	0	0
1	1	0	0	X	X	X	X
1	1	1	0	X	X	X	X

D1		Q0
Q1	PAI	
0	0	1
0	1	0
1	0	0
1	1	0

$$D1 = PAI'Q0Q1' + PAIQ0'Q1$$

D0		Q0
Q1	PAI	
0	0	1
0	1	1
1	0	X
1	1	X

$$D0 = PAI'Q0'Q1' + PAIQ0$$

Antojen lausekkeet:

HERT		Q0
Q1	PAI	
0	0	1
0	1	X
1	0	X
1	1	X

RUUT		Q0
Q1	PAI	
0	0	1
0	1	X
1	0	X
1	1	X

RIST		Q0
Q1	PAI	
0	0	1
0	1	X
1	0	X
1	1	X

$$HERT = PAIQ0'Q1' \quad RUUT = PAIQ0 \quad RIST = PAIQ1$$

Maija-pelin valtinarvontakoneen JK-kiikkutoteutus

Tilataulu

Kiikkujen otot:

Nykytila		Otot	Uusi tila		JK-kiikkujen otot			
Q1	Q0	PAI	Q1	Q0	J1	K1	J0	K0
0	0	0	0	0	0	X	0	X
0	0	1	0	1	0	X	1	X
0	1	0	0	1	0	X	X	0
0	1	1	1	0	1	X	X	1
1	0	0	1	0	X	0	0	X
1	0	1	0	0	X	1	0	X
1	1	0	0	X	X	1	X	X
1	1	1	0	X	X	1	X	X

J1		Q0
Q1	PAI	
0	0	1
0	1	X
1	0	X
1	1	X

$$J1 = PAI'Q0$$

J0		Q0
Q1	PAI	
0	0	1
0	1	X
1	0	X
1	1	X

$$J0 = PAI'Q1'$$

K1		Q0
Q1	PAI	
0	0	X
0	1	X
1	0	X
1	1	X

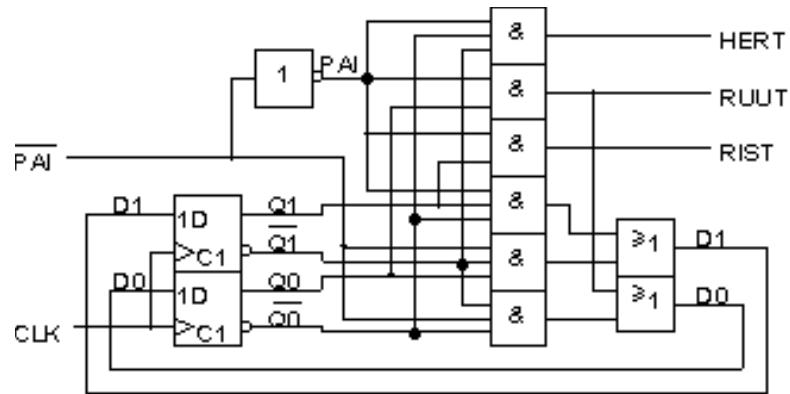
$$K1 = PAI' + Q0$$

K0		Q0
Q1	PAI	
0	0	X
0	1	X
1	0	X
1	1	X

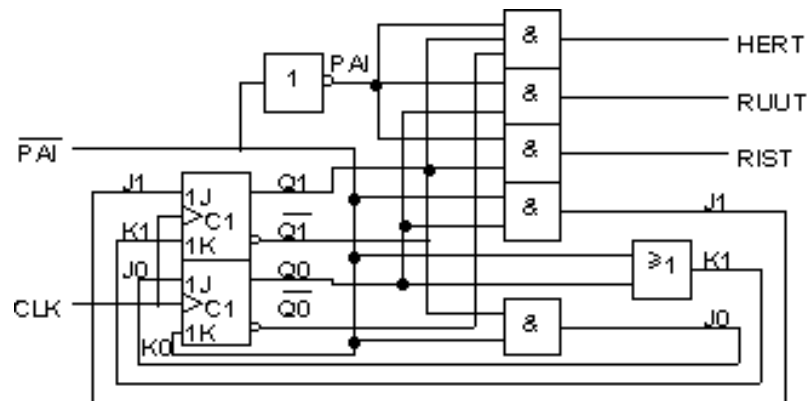
$$K0 = PAI'$$

JK-kiikkutoteutuksen antojen lausekkeet ovat samat kuin D-kiikkutoteutuksellakin. Seuraavalla sivulla esitetään molempien toteutuksien piirikaaviot. JK -kiikkutoteutuksen on hieman yksinkertaisempi:

D-kiikkutoteutuksen piirikaavio



JK-kiikkutoteutuksen piirikaavio



Signaalinkäsittelytekniikan laboratorio

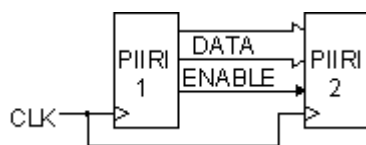
Digitaalitekniikan perusteet - luento 11

Tiedon siirto

Tiedolla on kaksi siirtotapaa: rinnakkais- ja sarjamuotoinen siirto. Rinnakkaismuotoisessa bitit siirretään kukin omaa linjaansa pitkin. Sarjamuotoisessa bitit siirretään peräkkäin samaa väylää pitkin.

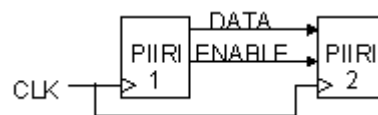
Rinnakkaissiirto:

- nopea
- paljon johtimia
- yksinkertainen
- samalla matkalla kalliimpi toteutus kuin sarjamuotoisella siirrolla

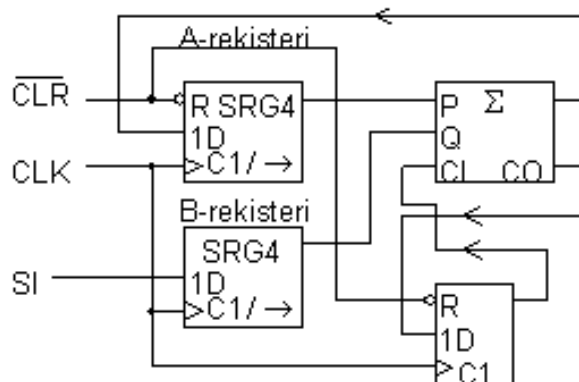


Sarjasiirto:

- hidas
- vähän johtimia
- mutkikkaampi logiikka
- samalla matkalla halvempi toteutus kuin rinnakkaismuotoisella siirrolla



Esim. sarjasummain



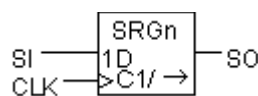
Siirtorekisterit

Siirtorekisteri sisältää joukon kiikkuja. Koska jokainen kiikku pystyy säilömään yhden bitin verran informaatiota, sisältää n-bittinen rekisteri n kiikkua. n-bittinen rekisteri pystyy siis luonnollisesti säilömään n bittiä binääridataa. Rekisterissä on joukko kiikkuja, mahdollisesti lisättynä kombinaatiopiiriporteilla. Rekisteri suorittaa datan prosessointia.

Käytännössä siirtorekistereitä on neljää tyyppiä. Tyypit luokitellaan datan vastaanoton ja lähetysmuodon mukaan neljään eri ryhmään: SISO (serial in - serial out), SIPO (serial in - parallel out), PISO (parallel in - serial out) ja PIPO (parallel in - parallel out). Alla esitellään eri tyyppien ominaisuudet ja piirrosmerkit lyhyesti. Piirrosmerkeissä olevat numerot ilmaisevat alisteisuutta (ks. aiempaa "Riippuvuusmerkinnöistä")

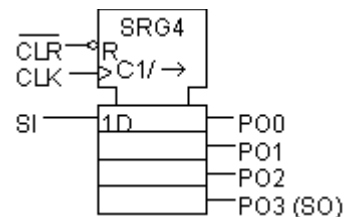
SISO (perustyyppi):

Koostuu D-kiikusta, anto kytketty seuraavaan D-ottoon. Koko 8:sta bitistä tuhansiin. Huomioitavaa: viive.



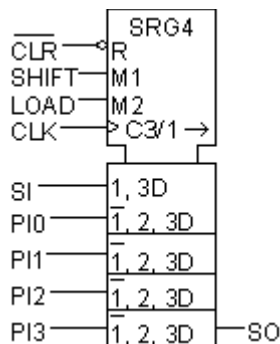
SIPO:

4, 8 tai 16 bittinen rekisteri. Rekisterissä tehdään sarja-rinnakkaismuunnos.



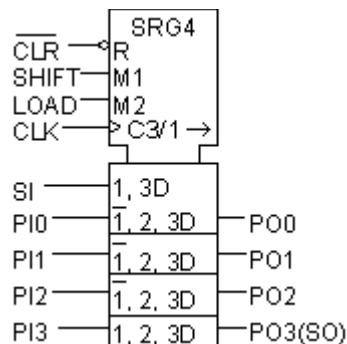
PISO:

4, 8 tai 16 bittiä. Rekisterin sisällä tapahtuu rinnakkais-sarjamuunnos.



PIPO:

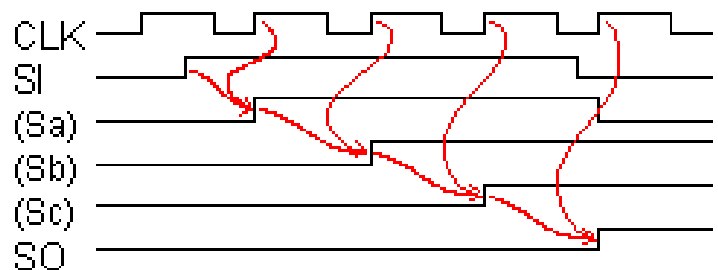
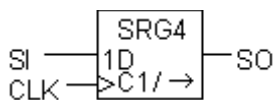
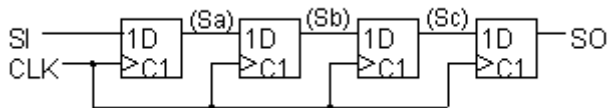
4, 8 tai 16 bittinen rekisteri. Monipuolista datan käsittelyä.



Esimerkkejä siirtokoneista

Neljän kiikun SISO siirtorekisteri

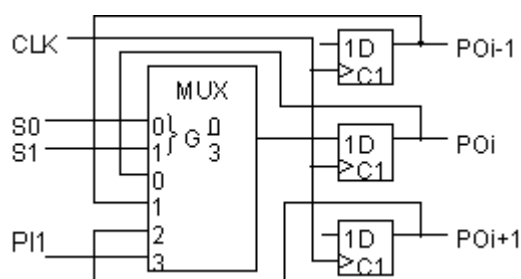
SISO eli Serial in - Serial out -siirtorekisteri voidaan muodostaa yksinkertaisesti laittamalla joukko D-kiikkuja peräkkäin. Alla esitellään neljäkiikkuisen SISO -siirtorekisterin piirikaavio, piirrosmerkki ja aikakaavio: (Piirikaavioon on selvyuden vuoksi merkitty myös 'välivaiheet' (Sa--Sc), jotka näkyvät aikakaaviossakin)



Kaksisuuntainen siirtorekisteri

Rekisterin siirtosuunta voidaan joskus valita: joko siirto eteenpäin eli alas tai siirto taaksepäin eli ylös. Usein käytännön piireissä on siirtosuunnan valinnan lisäksi myös rinnakkaislataus tai pysäytys. Alla kaksisuuntaisen rekisterin toimintataulukko sekä osatoteutus ottovalitsimella:

S1	S0	Toiminta
0	0	Ei muutosta
0	1	Siirto eteen
1	0	Siirto taakse
1	1	Rinnakkaislataus



Laskurit

Laskuri on tilakone, jonka jokainen tila toistuu määrävälein (= laskurin laskentajakso). Binäärilaskurissa laskentajakso on jokin kakkosen potenssi. n-bittisessä binäärilaskurissa on n kappaletta kiikkuja ja sen laskentajakso on 2^n . Laskentasekvenssi on yleensä peräkkäisiä binäärilukuja: 0000, 0001, 0010, 0011, ..., 1110, 1111, 0000, 0001..

Dekadilaskureissa laskentajakso on kymmenen potenssi. Dekadilaskurissa on neljä kiikkua / dekadi. Laskentajaksoja 10 ja 100.

Laskurit voidaan jakaa asynkronisiin ja synkronisiin laskureihin. Alla selvitetään näiden ominaisuuksia ja eroja.

Asynkroniset laskurit

- laskureissa seuraava kiikku saa kellopulssinsa (CLK) edellisen kiikun ulostulosta (Q) (taaksepäin laskevassa laskurissa edellisen kiikun invertoidusta annosta (Q'))
- ratkaisu on erittäin yksinkertainen
- antojen välillä havaittavissa selvä viive-ero
- pitkissä laskureissa suurin kellotaajuus alenee
- asynkronisten laskurien käyttöä tulee joko välttää suunnittelussa tai heikkoudet täytyy ottaa huomioon

Synkroniset laskurit

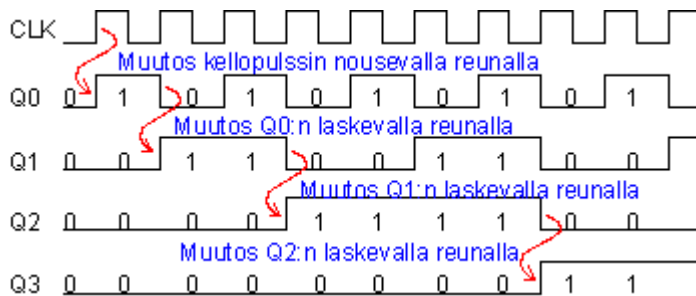
- yksi ja sama kellopulssi tulee kaikkien kiikkujen kello-ottoihin (CLK)
- Toteutus JK- tai D-kiikuilla
- Käytännön laskureissa yleensä myös nollausotto (RESET)
- Laskurin ketjutusta varten laskurin sallintaotot (ENABLE) ja laskuri täynnä -otto

4-bittinen asynkroninen binäärilaskuri

Tutkitaan tässä nelibittistä (asynkronista) binäärilaskuria:

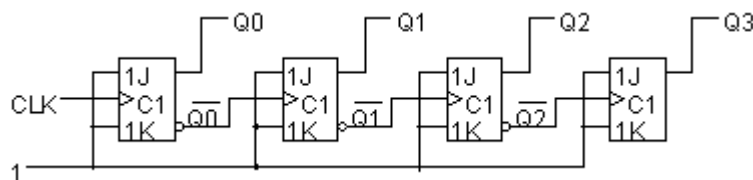
Annot				Eteenpäin laskeva	Taaksepäin laskeva
Q3	Q2	Q1	Q0		
0	0	0	0	↓	↑
0	0	0	1	↓	↑
0	0	1	0	↓	↑
0	0	1	1	↓	↑
		⋮		⋮	⋮
1	1	0	0	↓	↑
1	1	0	1	↓	↑
1	1	1	0	↓	↑
1	1	1	1	↓	↑
0	0	0	0	↓	↑
		⋮		⋮	⋮

4-bittisen laskurin laskentasekvenssi. Taulua luetaan alaspäin, jos käsitellään eteenpäin laskevaa binäärilaskuria ja ylöspäin, jos käsitellään taaksepäin laskevaa binäärilaskuria.

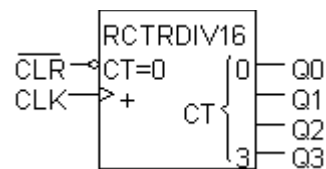


Laskuri voidaan muodostaa esim. JK-kiikuista käyttämällä aina kiikun kellona (CLK) edellisen laskurin invertoitua antoa (Q'), sekä kytkemällä otot J ja K 1:een (eli $J=1$ ja $K=1$). Tämähän tarkoittaa sitä, että aina kun JK-kiikku tarkistaa tilansa sen uusi tila on edellisen komplementti ks. JK-kiikun totuustaulu aiemmin kohdasta "kiikkuyhteenveto").

Ajoituskaaviossa muut kiikut (paitsi ensimmäinen) toimivat aina esim. Q0:n laskevalla reunalla, Q1:n laskevalla reunalla jne. koska niiden ottama kellopulssi (CLK) on peräisin edellisen kiikun **invertoivasta** lähdöstä ja mikäli Q0 laskee, Q0' nousee ja muutos tapahtuu tällöin Q0:n laskevalla reunalla!



Rakenne on erittäin yksinkertainen ja bittimäärää voidaan lisätä helposti laittamalla lisää kiikkuja jonon jatkoksi.



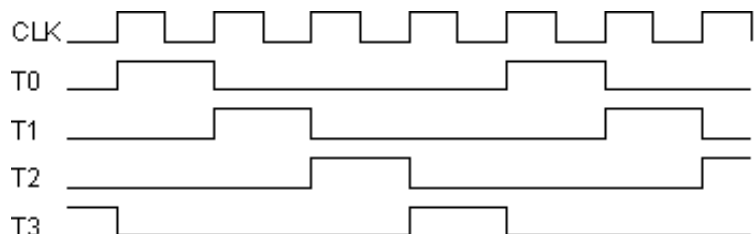
4- bittisen asynkronisen binäärilaskurin piirrosmerkki

Ajoituspulssien synnyttäminen

Jotta saadaan ajallisesti toistuvia tapahtumia, tarvitaan ajoituspulssi. Tämä voidaan järjestää eri tavoin. Tässä katsotaan esimerkinomaisesti niistä kaksi.

Tapa 1 Muodostetaan neljän aloitussignaalin ketju laskurilla ja dekooderilla

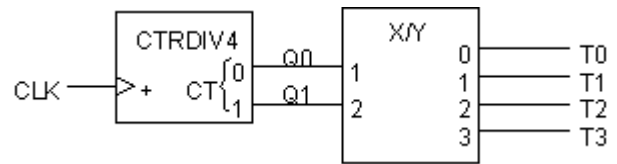
Tapa 2 Muodostetaan ajoituspulssi SIPO siirtorekisterillä ja portilla.



Haluttu aikakaavio on esitetty oikealla:

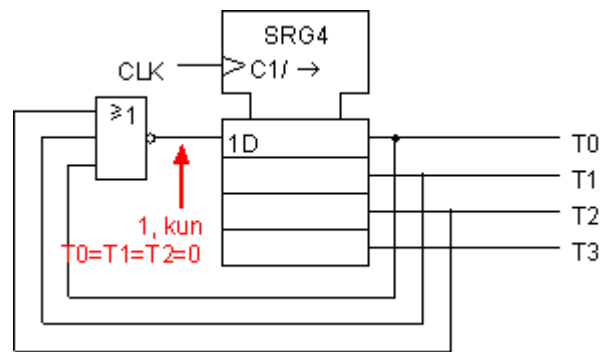
Ajoituspulssin teko laskurilla ja dekooderilla. Tämä on 'vaarallinen tapa', koska olemassa on virhepulssimahdollisuus.

Eli tässä laskuri laskee sekvenssiä 00 -> 01 -> 10 -> 11 ja dekooderista tulee ottosignaalikombinaation mukaan ulostulo T0 -- T3



Ajoituspulssi siirtorekisterillä ja portilla. Suositeltava tapa.

Eli tässä NOR-portti tuottaa siirtorekisterin ottoon ykkösen kun T0=0, T1=0 ja T2=0. Ykkönen otetaan siirtorekisteriin sisään kellopulssin (CLK) tahdittamana. Tämän jälkeen juuri sisäänotettu 1:nen etenee (laskeutuu) siirtorekisterissä eteenpäin kellopulssin (CLK) tahdissa. Piirin antosignaaleista (T0 - T3) on ykkösenä aina yksi sen mukaan missä kohtaa rekisteriä "etenevä ykkönen" sijaitsee.



Signaalinkäsittelytekniikan laboratorio

Digitaalitekniikan perusteet - luento 12

Muistit

Muisti on olennainen osa digitaalista tietokonetta ja on tärkeässä asemassa kaikissa digitaalisysteemeissä. Keskeiset muistityypit ovat vaihtomuisti (esim. RAM, RWM) ja kiintomuisti (ROM). Vaihtomuistissa pidetään suoritusten aikaisia välituloksia ja lopputuloksia. Vaihtomuistista tieto häviää, kun syöttöjännite katkaistaan. Vaihtomuistiin voidaan kirjoittaa ja siitä voidaan lukea. Kiintomuistin muistiaines on pysyvää, eikä asiakas pääse muuttelevaan sen sisältöä. Kiintomuistissa pidetään ohjelmatietoja. Kiintomuistin sisältö ohjelmoidaan muistiin ja ohjelma säilyy, vaikka syöttöjännite katkaistaan. ROMmin käyttö on yksi tapa tehdä ohjelmoitavaa logiikkaa (PLD - Programmable Logic Device).

Vaihtomuisti

Vaihtomuisti koostuu muistisoluista ja solujen käsittelyyn tarvittavasta logiikasta. Jokainen muistisolu tallettaa yhden bitin verran tietoa. Solut on ryhmitelty kokonaisuuksiksi, joita nimitetään muistipaikoiksi ja sanoiksi. Yksi muisti sisältää useita tällaisia muistipaikkoja (eli sanoja). Sanan pituus on yleensä $n \times 8$ bittiä. 8 bitin ryhmää nimitetään tavuksi (byte) ja sen puolikasta puolitavuksi (nibble). Jokaisella sanalla on osoite, jonka avulla se voidaan osoittaa muistista. Muistipaikan sisältämää tietosisältöä puolestaan käsitellään dataväylien kautta, **jokaiselle bitille on oma väylänsä**. Tieto kulkee siis rinnakkaismuotoisesti muistiin ja muistista ulos. Osoitetietoja varten on osoiteväylät.

Vaihtomuistin luku antaa ulos muistipaikan sisällön muuttamatta sitä. Kirjoitus asettaa muistipaikan sisällön halutuksi.

Vaihtomuistin lukeminen:

- asetetaan halutun muistipaikan osoite osoiteottoihin (ADDRESS)
- asetetaan piirin valintasignaali eli niin sanottu piirin enable -signaali (EN) aktiiviseksi, jolloin kyseinen piiri on käytössä. (Piirin valintasignaali määrää, onko piiri toiminnassa vai ei.)
- asetetaan lukusignaali aktiiviseksi (READ, erotuksena kirjoitussignaalista WRITE), koska nyt halutaan lukea dataa.
- muistin sisältö saadaan data-antoihin (DATA I/O)

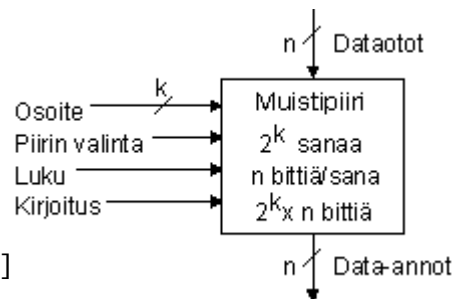
Vaihtomuistin kirjoitus:

- annetaan halutun muistipaikan osoite osoiteottoihin (ADDRESS)
- asetetaan kirjoitettava sana dataottoihin (DATA I/O)
- asetetaan piirin valintasignaali (EN) aktiiviseksi, eli valitaan tietty muistipiiri
- asetetaan piirin kirjoitussignaali aktiiviseksi (WRITE erotuksena lukuoperaatioon tarvittavasta READ signaalista)
- lopuksi piiri muuttaa muistipaikan sisällön

Vaihtomuistin rakenne

Vaihtomuistissa on seuraavia liitäntöjä:

- osoiteotot [ADDRESS 0:k] (k ottoa, joilla saadaan eroteltua 2^k eri muistipaikan osoitetta)
- piirin valintaotto [EN] eli enable (Joissain tapauksissa valintaottoja voi olla useampia. Perusmerkitys on kuitenkin vain joko sallia tai estää piirin toiminta.)
- luvun tai kirjoituksen valintataotot [READ/WRITE]
- dataotot ja -annot (n ottoa, n antoa) [DATA I/O 0:n] (Yleensä käytetään samoja datalinjoja sekä otto- , että antosignaaleille. Ainoastaan READ/WRITE-signaali määrää sen kumpaan suuntaan datalinjoja tieto kulkee)



Muistin koon mittayksiköissä käytetään tavallisia etuliitteitä. **HUOM ! Tässä tapauksessa, (kuten usein muulloinkin tietokonetekniikassa), etuliitteet viittaavat kuitenkin kakkosen potensseihin, eivätkä suoraan kymmenen kertoimiin (kuten on ehkä totuttu):**

- k - kilo - 2^{10} - 1024
- M - mega - 2^{20} - 1 048 576
- G - giga - 2^{30} - 1 073 741 824
- T - tera - 2^{40} - 1 099 511 627 776

Muistin koko voidaan ilmoittaa eri tavoin. Havainnollista on kertoa, montako sanaa muistissa on ja miten pitkiä yksittäiset sanat ovat. Esimerkiksi muistin koko ilmoitetaan usein lausekkeella $s \times n$, mikä tarkoittaa, että muistissa on s kappaletta sanoja, ja että jokaisen yksittäisen sanan pituus on n bittiä. Toisaalta muistin koko voidaan ilmoittaa pelkästään bitteinä tai tavuina.

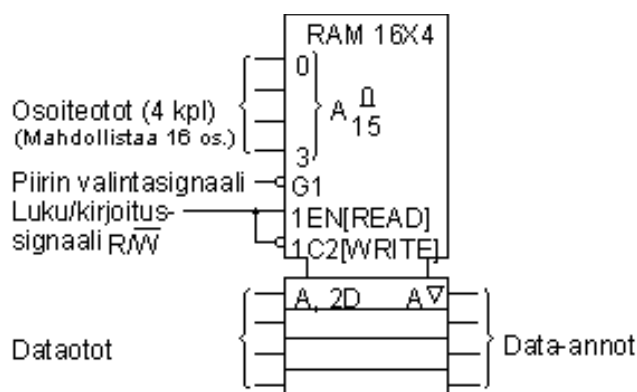
Esim. Jos muistin koko on $1k \times 8$ on siinä $2^{10} \times 8 = 8192$ bittiä

Muistin sanojen lukumäärä määrää samalla muistin osoiteottojen [ADDRESS] määrän ja osoiteavaruuden. s sanaa $\rightarrow \log_2 s$ osoiteottoa ylöspäin pyöristettynä. Edelleen, jos käytössä on s sanaa, on osoiteavaruus $0 \dots s-1$.

Esim 1. Muistissa olevien sanojen määrä on $1k = 1024$. Tällöin tarvitaan siis $\log_2 1024 = 10$ osoitelinjaa.

Esim 2. Jos muistissa olevien sanojen määrä on 25. Tällöin tarvitaan $\log_2 25 = 4,6439 \rightarrow$ Tarvitaan 5 osoitelinjaa ($2^5 = 32$ osoitetta)

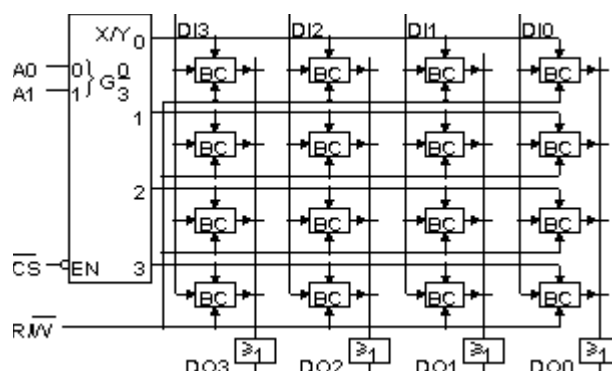
Vaihtomuistin piirrosmerkki



Jos piiri on iso, voidaan dataotot ja -annot yhdistää.

Vaihtomuisti koostuu seuraavista osista:

- muistisolut (kuvassa BC)
- dekooderi (ohjaa muistien osoitusta, kuvassa vasemmalla)
- ohjauslogiikka (R/W')
- ottojen / antojen muodostuslogiikka



Prosessori tai kontrolleri ohjaa vaihtomuistin toimintaa. Se siis lukee tai kirjoittaa muistiin. Vaihtomuisti toimii prosessorin kellon tahdissa. Prosessori antaa osoitteen ja ohjaussignaalit, muisti antaa tai ottaa dataa.

Esimerkki: 4 x 4 RAM

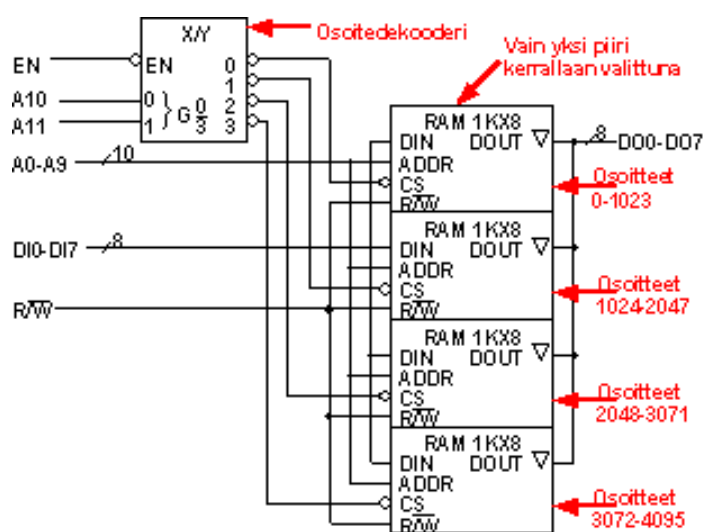
Vaihtomuistin laajentaminen

Vaihtomuistia laajentaessa voidaan laajentaa kahta eri ominaisuutta: voidaan joko kasvattaa muistiin talletettavien sanojen määrää (osoiteavaruuden laajentaminen) tai pidentää yksittäisen sanan pituutta (sananpituuden laajentaminen). Osoiteavaruuden laajentamista tarvitaan tilanteessa, jossa monta pientä muistia yhdistetään yhdeksi suureksi: tällöin osoiteottojen määrä pienissä ei riitä. Osoiteavaruus voidaan tuplata invertterillä. Jos halutaan suurempaa laajennusta, joudutaan käyttämään dekooderia.

Osoiteavaruuden laajentaminen dekooderilla

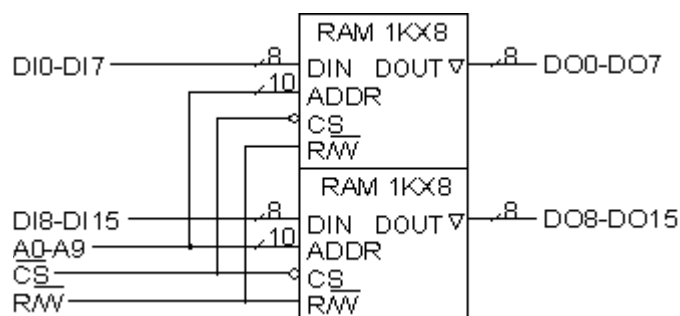
Tehdään neljästä 1k x 8 vaihtomuistista yksi isompi, 4k x 8 kokoinen muisti. Käytännössä tämä tarkoittaa, että osoitelinjojen kaksi eniten merkitsevää bittiä (A10 ja A11) ohjataan dekooderiin (Nythän meillä on oltava 12 osoitelinjaa [A0:A11], jotta saataisiin $4K = 4096$ eri osoitetta ($2^{12} = 4096$)). Dekooderi taas valitsee, mitä pienistä piireistä kulloinkin käytetään (Dekooderiin menevällä kahdella osoitelinjalla saadaan muodostettua 4 kombinaatiota, eli voidaan valita yksi neljästä vaihtoehtoisesta muistipiiristä. Jokaiseen pikkupiiriin menee 10 osoitelinjaa -> 1024 osoitetta). Tällöin kokonaisuus toimii, kuten yksi suurempi muisti.

Käytännössä on valittu mukavasti, joko pienten piirien koko (1k X 8) tai halutun suuren muistipiirin koko (4k x 8), koska osoiteavaruuden laajentaminen menee näin mukavasti.



Sananpituuden lisääminen

Sananpituuden lisäämiseksi kytketään piirejä rinnakkain tarvittava määrä ja ohjataan kaikkiin samat ohjaussignaalit. Tämä on siis paljon helpompaa kuin osoiteavaruuden laajentaminen. Esimerkkinä on muodostettu 1k x 16 - vaihtomuisti 1k x 8 piireistä:



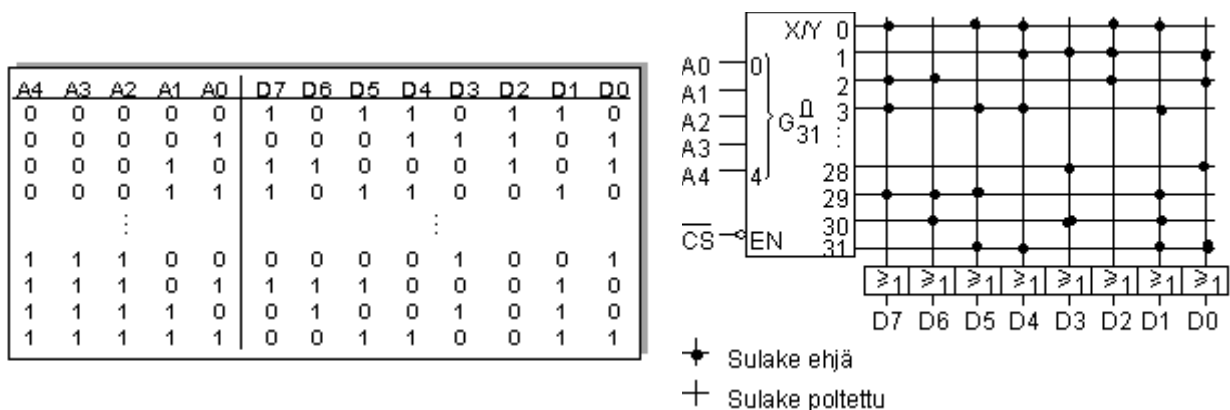
Kiintomuistit

Kiintomuisti (ROM) on muistipiiri, jonka sisältöä ei käytön aikana muuteta lainkaan tai muutetaan vain hyvin harvoin. Kiintomuistin muuttaminen tapahtuu ohjelmoimalla se uudestaan. Kiintomuisti säilyttää sisällönsä, vaikka syöttöjännite katkaistaan. Kiintomuistin sisältö kirjoitetaan muistiin eli muisti ohjelmoidaan jollakin seuraavista tavoista:

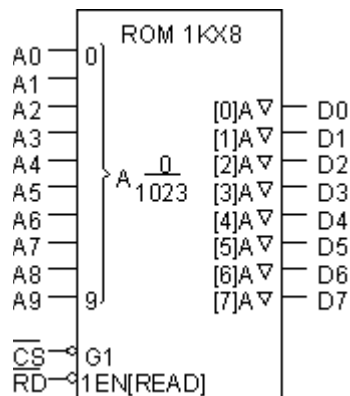
- valmistettaessa tehtaassa
- ennen käyttöä (asiakas ohjelmoi itse)
- laitteessa

Kiintomuistissa muistisolujen paikalla on joko sulakkeet tai eristeen sisäiset kondensaattorit. Muistipaikan valintaan käytetään dekodeeria kuten vaihtomuisteissakin. Antosignaalien muodostukseen käytetään TAI (OR) -piirejä.

Kiintomuistin ohjelma esitetään taulukkona. Taulukko on itse asiassa kiintomuistin totuustaulu. Esimerkkinä esitetään 32 x 8 -ROM piirin ohjelma. Piiri ohjelmoidaan taulukon mukaan:



Kiintomuisteja jaetaan eri tyyppeihin toteutusteknologian ja ohjelmointiominaisuuksien mukaan. Tyyppejä ovat esimerkiksi: Maski-ROM, PROM, OTPROM, EPROM, FLASH ja EEPROM. Kiintomuistin piirrosmerkki esitellään alla:



Ohjelmoitava logiikkaverkko

Ohjelmoitava logiikkaverkko on integroitu piiri, jossa on siis tavallaan ristiin menevä matriisi (vaaka- ja pystysuunnat, kuten ROM-piirissä aiemmin), Ohjelmoitavassa logiikkaverkossa on logiikkaportteja ja/tai erilaisia kytkentämahdollisuuksia. Ohjelmoitava logiikkaverkko ohjelmoidaan valmiiksi (esim. sulakkeita polttamalla) käyttöä varten. Matriisin ansiosta voidaan ohjelmoitava logiikkaverkko ohjelmoida eri käyttötarkoituksia varten eri tavoin kuitenkin niin, että lähtökohta on aina sama (eli verkko, josta ei ole vielä yhtään sulaketta poltettu [tässä esimerkissä. ks. ryhmittely alta]).

Ohjelmoitavia logiikkaverkkoja voidaan ryhmitellä eri tavoin. Alla on esitelty yleisimpiä ryhmittelyjä.

a) Ohjelman säilyvyyden perusteella:

- haihtumaton ohjelmaiset
- haihtuvaohjelmaiset

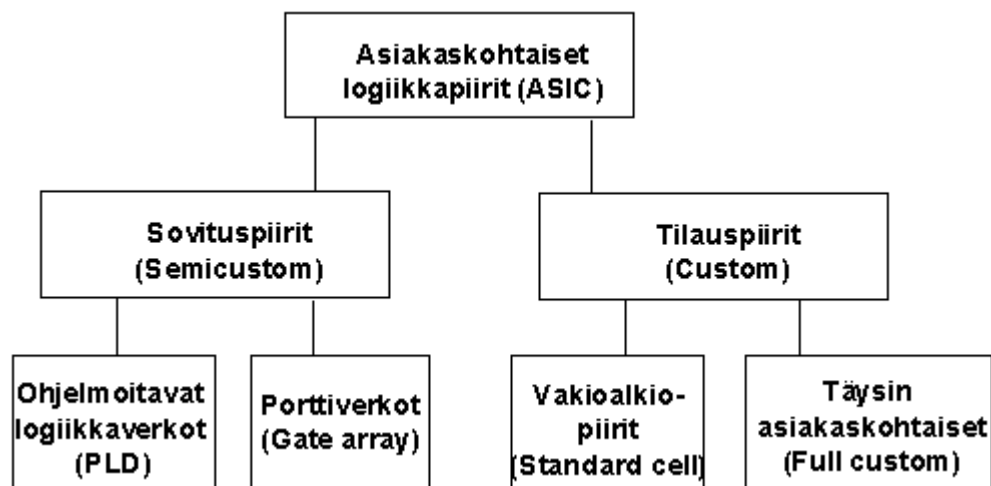
b) Toteutusteknologian perusteella:

- maskilla ohjelmoitavat (ROM)
- sulakeperusteiset (programmable ROM, PROM)
- EPROM -tyyppiset (erasable , programmable ROM)
- EEPROM (electrically erasable, programmable ROM)
- RAM -tyyppiset
- bipolaariset (TTL, ECL)
- unipolaariset (CMOS)

c) Arkkitehtuurin perusteella (joista enemmän seuraavilla sivuilla):

- Programmable Read-Only Memory (PROM)
- Programmable Logic Array (PAL)
- Programmable Array Logic (PLA)

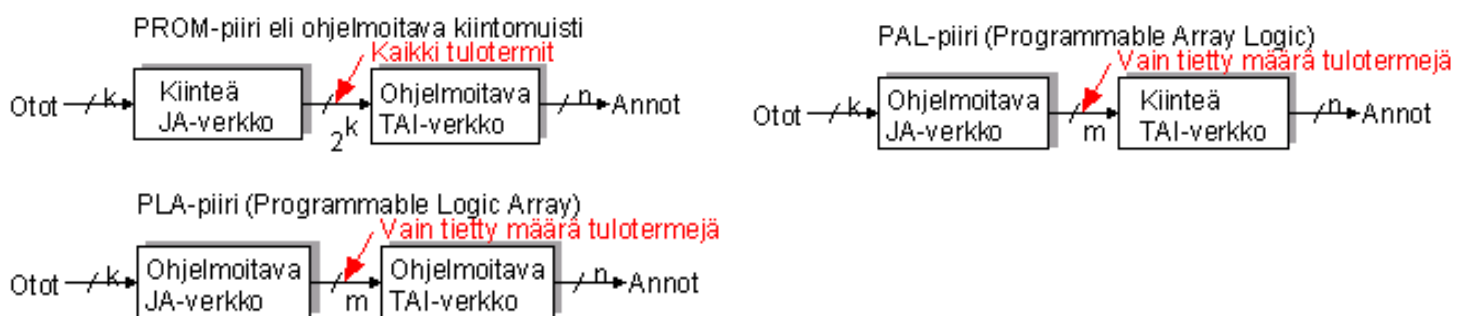
Ohjelmoitavat logiikkapiirit ovat ns. asiakaskohtaisia logiikkapiirejä (ASIC = Application Specific Integrated Circuit). Ne luokitellaan allaolevan kaavion mukaisesti.



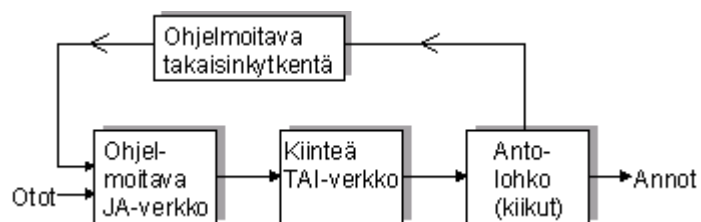
Kombinaatiopiiriverkot

Ohjelmoitavilla logiikkapiireillä, joilla muodostetaan kombinaatiopiirejä on kolme eri perusarkkitehtuuria: PROM (Programmable read-only memory, PAL (Programmable array logic) ja PLA (Programmable logic array). Kaikki sisältävät JA-sekä TAI- verkot. Erona on, että kummatkin verkot voivat olla joko kiinteitä tai ohjelmoitavia:

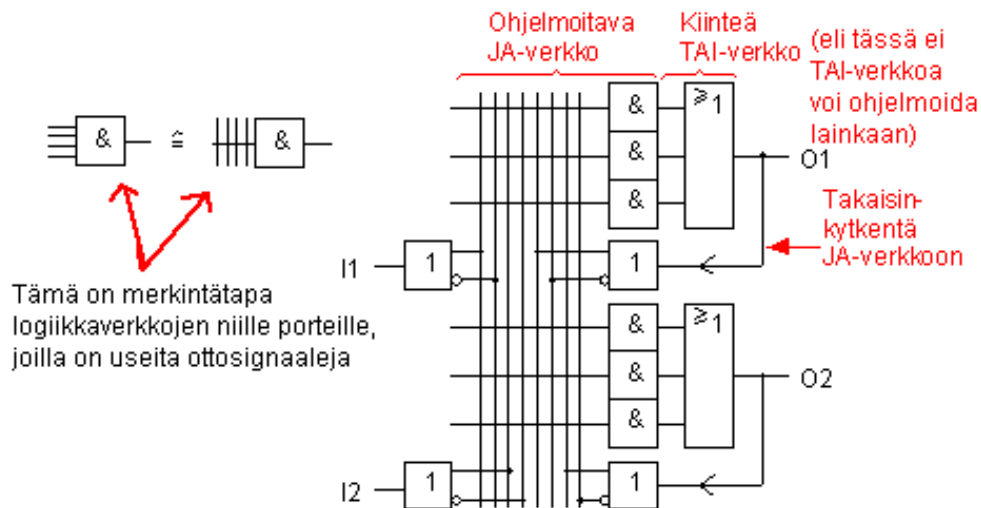
- PROM: kiinteä JA- verkko + ohjelmoitava TAI-verkko
- PAL: ohjelmoitava JA-verkko + kiinteä TAI-verkko
- PLA: ohjelmoitava JA-verkko + ohjelmoitava TAI-verkko



Jos halutaan tarkastella tavallista rekisterillä varustetun verkon arkkitehtuuria, voidaan se esittää oheisen kuva mukaan. (Tässä on huomattava, että antolohko eli kiikut eivät ole välttämätön ominaisuus: saman perusrakenteen voi hyvin esittää ilman antolohkoa. Tällöin se on ehkä helpompi mieltää ylempänä esiteltyjen kombinaatiopiiriverkkojen yleiseksi kaavioksi.)



Ohjelmoitavissa logiikkapiireissä kombinaatioverkkojen suunnittelu on suoraviivaista: ensin poltetaan funktion lausekkeiden mukaiset sulakkeet ja sitten syötetään saadut tulokset mahdolliseen kiinteään verkkoon jäljempänä. Huomattavaa suunnittelussa on se, että verkkojen perusrakenteesta (JA-TAI) johtuen, on halutusta funktiosta muodostettava SOP-muotoinen funktio, jotta sen ohjelmoiminen onnistuisi. Tämä siis johtuu siitä, että ohjelmoitavassa logiikkaverkossa (PLA, PAL ja PROM) tulevat ensin JA-portit ja vasta sitten TAI-portit. Seuraavalla sivulla esitellään PAL -piirin rakennetta lähemmin:



Esimerkit

Toteutetaan seuraavat lausekkeet PAL piirinä:

$$W = ABC' + A'B'CD'$$

$$X = A + BCD$$

$$Y = A'B + CD + B'D'$$

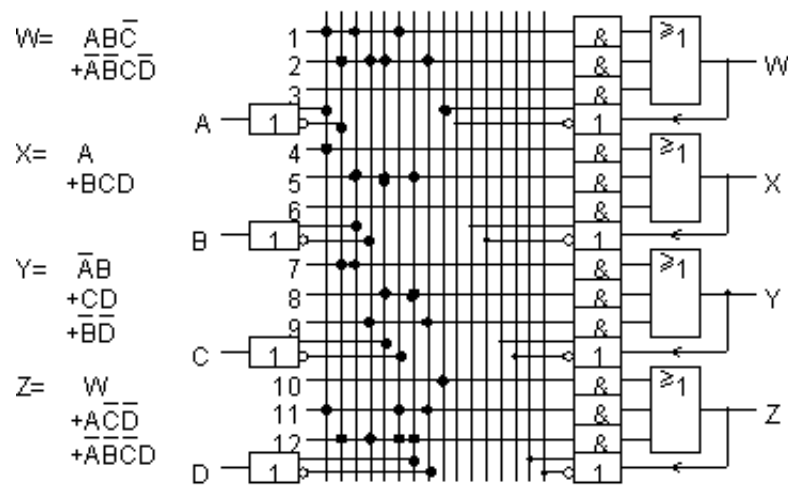
$$Z = ABC' + A'B'CD' + AC'D' + A'B'C'D$$

$$= W + AC'D' + A'B'C'D$$

(Huomataan, että Z:n lausekkeessa voidaan käyttää hyväksi W:n lauseketta.) Luodaan lausekkeista taulukko sen mukaan, mitä muuttujia tarvitaan mihinkin termeihin:

Tulo-termi	Otot						Annot
	A	B	C	D	W		
1	1	1	0	--	--		$W = \overline{A}BC' + A\overline{B}CD'$
2	0	0	1	0	--		
3	--	--	--	--	--		
4	1	--	--	--	--		$X = A + BCD$
5	--	1	1	1	--		
6	--	--	--	--	--		
7	0	1	--	--	--		$Y = \overline{A}B + CD + \overline{B}D'$
8	--	--	1	1	--		
9	--	0	--	0	--		
10	--	--	--	--	1		$Z = W + AC'D' + A'B'C'D$
11	1	--	0	0	--		
12	0	0	0	1	--		

Tämän jälkeen muodostetaan tarvittavat kytkennät: jos lankojen välillä on kytkentä, siinä on kaaviossa pallo. Muussa tapauksessa kosketuspintaa ei ole:



Kuvassa on selkeästi näkyvissä PAL -piirin rakenne: Ensin ohjelmoitava JA -verkko, sitten kiinteä TAI -verkko. PLA-piirien rakenne on muuten saman kaltainen, mutta siinä on käytännössä kolme eri ohjelmoitavaa verkkoa: ensin ohjelmoitava JA -verkko, sitten ohjelmoitava TAI -verkko ja lopuksi ohjelmoitava invertointi:

