TEKNILLINEN KORKEAKOULU
Sähkötekniikan osasto

Maarit Laanti

Nopeuden 2 Mbit/s tilaajamultiplekserin linjatestausmoduulin ohjelmisto

Diplomityö, joka on jätetty opinnäytteenä tarkistettavaksi diplomi-insinöörin tutkintoa varten Espoossa 30.11.1993

TEKNILLINEN KORKEAKOULU **DIPLOMITYÖN TIIVISTELMÄ**

Tekijä ja työn nimi:	Maarit Laanti Nopeuden 2Mbit/s tilaajamultiplekserin linjatestausmoduulin ohjelmisto
Päivämäärä:	30.11.1993 **Sivujen lukumäärä:** 81
Osasto:	Sähkötekniikan osasto
Professuuri:	Tiedonsiirtojärjestelmät, Ele-72
Työn valvoja:	Prof. Seppo J. Halme
Työn ohjaaja:	Dipl. ins. Kimmo Kaitala

Tämä diplomityö käsittelee digitaalisia multipleksereitä ja puhelinjohtojen sähköisten ominaisuuksien mittaamista linjatestausmoduulia käyttäen. Työn varsinaisena aiheena on mittaukset suorittavan ohjelmiston suunnittelu ja toteuttaminen. Johdannossa perehdytään puhelinverkkojen rakenteeseen, niiden digitalisointiin ja tulevaisuuden kehitysmahdollisuuksiin. Samalla tutustutaan lyhyesti aikajakoisen tiedonsiirron periaatteisiin. Luvun lopussa esitellään tilaaja- eli access-verkkojen käsite ja tällaisissa verkoissa käytettävät laitteet.

Toisessa luvussa kuvataan työn lähtökohtia ja sille asetettuja vaatimuksia. Aluksi kerrotaan lyhyesti millaisia aikaisempia johtojen mittalaitteita ja multipleksereitä on olemassa. Tämän jälkeen perehdytään rakennetun uuden tilaajamultiplekserin ohjelmiston ulkoisiin rajapintoihin ja toiminnallisuuteen. Luvun lopussa kartoitetaan linjatestausohjelmiston vaatimukset yksityiskohtaisesti.

Työn kolmas luku määrittelee linjatestausohjelmistossa käytetyn arkkitehtuurin, tietorakenteet ja rajapinnat multiplekserin ohjelmistoon päin. Tämän jälkeen kuvataan linjatestaus verkonhallinnan näkökulmasta ja tarkastellaan valvonnan keskittymisen tuomia muutoksia.

Työn neljäs ja viimeinen luku käsittelee ohjelmiston toteutusta. Aluksi kuvataan käytetyt työkalut ja toteutusympäristö. Tämän jälkeen perehdytään yleisesti reaaliaikaisiin käyttöjärjestelmiin ja havainnollistetaan varsinaista toteutusta käyttäen yhtä ohjelmiston prosessia esimerkkinä. Lopuksi tarkastellaan testausta ja työssä kohdattuja ongelmia sekä niiden ratkaisuja. Liitteet sisältävät esimerkkejä kirjoitetusta koodista, prosessien rakenteesta ja käyttöliittymistä.

Hakusanat:	linjatestaus, tiedonsiirtojärjestelmät, multiplekseri, digitaalinen multiplekseri, digitaalitekniikka, pulssikoodimodulaatio, access network, tilaajaverkko

HELSINKI UNIVERSITY OF TECHNOLOGY

ABSTRACT OF THE MASTER'S THESIS

Author and name of the thesis:	Maarit Laanti Line Test Module Software in Access Multiplexer Operating at 2 Mbit/s
Date:	30.11.1993 **Number of pages:** 81
Faculty:	Faculty of Electrical Engineering
Professorship:	Data communication systems, Ele-72
Supervisor:	Professor Seppo J. Halme
Instructor:	M.Sc. Kimmo Kaitala

This thesis describes digital multiplexers and how telephone lines can be measured in terms of simple electrical parameters by using a line testing module. The subject of the thesis is the specification and implementation of the testing software. In the introduction, the reader is familiarized with the structure and digitalization process of telephone networks and the possible development paths in the future. He is also introduced to the principles of time division based information transfer technology. The concept of the access network and its devices are presented at the end of the chapter.

The second chapter illuminates the starting-point and the requirements of the project. First, the previous generations of digital multiplexers and line testing devices are described. Second, the external interfaces and the functionality of the new access multiplexer are revealed. Finally, the requirements set for the software in line testing module are specified in detail.

The third chapter presents the architecture and data structures used in line testing software. Also the interfaces to the access multiplexer software are described. Finally, line testing is examined from the network management point of view. Also the effects of centralization of network management are discussed.

The subject of the fourth (and last) chapter is the implementation process. First the programming environment and the tools used are described. Next the concept of the real time operating system is introduced and the real implementation process is illustrated by using a sample process. Finally testing and some problems and their solutions encountered during the project are described. The appendices include examples of the process structures, the code written for the project and screen copies from the user interfaces employed in the line testing module.

Keywords: line testing, data transfer, digital technology, data communication systems, multiplexer, digital multiplexer, pulse code modulation, access network

Aluksi

Tämä diplomityö on tehty Nokia Telecommunicationsissa Access Network Products -osastolla, ja kertoo osaltaan ACM2-tuoteperheeseen kuuluvien digitaalisten multiplekserien kehityksestä. Olen ollut mukana mainitun projektin ohjelmistonkehitysryhmässä heinäkuusta -92 lähtien. Ensimmäisessä vaiheessa olen osallistunut tilastointimoduulin ja käyttöliittymän toteutukseen. Projektin toisessa vaiheessa olen suunnitellut linjatestausmoduulin ohjelmiston rakenteen ja arkkitehtuurin sekä toiminut koko projektin laatukoordinaattorina. Tämän lisäksi olen toteuttanut testien käsittelyprosessin, jonka osuus koko linjatestausmoduulin ohjelmistosta on varsin suuri (1384 koodiriviä, n. 45 % linjatestausohjelmistosta poislukien käyttöliittymään liittyvä koodi).

Haluan esittää kiitokseni työni valvojalle, professori Seppo J. Halmeelle, hänen antamastaan kannustuksesta ja osoittamastaan kiinnostuksesta työtäni kohtaan. Samoin haluan kiittää osastoni esimiestä, dipl. ins. Kari Kulkkia mahdollisuudesta tehdä tämä työ. Haluan myös kiittää esimiestäni ja työni valvojaa, dipl. ins. Kimmo Kaitalaa hänen osoittamastaan kiinnostuksesta ja antamistaan ohjeista sekä tämän työn asiasisällön tarkistamisesta. Kiitän myös ACM2-projektiryhmiä antoisasta yhteistyöstä; erityisesti ensimmäisen vaiheen projektipäällikköä, dipl. ins. Hannu Jokista, joka on auliisti jakanut tietotaitoaan muidenkin käytettäväksi. Kiitän myös kaikkia niitä työtovereitani, ystäviäni ja tuttaviani, joiden kanssa käymäni antoisat keskustelut ovat omalta osaltaan vaikuttaneet tämän työn lopputulokseen tai jotka ovat muuten edistäneet työtäni. Lopuksi haluan kiittää aviomiestäni dipl. ins. Harri Kiljanderia hänen antamastaan tuesta opiskeluni kaikissa vaiheissa. Ilman hänen antamiaan kieliasua ja ulkonäköä koskevia ohjeita tätä työtä olisi paljon ikävämpi lukea.

Espoossa, 30.11.1993

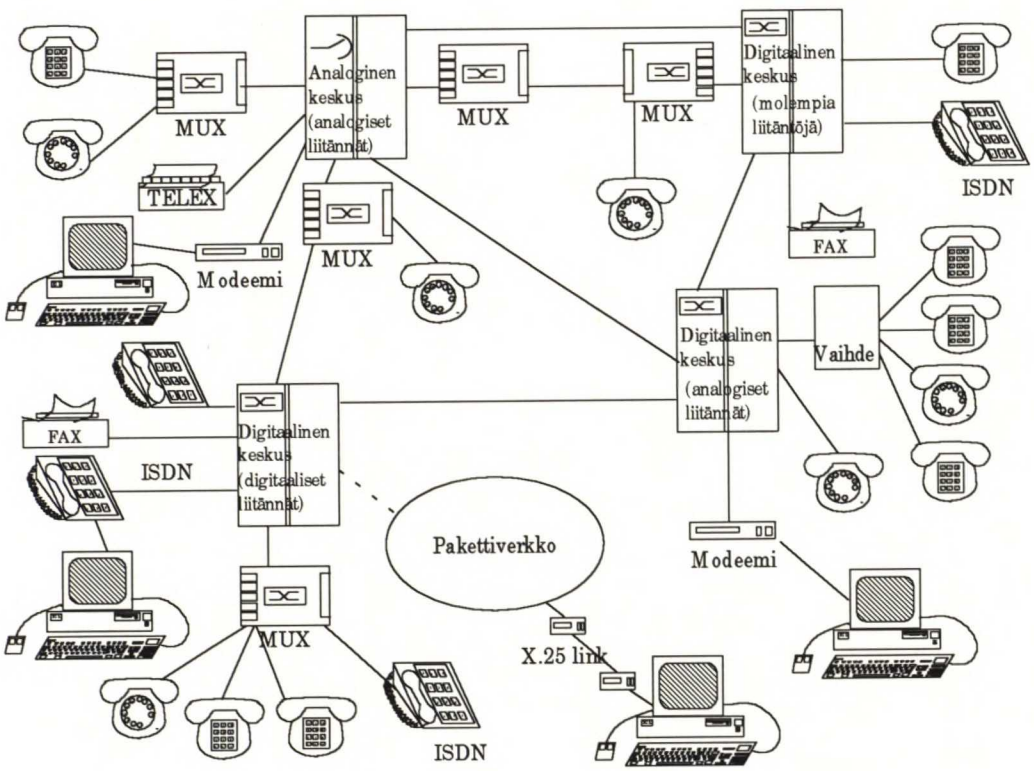
Maarit Laanti

Sisällysluettelo

1. Johdanto

Tässä luvussa käsitellään puhelinverkon rakennetta, kehitystä ja digitalisoimista sekä perehdytään access-verkon käsitteeseen. Tarkoituksena on antaa lukijalle riittävät taustatiedot ACM2-projektissa kehitetyn digitaalisen multiplekserin vaatimusten ja toimintaympäristön ymmärtämiseen.

1.1 Puhelinverkon rakenne



Kuva 1. Puhelinverkon rakenne

Yllä on yksinkertainen kaavakuva siitä, millaisista elementeistä yleinen puhelinverkkomme (PSTN) koostuu. Verkko sisältää kuvan mukaisesti analogisia ja digitaalisia keskuksia, vaihteita, analogisia ja digitaalisia puhelimia, multipleksereitä, kaukokirjoitin- ja telekopiolaitteita, tietokoneita modeemeineen, paketti- ja ISDN-yhteyksiä sekä siirtojohtoja. Näiden lisäksi verkossa voi

olla muitakin laitteita, kuten kuvapuhelimia, videotex- ja teletex-laitteita tai puhepostilaitteita. Pakettiverkon tilalla voisi aivan hyvin olla myös piirikytkentäinen datex-verkko, digitaalinen (GSM) tai analoginen (NMT) matkapuhelinverkko.

Keskukset voidaan luokitella kansainvälisen liikenteen keskuksiin, kauttakulkukaukokeskuksiin, päätekaukokeskuksiin, verkkoryhmäkeskuksiin, solmukeskuksiin ja päätekeskuksiin.

Yhteydet voidaan jaotella yhteystyypin mukaan kiinteisiin ja valinnaisiin yhteyksiin, tai sijaintinsa mukaan keskusten välisiin yhteyksiin sekä tilaajajohtoihin. Lisäksi siirtojohtojen kapasiteetti, tiedonsiirtotapa tai valmistusmateriaali voi vaihdella. Käytetyimmät tiedonsiirtotavat ovat kantataajuinen siirto, kantoaaltoyhteydet sekä PCM-yhteydet. Joissakin tapauksissa yhdysjohdon voi korvata myös mikroaaltolinkki.

Historian varhaisimmassa vaiheessa puhelinverkko koostui pelkästään kahden tilaajan välisistä kiinteistä analogisista yhteyksistä. Myöhemmin mukaan tulivat ensin käsivälitteiset ja sittemmin mekaaniset keskukset, jolloin suurin osa siirtoyhteyksistä muutui valinnaisiksi.

1.2 Digitalisointi ja kehitysnäkymät

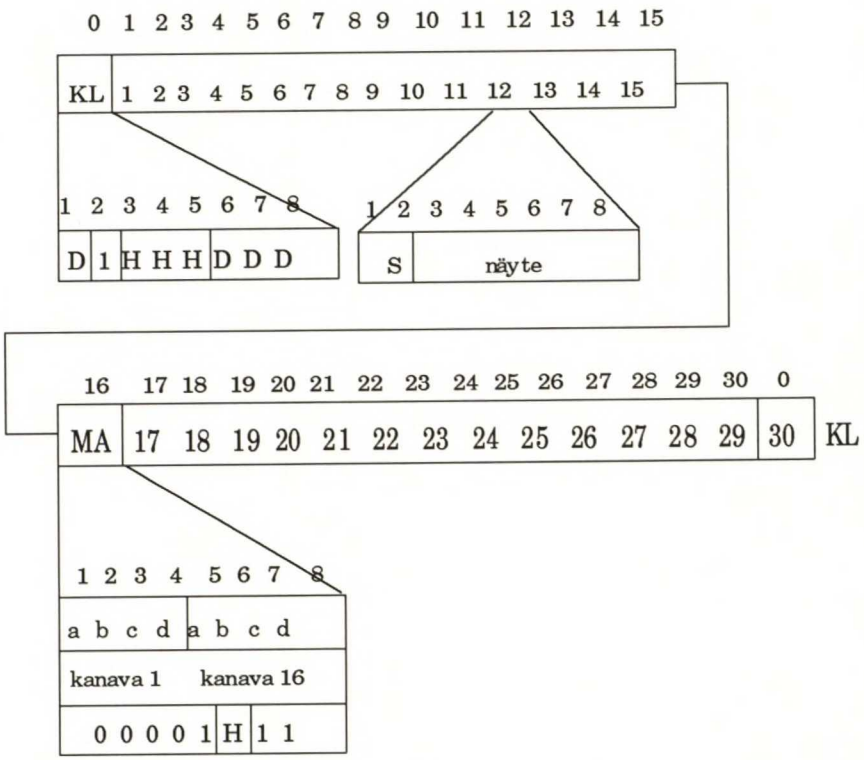
Puhelinverkon digitalisointi on kulkenut käsi kädessä yhdessä aikajakoisen tekniikan kehittymisen kanssa. Aikajakoisen informaation siirron kehitti englantilainen Alec Reeves 1937. [Nordman 1973, s. 1] Tätä ennen kaikki siirto oli tapahtunut tilajakoisesti. Tilajakoisella siirrolla tarkoitetaan sitä, että tilaajalle ja puhelun vastaanottajalle varataan oma, yksityinen siirtoyhteys puhelua varten. Aikajakoisella siirrolla tarkoitetaan puolestaan sitä, että useampi puhelu välitetään samassa siirtotiessä, joka on jaettu useaan aikaväliin. Kullekin tilaajalle ja puhelun vastaanottajalle varataan vain yksi aikaväli, ei koko siirtotietä.

Reevesin ajatuksia ei heti sellaisenaan voitu soveltaa käytännössä. Sen aikaisella tekniikalla tarvittavia kytkentöjä ja loogisia laskutoimituksia ei voitu suorittaa riittävän nopeasti. Transistorin (ja myöhemmin loogisten piirien) keksimisen jälkeen 1948 tilanne muuttui oleellisesti. [Volotinen 1984, s. 12] Ensimmäiset koekäytöt suoritettiin Yhdysvalloissa 24 kanavan PCM-järjestelmällä, joka sittemmin tuli tunnetuksi tyyppimerkinnällä T1. [Norman 1973, s. 1] Suomeen PCM:n perusajatukset tulivat eestiläistä syntyperää olevan yhdysvaltalaisen ATT:n insinöörin Helmo Raagin siirtyessä Nokia Elektroniikan palvelukseen. [Leivo 1992, s. 24] Ensimmäinen suomalaisvalmisteinen 30-kanavainen PCM-järjestelmä esiteltiin CCITT:lle 1968. Järjestelmä otettiin Suomessa käyttöön jo samana vuonna kahden Helsingissä sijainneen analogisen keskuksen välisellä yhdysjohdolla. [Volotinen 1984, s. 13]

Pulssikoodimodulaation perusjärjestelmässä lomitetaan 30:n kanavan näytteet peräkkäin (kuva kaksi). Tarkasteltava kehys käsittää 32 aikaväliä ja sen kesto on

125 µs, jolloin kehystaajuus on sama kuin näytetaajuus (8.0 kHz). Kehyksen nollas aikaväli käsittää ns. kehyslukitussanan, jonka perusteella vastaanotettavaa signaalia pystytään lukemaan. Aikaväliin 16 on sijoitettu merkinantosignaalit, joita varten on käytössä neljä kanavaa a, b, c ja d. Merkinannon hallitsemiseksi on muodostettu 16 kehyksen ylikehys (pituus 2 ms, toistotaajuus 500 Hz), jonka alku tunnistetaan ylikehyslukitusmerkistä.

2048 kbit/s PCM-signaali muodostaa ensimmäisen asteen Euroopan digitaalisten PCM-moduloitujen signaalien hierarkiassa. Useampia pulssikoodimoduloituja signaalivirtoja voidaan edelleen multipleksoida eli lomittaa. DM2 ja ACM2 kuuluvat multiplekserien joukossa ensimmäisen asteen multipleksereihin. [Metsälä 1992, s. 28] Toisen asteen laitteissa on yhdistetty $4 \times 30 = 120$ kanavaa 8448 kbit/s signaaliksi. Kolmannessa asteessa yhdistetään $4^2 \times 30 = 480$ kanavaa 34 Mbit/s signaaliksi jne. [Halme 1981, s. 205...208, CCITT G.704, s. 76...96]



Kuva 2. Pulssikoodimodulaation kehysrakenne

Digitaalikomponenttien kehittyminen on mahdollistanut myös keskusten digitalisoimisen. Samalla voidaan siirtyä myös keskuksissa tilajakoisesta kytkennästä aikajakoiseen kytkentään. Ensimmäinen digitaalinen puhelinkeskus otettiin Suomessa käyttöön 1978. [Volotinen 1984, s. 12]

Edullisimmillaan digitaaliset välitysjärjestelmät ovat digitaalisiin siirtojärjestelmiin kytkettyinä. Analogiset siirtotiet voidaan kytkeä digitaaliseen keskukseen digitalisoimalla siirtotieltä tuleva analoginen signaali AD-muunninta apuna käyttäen. AD-muunnin poimii signaalista näytteitä tietyllä taajuudella ja kvantisoi saamansa näytteet. Muunnos digitaalipuolelta analogiapuolelle tapahtuu DA-muuntimella,

joka muuttaa peräkkäiset näytteet peräkkäisiksi jännitepiikeiksi ja interpoloi jollakin sopivalla algoritmilla näytteiden välit.

Aikajakoisen järjestelmän etuna taajuusjakoiseen (tilajakoiseen) järjestelmään verrattuna on mm. parempi siirtoteiden häiriöiden sietokyky, datasiirron nopeuden kasvu, optisen kuidun soveltamismahdollisuudet, luotettavuus (viallinen yksikkö helposti vaihdettavissa toiseen) ja lisäksi puheen laadun riippumattomuus käytetyn siirtotien pituudesta (regeneroivia toistimia käytettäessä). [Volotinen 1984, s. 10] Siirtoetäisyyttä digitaalisilla siirtojohdoilla rajoittavat kuitenkin mm. ylikuuluminen ja impulssihäiriöt. [Halme, 1981, s. 25]

Viimeisenä digitalisointi ulotetaan tilaajajohtoihin. Suomessa tilaajajohtojen digitalisointi aloitettiin 1986. [Tammelin 1990, s. 1] Tämän jälkeen tilaajalla on mahdollisuus kokonaan digitaalisen siirtojärjestelmän käyttöön. Samalla mahdollistuvat eräät uudet puhelinverkon palvelut, kuten koputus, kutsunsiirto, erilaiset hälytys- ja hätäkutsupalvelut, ryhmäneuvottelut, puhepostilaatikot (digitaalisen keskuksen tarjoamia ominaisuuksia), suuremmat datasiirtonopeudet, parempi puheen laatu ja suurempi käyttömukavuus.

Oman lisänsä digitalisointikehitykseen antaa myös suunniteltu monipalveluverkko ISDN, josta kansainvälinen telealan standardoimisliitto CCITT antoi ensimmäiset suosituksensa 1984. Monipalveluverkon perusideana on tarjota äänen, kuvan, tekstin ja datan siirtoon liittyvät palvelut samasta liittymästä. ISDN:ssä tilaajalle tarjotaan kaksi datan tai äänen siirtoon tarkoitettua 64 kbit/s B-kanavaa ja yksi 16 kbit/s merkinantoon tarkoitettu D-kanava. Myöhemmässä vaiheessa D-kanavaa voidaan käyttää myös pakettivälitteiseen datasiirtoon. ISDN-palveluita on ollut tarjolla Suomessa pääkaupunkiseudulla vuodesta 1992. [Tammelin 1990, s. 2...4]

1.3 Tilaajaverkon laitteet

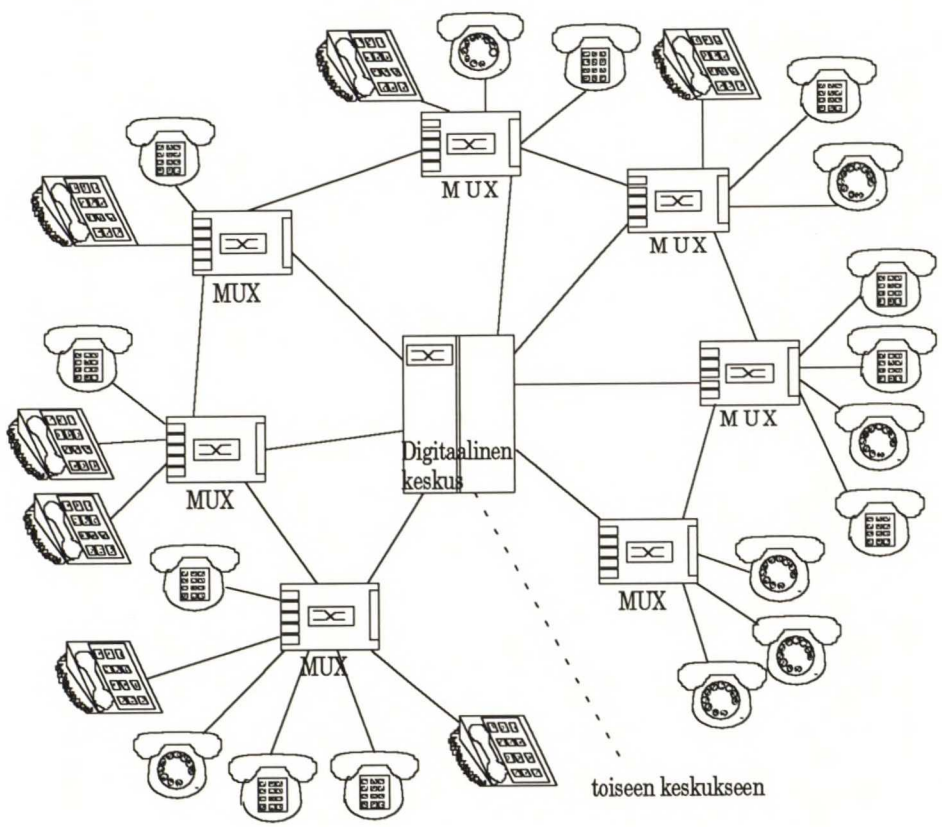
Tilaajaverkolla (*access network*) tarkoitetaan sitä puhelinverkon osaa, joka on tilaajan ja keskuksen välillä. Tilaajaverkko on usein plesiokroniseen PDH-tekniikkaan perustuva pienikapasiteettinen verkko, mutta se voi sisältää myös suurikapasiteettisia synkronisia SDH-renkaita. Ominaisuuksiltaan se on helposti muuntuva ja nopeasti laajennettavissa. Vastakohta tilaajaverkolle on ydinverkko (*core network*), jolla tarkoitetaan etupäässä puhelinkeskusten välillä sijaitsevaa muuttumatonta, suurikapasiteettista ja usein kuitupohjaista verkkoa.

Tässä esityksessä tarkasteltavana oleva laite, ACM2, luetaan tilaajaverkon tuotteisiin. ACM2:een voidaan liittää ensimmäisessä vaiheessa enintään 30 kappaletta tavallisia analogisia tilaajajohtoja. Toisessa vaiheessa siihen voidaan liittää vastaava määrä puheen siirtoon tarkoitettuja digitaalisia 64 kbit/s tilaajajohtoja tai 31 kappaletta pelkän datan siirtoon tarkoitettuja johtoja (koska siirrettäessä pelkkää dataa voidaan myös merkinantokanavaa käyttää datan

siirtoon). ACM2 pakkaa keräämänsä bitit PCM-kehykseen, ja lähettää ne yhtä nopeuden 2048 kbit/s digitaalista linjaa pitkin keskukseen. [Metsälä 1992, s. 8]

Multiplekserеitä voidaan käyttää myös muualla kuin access-verkossa. Keskusten välisistä yhteyksistä osa saattaa olla toteutettu multiplekserеillä. Näin vältytään muuten tarvittavilta lisäkaapeloinneilta. Osa multiplekseriin tulevista linjoista voidaan myös haaroittaa tilaajille ja vain osa viedä toiseen keskukseen erityisillä haaroitusta varten rakennetuilla haaroitinlaitteilla, joissa on yhden 2 Mbit/s liitännän sijasta kaksi tai useampia 2 Mbit/s liitäntöjä. Kustakin liitännästä voidaan ottaa käyttöön vain osa kapasiteetista. Tässä kappaleessa kuvatut multiplekserin käyttötavat näkyvät myös kuvasta yksi. [DB2 1991, s. 1...5 ja DN2 1992, s. 1...11]

Teleoperaattorille panostus access-verkkoon tulee takaisin paitsi keskuksen pienempänä tilantarpeena myös pienempien jännitelähteiden tarpeena. Joissakin tapauksissa selvitään myös pienemmillä jäähdytysjärjestelmillä. Lisäksi säästöjä voidaan hakea vähentämällä vanhan kuparikaapeliverkon ylläpitoa. Tulipalon tai muun onnettomuuden sattuessa on hyvä, että kaikki laitteet eivät sijaitse samoissa tiloissa. Teollisuusalueilla teleoperaattori voi muuttaa keskuksen ja multiplekserin välisen yhteyden optiseksi, jolloin asiakkaan vaatiessa suurempikapasiteettista siirtoyhteyttä se on nopeammin toteutettavissa. [Sivertsen 1992, s. 2]



Kuva 3. Esimerkki access-verkon toteutuksesta yhden keskuksen alueella

Kuvassa kolme on esimerkki access-verkon toteutuksesta yhden keskuksen alueella. Kuvassa on multiplekserien ja keskuksen väliset johdot varmistettu solmujen välisin siirtojohdoin. Keskus voidaan automaattisesti ohjelmoida muuttamaan siirtotietä, mikäli jokin yhdyskaapeli havaitaan vialliseksi.

Teleoperaattorit toteuttavat keskusten ja multiplekserien väliset yhteydet usein hyödyntämällä vanhaa kuparikaapeliverkkoa. Koko access-verkko voidaan myös toteuttaa käyttämällä optista kuitua. Tällöin eräs ratkaisu on käyttää optista passiivista jakeluverkkoa (PON, OPDN), jossa koko verkko on toteutettu käyttämällä optisia, passiivisia komponentteja. Aikajakokanavoinnin sijaan voidaan käyttää siirrettävän datan kompressointia. Optinen tilaajaverkko voidaan kytkeä muuhun siirtoverkkoon joko optisesti tai sähköisesti. Tilaajaliitäntä voi olla optinen (FTTH, FTTO) tai sähköinen (FTTA, FTTB). [ETR 1992, s. 4...12]

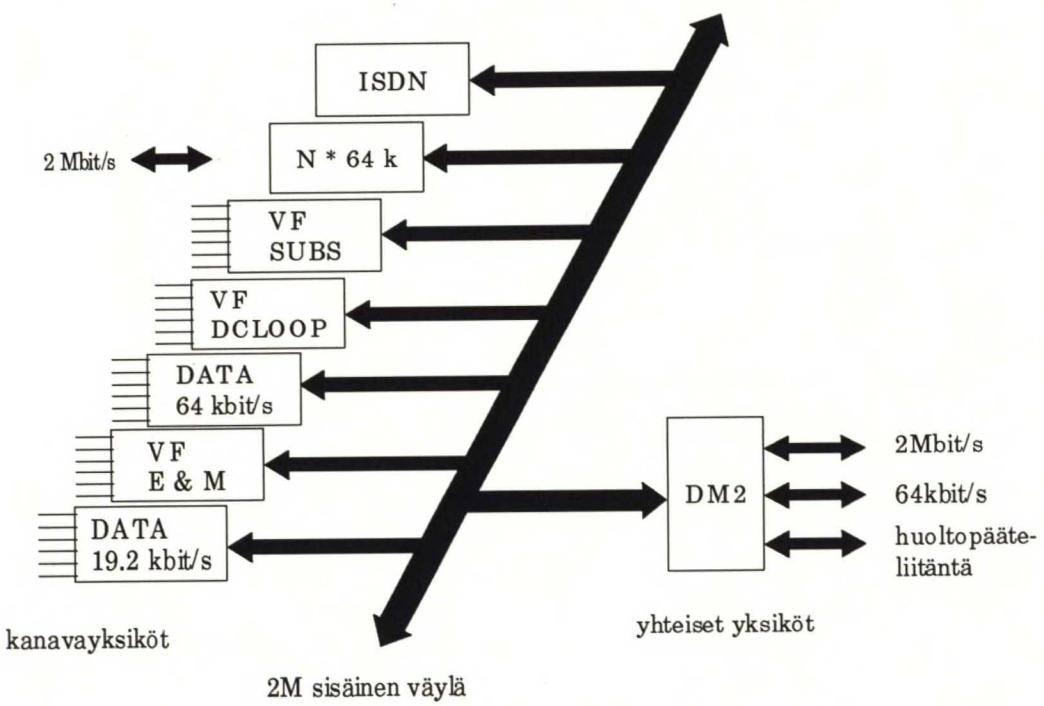
Puhelinpalvelut voidaan myös vaihtoehtoisesti tarjota jonkun muun palvelun (esim. digitaalisen televisiosignaalin) yhteydessä. Tällöin käyttäjä saa molemmat palvelut saman siirtotien välityksellä. Lähelle loppukäyttäjää sijoitetaan purkulaite, joka erottaa signaalit toisistaan.

2. Lähtökohdat ja vaatimukset

Tässä luvussa käsitellään niitä vaatimuksia, joita linjatestauksen osaprojektilla oli sekä niitä lähtökohtia, jotka toimintaympäristö ja aiemmin tehdyt ratkaisut projektille asettivat.

2.1 DM2 ja LTU, edellinen laitesukupolvi

DM2



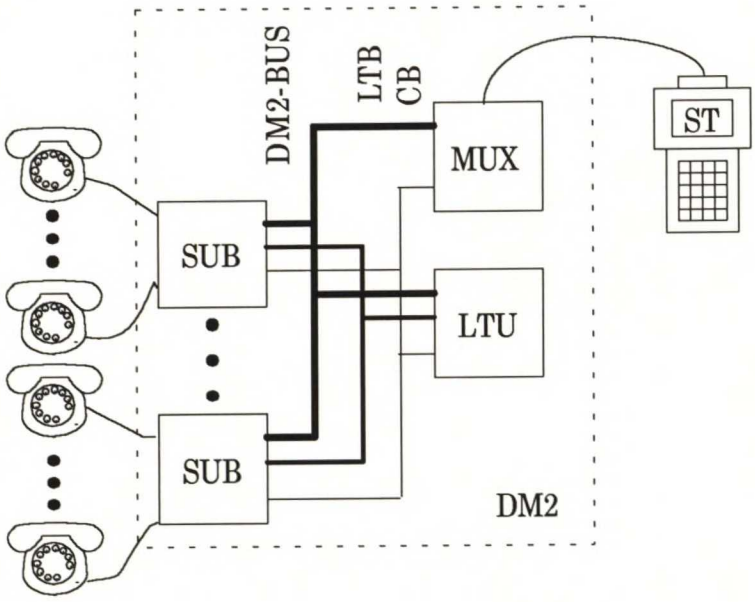
Kuva 4. DM2:n lohkokaavio

DM2 yhdistää maksimissaan 30 PCM-koodattua puhekanavaa tai 31 datakanavaa yhdeksi 2048 kbit/s signaaliksi. DM2 edustaa ensimmäisen asteen laitetta Euroopan digitaalisten multiplekserien hierarkiassa. DM2 koostuu yhdestä yhteisestä multiplekseriyksiköstä ja useista analogisista tai digitaalisista tilaajan liitäntäyksiköistä. Tilaajan liitäntäyksiköinä käytetään prosessorillisia Dynacard-

yksiköitä. Dynacard-yksiköitä käytetään myös keskuksen päässä liittämään 2M:n signaali analogiseen keskukseen. [DM2 1991, s. 1...4]

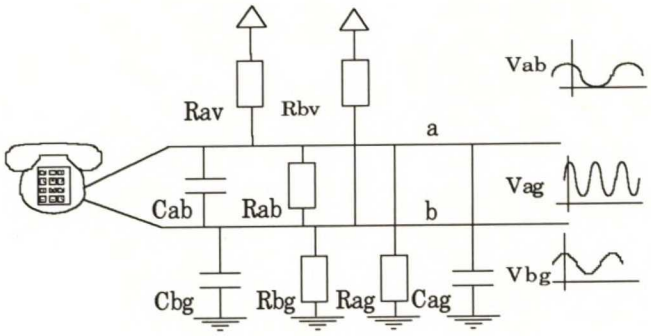
Line Test Unit

Line Test Unit (LTU) on kanavakortti, jolla voidaan mitata DM2:een liitettyjä puhelinjohtoja. LTU-kortti sisältää prosessorin, ohjausreleitä, vahvistimia, digitaalisuodattimia ja AD- sekä DA-muuntimia. LTU:n parametreja ja asetuksia voidaan muuttaa joko huoltopäätteen (ST) tai sitä emuloivan verkonhallintatietokoneen (TMS) avulla. Huoltopääte liittyy MUX:iin, josta linjatestausta koskevat komennot ohjataan komentoväylälle (CB). DM2 ohjaa LTU:ta saman komentoväylän kautta. Varsinainen tilaajajohtojen testaus tapahtuu erillisen testausväylän (LTB) avulla.



Kuva 5. LTU ja DM2. LTU:n ohjaus ST:n avulla

Kun LTU saa komentoväylältä mittauspyynnön, kytkeytyy se ohjausreleiden avulla testausväylälle. Kukin tilaajajohto testataan vain, mikäli se on vapaana (IDLE-tilassa) ja johdon testausta ei ole valinnoilla estetty. LTU tekee seuraavat mittaukset:



Kuva 6. LTU:n mittaukset

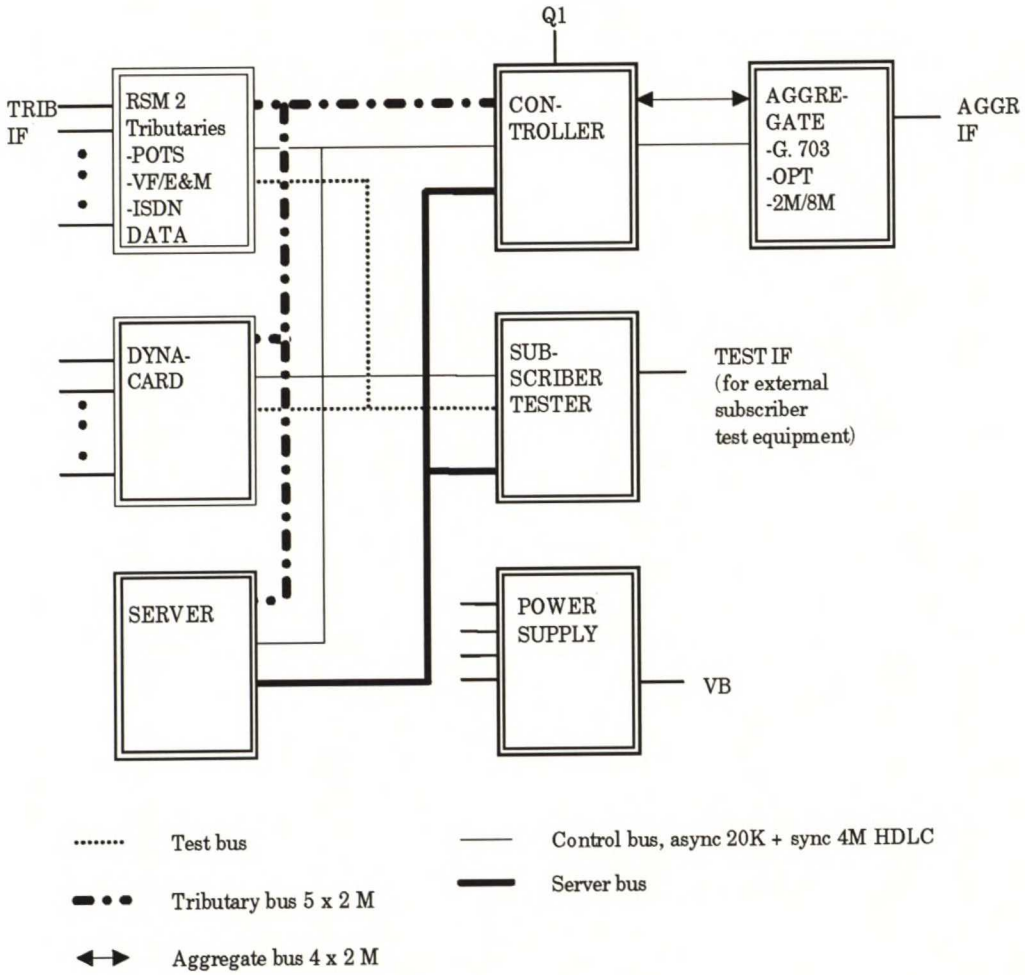
1) *Ylijännitteet*
2) *Vieraat AC- ja DC-komponentit A- ja B-johdoissa*
3) *Vuotoresistanssit maan suhteen*
4) *Vuotoresistanssit käyttöjännitteen suhteen*
5) *A- ja B-johtojen välinen vuotoresistanssi*
6) *Kapasitanssit maan suhteen*
7) *A- ja B-johtojen välinen kapasitanssi*

LTU tallettaa mittaustiedot ja mitattavien johtojen tilat tietokantaan, josta operaattori voi halutessaan nähdä testien tulokset. Mikäli joku mittaustuloksista ylittää käyttäjän määrittelemät raja-arvot, annetaan hälytys. Käyttäjä näkee tämän LTU-kortissa olevan keltaisen ledin syttymisenä. [Awad 1992, s. 3...5]

2.2 ACM2-projekti

Prosessorien kehittyminen nopeammiksi ja suorituskykyisemmiksi on johtanut tilanteeseen, jossa on voitu kyseenalaistaa kunkin kanavakortin tarve omaan prosessoriin. Tästä syystä päädyttiin kehittämään laitetta, jossa yksi, emokortilla sijaitseva prosessori hoitaisi myös kanavayksiköiden signalointitarpeet. ACM2-kehitysprojekti alkoi keväällä 1992 ja jatkuu edelleen. Muita projektille asetettuja tavoitteita oli mm. lisätä yhdellä kortilla olevien kanavien määrää.

ACM2-projektin ensimmäisessä vaiheessa toteutetaan laite, joka pystyy kommunikoimaan vain ACM-SUB-yksiköiden kanssa. ACM-SUB:it mahdollistavat ensimmäisessä kehitysvaiheessa vain liitännän konventionaalisiin analogisiin tilaajajohtoihin. ACM2:n toisessa vaiheessa on tarkoitus lisätä Dynacard-tuki, SUB-EXC-toiminta sekä funktiot tilaajajohtojen testausta varten.



Kuva 7. ACM2:n sisäinen rakenne

Kuvassa seitsemän on esitetty ACM2:n sisäinen rakenne lähinnä väylöityksen näkökulmasta. Kuvassa vasemalla näkyvät liitäntäpiirit tilaajaan päin sekä serveri eli palvelin. Kuvassa oikealla näkyvät kontrolleri, liitäntä keskukseen päin, mahdollinen testauslaite sekä virtalähde. [Riskilä 1992, s. 6...7]

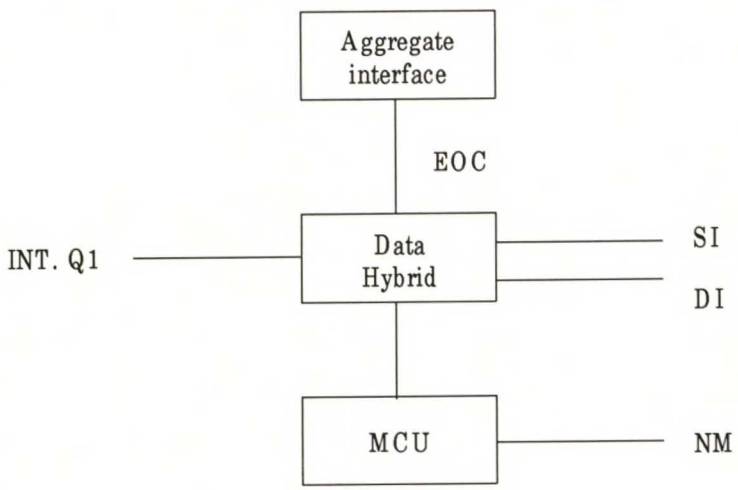
Eräs projektille asetettu vaatimus oli, että ACM-SUB-kortteja ja Dynacard-kortteja voidaan käyttää sekaisin samassa laitteessa. Tämä mahdollistuu sillä, että molemmat keskustelevat MCU:n kanssa saman tributary-väylän kautta. (Tributary-väylä koostuu viidestä 2M:n väylästä, joista yhdellä kulkee ACM-SUB-dataa ja toinen on Dynacard-väylä. Lisäksi on kolme GCI-väylää.) Väylällä siirrettävä tieto on kuitenkin luonnollisestikin erilaista, koska Dynacardit hoitavat signalointinsa itse, ja MCU hoitaa ACM-SUB:ien signaloinnin.

ACM2 käyttää pulssikoodimodulaatioon CCITT:n A-lakia. Siirtoa varten saatu signaali koodataan vielä HDB3-johtokoodilla. [CCITT G.703, s. 45..75] Kellosignaaliksi voidaan valita joko sisäisen kellon signaali, tulopuolen 2M:n tai 64k:n digitaalisesta signaalista erotettu kellosignaali tai ulkoisesta kellosta saatu signaali. Prosessorina käytetään reaaliaikasovelluksia varten kehitettyä MC68302-prosessoria. [Metsälä 1992, s. 28...30]

Teoreettisesti ACM2 on palveluja tasapuolisesti kaikille tilaajille tarjoava multiplekseri. Käytännössä kuitenkin jotkut signalointifunktiot saattavat rajoittaa yhtäaikaisten puheluiden määrää. Tällainen funktio voi olla esimerkiksi soittoäänen generointi, mikäli se vaaditaan yhtä aikaa kaikille tilaajille. Tarvittaessa kapasiteettia voidaan lisätä ylimääräisellä ulkoisella soittogeneraattorilla.

ACM2:ssa suunnittelun tavoitteina ovat olleet etenkin luotettavuus, toimintavarmuus, ylläpidettävyys, ymmärrettävyys ja joustavuus. [Väärä 1993a, s. 13-14]

Ulkoiset rajapinnat



Kuva 8. ACM2:n ulkoiset rajapinnat

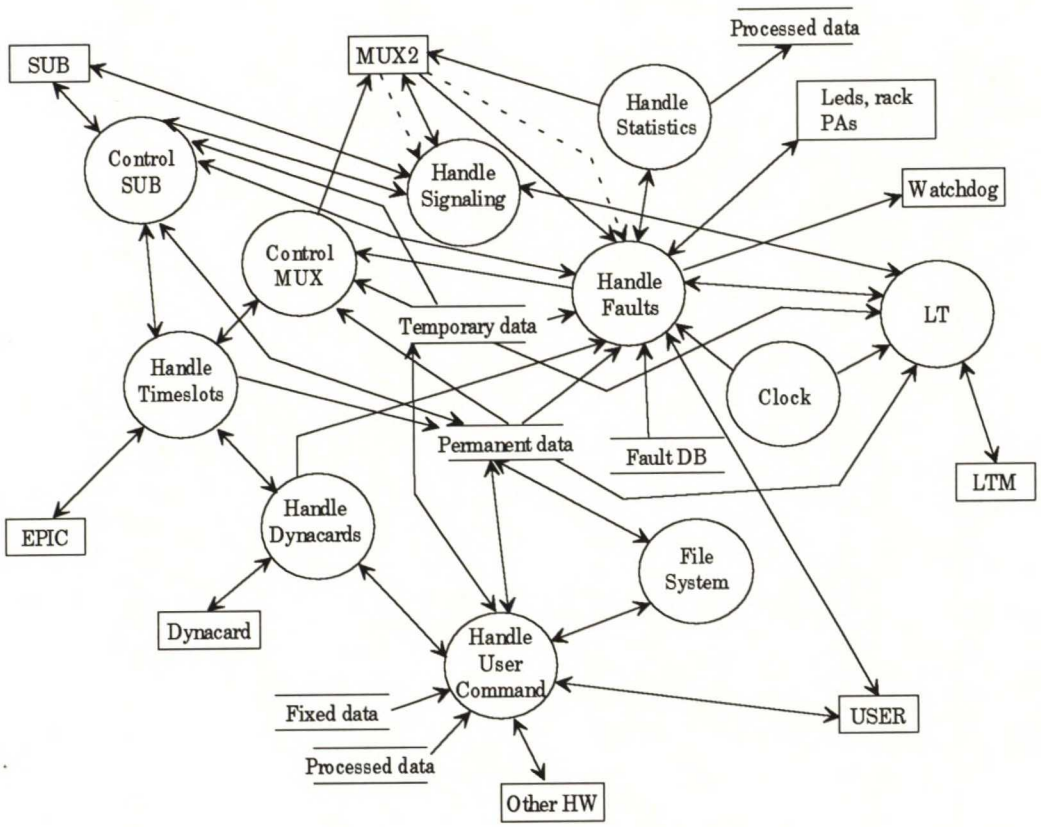
Kuvassa kahdeksan on esitetty ACM2:n käyttäjärajapinnat. Tilaajamultiplekseriä voidaan ohjata joko huoltopäätteellä (*Service Terminal*), huoltopääteemulaattorilla (STE, *Service Terminal Emulator* tai TMC, *Transmission Management Computer*) tai ns. ACM2 Node Managerilla (NM).

> *SI-rajapinta (*Service Interface*) käsittää CCITT:n V.11-suosituksen mukaisen liittimen, johon voidaan liittää joko huoltopääte (*Service Terminal*) tai TMC. Rajapinta noudattaa Q1-protokollaa. [CCITT, Q.771] DI-rajapinta (*Data Interface*) käsittää samoin V.11-liittimen. Rajapintaa voidaan käyttää Q1-väylän laajennuksena ohjaamalla datahybridiä siten, että SI on yhdistetty DI-rajapintaan. Aggregate-liitäntäsignaali kuljettaa mukanaan EOC:tä (*Embedded Operation Channel*). Kanavaa käytetään Q1-datan siirtämiseen laitteelta toiselle. EOC voi käyttää joko yhtä, kahta tai neljää aikavälin nolla (käyttäjä voi valita aikavälin) vapaista biteistä. Kanava toimii näytteenottoperiaatteella näytteenottotaajuuden ollessa 4, 8 tai 20 kHz riippuen käytettyjen bittien lukumäärästä. NM-rajapinta (*ACM2 Node Manager*) koostuu RS-232-liittimestä (CCITT:n suositus V.24/V.28). INT Q1 tarkoittaa sisäistä Q1-väylää. MCU:lla sijaitsevaa datahybridiä käyttämällä voidaan liittyä Q1-väylään EOC:n kautta, ja Q1-väylä voidaan jatkaa joko sisäisen Q1-väylän tai DI-liitännän kautta. [Väärä 1993a, s. 15...17]*

Huoltopääte on kannettava laite, jossa on neljän rivin matriisinäyttö ja yksinkertaistettu näppäimistö. Sen käyttöliittymä koostuu hierarkisista valikoista.

TMC:n ruudulla käyttäjä näkee vastaavat huoltopäätteen valikot. TMC:llä on mahdollista tehdä täsmälleen samat toimenpiteet kuin huoltopäätteelläkin. Vielä kehitteillä oleva Node Manager sisältää sen sijaan graafisen käyttöliittymän, ja on myös toiminnoiltaan laajin. Node Managerilla on mm. mahdollista määritellä makroja, jotka suorittavat pitkiä komentosarjoja yhdellä komennolla.

Toiminnallisuus



Kuva 9. 2M tilaajamultiplekserin toiminnallisuus

Kuvassa yhdeksän on esitetty 2M tilaajamultiplekserin toiminnallisista vaatimuksista piirretty SA-kaavio, jossa moduulit on esitetty ympyröin ja niiden välistä sanomaliikennettä on kuvattu nuolin. Tietovarastot on esitetty kahdella vaakasuoralla viivalla, ja ohjelmistolle näkyvät laitteiston osat (joita SA-kielellä kutsutaan terminaattoreiksi) on kuvattu suorakaitein. [Väärä 1993a, s. 13]

Kuvasta puuttuu initialisointi- eli alustusprosessi, joka käynnistyy itse aina automaattisesti ja herättää tämän jälkeen kaikki muut prosessit säädetyssä, järkevässä järjestyksessä.

Aikavälien käsittely (Handle Timeslots)

Aikavälien käsittelyprosessi pitää kirjaa aikavälien käytöstä ja konfiguroi EPIC-piiriä. Aikavälejä (31 kpl) annetaan käyttöön pyyntöjärjestyksessä joko kokonaisina tai osina. Yhden aikavälin koko on 64k, minimissään annetaan käyttöön 8k. Aikavälin 16 käyttö riippuu valitusta signalointitavasta (ja viime kädessä siitä, käytetäänkö kanavia puheen vai datan siirtoon, katso luku 1.3 Tilaajaverkon laitteet).

EPIC on kytkentäpiiri, jolla voidaan ristikytkeä jopa 128 kanavaa, joiden kaistanleveys on 16, 32 tai 64 kbit/s. EPIC voidaan ohjelmoida suorittamaan kytkentä PCM-signaalin ja konfiguroitavan liitäntäsignaalin välillä. Molemmat liitännät koostuvat neljästä kaksisuuntaisesta (*duplex*) portista. [Väärä 1993a, 37...39]

Tilaajan liitäntäyksikön hallinta (Control SUB)

MCU hoitaa tilaajan liitäntäyksikön signaloinnin. Lähetyssuuntainen signalointi tulee puhekanavilta MCU:lle sisäisen väylän kautta. Vastaanottavassa suunnassa signalointi tulee aikavälissä 16. Työnjako varsinaisen signalointiprosessin ja tilaajan liitäntäyksikön hallintaprosessin välillä on sellainen, että signalointi toimii pelkästään signaloinnin suorittavana tilakoneena. Tilaajan liitäntäyksikön hallintaprosessi sen sijaan hoitaa kaiken kommunikaation ulkomaailmaan päin, eli se myös välittää väliaikaiset ja pysyvät asetukset tilaajan liitäntäyksiköille ja raportoi niissä ilmenevistä vioista. Asetuksia voidaan luonnollisestikin muuttaa vain silloin, kun tilaajajohdolla ei ole puhelua.

Kaiken muun toiminnan lisäksi tilaajan liitäntäyksikön hallintaprosessi hoitaa soittogeneraattoria. Soittojännite ei nimittäin joissakin tapauksissa riitä kaikkien puhelimien yhtäaikaiseen soittamiseen. Tällöin soitot lomitetaan ohjelmallisesti niin, että toisen puhelimen soitossa esiintyvän tauon aikana toinen puhelin soi. [Väärä 1993a, s. 39...42]

Multiplekserin hallinta (Control MUX)

Multiplekserin hallintaprosessi huolehtii MUX-piirin parametrisoinnista sekä pysyvien että väliaikaisten asetusten edellyttämällä tavalla. Tällaisia asetuksia ovat mm. synkronointisignaalin valinta, synkronoinnin poistaminen vikatapauksissa, CRC:n käyttö, CRC:n polariteetti, signalointimoodi, 64k:n signaalin valinta ja 2M:n signaalin kontrollointi. [Väärä 1993a, s. 26...37]

Dynacardien hallinta (Handle Dynacards)

Dynacardien hallintaprosessi sisältää joitakin apufunktioita Dynacardien hallintaa varten. Dynacardit mm. ylläpitävät itsenäisesti omia vikataulukoitaan. On kuitenkin suotavaa, että myös MCU on selvillä Dynacardeissa esiintyneistä vioista. Tästä syystä Dynacardien hallintaprosessi välittää mm. nämä tiedot Dynacardeilta MCU:lle.

Dynacardit tallettavat muistiin paitsi omat parametrinsa myös viereisen Dynacardin parametrit. Näin syntyy eräänlainen ketju, jossa kaikkien korttien parametrit ovat talletettuina kahteen kertaan. Taustaprosessissa tarkistetaan jatkuvasti, täsmäävätkö naapurikortin parametrit talletettuihin parametreihin. Mikäli havaitaan viallista dataa, äänestetään oikeista parametreista.

Jos toiminnassa olevan Dynacardin tilalle vaihdetaan toinen Dynacard-kortti, tietää tämä uusi kortti olevansa uusi, sillä jokaisella Dynacard-kortilla on oma, tuotantovaiheessa annettu tunnistenumero. Tästä syystä konfigurointi voi tapahtua automaattisesti naapurikorttiin talletettujen toimintaparametrien perusteella.

ACM2:ssa Dynacardeja voidaan käyttää sekaisin ACM-SUB-yksiköiden kanssa, joten em. kaltaisen ketjun ylläpitäminen on mahdotonta. Tästä syystä on päädytty siihen, että MCU (eli Dynacardien hallintaprosessi) ylläpitää tiedostoa kaikkien Dynacardien parametreista.

Dynacardien välinen kommunikointi tapahtuu aina isäntä/renki-tyyppisesti (*in master/slave mode*). Isäntä lähettää komennon komentoväylälle, josta renki poimii sen ja vastaa vaaditulla tavalla. Isäntä voi myös käskeä kaikkia renkejä yhtäaikaa, tai antaa komennon, johon ei vaadita vastausta. Isännän tehtäviin kuuluu huolehtia mm. initialisoinnista laitteiston resetin jälkeen ja vikataulukoiden hallinnasta. On luonnollista, että ACM2:ssa MCU toimii aina isäntänä. [Jyllilä, 1993, s. 6...11]

Tiedostojärjestelmä (File System)

Tiedostojärjestelmää käytetään ensisijaisesti signalointitaulukoiden syöttöön ja hakuun. Järjestelmä on kuitenkin yleiskäyttöinen, eli muidenkin tiedostojen käsittely on mahdollista. Ensimmäisessä vaiheessa tiedostojärjestelmä toimii E^2PROM:illa, mutta tulevaisuudessa siirryttäneen flash-muistin käyttöön.

Käyttöliittymän hallinta (Handle User Command)

Käyttöliittymän hallinta huolehtii nimensä mukaisesti käyttöliittymästä. Se koostuu lukuisista erillisistä dynaamisista prosesseista, jotka käyttäjän antamat syötteet herättävät. Tämän jälkeen etsitään tietokannasta vastaus käyttäjän esittämiin kysymyksiin (esim. laitteiston id-numero, *timeoutin* pituus) tai muutetaan toimintaparametreja käyttäjän antamalla tavalla (esim. vaihdetaan ensisijaisesti käytettävää kelloa). Lopuksi dynaaminen prosessi palauttaa vastauksen käyttäjälle ja lopettaa toimintansa. Dynaamisten prosessien osuus kaikista prosesseista on huomattava: laitteiston koko koodimäärästä on noin puolet käyttöliittymäriippuvaista koodia.

Kello (Clock)

ACM2 sisältää reaaliaikaisen kellon mm. tilaajajohtojen testauksen ajoitustarpeita varten, koska käyttäjän tulee voida määritellä esim. tietty kellonaika, jolloin testi suoritetaan. Reaaliaikakellon päivitystaajuus on yksi sekunti, ja päivitys tapahtuu sanomapohjaisesti.

Vikatietojen hallinta (Handle Faults)

Eräs vikatietokantaprosessin tehtävä on päivittää ns. vahtikoiraa (*watchdog*). Vahtikoiran tarkoituksena on taata laitteiston jatkuva toiminta. Käytännössä vahtikoirana toimii yksinkertainen laskuri, jota täytyy päivittää ohjelmallisesti puolen sekunnin välein. Päivitykseen käytetty sanoma kierrätetään riittävän monen kriittisesti laitteiston toimintaan vaikuttavan ohjelmistoprosessin kautta vahtikoiralle. Mikäli päivitystä ei tehdä, aiheuttaa vahtikoira laitteiston resetoinnin, mikä johtaa ohjelmiston alustukseen.

Vikatietokannassa säilytetään nimensä mukaisesti tietoja sekä laitteiston ulkopuolisista että sisäisistä vioista. Vikatietokantaprosessi myös huolehtii mahdollisista korjaavista toimenpiteistä vian poistamiseksi. Viat voidaan jakaa kuuteen eri luokkaan: kaukopään viat, konfigurointivirheet (esim. puuttuva yksikkö), MUX2-piirin suoraan havaitsemat viat (esim. 2M:n signaali puuttuu), ON/OFF-tyyppiset viat (esim. jännitelähde ei toimi), tilaajan liitäntäyksiköiden viat, sekä laskennalliset viat (esim. tulevan 2M:n signaalin bittivirhesuhde on yli 10^3). Käyttäjä voi itse vaikuttaa joidenkin vikojen seuraamuksiin.

Käyttäjälle vioista tiedotetaan sytyttämällä kortilla olevia ledejä (punainen, keltainen ja vihreä), jotka toimivat liikennevalojen tavoin (punainen ilmaisee vakavaa, laitteen toiminnan estävää vikaa; keltainen pakotettua ohjausta tai pienempää vikaa esim. laitteen vastaanottamassa signaalissa; vihreä palaa kun huoltopääte tai käytönhallintatietokone on yhteydessä laitteeseen). Ledien lisäksi käyttäjä saa tietoa laittesta myös PA- ja RACK-hälytysten perusteella. PA-hälytysten merkitys on käyttäjän ohjelmoitavissa. PA-hälytyksinä toimii kaksi Earth & Mark -tyyppistä hälytystä. RACK-hälytyksillä (A, B ja D) ohjataan sen kehikon kolmea lamppua (punainen, valkoinen ja keltainen), jossa ACM2-kortti sijaitsee. RACK-hälytys annetaan, mikäli jokin korttikohtainen hälytys on ollut pitkään päällä, eikä siihen ole millään tavalla reagoitu. Punainen lamppu palaa vian ollessa vakava, valkoinen valo kun vika ei ole vakava ja keltainen indikoi pakotettua hälytyksen poistoa (käyttäjän kuittaamaa hälytystä).

Näiden toimintojen lisäksi vikatietokantaprosessi huolehtii joistakin konfigurointiin liittyvistä toiminnoista, koska vikatietokannan täytyy pystyä yhdistämään tietty vika ja sen sijainti toisiinsa. [Väärä 1993a, 18...25]

Signalointi (Handle Signaling)

Signalointiprosessi hoitaa tilaajan liitäntäyksikön (SUB) tarvitseman PCM-liitännän ja puhekanavaliitännän välisen signaloinnin valitun signalointitaulukon perusteella. Prosessointitehosta signalointiprosessi vie noin kolmanneksen.

A-, b-, c- ja d-bittien jatkuva seuraaminen (kts. kuva 2) kuormittaa signalointiprosessina toimivaa tilakonetta niin paljon, että sille ei jää aikaa kommunikoida sanomapohjaisesti muiden prosessien kanssa. Tästä syystä tilaajayksikön hallintaprosessi toimii signaloinnin informaatioväylänä muihin prosesseihin. Signalointiprosessin ja tilaajayksikön hallintaprosessi kommunikoivat keskenään yhteisen muistialueen avulla. Muistialue on suojattu siten, että muut prosessit eivät voi käsitellä sitä. Yhtäaikaiset prosessien luku- ja kirjoitusoperaatiot on myös estetty informaation muuntumisen estämiseksi.

Tilaajan liitäntäyksikön hallintaprosessi toimii myös signalointiprosessin varmistusprosessina. Mikäli signalointiprosessin toiminta jostain syystä estyy, estyy samalla koko laitteen toiminta. Tästä syystä signalointiprosessin toimintaa tarkkaillaan jatkuvasti, ja tarvittaessa se herätetään ohjelmallisesti uudestaan henkiin pysähdyksen tapahduttua. Signalointiprosessi käynnistetään myös ensimmäisenä alustuksessa.

Tilastotiedot (Handle Statistics)

Tilastointiprosessi kerää CCITT:n suosituksen G.826 mukaisia tilastotietoja (esim. käytettävyysaika, virheelliset sekunnit, vakavasti virheelliset sekunnit) sekä muutamia muitakin suosituksessa mainitsemattomia tilastoja (esim. kehyslukitussanassa havaittujen virheiden määrä, kadotettujen kehyslukitussanojen määrä) MUX2:lta saatavien rekisteritietojen perusteella. [CCITT G.826, s. 1...15] Tämän lisäksi tilastoprosessi antaa CCITT:n suosituksen G.732 mukaisia hälytyksiä (BER > 10^3) ja joitakin ylimääräisiä hälytyksiä (esim. BER > 10^6). [CCITT G.732, s. 375...381]

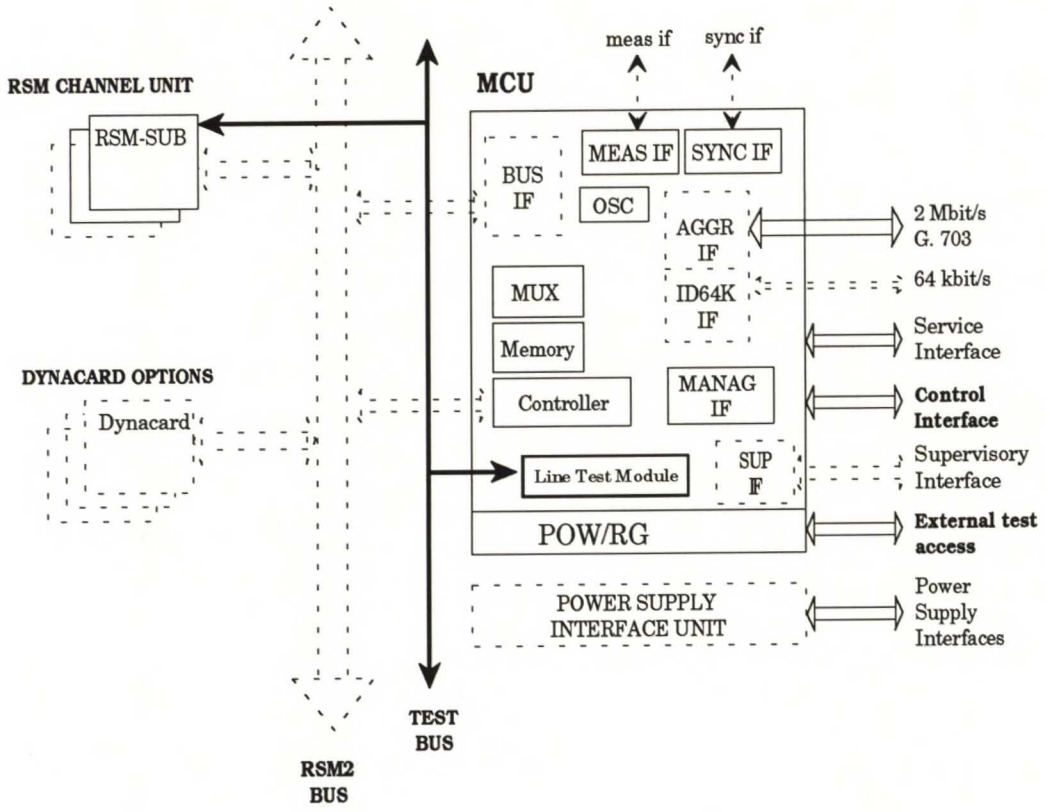
Linjatestausmoduuli (LT)

Linjatestausmoduuli sisältää tilaajajohtojen testauksen tarvitsemat funktiot. Moduulia käsitellään lähemmin luvuissa kolme ja neljä.

Varsinainen toteutus eroaa yllä olevasta kuvauksesta siten, että osa toiminnoista on toteutettu käyttäen apuna Q1_302-yhteisohjelmistoa. Siihen on kerätty joukko prosesseja, jotka ovat samantyyppisiä kaikissa MC68302-pohjaisissa reaaliaikasovelluksissa. Prosessit sisältävät mm. Q1-protokollaan liittyviä toimintoja [CCITT, Q.771], E^2PROM:in käsittelyfunktioita, vikatietokannan ylläpitoa helpottavia toimintoja sekä joitakin pienempiä yleiskäyttöisiä toimintoja (esim. ohjelmiston kuormituksen mittaus).

2.3 Linjatestausmoduulin toiminnalliset vaatimukset

Linjatestausyksikön (LTU:n) ongelmaksi on noussut paitsi sen viemä tila, myös sen sisältämien komponenttien kalleus (prosessori ja suuri määrä releitä). Tästä syystä linjatestaus haluttiin siirtää samalle kortille MCU:n kanssa, jolloin voitiin myös siirtyä yhden, tehokkaamman prosessorin käyttöön. LTU:n toiminnallisuus ja mittaukset haluttiin kuitenkin säilyttää vastaavanlaisina.



Kuva 10. Linjatestaus laitteistonäkökulmasta

Kuvassa kymmenen on esitetty linjatestaus suhteessa ACM2-laitteistoon sekä kuvattu laitteistorajapinnat. Kuten kuvasta näkyy, ei Dynacardien avulla liitettyjen tilaajajohtojen testausta ole tarkoitus vielä toteuttaa projektin tässä vaiheessa. Samoin keskuspään testaus sekä piirikortin testaus (*line circuit testing*) jätetään toteuttamatta. Piirustusteknisistä syistä kuvassa oikealla esiintyvät rajapinnat eivät liity oikeisiin lohkoihin. Linjatestausmodulin rajapinnat on tästä syystä esitetty **lihavoidulla** tekstillä.

Linjatestausmoduulin laitteistorajapinnat käsittävät ulkoisen testiliitännän (*external test access*) ja kontrollirajapinnan (*control interface*). Ulkoinen testiliitäntä koostuu signaaleista TLAEXT (*Test Line A*), TLBEXT (*Test Line B*), TIAEXT (*Test Interface A*) ja TIBEXT (*Test Interface B*). Näiden neljän signaalin avulla voidaan kytkeä ulkoinen testilaite tilaajan analogiseen kaksijohdinrajapintaan linjatestausmoduulin kautta. Kontrollirajapinta (*control interface*) sisältää neljä analogista jännitetuloa (mm. lämpötilan mittausta varten), jännitelähteen jännitetason seuraamiseen tarkoitetun liitännän, soittogeneraattorin jännitetason seuraamiseen tarkoitetun liitännän sekä joitakin yleisiä ohjelmoitavia tuloja ja lähtöjä. Kontrollitulot on tarkoitettu laitteen toimintaympäristössä tapahtuvien muutosten havaitsemiseen. Mikäli käyttäjä niin haluaa, voidaan tuloille määrätä raja-arvot, joiden ylittyessä tai alittuessa annetaan hälytys. Lähdöillä taas voidaan välittää tietoa muille laitteille. Lähtöjen arvot ovat sekä ohjelmallisesti että käyttäjän muutettavissa.

Linjatestaustoiminnot toteutetaan erillisenä lisämoduulina (*add-on-module*). Taustapaneelissa on kaksi kaksijohtimista testiväylää, joista toinen on tarkoitettu tilaajan yhdysjohdon testaukseen (*test out*) ja toinen koko puhetien testaamiseen (*test in*).

Toiminnallisuus

Linjatestausmoduuli tulee olemaan toiminnoiltaan samanlainen kuin LTU (katso kuva 6, sivu 15). Ennen ensimmäistä mittausta (so. kytkettäessä laitteeseen virta) testataan laitteiston kunto. Tämän jälkeen mitataan vieraat AC- ja DC-jännitekomponentit a- ja b-johdoissa. Näistä voidaan havaita myös ylijännitteet, mikäli sellaisia esiintyy. Periaatteessa muut mittaukset keskeytetään, mikäli johdoissa on ylijännitteitä. Käytännössä tällaista tilannetta tuskin koskaan esiintyy, sillä SUB-kortti kestää ylijännitteitä varsin huonosti. Asiakas havaitsee ylijännitteet pikemminkin tilaajakortin rikkoontumisena kuin LTM:n tekemänä mittauksena. AC- ja DC-komponenttien mittausalue on 0...Vbat (Vbat, paristojännite -48V). AC-jännitteen mittaustarkkuus on ± 10%, ja DC-jännitteen ± 5%. Mittausalue DC-jännitteille on riittävä, mutta mahdollisia positiivisia AC-jännitehuippuja joudutaan arvioimaan aaltomuodon ja negatiivisten jännitenäytteiden perusteella.

Seuraavaksi mitataan vuotoresistanssit (a-johdosta maahan, b-johdosta maahan, a-johdosta b-johtoon, a-johdosta paristojännitteeseen ja b-johdosta paristojännitteeseen). Mittausalue on 0...1 MΩ tarkkuudella ± 10%. Pienillä resistanssiarvoilla (< 10 kΩ) mittaukset keskeytetään, sillä tällöin epäillään kuulokkeen nostoa.

Seuraavana on vuorossa kapasitanssimittaus. Tämä suoritetaan pelkkänä ON/OFF-tyyppisenä mittauksena a- ja b-johtojen väliltä. Käyttäjä voi määritellä kapasitanssille raja-arvon väliltä 0.5...2 µF. LTM pystyy havaitsemaan kapasitanssin, jos se on suurempi kuin annettu raja-arvo tarkkuudella ± 20%. Joissakin maissa (esim. Iso-Britanniassa, mutta ei esim. Suomessa) puhelinpistorasioissa on a- ja b-johdon välillä tietyn suuruinen kapasitanssi. Tällä testillä voidaan siis havaita kyseinen kapasitanssi. Mikäli kapasitanssia ei havaita, voidaan epäillä vikaa tilaajajohdossa tai asiakkaan puhelinkoneessa.

Lisäksi voidaan mitata tilaajaliittymän signalointijännite (*subscriber signalling voltage*, 40...70 Vdc) ja tilaajapiirin liitäntäjännite (*subscriber interface circuit voltage*, 0...5 Dac).

AC- ja DC-komponenttien, vuotoresistanssien, kapasitanssimittauksen ja signalointi- sekä liitäntäjänniteen mittaaminen yhdellä asiakasjohdolla kestää alle viisi sekuntia. Käyttäjä voi määrätä kullekin mittaukselle erikseen raja-arvoja, joiden ylittymisestä tai alittumisesta kirjataan tieto varsinaisten mittaustulosten yhteyteen. Raja-arvon ylityksestä tai alituksesta voidaan myös antaa hälytys ja sytyttää tilaajakortin keltainen ledi, mikäli käyttäjä niin haluaa. Käyttäjä voi vapaasti määrätä suoritettavat mittaukset (*enable/disable*).

Käyttäjä valitsee myös tehdäänkö mittaukset välittöminä vai periodisina. Välittömät testit suoritetaan heti. Periodisessa testissä valitaan testin ensimmäinen suoritusajankohta sekä toistoväli. Toistoväli voi lyhimmillään olla kolme minuuttia ja pisimmillään useita vuorokausia. Välitön testi voidaan tehdä joko yksittäiselle johdolle tai kaikille johdoille yhtä aikaa (so. yhdellä komennolla). Kussakin testissä voidaan erikseen valita tehtävät mittaukset (esim. mitataan vain resistanssit tai vain kapasitanssit). Periodinen testi tehdään aina kaikille johdoille. Kuitenkin kukin johto voidaan erikseen merkitä testauskieltoon (johto- eli kanavakohtainen testauskielto). Tällöin kaikkien yhdysjohtojen testitapauksissa testataan vain ne yhteydet, jotka eivät ole testauskiellossa.

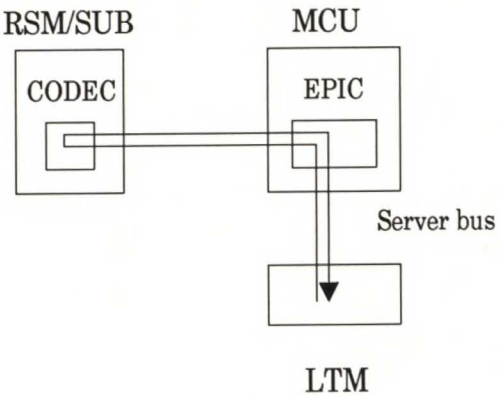
Kukin tilaajajohdon testi suoritetaan vain mikäli yhdysjohto on IDLE-tilassa. Puhelun aikana testejä ei tehdä. Kunkin testin välillä tarkastetaan, ettei tilaaja ole nostanut puhelimensa kuuloketta. Mikäli kuuloke on nostettu, keskeytetään testaus välittömästi. Tilaajalle soitetaan puhelun valintaääntä, ja johto merkitään "ei testatuksi". Käyttäjä saa kuitenkin halutessaan näkyviin edellisten mittausten tulokset, jos sellaisia on olemassa.

Tilaajajohtojen testauksesta ei tällä hetkellä ole olemassa voimassaolevia standardeja. CCITT on valmistelemassa suositusta Q.542, jossa tilaajajohtojen testausta käsitellään keskuksen kannalta. Suosituksessa keskitytään lähinnä mitattaviin parametreihin. Mittausalueista ja -tarkkuuksista siinä ei ole mainintaa. [CCITT, Q.542] Eri maiden teleoperaattorit ovat sen sijaan yhteistyössä CCITT:n kanssa esittäneet omat ehdotuksensa suositukseksi digitaaliseen keskukseen liitettyjen analogisten tilaajajohtojen ylläpidosta. Ehdotus sisältää yksityiskohtaisemmat ohjeet mitattavista parametreista kuin CCITT:n valmisteilla oleva suositus. Myös mittausalueet ja tarkkuudet on määritelty. [CCITT, 1983] Ehdotus on otettu huomioon linjatestausmoduulia suunniteltaessa, mutta asiakkaan asettamat vaatimukset ovat silti olleet etusijalla.

Kun mittausrele kytketään, aiheuttaa se mittaukseen häiriöitä, jotka näkyvät vaimenevina värähtelyinä. LTU:ssa odotettiin tästä syystä kussakin yksittäisessä mittaustapauksessa aina tietyn aikavakion τ verran, jossa τ on niin pitkä, että stabiloituminen varmasti tapahtuu. Linjatestausmoduulissa tätä mittausalgoritmia on sen sijaan tarkoitus muuttaa siten, että saaduista mittausarvoista päätellään, koska stabiloituminen on tapahtunut (tällöin arvot jäävät tietylle vaihteluvälille Δx). Näin mittaus nopeutuu hieman. Tosin tässäkin algoritmissa tarvitaan aikavakio τ varmistamaan se, että tulos todella saadaan järkevän ajan puitteissa, sillä ulkopuolelta mittaukseen kytkeytyvä kohina saattaa pahimmassa tapauksessa estää mittausarvojen stabiloitumisen.

Puhetietä ja laitteiston tilaa voi tutkia myös muodostamalla erilaisia signaalin takaisinkytkentöjä eli silmukoita (*loopback*). Käyttäjä valitsee kanavan, jolle silmukkatesti tehdään. Linjatestausmoduulista lähetetään signaalia MCU:n EPIC:ille, josta se viedään ACM-SUB-yksikössä sijaitsevalle CODEC:ille (katso kuva 11). Varsinainen silmukka tehdään CODEC:issa. Digitaalisessa silmukassa signaali heijastetaan suoraan takaisin. Analogisessa silmukassa digitaalinen signaali muutetaan ensin analogiseksi, jonka jälkeen se kytketään toiseen muuntimeen ja muutetaan taas digitaaliseksi. Analogisen ja digitaalisen silmukan lisäksi voidaan tutkia soiton aikana (soittoreleen yhdistäessä tilaajajohdot tilaaja-

yksikköön) heijastuvaa signaalia. Koska tilaajajohdolla esiintyvä impedanssi on tuntematon, saadaan heijastetun signaalin analyysistä vain karkeita arvioita tilaajayksikön komponenttien tilasta ja kunnosta.



Kuva 11. Silmukkatestaus

Kun käyttäjä haluaa käyttää jotain ulkoista mittalaitetta, täytyy hänen ensin määritellä, mitä kanavaa mitataan. Tämän jälkeen linjatestausmoduuli tarkistaa, onko kyseinen johto käytössä vai ei. Mikäli johto on vapaa, varataan se ulkoisen mittalaitteen testattavaksi. Johto on varattuna, kunnes käyttäjä ilmoittaa testauksen päättyneen tai käyttäjän määrittelemä aikavakio kuluu. Aikavakio on oletusarvoisesti 1000 s ja enintään 65000 s. Käyttäjä voi myös halutessaan estää aikavakion käytön, mutta tällöin yhteyden vapautuminen jää pelkästään käyttäjän vastuulle. Ulkoisen testilaitteen käytöstä voidaan haluttaessa ilmoittaa keltaisella ledillä.

Tilaajajohtoja koskevien mittausten lisäksi linjatestausmoduuli sisältää joitakin valvontamittauksia. Näitä ovat paristojännitteen mittaus (20...70Vdc), soittojännitegeneraattorin jännitteen mittaus (0...5Vdc), sekä neljän ulkoisen jännitteen mittaus, joiden mittausalueet ovat:

- input 1: 0...70 Vdc tarkkuudella ± 5%
- input 2: -5...+5 Vdc tarkkuudella ± 5%
- input 3: 0...10 Vdc tarkkuudella ± 5%
- input 4: -5...+5 Vdc tarkkuudella ± 5%.

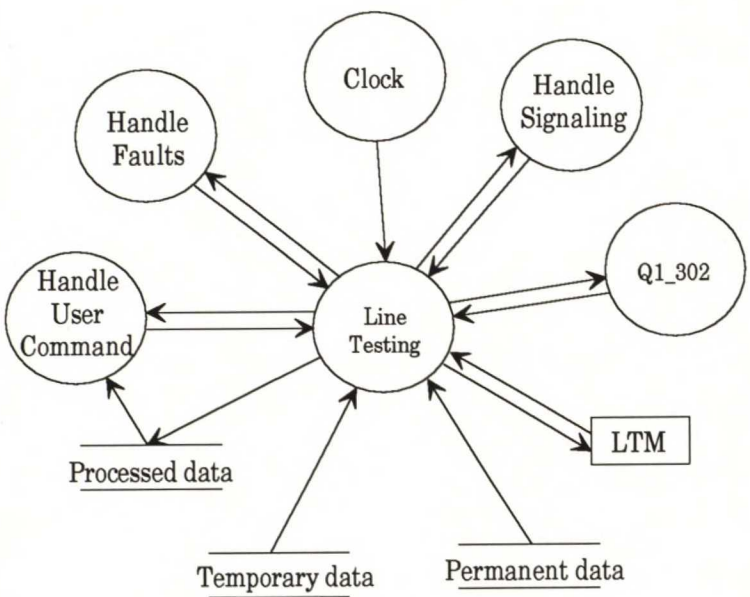
Ulkoiset jännitetulot soveltuvat esim. lämpötila-anturin kytkentään, jolloin saadaan tietoa laitteen toimintaympäristön lämpötilasta.

Lisäksi valvontaominaisuuksiin kuuluu kaksi ohjelmoitavaa tuloa ja kaksi ohjelmoitavaa lähtöä (nastoja, joiden arvon käyttäjä voi ohjelmallisesti määrätä joko ykköseksi tai nollaksi). Tämän lisäksi käyttäjä voi testata AD- ja DA-muuntimien toiminnan DA/AD-silmukkatestillä, ja kontrollitulojen sekä -lähtöjen toiminnan niiden silmukkatestillä.

3. Arkkitehtuuri

Tässä luvussa käsitellään linjatestausohjelmiston arkkitehtuuria sekä rajapintoja muuhun ohjelmistoon ja käyttäjään päin. Lisäksi pohditaan niitä vaatimuksia, joita verkonhallintakonsepti testaukselle asettaa.

3.1 Ohjelmistojen välinen rajapinta



Kuva 12. Linjatestausohjelmiston rajapinnat

Kuvassa 12 on esitetty linjatestausohjelmiston rajapinnat ACM2-perusohjelmistoon päin. Vikatietojen hallinta (*Handle Faults*), kello (*Clock*) ja signalointi (*Handle Signaling*) sisältävät kukin yhden ainoan prosessin. Käyttöliittymän hallinta (*Handle User Command*), Q1_302-yhteisohjelmisto ja linjatestausohjelmisto (*Line Testing*) sisältävät sen sijaan useita prosesseja. Ohjelman käsittelemällä datalla (*Processed data*) kuvataan sitä ohjelmamuistissa sijaitsevaa tietokantaa, jonne testaustulokset talletetaan. Kuvassa esiintyvällä notaatiolla on tarkoitettu sitä, että käyttäjän kysellessä mittaustuloksia niiden lisäprosessointia ei enää tarvita. Väliaikainen tietokanta (*Temporary data*) kuvaa sitä RAM-muistissa sijaitsevaa tietokantaa, johon usein muuttuvat tiedot (esim. tiedot yhteyden tilasta) talletetaan. Pysyvä tietokanta (*Permanent data*) taas sisältää nimensä mukaisesti pysyvämpää, E^2PROM:iin talletettua tietoa (esim.

mittaustuloksista annettavien hälytysten raja-arvot). Tietokannat on kuvattu tarkemmin luvussa 3.2.

Käyttäjärajapinnan kautta kulkevat käyttäjän antamat linjatestausta koskevat komennot sekä niihin annetut vastaukset. Käyttäjä voi pyytää testaamaan joko yhden yhteyden tai kaikki yhteydet yhtäaikaa. Lisäksi hän voi määrätä suoritettavaksi periodisia testejä (so. testejä, joiden suorittamisen alkamisajankohta ja suorituskertojen välinen aikajakso on määrätty). Tuotantotestausta varten on lisäksi omat testit. Käyttäjä voi kysellä testien tuloksia, määrätä tilaajajohtoon silmukan tai määrätä suoritettavaksi testaus, jossa käytetään ulkoista testilaitetta. Kalibrointia, kontrollitulojen hälytysrajojen määritystä ja kontrollilähtöjen ohjausta varten on myös omat komentonsa. Jotkut käyttäjän antamat komennot eivät ohjaudu linjatestausohjelmistolle lainkaan, vaan hoituvat suorina E^2PROM:in luku- ja kirjoitusoperaatioina. Tällaisia ovat esimerkiksi tilaajajohdon asetus testauskieltoon tai hälytysrajojen muutokset.

Signalointirajapinnan kautta kulkevat pyynnöt tilaajajohtojen ja testiväylän välisen yhteyden muodostamisesta ja purkamisesta. Signalointiprosessilla on tieto kunkin johdon tilasta, joka voi olla varattu, vapaa tai ei-kytketty. Mikäli johto on vapaa, signalointiprosessi asettaa testireleet, tarkistaa kytkennän onnistumisen ja ilmoittaa linjatestausohjelmistolle, että kytkentä on suoritettu onnistuneesti. Kytkennän epäonnistuminen aiheuttaa vikatilanteen. Linjatestausohjelmisto on itse vastuussa siitä, että sellaisia yhteyksiä ei testata, joiden testauksen käyttäjä on kieltänyt. Kytkennän purkaminen tapahtuu vastaavasti.

Mikäli signalointiprosessi havaitsee, että testattavalle johdolle on tulossa puhelu, lähettää se linjatestausohjelmistolle komennon, joka käskee lopettaa välittömästi kaiken testaamisen. Saatuja osittaisia mittaustuloksia ei talleteta, ja johto merkitään osittain testatuksi johdon tilaa osoittavaan taulukkoon (väliaikainen tietokanta).

Ulkoista testilaitetta käytettäessä signalointiprosessilta tarkistetaan ensin testattavan johdon tila. Mikäli johto on vapaana, kytketään testattava johto ja ulkoinen testilaite testausväylälle. Tämän jälkeen käyttäjälle kerrotaan, että hän voi aloittaa testauksen. Testauskytkentä puretaan joko tietyn käyttäjän asettaman aikavakion ilmoittaman ajan kuluttua tai erityisestä käyttäjältä tulevasta purkupyynnöstä johtuen.

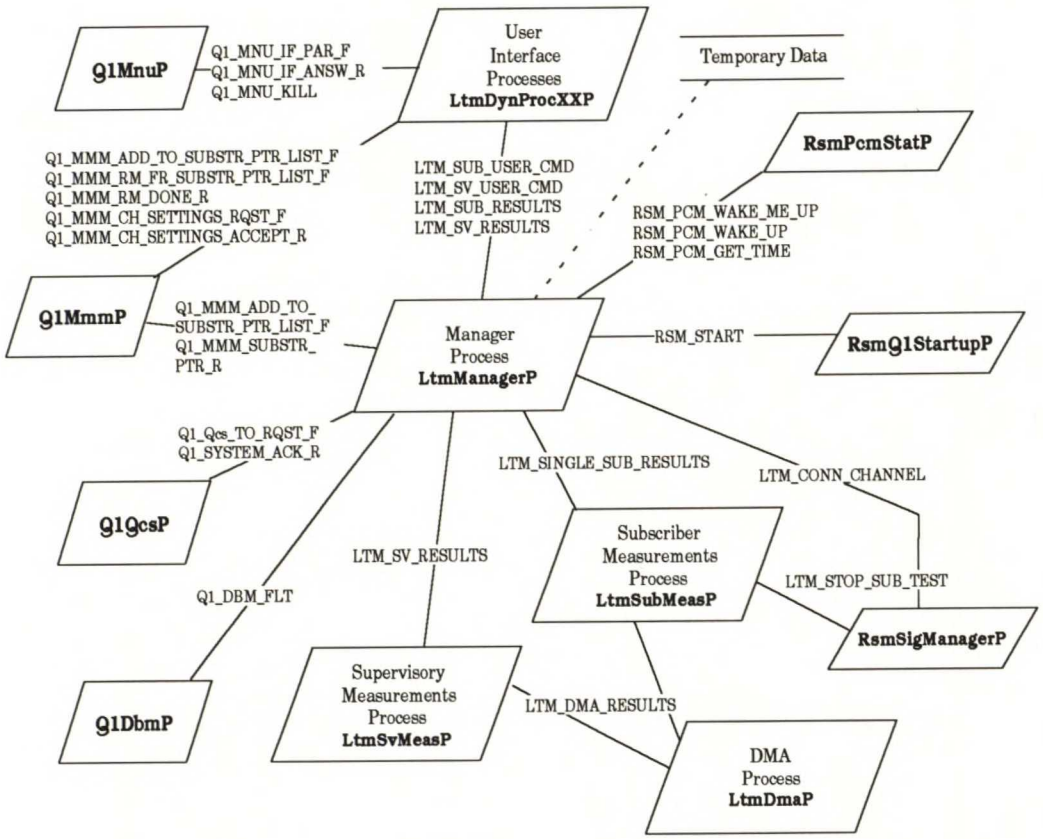
Kun käyttäjä antaa käyttöliittymän kautta komennon muodostaa tilaajakortissa takaisinkytkentä, lähetetään pyyntö ensin käyttöliittymän hallintaprosessilta signalointiprosessille. Signalointiprosessi lähettää tarvittavat ohjaustiedot EPIC- ja CODEC-piireille. Silmukan purku tapahtuu vastaavasti käyttäjän antaman komennon perusteella. Edellä mainittujen komentojen lisäksi myös käynnistyssanoma tulee signalointirajapinnan kautta.

Periodisissa testeissä linjatestausmoduuli lähettää reaaliaikakellolle sanoman, jossa se pyytää herätystä tiettyyn aikaan. Herätyksen se saa samoin sanomapohjaisesti. Herätyspyyntö- ja vastaussanomat sisältävät tunnistenumeron, jonka avulla yhteenkuuluvat herätyspyynnöt ja vastaukset ovat tunnistettavissa. Herätykset perutaan lähettämällä herätyspyyntö, jossa sekä herätysaika ja periodi on merkitty nolliksi.

Vikatietokantaa ylläpitävä prosessi informoi linjatestausohjelmistoa testauksen estävistä tilaajajohtojen vikatilanteista. Vastaavasti linjatestausohjelmisto ilmoittaa vikatietokannalle testauksessa havaituista vioista. Vikatietokantaa ylläpitävä prosessi huolehtii myös ledien sytyttämisestä ja sammuttamisesta sekä muista vikojen aiheuttamista seuraamuksista.

Q1_302-rajapinta tarjoaa joitakin E^2PROM:in käsittelyfunktioita ja Q1-rajapintaan liittyviä palveluja. [CCITT, Q.771] Lisäksi yksittäisissä mittauksissa käytetään Q1:n keskeytyspalveluita.

3.2 Linjatestausohjelmiston arkkitehtuuri



Kuva 13. Linjatestausohjelmiston arkkitehtuuri

Arkkitehtuurin suunnittelun lähtökohtana oli LTU:n ohjelmistoarkkitehtuuri, joka sisälsi erilliset prosessit signalointia, ledien käsittelyä, vikatilanteiden hallintaa, laitteisto-operaatioita (toteutettu erillisenä keskeytysprosessina), testien hallintaa, periodisia testejä, käyttöliittymän hallintaa ja mittausoperaatioita varten. Linjatestausmoduulin tapauksessa tarvittava arkkitehtuuri on kuitenkin

yksinkertaisempi, koska osa toiminnoista (signalointi, ledien käsittely ja vikatilanteiden hallinta) on hoidettu jo ACM2:n perusohjelmistossa. Kuvassa 13 on esitetty arkkitehtuuri, jollaiseen lopulta päädyttiin. Suunnikkaat kuvaavat prosesseja tai moduuleja. Laitteistoriippuvat asiat on eristetty omaan prosessiinsa (DMA-prosessi). Kuvassa prosessien välillä esiintyvät sanat kuvaavat prosessien välillä siirrettävää informaatiota, ts. ne ovat sanomien nimiä.

Testien käsittely (Line Test Manager)

Testien käsittelyprosessi hoitaa testien hallinnan yleisellä tasolla. Yksittäisen yhteyden välittömän testin tapauksessa se tarkistaa ensin onko linjan testaus yleensä sallittua. (Käyttäjä voi kieltää kunkin linjan testauksen erikseen. Johtoa ei testata ennen kuin kielto perutaan. Komento on hyödyllinen etenkin silloin, kun joku johdoista on varattu kokoaikaisesti jotain erityistarkoitusta, esimerkiksi datan siirtoa varten.) Tämän jälkeen se kysyy signalointiprosessilta voidaanko johto testata vai ei. Mikäli johto on idle-tilassa, signalointiprosessi vastaa myöntävästi ja kytkee tilaajajohdon testausväylälle ohjaamalla testirelettä. Muussa tapauksessa testien käsittelyprosessi saa vastaukseksi "tilaajajohto on varattu", "tilaajajohtoa ei ole kytketty" tai "kytkentä ei onnistunut". Tämän jälkeen testien käsittelyprosessi lähettää sanoman tilaajajohtojen mittausprosessille (*Subscriber measurements*), joka suorittaa varsinaisen mittauksen. Kun testi on suoritettu, lähettää tilaajajohtojen mittausprosessi testitulokset testien käsittelyprosessille, joka tallentaa ne sisäiseen tietokantaansa. Lopuksi se ohjaa signalointiprosessia purkamaan testikytkennän.

Testien käsittelyprosessi tarkastaa saamansa testitulokset. Käyttäjä voi halutessaan määrätä tietyt hälytysrajat. Mikäli mitatut arvot ylittävät tai alittavat nämä rajat, annetaan hälytys. Testien käsittelyprosessi huolehtii tästä lähettämällä sanoman ACM:n perusohjelmistoon kuuluvalle virheiden käsittelyprosessille, joka huolehtii tarvittavista lisätoimenpiteistä (mahdollisesta tilaajakortin keltaisen ledin sytyttämisestä ja virheiden tallettamisesta tietokantaan myöhempää tarkastelua varten).

Testattaessa kaikki johdot kerralla, testien käsittelyprosessi tutkii yksitellen onko käyttäjä kieltänyt johdon testauksen vai ei, ja lähettää testikomennon tilaajajohtojen testausprosessille. Jos joku johdoista on varattuna testihetkellä tai jos jonkun johdon testaus keskeytyy tulevan puhelun seurauksena, merkitään tämä johto "ei testatuksi". Johdon testausta yritetään myöhemmin vielä kaksi kertaa uudelleen. Ellei testaus tällöinkään onnistu, merkitään johdon tilaksi "testiä ei suoritettu".

Periodisen testin tapauksessa testien hallintaprosessi tallettaa testin suoritusta koskevat tiedot sisäiseen tietokantaansa. Tällaisia tietoja ovat suoritettavat testit, testien alkamisajankohta sekä se, kuinka usein testit suoritetaan. Periodinen testi suoritetaan muutoin samalla tavalla kuin kaikkien tilaajajohtojen testi paitsi että testiä viivästytetään kellon avulla tietty aika. Testien käsittelyprosessi lähettää ACM2-perusohjelmistoon kuuluvalle reaaliaikakellolle herätyspyynnön. Kello vastaa herätyssanomalla, kun pyynnössä määritelty aika on kulunut.

Testien käsittelyprosessi huolehtii ulkoista testilaitetta käytettäessä kytkentäpyynnön lähettämisestä signalointiprosessille. Linja on lepotilassaan aina kytkettynä ulostuloon, joten muita toimenpiteitä ei tarvita. Lopuksi käyttäjälle ilmoitetaan vielä, että ulkoisen testilaitteen käyttö on nyt mahdollista.

Silmukkatestin tapauksessa testien hallintaprosessi huolehtii samoin kommunikaatiosta signalointiprosessiin päin. Silmukan muodostamispyyntö lähetetään tilaajajohtojen testausprosessille jatkotoimenpiteitä varten.

Tilaajajohtojen mittausprosessi (Subscriber Measurements)

Tilaajajohtojen mittausprosessi huolehtii varsinaisten mittausten suorittamisesta. Mittaukset on kuvattu yksityiskohtaisesti luvussa 2.3 (toiminnallisuus). Mittaukset koostuvat viidestä osasta: ensin mitataan jännitteet, resistanssit sekä kapasitanssi. Lopuksi mitataan signalointijännite ja tilaajapiirin liitäntäjännite. Jännitemittaukset koostuvat ylijännitteiden ja vieraiden tasa- ja vaihtosähkökomponenttien mittaamisesta. Ylijännitetesti tehdään ON/OFF-tyyppisenä testinä. Jännitteiden mittausalue on 0...-48 V. Käytännössä mittausalue on riittävä, sillä tilaajajohtojen tasasähkökomponentit osuvat lähes poikkeuksetta tälle välille. Vaihtojännitteet sen sijaan saattavat osittain kohota 0 V:n yläpuolelle. Tällöin positiivinen huippuarvo ja signaalin aaltomuoto voidaan arvioida mitattujen negatiivisten jännitearvojen perusteella.

Resistanssimittauksessa mitataan vuotoresistanssit maatason ja a-johtimen sekä maatason ja b-johtimen välillä. Tämän jälkeen mitataan resistanssit a- ja b-johtimista jännitelähteeseen. Lopuksi mitataan a- ja b-johtimien välinen resistanssi.

Kapasitanssimittaus tehdään ON/OFF-tyyppisenä mittauksena.

Mittausalgoritmissa on erityisesti kiinnitetty huomiota nopeaan off-hook -tilan havaitsemiseen.

Silmukkatestauksessa tilaajajohtojen mittausprosessi ohjaa silmukalle generoitavaa signaalia ja vertaa takaisinsaatua signaalia lähetettyyn.

Valvontaprosessi (Supervisory Measurements)

Valvontaprosessi sisältää joukon jännitteiden mittauksia. Mittaukset on jo aiemmin kuvattu luvussa 2.3. Mittauksiin kuuluvat paristojännitteen mittaus, soittojännitteen mittaus ja neljän ulkoisen jännitetulon mittaus (käyttäjä voi liittää jännitetuloihin haluamansa anturit esim. ulkokaapin lämpötilan mittausta varten). Valvontaprosessissa on myös kaksi ohjelmoitavaa tuloa ja kaksi ohjelmoitavaa lähtöä. Ohjelmoitavilla tuloilla ja lähdöillä tarkoitetaan sitä, että käyttäjä voi asettaa niiden arvon ohjelmallisesti ON- tai OFF-tilaan. Valvontaprosessiin kuuluvat lisäksi digitaali/analogimuuntimen ja analogi/digitaalimuuntimen silmukkatestaus sekä kontrollitulojen ja -lähtöjen silmukkatestaus. ADC/DAC-testaus tehdään generoimalla muistipiiriä käyttäen analoginen jännite ja syöttämällä se AD-

muuntimeen. AD-muuntimesta jännite ohjataan DA-muuntimeen, jonka jälkeen saadusta analogisesta jännitteestä mitattuja näytteitä verrataan alkuperäisiin arvoihin. Tulokseksi annetaan karkea arvio siitä, ovatko muuntimet toimintakykyisiä vai ei. Periaatteessa näytteistä voitaisiin arvioida vian sattuessa jopa kunkin yksittäisen kvantisointiportaan oikeellisuus, mutta tällaiseen laskentaan ei ole nähty tarvetta. Kontrollitulojen ja lähtöjen silmukkatestauksessa lähdöt kytketään tuloihin. Tulot maadoitetaan. Lähdöt kytketään päälle (ON-tila).

Valvontaprosessin suorittamat mittaukset voidaan tehdä joko periodisina tai välittöminä linjatestausmittausten tavoin. Mittaukset voidaan ohjelmallisesti estää, ja testausta varten voidaan määrätä näytteiden lukumäärä. Mittauksille voidaan myös määritellä ylä- ja alahälytysrajat, sekä annetaanko hälytystä vai ei.

Käyttöliittymä (User Interface)

Käyttöliittymä on luonteeltaan hierarkinen puu. Puun solmuja kutsutaan menuiksi. Käyttöliittymä on toteutettu osittain Q1_302-yhteisohjelmistoon kuuluvalla Q1Mnu-prosessilla, ja osittain itse kirjoitetuilla dynaamisilla prosesseilla. Käyttäjälle näkyvät tekstit, ja itse menupuun rakenne on talletettu header-tiedostoihin. Kustakin menuvalinnasta on talletettu sallittujen syötteiden pienin ja suurin arvo (tai merkkejä syötettäessä pienin ja suurin kappalemäärä), syötettävän parametrin tyyppi, tieto siitä välitetäänkö parametri kutsuttavalle prosessille vai ei, mahdollinen menuhaaran suojaus (suojaamaton/suojattu), seuraavan solmun tyyppi (menu, dynaaminen menu, teksti tai prosessi), mahdollisesti kutsuttavan dynaamisen prosessin nimi ja prioriteetti (priorisoitu/ei priorisoitu), oletusarvoinen vastausteksti sekä osoitin seuraavaan solmuun.

Käyttäjän vaeltaessa pitkin puurakennetta tehdyt valinnat tallettuvat parametrilistaan. Saavuttaessa puun latvaan kutsutaan valintojen mukaan määräytyvää dynaamista prosessia, jolle kerätty parametrilista myös välitetään. Q1Mnu-prosessi poimii header-tiedostoista tarvittavat menut ja välittää ne Q1-protokollaa käyttäen kommunikointiväylälle, sekä huolehtii dynaamisen prosessin luomisesta. Dynaaminen prosessi suorittaa käyttäjän määrittelemät operaatiot (tallettaa tietoja E^2PROM:iin tai lukee sinne talletettuja tietoja, ohjaa staattisten prosessien toimintaa lähettämällä niille sanomia, lukee prosessien keräämää tietoa tms.), ja lopettaa tämän jälkeen toimintansa. Liitteessä viisi on esitetty hierarkinen menurakenne, ja liitteessä kuusi menun 7.3 (*Immediate SV measurements*) toteuttava header-tiedosto ja menun 7.0 (*SV measurement results*) toteuttava dynaaminen prosessi.

Käyttäjällä on valittavanaan useita eri tapoja liittyä laitteeseen (kts. luku 2.2, ulkoiset rajapinnat). Yksinkertaisin tapa on liittyä laitteeseen suoraan nelirivisellä matriisinäytöllä varustetulla huoltopäätteellä. Käyttöliittymä sopii erityisesti kokeneille käyttäjille yksinkertaisuutensa ja nopeutensa vuoksi. Uusille käyttäjille menupuun rakenne ja huoltopäätteen mahdollistamat toimenpiteet jäävät varmasti epäselviksi ilman tutustumista laitteen ohjekirjoihin. Kuva huoltopäätteestä ja esimerkki sen käytöstä löytyy liitteestä kolme (Huoltopääteliittymä).

Uusia käyttäjiä varten on kehitetty PC-pohjainen graafinen käyttöliittymä, joka mm. emuloi huoltopääteliittymää. Tätä kutsutaan *ACM2 Node Manageriksi*. Node Managerin etu on siinä, että menurakenne pysyy paremmin hallinnassa näytön paremman selkeyden vuoksi. Lisäksi joitain tietyllä komentosekvenssillä aikaansaatavia toimintoja voidaan ketjuttaa yhteen makroiksi, jolloin toiminnot saadaan suoritetulla yhdellä komennolla. Node Managerin varjopuoliin kuuluu se, että yksittäisissä komennoissa liittymä on hieman hitaampi (koska hiiren käyttö on hieman hitaampaa kuin yksinkertaisen numeronäppäimistön käyttö). Lisäksi kannettavaakaan tietokonetta ei ole välttämättä miellyttävää käyttää kovin pitkää aikaa ilman pöytää, ja taas huoltopääte on tehty käteen soveltuvaksi. Esimerkki ACM2 Node Managerin käyttöliittymästä löytyy liitteestä neljä.

Tietokannat

Linjatestausohjelmistossa tietoa talletetaan kahdella tavalla. Osa tiedosta talletetaan ns. väliaikaisena tietona (*temporary data*) prosessien sisälle. Osa tiedoista talletetaan sen sijaan E^2PROM:iin (*permanent data*), jossa ne säilyvät sähkökatkojenkin yli. E^2PROM:in luku- ja kirjoitusoperaatiot on toteutettu Q1-yhteisohjelmiston muistinkäsittelymoduulissa (*Q1Mmm, Q1 Memory Management Module*).

Yksittäiset tiedot eli muuttujien arvot on tallennettu E^2PROM:iin C-kielen sisäkkäisiä rakenteita käyttäen eli substruktuureilla. Liitteessä 1 on esimerkki tällaisesta rakenteesta. Linjatestauksen osalta E^2PROM:iin on talletettu erikseen molemmista mittaustyypeistä (linjamittaukset, valvontamittaukset) tiedot suoritettavista periodisista testeistä (aloitusaika ja intervalli). Kustakin yksittäisestä mittauksesta talletetaan oletusasetukset, onko mittaus sallittu vai ei (*enable/disable measurement*), ala- ja ylähälytysrajat sekä tieto siitä, annetaanko mahdollisesta rajan ylityksestä tai alituksesta hälytys vai ei. Testauskäyttöä varten talletetaan vielä otettavien näytteiden lukumäärä. Linjojen testausta varten talletetaan kustakin linjasta tieto siitä, onko linjan testaus estetty vai ei. Lisäksi pysyväismuistiin on talletettu valvontamittauksiin kuuluvat kontrollilähtöjen ja kontrollitulojen ohjaukset.

Mikäli useampi prosessi pääsee käsittelemään (lukemaan ja/tai kirjoittamaan) yhtä aikaa E^2PROM:ia, voivat seuraukset olla katastrofaalisia. Korkeammalla prioriteetilla oleva kirjoittava prosessi voi keskeyttää alemmalla prioriteetilla toimivan prosessin lukuoperaation, mikä johtaa siihen, että tieto muuttuu kesken lukutoiminnon. Luettu tieto on tällöin viallista. Tästä syystä päällekkäiset muistinkäsittelyoperaatiot on estetty Q1Mmm-moduulissa. Q1Mmm-moduuli ylläpitää listaa kaikista niistä prosesseista, jotka haluavat käsitellä E^2PROM:ia. Lista on substruktuurikohtainen, ja substruktuurit on järjestetty siten, että kunkin prosessin "tilaamien" substruktuurien määrä on minimoitu. Kukin lukemista tai kirjoittamista haluava prosessi lähettää sanoman Q1Mmm:lle, joka lisää sanoman FIFO-periaatteella toimivaan jonoon (*First-In-First-Out*). Q1Mmm pitää RAM-muistissa kopiota E^2PROM:in sisällöstä. Muistisisältöä muutettaessa Q1-Mmm:ltä täytyy ensin pyytää lupa struktuurin muuttamiseen. Kun lupa on saatu,

voidaan sisältöä muuttaa. Tämän jälkeen Q1Mmm:lle kerrotaan, mistä uusi struktuuri löytyy. Q1Mmm tarkistaa, että sisältö todella on muuttunut, ja aloittaa tämän jälkeen E^2PROM:in sisällön päivityksen. Päivityksen päätyttyä se lähettää kaikille kyseisestä substruktuurista kiinnostuneille prosesseille tiedon substruktuurin muuttumisesta (osoittimen kysessä olevaan struktuuriin). Prosessi voi halutessaan myös poistaa itsensä subtruktuurin tilaajalistasta lähettämällä Q1Mmm-moduulille tästä kertovan sanoman.

Prosessien sisäistä (väliaikaista) tietoa ovat mm. päiväys ja kellonaika, jolloin testi on tehty, mittaustulokset sekä kunkin prosessin sisäistä toimintaa varten tarvittavien muuttujien arvot.

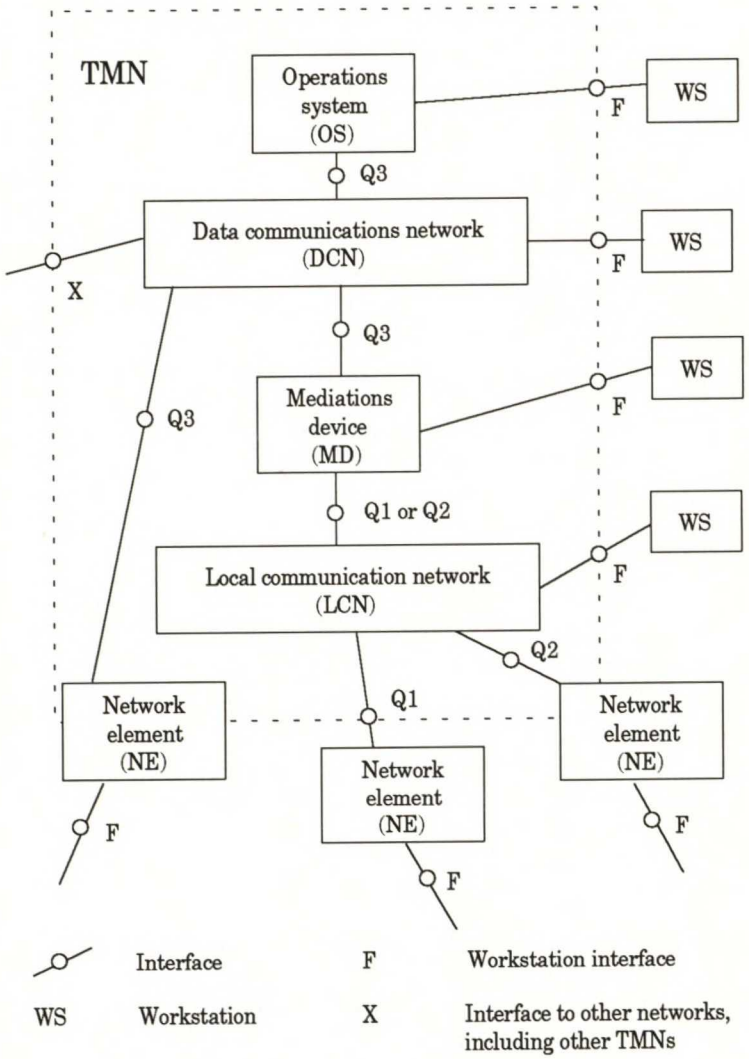
3.3 Linjatestaus ja verkonhallinta

Q1-rajapinta

CCITT:n suosituksessa G.771 määritellään rajapinnat Q1 ja Q2, sekä kuvaillaan protokolla, joka tarvitaan siirtolaitteen liittämiseksi tietoliikennelaitteiden hallintaverkkoon (TMN). Tietoliikennelaitteiden hallintaverkko on kuvattu tarkemmin suosituksessa M.30. Verkko on tarkoitettu paitsi siirtolaitteiden myös puhelinkeskusten hallintaan. [CCITT, M.30]

Siirtojärjestelmien laitteet, joihin ACM2:kin kuuluu, esitetään TMN-konseptissa verkon elementteinä (*network element*, NE). Elementtejä hallitaan verkon hallintalaitteilla rajapintojen Q1, Q2, Q3 ja F kautta (kuva 14). Suosituksen G.771 mukaan tällaisen rajapinnan tulee sijaita mahdollisimman lähellä fyysistä laitetta (siirtolaitetta tai keskusta).

Q3-rajapinta on paljon monimutkaisempi kuin Q1- ja Q2-rajapinnat. Q3 toteuttaa kaikki OSI-mallin mukaiset seitsemän kerrosta, kun Q1 ja Q2 toteuttavat vain kolme alinta kerrosta. Q1:n toteuttava protokolla on yksinkertaisempi kuin Q2:n toteuttava protokolla, sillä Q1 ei sisällä välitystoimintoa (*mediation function*).



Kuva 14. Tietoliikenteen hallintaverkon (TMN) arkkitehtuuri suosituksen M.30 mukaan

Q1-rajapinta ja -protokolla on toteutettu Q1Mnu-moduulissa eli Q1_302-yhteisohjelmistoon kuuluvassa menuohjelmassa. Käytetty protokolla TMSP (*Transmission Management System Protocol*) on osittain määritelty itse, sillä suositukset eivät suoranaisesti kerro, miten Q1-protokolla tulisi toteuttaa. Verkon hallintalaitteena toimii huoltopääte tai TMC (*Transmission Management Computer*).

Protokollan toteuttamaksi liikennöintitavaksi on valittu vuorosuuntaisuus (*half-duplex*). Tämä tarkoittaa käytännössä sitä, että huoltopääte (tai TMC) ei lähetä uutta komentoa ennen kuin vastaus on saatu tai tietyn ajan kuluttua (valittavissa välillä 200 ms...5 s) luovutaan odottamasta vastausta.

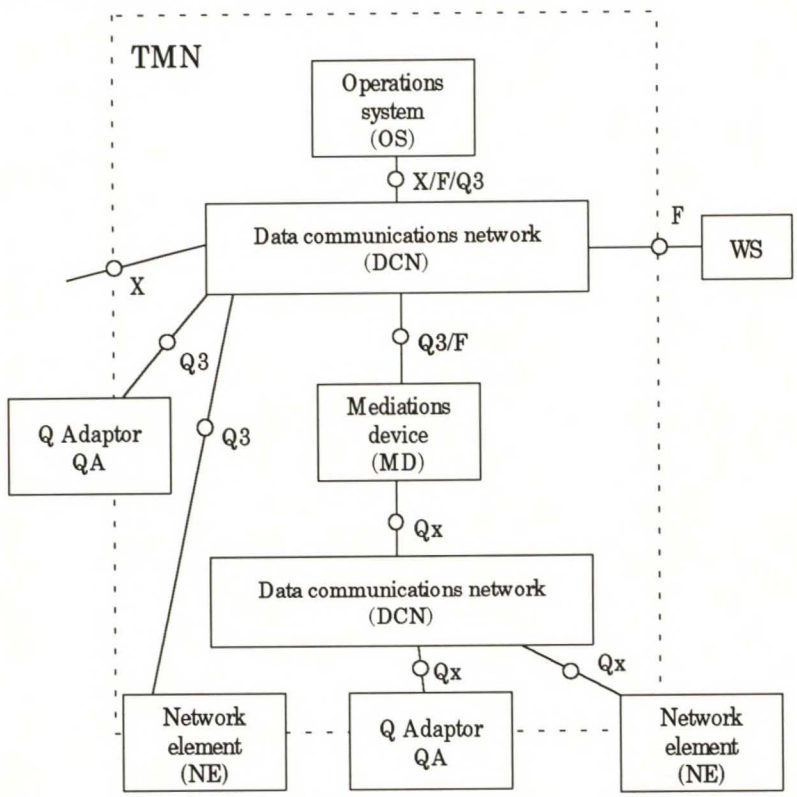
Protokolla käsittää yhteensä 47 komentoa, joita käytetään konfigurointiin, varmistusten ylläpitoon (ei käytetä ACM2:ssa), kiertokyselyyn (*polling*), vika-tietojen tiedustelemiseen, aikavälien hallintaan ja muuhun tietojen välittämiseen.

TMSP-protokolla määrittelee hälytysyksikön (*Alarm Unit,* AU), joka koostuu toiminnallisista kokonaisuuksista (*Functional Entity,* FE). Tyypillinen toiminnallinen kokonaisuus käsittää yhden laitteen (esim. multiplekserin). Kukin

toiminnallinen kokonaisuus sisältää yhden tai useampia (kuitenkin enintään 255) hallintalohkoja (*Supervision Block,* SB). Tyypillinen hallintalohko käsittää osan toiminnallista kokonaisuutta (esim. yhden kanavakortin). Kukin hallintalohko sisältää yhden vikakooditaulukon. [Lassus 1992, s. 10...15]

M.30-suositus on parhaillaan korvautumassa uudella suosituksella M.3010. [CCITT M.3010] Uudessa suosituksessa hallittavien verkkoelementtien määrää on lisätty ja rajapintoja muutettu. Q1- ja Q2-rajapinnat on kokonaan korvattu uudella rajapinnalla Qx. Suosituksen mukaan Q3 ja Qx eroavat toisistaan lähinnä kuljettamansa informaation mukaan. Uutena elementtinä mukaan on tullut Q1-sovitin (*Q1 Adaptor*). Q1-sovitin on laite, jonka avulla on mahdollista kytkeytyä verkon elementistä (NE) tai käyttöjärjestelmästä (OS) Qx- tai Q3-rajapinnan kautta laitteeseen, jolla ei ole TMS-yhteensopivaa rajapintaa.

Kuvassa 15 on esitetty tietoliikenteen hallintaverkon arkkitehtuuri uuden suosituksen mukaisesti.

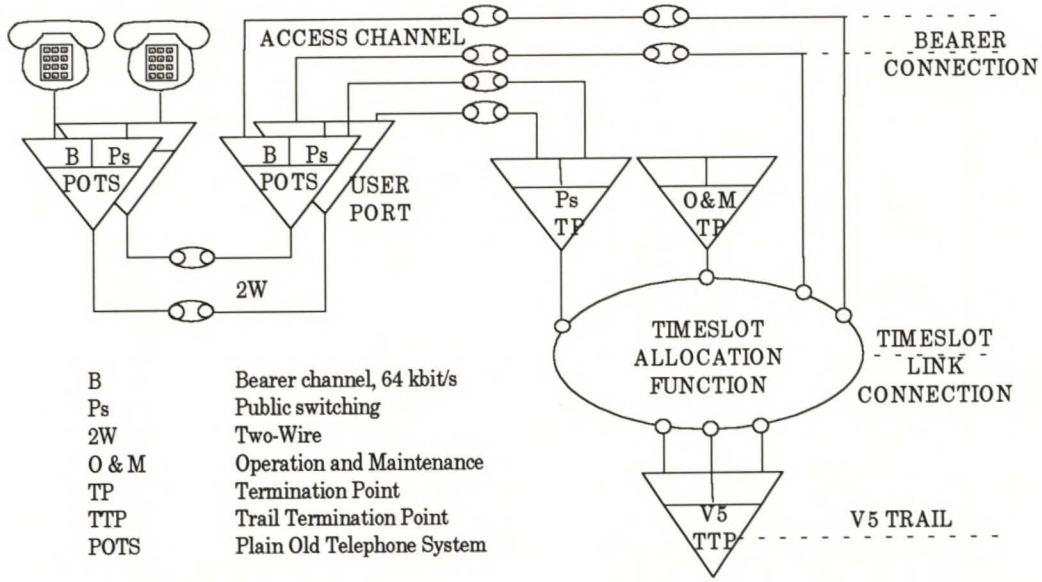


Kuva 15. Tietoliikenteen hallintaverkon (TMN) arkkitehtuuri suosituksen M.3010 mukaan

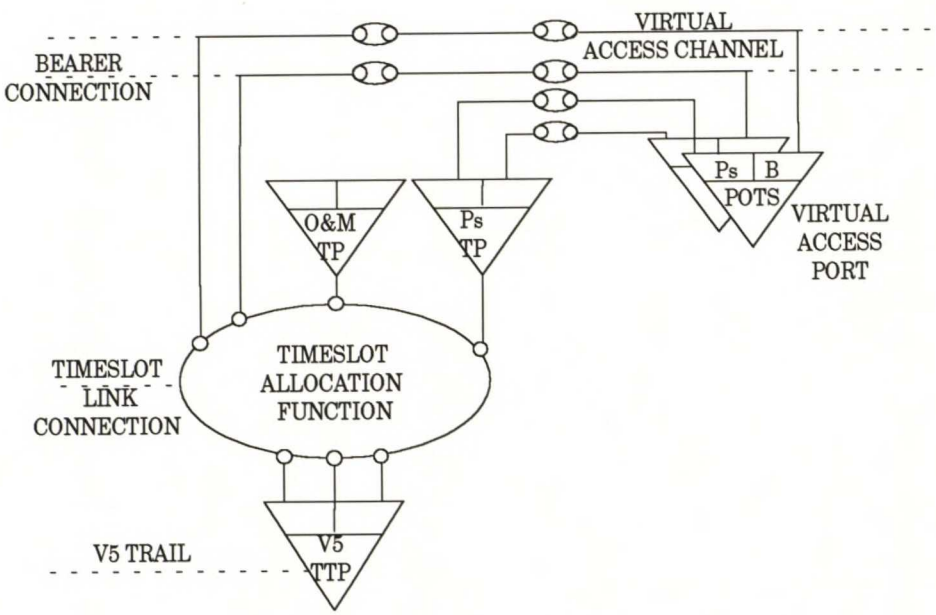
V5- ja Q3-rajapinta

ETSI:n standardi DE/SPS3013.1 [DE/SPS 1993, s. 1...58] määrittelee tilaajaverkon (AN) ja tietoliikennelaitteiden hallintaverkon (TMN) välisen Q3-rajapinnan V5-rajapintojen (verkon)hallinnan näkökulmasta. V5-rajapinnalla tarkoitetaan keskuksen ja tilaajaverkon välistä rajapintaa, jossa on toteutettu joitakin kapeakaistaisia palveluja. Q3-rajapintaan liittyvän Q3-protokollan avulla voidaan

hallita TMN:n kautta kyseistä V5-rajapintaa ja siihen liittyviä käyttöteitä (*user port*). V5-rajapinta liittyy ylemmällä tasolla sijaitseviin käyttöteihin ristikytkentöjen kautta.



Kuva 16. Tilaajaverkon V5-rajapinnan toiminnallinen arkkitehtuuri



Kuva 17. Paikallisen keskuksen V5-rajapinnan toiminnallinen arkkitehtuuri

Kuvissa 16 ja 17 on esitetty V5-rajapinnan toiminnallinen arkkitehtuuri. Yksi tilaajaverkko voi käsittää useita V5-rajapintoja ja -käyttöteitä. V5-konseptissa keskus hallitsee tilaaja eli access-verkon aikavälien käyttöä tätä tarkoitusta varten varatun kommunikointikanavan avulla *(timeslot link connection)*, ja kontrolloi tilaajaverkon (fyysisiä) yhteyksiä (V5 *trail*). *Bearer connection*:illa kuvataan normaalia puhesignaalin siirtotietä. Kuvissa esiintyvät lyhenteet on selitetty lyhenneluettelossa.

V5-rajapinta toteutetaan vasta tulevissa ACM2-laitesukupolvissa. Linjatestauksen kannalta on mielenkiintoisinta, että myös käyttö ja ylläpito (O&M) voi tapahtua keskuksesta käsin. Käytännössä tämä tarkoittaa sitä, että multiplekserin on osattava tulkita keskukselta tulevat testipiirin ohjainkomennot.

4. Toteutus

Tässä luvussa käsitellään linjatestausohjelmiston toteutusta, testaamista ja integrointia muuhun ohjelmistoon ja laitteistoon. Lopuksi pohditaan projektin onnistumista sekä sen heikkouksia ja vahvoja puolia.

4.1 Linjatestausmoduulin ohjelmiston toteutus

Ohjelmointiympäristö

Linjatestausmoduuli toteutettiin käyttäen Motorolan MC68302-prosessoria ja Enea Datan OS68-reaaliaikakäyttöjärjestelmän versiota 2.50. Käännös tehtiin käyttäen Intermetricsin C-kääntäjän versiota 7.2. Kehitysympäristönä olivat Hewlett-Packardin Apollo-työasemat. Simulointiympäristöön käännös tehtiin Apollo-työasemien omalla C-kääntäjällä. Versionhallintaan käytettiin DSEE:tä (*Domain Software Engineering Environment*).

Reaaliaikakäyttöjärjestelmällä tarkoitetaan ohjelmaa, jonka täytyy vastata ulkoiseen tapahtumaan tietyn ajan kuluessa. [OS68 1992, s. 3] Reaaliaikakäyttöjärjestelmiä käytetään siis aikariippuvissa sovelluksissa, kuten juuri tietoliikenteessä, lentoliikennejärjestelmissä jne. Tyypillistä reaaliaikakäyttöjärjestelmälle on, että itse käyttöjärjestelmässä on toteutettu asioita, jotka muuten hallitsisivat ohjelman suoritusta. Tällaisia ovat mm. kontrollisilmukat, hypyt, suorituskutsut ja tietyt globaalit muuttujat. Lisäksi reaaliaikainen käyttöjärjestelmä huolehtii resurssien hallinnasta, prosessointiajan jakamisesta sekä prosessien välisestä kommunikaatiosta.

Prosesseja voi olla kahdenlaisia, jo aiemmin mainittuja staattisia prosesseja sekä dynaamisia prosesseja. Staattiset prosessit käynnistetään kerran käyttöjärjestelmän käynnistyessä, ja siitä lähtien ne pysyvät toiminnassa, kunnes koko järjestelmän toiminta lakkaa. Dynaaminen prosessi saa alkunsa prosessin luomisen aiheuttavan käskyn suorittamisesta. Dynaaminen prosessi on tyypillisesti rakennettu suorittamaan vain joku tietty, rajallinen tehtävä. Tämän tehtävän päätyttyä se tappaa itsensä ja vapauttaa kaikki varaamansa muistialueet. ACM2:ssa dynaamisia prosesseja on käytetty käyttöliittymän rakentamiseen. Staattisten ja dynaamisten prosessien lisäksi järjestelmässä saattaa olla joitakin keskeytysprosesseja. Keskeytysprosessi keskeyttää nimensä mukaisesti kaikkien muiden prosessien (paitsi korkeammalla prioriteetilla olevan toisen keskeytysprosessin) suorituksen oman suorituksensa ajaksi aina tietyn keskeytystapahtu-

man sattuessa. Keskeytyksen aiheuttaa useimmiten jokin kiireellinen laitteistoriippuvainen tapahtuma.

Prosessit voivat kommunikoida keskenään lähettämällä toisilleen sanomia. Käyttöjärjestelmä huolehtii sanoman sisältämän informaation välittämisestä prosessilta toiselle (toteutettu osoittimen välityksenä).

Prosessilla on suorituksen aikana kolme mahdollista tilaa; se on joko CPU:n käsiteltävänä (*running*), valmiina (*ready*) tai odottamassa (*waiting*). Prosessi on suoritettavana silloin kun prosessori suorittaa jotain siihen kuuluvan koodin osaa. Prosessi on valmiina silloin kun joku toinen, mahdollisesti korkeammalla prioriteetilla oleva prosessi on suoritettavana, ja tämä prosessi odottaa käsiteltäväksi pääsyä. Prosessi on odottamassa silloin, kun se joko odottaa sanomaa tai se on ohjelmallisesti laitettu odottavaan tilaan tätä tarkoitusta varten tehdyllä käyttöjärjestelmän kutsulla.

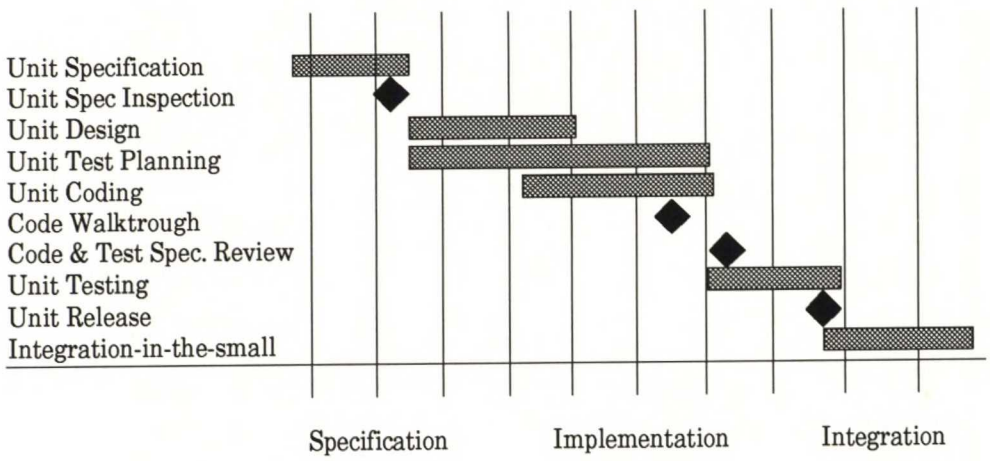
Koska korkeammalla prioriteetilla oleva prosessi voi aina keskeyttää alhaisemmalla prioriteetilla olevan prosessin käsittelyn, on reaaliaikajärjestelmään kirjoitetulla koodilla myös tiettyjä erikoisvaatimuksia. Tärkein vaatimus on se, että koodin tulee olla vapaakäyntistä (*re-entrant*). Tällä tarkoitetaan sitä, että koodin tulee olla keskeytettävissä ja sen suoritusta on kyettävä jatkamaan virheettä milloin tahansa. Esim. kaksi prosessia ei voi käyttää yhteistä funktiota, jossa on funktion eri kutsukertojen välillä arvonsa säilyttäviä static-tyyppisiä muuttujia, koska tällöin funktion muuttujien arvot vaihtelevat sen mukaan, missä vaiheessa funktion suoritusta nämä kaksi prosessia keskeyttävät toisensa.

On myös olemassa joitakin laitteistoriippuvaisia operaatioita (rekisterien luku- ja kirjoitusoperaatiot), jotka on pystyttävä suorittamaan ilman keskeytyksiä. Prosessin prioriteetin nosto tällaisissa tapauksissa on eräs (useimmiten tosin huono) ratkaisu. Prioriteetin nosto ei useimmiten riitä ratkaisuksi, sillä järjestelmässä saattaa olla monia muitakin prosesseja, jotka tarvitsevat korkeaa prioriteettia (esim. ACM2:ssa signalointi). Tällöin keskeytysten esto kriittisten laitteisto-operaatioiden ajaksi jää ainoaksi vaihtoehdoksi.

Yksikkötason toteutus

Yksiköksi eli unitiksi nimitetään sellaista loogista kokonaisuutta, jonka toteutus on yhden ihmisen hallittavissa, ja joka on toteutettavissa yhden ihmisen voimin riittävän lyhyessä ajassa projektin kokonaisaikataulua silmällä pitäen. Tyypillisesti yksi unitti on yksi prosessi tai moduuli.

ACM2-projektin vaiheessa kaksi on kokeiltu uutta, hallittua tapaa unitin toteuttamiseen. Keskeisintä tavassa on koodauksen hallittavuus tarkkojen määritysten ja tiettyjen rajapyykkien avulla.



Kuva 18. Unitin toteutuksen eri vaiheet

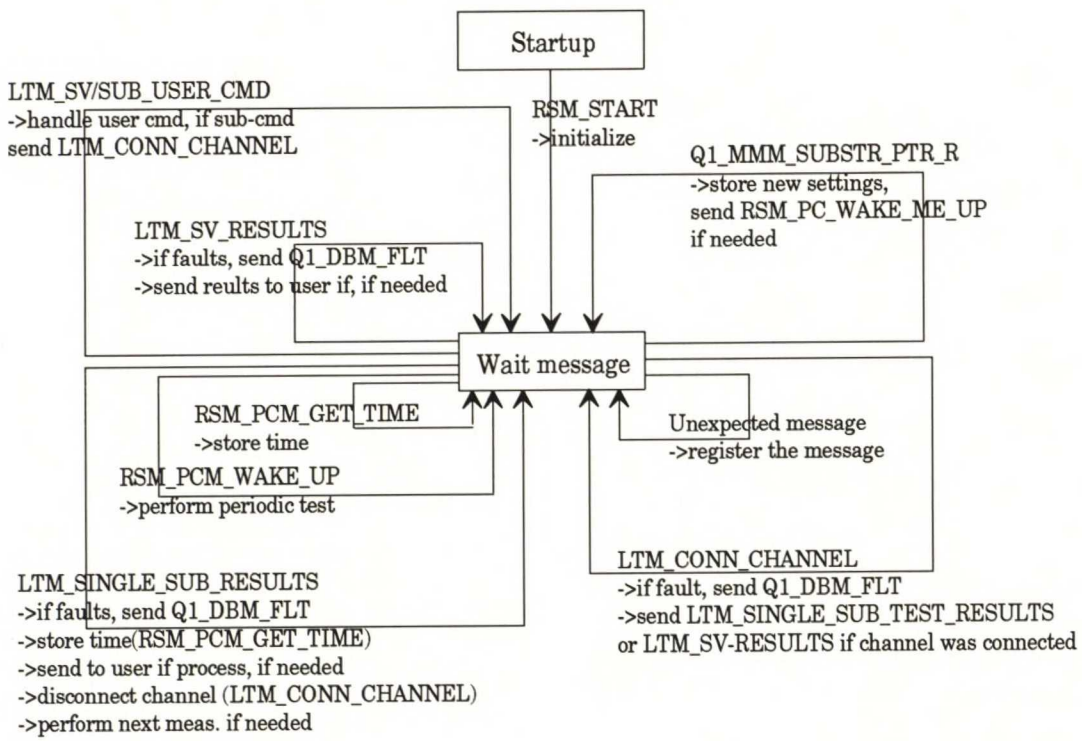
Kuvassa 18 on esitetty nämä uudet unitin toteutuksen eri vaiheet (määrittely, toteutus ja integrointi). Toteutukseen liittyvät tehtävät (esim. koodaus, *unit coding* tai määrittely, *unit specification)* on lueteltu kuvan vasemmassa laidassa. Viimeisenä tehtävänä on unitin integroiminen muihin samassa osaprojektissa toteutettuihin unitteihin (*integration-in-the-small*). Kuvan suorakaiteet kuvaavat eri tehtävien suoritusjärjestystä toisiinsa nähden. Rajapyykit on kuvattu vinoneliöillä *(unit specification inspection, code & test specification review, unit release* ja *code walktrough*). Varsinkin viimeksimainittu rajapyykki (*code walktrough*, virheiden metsästäminen koodista käymällä se yksityiskohtaisesti läpi projektiryhmässä) on koettu projektin aikana hyödylliseksi, sillä se lyhentää virheiden etsimiseen kuluvaa aikaa. Varsinainen koodaus on useimmiten suoritettu käyttäen joko emacs- tai vi-editoreita. Dokumentointi on yleensä tehty käyttäen Interleaf-julkaisujärjestelmää. [Kaitala 1993, s. 8..12]

Projektissa on myös noudatettu joitakin nimeämissääntöjä, joista ehkä tärkein on muuttujan nimeäminen sen tyypin mukaan. Esimerkiksi osoitintyyppisen muuttujan etuliite on p, ja CHAR-tyyppisen c. Integerien (C-kielen kokonaislukutyyppi) sijasta käytetään itse määriteltyä tyyppiä WORD (16-bittinen etumerkillinen kokonaisluku) tai UWORD (16-bittinen etumerkitön kokonaisluku) helpottamaan mahdollista siirtymistä Motorola-maailmasta Intel-maailmaan ja päinvastoin. Lyhyiden kokonaislukujen (C-kielen *short int*) sijasta käytetään tyyppiä BYTE (kahdeksanbittinen etumerkitön kokonaisluku). Esimerkkejä näiden sääntöjen käytöstä löytyy liitteistä, esim. liite yksi (linjatestaustietokanta E^2PROM:ssa). Sanomat on nimetty käyttäen isoja kirjaimia ja erottamalla nimen eri osat alaviivoilla toisistaan. Sanoman ensimmäinen osa on aina sen moduulin nimi, johon sanoma kiinteimmin liittyy (esim. LTM_SUB_USER_CMD). Funktioiden nimet on kirjoitettu käyttäen isoja alkukirjaimia ja kirjoittamalla sanat joko yhteen tai käyttäen väliviivaa (esim. *Use_Exte, UpdateSubstructureValues*). Prosessien nimet kirjoitetaan muuten vastaavalla tavalla, paitsi että ne alkavat aina sen moduulin nimellä, johon ne kuuluvat, ja päättyvät aina isoon P-kirjaimeen (esim. *LtmManagerP*). Lisäksi kaikki nimet ja kommentit on kirjoitettu englannin kielellä.

Projektilla on ollut käytössään C-kielen tyyliopas, jossa on ollut yksityiskohtaisia neuvoja epäilyttävien rakenteiden käytön välttämisestä (*goto*-lauseet) funktioiden kokoa koskeviin suosituksiin saakka. [Eriksson 1991, s. 1...35]

Testien käsittelyprosessin toteutus

Koska koko linjatestausohjelmiston toteutuksen kuvaaminen ei tässä yhteydessä ole järkevää, on esimerkiksi otettu testien käsittelyprosessin toteutus. Testien käsittelyprosessin ottaminen esimerkkiprosessiksi on perusteltua siksi, että se muodostaa toiminnoiltaan varsin laajan kokonaisuuden. Prosessin keskeisyydestä johtuen laajatkin asiat tulevat hyvin kuvatuiksi.



Kuva 19. Testien käsittelyprosessin tilakaavio

Kuvassa 19 on esitetty testien käsittelyprosessin tilakaavio. Käynnistyttyään prosessi jää odottamaan RSM_START-sanomaa, joka saapuessaan kertoo Q1_302-moduulin olevan alustettu ja valmis ottamaan vastaan sanomia. Tämän jälkeen testien käsittelyprosessi lähettää Q1_302:lle sanoman (*Q1_MMM_ADD_TO_SUBSTRPTR_LIST_F*), joka kertoo sen olevan kiinnostunut E^2PROM:in linjatestausstruktuuriin talletetuista tiedoista. Q1_302 vastaa kyselyyn sanomalla *Q1_MMM_SUBSTR_PTR_R*. Testien käsittelyprosessi saa saman sanoman aina myös silloin, kun struktuuri on muuttunut. Mikäli sanomassa on mukana pyyntö suorittaa periodisia testejä, lähettää testien käsittelyprosessi herätyspyynnön *RSM_PCM_WAKE_ME_UP* tilastointiprosessille. Itse herätys saadaan sanomassa *RSM_PCM_WAKE_UP*.

Tämän jälkeen testien käsittelyprosessi jää odottamaan lisää sanomia. Tilakaaviossa näkyvistä sanomista suurin osa lähetetään käyttäjän aikaansaamien toimenpiteiden seurauksena. Sanoma *LTM_SV_USER_CMD* sisältää käyttäjän antamia valvontatoimenpiteisiin liittyviä käskyjä (esim. mittaustuloksien luku, välittömät mittauskäskyt), ja *LTM_SV_SUB_USER_CMD* vastaavia tilaajajohtojen mittauksiin liittyviä komentoja. Sanomaa *LTM_CONN_CHANNEL* käytetään kommunikointiin signalointiprosessin kanssa, jolla on tieto kunkin tilaajajohdon tilasta (*IDLE*, non-*IDLE*, ei-kytketty) ja valta yhdistää linjatestausväylä tilaajajohtoon tai purkaa kytkentä. Sanomaa *Q1_DBM_FLT* käytetään vikatietojen raportoimiseen, ja sanomaa *RSM_PCM_GET_TIME* ajan kysymiseen tilastointiprosessilta. Tilaajajohtojen mittaustulosten yhteyteen talletetaan aina aika, jolloin mittaukset on tehty. Sanomilla *Q1_QCS_TO_RQST_F* ja *Q1_SYSTEM_ACK_R* käytetään Q1Qcs:n (*Q1 Common Services module*) ajastuspalveluja. Niitä tarvitaan johdon vapauttamiseen tietyn ajan kuluttua kun se on ensin varattu ulkoisen mittalaitteen käyttöön.

Mittaustulokset tallennetaan sanomiin *LTM_SUB_RESULTS* ja *LTM_SV_-RESULTS*. Nämä sanomat allokoidaan kerran testien käsittelyprosessin käynnistymisen yhteydessä, ja niitä käytetään paitsi testitulosten tallettamiseen myös mittaustulosten välittämiseen käyttäjälle ja valvontatulosten osalta myös uusien mittaustulosten välittämiseen mittausprosessilta testien käsittelyprosessille. Kun sanomaa ei tarvita kommunikointiin, varastoi testien käsittelyprosessi sitä itsellään. Yksittäiset tilaajajohtojen mittaustulokset välitetään mittausprosessilta testien käsittelyprosessille käyttämällä sanomaa nimeltä *LTM_SINGLE_SUB_TEST_RESULTS*. Tällaiseen ratkaisuun on päädytty rajoitetun muistikapasiteetin vuoksi.

Kuvissa 20 ja 21 on esimerkinomaisesti näytetty sanomajonokaavioin ensin välittömän valvontamittauksen ja sitten periodisen tilaajajohtojen mittauksen aikaansaamat sanomien lähetykset ja vastaanotot. Kuvassa *DynProc* tarkoittaa käyttöliittymän dynaamista prosessia. *SvMeas* tarkoittaa valvontaprosessia, ja *SubMeas* tilaajajohtojen mittausprosessia. *StatP* on tilastointiprosessi, *SigManager* signalointiprosessi ja *Q1Mmm* Q1_302:een kuuluva E^2PROM:in hallintaprosessi. Periodisessa mittauksessa mitataan kaikki tilaajajohdot, ts. kuvan 21 sanomia 9-24 toistetaan kullekin linjalle erikseen. Sanomissa 1-6 tapahtuu herätyksen asetus. Seuraavalla mittauskerralla vaiheita 1-6 ei siis enää toisteta.

Esimerkki linjatestausohjelmistossa käytetystä sanomasta löytyy liitteestä kaksi.

Messages/modules	DynProc	Ltm Manager	SvMeas
LTM_SV_USER_CMD	1	2	
LTM_SV_RESULTS		3	4
LTM_SV_RESULTS		6	5
LTM_SV_RESULTS	8	7	
LTM_SV_RESULTS	9	10 *END*	

Kuva 20. Välittömän valvontamittauksen sanomajonokaavio

Messages/modules	DynProc	Q1Mmm	Ltm Manager	StatP	Sig Manager	SubMeas
Q1_MMM_SUBSTR_PTR_F	1	2				
Q1_MMM_SUBSTRPTR_R		3	4			
RSM_PCM_WAKE_ME_UP			5	6		
RSM_PCM_WAKE_UP			8	7		
LTM_CONN_CHANNEL			9		10	
LTM_CONN_CHANNEl			12		11	
LTM_SINGLE_SUB_TEST_RESULTS			13			14
LTM_SINGLE_SUB_TEST_RESULTS			16			15
LTM_CONN_CHANNEL			17		18	
LTM_CONN_CHANNEL			20		19	
RSM_PCM_GET_TIME			21	22		
RSM_PCM_GET_TIME			24 *END*	23		

Kuva 21. Periodisen tilaajajohtojen testauksen sanomajonokaavio

Liitteessä seitsemän on esitetty testien hallintaprosessin SDL-kaavio, ja liitteessä kahdeksan esimerkkifunktio samasta prosessista.

4.2 Testaus

ACM2:n testausta on tehostettu jakamalla se osiin. Ensimmäisessä vaiheessa testataan kutakin unittia erikseen. Toisessa vaiheessa unitit integroidaan toisiinsa, ja niitä testataan yhdessä. Kolmannessa vaiheessa testataan rakennettua järjestelmää yhtenä kokonaisuutena.

Ohjelmiston unit-testaus

Unit-testauksessa kutakin unittia testataan ensin omassa simulointiympäristössään, jossa ainoastaan testattava prosessi esiintyy todellisena prosessina. Muut ympäristössä olevat prosessit ovat tyhjiä, eli ns. *dummy*-prosesseja, jotka vain vastaanottavat sanomia eivätkä käsittele niitä millään lailla. Tämän jälkeen ympäristöön luodaan toinen todellinen prosessi, jonka tehtävänä on tuottaa sopivia sanomia ensimmäisen prosessin testaamisen automatisoinniksi. Testattavan prosessin luonteesta johtuen toinen vaihe voi myös puuttua kokonaan. Kolmannessa vaiheessa prosessia testataan CTB/CTC-työkalulla (*C Test Bed generator/C Test Coverage analyzer*) [Lundström 1993, s. 1...15] funktio kerrallaan. CTB:tä voidaan käyttää yksittäisten unittien ja/tai funktioiden testaamiseen. CTC laskee haarautumien lukumääriä. Testauksen päämääränä on kutsua testattavaa funktiota mahdollisimman edustavalla joukolla parametreja. CTB/CTC antaa luettelon kaikista suoritetuista ehtolauseista ja niiden saamista arvoista, sekä kertoo missä haaroissa ei ole käyty. Kun testauksella on saavutettu 100%:n lause- ja päätöskattavuus (so. jokainen ehto on suoritettu ainakin kerran), saadaan listausta tutkimalla selville toimiiko ohjelman logiikka sen sunnittelijan ajattelemalla tavalla. CTB/CTC:n puutteena on testauksessa tarvittavan määrittelytyön suuri osuus. Tuloksiin ei myöskään pidä luottaa liikaa, sillä ohjelmahan ei testattaessa toimi varsinaisessa ympäristössään, jolloin mm. ajoitusongelmat jäävät huomioimatta.

Ohjelman kompleksisuuden selvittämisessä on käytetty CMT-työkalua (*Complexity Measures Tool*). CMT laskee ohjelmasta tiettyjä tunnuslukuja, kuten McCaben syklomaattisen luvun (joka itse asiassa on sama kuin ohjelmasta piirretyssä vuokaaviossa olevien haarautumien lukumäärä), ohjelmarivien lukumäärän ja kommenttirivien lukumäärän sekä joukon ns. Halstedin tunnuslukuja, jotka perustuvat operaattoreiden ja operandien frekvenssien laskemiseen. CMT:llä saadaan hyviä vihjeitä siitä, mitä ohjelman osia tulisi parantaa. Toisaalta on tilanteita, jolloin kompleksisuusasteen kertovat tunnusluvut eivät suoraan ole verrannollisia ohjelman ymmärtämiseen kuluvaan aikaan. Tällainen tilanne on vaikkapa silloin, kun ohjelma sisältää monia peräkkäisiä keskenään samankaltaisia ehtolauseita, joita ei kuitenkaan voida yhdistää yhdeksi funktioksi (esim. komentotulkki). [CMT++ 1989, s. 1...18]

Itse testaustavat voidaan jakaa kolmeen eri luokkaan. Tyypillisesti kunkin unitin simulointitestit alkavat mustan laatikon testeillä (*black box testing*), joissa testattavaa unittia tarkastellaan lähinnä eri syötteiden (sanomien) aikaansaamien vasteiden (vastaussanomien) perusteella. Tämän jälkeen testaus muuttuu harmaan

laatikon testaukseksi (*grey box testing*), jossa syötteiden arvot valitaan unitin päätöslauseissa esiintyvien raja-arvojen mukaan. CTB/CTC-testaus taas on luonteeltaan valkoisen laatikon testausta (*white box testing*), jossa itse unitissa oleva koodi vaikuttaa testin kulkuun (esim. lause- ja päätöskattavat testit).

Unittitestauksen tarkoituksena on löytää kaikki unitin sisällä piilevät virheet. Siirtymisen integrointitestausvaiheeseen tulisi tapahtua vasta, kun unitin on todettu toimivan määritelmien mukaisesti.

Integrointitestaus

Integrointitestauksessa kukin unitti integroidaan ensin simulointiympäristössä muihin unitteihin. Simulointiympäristössä järjestelmästä löytyvät tässä vaiheessa enää vain kaikkein silmiinpistävimmät viat, rajapintojen epäselvyydet ja selvät virhetoiminnat. Käyttämässämme simulointiympäristössä ajoituksiin liittyviä ongelmia ei voi todentaa. Tässä suhteessa simulointiympäristöön käännetty koodi eroaa todelliseen laiteympäristöön käännetystä koodista, mikä näkyy etenkin priorisoinniltaan normaalista poikkeavien prosessien (keskeytysprosessien ja taustalla toimivien prosessien eli *background*-prosessien) toiminnassa.

Tämän jälkeen siirrytään testaamaan koko järjestelmää varsinaisessa laiteympäristössä emulaattorin avulla. Tähän mennessä laitteisto on jo tietysti täytynyt testata erikseen, jotta nähdään toimiiko se määrittelyjen mukaisesti. Ohjelmiston ja laitteiston integroinnissa täytyy varmistua siitä, että a) ohjelmistoa voidaan ajaa laiteympäristössä, b) ohjelmisto voi kontrolloida laitteistoa halutulla tavalla ja c) kaikki tarvittava toiminnallisuus on toteutettuna. [Kaitala 1993, s. 13] Emulaattoria käytettäessä on apua erilaisista vianetsintään suunnitelluista ohjelmista, kuormitusanalysaattoreista ja hyvästä assembler-tuntemuksesta. On huomattava, että osa uniteista voidaan testata kokonaisuudessaan vasta tässä vaiheessa (laitteistoriippuvat ohjelmiston osat), joten kunnon emulaattorin merkitystä ohjelmiston testauksen kannalta voi tuskin aliarvioida.

Emulaattoritestauksen jälkeen siirrytään vielä integrointitestaukseen todellisen prosessorin kanssa. Tässä vaiheessa yksittäisten muuttujien viallisia arvoja on enää hyvin vaikea paikallistaa. Integrointitestauksen toisessa vaiheessa esiin tulevat ongelmat ovatkin yleensä kaikkein hankalimpia löytää, ja useimmiten niiden aiheuttajana on joko väärinymmärretty vaatimus, ajoitusongelma tai monen eri prosessin yhteistoiminnan summa. Pahimmassa tapauksessa löytyy muisti- ja suorituskykyongelmia.

Systeemitestaus

Lopuksi suoritetaan vielä koko ohjelmiston ja laitteiston kattava systeemitestaus, jossa varmistutaan siitä, että tuote todellakin toimii (*end-to-end* testaus). Kattavaan multiplekserin systeemitestaukseen kuuluu mm. laitetelineen (*subrack*) testaus, jännitelähteen ja soittogeneraattorin testaus, lämpötestaukset, johdotusten testaukset, sähköisten häiriöemissioiden mittaukset (EMC, *Electro-*

magnetic Compatibility), sähköisten ominaisuuksien testaus (esim. häiriönsietokyky), laitekonfiguraation testaus, oletusasetusten testaus, siirto-ominaisuuksien testaus, signalointilogiikan ja vikakäyttäytymisen testaus, käyttöliittymän testaus ja erilaiset laitteen hallintatestaukset. Systeemitestausta jatketaan niin kauan, kunnes kaikki havaitut viat on korjattu ja uusia ei enää havaita tai ne todetaan luonteeltaan vähäpätöisiksi.

Testauksen ongelma-alueita

Testausaikataulujen laadinta on osoittautunut ongelmalliseksi, sillä näyttää siltä, että testaukseen kuluu aina enemmän aikaa kuin on arvioitu. Osasyy tähän saattaa olla se, että määräaikojen lähestyessä siirrytään liian nopeasti koodausvaiheesta testausvaiheeseen ja testataan puolittain toimimatonta koodia. Tällöin itse testit täytyy tietenkin suorittaa moneen eri kertaan, sillä pienikin muutos koodissa saattaa näkyä ulkoisesti useaan eri paikkaan.

Eräs ongelma-alue testauksessa on myös testauksen lopettaminen. Kun suoritan mittauksen vakiosyötteillä kerran ja saan oikean tuloksen, onko se riittävä varmistus siitä, että mittaus toimii? Kuinka epäsäännöllisesti esiintyvä häiriö tulisi mittauksissa vielä havaita? Riittääkö tarkkuus, jos voidaan todeta että ainakin 99 mittausta sadasta antaa oikean tuloksen? Ongelmia aiheuttavat mm. komponenttien lämpeneminen, toimintaympäristön säätilan vaihtelut, kosteus, toimintaympäristöstä indusoituvat jännitteet ja kapasitiivinen kytkeytyminen. Laitteistosuunnittelussa lämpenemistä voi jonkun verran ottaa huomioon sijoittamalla komponentit niin, että vähemmän herkät komponentit sijaitsevat enemmän lämpenevillä alueilla. Sama pätee erilaisiin häiriöihin. Piirilevyn infrapunakuvaus ja sääkaapit ovat hyviä apuneuvoja ongelmia ehkäistessä.

Ainoa apu testauksen kattavuuteen on ohjelmallisesti suorittaa samaa testiä kerta toisensa jälkeen niin, että vain poikkeamat raportoidaan erilliseen lokitiedostoon. Ensisijaisesti häiriöitä voi yrittää korjata laitteistoa muuttamalla. Ellei tämä onnistu, voidaan säännöllisen häiriön ollessa kysymyksessä poistoa yrittää myös ohjelmallisesti, mutta tämä vaatii yleensä useita mittauksia ja testejä oikeiden korjauskertoimien löytämiseksi.

4.3 Ongelmia ja niiden ratkaisuja

EPROM

ACM2:n ohjelmisto poltetaan laitetuotannossa EPROM:ille, joka toimitetaan muun laitteiston ohella käyttäjälle. Kun käyttäjä kytkee laitteeseen ensimmäistä

kertaa virran, latautuu ohjelmisto EPROM:ilta dynaamiseen muistiin. EPROM:in koko on tällä hetkellä 512 kilotavua, josta vaiheen yksi ohjelmisto kuluttaa ilman *terminal debuggeria* 360 kB, ja terminal debuggerin kanssa 460 kB. Terminal debuggeria tarvitaan etsittäessä vikoja emulaattorin kanssa. Linjatestausohjelmisto vaatii EPROM:ia noin 100 kB, ja Dynacard-ohjelmisto noin 50 kB. Yhteensä arvioitu EPROM:in tarve on siis terminal deguggerin kanssa noin 610 kB, ja ilman debuggeria 510 kB. Komponenttitoimittajalla ei suoraan ole tarjota suurempikapasiteettista EPROM:ia nykyisen dimensioilla ja nastajärjestyksellä varustettuna, joten EPROM:in koon muutos vaatisi huomattavia muutoksia nykyiseen laitteistoon. Tästä syystä ensisijainen ongelman ratkaisuyritys on vain yritys sopeuttaa projektin toisessa vaiheessa kirjoitettava koodi olemassaolevaan tilaan. Ellei tämä yksin auta, joudutaan muuttamaan projektin ensimmäisessä vaiheessa kirjoitettua koodia. Eräs mahdollisuus selviytyä tilanteesta olisi pakata ohjelma ennen polttamista ja purkaa se sitten dynaamisessa RAM:issa. Tällöin tosin joudutaan kasvattamaan käytössä olevan dynaamisen RAM:in kokoa, joka taas lisää laitteen yksikkökustannuksia. Tällä hetkellä käytetyn dRAM:in koko on 1 M. Jos ohjelma joudutaan purkamaan dRAM:issa, sitä tarvitaan noin puolet lisää.

Käytettävissä olevan EPROM:in rajallisuus on vaikuttanut huomattavasti linjatestausmoduulin arkkitehtuuriin. Useista prosesseista koostuvien moduulien sijasta on suosittu yksittäisiä prosessesseja missä vain mahdollista. Kunkin sanoman sisältämä informaatio on pyritty tiivistämään mahdollisimman pieneen tilaan. Esimerkiksi testien hallintaprosessin olisi hyvin voinut jakaa useampaan prosessiin, jolloin tehtävän työn olisi voinut myös jakaa useammalle ihmiselle. Samalla prosessin kompleksisuus olisi pienentynyt. Myös tilaajajohtojen mittausprosessin olisi voinut toteuttaa kahtena eri prosessina, jolloin varsinaiset mittaukset ja toisaalta laskenta olisivat olleet omina prosesseinaan. Tämä olisi nopeuttanut mittaustulosten käsittelyä huomattavasti, sillä nyt itse mittauksen aikana osa ajasta menee hukkaan prosessin odottaessa releiden kytkennästä aiheutuvien vaimenevien värähtelyjen (transienttien) poistumista. Sama koskee tietenkin myös valvontaprosessia.

Prosessorikapasiteetti

Signalointitilakoneen ajaminen vie tällä hetkellä noin 30-40% prosessorin kapasiteetista. Projektin toisessa vaiheessa mukaantuleva Dynacard-tilaajayksiköiden tapahtumien jatkuva kiertokysely tulee oletettavasti kuluttamaan prosessoritehoa myös huomattavasti. Koska ohjelma sisältää runsaasti aikariippuvia toimenpiteitä, niiden osalta saattaa olla odotettavissa ongelmia. On mahdollista, että aikariippuvaa koodia joudutaan jollakin tavalla optimoimaan.

Q1-yhteisohjelmistossa on jotkut ei-aikariippuvat prosessit (esim. käyttöliittymämoduuli Q1Mnu) kirjoitettu käyttäen timeouteja, eli sanomia odotetaan vain tietyn aikavakion ajan, jonka jälkeen ilmoitetaan virhetilanteesta. Q1_302-ohjelmistoa kirjoitettaessa ei ollut käsitystä sopivien aikavakioiden pituudesta, joten ne on virheellisesti määritelty liian lyhyiksi. Todennäköisesti aikavakioita joudutaan kasvattamaan tai ne poistetaan kokonaan.

ACM2-projektin vaiheistus

Kun suunnitellaan pitkiä projekteja on järkevää, että ne jaetaan useaan työvaiheeseen, sillä kaiken toiminnallisuuden toteuttaminen samanaikaisesti vaatisi usein kohtuuttomasti resursseja. Ongelmia tulee kuitenkin usein siinä vaiheessa kun projektin vaiheesta pitäisi siirtyä toiseen. Projektien vaiheittainen läpivienti vaatii tiukkaa kurinalaisuutta. Eräs vaihtoehto on suorittaa siirtyminen jäykästi siten, että vasta projektin ensimmäisen vaiheen päätyttyä aloitetaan toinen vaihe. Toinen vaihtoehto on aloittaa toinen vaihe jo ennen ensimmäisen vaiheen valmistumista. Etuna on lyhyempi kalenteriaika, vaarana sekava versionhallinta ja haittapuolena lisääntyvä resurssien tarve.

ACM2:ssa alkuperäinen ajatus oli toteuttaa molemmat vaiheet täysin toisistaan itsenäisinä projekteina. Ensimmäisessä vaiheessa toteutetun laitteiston pääosa, MCU-kortti, myöhästyi alkuperäisestä aikataulusta noin neljä kuukautta. Tästä syystä laitteistoriippuvaisia ohjelman osia päästiin testaamaan hyvin myöhään, josta aiheutui myös ensimmäisen vaiheen ohjelmistoprojektin myöhästyminen. Alunperin ohjelmistoprojektin toinen vaihe oli tarkoitus toteuttaa itsenäisenä ensimmäisestä vaiheesta. Tarkoitus oli, että kaikki ensimmäisen vaiheen tiedostot olisi kopiotu toisen vaiheen käyttöön, jossa niitä olisi itsenäisesti kehitetty ja päivitetty. Koska ensimmäisen vaiheen ohjelmisto oli ollut pitkään epästabiilissa tilassa, eivät varatut henkilöresurssit riittäneet muutosten päivittämiseen, joten päädyimme ainoastaan voimakkaasti muuttamiemme tiedostojen kopioimiseen.

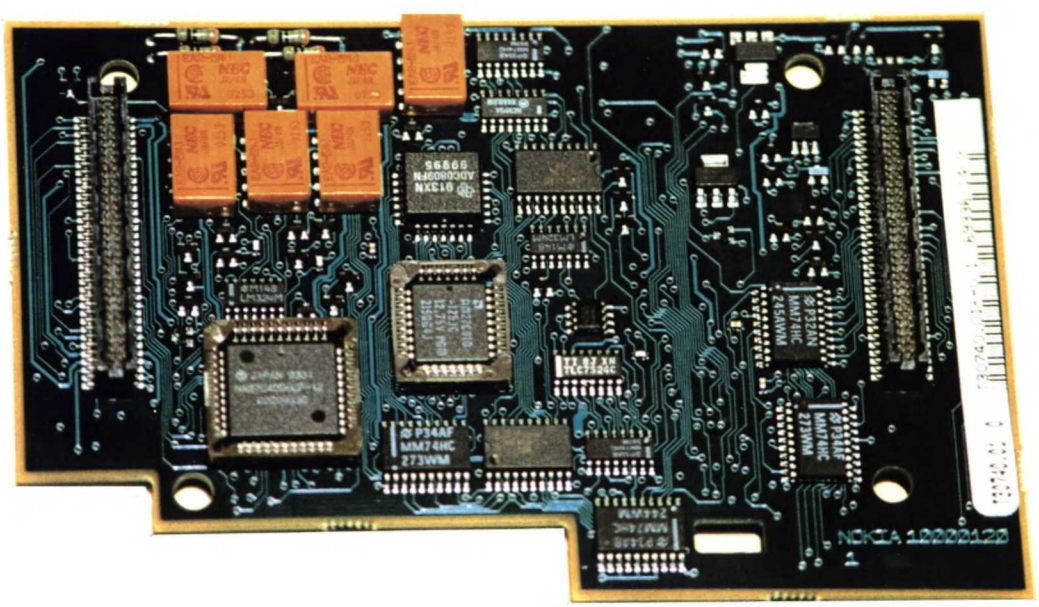
Parempi ratkaisu olisi ollut, jos toinen vaihe olisi alunperin suunniteltu jo yhdessä ensimmäisen vaiheen kanssa. Tällöin oltaisiin voitu jo ensimmäistä vaihetta toteutettaessa ottaa huomioon toisen vaiheen tarpeita, jolloin muutoksia ensimmäisen vaiheen ohjelmistoon olisi ollut vähemmän.

On lisäksi huomattava, että toteutettaessa projekti useammassa vaiheessa jotkut työtehtävät joudutaan toteuttamaan samanlaisina useampaan kertaan. Hyvä esimerkki tällaisesta työstä on systeemitestaus. Jokainen toimitettava kokoonpano, eli jokaisen projektivaiheen tulos on nimittäin testattava erikseen.

4.4 Lopuksi

ACM2-projektin toinen vaihe on tällä hetkellä integrointi- ja systeemitestausvaiheessa. Vaikka projekti ei vielä tällä hetkellä olekaan päättynyt, voidaan silti arvella sen onnistuneen erinomaisesti kaikista edellisessä luvussa luetelluista vastoinkäymisistä huolimatta. Projekti on onnistuttu viemään läpi hyvin, vaikka aikataulu- ja suoritusvaatimukset ovatkin olleet kireät. Otaksun, että en ole projektiryhmästämme ainoa, joka kokee oppineensa paljon uusia asioita paitsi linjatestauksesta ja digitaalisista multipleksereistä myös projektien johtamisesta ja projektityön organisoinnista. Uuteen tapaamme suunnitella ja dokumentoida yksikkötason ohjelmia (katso luku 4.1) voin vain todeta, että pienellä karsimisella siitä voidaan saada kohtuullisen hyvä käytäntö.

Asiakkaiden puolelta ACM2 on otettu varsin hyvin vastaan. Tulevaisuus vaikuttaa mielenkiintoiselta ja haastavalta access-verkon monien kehittymismahdollisuuksien ja -suuntien osalta.



Kuva 22. Linjatestausmoduuli

Lyhenteet

ACM2 *Access Multiplexer* eli tilaajamultiplekseri. Kanavoi 30 64 kbit/s liitäntää aikajakoisesti 2048 kbit/s johdolle. Käyttää ensisijaisesti prosessorittomia SUB-kortteja.

ACM-EXC ACM2:n keskuspään prosessoriton liitäntäyksikkö.

ACM-SUB ACM2:n tilaajapään prosessoriton liitäntäkortti.

AN *Access Network* eli tilaajaverkko. Teleoperaattorin ylläpitämä verkko, joka sijoittuu tilaajan ja keskuksen välille. Vastakohta *Core Network* eli ydinverkko. Vanhastaan tilaajaverkolla on tarkoitettu asiakkaan tiloissa olevaa, asiakkaan ylläpitämää verkkoa. Tässä työssä tällaista merkitystä ei ole.

AU *Alarm Unit* eli hälytysyksikkö. Yhtä verkon elementtiä, NE:tä vastaava looginen yksikkö, jota käytetään hälytystietojen hallintaan.

B *Bearer channel* eli B-kanava. 64k liitäntä.

BER *Bit Error Rate* eli bittivirhesuhde. Saadaan viallisten ja virheettömien bittien osamäärästä.

CCITT *Comité Consultatif International de Telegraphique et Telephonique*. ITU:n alainen, kansainvälinen, pääosin julkisia tietoliikennepalveluja antavien organisaatioiden muodostama järjestö, joka laatii suosituksia. Kokoontuu Genevessä. Vuodesta -93 alkaen nimeltään *Telecommunications Standardization Sector* (TSS).

CN *Core Network* eli ydinverkko. Suurinopeuksinen, optisella kuidulla toteutettu verkko, jota käytetään tiedon siirtoon puhelinkeskuksien välillä.

CODEC *Coder/Decoder*, kooderi/dekooderiyhdistelmä. Sisältää logiikan AD- ja DA-muunnosta, kvantisointia ja kompressointia varten.

CPU *Central Processing Unit*, keskusyksikkö.

CRC *Cyclic Redundancy Check*. Eräs lohkorakenteinen siirtovirheiden havaitsemistapa.

DB2 *Digital Branching Equipment*. 2048 kbit/s nopeudella toimiva digitaalinen haaroitinlaite.

DCN	*Data Communication Network.*
DI	*Data Interface*, DI-rajapinta eli dataliitäntä.
DM2	*Digital Multiplexer.* 2028 kbit/s:n digitaalinen multiplekseri. ACM2:n edeltäjä. Vaatii älykkäitä tilaajakortteja (nk. Dynacard -kortteja).
DN2	*Dynamic Node.* 2048 kbit/s nopeudella tai sen monikerroilla toimiva dynaaminen solmulaite.
dRAM	Dynaaminen RAM.
Dynacard	DM2-sukupolven tilaajakortti, joka sisältää prosessorin. Yksi Dynacard-kortti voi sisältää esim. viisi kappaletta 64 kbit/s liitäntää. ACM2:n kehityksen toisessa vaiheessa myös ACM2 pystyy tukemaan Dynacardeja.
EOC	*Embedded Operation Channel.* Osaa aikavälin nolla biteistä käyttämällä erotettu käyttökanava.
EPIC	*Extended PCM Interface Circuit* eli laajennettu PCM-liitäntäpiiri. Tulopuolelle voidaan liittää GCI-väylä ja lähtöpuolelle PCM-väylä.
EPROM	*Erasable Programmable Read Only Memory.* Luku- ja kirjoitusmuisti. Lukeminen tapahtuu sähköisesti. Kirjoittaminen tehdään polttamalla. Tallennetut tiedot säilyvät muistissa, kunnes ne poistetaan säteilyttämällä. Yleisimmin säteilytys tapahtuu UV-valolla.
EEPROM, E²PROM	*Electrically Erasable Programmable Read Only Memory.* Luku- ja kirjoitusmuisti. Sekä lukeminen että kirjoittaminen tapahtuu sähköisesti. Luku- ja kirjoituskertojen määrä on rajattu muistipiiristä riippuen joihinkin tuhansiin, jonka jälkeen piiri tuhoutuu.
ETSI	*European Telecommunications Standards Institute.* Eurooppalainen telealan standardoimisjärjestö.
F	Työasemarajapinta (TMN-konseptissa).
FE	*Functional Entity* eli toiminnallinen kokonaisuus. Käsittää tavallisimmin yhden laitteen; esim. yhden multiplekserin.
FTTA	*Fibre to the Apartment.* Optinen yhteys huoneistoon.
FTTB	*Fibre to the Building.* Optinen siirtotie rakennukseen.
FTTO	*Fibre to the Curb.* Optinen yhteys jakelukeskukseen.
FTTH	*Fibre to the Home.* Optinen siirtoyhteys kotiin.
GCI	*General Circuit Interface.*

ISDN *Integrated Services Digital Network.* Integroitujen palveluiden digitaalinen verkko, jossa peruskäyttäjälle tarjotaan kaksi 64 kbit:n B-kanavaa ja yksi merkinantoon tarkoitettu 16 kbit:n D-kanava.

ISO *International Organization for Standardization.* Kansainvälisten standardointiorganisaatioiden vuonna 1946 muodostama vapaaehtoinen elin. ISO:n toiminta jakaantuu teknisiin komiteoihin (TC) ja työryhmiin (WG).

ITU *International Telecommunication Union.*

kbit kilobitti, 1000 bittiä. Tietoliikenteessä usein 2^{10}.

LCN *Local Communication Network.*

LISC *Line Interface Setting and Control circuit.*

LTM *Line Test Module* eli linjatestausmoduuli. Laite, jonka suunnittelua ja toteutusta tässä työssä käsitellään.

LTU *Line Test Unit* eli linjatestausyksikkö. LTM:n edeltäjä. Kortilla on oma prosessori.

M Mega eli 10^6.

MCU Main Control Unit.

MD *Mediation Device,* välityslaite.

MUX2 Multiplekseri- ja kehyslukituspiiri.

NE *Network Element,* verkon elementti.

O&M *Operation & Maintenance,* käyttö ja ylläpito.

OPDN *Optic Passive Distribution Network.* Passiivinen jakeluverkko, jossa jakelutienä käytetään optista kuitua. Sama kuin PON.

OS *Operating System,* käyttöjärjestelmä. CCITT käyttää muotoa *Operations System.*

PA *Programmable Alarm* eli ohjelmoitava hälytys.

PCM Pulssikoodimodulaatio. Analoginen signaali kvantisoidaan ottamalla siitä näytteitä tietyllä taajuudella. Näin saadaan digitaalista PCM-signaalia.

PDH *Plesiochronous Digital Hierarchy,* plesiokroninen digitaalinen hierarkia. Käsittää tiedonsiirtonopeudet 2...140 Mbit/s. Vrt. SDH.

PON *Passive Optical Network,* valokuitutekniikkaan perustuva passiivinen jakeluverkko. Sama kuin OPDN.

POTS	*Plain Old Telephone Service*, eli konventionaalinen analoginen puhelinpalvelu.
PROM	*Programmable Read Only Memory*, eli ohjelmoitava lukumuisti. Muistin kirjoitus tapahtuu polttamalla ja lukeminen sähköisesti.
Ps	*Public Switching*, analogisen puhelimen liitäntäkanava.
PSTN	*Public Switched Telephone Network* eli yleinen puhelinverkko.
Q1, Q2, Q3, Qx	Rajapintoja siirtolaitteiden liittämiseksi tietoliikennelaitteiden hallintaverkkoon TMN:ään. Q1 on rajapinnoista suppein, Q3 laajin. Katso CCITT:n suositusta G.771.
Q1_302	Q1-tiedonsiirtoprotokollaa käyttävä MC68302-prosessorille kehitetty yhteisohjelmisto, joka tarjoaa sovellusohjelmille mm. tiedonsiirto- ja muistinkäsittelypalveluita.
RAM	*Random Access Memory*. Useita luku- ja kirjoitusoperaatioita salliva lyhytkestoinen muisti, jossa suoritettava ohjelmakoodi sijaitsee.
ROM	*Read Only Memory*, lukumuisti.
RSM2	*Remote Subscriber Multiplexer*. Kilpaileva nimiehdotus ACM2:lle.
SB	*Supervision Block*, hallintalohko. Käsittää tyypillisesti yhden laitteen osan, esim. yhden kanavakortin.
SDH	*Synchronous Digital Hierarchy*, synkroninen digitaalinen hierarkia. Tiedonsiirtonopeudet > 140 Mbit/s. Vrt. PDH.
SI	*Service Interface*, eli palvelurajapinta.
SICOFI	*Subscriber Interface Codec Filter Circuit.*
SLIC	*Subscriber Line Interface Circuit* eli tilaajan liitäntäpiiri.
ST	*Service Terminal*. Huoltopääte, joilla voidaan muuttaa multiplekserien asetuksia ja parametreja.
TMC	*Transmission Management Computer*. Tietokoneohjelmisto, jolla voidaan etäältä hallita siirtolaitteita.
TMN	*Telecommunications Management Network*. Tietoliikennelaitteiden hallintaverkko.
TMSP	*Transmission Management System Protocol*. Q1-rajapinnan toteuttava verkonhallintaprotokolla.

TP	*Termination Point.*
TTP	(V5) *Trail Termination Point.*
VB	*Battery Voltage* eli pariston jännite.
WS	*Workstation* eli työasema.
X	Rajapinta TMN:stä muihin verkkoihin (toiseen TMN:ään).
2-w	*Two-wire*. Kaksijohdinyhteys. Puhelinlaitteesta lähtevä johdinyhteys, joka sisältää a- ja b-johtimet.

Lähdeluettelo

[1] Awad, Maher. *Design of Subscriber-line Test Unit in Telephone Networks.* Diplomityö. Teknillinen korkeakoulu, Sähkötekniikan osasto. Helsinki 1989. 67 s.

[2] Awad, Maher. *Line Testing Unit. Product Description.* Nokia Telecommunicationsin sisäinen dokumentti, no E00844812QE-00. 1992, 14 s.

[3] CCITT. Sininen kirja, osa III.4. Suositus G. 703. *Physical/Electrical Characteristics of Hierarchical Digital Interfaces.* Geneve 1972, 31 s.

[4] CCITT. Sininen kirja, osa III.4. Suositus G.704. *Synchronous Frame Structures Used at Primary and Secondary Hierarchical Levels.* Malaga-Torremolinos 1984, 21 s.

[5] CCITT. Sininen kirja, osa III.4. Suositus G.732. *Characteristics of Primary PCM Multiplex Equipment Operating at 2048 kbit/s.* Geneve 1972, 6 s.

[6] CCITT. Sininen kirja, osa III.4. Suositus G.771. *Q-interfaces and Associated Protocols for Transmission Equipment in the Telecommunications Management network (TMN).* Melbourne, 1988, 27 s.

[7] CCITT. Suositus G.826. *Error Performance Parameters and Objectives for International Constant Bit Rate Digital Paths at or above the Primary Rate.* Geneve 1993, 15 s.

[8] CCITT. Sininen kirja, osa IV.1. Suositus M.30. *Principles for a Telecommunications Management Network*. Geneve, 1989. 40 s.

[9] CCITT. Valkoinen kirja, osa II.1. Suositus M.3010 (luonnos). *Principles for a Telecommunications Management Network*. Pariisi, 1992. 85 s.

[10] CCITT. COM X1-R 175-E. Suositus ehdotukseksi Q.542. *Digital Exchange Design Objectives - Operations and Maintenance*. 3 s.

[11] CCITT. Työryhmä XI/3. *Maintenance of Analogue Subscriber Lines Connected to Digital Exchanges*. Geneve, 1983. 21 s.

[12] CMT++, Version 1.2. *Complexity Measures Tool for C++ (and C)*. Käyttökäsikirja. Nokia Data Systems, C Tools, 1989. Dokumentti no XT9220. 18 s.

[13] *DB2 Training Manual I*. Nokia Telecommunications, koulutusmateriaali. 1991, 20s.

[14] DE/SPS. ETSI:n standardi (luonnos). *Signaling Protocols and Switching - Q3 Interface at the Access Network for Configuration Management of V5 Interfaces and Accociated User Ports*. DE/SPS 3013.1. Bristol 1993, 58 s.

[15] *DM2 Training Manual I*. Nokia Telecommunications, koulutusmateriaali. 1991, 9 s.

[16] *DN2 Training Manual I*. Nokia Telecommunications, koulutusmateriaali. 1992, 28 s.

[17] Eriksson Kjell, Nygaard Tore, Pohjanpalo Hannu, Pärnänen Matti ja Sairila Petri. *C Style Guide*. Nokia Telecommunicationsin sisäinen dokumentti. 1991, 35 s.

[18] ETR. ETSI:n tekninen raportti (luonnos). *The Use of Singlemode Fibre in the Access Network.* DTR TM 3003. 1992, 34 s.

[19] Halme, Seppo J. ja Rahko, Kauko. Tietoliikennetekniikan perusteet. Otakustantamo no 471 1981. 328 s.

[20] Jyllilä, Hannu. *ACM2 Dynacard Compatibility. Functional Specification.* Grapple Oy. 1993, 26s.

[21] Kaitala, Kimmo. *ACM2 Phase 2 Quality Plan.* Nokia Telecommunicationsin sisäinen dokumentti. 1993, 17 s.

[22] Kaitala, Kimmo ja Laanti, Maarit. ACM2. *Line Testing Functional Specification.* Nokia Telecommunicationsin sisäinen dokumentti no E10007520QE_00. 1993, 35 s.

[23] Laanti, Maarit ja Pager, Istvan. ACM2. *Line Testing Technical Specification.* Nokia Telecommunicationsin sisäinen dokumentti no E1007684QE_00. 1993, 34 s.

[24] Lassus, Anette. *Design of a Message-based Software Implementing a Telecommunications Management Network (TMN) Protocol.* Diplomityö. Teknillinen korkeakoulu, Sähkötekniikan osasto. Helsinki 1992, 64 s.

[25] Leivo, Heikki. Relepuhelintekniikasta optiseen teletekniikkaan. Teknillinen korkeakoulu, teletekniikan laboratorio. 92/1. Otaniemi 1992. 84 s.

[26] Longstaffe, William. *ACM SUB Hardware/Software Specification.* Nokia Telecommunicationsin sisäinen dokumentti no T30705008OE. 1993, 44 s.

[27] Lundström, Pekka. *CTB/CTC module testing tools* (V1.6). Julkaisematon. 1993, 15 s.

[28] Lähteenmäki, Vesa. *Line Test module for ACM2 Technical Specification.* Nokia Telecommunicationsin sisäinen dokumentti no T3074000QE_00. 1993, 14 s.

[29] Metsälä, Esa. *ACM2 Functional Specification.* Nokia Telecommunicationsin sisäinen dokumentti. 1992, 57 s.

[30] Nordman, K. PCM-puheensiirtojärjestelmien tarve. PCM-järjestelmät. Insinöörijärjestöjen koulutuskeskuksen julkaisu 18/73. Helsinki 1973. 200 s.

[31] Norri, Timo. *ACM2 Primary Multiplex Equipment. Product Description.* Nokia Telecommunicationsin sisäinen dokumentti. 1993, 22 s.

[32] OS68 käyttökäsikirja. Enea Data 1992. 290 s.

[33] Riskilä, Olavi. *ACM2. Bus Interface Specification.* Nokia Telecommunicationsin sisäinen dokumentti. 1992. 45 s.

[34] Sivertsen, Sigurd. *New Access Network Structure for Metropolitan Areas in Norway.* Norjan telelaitos. 1992. 6 s.

[35] Tammelin, Markku. Digitaalisen puhelinverkon siirtopalvelut. HPY Data:n koulutusmateriaali 18.1.1990. 9 s.

[36] Teppo, Pentti. Siirtojärjestelmät tietoliikennetekniikassa. KS-tietokirjat. Huhmari, 1992. 454 s. ISBN 951-95399-9-9.

[37] Volotinen, Vesa. Aikajakoisen tekniikan perusteet. WSOY Porvoo, 1984. 180 s. ISBN 951-0-12151-7.

[38] Väärä, Hannu. *ACM2. Software Phase 2 Functional Specification.* Nokia Telecommunicationsin sisäinen dokumentti. 1993a, 43 s.

[39] Väärä, Hannu. *ACM2. Software Phase 2 Technical Specification*. Nokia Telecommunicationsin sisäinen dokumentti. 1993b, 84 s.

Liite1: Linjatestaustietokanta E²PROM:ssa

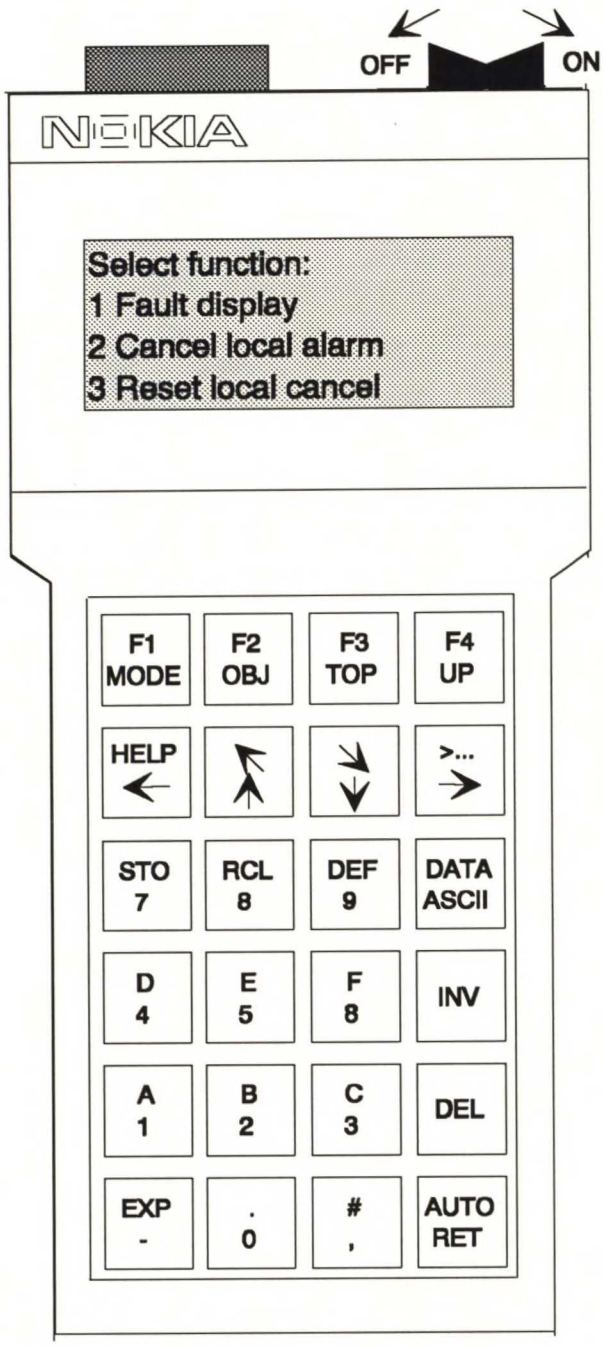
```
/*********************************************************
 * Nokia Telecommunication
 * Transmission Products
 * Access Network Group
 *
 * Subsystem header file
 *
 * Subsystem    : LTM in ACM2 phase 2
 *
 * File         : LtmSubstrId.h
 *
 * Created      : July 14 1993 Maarit Laanti
 *
 * Modified     :
 *
 * Description : This file  defines the substructure for
 *               Line Testing.
 *               Controls:
 *                       byte 0 stands for control output 1
 *                       byte 1 stands for control output 2
 *                       byte 2 stands for control input 1
 *                       byte 3 stands for control input 2
 *
 *********************************************************/

#ifndef LTMSUBSTR
#define LTMSUBSTR

#ifndef NUMBER_OF_CHANNELS
#define NUMBER_OF_CHANNELS 30
#endif

struct PeriodicTest
{
    UWORD uwMonthStartTime;
    UWORD uwDayStartTime;
    UWORD uwHoursStartTime;
    UWORD uwMinutesStartTime;
    UWORD uwDayPeriod;
    UWORD uwHoursPeriod;
    UWORD uwMinutesPeriod;
};

struct MeasPara {
    WORD   Enable;
    WORD   LowerLimit;
    WORD   UpperLimit;
    WORD   Alarm;
    WORD   NumberOfSamples;
};

struct SvMeasPara {
    struct MeasPara BatteryVoltage;
    struct MeasPara ExtVoltage1;
    struct MeasPara ExtVoltage2;
    struct MeasPara ExtVoltage3;
    struct MeasPara ExtVoltage4;
```

```
    struct MeasPara RingingCurrent;
    struct MeasPara AdcDacLoop;
    struct MeasPara ControlLoop;
};

struct SubMeasPara {
    struct MeasPara DcVoltage;
    struct MeasPara AcVoltage;
    struct MeasPara Resistance;
    struct MeasPara Capacitance;
    struct MeasPara SingVoltage;
    struct MeasPara IntCircVoltage;
};

struct LtmSubstrId
{
    struct      SubMeasPara     SubParameters;
    struct      SvMeasPara      SvParameters;
    struct      PeriodicTest    SubPeriod;
    struct      PeriodicTest    SvPeriod;
    BYTE        Controls;
    UWORD       uwTimeout;  /* timeout for exte usage */
    BYTE        PermissionToTest[NUMBER_OF_CHANNELS];

};


#endif /* LTMSUBSTR */
```

Liite 2: Esimerkkisanoma linjatestausohjelmistosta

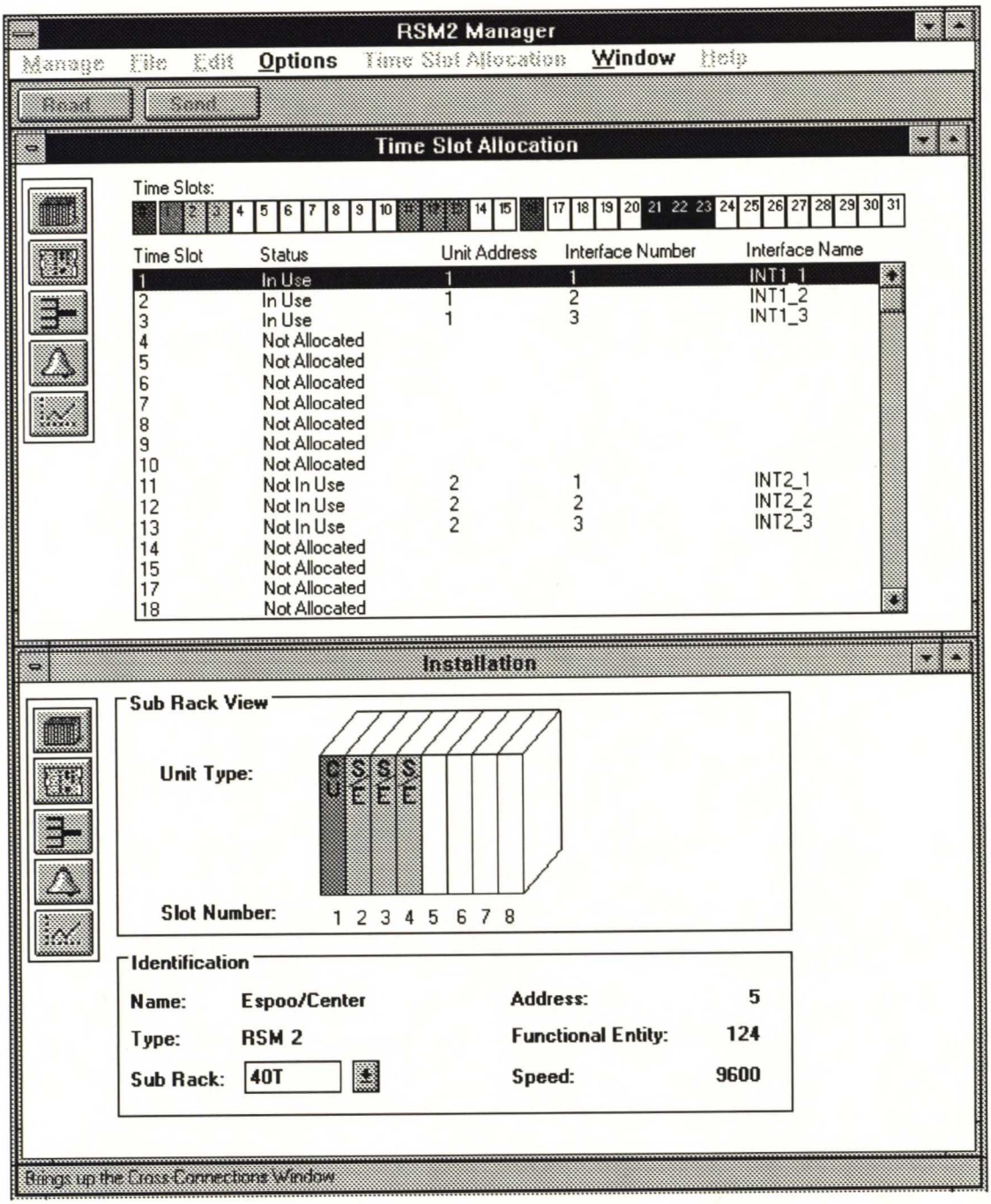
```
/*
 * Subsystem message specification
 *
 * Subsystem:       ACM
 *
 * File:            LtmSubUserCmd.h
 *
 * Created:         JUL 8 1993 M.Laitinen
 *
 * Modified:        Date        Author      Comments
 *
 * Description:     Commands Test Manager to carry out
 *                  tests or send results on a single
 *                   subscriber line.
 */

 /* definitions for SubUserCmd */

#define START_SUB_MEASUREMENT                    1
#define VOICE_LOOP_ANALOG                        2
#define VOICE_LOOP_DIGITAL                       3
#define VOICE_LOOP_REFL_SIG_ANAL                 4
#define VOICE_LOOP_DAC                           5
#define VOICE_LOOP_ADC                           6
#define CONNECT_EXT_EQUIP                        7
#define DISCONNECT_EXT_EQUIP                     8
#define READ_SUB_RESULTS                         9

/* definitions for CmdStatus */

#define REQUEST_ACCEPTED                         8
#define TESTER_BUSY                              9

#define LTM_SUB_USER_CMD (0x0401)
/* !-SIGNO( struct LtmSubUserCmd )-! */

struct LtmSubUserCmd
{
 SIGSELECT   SigNo;
 BYTE        SubUserCmd;
 BYTE        SelectChannel;
 BYTE        CmdStatus;

      };
```

Liite 3: Huoltopääteliittymä

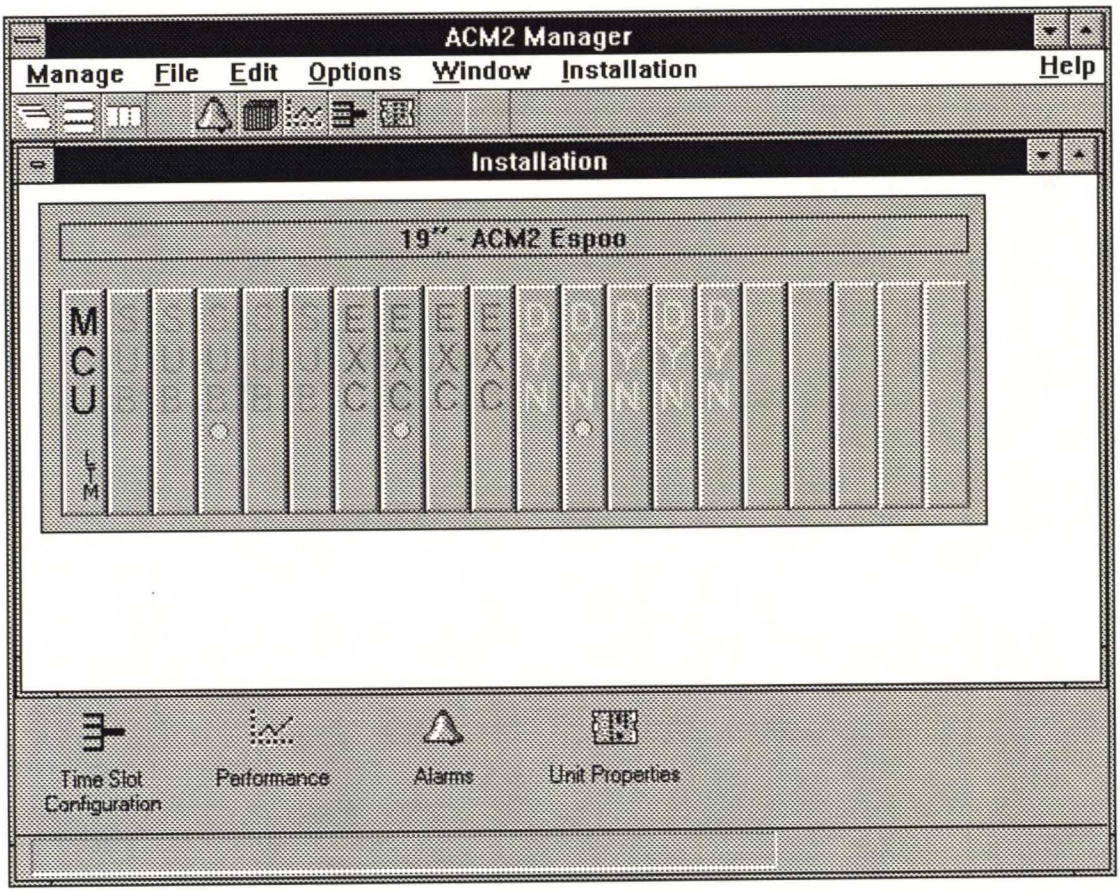


Liite 3, kuva 1. Huoltopääte

Liite 4: ACM2 Node Managerin käyttöliittymä

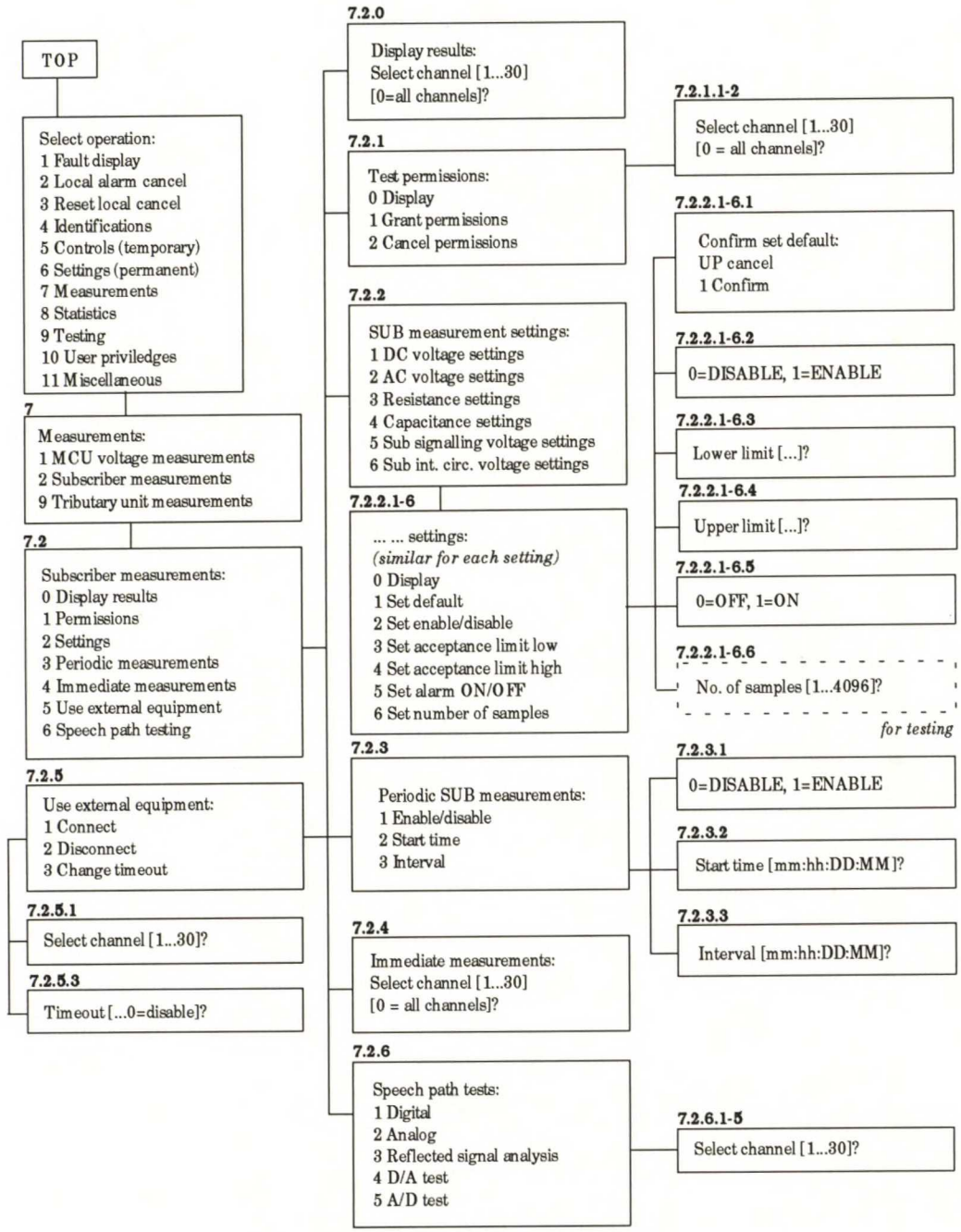


Liite 4, kuva 1. Aikavälien allokointi ACM2 Node Managerilla

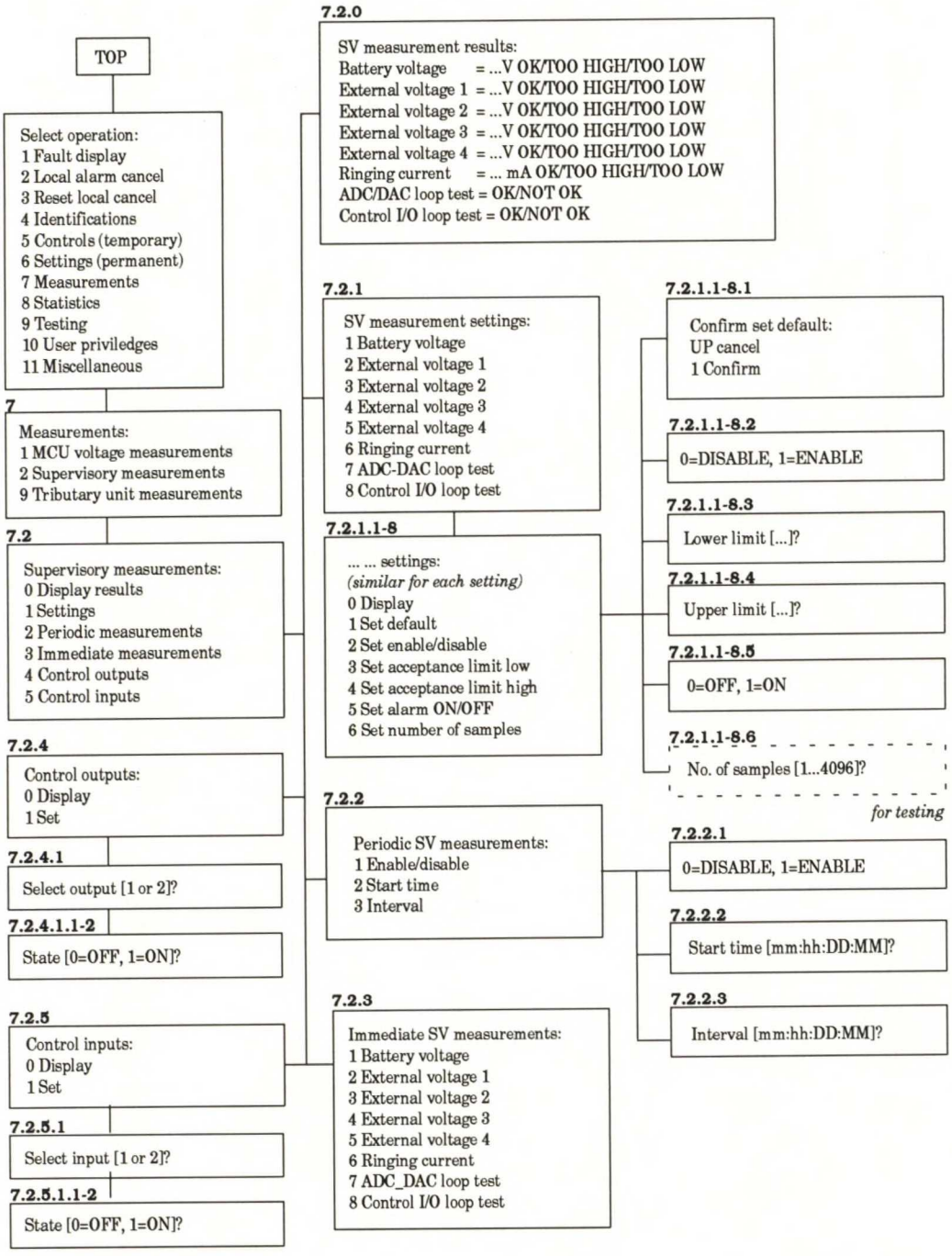


Liite 4, kuva 2. Installointi ACM2 Node Managerilla. Huomaa LTM:n sijainti MCU:lla.

Liite 5: Linjatestausmoduulin menurakenne



Liite 5, kuva 1. Linjamittausten menuvalinnat, FE = 0



Liite 5, kuva 2. Valvontamittausten menuvalinnat, FE = 1

Liite 6: Menurakenne ja dynaaminen prosessi

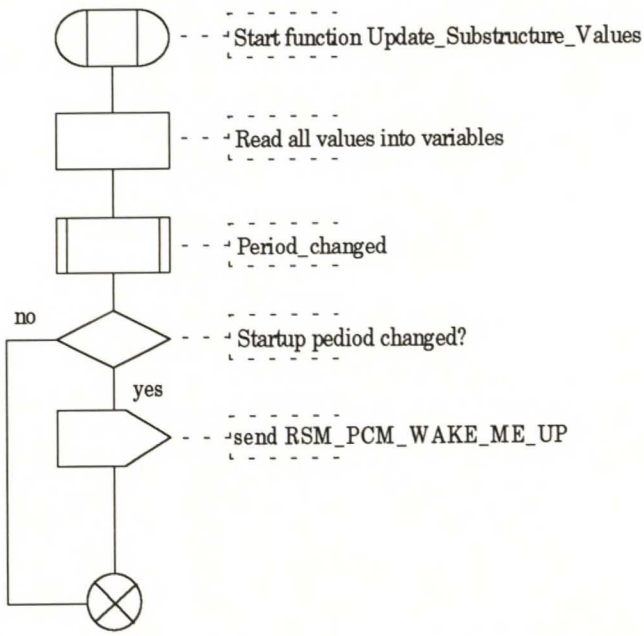
```
/****** Immediate SV measurements ******/
const CHAR *asFtSv_3[DEFINED_LANGS][13]=
{{
"Battery voltage       = %.1f V  %s\r",
"External voltage 1    = %.1f V  %s\r",
"External voltage 2    = %.1f V  %s\r",
"External voltage 3    = %.1f V  %s\r",
"External voltage 4    = %.1f V  %s\r",
"Ringing current       = %.0f mA %s\r",
"ADC-DAC loop test     = %s\r",
"Control I/O loop test = %s\r",
"OK", "NOT OK", "TOO HIGH", "TOO LOW","NOT DONE"}};

const BYTE_BRANCH rBtSv_3[] = {
{1,8,PAR,NOT_PROT,PROC,LTM_SV_3P,PRIOR,asFtSv_3,NO_LINK,
NULL}
};

const MENUNODE rNodeSv_3 = {NODE, BYTE_PAR, 1, rBtSv_3 };

const MENUTEXT rMtSv_3 ={ DOWN, {
"Immediate SV measurements:\r\n\
1 Battery voltage\r\n\
2 External voltage 1\r\n\
3 External voltage 2\r\n\
4 External voltage 3\r\n\
5 External voltage 4\r\n\
6 Ringing current\r\n\
7 ADC-DAC loop test\r\n\
8 Control I/O loop test\r"}};

/* ***** Periodic measurements ******/
/* enable/disable */
const BYTE_BRANCH rBtSv_2_1[] = {
{ 0,1,
PAR,NOT_PROT,PROC,LTM_SV_2P,PRIOR,asFtSvTmpl,NO_LINK,NULL}
};

const MENUNODE rNodeSv_2_1 = {NODE, BYTE_PAR, 1,
rBtSv_2_1};
```

Liite 6, kuva 1. Esimerkki menupuun rakenteen toteutuksesta

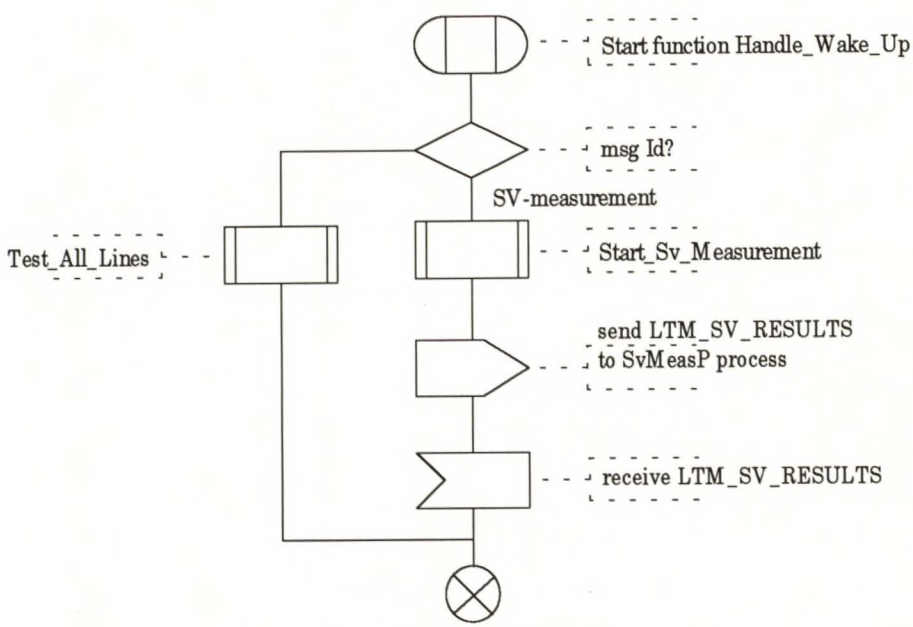
```
/*******************************************************
*             OS_PROCESS(LTM SV_1P)
*******************************************************
* DESCRIPTION: This menuprocess displays results of
*                 measurements.
* INPUT:
* sAnsw                 Pointer to Answer text string
* msgIn                 Message received from Q1 menumodule
* OUTPUT:
* sTemplate              Answer templates
* msgOut                 Message send to Q1 menumodule
* LITERALS:
* USED BY: Menubranch 0
* STATUS : W (W=work T=test R=ready)
*******************************************************/
OS_PROCESS(LTM_SV_1P)
{
   CHAR *sAnsw;
   CHAR *sTemplate;
   MESSAGE *msgIn;
   MESSAGE *msgOut;
   MESSAGE *msgSvResults;
   MESSAGE *msgSvUserCmd;
   static SIGSELECT aResu[] = {1, LTM_SV_RESULTS};
   float afMeasure[6];
   char *apStatus[SV_STAT_TAB_SIZE];
   PROLOGUE
   msgSvUserCmd = alloc(sizeof(struct LtmSvUserCmd),
LTM_SV_USER_CMD);
   msgSvUserCmd->rLtmSvUserCmd.SvUserCmd = READ_SV_RESULTS;
   send(&msgSvUserCmd, LtmManagerP_);
   msgSvResults = receive(aResu);
   HandleAllResults(msgSvResults, afMeasure, apStatus,
(char **)msgIn->rQ1MnuIfParF.sAnswTemp);
   send(&msgSvResults,LtmManagerP_);
   sprintf(sAnsw,sTemplate,afMeasure[0],apStatus[0],
                           afMeasure[1],apStatus[1],
                           afMeasure[2],apStatus[2],
                           afMeasure[3],apStatus[3],
                           afMeasure[4],apStatus[4],
                           afMeasure[5],apStatus[5],
                           apStatus[6],
                           apStatus[7]);
   EPILOGUE
}
```

Liite 6, kuva 2. Esimerkki dynaamisesta menuprosessista

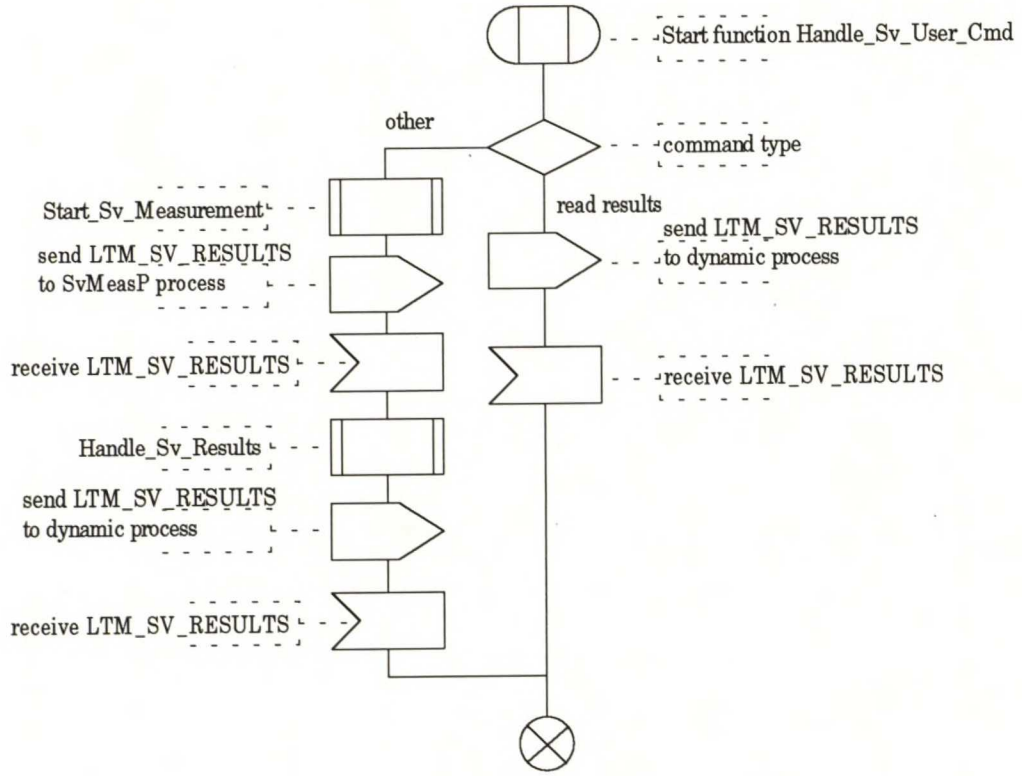
Liite 7: SDL-kaavio testien hallintaprosessista



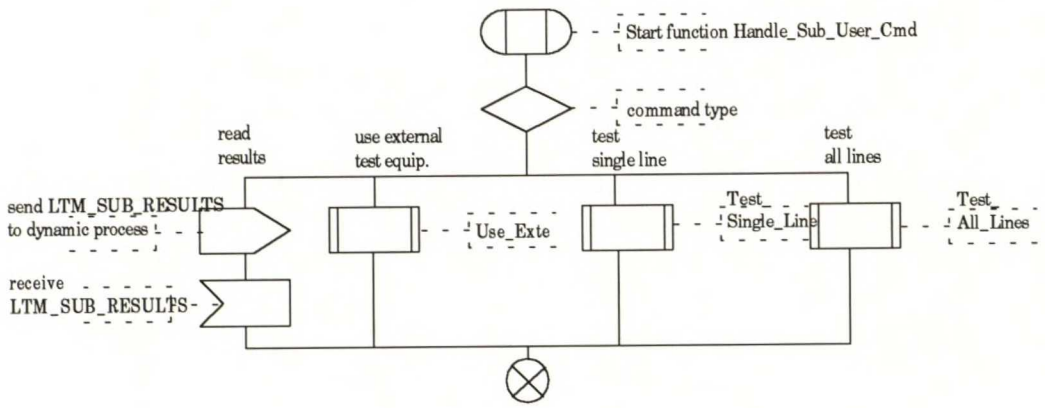
Liite 7, kuva 1. Funktio Update_Substructure_Values



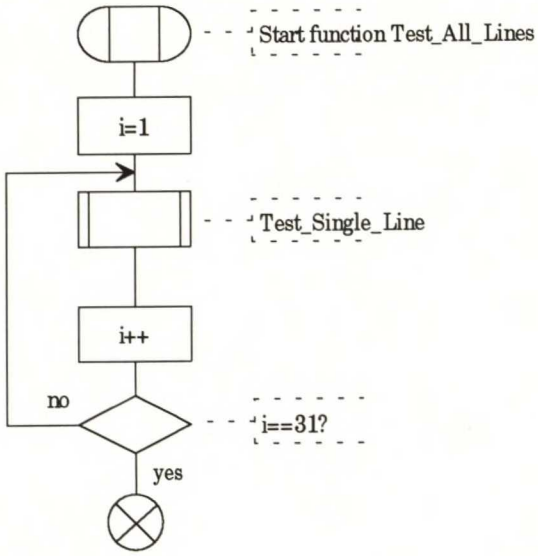
Liite 7, kuva 2. Funktio Handle_Wake_Up



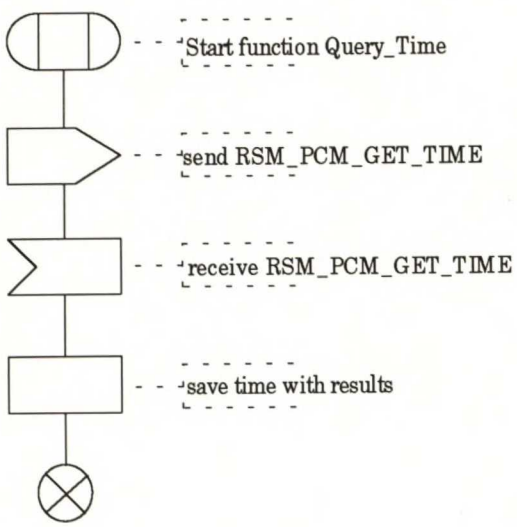
Liite 7, kuva 3. Funktio Handle_Sv_User_Cmd



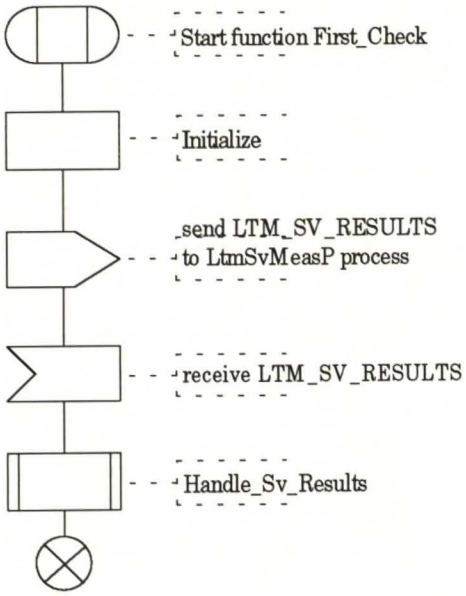
Liite7, kuva 4. Funktio Handle_Sub_User_Cmd



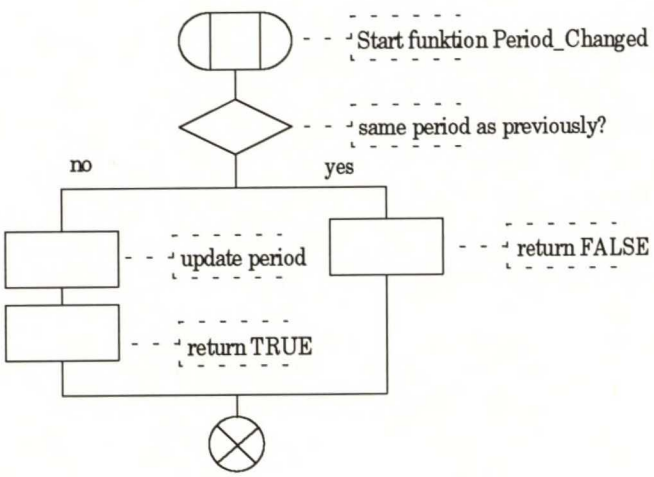
Liite 7, kuva 5. Funktio Test_All_Lines



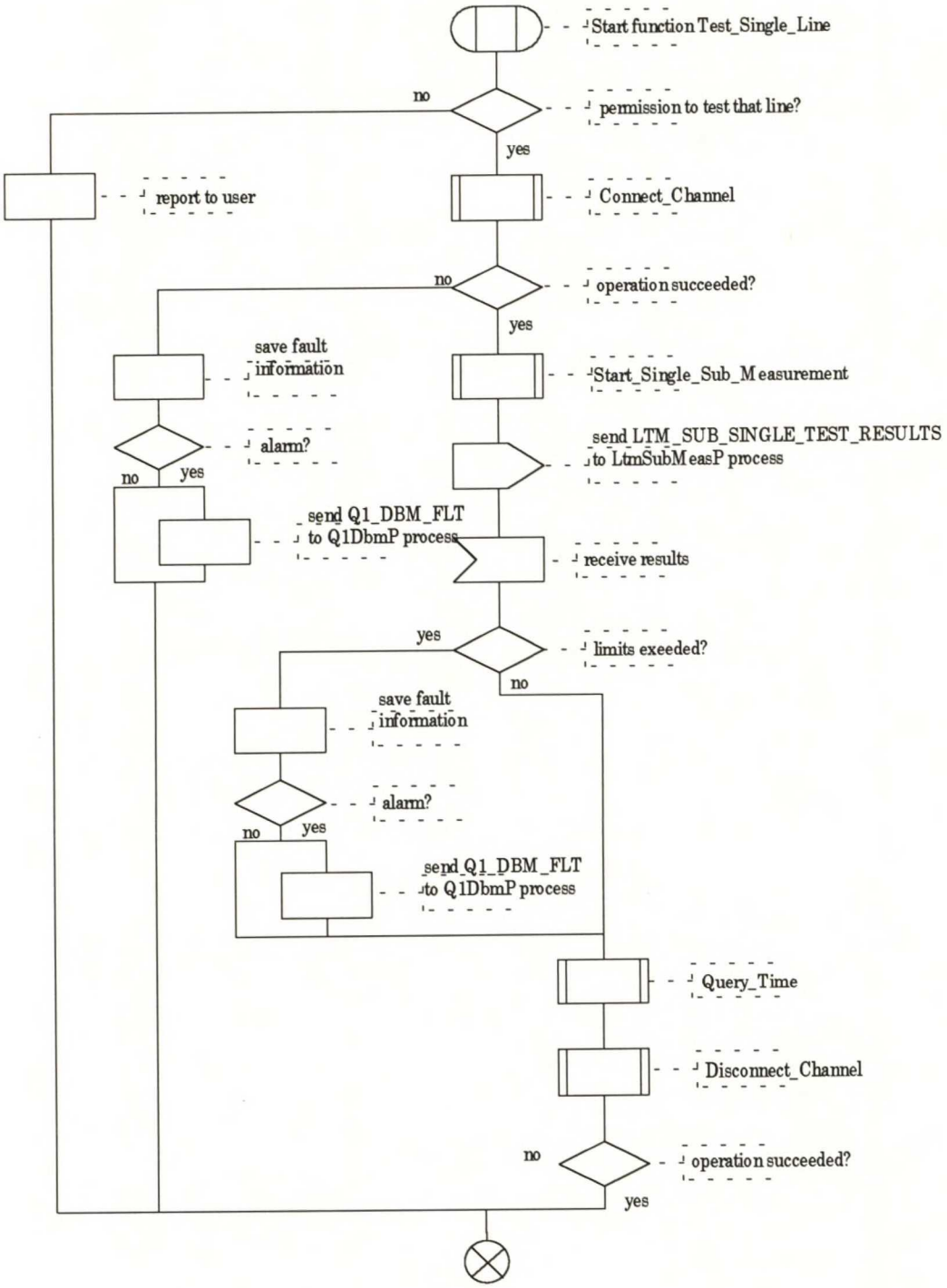
Liite 7, kuva 6. Funktio Query_Time



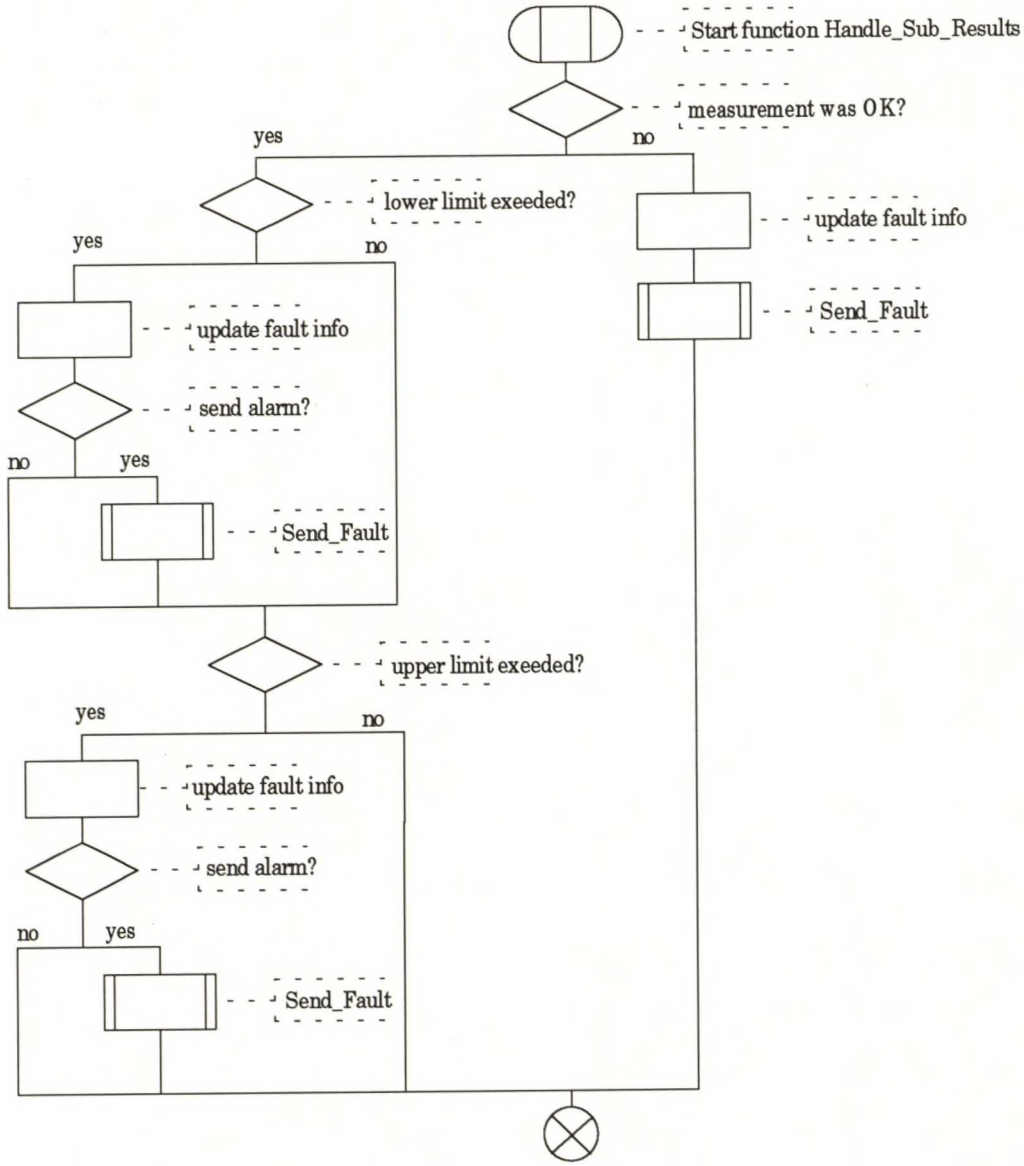
Liite 7, kuva 7. Funktio First_Check



Liite 7, kuva 8. Funktio Period_Changed

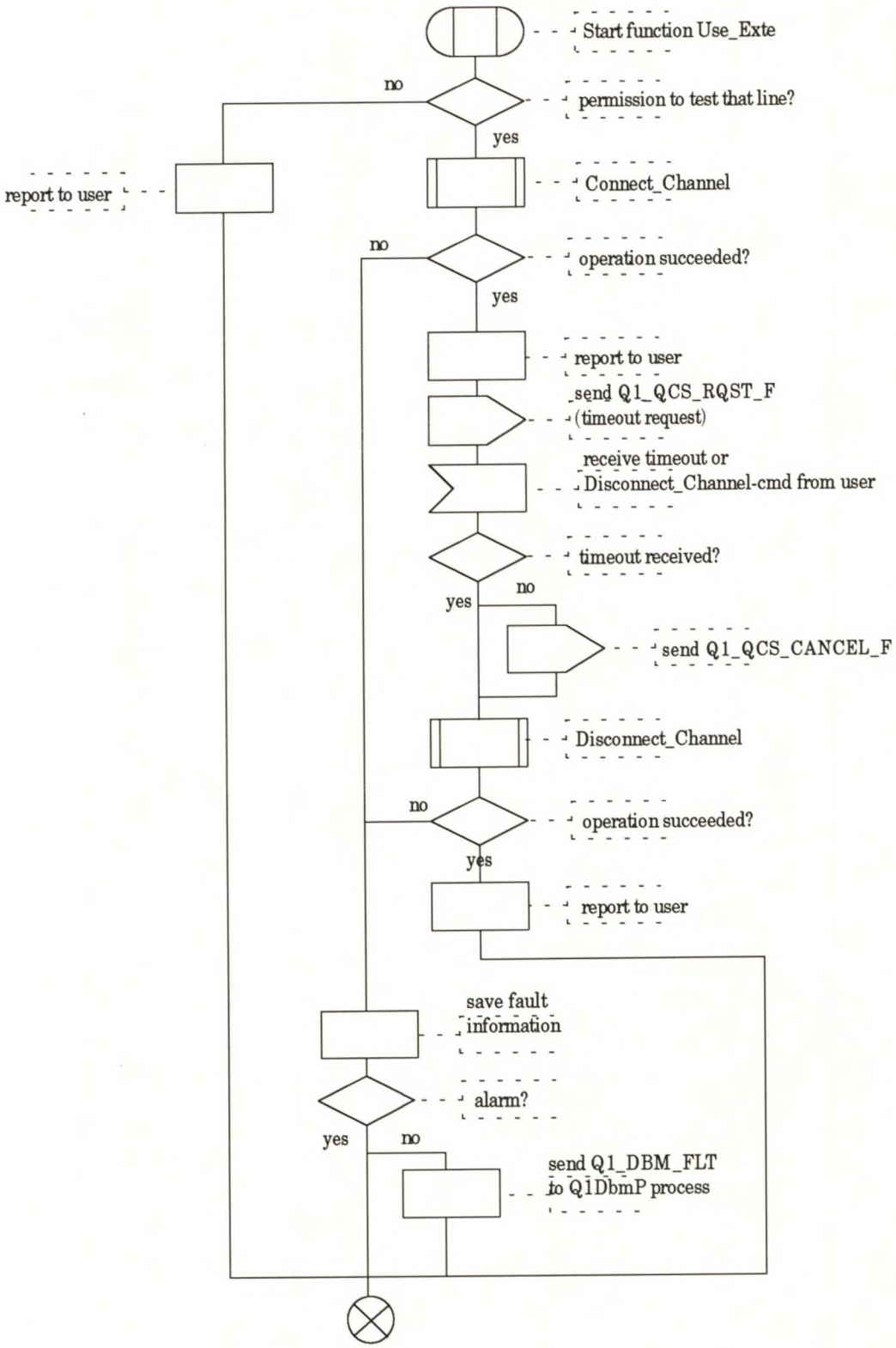


Liite 7, kuva 9. Funktio Test_Single_Line

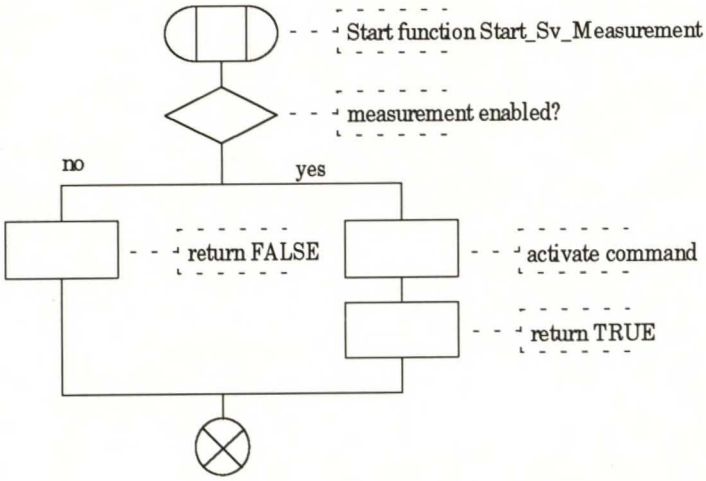


Liite 7, kuva 10. Funktio Handle_Sub_Results [1]

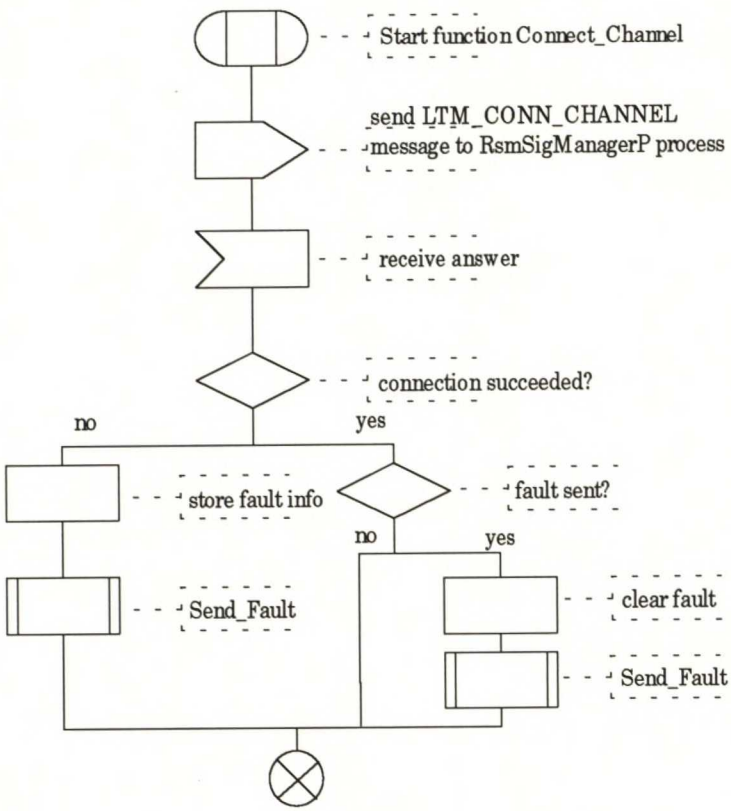
[1] *Funktio Handle_Sv_Results on samanlainen.*



Liite7, kuva 11. Funktio Use_Exte



Liite 7, kuva 12. Funktio Start_Sv_Measurement [2]

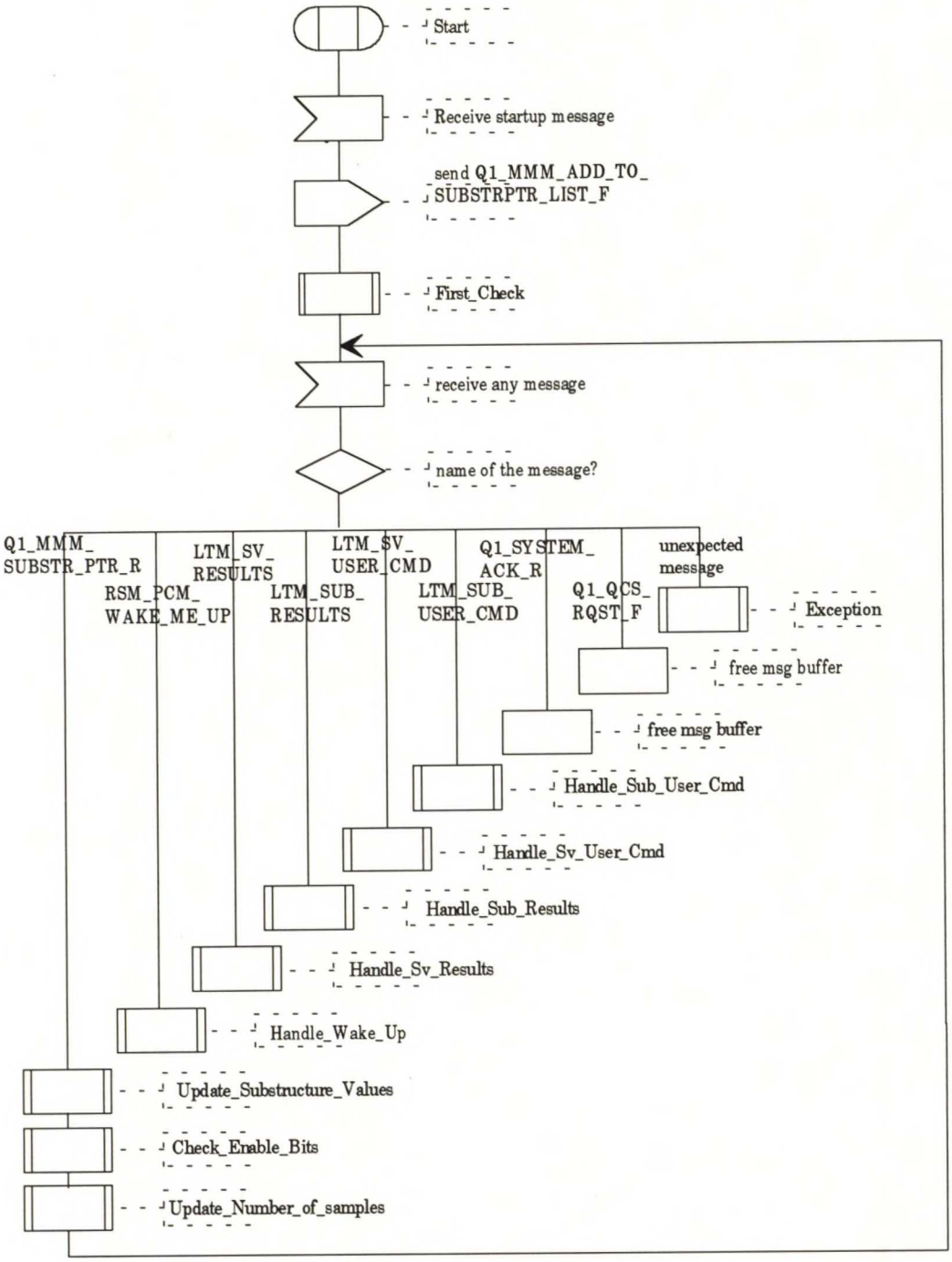


Liite 7, kuva 13. Funktio Connect_Channel [3]

---

[2] *Funktio Start_Single_Sub_Measurement on muuten samanlainen kuin Start_Sv_-Measurement, paitsi että sillä ei ole paluuarvoa.*

[3] *Funktio Disconnect_Channel on samanlainen.*



Liite7, kuva 14. Pääohjelma [4]

[4] *Funktiot Check_Enable_Bits ja Update_Number_Of_Samples sisältävät vain yksinkertaisia sijoituksia.*

Liite 8: Esimerkkifunktio testien hallintaprosessista

```
/*********************************************************
* Subsystem     :     LTM in RSM2 phase 2
* File          :     LtmManagerP.c
* Created       :     Jul 9  1993 Mahi (= Maarit Laanti)
* Modified      :
* Process       :     LtmManagerP
* Description   :     This is a process that handles the
*                     general test management i.e.
*                     permissions, connections, results etc.
*
*********************************************************/


static void Test_Single_Line(UWORD uwChannel, MESSAGE *msgIn,
                                MESSAGE *msgSubResults,
                                MESSAGE *msgConnChannel,
                                MESSAGE *msgSingleSubResults)
/* This function is used to test a line */
{
    if (SUB_PARAMETERS.PermissionToTest[uwChannel] == FALSE) {
        msgSubResults->rLtmSubResults.SubStatus[uwChannel] =
        CHAN_NO_PERMISSION;
         if (msgIn != NULL) {
              msgIn->rLtmSubUserCmd.CmdStatus =
                    CHAN_NO_PERMISSION;
              send(&msgIn, (sender(&msgIn)));
           } /* msgIn not null */
        return;
    } /* if permission */
    if (Connect_Channel(uwChannel, msgConnChannel) == FALSE) {
        if (msgConnChannel->rLtmConnChannel.uwConnStat ==
              CHAN_NOT_CONNECTED) {
              msgSubResults->rLtmSubResults.SubStatus[uwChannel]
                    = CHAN_NOT_CONNECTED;
              if (msgIn != NULL) {
                    msgIn->rLtmSubUserCmd.CmdStatus =
                          CHAN_NOT_CONNECTED;
                    send(&msgIn, (sender(&msgIn)));
              } /* msgIn not null */
        } /* channel not connected at all */
        else if (msgConnChannel->rLtmConnChannel.uwConnStat ==
                    BUSY){
              if (msgIn != NULL) {
                    msgIn->rLtmSubUserCmd.CmdStatus =
                          CHAN_NOT_IDLE;
              send(&msgIn, (sender(&msgIn)));
              } /* msgIn not null */
        } /* line not idle */
```

```
        else if (msgConnChannel->rLtmConnChannel.uwConnStat ==
                        FAILED){
                if  (msgIn != NULL) {
                        msgIn->rLtmSubUserCmd.CmdStatus =
                                CHAN_STUCK;
                        send(&msgIn, (sender(&msgIn)));
                } /* msgIn not null */
        } /* connection failed */
        return;
    } /* if not connected */
    if ((msgIn != NULL) && (msgIn->rLtmSubUserCmd.SubUserCmd ==
                CONNECT_EXT_EQUIP)) {
                msgIn->rLtmSubUserCmd.CmdStatus =
                        REQUEST_ACCEPTED;
                send(&msgIn, (sender(&msgIn)));
                return;
 } /* if exte measurement no connection to LtmSubMeasP needed */
         Start_Single_Sub_Measurement(msgIn,
         msgSingleSubResults);
                if (msgIn != NULL) {
                msgIn->rLtmSubUserCmd.CmdStatus =
                        REQUEST_ACCEPTED;
                send (&msgIn, (sender(&msgIn)));
        } /* if not null */
        send (&msgSingleSubResults, LtmSubMeasP_);
        msgSingleSubResults = receive(aSingleSubResults);
        if (msgSingleSubResults-
>rLtmSingleSubTestResults.SubMeasStatus
 != MEAS_INTERRUPTED)
 Disconnect_Channel(uwChannel, msgConnChannel);
    Handle_Sub_Results(msgSingleSubResults, uwChannel,
msgSubResults);
    Query_Time(msgSubResults, uwChannel);

} /* function Test_Single_Line */
```