UNIVERSITETET I OSLO
Fysisk institutt

Evaluering av EMCal Trigger Region Unit (TRU) for bruk i ALICE PHOS detektor

Masteroppgave

(30 studiepoeng)

Georg Jansen

07.06 2011

SAMMENDRAG

I CERN ligger verdens største partikkelakselerator hvor ALICE er en av de fire store eksperimentene som måler energi når partikler kolliderer i tilnærmet lysets hastighet. ALICE detektoren består av 13 subdetektorer og blant disse så finner man PHOS og EMCal. EMCal Trigger Region Unit (TRU) baserer seg på PHOS elektronikken, men har en nyere FPGA med større plass og hurtigere dataoverføring. I denne oppgaven skal man undersøke om det er muligheter for å bruke et EMCal TRU kort i en PHOS detektor. Dette kortet står for blant annet generering av en hurtig avgjørelse L0 som vurderer hvorvidt det er relevant data i kollisjonen. Grunnet oppgavens begrensninger med hensyn til tid og lengde er det tatt utgangspunkt i Fake ALTRO delen av TRU kortet. Fake ALTRO er relevant fordi en utlesning fra disse bufferne kan gi informasjon om TRU også kan brukes for PHOS detektoren. Først presenteres PHOS elektronikken og triggersystemet, deretter EMCal programvaren til TRU kortet, og kort om utviklingsverktøyet Xilinx ISE. Videre er EMCal TRU kortet testet i en PHOS lab ved Universitetet i Bergen. Data har blitt lest ut fra Fake ALTRO bufferen og vist at en kommunikasjon mellom PHOS elektronikken og EMCal TRU er oppnådd. Det ser også ut til at det blir lest ut riktig dataformat fra riktig antall kanaler i bufferen. Disse innledende resultatene tyder på at det kan være mulig å bruke EMCal TRU i PHOS detektoren.

Forord

Jeg vil takke for all hjelpen jeg har fått fra Lijiao Liu ved Universitet i Bergen som lot meg få låne testlab-en og all støtte for arbeidet med TRU kortet, og at jeg fikk utkastet til hennes Ph.D avhandling. Jeg vil takke CERN og EMCal gruppen med Jiri Kral i front for lån av EMCal TRU og feilsøking av programvaren over en god del mail fram og tilbake. Sist men ikke minst vil jeg takke veilederen Bernhard Skaali for all støtten og hjelpen jeg har fått underveis. Uten dette hjelpeapparatet ville det ikke vært noen oppgave å skrive.

Innholdsfortegnelse

1. Innledning

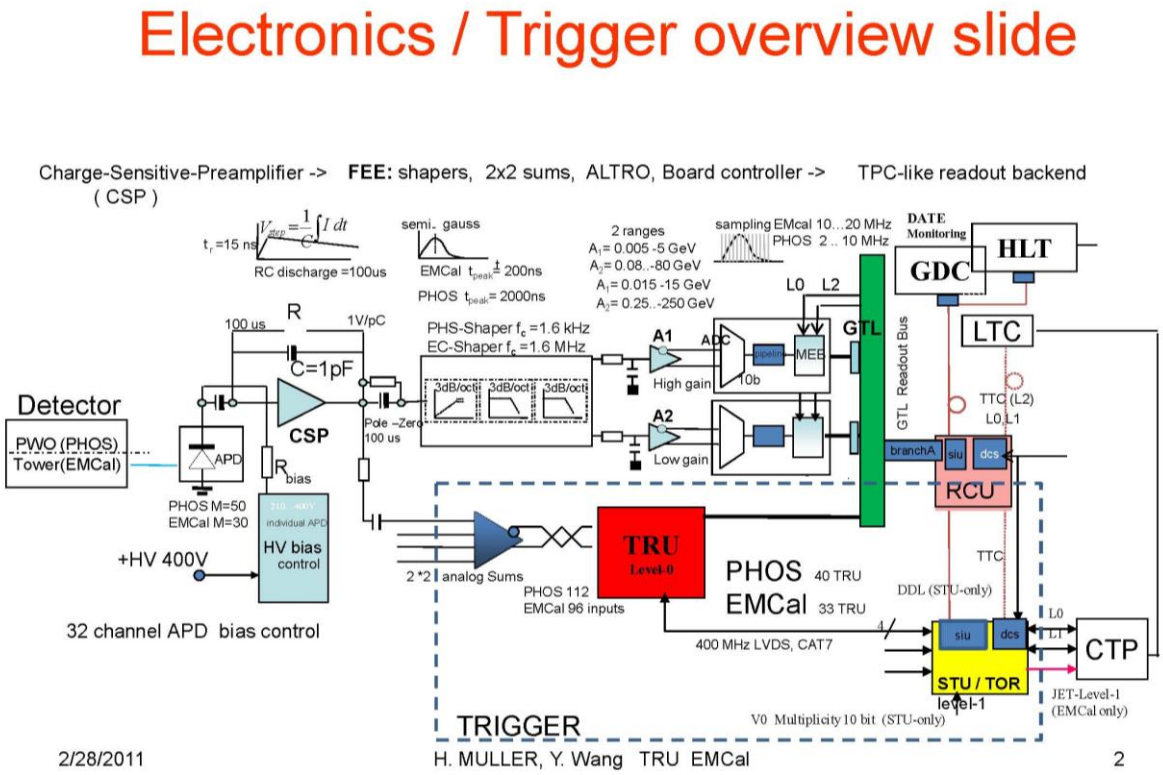
Large Hadron Collider (LHC) ved forskningslaboratoriet CERN er verdens største partikkelakselerator. Eksperimentene startet i 2009 etter en konstruksjonsperiode på ca 10 år. Akseleratoren er bygget i en 27km lang sirkulær tunnel ca 100m under bakken. Den vitenskapelige målsetning med LHC er å undersøke naturens fundamentale byggesteiner og kreftene mellom dem. I LHC sirkulerer to stråler av protoner eller blykjerner i motsatt retning med nesten lysets hastighet og styres mot hverandre i fire eksperimentstasjoner. Strålene består av flere tusen "pakker" av noen cm lengde med ca 10^{13} partikler, og som kolliderer hvert 25 ns, dvs. 40 MHz frekvens. Denne radiofrekvensen representerer tidsreferansen for dataregistreringene i eksperimentene.

De fire eksperimentene er ALICE (A Large Ion Collider Experiment), ATLAS, CMS og LHCb. Bare ALICE vil bli nærmere beskrevet her. ALICE er bygget opp av 13 detektorer, hver med sin spesifikke oppgave for registrering av energi og romlig fordeling av strålingen etter en kollisjon. Man antar at etter kollisjon mellom to blykjerner så er temperaturen og energitettheten i kollisjonsvolumet den samme som umiddelbart etter Big Bang. Grupper ved universitetene i Bergen og Oslo deltar i to ALICE prosjekter: PHOS detektoren og High Level Trigger (HLT).

Den ufiltrerte datamengden i ALICE er flere 100 GB/s, som overskrider langt hva som kan prosesseres og lagres i sann tid. Et sentralt element i datainnsamlingssystemet er derfor en "trigger" som i sann tid velger ut de potensielt mest interessante dataene. Triggersystemet i ALICE mottar data fra noen av detektorene i ALICE og genererer 3 nivåer av triggere kalt L0, L1 og L2 som distribueres til alle detektorer, med verdier YES/NO. L0 er en hurtig trigger 1.2μs etter kollisjonen og brukes av detektorer som lagrer data umiddelbart, L1 etter 5.5 μs, og L2 etter 100 μs. Utlesning til datainnsamlingssystemet og lagring på masselager starter etter en L0-L1-L2 YES sekvens. Da vil datamengden være redusert til noen hundre MB/s.

PHOS består av 5 moduler hvor bare 3 er installert i ALICE. En modul består av 56×64 $PbWO_4$ krystaller, hver med dimensjon 20x20x180 mm^3 og en Avalanche Photo Diode (APD) som konverterer lyset til et elektrisk signal. Signalet blir så forsterket av en Charge Sensitive

Preamplifier (CSP) før den blir sendt videre til analog shaper og digitalisering på et Front End Card (FEC). Hvert FEC prosesserer 32 krystaller. Utlesningskjeden er vist i Figur 1.



Figur 1.1 Utlesningskjeden for PHOS og EMCAL (1)

PHOS er en av detektorene som genererer triggerinformasjon til ALICE *Central Trigger Processor* (CTP). En PHOS L0 eller L1 trigger er karakterisert ved en energiavsetning i et klynge[1] av krystaller som overstiger et visst nivå.

Prosesseringen i en Trigger Region Unit (TRU) gjøres i *firmware* til en stor FPGA. TRU-modulen mottar hurtige analoge signaler fra en *branch* med 14 FEC som er summert 2x2 på hvert FEC, digitaliserer disse og lagrer verdiene i en FPGA buffer. I sann tid søker FPGA-koden i en matrise med de 8 x 14 energiverdiene etter en eller flere energiklynger. Pga den fysiske avstanden til CTP må en PHOS L0 foreligge etter ca 600 ns for at en ALICE L0 kan distribueres til samtlige detektorer etter 1.2µs.

PHOS-prosjektet skal i 2011-2012 bygge en fjerde modul, og elektronikken for denne modulen må produseres. TRU-kortet i de eksisterende PHOS-moduler ble utviklet og

[1] Når høyenergetisk stråling treffer en bly-wolfram krystall vil absorpsjonsprosessen medføre en skur av sekundærpartikler som sprer seg til nabokrystallene. Typisk vil det være data i 8-10 nabokrystaller.

produsert i 2005-06. En nyere versjon av TRU er utviklet av EMCal detektoren og et antall skal produseres for den nye DCal detektoren som skal installeres i 2013. EMCAL og DCal er ALICE kalorimetre som bruker den samme elektronikken som ble utviklet for PHOS, med noen mindre modifikasjoner. EMCal og DCal vil ikke bli nærmere beskrevet i denne oppgaven.

Målsetningen med denne oppgaven å evaluere en EMCal TRU for bruk i PHOS, for en eventuell felles produksjon med Dcal. Evalueringen skal omfatte:

1) Bruke utviklingsverktøyet Xilinx ISE 10.1 for å kompilere den eksisterende *firmware* for EMCal TRU. Xilinx ISE 10.1 brukes av gruppene ved CERN og i Bergen.
2) Studere hvilke modifikasjoner som kreves for PHOS. På grunn av tiden til disposisjon for hovedoppgaven skal studiet konsentreres om grensetversnittet mellom TRU og utlesningsbussen, kalt *Fake ALTRO* fordi det emulerer protokollen mellom ADC-kretsen ALTRO på FEC og utlesningsbussen.
3) Teste en TRU lånt fra EMCal i en PHOS utlesningskjede i Bergen.

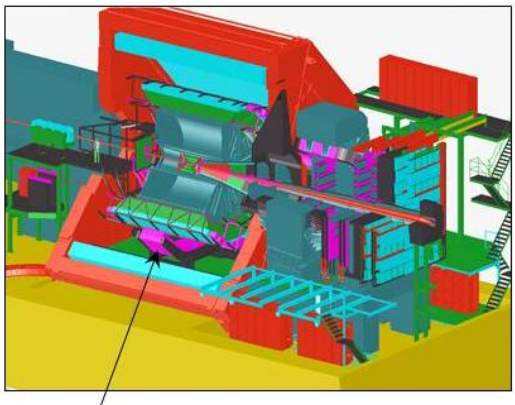
Oppgaven er delt i to hoveddeler, herunder en teoretisk del og en praktisk del. Først presenteres grunnleggende bakgrunnsinformasjon som er viktig for en tilstrekkelig forståelse av oppgaven. Her beskrives først PHOton Spectrometer (PHOS) detektoren og utlesningselektronikken i kapittel 2. I og med at EMCal elektronikken baserer seg på PHOS så er meste parten av elektronikken det samme. Utlesningskjedene er også likt for begge detektorene. Deretter, i kapittel 3, presenteres PHOS triggersystem som er en av hovedelementene i lagring av data fra en kollisjon. TRU programvaren er skrevet i Xilinx og er på den måten vesentlig med hensyn til kjøring og simulering av programkoden. Xilinx presenteres derfor i kapittel 4. Det er videre valgt å fokusere på Fake ALTRO delen av EMCal TRU i denne oppgaven. Dette er fordi en utlesning fra Fake ALTRO kan sammenlignes med utlesningen av ALTRO fra et PHOS FEC og se om det er noen korrelsasjon mellom dataene. Det skal være en korrelasjon og videre vil det tyde på at TRU kortet er riktig implementert. Fake ALTRO programkoden presenteres derfor mer inngående i kapittel 5. Den teoretiske delen av oppgaven blir deretter etterfulgt av kapittel 6 som omhandler hvordan testlab-en i Bergen er satt opp og utlesningsprosedyren, samt resultatene som er oppnådd. Oppgaven avsluttes med en kort konklusjon som besvarer hvorvidt resultatene tyder på at EMCal TRU kan benyttes i PHOS.

2. PHOton Spectrometer (PHOS)

I dette kapittelet vil det bli presentere hva PHOS er. Det vil bli gått gjennom elektronikken til FEC, ALTRO, ALTRO data format og ALTRO buss. For å få en forståelse av oppgaven så er det viktig og først sette seg inn i hva PHOS er for noe og hvordan signalflyten fungerer her. Med hensyn til at Fake ALTRO skal oppføre seg som en ALTRO ved utlesning så er det også viktig å spesifisere hva en ALTRO krets gjør. Deretter blir det også naturlig med en liten introduksjon av hva Fake ALTRO er.

PHOS er konstruert for å registrere energi og posisjon til høyenergetisk elektromagnetisk stråling i energiområdet 0.1-100 GeV. Elektromagnetisk stråling kan gi informasjon om den primære *fireball* i kollisjonen fordi denne strålingen ikke blir modifisert av senere prosesser. PHOS vil også registrere elektrisk ladede og nøytrale partikler.

ALICE detektoren og konstruksjonen av PHOS er vist i henholdsvis **Figur 2.2**.2 og **Figur 2.3**. Begge figurene er fra (2)



PHOS (PHOton Spectrometer) is a high resolution electromagnetic calorimeter consisting of 17920 detection channels based on lead-tungstate crystals(PWO).

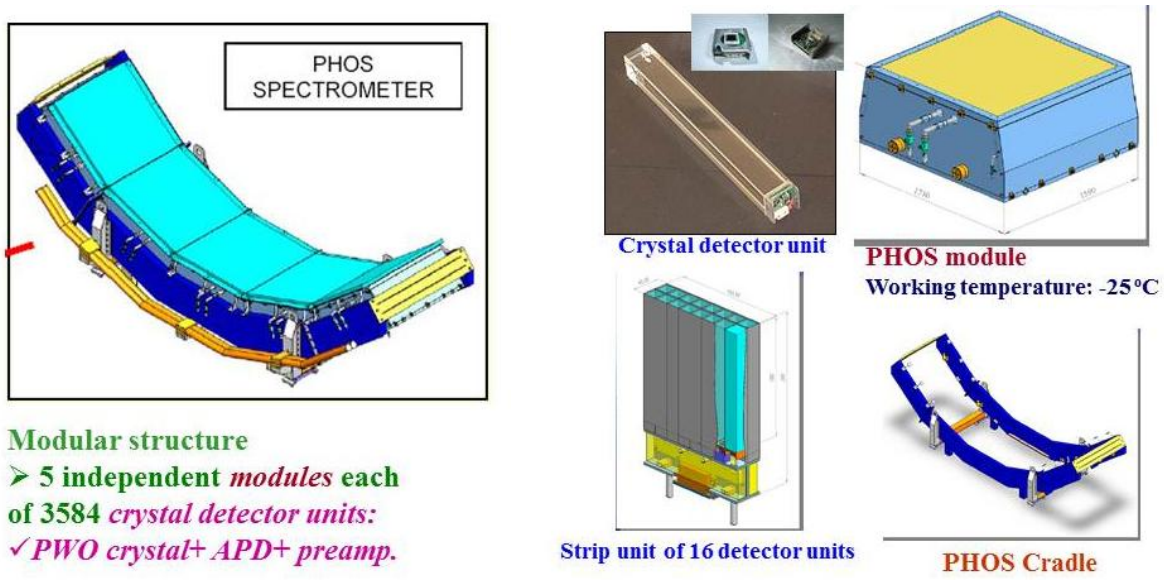
- PHOS provides unique coverage of the following physics topics:
  - -study initial phase of the collision of heavy nuclei via direct single photons and diphotons,
  - -jet-quenching as a probe of deconfinement, studied via high p_T γ and π^0,
  - -signals of chiral-symmetry restoration.

Technical data:

17920 lead-tungstate crystals(PWO)

-distance to IP	4400mm
-coverage in pseudorapidity	-0.12;+0.12
-coverage in azimuthal angle	100°
-crystal size	22x22x180 mm^3
-depth in radiation length	20
-modularity	5 modules
-total area	$8m^2$
-total crystal weight	12.5 t
-operating temperature	-25 °C
-photoreadout	APD

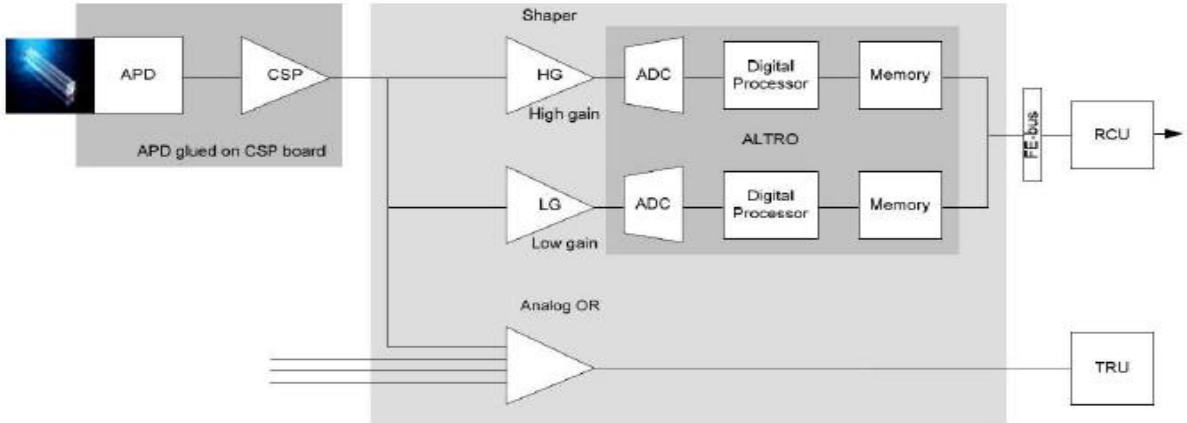
Figur 2.2 ALICE detektoren



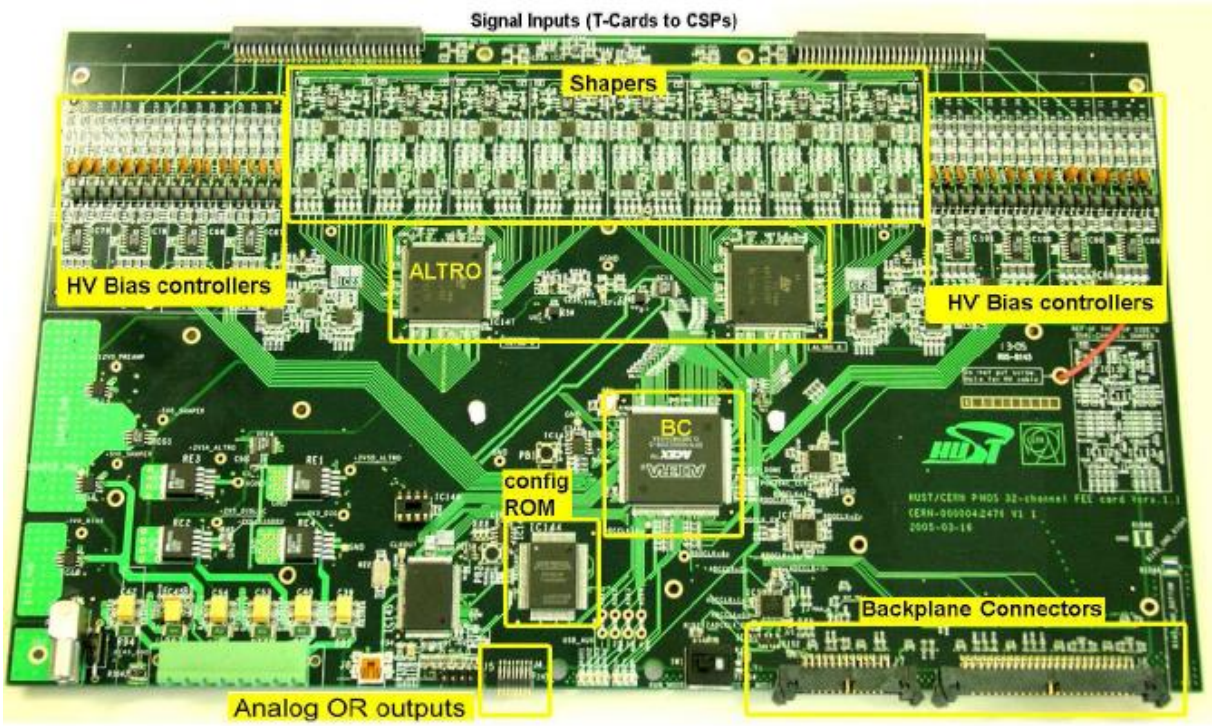
Figur 2.3 Konstruksjonen av PHOS

2.1 PHOS Front-End elektronikk

En PHOS modul har 56×64 = 3584 $PbWO_4$ krystaller. Lyset fra en absorpsjonsprosess i en krystall blir konvertert til en elektrisk ladning i APD-en og til et spenningssignal i etterfølgende CSP. Her går signalet inn på FEC hvor det blir delt i to deler. Den ene delen går videre til en analog summering av fire og fire signaler fra CSP, se figur 2.4. Hvert FEC får inn signaler fra 32 krystaller, disse blir summert i den ene delen og ut går 8 delsummer videre til TRU (Trigger Region Unit, kommer tilbake til dette). Antall elektroniske kanaler i en PHOS modul er 2 x 3584 fordi det analoge signalet fra hver CSP går også til to shapere med relativ forsterkningsfaktor 1 og 16, som digitaliseres i to ADC-er. Dette er gjort for å øke det dynamiske området



Figur 2.4 PHOS signalkjede (3)

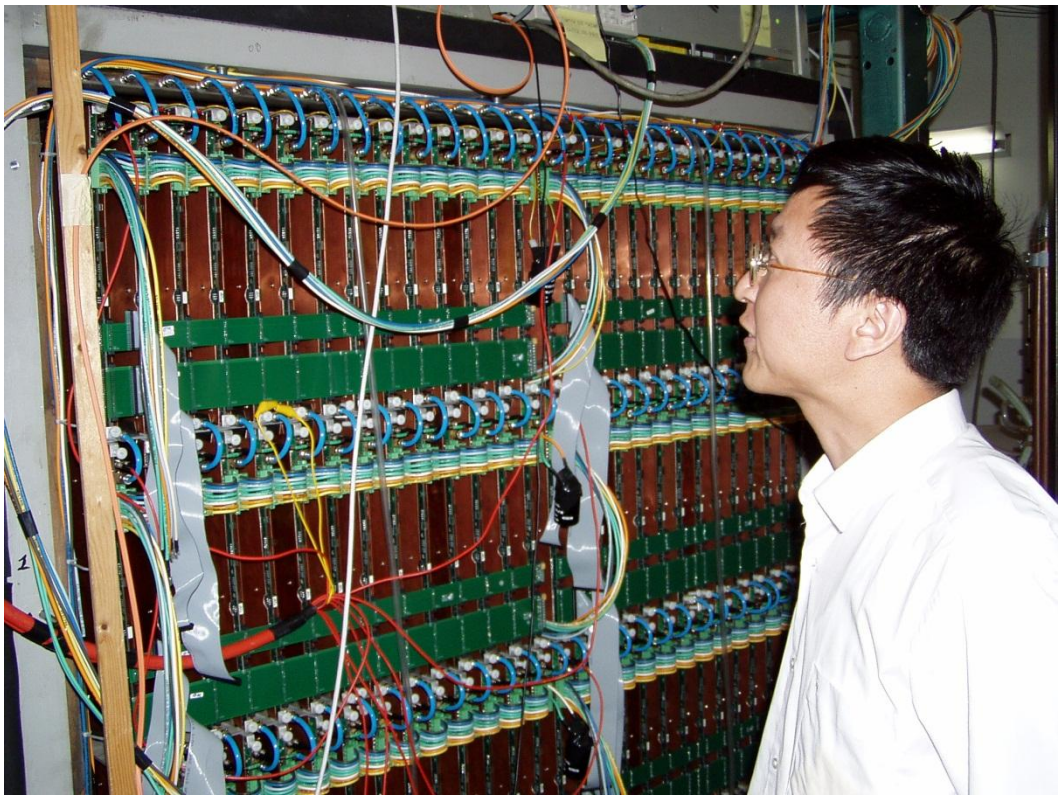


Figur 2.5 PHOS Front End Card (FEC) (4)

"Front-end" elektronikken er organisert i 8 grener, hver gren med 14 FEC og en TRU (5), i alt 120 kort. Utlesningskjeden er vist i Figur 1 og et Front End Card (FEC) er vist i figur 2.5.

Datautlesningen styres av en RCU (Readout Control Unit, kommer tilbake til dette) basert på trigger signaler fra ALICE Trigger System og de formatterte dataene sendes videre til datainnsamlingssystemet DATE, se Figur 1**Feil! Fant ikke referansekilden.**. Hver RCU kontrollerer to grener. Konfigureringen av Front-End elektronikken gjøres også fra RCU. Alle kortene i en gren er koblet sammen med RCU-en over en kabelbuss med GTL (Gunning Transistor Logic) signalering.

En del av elektronikken for en PHOS modul er vist i figur 2.6. Hver rad har to grener, A og B. En gren består som sagt av 14 FEC og en TRU og kommunikasjonen går via de grønne kabelbussene (GTL).



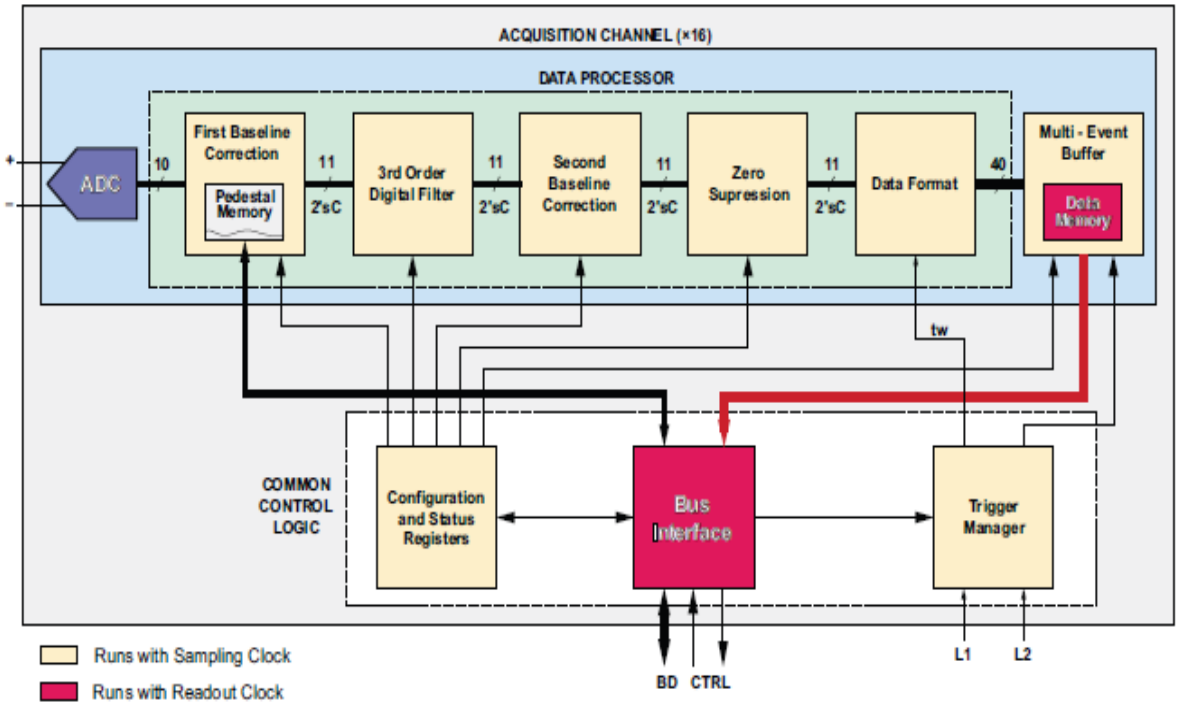
Figur 2.6 PHOS Front End Cards med GTL buss (4)

2.1.1 ALTRO krets

ALTRO er en spesielt designet ASIC (Application Specific Integrated Circuit) (6) utviklet i samarbeid med CERN og firmaet STMicroelectronics. Hver ALTRO inneholder 16 10-bit ADC-er (Analog-Digital Converter), hvor det differensielle inngangssignalet samples fortløpende med opptil 20 MHz. For PHOS er samplingsfrekvensen valgt til 10 MHz. Ved mottagelse av et signal fra ALICE Trigger System vil samplede verdier bli lagret i en multi-event[2] buffer hukommelse. Antall dataord som lagres velges ved programvare konfigurering av kretsen.

Oppbyggingen av ALTRO er vist på figur 2.7. Kretsen inneholder digitale filtre for prosessering av de digitaliserte data. De 10 bit digitale verdiene formateres til 40 bit ord i Data Format enheten og lagret i Multi-event buffer etter en trigger.

[2] Multi-event buffer vil si at data fra flere triggere kan lagres før de leses ut. Dette vil utjevne datastrømmen.



Figur 2.7 ALTRO krets (6)

For utlesning av data fra TRU-ene vil firmwaren emulere grensesnittet for ALTRO.

2.1.2 ALTRO buss

ALTRO bussen står for kommunikasjonen mellom TRU og RCU. Her kommer blant annet kommando for en utlesning til TRU, hvor data til utlesning blir sendt tilbake på samme buss. Tabell 2.1 gir en oversikt over de viktigste signalene til ALTRO bussen (6).

ALTRO BUSS				
Signal navn	**Funksjon**	**# Bits**	**Retning**	**Polaritet**
AD	Adresse / Data	40	Toveis	H
WRITE	Write / Read	1	Input	L
CSTB	Command Strobe	1	Input	L
ACKN	Acknowledge	1	Output	L
ERROR	Error	1	Output	L
TRSF	Transfer	1	Output	L
DSTB	Data Strobe	1	Output	L
LVL1	Level-1 Trigger	1	Input	L
LVL2	Level-2 Trigger	1	Input	L
GRST	Global Reset	1	Input	L
SCLK	Sampling Clock	1	Input	-
RCLK	Readout Clock	1	Input	-

Tabell 2.1 ALTRO buss signaler

AD [39:0] Toveis

Dette er en 40-bit toveis data/adresse buss som består av, fra det minst signifikante bit: Datafelt (20 bit), Instruksjonsfelt (5 bit), Adressefelt (14 bit) og et paritet bit. Med en størrelse på 14 bit så har adressefeltet en størrelse på 16384 som er delt inn i to like store størrelser: ALTRO (AL partition) og en Board controller (BC partition). Se Tabell 2.2.

PAR[39]	ADRESSE[38:25]			INSTRUKSJONS KODE[24:20]	DATA[19:0]
	BCAST[38]	BC/AL[37]	KANAL ADRESSEN[36:25]		

Tabell 2.2 Toveis 40-bit Adresse / Data buss

AD [39]

Dette er paritetsbit-et for de 20 mest signifikante bitsene og det er satt sånn at disse bit-ene alltid er like (like mange 1-ere som det er 0). Paritet bit-et kan detektere om det oppstår en enkel error på signaloverføringen mellom RCU og FEC.

AD [38]

Når Broadcast blir satt høy så blir ”skrive syklusen” til bussen initiert av RCU (som er master) adressert til en hel partisjon av adressefeltet (AL eller BC). Når dette oppstår så tar slaven og ignorer det som står i kanal adressefeltet og sender ut til alle.

AD [37]

Dette bit-et spesifiserer hvilket av partisjonene som blir valgt ved Broadcast. Ved ”1” så er det BC grenen og AL hvis ”0”.

AD [36:25]

Fra det mest signifikante bit-et: gren A eller B (1 bit), FEC adressen (4 bit), hvilken ALTRO chip (3 bit) og de interne kanaladressene i ALTRO (4 bit). Disse er satt sånn at det kan være 8 ALTRO på brettet og to grener som hver består av opp til 16 FEC.

AD [24:20]

Her settes instruksjonene som kontrollerer BC og ALTRO chipen. Dette kan være enten adgang til et Konfigurering / Status Register (CSR) hvor adressen er en del av

instruksjonskoden, eller så kan det være en kommando. En kommando som er spesielt interessant er ”CHRDO” kommandoen (AD [24:20] = ”11010), denne kommando setter i gang en utlesningsprosess (kommer tilbake til dette). Når det er adgang til CSR så innebærer instruksjonene en skrive eller lese syklus til / fra CSR i forhold til om WRITE signalet er lavt eller høyt.

AD [19:0]

Her blir data lagt ved skrive og lese instrukser.

WRITE

Skrive signal blir drevet av RCU og definerer om adgangen til det adresserte feltet er i skrive eller lese modus (Lav / Høy).

CSTB

Command Strobe signalet er drevet RCU og når dette bit-et er satt så indikerer det at et gyldig ord er satt i AD bussen. Bit-et godkjenner også WRITE signalet og blir først avsatt etter at slaven har satt ACKN signalet. Dette gjelder alltid, bortsett fra når Broadcast er satt, da det ikke er noen ACKN.

ACKN

Den adresserte enheten setter acknowledge når den får en skrive eller kommando syklus, dette er for å signalisere at den har fylt opp minne med innholde til bussen og satt i gang de instruksjonene som ble gitt. Ved en read syklus så settes acknowledge hos den adresserte enheten for å si ifra at den har plassert dataen som ble spurt om på bussen. Eneste avviket her er når det oppstår en broadcast instruksjon, da denne ikke trenger noen acknowledge.

ERROR

Slave enheten setter dette bit-et for å signalisere at en error betingelse har oppstått. Under en instruksjons syklus så kan det oppstå både paritet- og instruksjonskode ERROR og hvis dette oppstår så kjenner ikke slaven igjen instruksjonssyklusen og setter ERROR signalet.

TRANSFER, TRANSFER_EN, DOLO_EN, DSTB

Utlesning av en ALTRO chip foregår i to steg: Første steg er når det sendes ut en kommando med instruksjonskoden CHRDO (channel readout, 11010) fra RCU. Andre steg skjer etter et antall klokkesykluser når ALTRO chipen tar over bussen ved å sette transfer signalet og instruksjonen blir godkjent. TRANSFER er så satt helt til datablokken er ferdig overført. Dataoverføringen er ikke alltid kontinuerlig og derfor settes DSTB for hvert ord som blir

overført. TRANSFER_EN blir brukt for å kjøre den toveis AD bussen ved en event, mens DOLO_EN blir brukt når det skal leses ut et register.

LVL1 – LVL2

Disse to signalene er broadcastet til alle FEC fra RCU og er brukt for å distribuere trigger informasjon. LVL1 er synkronisert med SCLK signalet og varer minimum i to sykluser, men LVL2 signalet er synkronisert med RCLK og varer i to klokkesykluser.

GRST, SCLK, RCLK

Global reset igangsetter alle interne registre, tellere og tilstandsmaskiner. ALTRO sample klokken SCLK kan ha en maks frekvens på 20MHz, alt av dataprosessering til ALTRO er gjort synkronisert sammen med SCLK. Utlesningsklokken RCLK til ALTRO har en maks frekvens på 40 MHz og er klokken for ALTRO bussens mester og slave grensesnittet.

2.1.3 Fake ALTRO

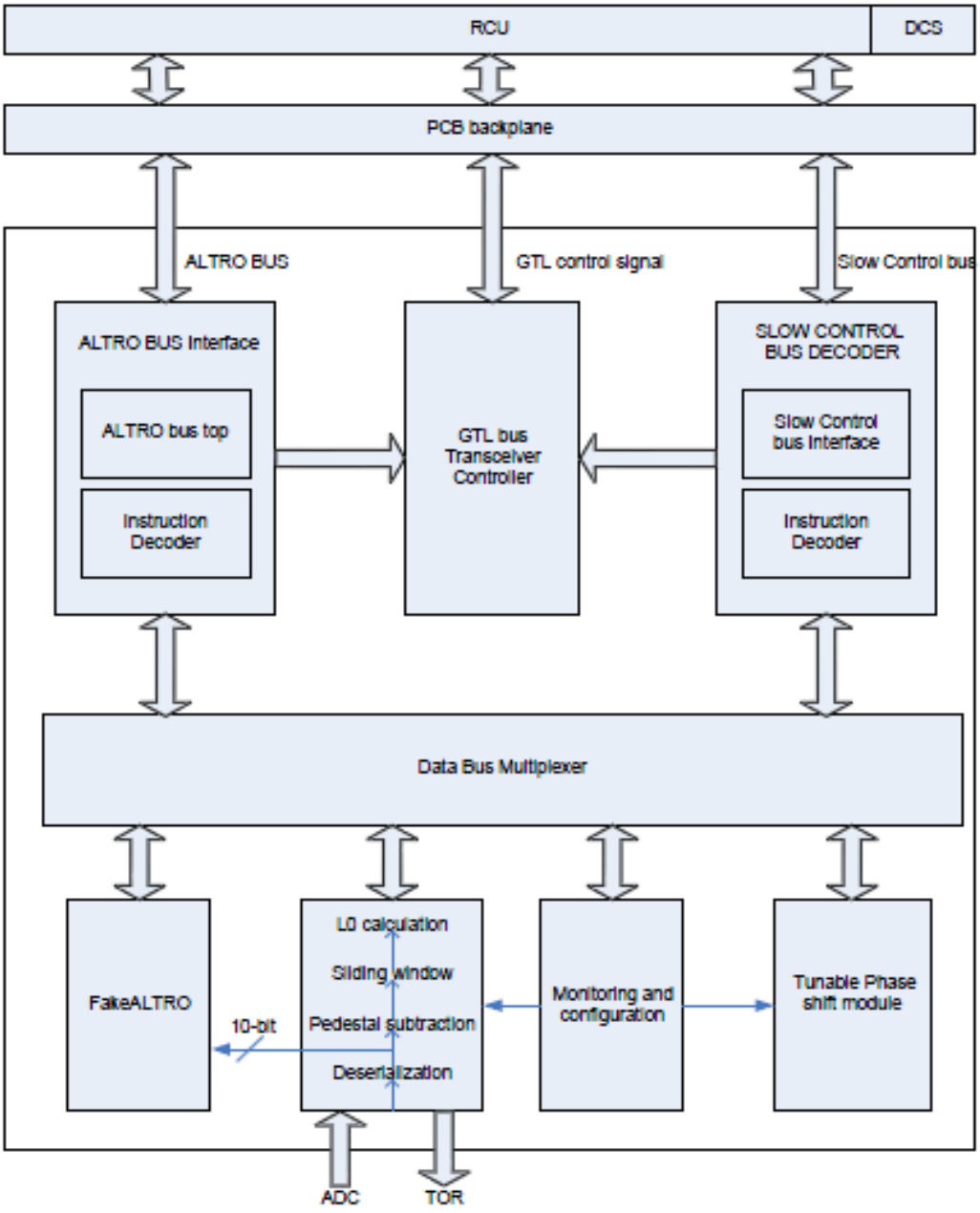
I tilegg til å generere en L0 avgjørelse så kjører programvaren til TRU også konfigureringen av ADC-ene og monitorerer registerstatusen. En av de viktigste oppgavene er å implementere en ALTRO lignende minne som kalles Fake ALTRO. Dette er lest ut av RCU via standard ALTRO buss på akkurat samme måte som ALTRO.

Signalene som kommer inn på TRU blir konvertert til digitale signaler av en 12-bit ADC. De tolv-bit digitale signalene blir deretter deserialisert til en parallell datastrøm. Herfra går en del til Fake ALTRO, og en annen del går videre for å avgjøre en LO beslutning (kommer tilbake til dette). Før signalene kommer inn til Fake ALTRO så reduseres de to minst signifikante bitene av hvert signal. Fake ALTRO signalene består da av 10 bit hver som gjør den kompatibel med ALTRO data format (3).

Vi ser også på figuren at ”ALTRO Bus Interface” fungerer som en kobling mellom RCU og Fake ALTRO, så alle kommandoer som er utstedt fra RCU og all data som er drevet av Fake ALTRO går alle gjennom ALTRO bussen. Den er delt inn i to deler, en ”ALTRO bus top” og en ”instruction decoder”. Hovedoppgaven til den første er å håndtere ALTRO buss protokollen, som innebærer å sample signaler på bussen til riktig tid og sjekke adresser og paritets feil. ”Instruction decoder” dekoder instruksjonene som er sent fra RCU og her er det

spesielt interessant med CHRDO kommandoen (AD [38:37] = ”00”, AD[35:32] = ”0000”, write = ”0” og AD[24:20] = ”11010”) som setter i gang en utlesning.

PHOS Fake ALTRO er implementert ved å bruke to buffere, først så er det en sirkulær RAM hvor informasjonen pr sample går inn på nederste blokk og blir flyttet oppover ettersom samplene kommer inn. Når første sample når toppen så blir det dyttet ut av bufferen (slettet). Den sirkulære bufferen har en bredde på 1120 bit inn som da inkluderer alle 10-bit ordene fra 112 kanaler. Etter parallelliseringen så blir alle 12-bitene fra 112 kanaler redusert til 10-bit og formatert til ett 1120-bit ord. Når L0 signalet fra ALTRO bussen blir satt så går data videre ut fra den sirkulære bufferen og videre inn til lagring på MEB bufferen for å bli lest ut av RCU. Dermed så har man et visst antall sample tilbake i tid når L0 først er satt (6). For at en L0 skal gå ut til detektorene innen 1.2 μs så må avgjørelsen i TRU være tatt på 600 ns. Det tar da 200 ns å komme seg til CTP som da sender ut en L0 til detektorene. Når L0 signalet kommer ut så er det allerede rådata i den sirkulære bufferen som går rett inn i MEB.



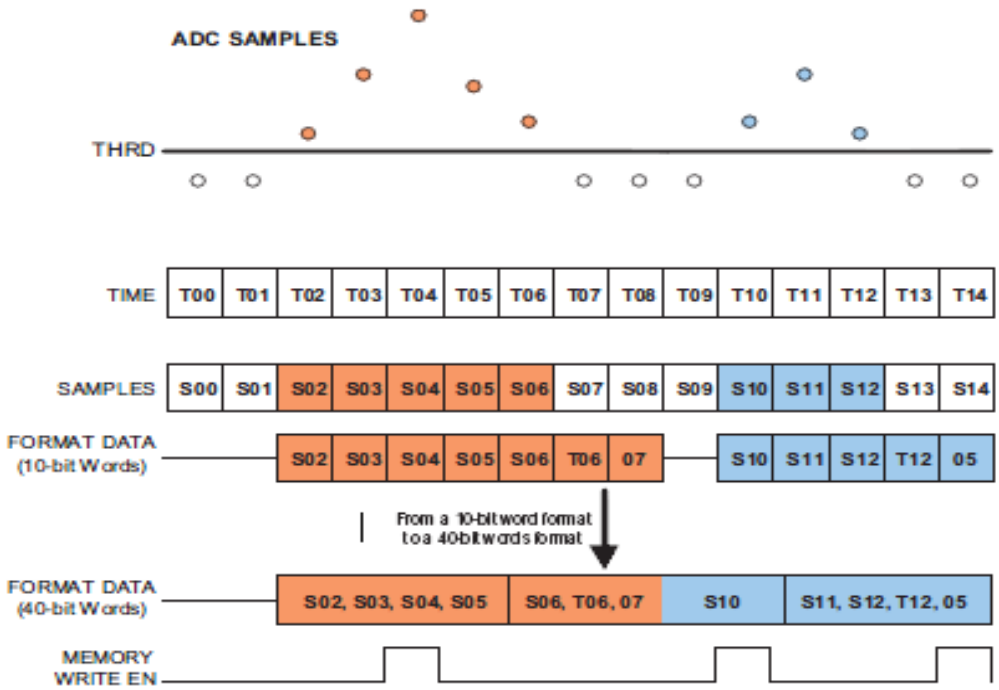
Figur 2.8 Blokkskjema av TRU (3)

2.1.4 Data format til ALTRO

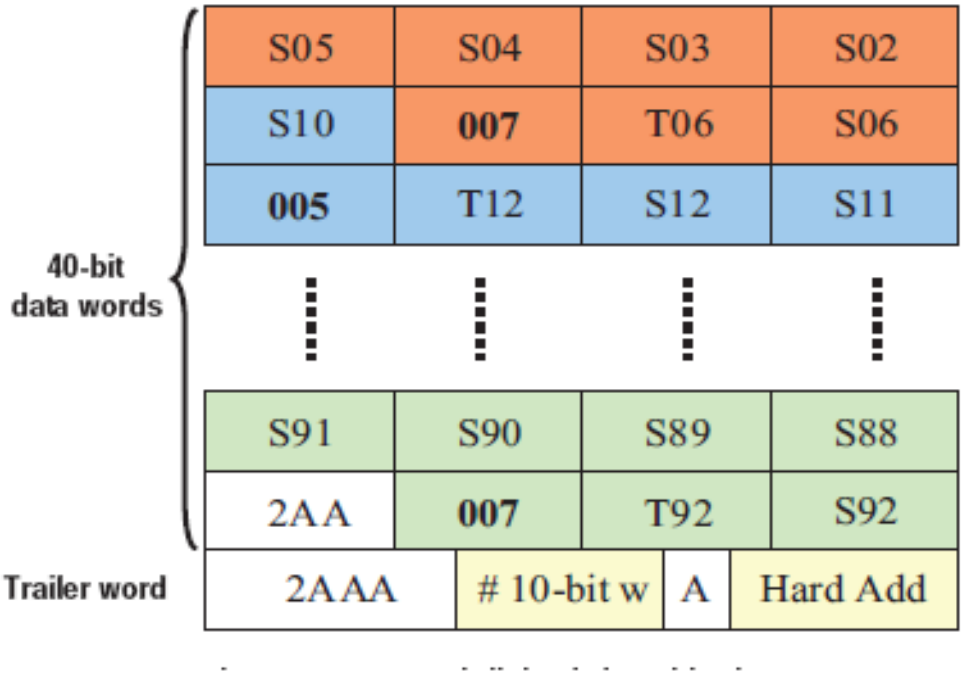
En av oppgavene til ALTRO chipen er data formateringen og enkoding av ti-bit ord til 40-bits ord. Før dette så gjør ALTRO chipen en ”zero suppression”. Man forkaster da de samplene som er så nærme den bestemte pidestallverdien, for de har verdiløs data og kan bli betraktet

som støy. I og med at tidsinformasjonen blir borte ved en sånn operasjon så må det legges til ekstra informasjon i data formateringen. Derfor blir det lagt til to ti-bit ord, hvor den ene inneholder informasjon om tiden og den andre om hvor mange samples som er i klyngen.

Figur 2.9 viser prosessen til data formateringen hvor ”S” står for samples, og ”T” står for time. Som vi ser i figur 2.9 og 2.10 så blir ti-bit ordene samlet sammen til 40-bit ord. Når det er noe data som mangler for å fullføre et 40-bit ord så settes det inn et heksadesimalt mønster ”A”. For å fullføre datapakken blir det satt til et ”trailer” ord. Dette ordet er det siste 40-bit ordet i datapakken og er sammensatt av forskjellig relevant data som blant annet: Totalt antall nummer av ti-bit ord og posisjonen til det siste ti-bit ordet i pakken. Den består også av adressen til hardware. Dette er da FEC adressen (4 bit), adressen til en av de åtte ALTRO (3 bit) og kanal adressen (4 bit). Denne adressen representerer en geografisk adresse som er brukt i datapakker for å identifisere hvilken kanal datapakken hører hjemme i. Den gjenværende informasjonen er fylt ut med mønsteret ”A” (heksadesimalt). Figur 2.10 viser datapakken hvor ”A”, ”2AA” og ”2AAA” er fyllmasse (6) (4).

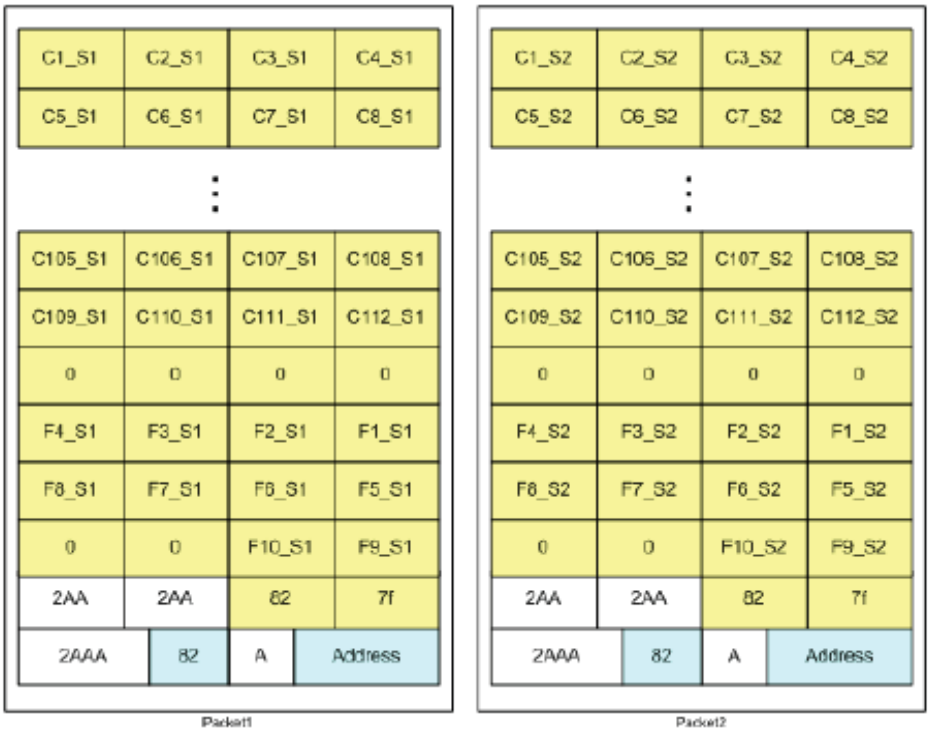


Figur 2.9 Data formatering for ALTRO (6)



Figur 2.10 Data blokk for ALTRO chip (6)

Det er ingen zero suppression eller tids informasjon som blir lagt til i TRU. Den digitale 12-bit dataen blir redusert ned til 10-bit etter den har blitt deserialisert for så å bli formatert til 40-bit ord. I TRU så er alle 10-bit ordene samples, alle 112 kanalene for en sample er sett som en dataklynge med en pre-trailer som inkluderer antall 10-bit ord i pakken og tidsinformasjon som gir den samme informasjonen som de to ekstra 10-bit ordene på samplene hos ALTRO. I hver datapakke så finnes det bare en dataklynge og den er etterfulgt av en trailer som også inneholder informasjon om antall ti-bit ord i pakken og i tillegg så gir den hardware adressen. Adressen inneholder 1-bit branch, 4-bit FEC (hvor adressen til TRU er ”0000”) og RAM adressen. Fake ALTRO minne er implementert med RAM i FPGA. Som man ser i figur 2.10 så står det C1_S1 i den øverste gule firkanten til venstre. Det betyr at det er kanal 1 og sample 1. Siden samplene ved forskjellig tidspunkt blir lagret i MEB i forhold til disse tidspunktene så kan man ut fra RAM adressen få tidsinformasjonen for data dekodingen (3).



Figur 2.11 Fake ALTRO data blokk. Her står hele den gule delen for en data klynge (3)**.**

3. PHOS Trigger system

I dette kapittelet skal jeg gå gjennom trigger systemet, TRU, TOR, RCU og DAQ systemet. Dette er essensielt for å få det store bildet av prosessen. Trigger systemet er det som gjør at man kan begynne å lagre data i blant annet Fake ALTRO bufferen. Dette minnet ligger i TRU så blir da naturlig å ta med en forklaring på hva TRU er og hva som skjer på dette kortet. TOR er bindeleddet mellom FEE og CTP. RCU og DAQ samarbeider for å få ut dataen fra bufferne til noe forståelig og lesbart på datamaskinen.

3.1 Trigger oversikt

Triggersystemet til ALICE er bygd opp av to uavhengige deler, en CTP og en trigger fordelings nettverk som inneholder både LTU (Lokal Trigger Unit) og TTC system (Trigger, Timing og Control) (7) (8) (9) (10). De triggerne som blir generert blir sent videre til CTP hvor triggerne er prosessert og sent videre til detektorene via trigger fordelingsnettverket.

PHOS detektoren er en av subdetektorene som bidrar med trigger input til CTP (Central Trigger Prosessor). L0 trigger er den kjappeste trigger og må være hos CTP innen 800ns etter at en kollisjon har tatt sted. Når en PHOS L0 trigger blir generert så har PHOS registrert stråling over en viss energi. Dette er den eneste informasjonen vi får fra en generert L0 i PHOS, og dette er ikke tilstrekkelig informasjon for å lagre data. En av oppgavene til trigger systemet er for å skalere ned antall hendelser som blir lagret og for det så er ikke L0 på langt nær godt nok. En generert L1 trigger gir videre filtrering og må være hos CTP innen 6.1 μs etter en kollisjon (11). Denne triggeren gir også informasjon om hvor i den 56×64 matrisen av krystaller kollisjonen oppsto. Den siste triggeren L2 sier at data er akseptert, begynn utlesning. L2 skal være ute hos detektorene innen 100 μs.

Arkitekturen til PHOS og EMCal ReadOut og trigger system er veldig like siden de begge er basert på det samme FEE og TRU hardware, og har det samme ReadOut konseptet. Hovedforskjellen er den globale trigger enheten. PHOS bruker TOR(Trigger OR, kommer

tilbake til denne) som prosesserer alle innkomne trigger signaler fra TRU-kortene og tar en logisk ”eller” av alle L0 triggere fra de fem modulene. Konseptet til EMCal er veldig likt, men har et par ekstra input som blir tilført av STU (Summing Trigger Unit) modulen som brukes istedenfor TOR (12).

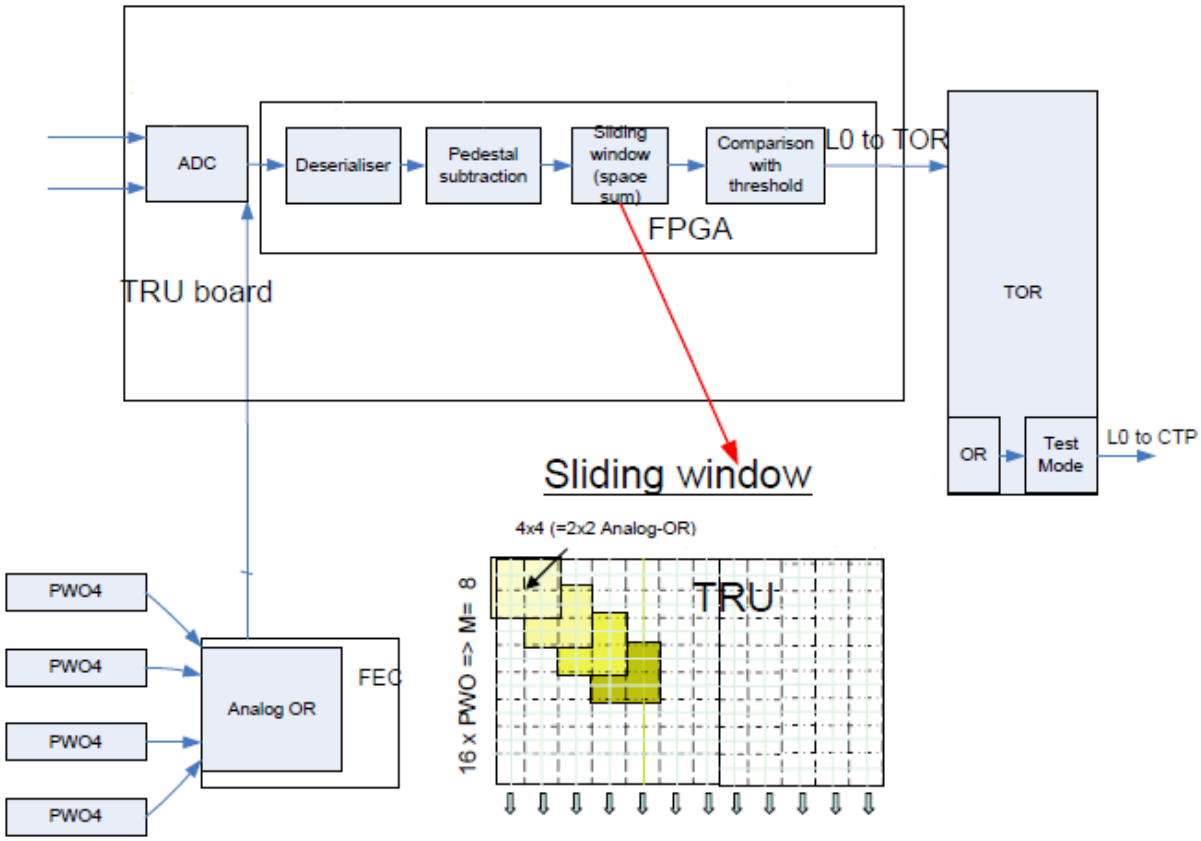
3.2 PHOS TRU

FEE (Front End Electronics) i hver PHOS modul består av 8×14 =112 FEC som hver består av 4 ALTRO og hver av de 8 grenene har en TRU. Energiklyngen blir funnet basert på en 2×2 delvis parallell summering av de 8×14 kanalene på inngangen til TRU-en i en gren. Her blir en hver summering sammenlignet med et pidestallnivå og hvis summen overstiger dette nivået så vil ett logisk JA-signal bli sendt til TOR.

To av de viktigste oppgavene til TRU er å generere en L0 trigger og å lagre data i Fake ALTRO. Oversikten til L0 genereringen i TRU er vist i figur 3.1. Signalene som er 2×2 analogt summert kommer inn til en 12-bit ADC med 8 kanaler på TRU kortet hvor de blir konvertert fra analoge til digitale signaler. Disse signalene blir sendt i serie inn på FPGA-en til TRU hvor de først blir omgjort til en parallell datastrøm. Så går signalene videre til en ”Pedestal subtraction” hvor en gitt verdi blir trukket fra det digitale signalet før den så blir ført videre til et 4×4 ”Sliding window”. Energien til et foton eller en partikkel fordeler seg utover flere krystaller sånn at man må ta summen av krystallene som grenser til hverandre for å få den totale energiverdien hvilket en L0 trigger er generert fra. Her vil de 2×2 analoge summene fra FEC bli summert 2×2 sånn at det blir en total summering på 4×4 (4×4 sliding window, se figur 3.1). Siden strålingen kan treffe en hvilken som helst krystall så må man gå gjennom og summere alle utfall av en 4×4 summering inne i 8×14 matrisen, se figur 3.1. Fra FEC-ene i en gren kom det 112 kanaler inn på TRU som blir summert 2 og 2, sitter da igjen med 7×13=91 delsummer som hver og en blir vurdert opp mot verdien for en L0 trigger. En L0 blir generert hvis minst en av de 91 summene er over pidestallverdien.

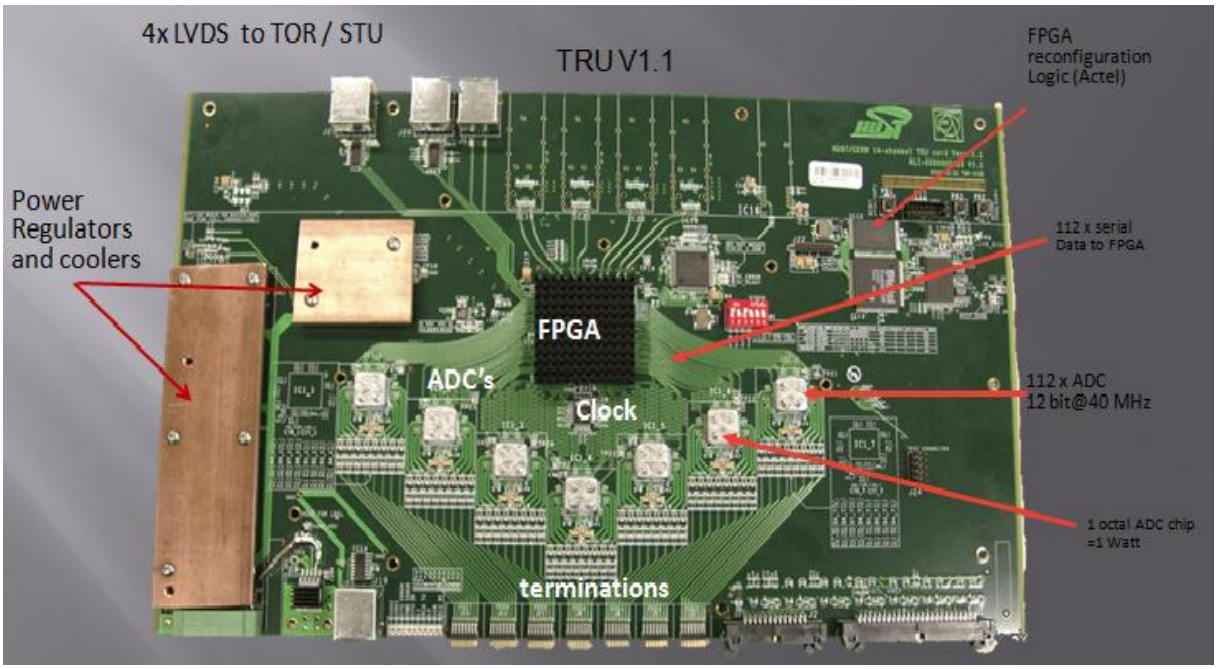
Fake ALTRO delen gir en måte å analysere den analoge summeringen fra FEC og godkjenner at deserialiserings modulen fungerer som den skal. Videre kan man sammenligne

dataen til ALTRO og Fake ALTRO for å finne ut om triggeren fungerer i henhold til de kravene som er satt (3). Simulering viser at den optimale trigger effektiviteten får man ved 4×4 sliding window (13).



Figur 3.1 Oversikt over en L0 generering i TRU (3)

På figur 3.2 ser vi en oversikt over TRU-kortet. Det blir mottatt 112 signaler fra 14 FEC. På figuren ser vi at vi har 7 ADC på toppen av kortet, det er også 7 stykker til på undersiden.



Figur 3.2 Oversikt over TRU (3)

Hver av disse mottar åtte analoge signaler fra hvert FEC som blir digitalisert til åtte 12 bit signaler og går serielt videre til FPGA-en (Field Programmable Gate Array).
Siden dataen fra ADC er seriell så må den først parallelliserer og så deles informasjonen i to. Den ene er dataen fra FEC som blir lagret i Fake ALTRO i samme format som på ALTRO for videre sammenligning når det kommer en L0 trigger. TRU programvaren genererer også en 2×2 summering over FEC dataen og vi ender opp med en 7×13 matrise av delsummer. På dette signalet begynner både Fake ALTRO og ALTRO å lagre data i MEB (14).

EMCal TRU sin signalflyt går stort sett likt som hos PHOS TRU. EMCal har derimot bare 12 FEC (ikke 14 som i PHOS). PHOS kjører 20 MHz hvor EMCal kjører med 40Mhz. EMCal TRU prosesserer 96 signaler mot 112 for PHOS. Fra hvert FEC går det fortsatt åtte kanaler hvor hver kanal består av en analog sum av fire kanaler. I EMCal TRU så blir signalene summert 2×2 og man ender opp med en matrise på 7×11 for en L0 avgjørelse (15).

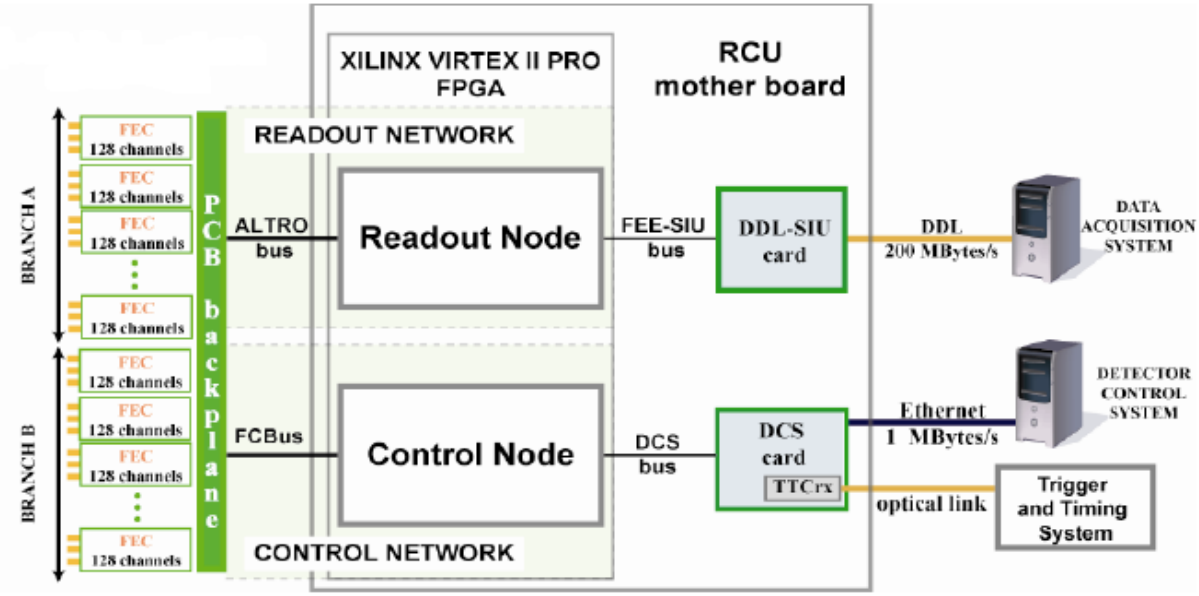
3.3 TOR

Alle 8 TRU-kortene i en modul er koblet opp mot TOR (Trigger OR) som i praksis tar en logisk ”eller” mellom alle innputene. Om et av signalene fra TRU er over pidestallverdien så sender TOR signal til CTP om å utstede en L0 trigger. L1 avgjørelsen skal bli avgjort i TOR, men den vil vi ikke komme nærmere inn på i denne oppgaven. L1 trigger er foreløpig ikke implementert i PHOS. Konfigurering av TOR blir gjort fra avstand via et DCS (Detector Control System) kort som er koblet til TOR. Dette kortet er ansvarlig for å konfigurere, lese av og kontrollere FEE over Ethernet (3).

3.4 RCU

RCU (Readout Control Unit) består av tre kort og står for utlesningen av FEC og TRU og å sende det til DAQ systemet. Disse kortene er hovedkortet RCU, et SIU (Source Interface Unit) kort som er optisk koblet til DAQ systemet for data overføring og et DCS kort som er optisk koblet til TTC (16) systemet. RCU kjernen består av en FPGA hvor utlesningen og kontrollen av FEC og TRU er implementert. Utlesningsdelen bruker ALTRO bussen til dataoverføring på DAQ sin kommando, mens kontrolldelen bruker DCS og ”slow control bus” (DCS buss, se figur 3.3). RCU er brukergrensesnittet mellom FEE og ALICE online system hvor funksjonene og protokollene er like for alle detektorene. Triggere fra TTCrx kommer inn og blir dekodet på DCS kortet og sender disse videre inn på to kontroll linjer i kontroll bussen, L1 og L2 (egentlig L0 og L2 for PHOS). Se figur 3.3.

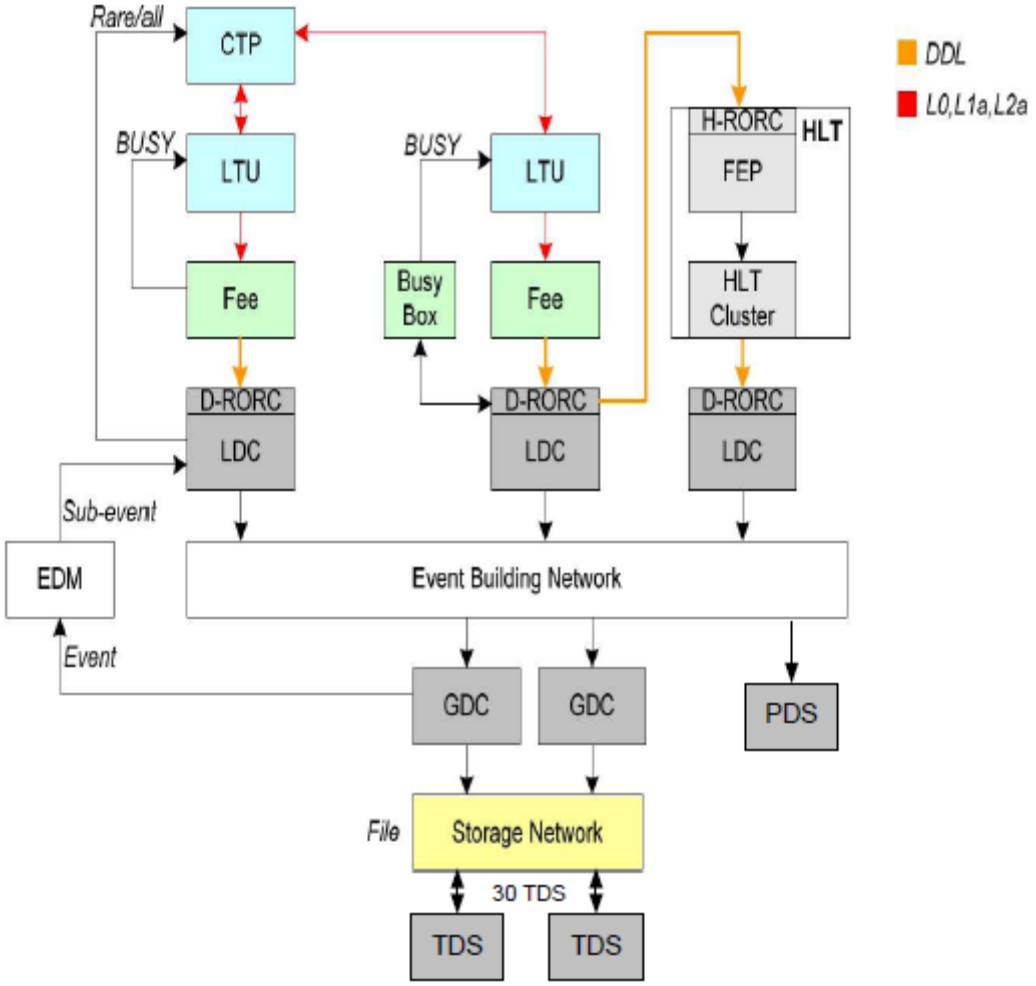
Ved en L0 trigger så begynner lagring av data i buffere samtidig som bufferen som skal videre til DAQ blir flagget. Etter en L2 trigger så overføres dataen fra den flaggede ALTRO bufferen til DAQ via DDL (Data Detector Link). DDL er en optisk fiber kabel som overfører dataen fra RCU til DAQ (se figur 3.3). Før dataen går over til DDL så blir den dekodet og enkodet i RCU for å være kompatibel med DAQ. Selve utlesningen blir styrt fra DAQ, mens konfigurering av registre blir styrt fra ALICE via DCS-kortet på RCU som kjører Linux i en softcore prosessor. Kommunikasjon til FEC og TRU foregår på samme metode, eneste forskjellen er adressen.



Figur 3.3 Oversikt over RCU kortets tilknytning til FEE og DAQ systemet (3)

3.5 DAQ oppsettet

Hovedfunksjonen til DAQ systemet er å få dataen opp fra detektoren til den sentrale datalagringsplassen til ALICE. På figur 3.4 så ser man en oversikt over hvordan DAQ er bygd opp, når trigger signalene når detektorene fra CTP så vil de sende rådataen til DAQ via DDL. DDL kan overføre data i begge retninger og alle subdetektorene har den samme protokollen for dataoverføring. DDL består av en SIU del som er knyttet til FEE og en DIU (Destination Interface Unit) som er knyttet til D-RORC. Dataen som blir overført til D-RORC (Data ReadOut Receiver Card) er knyttet til LDC (Local Data Concentrator). Fra D-RORC blir dataen overført til minne på en LDC, hver LDC-en kan få inn data fra forskjellig D-RORC samtidig inn i minne. Dataen som kommer inn på LDC er del-hendelser og uansett hvor forrige del-hendelse ble sendt så blir destinasjonen til denne bestemt ut i fra direksjonene til EDM (Event Destination Manager). EDM har en kommunikasjon med både LDC og GDC og vet dermed hvor neste del-hendelse passer best. Her går dataen videre til GDC (Global Data Concentrator) hvor dataen blir bygd opp igjen til hele hendelser og sendt til lagring (17). Event Building Network som man ser på figur 3.4 er et standard kommunikasjonsnettverk støttet av TCP/IP protokoll.

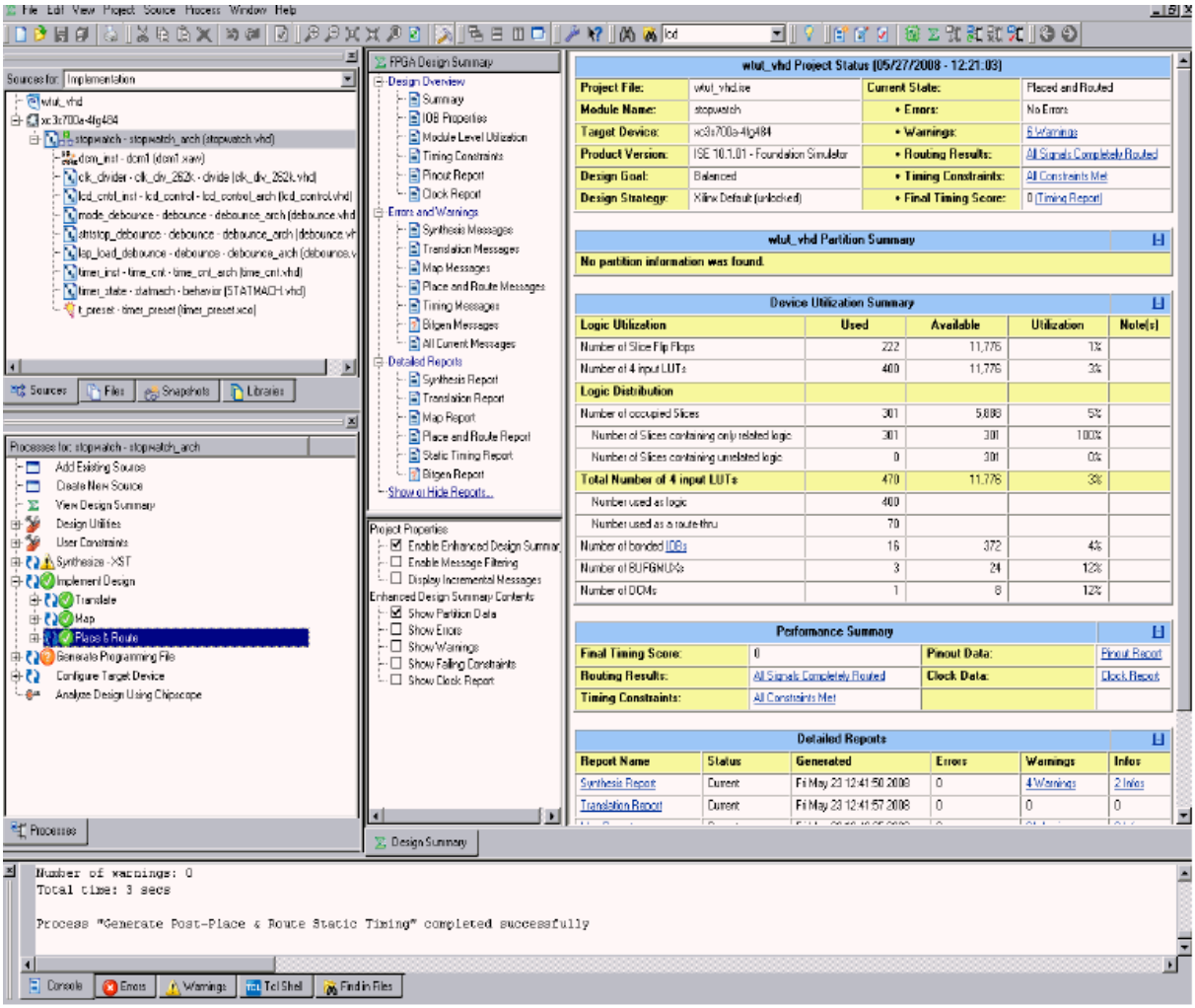


Figur 3.4 Oversikt over DAQ systemet (3)

HLT vil også få en kopi av rådataen for prosessering via DDL og H-RORC (HLT ReadOut Receiver Card), hvor den prosesserte dataen er sendt til LDC. Programmet DATE (Data Acquisition and Test Environment) blir brukt for å kjøre DAQ systemet. DATE har blitt lagd slik at den skal fungere like bra på både små og store systemer. Ved bruk i lab så kan systemet bruke bare en enkel prosessor og utføre alle funksjonene sånn som LDC, GDC osv. Konfigurasjonen av detektorer, oppbygging av hendelsene og utlesningen er alle utført av DATE (17). Etter hendelsene har blitt bygd opp igjen så blir de først lagret i TDS (Transient Data Storage) som utfører den midlertidige lagringen før GDC flytter den videre til PDS (Permenant Data Storage) som videre kan bli hentet via nettverk.

4 Xilinx Integrated Software Environment (ISE)

I 1985 oppfant Xilinx den første kommersielle FPGA (Field Programmable Gate Array). En FPGA er en integrert krets som er designet for å kunne konfigureres og rekonfigureres om og om igjen ut ifra hva den skal brukes til. Konfigurasjonen blir spesifisert i denne oppgaven ved hjelp av dataspråket VHDL. En FPGA kan brukes til å implementere en hver logisk funksjon som man tidligere har brukt ASIC (Application Specific Integrated Circuit) til. Fordelen med FPGA er det at man kan forandre deler av hardware (kalt firmware) for å tilpasse seg nye bruksområder. Kretsen inneholder programmerbar logikk som kan brukes til både komplekse kombinatoriske funksjoner og enkle logiske porter som ”OG” og ”ELLER”.
Xilinx ISE er et utviklingsverktøy produsert av Xilinx for å kunne kompilere og analysere DHL design. Utvikleren kan med dette kompilere og kjøre tidsanlayser, utforske signalflyten og simulere reaksjonen til programmet i forhold til forskjellige scenario som skal simuleres. ISE kontrollerer alle aspekter av designflyten gjennom ”Project Navigator Interface” se figur 4.1. Herfra kan man få adgang til alt av implementeringsverktøy og filer og dokumenter som er knyttet til prosjektet. Brukergrensesnittet er delt opp i fire deler, øverst i venstre hjørne er ”Sources vinduet” som viser en hierarkisk oversikt over elementene i prosjektet. Under dette vinduet ligger ”Processes vinduet” og her ser man hvilke prosesser som er tilgjenglig for den valgte kilden (fra Source vinduet). Helt nederst på figuren under så er det et vindu som kalles ”Transcript window” som viser status, feil og advarsler. Siste vinduet er et multi document interface (MDI) vindu som kalles ”Workspace”. Her kan man se på blokkskjema, skrive kode og simulering for å sjekke om programkoden virker som den skal.



Figur 4.1 Brukervinduet til Xilinx ISE

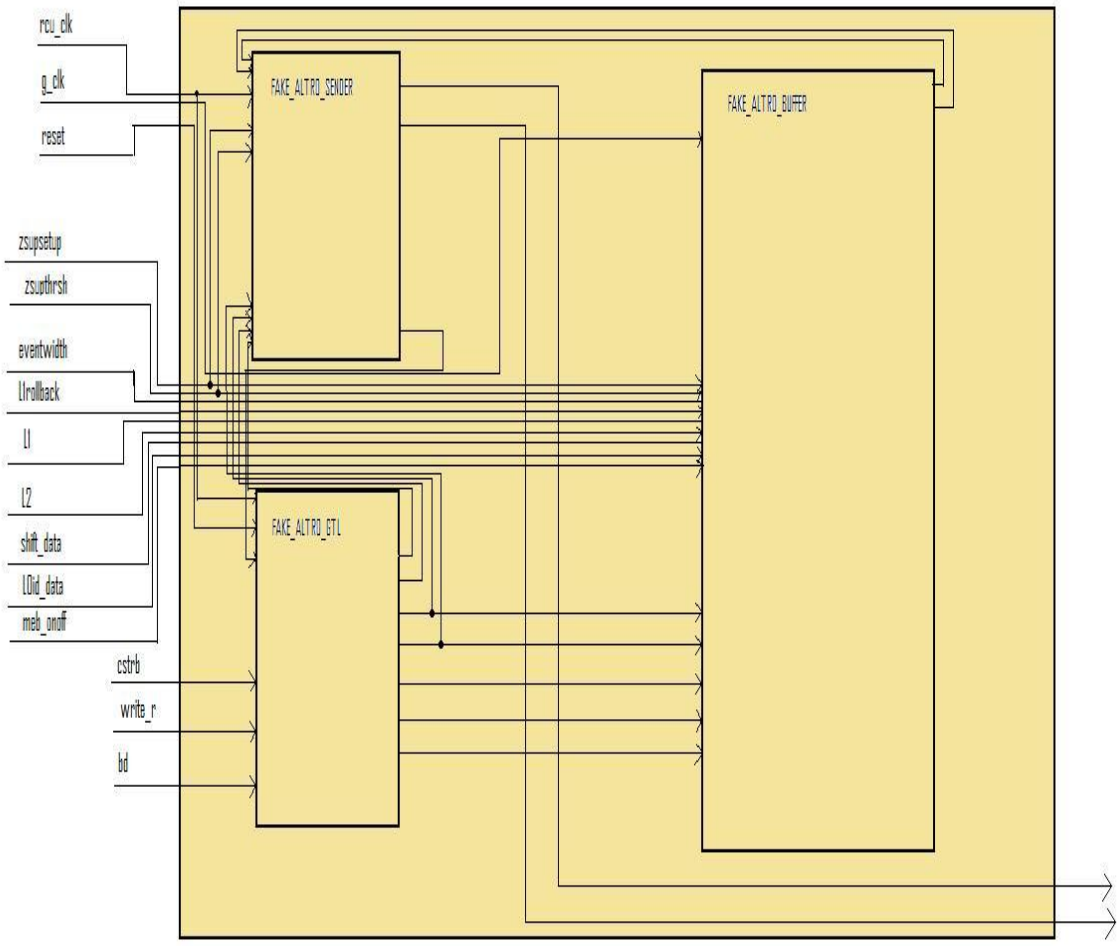
5. Fake ALTRO programflyt i EMCal TRU

I dette kapittelet blir det gått gjennom programvaren i EMCal TRU, utviklet av Jiri Kral, med ((18)), hvordan programmet skal svare på input fra RCU og hvilke signaler som blir sendt tilbake. På grunn av omfanget til programmet og den korte tiden som er til rådighet, så vil prioritering ligge på Fake ALTRO delen av programmet. Dette er fordi man ved sammenligning av Fake ALTRO data og ALTRO data kan se om det er riktig data som blir lagret i buffrene til Fake ALTRO. For å kunne videreutvikle EMCal TRU til å fungere i en PHOS detektor så er det viktig å vite hvordan Fake ALTRO fungerer. Hvor mange signaler og hvor signalene kommer fra, hva som blir gjort med dem og hvor de så går ut igjen. Dette er ting som finnes i programvaren og som må tilpasses for å bruke i PHOS.

Som vi ser i figur 5.1 så er Fake ALTRO bygd opp av flere subprogrammer, vi har: FAKE_ALTRO_BUFFER, FAKE_ALTRO_GTL og FAKE_ALTRO_SENDER. Disse tre subprogrammene kommuniserer med hverandre inne i hovedblokken Fake ALTRO. FAKE_ALTRO_GTL er der hvor instruksjonene fra RCU kommer inn (via ALTRO bussen). FAKE_ALTRO_SENDER får dataen fra bufferen og adressene fra GTL og sender ALTRO bussen tilbake til RCU med den dataen som ble forespurt. FAKE_ALTRO_BUFFER er der hvor rådataen går konstant gjennom en sirkulær buffer, 960 bit kommer inn den globale klokkens stigende flanke hvert 25 ns (40MHz). Når L1 blir satt så begynnes det å lagre data fra den sirkulære bufferen inn på MEB (Multi Event Buffer) før den sendes videre til FAKE_ALTRO_ SENDER og ut fra Fake ALTRO til RCU. RCU klokken har også en 40 MHz frekvens og er synkronisert med den globale klokken LHC, så dataen ut fra MEB går hvert 25 ns. Med all dataen som blir lagret i MEB så er muligheten der for at lagrene blir fulle. Da vil det settes et ”busy” signal og CTP vil slutte å sende ut triggere helt til dette signalet blir sluppet. I EMCal så begynner lagringen av data på en L0 trigger og selve utlesningen starter når L2 blir satt. For PHOS, EMCal og TPC så er det et eget dedikert kort, ”BusyBox” som tar seg opptatt tilstanden ved å kommunisere med DAQ og CTP (18).

ALTRO og RCU var originalt lagd for TPC (Time Projection Chamber) detektoren og TPC startet å lagre data på en L1 trigger og ikke L0 som det blir gjort i dette tilfellet. Derfor så heter signalet inn på Fake ALTRO L1 istedenfor L0.

Her i oppgaven så er det en channel readout vi er interessert i og jeg vil derfor fokusere på dette når jeg går gjennom programmet.

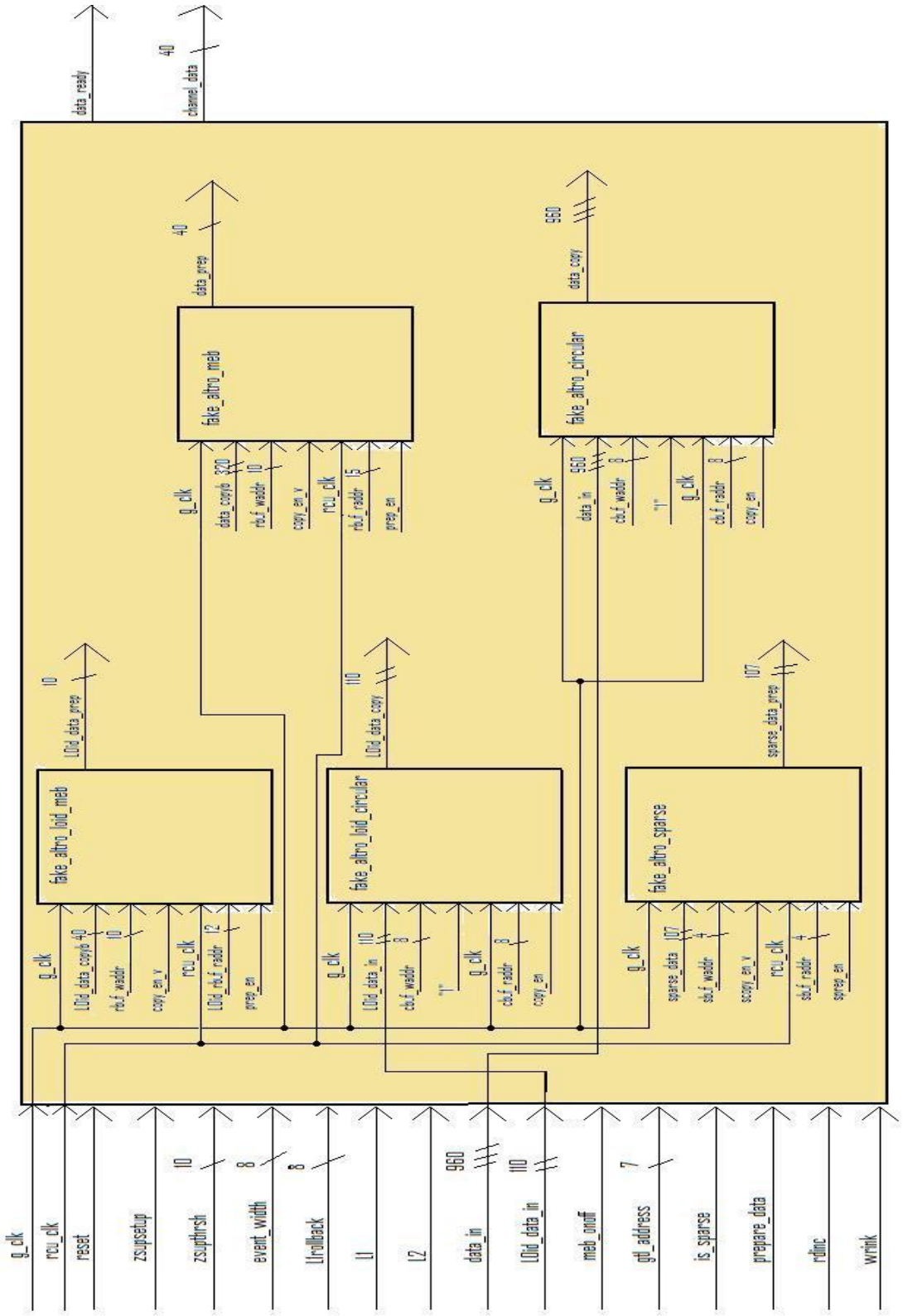


Figur 5.1 Blokkskjema over hele Fake ALTRO

5.1 Fake ALTRO buffer

Som vi ser ut fra programflyten på figur 5.2 så kommer det 960 bit inn på buffer-blokken, dette er 10 bit fra hver av de 96 kanalene. Når RCU sender en kommando til EMCal TRU for utlesning så er det fra en og en kanal om gangen. Hvilken kanal som skal leses ut kommer fra signalet gtl_address som går fra GTL til både sender og buffer. Hver kanal kan gi N antall 10-bit ord og N er satt som 16 ved default i Fake ALTRO. Disse 16 10-bit ordene består av 14 ord med data og 2 header ord når zero suppression[3] er av. Zero suppression slår man av på gjennom TRU registrene (mer om dette i kapittel 6). 16 10-bit ord er det samme som 4*40-bit som er sånn det kommer ut via channel_data der hvor det kommer 40 bit om gangen. Når man gjør en readout så leser man ut N-2 10-bit ord fra hver av de 96 kanalene og på denne måten får man skrevet ut alle 960 bit med en syklus bredde på N-2 10-bit ord. Hvis zero suppression er på så vil de 16 ordene til data og header endre seg i forhold til dataen som er lagret.

[3] Zero suppression sammenligner signalet med en gitt pidestallverdi og blir forkastet som støy hvis det er under denne verdien. Dette er for at kun de mest interessante dataene skal gå videre til lagring.



Figur 5.2 Blokkskjema av Fake ALTRO Buffer

Når det er zero suppression så antyder dette at noen signaler blir kuttet ut fordi de er under en viss verdi, da er det viktig at man har tidsinformasjon på hvor de ble kuttet ut og informasjon om hvor de signalene som ikke blir kuttet ut kommer fra. I buffer koden så står ”timestamp” for denne tidsinformasjon. Datapakkene går rett etter hverandre med en header som sier hvor mange 10-bit ord som er i pakken (subwordcount), dette er for at det skal være mulig for å dekode dataen. Når hele overføringen er ferdig så er det en global header som inneholder informasjonen om hvor mange 10-bit ord det var i alle datapakkene til sammen (wordcount).

960 bit går inn på bufferen og videre inn til den sirkulære, hvor dataen hopper ett hakk oppover for hver syklus og så ut av bufferen. Hvis det kommer en L1 trigger som da indikerer at det er sampler som er verdt å ta vare på så begynner denne dataen å bli lagret.
Programmet tar dataen ut fra sirkulære bufferen og setter dem inn i fire separate lese buffere. Dette er for å distribuere kanalene sånn at den første kanalen går til den første rammen og andre til andre helt opp til fire og så går det fra den 5. kanalen til den første rammen og sånn roteres det oppover 24 * 4 ganger til 96. Ut av sirkulær bufferen kommer det 24*10 bit, fire ganger. Hver og en av disse 240 bit-ene blir lagt til 80 nuller (=320 bit) for å tilpasse seg bredden på neste buffer. Denne måten å bufre dataen på er gjort for å optimalisere FPGA ressursene. Det er Fake ALTRO bufferen som har kontrollen over hvor dataen blir lagret og hvordan den blir sortert for så å bli sendt ut kanal for kanal ved en utlesning. Bufferen venter på et signal prepare_data fra GTL som vi ser i koden under.
Først så resettes signalene og telleren og når prepare_data går til ’1’ så settes prep_en for å kunne lese ut og bufferen gjør klar hvilke adresse som skal leses ut i forhold til hvilken informasjon den fikk fra gtl_address og så går den videre til neste tilstand.

```
CASE prepstate IS
                WHEN idle =>
                  -- reset all
                  data_ready <= '0';
                        prep_en <= '0';
                        sprep_en <= '0';
                        counter <= (OTHERS => '0');

                  IF prepare_data = '1' THEN
                          prep_en <= '1' AND (NOT is_sparse); -- enable memory read for data
                                sprep_en <= '1' AND is_sparse;


                          rbuf_rframe <= rbuf_rbase & "00000";
        --
```

```
                    CASE ram_id IS
                    WHEN "00" =>
                      rbuf_rplus1 <= "000000000000000"; -- no shift     32*0
                      rbuf_rplus2 <= "000000000100000"; -- one shift    32*1
                      rbuf_rplus3 <= "000000001000000"; -- two shifts   32*2
                      rbuf_rplus4 <= "000000001100000"; -- three shifts 32*3
                    WHEN "01" =>
                      rbuf_rplus1 <= "000000001100000"; -- three shifts
                      rbuf_rplus2 <= "000000000000000"; -- no shift
                      rbuf_rplus3 <= "000000000100000"; -- one shift
                      rbuf_rplus4 <= "000000001000000"; -- two shifts
                    WHEN "10" =>
                      rbuf_rplus1 <= "000000001000000"; -- two shifts
                      rbuf_rplus2 <= "000000001100000"; -- three shifts
                      rbuf_rplus3 <= "000000000000000"; -- no shift
                      rbuf_rplus4 <= "000000000100000"; -- one shift
                    WHEN "11" =>
                      rbuf_rplus1 <= "000000000100000"; -- one shift
                      rbuf_rplus2 <= "000000001000000"; -- two shifts
                      rbuf_rplus3 <= "000000001100000"; -- three shifts
                      rbuf_rplus4 <= "000000000000000"; -- no shift
                    WHEN OTHERS =>
                      rbuf_rplus1 <= "000000000000000"; -- no shift     32*0
                      rbuf_rplus2 <= "000000000100000"; -- one shift    32*1
                      rbuf_rplus3 <= "000000001000000"; -- two shifts   32*2
                      rbuf_rplus4 <= "000000001100000"; -- three shifts 32*3
    END CASE;
```

I denne tilstanden settes data_ready = ‘1’ og bufferen blir rotert hver syklus samtidig som telleren hopper ett hakk oppover. I tilstanden etter så blir channel_data fylt opp med ønsket data for å sendes videre og så stoppes det når det er telt til 4*40 bit som er sendt og så går data_ready tilbake til 0.

5.2 Fake ALTRO GTL

I denne blokken får man som sett ut fra figur 5.3 kommando fra RCU om hva som skal gjøres og det vi er på utkikk etter er en CHRDO (channel readout), det vil si at bitene 24 ned til 20 i bd (se figur 5.3) må være satt til ”11010”. I programmet under ser vi at når tilstanden er ”idle” så sjekkes det om cstrb = ’0’ som da indikerer at det er en kommando på bussen, sjekker så videre hva slags kommando det er. Ser da på de fem siste bitene av ”0x01a = 0000 0001 1010” og hvis dette er en CHRDO kommando så settes signalet is_data_ready høyt (dette signalet og dets innvirkning kommer vi tilbake til senere), s_ackn blir satt til ’0’ (alle signalene på ALTRO-bussen bortsett fra AD[39:0] er aktiv lave signaler). Gtl_address blir

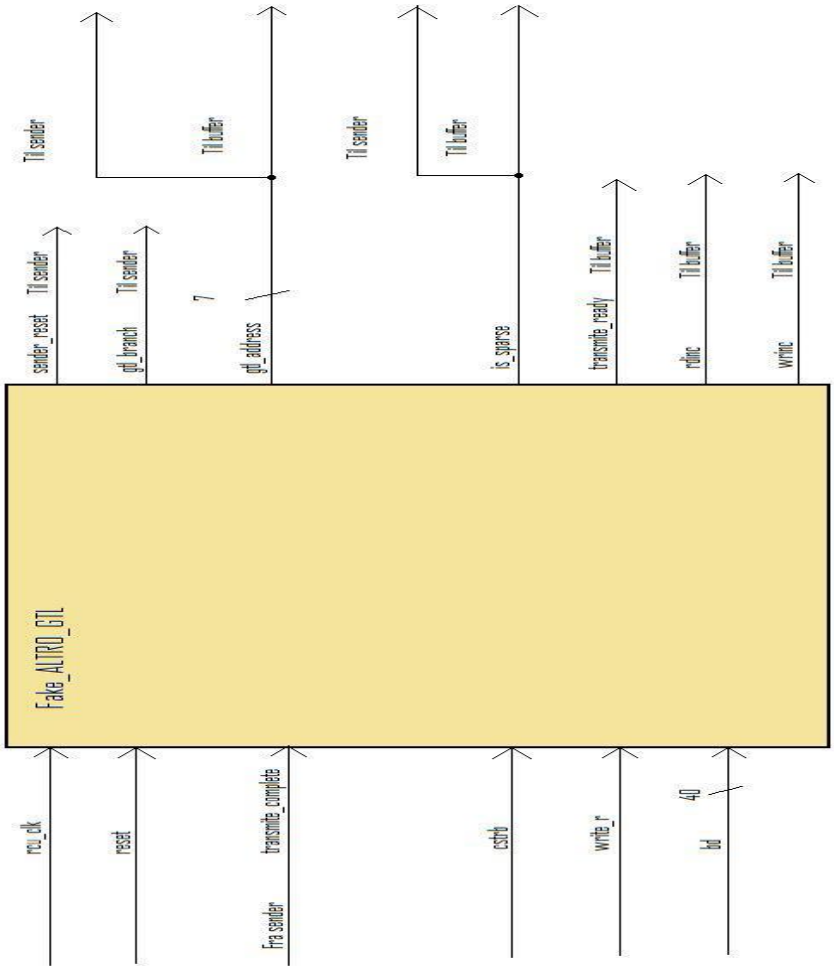
satt med bitene fra bd[31:25] og går videre inn på Fake ALTRO bufferen og sender, gtl_branch blir satt med bit 36 til bd og går til sender.

```
CASE state IS
   WHEN idle =>

      wrinc <= '0';
      gtl_busy <= '0';

        IF cstb = '0' THEN  -- cstb asserted (to 0) = command arriving        -- remove the sparse flag
          is_sparse <= '0';

                              -- check the broadcast, FEC address, write and instruction
                              -- channel readout command
            IF (( bd(38 downto 37) & bd(35 downto 32) & writeb & bd(24 downto 20)) = x"01a ")
            THEN
                        is_data_read <= '1';
                        s_ackn <= '0'; -- set acknowledge
                        gtl_address <=  bd(31 downto 25);
                        gtl_branch <= bd(36);
                        gtl_busy <= '1';
                        state <= release_ackn;
```

Når dette er gjort går programmet videre til tilstanden ”release_ackn”. Her venter man til cstb har gått tilbake til ’1’ og så gjør s_ackn det samme. Nederst i dette subprogrammet så blir s_ackn satt til ackn og cstb kan som sagt tidligere ikke gå tilbake til ’1’ før ackn er satt. I denne tilstanden kommer vi tilbake til is_data_read og som man ser i programteksten under så går man videre til tilstanden ”wait_out”. Når man går gjennom denne tilstanden og neste så blir transmit_ready signalet som går til bufferen satt (mer om dette signalet under Fake Altro buffer)før man så sjekker om transmite_complete signalet fra sender er satt og stopper der for å så gå tilbake til begynnelsen.

```
 WHEN release_ackn => -- wait for cstb release and release ack
                        IF cstb ='1'
                        THEN
                                s_ackn <= '1';
                                IF is_data_read = '1'
                                THEN
                                  state <= wait_out;
```

```
WHEN wait_out => -- wait
                        state <= ctrls_out;
                WHEN ctrls_out =>
                        state <= data_out;

                WHEN data_out => -- wait for data transmit end
                        IF transmit_complete = '1' OR cstb = '0'
                         THEN
                                state <= stop;
                        ELSE
                                state <= data_out;
                        END IF;
                WHEN stop =>
                        state <= idle;
```



Figur 5.3 Blokkskjema av Fake ALTRO GTL

5.3 Fake ALTRO Sender

Denne blokken sammenligner verdiene i channel_data med en gitt verdi og luker ut de dataene som ligger under dette nivået, zero suppression. I tillegg så holder den orden på hvor mange 10-bit ord som blir sendt og tidsinformasjonen til de forskjellige ordene. Subprogrammet består av fire tilstander: Idle, Bufferout, Lastout og finish. Som vist på figur 5.4 så får ”sender” både signaler fra ”buffer” og ”GTL” og signaliserer tilbake til GTL når den er ferdig med å overføre data (transmit_complete).

I koden under så ser vi sammenligningen av de lagrede samplene med zero suppresion threshold. Hvis ”intersum(0:3) = ’1’ så vil det si at dataen fra channel_data ikke nådde over den gitte verdien og er derfor ikke verdt å ta vare på (mest sannsynlig støy).

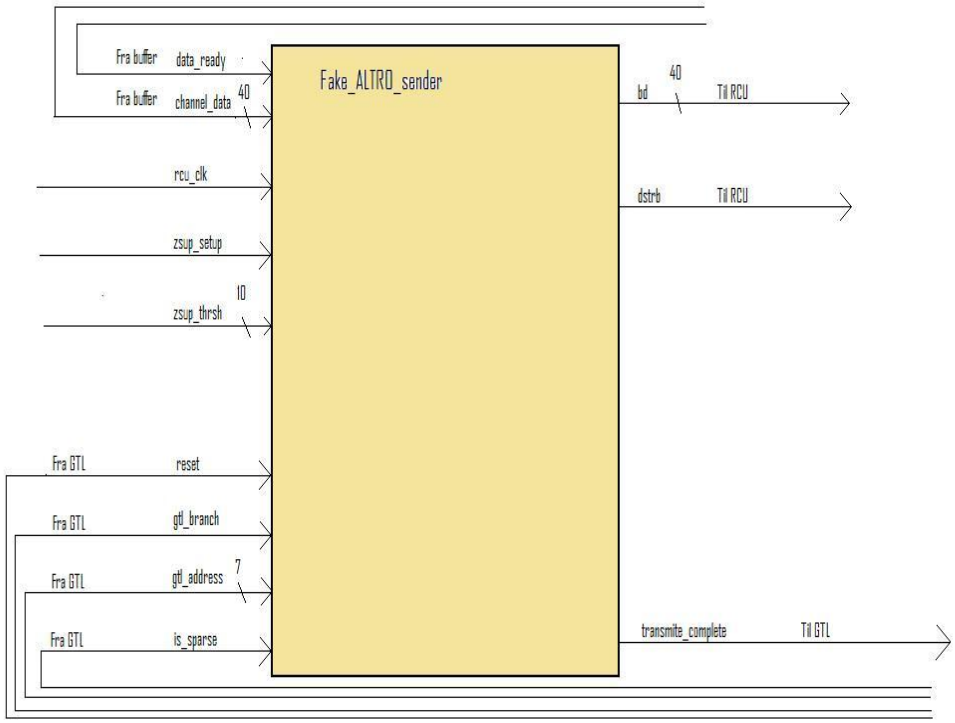
```
intersum(0) <= ('1' & zsup_thrsh) - ('0' & channel_data(9 downto 0));
intersum(1) <= ('1' & zsup_thrsh) - ('0' & channel_data(19 downto 10));
intersum(2) <= ('1' & zsup_thrsh) - ('0' & channel_data(29 downto 20));
intersum(3) <= ('1' & zsup_thrsh) - ('0' & channel_data(39 downto 30));

-- zero suppression green light on two higher words below threshold
-- or end of data
zsuppoknorm <= (intersum(3)(10) AND intersum(2)(10)) OR (NOT data_ready);
-- last frame suppression green light
```

I den første tilstand “idle” så blir signaler som vi skal bruke nullstilt og det ventes på en “data_ready = ‘1’”, til denne kommer så forblir man i denne tilstanden. Når signalet “data_ready = ‘1’” kommer så går man til ”bufferout, hvor dataen fra bufferen blir satt inn på ALTRO-bussen og sendt til RCU. For hver gang det blir sendt 40 bit data til RCU så går ”wordcount” opp med fire, dette er fordi det er fire 10-bit ord som blir sendt. ”Timestamp” øker også med fire for hver gang for å holde orden på tidsinformasjonen.

```
IF data_ready = '0' THEN -- put in the pre-last frame with headers
  IF is_sparse = '1' THEN -- no trailer frames for sparse list
          bd <= channel_data;
                state <= finish;
                transmit_complete <= '1'; -- transmit complete with last data
        ELSE
   state <= lastout;
          -- wordcount timestamp frame frame
   bd <= (wordcount + "0000000100") & (timestamp + "0000000001")
        & channel_data(19 downto 0);
  END IF;
ELSE
  bd <= channel_data;
  state <= bufferout;
END IF;

wordcount <= wordcount + 4; -- 4 ten bit words in frame
```

Dette skjer helt til data_ready går til ’0’ hvor vi da går videre til neste tilstand ”lastout”. Her settes: bd <= (x"aaa" & "10" & wordcount & x"a" & gtl_branch & x"0" & gtl_address). Alle heksadesimale a-ene som er her er kun for å fylle opp bussen. De 14 første bitene er fyllmasse så kommer 10 bit (wordcount) som inkluderer hvor mange 10-bit ord som er sendt og tidsinformasjonen. En 0xa som er mer fyll og de siste 12 bit er adressen (1 bit branch, 4 bit som er FEC plasseringen som vi ser er 0x0 og dette er plassen til TRU). 3 bit er for hvilken

chip og de 4 siste bit er for hvilken kanal inne i chipen). Bussen blir sendt og transmit_complete blir satt til ’1’ og går så videre til tilstand finish hvor dstb_block blir satt til ’1’ (aktiv lav) og vi går tilbake til første tilstand idle.



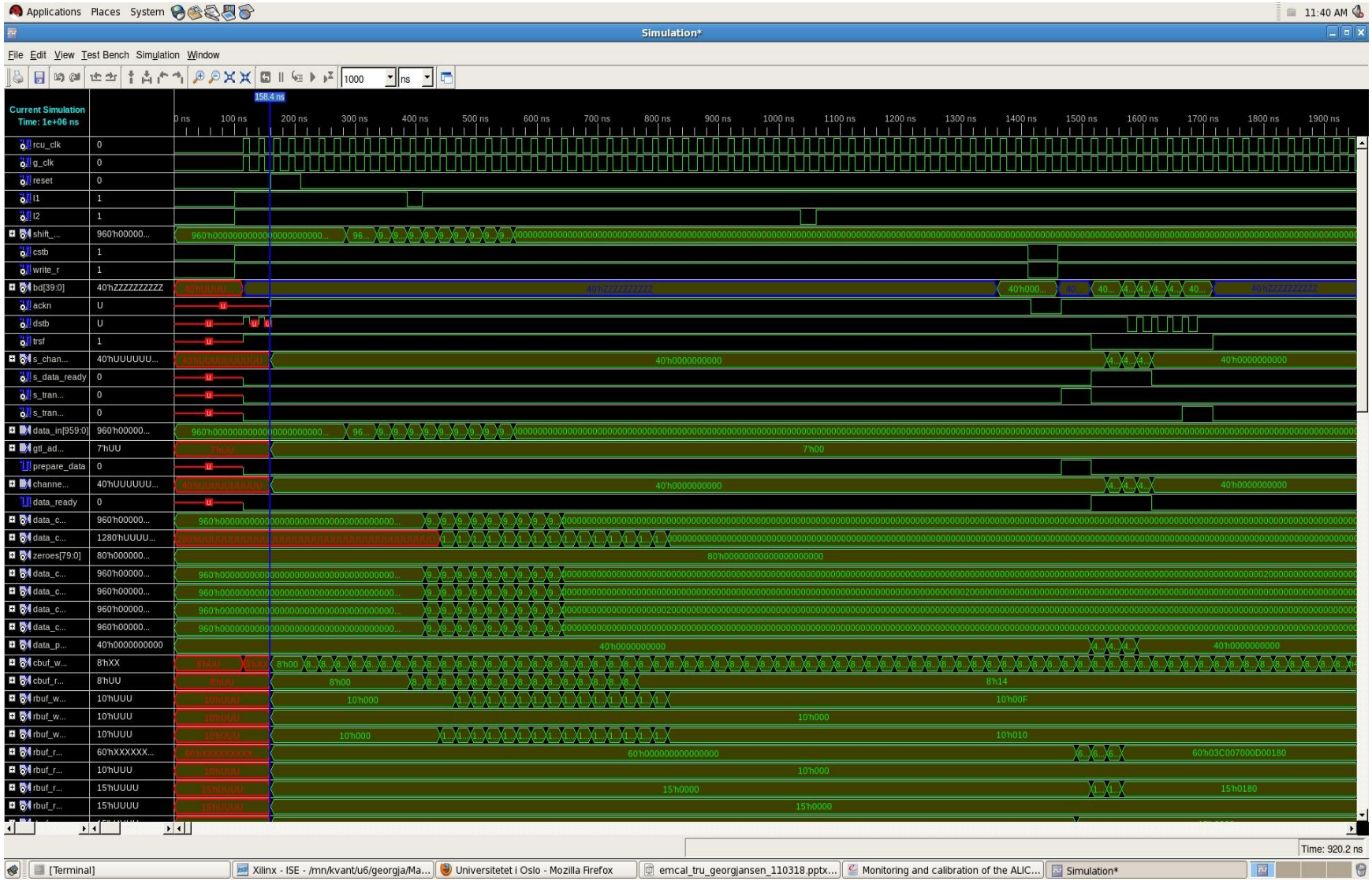
Figur 5.4 Blokkskjema av Fake ALTRO sender

5.4 Simulering av prosessen

I figur 5.5 så ser vi en simulering av hele prosessen når en kanal blir utlest. Dette bekrefter at programmet over virker som det skal. L1 går aktiv lav og copy_en åpner bufferen for lagring ved neste rising_edge på klokken. Neste slag så blir dataen delt inn i de fire bufferene for videre klargjøring. Vi får en bekreftelse om at det virkelig skjer en utlesning når L2 settes. RCU begynner som masteren og så tar Fake ALTRO over som master og styrer hva som går på bussen og går tilbake til slave igjen. Signalene cstb og write_r fra RCU settes og TRU svarer med å sette ackn signalet. Prepare_data settes og rett etter følger data_ready og trsf og dataen sendes mens dstb bekrefter at det blir sendt data. På bd-linjen så ser vi at bussen blir fylt opp med med bare 1–ere før de 4*40-bit-ene kommer og til slutt så sendes de 40 bit-ene som vi så i lastout tilstanden i senderblokken ”bd <= (x"aaa" & "10" & wordcount & x"a" &

gtl_branch & x"0" & gtl_address)”. Det var informasjonen RCU ba om for den ene kanalen og så sender den for neste og prosessen gjentar seg selv helt til siste kanal har blitt lest ut.

Denne simuleringsprosessen er som nevnt en av delmålene i oppgaven.



Figur 5.5 Simulering av programflyten i Xilinx

6. Fremgangmåte og resultater

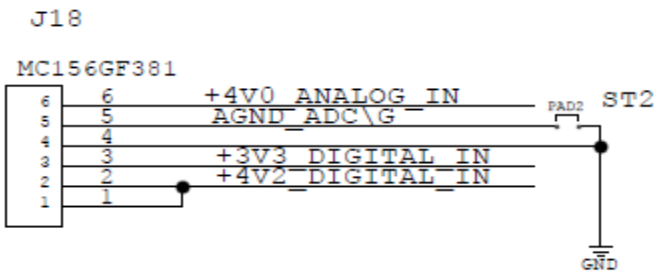
Dette kapittelet står for den praktiske biten av oppgaven. Her vil det bli vist hvordan elektronikken har blitt satt opp med EMCal TRU istedenfor PHOS TRU, og hvordan FPGA-en til EMCal har blitt programmert og konfigurert for utlesning av data fra Fake ALTRO. Det vil også vises hvilke resultater som har kommet ut fra de forskjellige stadiene i prosessen. Herunder også hvilke endringer som har blitt gjort underveis.

6.1 Xilinx

Til å begynne med så måtte man sette seg inn i utviklingsverktøyet Xilinx ISE, se kapittel 4. Programvaren ble lastet ned fra nettsiden ”twiki” (19) og ble vellykket kompilert etter kort tid. For å simulere blant annet figur 5.5 så var det noen verdier i simuleringsfilen som måtte defineres og legges til før simuleringen kunne begynne. Dette var veldig nyttig for å kunne se at signalflyten gikk som den skulle og for å få en videre forståelse av Fake ALTRO kommunikasjonen med RCU under en utlesning.

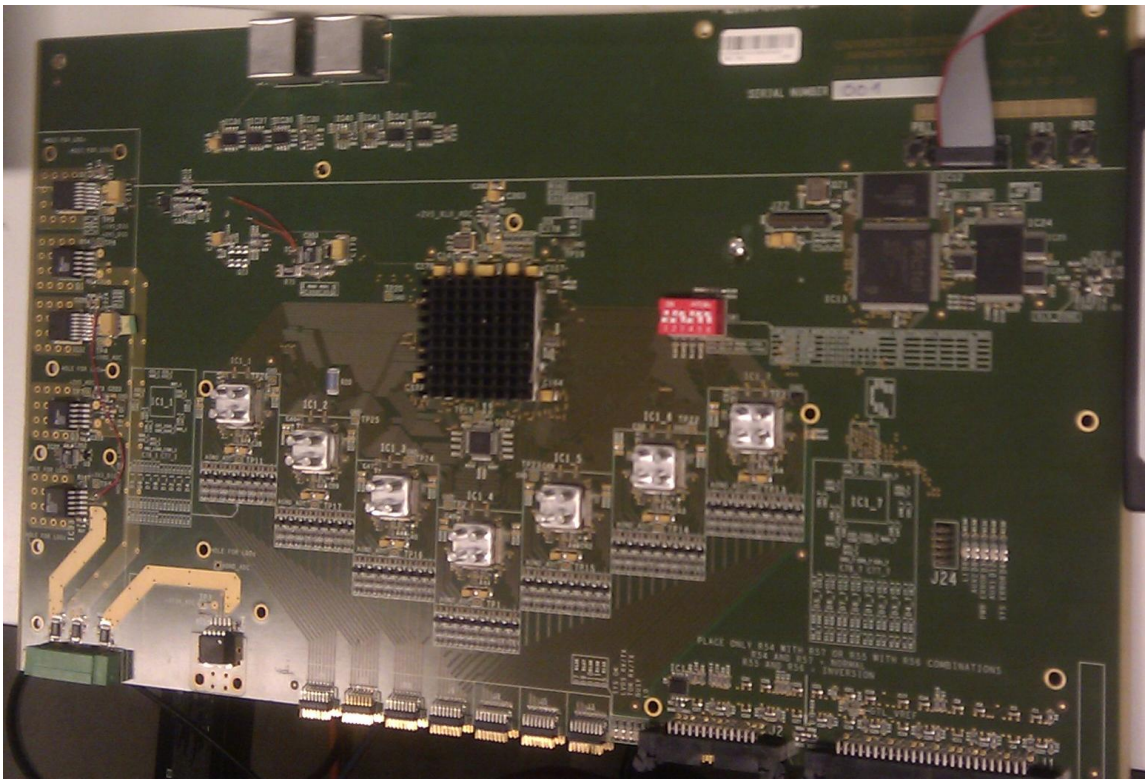
6.2 Oppkobling med EMCal TRU i Bergen

Figur 6.1 er hentet fra TRU_Schematics (20)og leser her hvordan kortet kobles opp.



Figur 6.1 Spenninger tilkoblet TRU

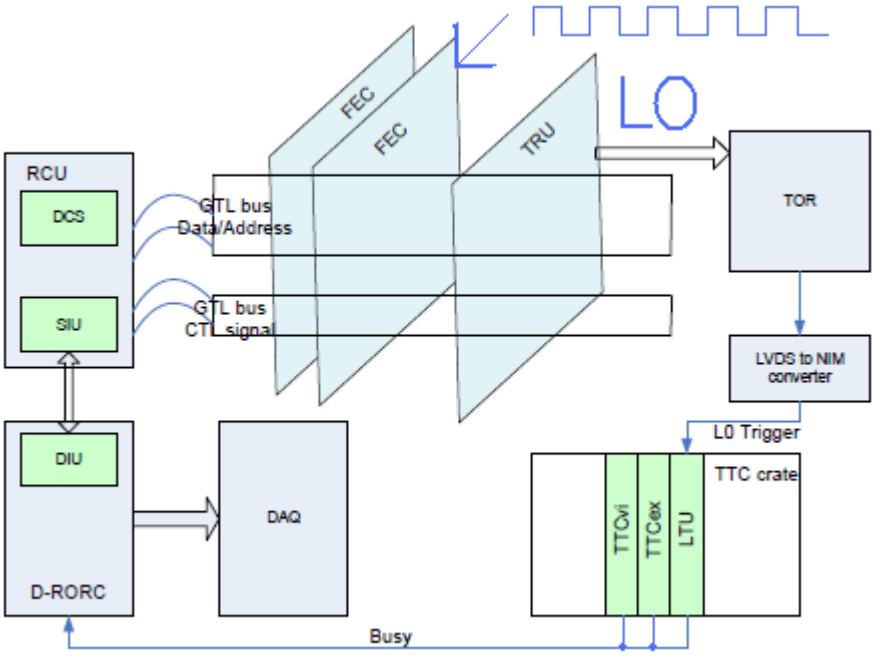
Pinne 1 og 2 kobles til 4.2 V, pinne 3 kobles til 3.3 V, 4 og 5 går til jord og pinne 6 kobles til 4.0 V. I figur 6.2 så ser vi EMCal TRU som ble lånt fra EMCal-gruppen i CERN og nederst til venstre er strømtilkoblingen. TRU kortet er utstyrt med 14 ADC. 7 av de ses på bildet under mens de resterende er på undersiden av kortet.



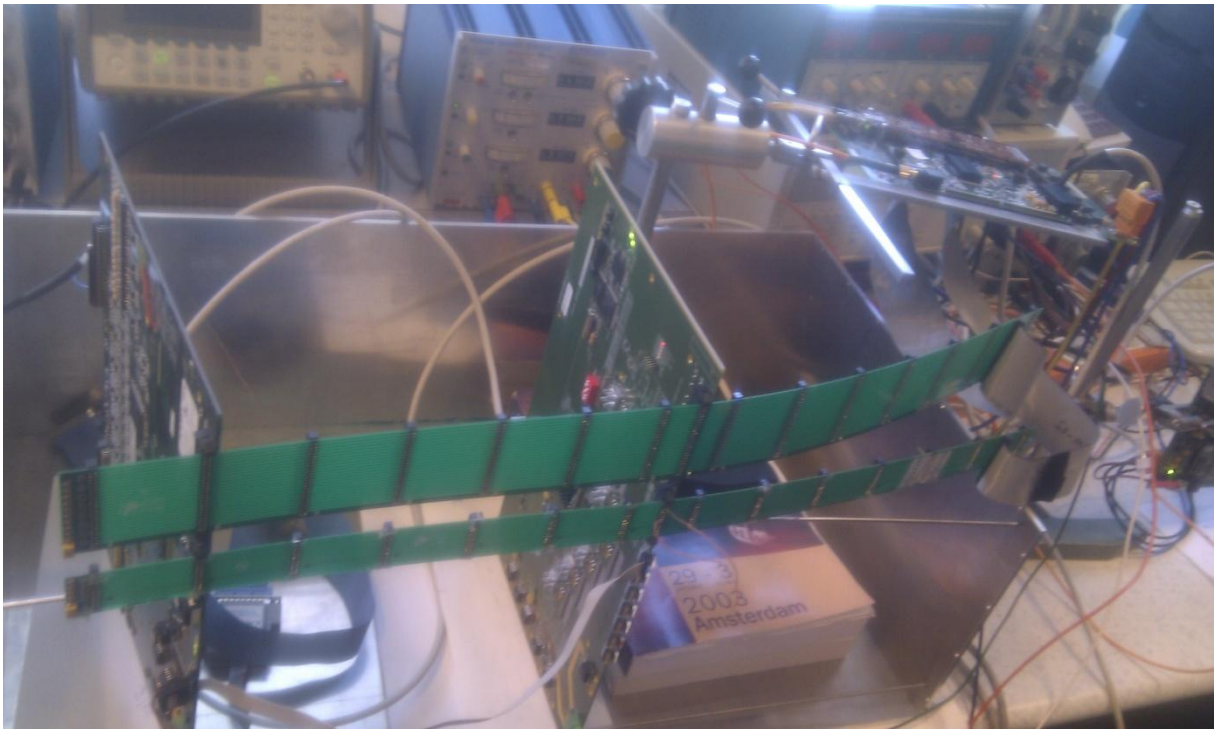
Figur 6.2 TRU-kortet som ble lånt fra CERN

På figur 6.4 så er det kortet i midten som er TRU kortet. PHOS TRU kortet hadde samme plassering før det ble byttet ut med EMCal.

Oversikten over testlab oppsettet i bergen ses ut fra figur 6.3. Den består av en TTC-del, et FEC, en TRU, en TOR, en RCU og LDC. FEC og TRU er koblet til RCU via en GTL buss mens TOR og TRU overfører signaler seg imellom med en nettverkskabel. RCU og TOR har hvert sitt DCS kort som muliggjør nettverkskommunikasjon. Testsignalene inn på FEC kommer fra en puls generator. Utgangssignalet fra pulsgeneratoren har en stigningstid på 20ns og frekvens på 1 Hz



Figur 6.3 oversikt over testlab-en i Bergen (3)



Figur 6.4 Bilde av den fysiske lab-en i Bergen

6.3 Programmering av TRU

Etter oppkoblingen så var FPGA-en klar for programmering. Dette ble gjort ved å først slette programmet som ligger i flash minne for deretter å programmere med det aktuelle programmet (se bruksanvisning (21)). Selve programmeringsfilene er blitt produsert ved hjelp av Xilix Impact softvare og er hentet ned sammen med resten av programvaren. Minnet i selve FPGA-en blir slettet når strømmen kobles fra. Det er derfor vanlig å programmere flash-driveren, fordi denne beholder minne selv uten strøm og programmerer FPGA-en automatisk ved oppstart. Etter alt var koblet opp så ble TRU flash programmert, to LED lyste grønt som indikasjon på at programmeringen av FPGA var vellykket. Leste i tillegg ut register 0x20 for å sjekke om det var riktig firmware versjon. Gikk deretter gjennom resten av registrene for å sjekke at de var satt opp riktig, kommunikasjon med TRU var etablert. Tabell 6.1 viser verdiene til de forskjellige registrene og dette stemmer med spesifikasjonene som var satt opp. Register 0x20 leses ut ved kommandoen ”./TRU_read.sh A 20”. Det at det kommer ut ”error” på de fire øverste bitene i status registeret 0x21 var ikke noe man skulle ta hensyn til i følge Jiri Kral.

TRU REGISTER	NAVN	READ		DEKODING
0x20	FWVER	0x0049		Default
0x21	STAT	0xbb86	Bit 15:1	Regulator 2.5V digital error
			Bit 14:0	Regulator 3.3V digital error
			Bit 13:1	Regulator 2.5V analog error
			Bit 12:1	Regulator 3.3V analog error
			Bit 11:1	Small clock instability detected
			Bit 10:0	ADC pattern lock reverted
			Bit 09:1	PLL lock reverted
			Bit 08:1	Clock reverted to BRD
			Bit 07:1	L0 rate generated reset
			Bit 06:0	-
			Bit 05:0	-
			Bit 04:0	-
			Bit 03:0	Clock reset active
			Bit 02:1	PLL clock locked
			Bit 01:1	ADC bitslip locked to sync pattern for all ADCs
			Bit 00:0	40MHz clock line selected (0 for board)
0x22	CLKSEL	0x0111		Default
0x23	ADCDELTA	0x0008		Default
0x24	GTLDELAY	0x0000		Default
0x25 – 0x2a	ADCDEL	0x0000		Default

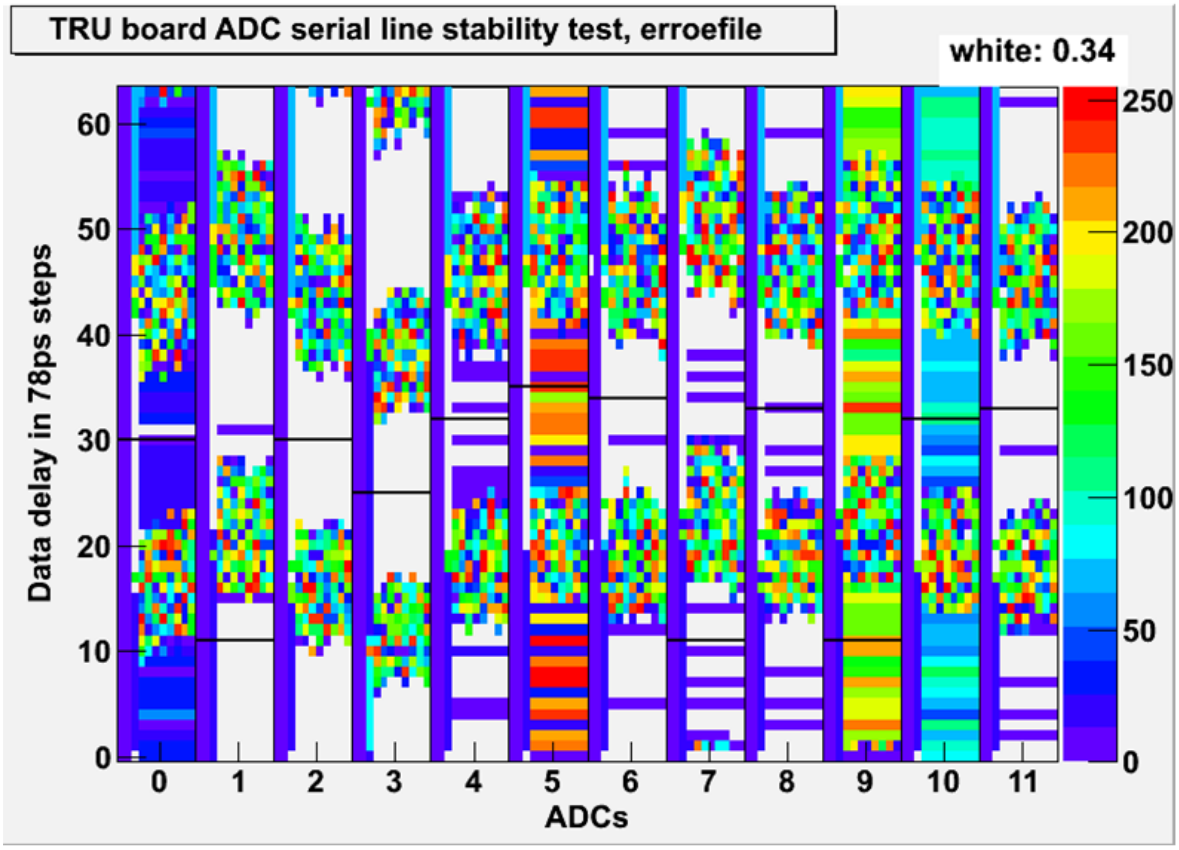
0x2b	PFSEL	0x1e1f		Default
0x2c	L0SEL	0x0101		Default
0x2d	THRSHCOSM	0x3fff		Default
0x2e	THRSHFOUR	0xffff		Default
Ox2f	L0DEL	0x0001		Default
0x30	L0DEAD	0x0000		Default
0x31	GTLDEAD	0x0050		Default
0x34	SYSMON0	0xa95c		Die temp (Value-503.975)/65536 – 236.15 = -235.496
0x35	SYSMON1	0x54a9		VCCINT Value*3/65536 = 0.992
0x36	SYSMON2	0xca2f		VPPAUX Value*3/65536 = 2.369
0x37	SYSMON3	0xa064		VP / VN Value*3/65536 = 1.8796
0x39	EVWID	0x0004		Default
0x3a	RLBKFALTRO	0x002e		Default
0x3b	RLBKSTU	0x002c		Default
0x3c	DATSEL	0x0101	Bit 08: 1	Enable (1) zero level suppression
0x3d	THRSHZSUP	0x0002		Default
0x3f	STUPATT	0x0aaa		Default
0x40	FALTROMEB	0x0000		Disables write/read memory MEB pointer rotation
0x42	TMODE	0x0000	Bit 07: 0	ADC deserializer stability test
			Bit 06: 0	-
			Bit 05: 0	-
			Bit 04: 0	Count arrived L0 confirms
			Bit 03: 0	L0 send to STU
			Bit 02: 0	L0 receive, L0 send to STU
			Bit 01: 0	Fake L0 generation every time interval set by TMDATA (Fast)
			Bit 00: 0	Fake L0 generation every time interval set by TMDATA
0x43	TMDATA	0x0000		Default
0x44	TMDATA2	0x0000		Default
0x45	RSTENABLE	0x0111		Default
0x46	L0RSTLIMIT	0x0000		Default
0x47	L0LIMIT	0xffff		Default
0x48	LORATERSTCNT	0x0000		Default
0x4c	L0RLOW	0x0000		Default
0x4d	LRHIGH	0x0000		Default
0x4e	CONST	0xf5ao		Default
0x4f	RESET	0x0000		Default
0x50	MASK0	0x0000		Mask reg. LSB 0-15 MSB
0x51	MASK1	0x0000		Mask reg. LSB 16-31 MSB
0x52	MASK2	0x0000		Mask reg. LSB 32-47 MSB
0x53	MASK3	0x0000		Mask reg. LSB 48-63 MSB
0x54	MASK4	0x0000		Mask reg. LSB 64-79 MSB
0x55	MASK5	0x0000		Mask reg. LSB 80-95 MSB
0x5a	mask_reg0	0x0000		Mask reg. LSB 0-15 MSB
0x5b	mask_reg1	0x3ff6		Mask reg. LSB 16-31 MSB

0x5c	mask_reg2	0x0000	Mask reg. LSB 32-47 MSB
0x5d	mask_reg3	0xffff	Mask reg. LSB 48-63 MSB
0x5e	mask_reg4	0x0000	Mask reg. LSB 64-79 MSB
0x5f	mask_reg5	0x0001	Mask reg. LSB 80-95 MSB
0xb0	ADCPLOCK	0x0fff	Pattern lock
0xb1	ADCMUX	0x0fff	Multiplexer state
0xb2	ADCDELAY	0x0c22	Data delay implied on the incoming serial data.
0xb3	ADCDELAY	0x1b21	Data delay implied on the incoming serial data.
0xb4	ADCDELAY	0x0b23	Data delay implied on the incoming serial data.
0xb5	ADCDELAY	0x0c0b	Data delay implied on the incoming serial data.
0xb6	ADCDELAY	0x0b0b	Data delay implied on the incoming serial data.
0xb7	ADCDELAY	0x220b	Data delay implied on the incoming serial data.

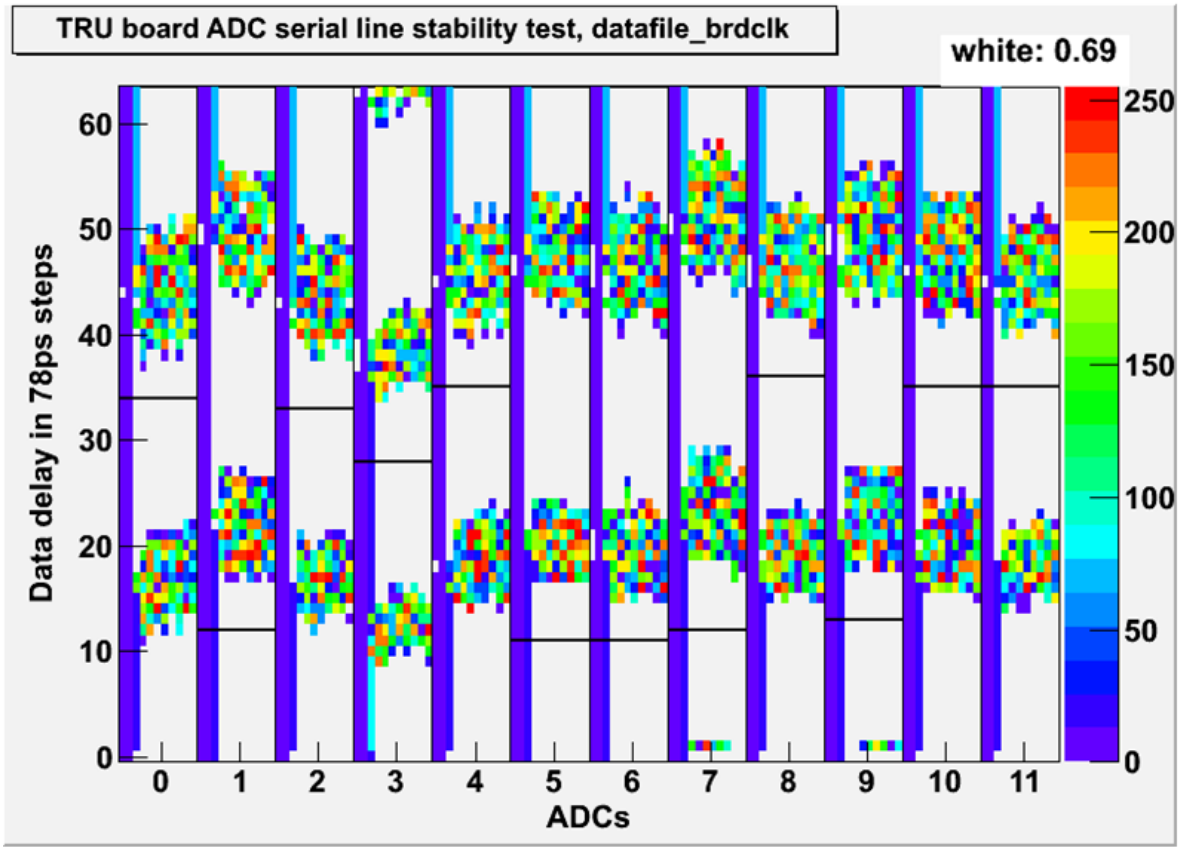
Tabell 6.1 Oversikt over verdiene i TRU-registrene

6.3.2 Kvaliteten på klokken

Etter å ha programmert TRU så ble blant annet status registeret 0x21 lest ut (se register oversikt (22)). Her var bit 8 satt til ’1’, dette vil si at klokken gikk tilbake til å bruke klokken fra brettet. Klokken var i utgangspunkt satt til å bruke klokken fra TOR, men hoppet fort tilbake til klokken på TRU kortet. Kjørte videre skriptet ” TRU_error_scann_channels_double.sh” (se Appendix B) og fikk fram figur 6.4. Det var tydlig at klokken ikke var bra nok. Kjørte med klokken fra TRU kortet etterpå ved å sette bit 00 til ’0’ (se tabell 6.1 register 0x21, bit 00) og kjørte TRU_error_scann_channels_double skriptet igjen og fikk ut plottet på figur 6.5 som var betydelig mye bedre. Det er da klart at klokken fra TOR ikke er bra nok og på en 40 MHz klokke så er det ikke mye forstyrrelser eller forsinkelser til for at det går galt. Klokken på TRU kortet er bra nok for simulering, men når dette skal implementeres i ALICE så kan ikke denne klokken brukes.



Figur 6.4 Feilanalyse av ADC-ene med klokken fra TOR



Figur 6.5 Feilanalyse av ADC-ene med klokken fra TRU-kortet

Figurene 6.4 og 6.5 viser hvordan TRU kjøres i forhold til hvor klokken kommer. Dataen som kommer fra ADC-ene er 480Mbps (Mega bits per sekund), 2ns per bit. Y-aksen viser inn-ut forsinkelsen i steg på datalinjen (1 steg = 78ps) og det er 64 mulige steg fra 0 – 63. Så på y-aksen så ser du hvilken fase dataen er i, i forhold til sampling klokken og hva stabiliteten av dekodingen av ett konstant mønster som ADC-ene er programmert til å sende. Z-aksen (fargeskalaen fra blå til rød helt til høyre) er feilraten. Ut fra figurene ser man da at det er flere ustabile områder i hver ADC separert av de 2 ns når data fasen møter sampling klokkefasen.Mellom disse ustabile områdene så er det stabile regioner som den automatiske instillingsalgoritmen skal posisjonere fasen (de svarte horisontale strekene (ADC 0, 37 steg opp y-aksen er der hvor fasen til ADC 0 blir automatisk stilt inn til)). X-aksen viser de tolv ADC-ene fra 0 til 11 og første kolonne i ADC 0 er binær informasjon, andre kolonne er irrelevant, tredje kolonne viser verdiene til bitskredene (12-bit ordenes grense posisjonert i den serielle data strømmen. De gjenværende 8 kolonnene er feilraten til de 8 kanalene av ADC-en.

6.3 Utlesningsprosedyre

DAQ systemet kalles for DATE (*Data Acquisition and Test Environment).*
Kommunikasjonen mellom RCU og DATE går via en *Detector Data Link* (DDL) som er et vanlig brukergrensesnitt mellom subdetektorene til ALICE og DAQ. DAQ består hovedsakelig av en *Global Data Concentrator* (GDC), en LDC, D-RORC og programvare. LDC samler utfall fragmenter fra DDL inn i hovedminne og sorterer de til del-utfall og så går de videre til GDC for å samle del-utfallene til hele enkelte utfall og lagres.
DATE som det er satt opp i lab-en er basert på en enkel prosessor og gjennomfører alle funksjonene som blant annet LDC og GDC. For å klargjøre LDC til å ta imot data så er det tre ting som må være fullført: DAQ konfigurasjon, Kjøre parameter og Data lagring. Den første forteller hvor mange LDC-er og GDC-er det er i systemet (en av hver i dette tilfelle). Kjøre parameterene så kan det bestemmes maksimum størrelse på utfallene. Data lagring tar seg av konfigurasjon av opptaket.

Pulsgeneratoren simulerer utgangen til en CSP som blir matet inn til analog summering på FEC. TRU og TOR jobber sammen for å generere triggere hver gang det kommer ett signal over threshold. *Level Trigger Unit* (LTU) bruker triggerne som er sent fra TOR som et signal for å eksikvere en L2 sekvens som igjen forteller RCU at det på tide å lese ut Fake ALTRO data fra TRU til D-RORC. Utfallene blir gjenskapt av LDC, GDC og DATE for så å bli lagret i minne på pc-en. Før utlesningen begynner er det viktig at alt er riktig konfigurert. Måten det blir programmert og testet på er :

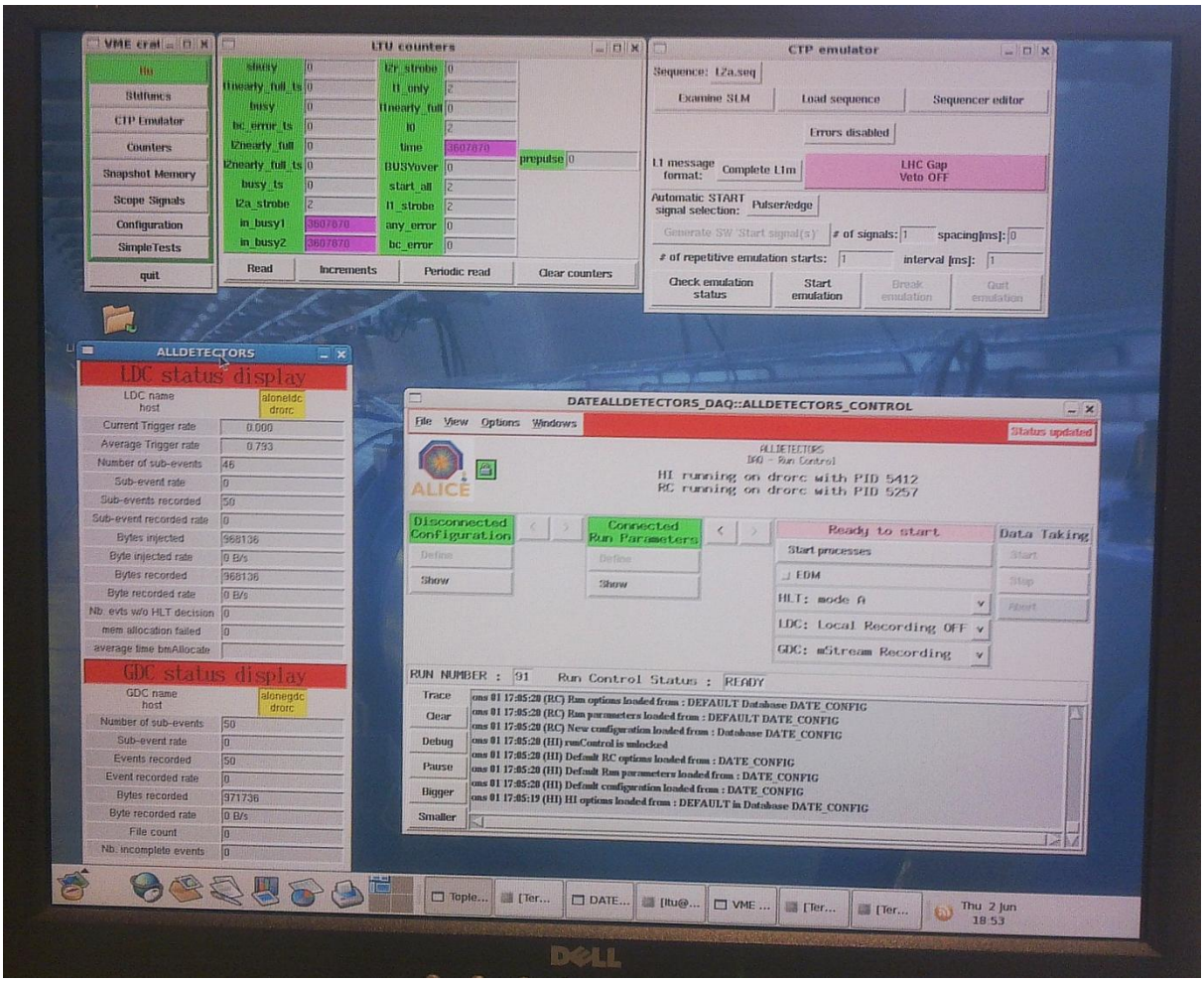
1. **Programmering og initialisering av TOR:** Først så logger man inn på DCS – kortet 0318 til TOR og skriver linjen ” ./program_tor tor_fpga4_080311.bit” , dette er for å programmere firmvaren til FPGA-en til TOR og deretter sjekkes det om FPGA-en er blitt programmert med kommandoen ” rcu-sh r 0x27”. Deretter så initialliseres registrene med skripetet ”set_register.scr” og TRU-en settes så det er kun en inngang fra det ene FEC inn på TRU.

2. **Programmer RCU:** Logger inn på DCS-kortet 0193 til RCU og programmerer firmvaren her ” ./program rcu_171109.bit” og velger kun en inngang på TRU her også, ” rcu-sh w 0x1c 0x10”.
3. **Initialisering av LTU:** Logger inn på *Virtual Machine Environment* vme1 med ”ssh –X ltu@vme1” og eksikverer kommandoen ”vmecreate ltu”, det vil nå komme opp et par bilder (se figur 6.6) på skjermen hvor konfigurasjonen gjøres og man får fram tellerene for triggere og busy-signal.
4. **Åpner CTP emulator og genererer et start signal:** Trykker på ”CTP emulator” på vinduet i forrige punkt og ett nytt vindu vil dukke opp. I dette vinduet velges sekvens ”L2a.seq”. Startsignalet i dette vinduet er en input som informerer om at triggeren skal startes.
5. **Programmering av TRU:** Kjører her kommanden ” rcu-sh b set_ped.scr”, dette konfigurerer TRU med å sette for eksempel ADC-er og pedestal verdiene.
6. **Legger til de kanalene som skal leses ut fra Fake ALTRO:** I EMCal så er skriptet ”set_tru_channel_emcal” laget.

```
w 0x1000 4096 0xfff # clear the readoutlist first
w 0x1000 0x0000 # TRUE channel 0
w 0x1001 0x0001 # TRUE channel 1
w 0x1002 0x0002 # TRUE channel 2
w 0x1003 0x0003 # TRUE channel 3
w 0x1004 0x0004 # TRUE channel 4
w 0x1005 0x0005 # TRUE channel 5
w 0x1006 0x0006 # TRUE channel 6
w 0x1007 0x0007 # TRUE channel 7
w 0x1008 0x0008 # TRUE channel 8
w 0x1009 0x0009 # TRUE channel 9
w 0x100a 0x000a # TRUE channel 10
w 0x100b 0x000b # TRUE channel 11
w 0x100c 0x000c # TRUE channel 12
w 0x100d 0x000d # TRUE channel 13
----
----
----
----
w 0x1061 0x0061 # TRUE channel 97
w 0x1062 0x0062 # TRUE channel 98
w 0x1063 0x0063 # TRUE channel 99
w 0x1064 0x0064 # TRUE channel 100
w 0x1065 0x0065 # TRUE channel 101
w 0x1066 0x0066 # TRUE channel 102
w 0x1067 0x0067 # TRUE channel 103
w 0x1068 0x0068 # TRUE channel 104
w 0x1069 0x0069 # TRUE channel 105
w 0x106a 0x006a # TRUE channel 106
wait 200 us
r 0x5110            # Read FECERRA Error Register
r 0x5111            # Read FECERRB Error Register
w 0x5307 0x0        # Clear error register
```

```
wait 1 us          # Execute "w 0x5306 0x0" for each pedestal trigger
wait 1 us          # or "b trigger_100.scr" for each 100 triggers
```

7. **Testsignaler til FEC:** Det er ingen CSP i lab-en så det blir brukt utgangen på pulsgeneratoren for å simulere inngangen til FEC. En adapter multipler utgangen til pulsgeneratoren og sender disse signalene inn på pinnene CSP_A(9-16) på FEC. Utgangen til pulsgeneratoren er en puls med 20ns stigningstid og frekvens 1 Hz.
8. **Koble fra zero suppression threshold:** Skriver "./TRU_read.sh A 3c" for å lese av zero suppression er på eller av. Forandrer deretter bit 8 til null hvis dette er satt, som regel hvis alt er riktig så leser man ut verdien 0x0101. Gir kommandoen "./TRU_read.sh A 3c 0x0001" for å slå av zero suppression. Dette er for å kunne lese ut "støy" av Fake ALTRO bufferen selv om det ikke har blitt lagret noe reell data.
9. **DATE:** Logger inn på DRORC (drorc.ift.uib.no) og skriver følgende "ssh –X date@localhost" etterfulgt av "startdate" og tre nye vinduer vil åpnes, blant annet vinduet på figur 4.6. I dette vinduet så åpner man først låsen ved å trykke på den, så konfigurerer man DATE ved å trykke på "define" under låsen. Velger "record mode" før man starter innhentingen av data. Valget "Mstreaming record" vil lagre dataen i et lokalt direktiv. Først så trykker man på "Start process", dette starter opp alle de involverte prosessene, men ingen data innhenting enda. Så trykker man på "start" og data innhenting starter, trykker så på "start emulation" på CTP emulator vinduet som igjen starter triggingen. Når du stopper prosessen igjen så er det viktig å stoppe emulatoren først og så data innhentingen.



Figur 6.6 Brukergrensesnittet til D-RORC (2 nederste figurene) og LTU (øverste vinduene)

6.4 Resultater fra utlesning

Ingen L0 trigger ble generert i dette forsøket så alt som ble utlest fra Fake ALTRO bufferen var støy. I første omgang så er det viktig å finne ut om kommunikasjonen mellom RCU og bufferen er der og om det er riktig data som blir lest ut. Ifølge rådataen så er det riktig data som kommer fra TRU og Fake ALTRO, men det er også 32 bit som dukker opp inne imellom som man ikke vet hvor kommer fra.

Som fortalt tidligere så blir hver kanal overført fra Fake ALTRO med 4 40 bit ord. Dette er 16 ti bit ord hvor 2 ti bit ord er header. Under så ses det et utdrag fra utlesningen hvor en ”timebin” er en kanal utlesning med 14 ti bit ord. Alle TRU kanalene blir lest ut (Se appendix E) som ikke kan produseres av andre kort. Hendelseslengden er den samme som for TRU og

adressen er 0x000, gren A som også er riktig. Dette viser at det er Fake ALTRO som blir lest ut.

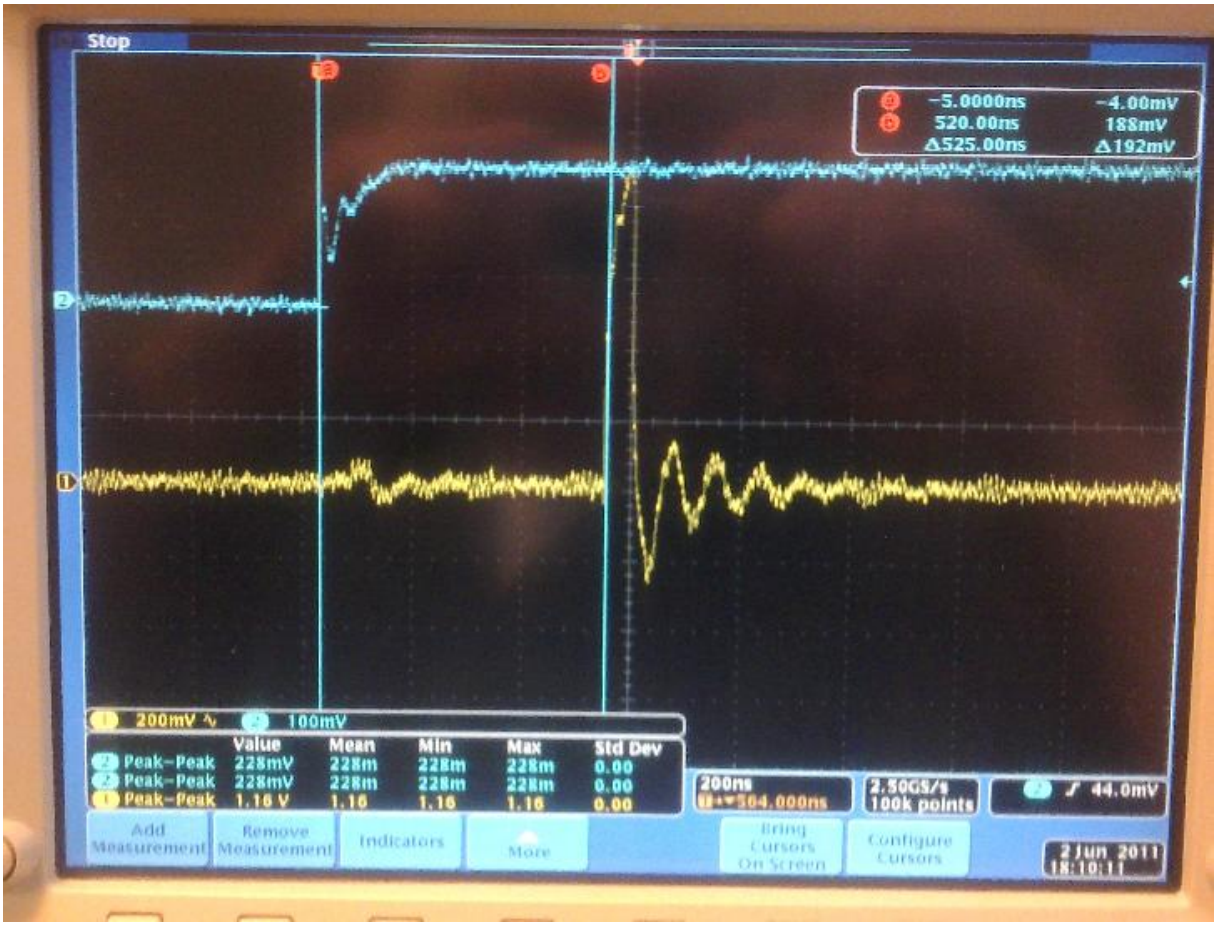
```
enter nextBunchTimebin: 73
The 0 value is 0
The 1 value is 0
The 2 value is 0
The 3 value is 0
The 4 value is 0
The 5 value is 0
The 6 value is 0
The 7 value is 0
The 8 value is 0
The 9 value is 0
The 10 value is 3cc
The 11 value is 84
The 12 value is 82
The 13 value is 80
enter nextBunchTimebin: 74
The 0 value is 0
The 1 value is 0
The 2 value is 0
The 3 value is 0
The 4 value is 0
The 5 value is 0
The 6 value is 0
The 7 value is 0
The 8 value is 0
The 9 value is 0
The 10 value is 3cc
The 11 value is 8c
The 12 value is 82
The 13 value is 0
enter nextBunchTimebin: 75
The 0 value is 0
The 1 value is 0
The 2 value is 0
The 3 value is 0
The 4 value is 0
The 5 value is 0
```

```
The 6 value is 0
The 7 value is 0
The 8 value is 0
The 9 value is 0
The 10 value is 3cc
The 11 value is 84
The 12 value is 82
The 13 value is 80
enter nextBunchTimebin: 76
```

Her blir også skriptet ”jkphs_tru_eventdisplay.cpp” (se Appendix C) til Jiri Kral kjørt på AliRoot for å videre analysere de dataene som har kommet ut fra Fake ALTRO bufferene.

6.4 Resultater hardware

På grunn av mye støy så ble det forsøkt å maskere enkelte av 2x2 signalene fra FEC. Det ble da funnet ut at ved å sette MASK register 0x53 (22) til 0x0008 så ble støyen betydelig redusert. Satt så videre ”Global level-0 threshold” register 0x2e fra 0xffff til0x0050, dette er pidestallverdien som signalet må overgå for at en L0 trigger skal bli generert. Når ”Global level-0 threshold” ble satt under 0x50 så trigget den hele tiden på blant annet støy og når den ble satt for høyt så trigget den ikke i det hele tatt.I kapittel 6.3 så var det VME som genererte L0 signalene sånn at det kunne bli utlest data. Her så er det TRU som genererer en L0 trigger etter 520 ns (se figur 6.7).



Figur 6.7 Tid fra signal går inn på FEC til L0 genereres og går ut fra TRU

6.5 Modifikasjoner

I Appendiks D så ser man en viderutviklet kode av Fake_ALTRO_buffer for å kunne fungere for PHOS detektoren. Det er her forandret fra 960 bit til 1120 som det er i PHOS og kompilert for sjekk av syntaks error. Videre så er disse signalene delt opp i fire og fordelt på samme måte i bufferene som EMCal signalene.

7. Konklusjon

Denne oppgaven viser at det kan se ut til at EMCal TRU kan erstatte PHOS TRU-en i PHOS detektoren, men det er i midlertid visse forbehold. Herunder blant annet at alle subprogrammene til EMCal TRU må modifiseres til å ta imot 112 kanaler istedenfor 96, som her gjort for Fake_ALTRO_Buffer. I tillegg er klokken til TOR som gir en 40 MHz klokke til TRU for dårlig, så her kan det hende at man skal koble til en ekstern klokke dedikert til TRU. Videre for å få lagre data i Fake ALTRO buffer før en utlesning så må man lage en løkke som starter hos TRU og ender opp hos RCU. Her vil det genererte L0 signalet i TRU skape L1 og L2 signaler i RCU. Så setter man pidestallverdien så lav at den trigger på støy (under 0x50) i første omgang for så å konfigurere Fake ALTRO rollback for å motta det korrekte tidsvinduet for hendelsen, dette baserer seg på lengden til løkken. Denne dataen kan da videre sammenlignes med dataen som blir utlest fra ALTRO. Dette ligger utenfor rekkevidden av denne oppgaven med hensyn til de begrensningene som er satt. Det ble satt opp en kommunikasjon mellom PHOS og EMCal TRU og dataene som ble lest så ut til å komme fra TRU i og med at eksakt antall kanaler med tilsynelatende riktig dataformat ble lest ut. Resultatene som er oppnådd i denne innledende undersøkelsen tyder på at det kan være mulig å benytte EMCal TRU i PHOS detektoren.

Referanser

1. **H. Muller, Y. Wang.** *CERN presentation EMCAL level-0 Trigger.* 2009.

2. **Manko, V.** *LHCC ALICE Comprehensive Review VI, March 20-21.* 2006.

3. **Liu, L.** *L0/L1 trigger development of ALICE PHOS detector.* Bergen : UiB, 2011.

4. **H. Muller, Z. Yin.** *PHOS basics for the users.* s.l. : CERN, 2007.

5. **H. Muller, D. Budnikov, M. Ippolitov, Q. Li, V. Manko, R. Pimenta, D. Roehrich, I. Sibiryak, B. Skaali, A. Vinogradov.** Front-end electronics for PWO-based PHOS calorimeter of ALICE. *Nuclear instruments & methods in physics research.* Vol. Section A, 2006, 567.Volume 567, Issue 1, p. 264-267.

6. **Collaboration, ALICE.** *ALICE TPC Readout Chip User Manual.* s.l. : CERN - EP/ED, June 2002.

7. *ALICE Trigger System.* **D. Evans, S. Fedor, I. Kralik, G. Jones, P. Jovanovic, A. Jusko, R. Lietava, L. Sandor, J. Urban, O. Villalobos-Bailie.** s.l. : 10th Workshop on Electronics for LHC and Future Experiments, Boston, MA, USA, 13 - 17 Sep 2004, pp.277-280, .

8. *The ALICE Trigger Electronics.* **M. Krivda, A. Bhasin, D. Evans, G. T. Jones, P. Jovanovic, A. Jusko, I. Kralik, C. Lazzeroni, R. Lietava, H. Scott, L. Sandor, D. T. Takaki, J. Urban, O.Villalobos-Bailie.** s.l. : Topical Workshop on Electronics for Particle Physics, Prague, Czech Republic, 03 - 07 Sep 2007, pp.259-263.

9. *TTC machine interface (TTCmi) User Manual.* **Taylor, B. G.** s.l. : Presented at the 6th Workshop on Electronics for LHC Experiments, Krakw, 11 - 15 September 2000.

10. *TTC laser transmitter TTCex, TTCtx, TTCmx) User Manual.* **Taylor, B. G.** s.l. : Presented at the 6th Workshop on Electronics for LHC Experiments, Krakw, 11 - 15 September 2000.

11. **H. Muller, R. Pimenta, L. Musa, Z. Yin, D. Roehrich, B. Skaali, I. Sibiriak, D. Budnikov.** Trigger electronics for the Alice PHOS detector. *Nuclear instruments & methods in physics research.* Section A, 2004, Vol. Volume 518, Number 1, 1 February 2004 , pp. 525-528(4).

12. **H. Muller, T. C. Awes, N. Novitzky, J. Kral, J. Rak, J. Schambach, Y. Wang, D. Wang, D. Zhou.** Hierarchical trigger of the ALICE calorimeters. *Nuclear instruments and methods in physics research.* Section A, 2010, Vol. VOL 617; NUMBER 1-3, pages 344-347.

13. *L0/L1 triggering with PHOS* . **Roehrich, D.** s.l. : Presentation of Simulation results in ALICE week, April 2003.

14. *Trigger Region Unit for the ALICE PHOS calorimeter.* **H. Muller, A. Oltean, R. Pimenta, D. Roehrich, B. Skaali, X. Cao, Q. Li.** s.l. : 11th Workshop on Electronics for LHC and Future Experiments, Heidelberg, Germany, 12 - 16 Sep 2005, pp.384-387.

15. **Awes, T. C.** The ALICE electromagnetic calorimeter. *Nuclear instruments and methods in physics research.* Section A, Vol. Volume 617, Issues 1-3, 11 May 2010-21 May 2010, Pages 5-8.

16. *Trigger and Timing Control system.* s.l. : http://ttc.web.cern.ch/ttc/.

17. *ALICE Technical Design Report of the Computing.* **ALICE-Collaboration.** s.l. : CERN-LHCC-2005-ALICE TDR 012, 15 June, 2005.

18. *Busy Box user manual.* s.l. : https://wikihost.uib.no/ift/images/0/01/BusyBoxUserGuide.pdf.

19. *EMCal TRU Firmware.* s.l. : https://twiki.cern.ch/twiki/bin/view/ALICE/EMCalTRUFirmwareFiles.

20. *Schematics.* s.l. : https://trac.cc.jyu.fi/projects/alice/wiki/EMCAL/update.

21. *TRU programming guide.* **Kral, Jiri.** s.l. : https://twiki.cern.ch/twiki/pub/ALICE/EMCalTRUProgramming/TRU-Programming.pdf, 02.06.11.

22. *Registers for EMCAL TRU controller.* **Kral, Jiri.** s.l. : https://twiki.cern.ch/twiki/pub/ALICE/EMCalTRURegistriesGuide/TRU-Register-Map_Draft-v74.pdf, 03.06.11.

23. **H. Muller, R. Pimenta, Z. Yin, D. Zhou, X. Cao, Q. Li, Y. Liu, F. Zou, B. Skaali, T. C. Awes.** Configurable electronics with low noise and 14-bit dynamic range for photodiode-

based photon detectors. *Nuclear instruments & methods in physics research.* Section A, 2006, Vol. Volume 565, Issue 2, p. 768-783.

24. **Munkejord, M.** *Development of the alice busy box.* Bergen : UiB, 2007.

Appendiks A Fake ALTRO programkode

A.1 Fake ALTRO

```
-- $Id: fake_altro.vhd 48 2009-03-04 22:26:05Z jkral $

--------------------------------------------------------------------------------
-- Title      : Fake altro
-- Project    :
--------------------------------------------------------------------------------
-- File       : time_sum.vhd
-- Author     :
-- Company    :
-- Created    : 2009-05-13
-- Last update: 2009-05-13
-- Platform   :
-- Standard   : VHDL'93/02
--------------------------------------------------------------------------------
-- Description:
--------------------------------------------------------------------------------
-- Copyright (c) 2009
--------------------------------------------------------------------------------
-- Revisions  :
-- Date        Version  Author  Description
-- 2009-03-04  1.0      jkral       Created, based on dwang
--------------------------------------------------------------------------------

LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
USE ieee.std_logic_unsigned.ALL;
LIBRARY UNISIM;
USE UNISIM.vcomponents.ALL;
USE work.tru_typedef.ALL;

--------------------------------------------------------------------------------

ENTITY fake_altro IS
  PORT (
    rcu_clk                 : IN std_logic;   -- RCU clock
    g_clk                   : IN std_logic;   -- global 40MHz clock
    reset           : IN std_logic;   -- reset

          L1rollback : IN std_logic_vector(7 downto 0); -- L1 rollback for circular copy
          eventwidth : IN std_logic_vector(7 downto 0); -- event size to transmit
          zsup_setup     : IN std_logic; -- zero suppression enable
          zsup_thrsh     : IN std_logic_vector(9 downto 0); -- zero suppression threshold
    meb_onoff  : IN std_logic; -- MEB on/off
```

```
    L1                  : IN std_logic;   -- GTL L1 (L0confirm)
    L2                  : IN std_logic;   -- GTL L2

    shift_data    : IN std_logic_vector(959 downto 0); -- 96 10bit values
    l0id_data     : IN std_logic_vector(109 downto 0); -- L0 ID, L0, masks, LED ID

    cstb                : IN std_logic;   -- GTL command strobe
    write_r             : IN std_logic;   -- GTL write/read

    bd                  : INOUT std_logic_vector(39 DOWNTO 0); -- GTL 40bit word

    ctrl_out      : OUT std_logic;  -- GTL control ??
    oeab_l              : OUT std_logic;  -- GTL ??
    oeab_h              : OUT std_logic;  -- GTL ??
    oeba_l              : OUT std_logic;  -- GTL ??
    oeba_h              : OUT std_logic;  -- GTL ??
    ackn                : OUT std_logic;  -- GTL acknowledge

    dstb                : OUT std_logic;  -- GTL data strobe
    trsf                : OUT std_logic; -- GTL transfer

          gtl_busy      : OUT std_logic   -- traffic on GTL bus (used for trigger mask)
);


END ENTITY fake_altro;

---------------------------------------------------------------------------

ARCHITECTURE str20 OF fake_altro IS

  COMPONENT fake_altro_buffers IS
    PORT (
      g_clk             : IN std_logic; -- 40MHz global clock
      rcu_clk     : IN std_logic; -- 40MHz RCU clock
            reset       : IN std_logic; -- reset

      -- settings
            eventwidth  : IN std_logic_vector(7 downto 0); -- event size to transmit
            L1rollback  : IN std_logic_vector(7 downto 0); -- L1 rollback for circular copy
                  zsup_setup      : IN std_logic; -- zero suppression enable
                  zsup_thrsh      : IN std_logic_vector(9 downto 0); -- zero suppression threshold
            meb_onoff   : IN std_logic; -- MEB on/off

      -- triggers
      L1                  : IN std_logic;   -- GTL L1
      L2                  : IN std_logic;   -- GTL L2

      rdinc               : IN std_logic;   -- read pointer increase signal
```

```
    wrinc               : IN std_logic;   -- write pointer increase signal

    data_in     : IN std_logic_vector(959 downto 0); -- 2x2 data 10 bit
    l0id_data_in        : IN std_logic_vector(109 downto 0); -- L0 ID, L0, masks, LED ID

            gtl_address  : IN std_logic_vector( 6 downto 0 ); -- which channel to readout
    prepare_data        : IN std_logic; -- signal to prepare data for given channel
    is_sparse   : IN std_logic; -- signal that the requested data is sparse mask

            channel_data        : OUT std_logic_vector( 39 downto 0 ); -- data for ALTRO
transfer, 8 frames
            data_ready   : OUT std_logic); -- channel_data ready signal
  END COMPONENT fake_altro_buffers;

  COMPONENT fake_altro_gtl IS
    PORT (
            -- GTL bus input lines
    rclk                        : IN std_logic;
    cstb                        : IN std_logic;
    writeb              : IN std_logic;
    reset                       : IN std_logic;
    bd                          : IN std_logic_vector( 39 downto 0 );

    -- internal signals
            transmit_complete   : IN std_logic;  -- transmit completed signal

            -- GTL bus output lines
    ctrl_out            : OUT std_logic;
    trsf                : OUT std_logic;
    oeab_h              : OUT std_logic;
    oeab_l              : OUT std_logic;
    oeba_h              : OUT std_logic;
    oeba_l              : OUT std_logic;
    ackn                        : OUT std_logic;

            -- control signals
    in_out              : OUT std_logic;   -- data IO direction control
            transmit_ready      : OUT std_logic; -- ready to transmit flag for transmitter
    is_sparse   : OUT std_logic; -- signal that the requested data is sparse mask
            sender_reset : OUT std_logic; -- resets sender before every transmission

    rdinc               : OUT std_logic;   -- read pointer increase signal
    wrinc               : OUT std_logic;   -- write pointer increase signal
            gtl_busy     : OUT std_logic;   -- traffic on GTL bus (used for trigger mask)

            -- address data
            gtl_address  : OUT std_logic_vector( 6 downto 0 ); -- GTL bus address
            gtl_branch   : OUT std_logic);

  END COMPONENT fake_altro_gtl;
```

```
  COMPONENT fake_altro_sender IS
    PORT (
      -- GTL bus input lines
      rclk                : IN std_logic;
      reset               : IN std_logic;

      -- control signals
            data_ready          : IN std_logic; -- data ready flag from buffers
                  zsup_setup          : IN std_logic; -- zero suppression enable
                  zsup_thrsh          : IN std_logic_vector(9 downto 0); -- zero suppression
threshold

      -- data and addresses
            gtl_branch          : IN std_logic; -- brach bit
            gtl_address         : IN std_logic_vector( 6 downto 0 ); -- which channel to readout
            channel_data        : IN std_logic_vector( 39 downto 0 ); -- data for ALTRO
transfer, 8 frames
      is_sparse           : IN std_logic; -- signal that the requested data is sparse mask

      -- control signals
            transmit_complete   : OUT std_logic;  -- transmit completed signal

            -- GTL bus output lines
      bd                  : OUT std_logic_vector( 39 downto 0 );
            dstb          : OUT std_logic);

  END COMPONENT fake_altro_sender;

  SIGNAL bd_in                : std_logic_vector( 39 downto 0 );
  SIGNAL bd_out       : std_logic_vector( 39 downto 0 );
  SIGNAL s_in_out     : std_logic;
  SIGNAL s_gtl_address        : std_logic_vector( 6 downto 0 );
  SIGNAL s_gtl_branch         : std_logic;
  SIGNAL s_channel_data       : std_logic_vector( 39 downto 0 ); -- data for ALTRO transfer, 8
frames
  SIGNAL s_data_ready         : std_logic;
  SIGNAL s_transmit_ready     : std_logic;
  SIGNAL s_transmit_complete        : std_logic;
  SIGNAL s_rdinc      : std_logic;
  SIGNAL s_wrinc      : std_logic;
  SIGNAL s_sender_reset : std_logic;
  SIGNAL s_is_sparse          : std_logic;

BEGIN

  -- buffer opperation
  -- cycles data in RAM and readies them for ALTRO sending according to s_gtl_address
  -- on s_transmit_ready receipt
  --
```

```
fake_altro_buffers_inst : fake_altro_buffers
  PORT MAP(
    g_clk                => g_clk,
    rcu_clk      => rcu_clk,
          reset          => reset,

    -- settings
          eventwidth => eventwidth,
          L1rollback => L1rollback,
                zsup_setup => zsup_setup, -- zero suppression setup
                zsup_thrsh => zsup_thrsh, -- zero suppression threshold
          meb_onoff  => meb_onoff,

                -- triggers
    L1                   => L1,
    L2                   => L2,

    rdinc          => s_rdinc,
    wrinc          => s_wrinc,

    -- raw data
                data_in=> shift_data,
                l0id_data_in    => l0id_data,

          -- signals from GTL control and sender
          gtl_address            => s_gtl_address,
    prepare_data         => s_transmit_ready,
                is_sparse                => s_is_sparse,

          -- signals to transmitter
          channel_data           => s_channel_data,
          data_ready             => s_data_ready);

-- GTL bus control
-- reads the GTL bus and waits for readout command. Issues
-- s_transmit_ready, gtl address, sets gtl lines (apart and dstb)
--
fake_altro_gtl_inst : fake_altro_gtl
  PORT MAP(
    reset                => reset,

          -- GTL bus input lines
    rclk                 => rcu_clk,
    cstb                 => cstb,
    writeb       => write_r,
    bd                   => bd_in,

    -- internal signals
          transmit_complete   => s_transmit_complete,  -- transmit completed signal
```

```
        -- GTL bus output lines
    ctrl_out    => ctrl_out,
    trsf        => trsf,
    oeab_h      => oeab_h,
    oeab_l      => oeab_l,
    oeba_h      => oeba_h,
    oeba_l      => oeba_l,
    ackn                => ackn,

        -- control signals
    in_out                  => s_in_out,   -- data IO direction control
        transmit_ready      => s_transmit_ready, -- ready to transmit flag for transmitter
            is_sparse               => s_is_sparse, -- if is sparse list request
        sender_reset => s_sender_reset, -- resets sender before every transmission

        rdinc => s_rdinc,   -- read pointer increase signal
        wrinc => s_wrinc,   -- write pointer increase signal
            gtl_busy => gtl_busy,

        -- address data
        gtl_address  => s_gtl_address, -- GTL bus address
        gtl_branch   => s_gtl_branch); -- GTL branch

-- data sender
-- formats data to packets and sends them setting dstb
--
fake_altro_sender_inst : fake_altro_sender
  PORT MAP(
    reset               => s_sender_reset,

    -- GTL bus input lines
    rclk                => rcu_clk,

    -- control signals
        data_ready          => s_data_ready, -- data ready flag from buffers
            zsup_setup              => zsup_setup, -- zero suppression setup
            zsup_thrsh              => zsup_thrsh, -- zero suppression threshold
            is_sparse               => s_is_sparse, -- data is smarse channel mask

    -- data and addresses
        gtl_branch          => s_gtl_branch, -- brach bit
        gtl_address         => s_gtl_address, -- which channel to readout
        channel_data        => s_channel_data, -- data for ALTRO transfer, 8 frames

    -- control signals
        transmit_complete  => s_transmit_complete,  -- transmit completed signal

        -- GTL bus output lines
    bd                  => bd_out,
        dstb            => dstb);
```

```
 -- IO buffers for GTL bus data
 bgen: FOR i IN 0 TO 39 GENERATE
   IOBUF_inst: IOBUF
     PORT MAP (
       O => bd_in(i),
            IO   => bd(i),
            I    => bd_out(i),
            T    => s_in_out);
 END GENERATE bgen;

END ARCHITECTURE str20;
```

A.2 Fake_ ALTRO_Buffer

```
-- $Id: fake_al`tro_buffers.vhd 48 2009-03-04 22:26:05Z jkral $
--------------------------------------------------------------------------------
-- Title      : Fake altro
-- Project    :
--------------------------------------------------------------------------------
-- File       : fake_altro_buffers.vhd
-- Author     :
-- Company    :
-- Created    : 2009-05-20
-- Last update: 2009-07-23
-- Platform   :
-- Standard   : VHDL'93/02
--------------------------------------------------------------------------------
-- Description:
--------------------------------------------------------------------------------
-- Copyright (c) 2009
--------------------------------------------------------------------------------
-- Revisions  :
-- Date        Version  Author  Description
-- 2009-03-04  1.0      jkral        Created
--------------------------------------------------------------------------------

LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
USE ieee.std_logic_unsigned.ALL;
USE ieee.numeric_std.ALL;
USE work.tru_typedef.ALL;
```

```
--------------------------------------------------------------------------------

ENTITY fake_altro_buffers IS
  PORT (
    g_clk               : IN std_logic; -- 40MHz global clock
    rcu_clk             : IN std_logic; -- 40MHz RCU clock
          reset         : IN std_logic; -- reset

          eventwidth : IN std_logic_vector(7 downto 0); -- L1 rollback for circular copy
          L1rollback : IN std_logic_vector(7 downto 0); -- L1 rollback for circular copy
          zsup_setup    : IN std_logic; -- zero suppression enable
          zsup_thrsh    : IN std_logic_vector(9 downto 0); -- zero suppression threshold
    meb_onoff  : IN std_logic; -- MEB on/off

    L1                  : IN std_logic;   -- GTL L1
    L2                  : IN std_logic;   -- GTL L2

    rdinc               : IN std_logic;   -- read pointer increase signal
    wrinc               : IN std_logic;   -- write pointer increase signal

    data_in             : IN std_logic_vector(959 downto 0); -- 2x2 data 10 bit
    l0id_data_in        : IN std_logic_vector(109 downto 0); -- L0 ID, L0, masks, LED ID

          gtl_address   : IN std_logic_vector( 6 downto 0 ); -- which channel to readout
    prepare_data        : IN std_logic; -- signal to prepare data for given channel
    is_sparse           : IN std_logic; -- signal that the requested data is sparse mask

          channel_data : OUT std_logic_vector( 39 downto 0 ); -- data for ALTRO transfer, 8
frames
          data_ready    : OUT std_logic); -- channel_data ready signal

END ENTITY fake_altro_buffers;

--------------------------------------------------------------------------------

ARCHITECTURE str21 OF fake_altro_buffers IS

COMPONENT fake_altro_circular IS
  PORT (
          clka:   IN std_logic;
          dina:   IN std_logic_VECTOR(959 downto 0);
          addra: IN std_logic_VECTOR(7 downto 0);
          wea:            IN std_logic_VECTOR(0 downto 0);
          clkb:   IN std_logic;
          addrb: IN std_logic_VECTOR(7 downto 0);
          enb:            IN std_logic;
          doutb: OUT std_logic_VECTOR(959 downto 0));
END COMPONENT;
```

```
COMPONENT fake_altro_meb IS
  PORT (
        clka:   IN std_logic;
        dina:   IN std_logic_VECTOR(319 downto 0);
        addra: IN std_logic_VECTOR(9 downto 0);
        wea:            IN std_logic_VECTOR(0 downto 0);
        clkb:   IN std_logic;
        addrb: IN std_logic_VECTOR(14 downto 0);
        enb:            IN std_logic;
        doutb: OUT std_logic_VECTOR(9 downto 0));
END COMPONENT;

COMPONENT fake_altro_l0id_circular IS
  PORT (
        clka:   IN std_logic;
        dina:   IN std_logic_VECTOR(109 downto 0);
        addra: IN std_logic_VECTOR(7 downto 0);
        wea:            IN std_logic_VECTOR(0 downto 0);
        clkb:   IN std_logic;
        addrb: IN std_logic_VECTOR(7 downto 0);
        enb:            IN std_logic;
        doutb: OUT std_logic_VECTOR(109 downto 0));
END COMPONENT;

COMPONENT fake_altro_l0id_meb IS
  PORT (
        clka:   IN std_logic;
        dina:   IN std_logic_VECTOR(39 downto 0);
        addra: IN std_logic_VECTOR(9 downto 0);
        wea:            IN std_logic_VECTOR(0 downto 0);
        clkb:   IN std_logic;
        addrb: IN std_logic_VECTOR(11 downto 0);
        enb:            IN std_logic;
        doutb: OUT std_logic_VECTOR(9 downto 0));
END COMPONENT;

COMPONENT fake_altro_sparse IS
  PORT (
        clka:   IN std_logic;
        dina:   IN std_logic_VECTOR(106 downto 0);
        addra: IN std_logic_VECTOR(3 downto 0);
        wea:            IN std_logic_VECTOR(0 downto 0);
        clkb:   IN std_logic;
        addrb: IN std_logic_VECTOR(3 downto 0);
        enb:            IN std_logic;
        doutb: OUT std_logic_VECTOR(106 downto 0));
END COMPONENT;

  SIGNAL data_copy : std_logic_vector(959 downto 0); -- 2x2 data 10 bit to copy
  SIGNAL data_copyb          : std_logic_vector(1279 downto 0); -- 2x2 data 10 bit to copy
```

```
into rbuf
  SIGNAL zeroes              : std_logic_vector(79 downto 0); -- to fill the copydata to ram
size
  SIGNAL data_copyb1         : std_logic_vector(959 downto 0); -- 2x2 data transformed 10 bit
to copy
  SIGNAL data_copyb2         : std_logic_vector(959 downto 0); -- 2x2 data transformed 10 bit
to copy
  SIGNAL data_copyb3         : std_logic_vector(959 downto 0); -- 2x2 data transformed 10 bit
to copy
  SIGNAL data_copyb4         : std_logic_vector(959 downto 0); -- 2x2 data transformed 10 bit
to copy
  SIGNAL data_prep  : std_logic_vector(39 downto 0); -- 40 bit prepare
  SIGNAL cbuf_waddr          : std_logic_vector(7 downto 0); -- circular buffer write address
  SIGNAL cbuf_raddr: std_logic_vector(7 downto 0); -- circular buffer read address
  SIGNAL rbuf_waddr          : std_logic_vector(9 downto 0); -- read buffer write address
  SIGNAL rbuf_wbase          : std_logic_vector(9 downto 0); -- event start address
  SIGNAL rbuf_wplus          : std_logic_vector(9 downto 0); -- waddr from event start
  SIGNAL rbuf_raddr : std_logic_vector(59 downto 0); -- 4x 15bit read buffer read address
  SIGNAL rbuf_rbase : std_logic_vector(9 downto 0); -- read buffer event address
  SIGNAL rbuf_rframe         : std_logic_vector(14 downto 0); -- read buffer frame address
  SIGNAL rbuf_rplus1         : std_logic_vector(14 downto 0); -- read buffer 10bit position
  SIGNAL rbuf_rplus2         : std_logic_vector(14 downto 0); -- read buffer 10bit position
  SIGNAL rbuf_rplus3         : std_logic_vector(14 downto 0); -- read buffer 10bit position
  SIGNAL rbuf_rplus4         : std_logic_vector(14 downto 0); -- read buffer 10bit position
  SIGNAL copy_en             : std_logic;
  SIGNAL copy_en_v: std_logic_vector(0 downto 0);
  SIGNAL prep_en             : std_logic;
  SIGNAL L2_del              : std_logic;
  SIGNAL wrinc_del  : std_logic;
  SIGNAL rdinc_del   : std_logic;

  SIGNAL l0id_data_copy      : std_logic_vector(109 downto 0); -- l0 data to copy between
bufs
  SIGNAL l0id_data_copyb   : std_logic_vector(159 downto 0);
  SIGNAL l0id_data_copyb1 : std_logic_vector(109 downto 0);
  SIGNAL l0id_data_copyb2 : std_logic_vector(109 downto 0);
  SIGNAL l0id_data_copyb3 : std_logic_vector(109 downto 0);
  SIGNAL l0id_data_copyb4 : std_logic_vector(109 downto 0);
  SIGNAL l0id_rbuf_raddr     : std_logic_vector(47 downto 0);
  SIGNAL l0id_data_prep      : std_logic_vector(39 downto 0); -- 40 bit prepare

  SIGNAL sparse_data                 : std_logic_vector(106 downto 0);
  SIGNAL sbuf_waddr                  : std_logic_vector(3 downto 0);
  SIGNAL sbuf_raddr          : std_logic_vector(3 downto 0);
  SIGNAL scopy_en_v                  : std_logic_vector(0 downto 0);
  SIGNAL sprep_en                    : std_logic;
  SIGNAL sparse_data_prep            : std_logic_vector(106 downto 0);

  TYPE copystate_type IS ( idle, copystart, copying );
  SIGNAL copystate : copystate_type;
```

```
  TYPE prepstate_type IS ( idle, readstart, sending );
  SIGNAL prepstate : prepstate_type;
  TYPE sparsestate_type IS ( idle, listmake, listdone );
  SIGNAL sparsestate : sparsestate_type;

  SIGNAL ram_id               : std_logic_vector(1 downto 0);
  SIGNAL ram_addr             : std_logic_vector(4 downto 0);
  SIGNAL l0id_ram_addr                : std_logic_vector(4 downto 0);

  -- do not change those unles you check the code, if it works for new values
--  CONSTANT eventwidth   : integer  := 8; -- number 40bit words in one event (samples/4)

  SIGNAL eventwidth4 : std_logic_vector( 9 downto 0);
  SIGNAL meb_onoff_v : std_logic_vector( 9 downto 0);

  SIGNAL counter : std_logic_vector( 7 downto 0 );

BEGIN

  -- compute event width in multiples of 4 samples
  eventwidth4 <= eventwidth & "00";

  -- MEB enable vector
  meb_onoff_v <= (OTHERS=>meb_onoff);

  -- get last two bits of address to decide how the data are ordered in ram
  ram_id <= gtl_address(1 downto 0);
  -- position in selected ram ordering
  ram_addr <= gtl_address( 6 downto 2 );
  -- l0 id address is 96 less, 96/4 = 11000 = 24
  l0id_ram_addr <= gtl_address( 6 downto 2 ) - "11000";

  zeroes <= (OTHERS => '0');

  -- rotate the circular buffer address
  PROCESS( g_clk, reset ) IS
  BEGIN
    IF reset = '1' THEN
            cbuf_waddr <= ( OTHERS => '0' );
          ELSIF rising_edge( g_clk) THEN
            cbuf_waddr <= cbuf_waddr + 1;
    END IF;
  END PROCESS;

  -- transfor data to copy from circular buffer to four separate read buffers.
  -- To distribute channels so, that first channel goes to the first RAM, second
  -- to the second ... fourth to the fourth, fifth to the first.
  ctr: FOR i IN 0 TO 23 GENERATE
    ctrb1: FOR j IN 0 TO 3 GENERATE
      data_copyb1( (i+1)*10-1+j*240 downto i*10+j*240 ) <=
```

```
                                    data_copy( ((i*4)+j+1)*10-1 downto ((i*4)+j)*10 );
        END GENERATE ctrb1;
  ctrb2: FOR j IN 0 TO 2 GENERATE
    data_copyb2( (i+1)*10-1+(j+1)*240 downto i*10+(j+1)*240 ) <=
                                    data_copy( ((i*4)+j+1)*10-1 downto ((i*4)+j)*10 );
    data_copyb4( (i+1)*10-1+j*240 downto i*10+j*240 ) <=
                                    data_copy( ((i*4)+j+1+1)*10-1 downto ((i*4)+j+1)*10 );
        END GENERATE ctrb2;
  ctrb3: FOR j IN 0 TO 1 GENERATE
    data_copyb3( (i+1)*10-1+(j+2)*240 downto i*10+(j+2)*240 ) <=
                                    data_copy( ((i*4)+j+1)*10-1 downto ((i*4)+j)*10 );
    data_copyb3( (i+1)*10-1+j*240 downto i*10+j*240 ) <=
                                    data_copy( ((i*4)+j+2+1)*10-1 downto ((i*4)+j+2)*10 );
        END GENERATE ctrb3;

  data_copyb2( (i+1)*10-1 downto i*10 ) <=
                                    data_copy( ((i*4)+3+1)*10-1 downto ((i*4)+3)*10 );
  data_copyb4( (i+1)*10-1+3*240 downto i*10+3*240 ) <=
                                    data_copy( ((i*4)+1)*10-1 downto ((i*4))*10 );
END GENERATE ctr;

-- the same for L0 IDs
l0id_data_copyb1 <= l0id_data_copy(79 downto 70) & l0id_data_copy(39 downto 30)
                                                 & l0id_data_copy(109 downto 100)
              & l0id_data_copy(69 downto 60) & l0id_data_copy(29 downto 20)
                                                 & l0id_data_copy(99 downto 90)
              & l0id_data_copy(59 downto 50) & l0id_data_copy(19 downto 10)
                                                 & l0id_data_copy(89 downto 80)
              & l0id_data_copy(49 downto 40) & l0id_data_copy(9 downto 0);
l0id_data_copyb2 <= l0id_data_copy(109 downto 100)
              & l0id_data_copy(69 downto 60) & l0id_data_copy(29 downto 20)
                                                 & l0id_data_copy(99 downto 90)
              & l0id_data_copy(59 downto 50) & l0id_data_copy(19 downto 10)
                                                 & l0id_data_copy(89 downto 80)
              & l0id_data_copy(49 downto 40) & l0id_data_copy(9 downto 0)
              & l0id_data_copy(79 downto 70) & l0id_data_copy(39 downto 30);
l0id_data_copyb3 <= l0id_data_copy(99 downto 90)
              & l0id_data_copy(59 downto 50) & l0id_data_copy(19 downto 10)
                                                 & l0id_data_copy(89 downto 80)
              & l0id_data_copy(49 downto 40) & l0id_data_copy(9 downto 0)
              & l0id_data_copy(79 downto 70) & l0id_data_copy(39 downto 30)
                                                 & l0id_data_copy(109 downto 100)
              & l0id_data_copy(69 downto 60) & l0id_data_copy(29 downto 20);
l0id_data_copyb4 <= l0id_data_copy(89 downto 80)
              & l0id_data_copy(49 downto 40) & l0id_data_copy(9 downto 0)
              & l0id_data_copy(79 downto 70) & l0id_data_copy(39 downto 30)
                                                 & l0id_data_copy(109 downto 100)
              & l0id_data_copy(69 downto 60) & l0id_data_copy(29 downto 20)
                                                 & l0id_data_copy(99 downto 90)
                                                 & l0id_data_copy(59 downto 50) &
```

```
l0id_data_copy(19 downto 10);

  -- copy data from circular to readout buffers
  -- after L1 received, 8 samples is coppied
  PROCESS( g_clk, reset ) IS
  BEGIN
    IF reset = '1' THEN
                  copy_en <= '0';
                  copy_en_v <= "0";
                  rbuf_waddr <= "0000000000"; -- to start writing to 0
                  rbuf_wbase <= "0000000000"; -- to start writing to 0
                  rbuf_wplus <= "0000000000"; -- to start writing to 0
                  cbuf_raddr <= x"00";
                  sbuf_waddr <= (OTHERS => '0');
                  L2_del <= '1'; -- no trigger
                  wrinc_del <= '0'; -- no command
           ELSIF rising_edge( g_clk ) THEN
                  -- shift the base address on L2 (or write increase command)
                  -- L2 lasts at 2 cycles at least, so one wants to be sure to shift the registry
                  -- once only. Old-new value comparison, trigger on new L2 only
                  --
                  IF (L2 = '0' AND L2_del = '1') OR (wrinc = '1' AND wrinc_del = '0') THEN
                    rbuf_wbase <= rbuf_wbase + ( eventwidth4 AND meb_onoff_v );
                    sbuf_waddr <= sbuf_waddr + ("000" & meb_onoff);
                  END IF;
      L2_del <= L2; -- save current L2 value
                  wrinc_del <= wrinc;

             CASE copystate IS
                  WHEN idle =>
                    -- no need to do the same trick for L1, since whole machinery starts on L1
start
                    --
                    copy_en_v <= "0";  -- disable copy write procedure
                    IF L1 = '0' THEN
                      copy_en <= '1'; -- enable copy read procedure

          -- roll back in circular buffer defined ammount of samples
                      cbuf_raddr <= cbuf_waddr - L1rollback;

                           copystate <= copystart;
        ELSE
                           copystate <= idle;
                  END IF;
      WHEN copystart =>
                  cbuf_raddr <= cbuf_raddr + 1; -- rotate the circular read address
                  rbuf_wplus <= "0000000000"; -- zero the in-event address

                  copystate <= copying;
                 WHEN copying =>
```

```
            -- chose transformation type for write data
            CASE rbuf_wplus(1 downto 0) IS
            WHEN "00" =>
      data_copyb <= zeroes & data_copyb1(959 downto 720)
                           & zeroes & data_copyb1(719 downto 480)
                           & zeroes & data_copyb1(479 downto 240)
                           & zeroes & data_copyb1(239 downto 0);
      l0id_data_copyb <= zeroes(19 downto 0) & l0id_data_copyb1(109 downto 90)
                           & zeroes(9 downto 0) & l0id_data_copyb1(89 downto 60)
                           & zeroes(9 downto 0) & l0id_data_copyb1(59 downto 30)
                           & zeroes(9 downto 0) & l0id_data_copyb1(29 downto 0);

            WHEN "01" =>
      data_copyb <= zeroes & data_copyb2(959 downto 720)
                           & zeroes & data_copyb2(719 downto 480)
                           & zeroes & data_copyb2(479 downto 240)
                           & zeroes & data_copyb2(239 downto 0);
      l0id_data_copyb <= zeroes(9 downto 0) & l0id_data_copyb2(109 downto 80)
                           & zeroes(9 downto 0) & l0id_data_copyb2(79 downto 50)
                           & zeroes(9 downto 0) & l0id_data_copyb2(49 downto 20)
                           & zeroes(19 downto 0) & l0id_data_copyb2(19 downto 0);



WHEN "10" =>
      data_copyb <= zeroes & data_copyb3(959 downto 720)
                           & zeroes & data_copyb3(719 downto 480)
                           & zeroes & data_copyb3(479 downto 240)
                           & zeroes & data_copyb3(239 downto 0);
      l0id_data_copyb <= zeroes(9 downto 0) & l0id_data_copyb3(109 downto 80)
                           & zeroes(9 downto 0) & l0id_data_copyb3(79 downto 50)
                           & zeroes(19 downto 0) & l0id_data_copyb3(49 downto 30)
                           & zeroes(9 downto 0) & l0id_data_copyb3(29 downto 0);

            WHEN "11" =>
      data_copyb <= zeroes & data_copyb4(959 downto 720)
                           & zeroes & data_copyb4(719 downto 480)
                           & zeroes & data_copyb4(479 downto 240)
                           & zeroes & data_copyb4(239 downto 0);
      l0id_data_copyb <= zeroes(9 downto 0) & l0id_data_copyb4(109 downto 80)
                           & zeroes(19 downto 0) & l0id_data_copyb4(79 downto 60)
                           & zeroes(9 downto 0) & l0id_data_copyb4(59 downto 30)
                           & zeroes(9 downto 0) & l0id_data_copyb4(29 downto 0);

            WHEN OTHERS =>
      data_copyb <= zeroes & data_copyb1(959 downto 720)
                           & zeroes & data_copyb1(719 downto 480)
                           & zeroes & data_copyb1(479 downto 240)
                           & zeroes & data_copyb1(239 downto 0);
```

```
      l0id_data_copyb <= zeroes(9 downto 0) & l0id_data_copyb2(109 downto 80)
                                 & zeroes(9 downto 0) & l0id_data_copyb2(79 downto 50)
                                & zeroes(9 downto 0) & l0id_data_copyb2(49 downto 20)
                                & zeroes(19 downto 0) & l0id_data_copyb2(19 downto 0);

    END CASE;

                IF rbuf_wplus = eventwidth4-2 THEN
                  copy_en <= '0'; -- disable read two cycles prior to write
                ELSIF rbuf_wplus = eventwidth4-1 THEN
                  copystate <= idle;
                ELSE
                  cbuf_raddr <= cbuf_raddr + 1; -- rotate the circular read address

                END IF;

                copy_en_v <= "1"; -- enable copy write one cycle after copy read

                rbuf_waddr <= rbuf_wbase + rbuf_wplus; -- rotate write buffer
                rbuf_wplus <= rbuf_wplus + 1;
              END CASE;
        END IF;
END PROCESS;

-- sparse readout channel list build during circular to meb copy
PROCESS( g_clk, reset ) IS
BEGIN
  IF reset = '1' THEN
          sparse_data <= (OTHERS => (NOT zsup_setup));
                sparsestate <= idle;
          scopy_en_v <= "0";

        ELSIF rising_edge( g_clk ) THEN
                CASE sparsestate IS
                WHEN idle => -- wait for circular copy start
                  -- mark all channels as full of data, when no zsupp
                  -- or mark as empty, when zsupp on
                  sparse_data <= (OTHERS => (NOT zsup_setup));

    -- memory save disable
                  scopy_en_v <= "0";

                  IF copy_en = '1' THEN
                    sparsestate <= listmake;
                  ELSE
                    sparsestate <= idle;
                  END IF;

                WHEN listmake => -- data from circular valid, make comparisons
                  -- keep previous state if data <= zsup threshold
```

```
                    -- mark as non-empty, if over threshold
                    FOR i in 0 TO 95 LOOP -- data channels
                      IF data_copy(i*10+9 downto i*10) > zsup_thrsh THEN
                        sparse_data(i) <= '1';
                      END IF;
                    END LOOP;
                    FOR i in 0 TO 10 LOOP -- l0id channels
                      IF l0id_data_copy(i*10+9 downto i*10) /= "0000000000" THEN
                        sparse_data(i+96) <= '1';
                      END IF;
                    END LOOP;

                    IF copy_en = '0' THEN
                      sparsestate <= listdone;
                    ELSE
                      sparsestate <= listmake;
                    END IF;

                  WHEN listdone =>
                    -- copy the mask to memory
                    scopy_en_v <= "1";
                    sparsestate <= idle;

      END CASE;
          END IF;
  END PROCESS;

  -- define the read address composition
  -- the sub variables are modified by the read cycle
  -- the formula remains the same
  rbuf_raddr <= ( rbuf_rframe + rbuf_rplus4 + ( "0000000000" & ram_addr ))
                  & ( rbuf_rframe + rbuf_rplus3 + ( "0000000000" & ram_addr ))
                                  & ( rbuf_rframe + rbuf_rplus2 + ( "0000000000" & ram_addr
))
                                & ( rbuf_rframe + rbuf_rplus1 + ( "0000000000" & ram_addr ));

  -- compose the L0 buffer read address
  -- since write addressing is exactly similar to channel data, just the write
  -- width is 8 times less, the read addressing is also +-the same, except of being
  -- 8 times divided, ram addr is subtracted by 96 to have 0 on 0 l0id channel
  l0id_rbuf_raddr <= ( rbuf_rframe(14 downto 3) + rbuf_rplus4(14 downto 3)
                + ( "0000000" & l0id_ram_addr ))
                  & ( rbuf_rframe(14 downto 3) + rbuf_rplus3(14 downto 3)
                                  + ( "0000000" & l0id_ram_addr ))
                                & ( rbuf_rframe(14 downto 3) + rbuf_rplus2(14 downto 3)
                                + ( "0000000" & l0id_ram_addr ))
                              & ( rbuf_rframe(14 downto 3) + rbuf_rplus1(14 downto 3)
                                  + ( "0000000" & l0id_ram_addr ));

  -- receive prepare data signal and provide the data to sender
```

```
 -- 40 bits on each clock. Assert data_ready one clock cycle before data
 -- and de-asserd with asserting of last data
 PROCESS( rcu_clk, reset ) IS
 BEGIN
  IF reset = '1' THEN
          prepstate <= idle;
    rbuf_rbase <= (OTHERS => '0');
    rbuf_rframe <= (OTHERS => '0');
    rbuf_rplus1 <= (OTHERS => '0');
    rbuf_rplus2 <= (OTHERS => '0');
    rbuf_rplus3 <= (OTHERS => '0');
    rbuf_rplus4 <= (OTHERS => '0');
    sbuf_raddr <= (OTHERS => '0');
    rdinc_del <= '0';
    channel_data <= (OTHERS => '0');
    data_ready <= '0';
    prep_en <= '0';
    sprep_en <= '0';
    counter <= (OTHERS => '0');
        ELSIF rising_edge( rcu_clk ) THEN
          -- increase event read pointer by one event on RCU read inc command
                IF rdinc = '1' AND rdinc_del = '0' THEN
                  rbuf_rbase <= rbuf_rbase + ( eventwidth4 AND meb_onoff_v );
                  sbuf_raddr <= sbuf_raddr + ( "000" & meb_onoff );
                END IF;

    -- keep current value
    rdinc_del <= rdinc;

          CASE prepstate IS
                WHEN idle =>
                  -- reset all
                  data_ready <= '0';
                        prep_en <= '0';
                        sprep_en <= '0';
                        counter <= (OTHERS => '0');

                  IF prepare_data = '1' THEN
                          prep_en <= '1' AND (NOT is_sparse); -- enable memory read for data
                                sprep_en <= '1' AND is_sparse; -- enable memory read for
sparse request

        -- set the base addres according to event read pointer position
                          rbuf_rframe <= rbuf_rbase & "00000";

        -- set 320bit shifts according to which channel is being read out
                          -- write width 320bits, read width 10 bits -> one shift is 32 = 100000
                          CASE ram_id IS
                          WHEN "00" =>
                            rbuf_rplus1 <= "000000000000000"; -- no shift     32*0
```

```
                    rbuf_rplus2 <= "000000000100000"; -- one shift    32*1
                    rbuf_rplus3 <= "000000001000000"; -- two shifts   32*2
                    rbuf_rplus4 <= "000000001100000"; -- three shifts 32*3
                  WHEN "01" =>
                    rbuf_rplus1 <= "000000001100000"; -- three shifts
                    rbuf_rplus2 <= "000000000000000"; -- no shift
                    rbuf_rplus3 <= "000000000100000"; -- one shift
                    rbuf_rplus4 <= "000000001000000"; -- two shifts
                  WHEN "10" =>
                    rbuf_rplus1 <= "000000001000000"; -- two shifts
                    rbuf_rplus2 <= "000000001100000"; -- three shifts
                    rbuf_rplus3 <= "000000000000000"; -- no shift
                    rbuf_rplus4 <= "000000000100000"; -- one shift
                  WHEN "11" =>
                    rbuf_rplus1 <= "000000000100000"; -- one shift
                    rbuf_rplus2 <= "000000001000000"; -- two shifts
                    rbuf_rplus3 <= "000000001100000"; -- three shifts
                    rbuf_rplus4 <= "000000000000000"; -- no shift
                  WHEN OTHERS =>
                    rbuf_rplus1 <= "000000000000000"; -- no shift     32*0
                    rbuf_rplus2 <= "000000000100000"; -- one shift    32*1
                    rbuf_rplus3 <= "000000001000000"; -- two shifts   32*2
                    rbuf_rplus4 <= "000000001100000"; -- three shifts 32*3
 END CASE;

                    prepstate <= readstart;
                  END IF;

          WHEN readstart =>
            data_ready <= '1'; -- signal data ready (one cycle prior to data)
                  sprep_en <= '0'; -- disable sparse mask read

IF counter = eventwidth-1 THEN
                    prep_en <= '0'; -- disable memory readout
                  ELSE
  -- rotate the read buffer
              rbuf_rframe <= rbuf_rframe + "000000010000000"; -- 32*4=128
                  END IF;

counter <= counter + 1;
                  prepstate <= sending;

          WHEN sending =>
            IF is_sparse = '1' THEN -- sparse channel mask request
                  IF counter = 4 THEN
              data_ready <= '0'; -- signal data ready done (with the last data)
                    prepstate <= idle;
                  END IF;
                  counter <= counter + 1;
            ELSIF counter < eventwidth-1 THEN
```

```
                        rbuf_rframe <= rbuf_rframe + "000000010000000"; -- 32*4=128
                        counter <= counter + 1;
                    ELSIF counter = eventwidth-1 THEN
                        prep_en <= '0'; -- disable ram readout
                        counter <= counter + 1;
        ELSE
                     data_ready <= '0'; -- signal data ready done (with the last data)
                            prepstate <= idle;
                        END IF;

        IF is_sparse = '1' THEN -- sparse channel mask
                        IF counter = 1 THEN -- first block
                              channel_data <= x"00" & sparse_data_prep(31 downto 0);
                            ELSIF counter = 2 THEN -- second block
                              channel_data <= x"00" & sparse_data_prep(63 downto 32);
                            ELSIF counter = 3 THEN -- third block
                              channel_data <= x"00" & sparse_data_prep(95 downto 64);
                            ELSE -- fourth block
                              channel_data <= x"0000000" & '0' & sparse_data_prep(106
downto 96);
                            END IF;
        ELSIF ram_addr < "11000" THEN -- channel data
                -- assign data in correct order
                        CASE ram_id IS
                        WHEN "00" =>
                          channel_data <= data_prep; -- no shift
                        WHEN "01" =>
                          channel_data <= data_prep(9 downto 0) & data_prep(39 downto 10);
-- one shift
                        WHEN "10" =>
                          channel_data <= data_prep(19 downto 0) & data_prep(39 downto
20); -- two shifts
                        WHEN "11" =>
                          channel_data <= data_prep(29 downto 0) & data_prep(39 downto
30); -- three shifts
                        WHEN OTHERS =>
                          channel_data <= data_prep; -- no shift
         END CASE;

                       ELSE -- L0 id data
                        CASE ram_id IS
                        WHEN "00" =>
                          channel_data <= l0id_data_prep; -- no shift
                        WHEN "01" =>
                          channel_data <= l0id_data_prep(9 downto 0) & l0id_data_prep(39
downto 10); -- one shift
                        WHEN "10" =>
                          channel_data <= l0id_data_prep(19 downto 0) & l0id_data_prep(39
downto 20); -- two shifts
                        WHEN "11" =>
```

```
                        channel_data <= l0id_data_prep(29 downto 0) & l0id_data_prep(39
downto 30); -- three shifts
                      WHEN OTHERS =>
                        channel_data <= l0id_data_prep; -- no shift
        END CASE;
                  END IF;

                END CASE;
  END IF;
 END PROCESS;


 -- circular RAM buffer
 fake_altro_circular_inst : fake_altro_circular
  PORT MAP (
          clka          => g_clk,
          dina          => data_in,
          addra                 => cbuf_waddr,
          wea           => "1",
          clkb          => g_clk,
          addrb                 => cbuf_raddr,
          enb           => copy_en,
          doutb                 => data_copy);

 memgen: FOR i IN 0 TO 3 GENERATE
  -- read buffer
  fake_altro_meb_inst : fake_altro_meb
    PORT MAP (
          clka          => g_clk,
          dina                  => data_copyb((i+1)*320-1 downto i*320),
          addra                 => rbuf_waddr,
          wea                   => copy_en_v,
          clkb                  => rcu_clk,
          addrb                 => rbuf_raddr((i+1)*15-1 downto i*15),
          enb                   => prep_en,
          doutb                 => data_prep((i+1)*10-1 downto i*10));
 END GENERATE memgen;

 -- L0ID circular RAM buffer
 fake_altro_l0id_circular_inst : fake_altro_l0id_circular
  PORT MAP (
          clka          => g_clk,
          dina          => l0id_data_in,
          addra                 => cbuf_waddr,
          wea           => "1",
          clkb          => g_clk,
          addrb                 => cbuf_raddr,
          enb           => copy_en,
          doutb                 => l0id_data_copy);
```

```
 memgenl0id: FOR i IN 0 TO 3 GENERATE
  fake_altro_l0id_meb_inst : fake_altro_l0id_meb
    PORT MAP (
      clka                => g_clk,
            dina          => l0id_data_copyb((i+1)*40-1 downto i*40),
      addra       => rbuf_waddr,
            wea           => copy_en_v,
            clkb          => rcu_clk,
            addrb         => l0id_rbuf_raddr((i+1)*12-1 downto i*12),
            enb           => prep_en,
            doutb         => l0id_data_prep((i+1)*10-1 downto i*10));
 END GENERATE memgenl0id;

 fake_altro_sparse_inst : fake_altro_sparse
 PORT MAP (
   clka           => g_clk,
   dina           => sparse_data,
   addra          => sbuf_waddr,
          wea             => scopy_en_v,
          clkb            => rcu_clk,
          addrb  => sbuf_raddr,
          enb             => sprep_en,
          doutb  => sparse_data_prep);

END ARCHITECTURE str21;
```

A.3 Fake_ALTRO_GTL

```
-- $Id: fake_altro_gtl.vhd 48 2009-03-04 22:26:05Z jkral $
-------------------------------------------------------------------------------
-- Title      : Fake altro
-- Project    :
-------------------------------------------------------------------------------
-- File       : fake_altro_gtl.vhd
-- Author     :
-- Company    :
-- Created    : 2009-05-25
-- Last update: 2009-05-25
-- Platform   :
-- Standard   : VHDL'93/02
-------------------------------------------------------------------------------
-- Description:
-------------------------------------------------------------------------------
-- Copyright (c) 2009
-------------------------------------------------------------------------------
```

```
-- Revisions  :
-- Date        Version  Author  Description
-- 2009-03-04  1.0      jkral       Created, based on dwang
------------------------------------------------------------------------------

LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
USE ieee.std_logic_unsigned.ALL;
USE ieee.numeric_std.ALL;
USE work.tru_typedef.ALL;

------------------------------------------------------------------------------

ENTITY fake_altro_gtl IS
  PORT (
    reset           : IN std_logic;

    -- GTL bus input lines
    rclk                    : IN std_logic;
    cstb                    : IN std_logic;
    writeb                  : IN std_logic;
    bd                      : IN std_logic_vector( 39 downto 0 );

    -- internal signals
    transmit_complete  : IN std_logic;  -- transmit completed signal

          -- GTL bus output lines
    ctrl_out       : OUT std_logic;
    trsf                    : OUT std_logic;
    oeab_h                  : OUT std_logic;
    oeab_l                  : OUT std_logic;
    oeba_h                  : OUT std_logic;
    oeba_l                  : OUT std_logic;
    ackn                    : OUT std_logic;

          -- control signals
    in_out                  : OUT std_logic;   -- data IO direction control
          transmit_ready          : OUT std_logic; -- ready to transmit flag for transmitter
    is_sparse     : OUT std_logic; -- signal that the requested data is sparse mask
          sender_reset   : OUT std_logic; -- resets sender before every transmission

    rdinc                   : OUT std_logic;   -- read pointer increase signal
    wrinc                   : OUT std_logic;   -- write pointer increase signal
          gtl_busy        : OUT std_logic;   -- traffic on GTL bus (used for trigger mask)

          -- address data
          gtl_address     : OUT std_logic_vector( 6 downto 0 ); -- GTL bus address
          gtl_branch              : OUT std_logic); -- GTL branch

END ENTITY fake_altro_gtl;
```

```
--------------------------------------------------------------------------------

ARCHITECTURE str22 OF fake_altro_gtl IS

  SIGNAL state                  : std_logic_vector(4 downto 0);
  CONSTANT idle             : std_logic_vector(4 downto 0) := "00000";
  CONSTANT release_ackn         : std_logic_vector(4 downto 0) := "00001";
  CONSTANT wait_out         : std_logic_vector(4 downto 0) := "00010";
  CONSTANT ctrls_out        : std_logic_vector(4 downto 0) := "00100";
  CONSTANT data_out         : std_logic_vector(4 downto 0) := "01000";
  CONSTANT stop             : std_logic_vector(4 downto 0) := "10000";

  SIGNAL is_data_read : std_logic;
  SIGNAL s_ackn : std_logic;

BEGIN

        PROCESS ( reset, rclk ) IS
        BEGIN
          IF reset = '1' THEN
            gtl_address <= (OTHERS => '0');
                gtl_branch <= '0';
                state <= idle;
                is_data_read <= '0';
                s_ackn <= '1';
                is_sparse <= '0';
                gtl_busy <= '0';

          ELSIF rising_edge( rclk ) THEN

      -- remove read pointer increase signal, if asserted
      rdinc <= '0';

                  CASE state IS
                  WHEN idle =>

                    -- write pointer increase remove, must be longer, since goes to gclk domain
                    wrinc <= '0';
                        gtl_busy <= '0';

                        IF cstb = '0' THEN  -- cstb asserted (to 0) = command arriving
                          -- remove the sparse flag
                                is_sparse <= '0';

                          -- check the broadcast, FEC address, write and instruction
                          -- channel readout command
                          IF (( bd(38 downto 37) & bd(35 downto 32) & writeb & bd(24
downto 20) ) = x"01a") THEN
                            is_data_read <= '1';
```

```
                    s_ackn <= '0'; -- set acknowledge
                    gtl_address <=  bd(31 downto 25);
                gtl_branch <= bd(36);
                    gtl_busy <= '1';
                state <= release_ackn;

                    -- broadcast or direct read
                ELSIF (( bd(38 downto 37) & writeb ) = "100" OR
            ( bd(38 downto 37) & writeb & bd(35 downto 32) ) = "0000000") THEN

                    -- 19 - read pointer increase command
        IF bd(24 downto 20) = "11001" THEN
                is_data_read <= '0'; -- no data read
                      rdinc <= '1';
                    -- 18 - write pointer increase command
                    ELSIF bd(24 downto 20) = "11000" THEN
                is_data_read <= '0'; -- no data read
                      wrinc <= '1';
                    -- 29 - sparse channel read
                ELSIF bd(24 downto 20) = "11101" THEN
                      is_data_read <= '1';
                is_sparse <= '1';
                          gtl_busy <= '1';
                    END IF;

                -- acknowledge on non-broadcast only
                IF bd(38) = '0' THEN
                      s_ackn <= '0';
                END IF;

                gtl_address <=  bd(31 downto 25);
                gtl_branch <= bd(36);
                state <= release_ackn;
                    ELSE
                      -- read commands monitored just for gtl_busy signal
                      -- sparse channel request or read command that belong to
someone else
                      IF bd(24 downto 20) = "11101" OR bd(24 downto 20) =
"11010" THEN
                        gtl_busy <= '1';
                      END IF;

                      state <= idle;
                  END IF;
                ELSE
            is_data_read <= '0';
                      state <= idle;
                END IF;

            WHEN release_ackn => -- wait for cstb release and release ack
```

```
                    IF cstb ='1' THEN
                            s_ackn <= '1';
                            IF is_data_read = '1' THEN -- go to data transmit if read command
                              state <= wait_out;
                     ELSE
                              state <= idle;  -- nothing to do, idle
                                  END IF;
                           ELSE
                            state <= release_ackn; -- wait
                           END IF;

                    WHEN wait_out => -- wait
                            state <= ctrls_out;

                    WHEN ctrls_out => -- ctrl assert and transmit ready assert and all other
friends
                            state <= data_out;

                    WHEN data_out => -- wait for data transmit end
                            IF transmit_complete = '1' OR cstb = '0' THEN -- protection on
command freeze
                                    state <= stop;
                            ELSE
                                    state <= data_out;
                            END IF;

                    WHEN stop =>
                            state <= idle;

                    WHEN OTHERS =>
                            state <= idle;
                     END CASE;
                   END IF;
          END PROCESS;

   ctrl_out <= (NOT (state(4) OR state(3) OR state(2) )) AND s_ackn;  -- stop or data out or
ctrls out or ackn
          trsf <= NOT (state(4) OR state(3));  -- data out or stop
   oeab_h <= NOT (state(4) OR state(3) OR state(2)); -- not (stop or trsfstop  or data out or
ctrls out)
   oeab_l <= NOT (state(4) OR state(3) OR state(2)); -- not (stop or trsfstop  or data out or
ctrls out)
   oeba_h <= state(4) OR state(3) OR state(2); -- stop or trsfstop or data out or ctrls out
   oeba_l <= state(4) OR state(3) OR state(2); -- stop or trsfstop  or data out or ctrls out
   transmit_ready <= state(2) OR state(1); -- ctrls out or wait out
--   ackn <= NOT state(3); -- set_ackn
   ackn <= s_ackn;
   in_out <= NOT (state(4) OR state(3)); -- data out or trsfstop
          sender_reset <= state(0) OR reset; -- release_ackn or reset
```

```
END ARCHITECTURE str22;
```

A.4 Fake_ALTRO_Sender

```
-- $Id: fake_altro_gtl.vhd 48 2009-03-04 22:26:05Z jkral $
--------------------------------------------------------------------------------
-- Title      : Fake altro
-- Project    :
--------------------------------------------------------------------------------
-- File       : fake_altro_gtl.vhd
-- Author     :
-- Company    :
-- Created    : 2009-05-27
-- Last update: 2009-05-27
-- Platform   :
-- Standard   : VHDL'93/02
--------------------------------------------------------------------------------
-- Description:
--------------------------------------------------------------------------------
-- Copyright (c) 2009
--------------------------------------------------------------------------------
-- Revisions  :
-- Date        Version  Author  Description
-- 2009-03-04  1.0      jkral       Created, based on dwang
--------------------------------------------------------------------------------

LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
USE ieee.std_logic_unsigned.ALL;
USE ieee.numeric_std.ALL;
USE work.tru_typedef.ALL;

--------------------------------------------------------------------------------

ENTITY fake_altro_sender IS
  PORT (
    reset           : IN std_logic;

    -- GTL bus input lines
    rclk                    : IN std_logic;

    -- control signals
          data_ready                    : IN std_logic; -- data ready flag from buffers
          zsup_setup                    : IN std_logic; -- zero suppression enable
          zsup_thrsh                    : IN std_logic_vector(9 downto 0); -- zero suppression
```

```
threshold
   is_sparse                  : IN std_logic; -- signal that the requested data is sparse mask

   -- data and addresses
         gtl_branch           : IN std_logic; -- brach bit
         gtl_address    : IN std_logic_vector( 6 downto 0 ); -- which channel to readout
         channel_data : IN std_logic_vector( 39 downto 0 ); -- data for ALTRO transfer, 8
frames

   -- control signals
         transmit_complete    : OUT std_logic;  -- transmit completed signal

         -- GTL bus output lines
   bd                   : OUT std_logic_vector( 39 downto 0 );
         dstb                 : OUT std_logic);

END ENTITY fake_altro_sender;

---------------------------------------------------------------------------

ARCHITECTURE str23 OF fake_altro_sender IS

  TYPE SenderState_type IS ( idle, bufferout, lastout, finish );
  SIGNAL state: SenderState_type;
  SIGNAL dstb_block: std_logic;
  SIGNAL wordcount : std_logic_vector( 9 downto 0 );
  SIGNAL subwordcount      : std_logic_vector( 9 downto 0 );
  SIGNAL timestamp : std_logic_vector( 9 downto 0 );
  SIGNAL zsuppok           : std_logic;
  SIGNAL lastsupp          : std_logic;
  SIGNAL zsuppoknorm       : std_logic;
  SIGNAL lastsuppnorm      : std_logic;
  SIGNAL zsuppokl0id       : std_logic;
  SIGNAL lastsuppl0id      : std_logic;
  SIGNAL suppressing       : std_logic;

  TYPE tintersum      is array(integer range 0 to 3) of std_logic_vector(10 DOWNTO 0);
  SIGNAL intersum          : tintersum;

  SIGNAL isidchannel       : std_logic;
  SIGNAL datainidchannel : std_logic_vector(3 downto 0);

BEGIN

   -- make the data strobe same phase as RCU clock
         -- data are set on rising RCU edge, probed on falling data strobe
         --
   dstb <= rclk OR dstb_block;

         -- threshold comparators, issue 1 on MSB on data <= threshold
```

```
        intersum(0) <= ('1' & zsup_thrsh) - ('0' & channel_data(9 downto 0));
        intersum(1) <= ('1' & zsup_thrsh) - ('0' & channel_data(19 downto 10));
        intersum(2) <= ('1' & zsup_thrsh) - ('0' & channel_data(29 downto 20));
        intersum(3) <= ('1' & zsup_thrsh) - ('0' & channel_data(39 downto 30));

        -- zero suppression green light on two higher words below threshold
        -- or end of data
        zsuppoknorm <= (intersum(3)(10) AND intersum(2)(10)) OR (NOT data_ready);
        -- last frame suppression green light
        lastsuppnorm <= intersum(1)(10) AND intersum(0)(10);

        -- check if is the L0id channel (96 and higher)
        -- 96 in binary is 1100000
        isidchannel <= gtl_address(6) AND gtl_address(5);

        datainidchannel(0) <= channel_data(9) OR channel_data(8) OR channel_data(7)
                    OR channel_data(6) OR channel_data(5) OR channel_data(4)
                    OR channel_data(3) OR channel_data(2) OR channel_data(1)
                    OR channel_data(0);
        datainidchannel(1) <= channel_data(19) OR channel_data(18) OR channel_data(17)
                    OR channel_data(16) OR channel_data(15) OR channel_data(14)
                    OR channel_data(13) OR channel_data(12) OR channel_data(11)
                    OR channel_data(10);
        datainidchannel(2) <= channel_data(29) OR channel_data(28) OR channel_data(27)
                    OR channel_data(26) OR channel_data(25) OR channel_data(24)
                    OR channel_data(23) OR channel_data(22) OR channel_data(21)
                    OR channel_data(20);
        datainidchannel(3) <= channel_data(39) OR channel_data(38) OR channel_data(37)
                    OR channel_data(36) OR channel_data(35) OR channel_data(34)
                    OR channel_data(33) OR channel_data(32) OR channel_data(31)
                    OR channel_data(30);

        -- zero suppression green light on two higher words non zero
        -- or end of data
        zsuppokl0id <= (datainidchannel(3) NOR datainidchannel(2)) OR (NOT data_ready);
        -- last frame suppression greem light
        lastsuppl0id <= datainidchannel(1) NOR datainidchannel(0);


  -- pick the correct zerosupp flag
        zsuppok <= (zsuppokl0id AND isidchannel) OR (zsuppoknorm AND (NOT
isidchannel));
        lastsupp <= (lastsuppl0id AND isidchannel) OR (lastsuppnorm AND (NOT
isidchannel));

  -- state shift
  PROCESS ( reset, rclk ) IS
        BEGIN
    IF reset = '1' THEN
      state <= idle;
```

```
                transmit_complete <= '0';
                dstb_block <= '1';
                bd <= (OTHERS => '1');
                wordcount <= (OTHERS => '0');
                subwordcount <= (OTHERS => '0');
                suppressing <= '1';
                timestamp <= (OTHERS => '0');
            ELSIF rising_edge( rclk ) THEN

              CASE state IS
        -- wait for data ready. data_ready is asserted one clock cycle prior to first data
                -- and disassderted with asserting of the last data
        WHEN idle =>
              transmit_complete <= '0';
                  wordcount <= (OTHERS => '0');
                  subwordcount <= (OTHERS => '0');
                  timestamp <= (OTHERS => '0');
                  suppressing <= '1'; -- suppressing set to 1, to start as if suppressing

              IF data_ready = '1' THEN
                        state <= bufferout;
                      ELSE
                        state <= idle;
                  END IF;

                WHEN bufferout =>
         -- if no zero supp selected, or sparse channel transmit
                  IF zsup_setup = '0' OR is_sparse = '1' THEN
                        dstb_block <= '0'; -- release the data strobe block

                        IF data_ready = '0' THEN -- put in the pre-last frame with headers
                          IF is_sparse = '1' THEN -- no trailer frames for sparse list
                                bd <= channel_data;
                                      state <= finish;
                                      transmit_complete <= '1'; -- transmit complete with last
data
                                    ELSE
                            state <= lastout;
                                  -- wordcount timestamp frame frame
                            bd <= (wordcount + "0000000100") & (timestamp + "0000000001")
                                  & channel_data(19 downto 0);
                          END IF;
                        ELSE
                          bd <= channel_data;
                          state <= bufferout;
                        END IF;

                        wordcount <= wordcount + 4; -- 4 ten bit words in frame
                      ELSE
                        -- do the zsupp
```

```
                        -- finish sending, when suppressable data encoutered
                        -- or when the last frame is going out (and not fully suppressable)
                        -- or start and finish sending at once for partially suppressable
                        IF zsuppok = '1' AND (suppressing = '0' OR lastsupp = '0') THEN
                          bd <= (subwordcount + "0000000100") & (timestamp +
"0000000001")
                                & channel_data(19 downto 0);
                                wordcount <= wordcount + 4;
                                subwordcount <= (OTHERS => '0');
                          dstb_block <= '0';
                                suppressing <= '1';

                        -- keep suppressing when full suppresable, or last frame suppressable
                        ELSIF zsuppok = '1' AND lastsupp = '1' THEN
                                wordcount <= wordcount;
                                subwordcount <= (OTHERS => '0');
                          dstb_block <= '1';
                                suppressing <= '1';

                        -- send data only (begin or continue)
                        ELSE
                          bd <= channel_data;
                                wordcount <= wordcount + 4;
                                subwordcount <= subwordcount + 4;
                          dstb_block <= '0';
                                suppressing <= '0';
                        END IF;

        -- state change on data end
                        IF data_ready = '0' THEN
                          state <= lastout;
                        ELSE
                          state <= bufferout;
                        END IF;
                      END IF;

                      timestamp <= timestamp + 4;

                WHEN lastout =>
                  -- "a" is the filler (1010)
                        -- composition:
                        -- 14 bits - "a"
                        -- 10 bits - ammount of 10 bit words (including timestamps and
wordcounts)
                        -- 4 bits -"a"
                        -- 12 bits address (1 bit branch, 4 bits FEC position and
                        --    7 bits chip and channel ( 3 bits chip, 4 bits channel )
                        --
                        bd <= ( x"aaa" & "10" & wordcount & x"a" & gtl_branch & x"0" &
```

```
gtl_address );
            dstb_block <= '0';
            transmit_complete <= '1'; -- signalize transmit complete with the last frame
       state <= finish;

                 WHEN finish =>
            transmit_complete <= '1';
                  --bd <= (OTHERS => '1');  -- XXX do we really need this ???
                  dstb_block <= '1';
       state <= idle;

     END CASE;
          END IF;
        END PROCESS;

END ARCHITECTURE str23;
```

APPENDIKS B TRU_err_scan_channels_double.sh

```
#!/bin/sh

if [ ! $1 ]
then
  echo "Usage: $0 <branch> [outfile]"
  exit
fi

if [ ! $2 ]
then
  outfilex=`cat /position`
  outfile="/$outfilex${1}_double.dat"
else
  outfile=$2
fi

if [ -f $outfile ]
then
  rm $outfile
fi

for i in 0 1 2 3 4 5 6 7 8 9 10 11
do
  echo -n "$i " >> $outfile
done

echo "" >> $outfile

for i in 2 3 4 5 6 7
do
  dl=`./TRU_read.sh $1 b$i | grep 8002`
  dllw=${dl#*8002: }
  if [ "$1" = B ]
  then
    dllb=${dllw#*0x10}
  else
    dllb=${dllw#*0x}
  fi
  dlln=${dllb# }
  echo -n "$dlln " >> $outfile
done


echo "" >> $outfile

echo -n "del lock mux bse bso" >> $outfile
for i in 0 1 2 3 4 5 6 7 8 9 10 11 12 13
do
  for j in 0 1 2 3 4 5 6 7
  do
    echo -n " $i-$j" >> $outfile
  done
done

echo "" >> $outfile


for i in 0 1 2 3
```

```
do
  for j in 0 1 2 3 4 5 6 7 8 9 a b c d e f
  do
    echo "setting single pattern"
    ./TRU_adc_write.sh $1 0xffff 0x0025 0x0013 > /dev/null
    sleep 1
    ./TRU_adc_write.sh $1 0xffff 0x0026 0xd400 > /dev/null
    sleep 2

    #./TRU_write.sh $1 23 0x$i$j  > /dev/null
    # write all the ADC shifts
    for adi in 5 6 7 8 9 a
    do
      ./TRU_write.sh $1 2$adi 0x$i$j$i$j
    done

    sleep 2
    setp=`./TRU_read.sh $1 25 | grep 8002`
    readback=`./TRU_read.sh $1 b0 | grep 8002`
    mux=`./TRU_read.sh $1 b1 | grep 8002`
    bse=`./TRU_read.sh $1 b6 | grep 8002`
    bso=`./TRU_read.sh $1 b8 | grep 8002`
    bsu=`./TRU_read.sh $1 b9 | grep 8002`
    bsy=`./TRU_read.sh $1 ba | grep 8002`
    rblw=${readback#*8002: }
        if [ "$1" = B ]
        then
          rblb=${rblw#*0x10}
        else
          rblb=${rblw#*0x}
        fi
    rbln=${rblb# }
    setpw=${setp#*8002: }
        if [ "$1" = B ]
        then
          setpb=${setpw#*0x10}
        else
          setpb=${setpw#*0x}
        fi
    setpn=${setpb# }
    muxw=${mux#*8002: }
        if [ "$1" = B ]
        then
          muxb=${muxw#*0x10}
        else
          muxb=${muxw#*0x}
        fi
    muxn=${muxb# }
    bsew=${bse#*8002: }
        if [ "$1" = B ]
        then
          bseb=${bsew#*0x10}
        else
          bseb=${bsew#*0x}
        fi
    bsen=${bseb# }
    bsow=${bso#*8002: }
        if [ "$1" = B ]
        then
          bsob=${bsow#*0x10}
        else
```

```
        bsob=${bsow#*0x}
      fi
    bson=${bsob# }
    bsuw=${bsu#*8002: }
      if [ "$1" = B ]
      then
        bsub=${bsuw#*0x10}
      else
        bsub=${bsuw#*0x}
      fi
    bsun=${bsub# }
    bsyw=${bsy#*8002: }
      if [ "$1" = B ]
      then
        bsyb=${bsyw#*0x10}
      else
        bsyb=${bsyw#*0x}
      fi
    bsyn=${bsyb# }

    echo -n $setpb $rblb $muxb $bseb $bsob $bsub $bsyb >> $outfile

    echo "setting double pattern"
    ./TRU_adc_write.sh $1 0xffff 0x0025 0x002f > /dev/null
    sleep 1
    ./TRU_adc_write.sh $1 0xffff 0x0026 0xd400 > /dev/null
    sleep 2
    ./TRU_adc_write.sh $1 0xffff 0x0027 0x5a00 > /dev/null
    sleep 2

    echo $setpb
    echo "toggling test mode and running error scan ..."
    ./TRU_write.sh $1 42 0x0080 > /dev/null
    sleep 10
    ./TRU_write.sh $1 42 0x0000 > /dev/null

    echo "reading data ..."

    for k in c d e
    do
      for l in 0 1 2 3 4 5 6 7 8 9 a b c d e f
      do
        rb=`./TRU_read.sh $1 $k$l | grep 8002`
        rblw=${rb#*8002: }
        if [ "$1" = B ]
        then
          rblb=${rblw#*0x10}
        else
          rblb=${rblw#*0x}
        fi
        rbln=${rblb# }
        echo -n " $rbln" >> $outfile
      done
    done
    echo "" >> $outfile
  done
done
```

APPENDIKS C Analyse av DATA

Kommunikasjon med AliRoot

```
-bash-3.2$  source /u2/cern/ALICE/precomp/packages/env_aliroot.sh VO_ALICE@AliRoot::v4-20-Rev-27


Welcome to ALICE

++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

Available AliRoot versions:
                        VO_ALICE@AliRoot::v4-20-Rev-27

To change AliRoot version do :
        source /u2/cern/ALICE/precomp/packages/env_aliroot.bash <VO_ALICE@AliRoot::aliroot verion>

Following example showing u how to setup latest Rev of AliRoot :
        source /u2/cern/ALICE/precomp/packages/env_aliroot.bash VO_ALICE@AliRoot::v4-20-Rev-27

By default latest AliRoot AN version is set : VO_ALICE@AliRoot::packages

++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

Currently running : AliEn   alien.v2-19.112 /u2/cern/ALICE/precomp/packages/alien.v2-19.112
              : ROOT    VO_ALICE@ROOT::v5-28-00d /u2/cern/ALICE/precomp/packages/VO_ALICE/ROOT/v5-28-00d/v5-28-00d
              : GEANT3  VO_ALICE@geant3::v1-11-24 /u2/cern/ALICE/precomp/packages/VO_ALICE/GEANT3/v1-11-24/v1-11-24
              : AliROOT VO_ALICE@AliRoot::v4-20-Rev-27 /u2/cern/ALICE/precomp/packages/VO_ALICE/AliRoot/v4-20-Rev-27/v4-20-
Rev-27

++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
ALIROOT variables OK
All OK
-bash-3.2$

-bash-3.2$ aliroot  jkphs_tru_eventdisplay.cpp'("10000001234001.10.root",6,10,1,0)'
 ******************************************
 *                                        *
 *       W E L C O M E  to  R O O T       *
 *                                        *
 *   Version  5.28/00d       7 May 2011   *
 *                                        *
 *  You are welcome to visit our Web site  *
 *         http://root.cern.ch            *
 *                                        *
 ******************************************

ROOT 5.28/00d (tags/v5-28-00d@39145, May 10 2011, 18:07:26 on linuxx8664gcc)

CINT/ROOT C/C++ Interpreter version 5.18.00, July 2, 2010
Type ? for help. Commands must be C++ statements.
Enclose multiple statements between { }.
root [0]
Processing jkphs_tru_eventdisplay.cpp("10000001234001.10.root",6,10,1,0)...
I-AliRawReader::Create: Creating raw-reader in order to read raw-data file: 10000001234001.10.root
events counted: 40
Physics events total: 40
11
 <message repeated 1 time>
I-TCanvas::MakeDefCanvas:  created default TCanvas with name c1
Event 1 with x max 13
TRU event display done.
root [1]
.q
-bash-3.2$
```

jkphs_tru_eventdisplay:

```
const int nColsFaltro = 48;
const int nTrus = 30;
const int nRowsFaltro = 60;
const int nFastOrChannels = 96;
const int nL0idChannels = 11;
const int nChannelBits = 10;
const int cTruFlag = 2;
const int cEventsDiv = 30;
const int cEventMaxFrames = 1024;
const int cYMin = 0;
const int cYMax = 200;


#include "jkemc_lib_tru.cpp"

void jkphs_tru_eventdisplay( char *filename, int tru, int evStart, int
evCount, int doCalib = 0 ){

        Int_t nPhysics, nCalib;
        Int_t iPhysics;
        Int_t iProcessed;
        UInt_t evtType;
        Int_t maxEvLen;
        UShort_t *sig;
        UShort_t sigmax;

        Int_t row, column, ddl, module, caloflag, branch, globRow,
globColumn, truPos;
        Int_t curBin;
        Int_t i,j,k;
        Int_t tlefttop, tleftbot, trighttop, trightbot;
        Int_t vlefttop, vleftbot, vrighttop, vrightbot;
        char orid, andid;

        char buf[30];
        char buf2[60];

        TH1F *channelhistos[nFastOrChannels];
        char l0ids[nL0idChannels*nChannelBits][cEventMaxFrames];

        TSortedList *fileList;
        TObject *obj;
        TObjString *objstr, *filenamestr;

        TCanvas *cc, *cm;
        TVirtualPad *pad;
        TBox *box;
        TLine *line1, *line2, *line3, *line4, *line5;
        TH2F *idhisto, *maxhisto;

        fileList = new TSortedList();
```

```
        filenamestr = new TObjString( filename );
        fileList->Add( filenamestr );
//      jkemc_getRunFileList( fileList, dir, mask );

        gStyle->SetPalette( 1 );

        // prepare histos
        for( i = 0; i < nFastOrChannels; i++ ){
                sprintf( buf2, "channel %d", i );
                channelhistos[i] = new TH1F( buf2, buf2, cEventMaxFrames, -
0.5, cEventMaxFrames - 0.5 );
        }

        sprintf( buf2, "L0IDs", i );
        idhisto = new TH2F(    buf2, buf2, 124, -0.5, 123.5,

nL0idChannels*nChannelBits, -0.5, nL0idChannels*nChannelBits - 0.5);

        sprintf( buf2, "channel maxima", i );
        maxhisto = new TH2F( buf2, buf2, nColsFaltro, -0.5, nColsFaltro -
0.5,
                                        nRowsFaltro, -0.5, nRowsFaltro -
0.5);

        AliCaloRawStreamV3 * inStream;

        AliRawReader *rawReader;
        AliRawEventHeaderBase *aliHeader;

        TIter *fliter;

        nPhysics = 0;
        nCalib = 0;

        // iterate over the file list
        fliter = new TIter( fileList );

        while( objstr = (TObjString*)fliter->Next() ){

                // set up a raw reader of the data
                rawReader = AliRawReader::Create(objstr->String());
                aliHeader = NULL;

                // reader init
                rawReader->Reset();

                inStream = new AliCaloRawStreamV3(rawReader,"PHOS");

                // no STU select
//              rawReader->Select("EMCAL", 0, 43);

                // compute number of physics events first
                while ( rawReader->NextEvent() ) {
                        aliHeader = (AliRawEventHeaderBase*) rawReader-
>GetEventHeader();
                        evtType = rawReader->GetType();
                        if( evtType == AliRawEventHeaderBase::kPhysicsEvent
)
                                nPhysics++;
                        if (evtType ==
AliRawEventHeaderBase::kCalibrationEvent && doCalib == 1)
```

```
                        nCalib++;
                }
                delete inStream;
                delete rawReader;
                cout << "events counted: " << nPhysics << endl;
        }
        cout << "Physics events total: " << nPhysics << endl;
        if( doCalib == 1 )
                cout << "Calib events total: " << nCalib << endl;

        if(( nPhysics == 0 && nCalib == 0 && doCalib == 1 )
                || (nPhysics == 0 && doCalib == 0 )){
                cout << "No physics events in the run, quit." << endl;
                exit();
        }

        // reset the file iterator
        fliter->Reset();

        iPhysics = 0;
        iProcessed = 0;

        // iterate over files
        while( objstr = (TObjString*)fliter->Next() ){

                // set up a raw reader of the data
                rawReader = AliRawReader::Create(objstr->String());
                aliHeader = NULL;

                // reader init
                rawReader->Reset();

                inStream = new AliCaloRawStreamV3(rawReader,"PHOS");

                // no STU select
//              rawReader->Select("EMCAL", 0, 43);

                // loop over events
                while ( rawReader->NextEvent() ) {

                        aliHeader = (AliRawEventHeaderBase*) rawReader-
>GetEventHeader();
                        evtType = rawReader->GetType();
                        if( evtType == AliRawEventHeaderBase::kPhysicsEvent
||
                                (evtType ==
AliRawEventHeaderBase::kCalibrationEvent && doCalib == 1)){
        // skip unwanted events
        if( iPhysics < evStart ){
                iPhysics++;
                continue;
        }

        if( iPhysics >= evStart + evCount )
                break;


                                iPhysics++;
        iProcessed++;

        //if(evtType != AliRawEventHeaderBase::kCalibrationEvent )
```

```
	//	continue;

	cout << iPhysics;
	if( evtType == AliRawEventHeaderBase::kCalibrationEvent )
		cout << " calib ";
	cout << endl;

	// clear everything
	maxEvLen = 0;
	for( i = 0; i < nL0idChannels*nChannelBits; i++ )
		for( j = 0; j < cEventMaxFrames; j++)
			l0ids[i][j] = 0;

			for( i = 0; i < nFastOrChannels; i++ ){
				channelhistos[i]->Reset();
			}
			maxhisto->Reset();

			while ( inStream->NextDDL() ) {
				while ( inStream->NextChannel() ) {

			// get position data
					caloflag = inStream->GetCaloFlag();
					module = inStream->GetModule();
					ddl = inStream->GetDDLNumber();
					branch = inStream->GetBranch();
					row = inStream->GetRow();
					column = inStream->GetColumn();

					jkemc_getTruPosition( &truPos,
module, ddl, branch );
					jkemc_getTruChannelPosition(
&globRow, &globColumn, module, ddl, branch, column );

/*					if( inStream->IsTRUData() )
						cout << module << " " << ddl
<< " " << branch << " " << row << " " << column << endl;*/

					// just maximas when not the selected
TRU
					if( truPos != tru ){
						if( inStream->IsTRUData() &&
column < nFastOrChannels ){
							sigmax = 0;
							while (inStream-
>NextBunch()){
								sig =
inStream->GetSignals();
								for( i=0; i <
inStream->GetBunchLength(); i++) {
									if(
sig[i] > sigmax )
									sigmax
= sig[i];
								}
							}	// bunches
							// fill maximas histo
							maxhisto->Fill(
globColumn, globRow, sigmax );
						}
						continue;
```

```
                                        }

                                        // TRU flag and fast OR channels
                                        if( inStream->IsTRUData() && column <
nFastOrChannels ){
                                                sigmax = 0;
                                                while (inStream-
>NextBunch()){
                                                        sig = inStream-
>GetSignals();
                                                        curBin = inStream-
>GetStartTimeBin();

                                                        // search for the
farthest timestamp
                                                        if( curBin > maxEvLen
)
                                                                maxEvLen =
curBin;

                                                        for( i=0; i <
inStream->GetBunchLength(); i++) {
                                                                if( sig[i] >
sigmax )
                                                                        sigmax
= sig[i];
                                                        // fill signal

        channelhistos[column]->Fill( curBin, sig[i] );
                                                                curBin--;
                                                        }
                                                }// bunches
                                                // fill maximas histo
                                                maxhisto->Fill( globColumn,
globRow, sigmax );
                                        }
                                        // TRU L0 ID data
                                        else if( inStream->IsTRUData() ) {
                                                while (inStream-
>NextBunch()){
                                                        sig = inStream-
>GetSignals();
                                                        curBin = inStream-
>GetStartTimeBin();

                                                        // search for the
farthest timestamp
                                                        if( curBin > maxEvLen
)
                                                                maxEvLen =
curBin;

                                                        for( i = 0; i <
inStream->GetBunchLength(); i++ ) {
                                                                for( j = 0; j
< nChannelBits; j++ ){

                                                                        //
check if the bit j is 1
```

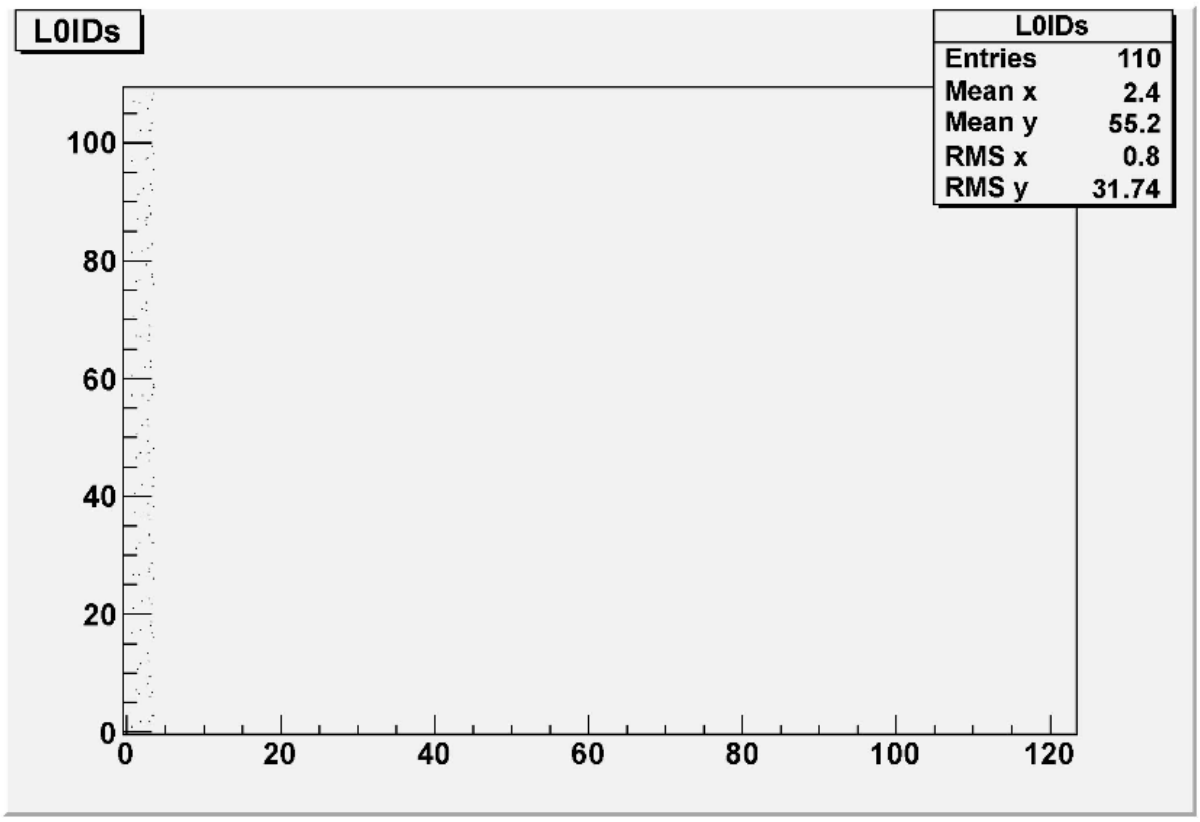
```
                                                                    if(
(sig[i] & ( 1 << j )) > 0 ){

        l0ids[(column-nFastOrChannels)*nChannelBits+j][curBin] = 1;

        idhisto->Fill( curBin, (column-nFastOrChannels)*nChannelBits+j );
                                                                    }
                                                            }
                                                            curBin--;
                                                    }
                                            }
                                    }// L0ID data
                            } // channel
                    } // DDL

                    idhisto->Draw();
//      break;

                                    // delete canvas for every consecutive
display
                                    if( iProcessed > 1 ){
                    delete cc;
                    delete box;

                    delete line1;
                    delete line2;
                    delete line3;
                    delete line4;
                    delete line5;
                    delete cm;
        }

                                    sprintf(buf,"TRU %d max", tru);

                                    cm = new TCanvas( buf, "signal maximas", 20,
20, 800, 1000 );
                                    cm->cd();
                                    cm->SetFillColor(kWhite);
                                    maxhisto->SetStats( kFALSE );
                                    maxhisto->SetOption("COLZ");
                                    maxhisto->Draw();
                                    line1 = new TLine( 23.5, -0.5, 23.5, 59.5 );
                                    line1->Draw();
                                    line2 = new TLine(     -0.5, 11.5, 47.5, 11.5
);
                                    line2->Draw();
                                    line3 = new TLine(     -0.5, 23.5, 47.5, 23.5
);
                                    line3->Draw();
                                    line4 = new TLine(     -0.5, 35.5, 47.5, 35.5
);
                                    line4->Draw();
                                    line5 = new TLine(     -0.5, 47.5, 47.5, 47.5
);
                                    line5->Draw();
                                    cm->cd();
        cm->Update();
        gSystem->Sleep(1000);

                                    sprintf(buf,"TRU %d", tru);
```

```
                                // new canvas
                                cc = new TCanvas( buf, "FALTRO data", 1440,
820 );
                                cc->Divide( 12, 8 );

        box = new TBox();

        // show histos
                                for( i = 0; i < 12; i++ ){ // top half
                for( j = 0; j < 8; j++ ){
                        // get inner TRU coordinates
                        row = j % 4;
                        if( j < 4 )
                                column = i;
                        else
                                column = i + 12;

                        // pad select
                        pad = cc->cd(j*12+i+1);
                        pad->SetMargin( 0., 0., 0., 0. );

                        // draw the histo
                                                channelhistos[column*4+row]-
>SetTitle("");
                                                channelhistos[column*4+row]-
>SetFillColor(5);
                                                channelhistos[column*4+row]-
>SetMaximum(cYMax);
                                                channelhistos[column*4+row]-
>SetMinimum(cYMin);
                        channelhistos[column*4+row]->SetStats(kFALSE);
                        channelhistos[column*4+row]->SetAxisRange(0.,
maxEvLen);
                                                channelhistos[column*4+row]-
>Draw();
//                                              channelhistos[column*4+row]-
>Write();

//                      // get the trigger ID coordinates
//                      if( row > 0 & column > 0 )
//                              tlefttop = (column-1)*3+row-1;
//                      else
//                              tlefttop = -1;
//
//                      if( row < 3 & column > 0 )
//                              tleftbot = (column-1)*3+row;
//                      else
//                              tleftbot = -1;
//
//                      if( row > 0 & column < 23 )
//                              trighttop = column*3+row-1;
//                      else
//                              trighttop = -1;
//
//                      if( row < 3 & column < 23 )
//                              trightbot = column*3+row;
//                      else
//                              trightbot = -1;
//
//                      cout << column*4+row << " " << column << " " << row
<< " ";
```

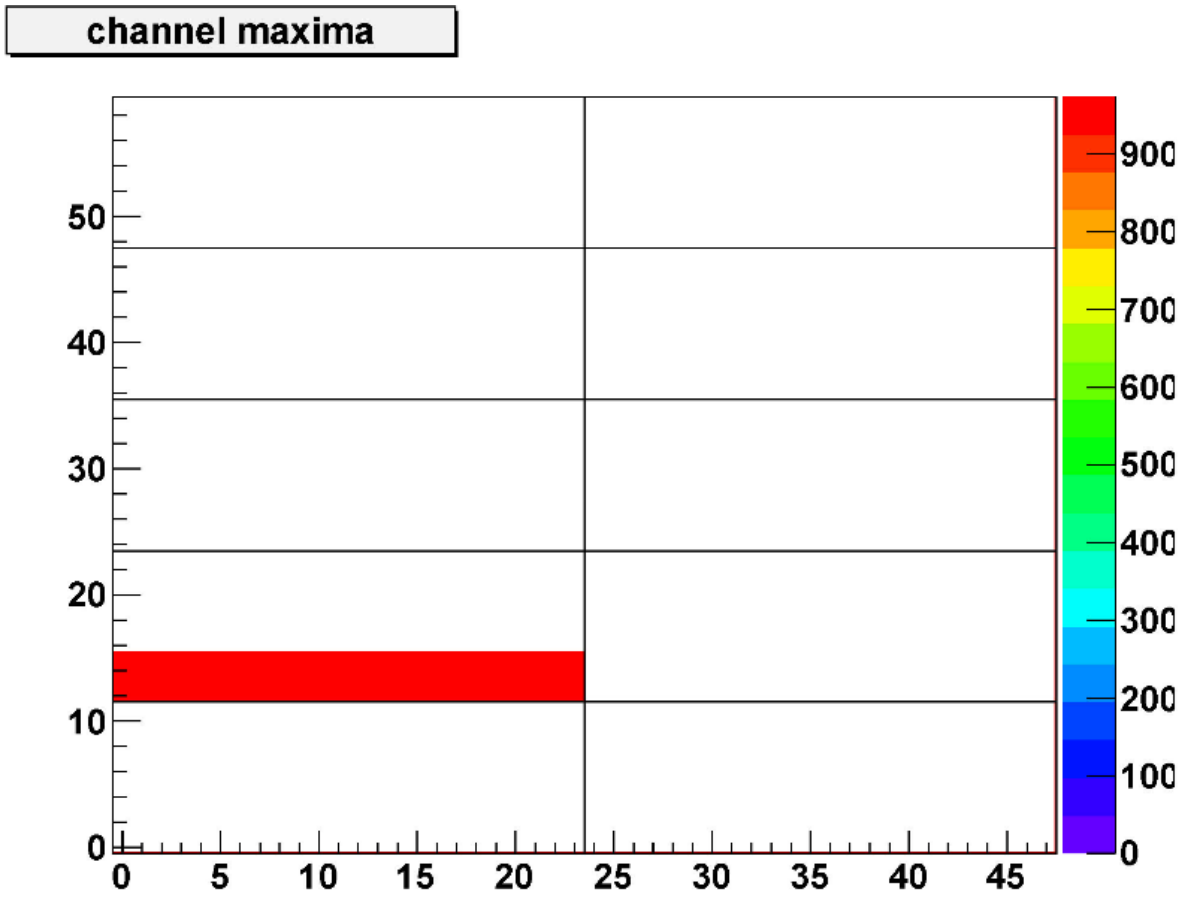
```
//                      cout << tlefttop << " " << tleftbot << " " <<
trighttop << " " << trightbot << endl;

                        for( k = 0; k <= maxEvLen; k++ ){
/*                              orid = 0;
                                andid = 1;

                                if( tlefttop != -1 ){
                                        orid = orid | l0ids[tlefttop][k];
                        andid = andid & l0ids[tlefttop][k];
                                }
                                if( tleftbot != -1 ){
                                        orid = orid | l0ids[tleftbot][k];
                        andid = andid & l0ids[tleftbot][k];
                                }
                                if( trighttop != -1 ){
                                        orid = orid | l0ids[trighttop][k];
                        andid = andid & l0ids[trighttop][k];
                                }
                                if( trightbot != -1 ){
                                        orid = orid | l0ids[trightbot][k];
                        andid = andid & l0ids[trightbot][k];
                                }

                                // or of all intersecting 2+2 trigger region
                                box->SetFillColor(3);
                                if( orid > 0 )
                                        box->DrawBox(k,60,k+1,65);

                                // and of all intersecting 2x2 regions
                                box->SetFillColor(4);
                                if( andid == 1 )
                                        box->DrawBox(k,65,k+1,70);*/

                                // L0 generated
                                box->SetFillColor(2);
                                if( l0ids[column*4+row][k] > 0 )
                                        box->DrawBox(k-0.5,70,k+0.5,75);

                                // inhibit
                                box->SetFillColor(1);
                                if( l0ids[97+column/2][k] > 0 )
                                        box->DrawBox(k-0.5,75,k+0.5,80);

                                // global L0
                                box->SetFillColor(3);
                                if( l0ids[96][k] > 0 )
                                        box->DrawBox(k-0.5,80,k+0.5,85);

                                // LED ID
                                box->SetFillColor(4);
                                if( l0ids[109][k] > 0 )
                                        box->DrawBox(k-0.5,90,k+0.5,95);
                        }

                //      box->DrawBox(10,10,20,20);
                }
                                }
```

```
        cout << "Event " << iProcessed << " with x max " << maxEvLen <<
endl;
                                cc->cd(0);
                                cc->Update();
                                // sleep for 5 sec
                                gSystem->Sleep(3000);
                        } // physics
                } // event
                // delete inStream;
                // delete rawReader;
        } // files

//      delete fliter;
//      delete fileList;

        cout << "TRU event display done." << endl;
}
```

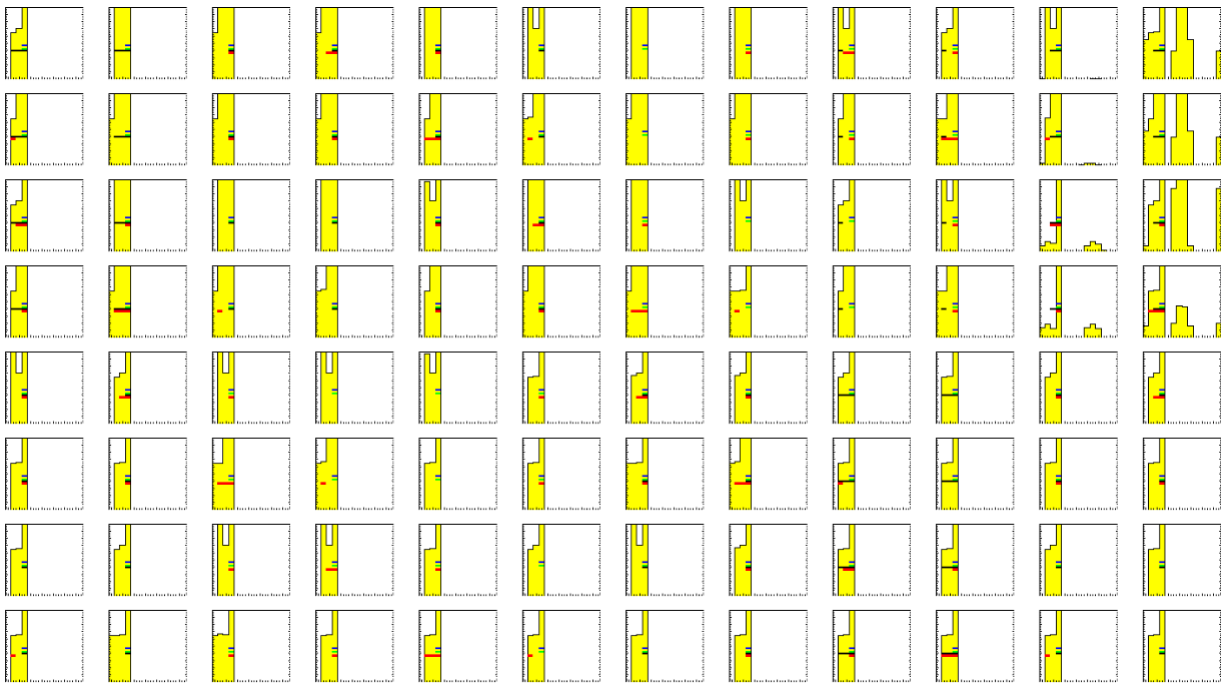
C.2 Resultater av rådataen fra Fake ALTRO buffer



Figur C.1 L0ID



Figur C.2 Maksimum signal i kanal



Figur C.3 Multiplot av rådataen

Figur C.1 er en maskering av L0 signalene av 2×2 kanaler, x-aksen er tid og y-aksen er kanaler.

Figur C.2 viser maksimum amplitude av signalet for en EMCal setup, x-aksen = eta og y-aksen = phi.

Figur C.3 viser 8×14 grafer av de 96 kanalene som er skrevet ut. X-aksen viser tiden mens y-aksen viser amplituden så dette er en oversikt over rådataen.

APPENDIKS D Fake_ALTRO_Buffer_phos

```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
USE ieee.std_logic_unsigned.ALL;
USE ieee.numeric_std.ALL;
USE work.tru_typedef.ALL;

--------------------------------------------------------------------------------

ENTITY fake_altro_buffers IS
  PORT (
    g_clk                   : IN std_logic; -- 40MHz global clock
    rcu_clk                 : IN std_logic; -- 40MHz RCU clock
          reset             : IN std_logic; -- reset

          eventwidth : IN std_logic_vector(7 downto 0); -- L1 rollback for circular copy
          L1rollback : IN std_logic_vector(7 downto 0); -- L1 rollback for circular copy
          zsup_setup        : IN std_logic; -- zero suppression enable
          zsup_thrsh        : IN std_logic_vector(9 downto 0); -- zero suppression threshold
    meb_onoff     : IN std_logic; -- MEB on/off

    L1                      : IN std_logic;   -- GTL L1
    L2                      : IN std_logic;   -- GTL L2

    rdinc           : IN std_logic;   -- read pointer increase signal
    wrinc                   : IN std_logic;   -- write pointer increase signal

    data_in                 : IN std_logic_vector(1119 downto 0); -- 2x2 data 10 bit
    l0id_data_in  : IN std_logic_vector(109 downto 0); -- L0 ID, L0, masks, LED ID

          gtl_address       : IN std_logic_vector( 6 downto 0 ); -- which channel to readout
    prepare_data : IN std_logic; -- signal to prepare data for given channel
    is_sparse               : IN std_logic; -- signal that the requested data is sparse mask

          channel_data    : OUT std_logic_vector( 39 downto 0 ); -- data for ALTRO transfer, 8 frames
          data_ready        : OUT std_logic); -- channel_data ready signal

END ENTITY fake_altro_buffers;

--------------------------------------------------------------------------------

ARCHITECTURE str21 OF fake_altro_buffers IS

COMPONENT fake_altro_circular IS
  PORT (
          clka:    IN std_logic;
          dina:    IN std_logic_VECTOR(1119 downto 0);
          addra:  IN std_logic_VECTOR(7 downto 0);
          wea:              IN std_logic_VECTOR(0 downto 0);
          clkb:    IN std_logic;
```

```
        addrb: IN std_logic_VECTOR(7 downto 0);
        enb:       IN std_logic;
        doutb: OUT std_logic_VECTOR(1119 downto 0));
END COMPONENT;

COMPONENT fake_altro_meb IS
  PORT (
        clka:  IN std_logic;
        dina:  IN std_logic_VECTOR(319 downto 0);
        addra: IN std_logic_VECTOR(9 downto 0);
        wea:       IN std_logic_VECTOR(0 downto 0);
        clkb:  IN std_logic;
        addrb: IN std_logic_VECTOR(14 downto 0);
        enb:       IN std_logic;
        doutb: OUT std_logic_VECTOR(9 downto 0));
END COMPONENT;

COMPONENT fake_altro_l0id_circular IS
  PORT (
        clka:  IN std_logic;
        dina:  IN std_logic_VECTOR(109 downto 0);
        addra: IN std_logic_VECTOR(7 downto 0);
        wea:       IN std_logic_VECTOR(0 downto 0);
        clkb:  IN std_logic;
        addrb: IN std_logic_VECTOR(7 downto 0);
        enb:       IN std_logic;
        doutb: OUT std_logic_VECTOR(109 downto 0));
END COMPONENT;

COMPONENT fake_altro_l0id_meb IS
  PORT (
        clka:  IN std_logic;
        dina:  IN std_logic_VECTOR(39 downto 0);
        addra: IN std_logic_VECTOR(9 downto 0);
        wea:       IN std_logic_VECTOR(0 downto 0);
        clkb:  IN std_logic;
        addrb: IN std_logic_VECTOR(11 downto 0);
        enb:       IN std_logic;
        doutb: OUT std_logic_VECTOR(9 downto 0));
END COMPONENT;

COMPONENT fake_altro_sparse IS
  PORT (
        clka:  IN std_logic;
        dina:  IN std_logic_VECTOR(106 downto 0);
        addra: IN std_logic_VECTOR(3 downto 0);
        wea:       IN std_logic_VECTOR(0 downto 0);
        clkb:  IN std_logic;
        addrb: IN std_logic_VECTOR(3 downto 0);
        enb:       IN std_logic;
        doutb: OUT std_logic_VECTOR(106 downto 0));
END COMPONENT;

  SIGNAL data_copy    : std_logic_vector(1119 downto 0); -- 2x2 data 10 bit to copy
  SIGNAL data_copyb   : std_logic_vector(1279 downto 0); -- 2x2 data 10 bit to copy into rbuf
```

```
 SIGNAL zeroes                : std_logic_vector(39 downto 0); -- to fill the copydata to ram size
 SIGNAL data_copyb1 : std_logic_vector(1119 downto 0); -- 2x2 data transformed 10 bit to copy
 SIGNAL data_copyb2 : std_logic_vector(1119 downto 0); -- 2x2 data transformed 10 bit to copy
 SIGNAL data_copyb3 : std_logic_vector(1119 downto 0); -- 2x2 data transformed 10 bit to copy
 SIGNAL data_copyb4 : std_logic_vector(1119 downto 0); -- 2x2 data transformed 10 bit to copy
 SIGNAL data_prep     : std_logic_vector(39 downto 0); -- 40 bit prepare
 SIGNAL cbuf_waddr  : std_logic_vector(7 downto 0); -- circular buffer write address
 SIGNAL cbuf_raddr   : std_logic_vector(7 downto 0); -- circular buffer read address
 SIGNAL rbuf_waddr  : std_logic_vector(9 downto 0); -- read buffer write address
 SIGNAL rbuf_wbase  : std_logic_vector(9 downto 0); -- event start address
 SIGNAL rbuf_wplus   : std_logic_vector(9 downto 0); -- waddr from event start
 SIGNAL rbuf_raddr   : std_logic_vector(59 downto 0); -- 4x 15bit read buffer read address
 SIGNAL rbuf_rbase   : std_logic_vector(9 downto 0); -- read buffer event address
 SIGNAL rbuf_rframe  : std_logic_vector(14 downto 0); -- read buffer frame address
 SIGNAL rbuf_rplus1  : std_logic_vector(14 downto 0); -- read buffer 10bit position
 SIGNAL rbuf_rplus2  : std_logic_vector(14 downto 0); -- read buffer 10bit position
 SIGNAL rbuf_rplus3  : std_logic_vector(14 downto 0); -- read buffer 10bit position
 SIGNAL rbuf_rplus4  : std_logic_vector(14 downto 0); -- read buffer 10bit position
 SIGNAL copy_en               : std_logic;
 SIGNAL copy_en_v   : std_logic_vector(0 downto 0);
 SIGNAL prep_en               : std_logic;
 SIGNAL L2_del                : std_logic;
 SIGNAL wrinc_del     : std_logic;
 SIGNAL rdinc_del     : std_logic;

 SIGNAL l0id_data_copy        : std_logic_vector(109 downto 0); -- l0 data to copy between bufs
 SIGNAL l0id_data_copyb       : std_logic_vector(159 downto 0);
 SIGNAL l0id_data_copyb1      : std_logic_vector(109 downto 0);
 SIGNAL l0id_data_copyb2      : std_logic_vector(109 downto 0);
 SIGNAL l0id_data_copyb3      : std_logic_vector(109 downto 0);
 SIGNAL l0id_data_copyb4      : std_logic_vector(109 downto 0);
 SIGNAL l0id_rbuf_raddr       : std_logic_vector(47 downto 0);
 SIGNAL l0id_data_prep        : std_logic_vector(39 downto 0); -- 40 bit prepare

 SIGNAL sparse_data           : std_logic_vector(106 downto 0);
 SIGNAL sbuf_waddr            : std_logic_vector(3 downto 0);
 SIGNAL sbuf_raddr            : std_logic_vector(3 downto 0);
 SIGNAL scopy_en_v            : std_logic_vector(0 downto 0);
 SIGNAL sprep_en                    : std_logic;
 SIGNAL sparse_data_prep            : std_logic_vector(106 downto 0);

 TYPE copystate_type IS ( idle, copystart, copying );
 SIGNAL copystate : copystate_type;
 TYPE prepstate_type IS ( idle, readstart, sending );
 SIGNAL prepstate : prepstate_type;
 TYPE sparsestate_type IS ( idle, listmake, listdone );
 SIGNAL sparsestate : sparsestate_type;

 SIGNAL ram_id                : std_logic_vector(1 downto 0);
 SIGNAL ram_addr              : std_logic_vector(4 downto 0);
 SIGNAL l0id_ram_addr               : std_logic_vector(4 downto 0);

 -- do not change those unles you check the code, if it works for new values
--  CONSTANT eventwidth       : integer  := 8; -- number 40bit words in one event (samples/4)
```

```
  SIGNAL eventwidth4 : std_logic_vector( 9 downto 0);
  SIGNAL meb_onoff_v : std_logic_vector( 9 downto 0);

  SIGNAL counter : std_logic_vector( 7 downto 0 );

BEGIN

  -- compute event width in multiples of 4 samples
  eventwidth4 <= eventwidth & "00";

  -- MEB enable vector
  meb_onoff_v <= (OTHERS=>meb_onoff);

  -- get last two bits of address to decide how the data are ordered in ram
  ram_id <= gtl_address(1 downto 0);
  -- position in selected ram ordering
  ram_addr <= gtl_address( 6 downto 2 );
  -- l0 id address is 96 less, 96/4 = 11000 = 24
  l0id_ram_addr <= gtl_address( 6 downto 2 ) - "11000";

  zeroes <= (OTHERS => '0');

  -- rotate the circular buffer address
  PROCESS( g_clk, reset ) IS
  BEGIN
    IF reset = '1' THEN
            cbuf_waddr <= ( OTHERS => '0' );
          ELSIF rising_edge( g_clk) THEN
            cbuf_waddr <= cbuf_waddr + 1;
    END IF;
  END PROCESS;

  -- transfor data to copy from circular buffer to four separate read buffers.
  -- To distribute channels so, that first channel goes to the first RAM, second
  -- to the second ... fourth to the fourth, fifth to the first.
  ctr: FOR i IN 0 TO 23 GENERATE
    ctrb1: FOR j IN 0 TO 3 GENERATE
      data_copyb1( (i+1)*10-1+j*280 downto i*10+j*280 ) <=
                                        data_copy( ((i*4)+j+1)*10-1 downto ((i*4)+j)*10 );
          END GENERATE ctrb1;
    ctrb2: FOR j IN 0 TO 2 GENERATE
      data_copyb2( (i+1)*10-1+(j+1)*280 downto i*10+(j+1)*280 ) <=
                                        data_copy( ((i*4)+j+1)*10-1 downto ((i*4)+j)*10 );
      data_copyb4( (i+1)*10-1+j*280 downto i*10+j*280 ) <=
                                        data_copy( ((i*4)+j+1+1)*10-1 downto ((i*4)+j+1)*10 );
          END GENERATE ctrb2;
    ctrb3: FOR j IN 0 TO 1 GENERATE
      data_copyb3( (i+1)*10-1+(j+2)*280 downto i*10+(j+2)*280 ) <=
                                        data_copy( ((i*4)+j+1)*10-1 downto ((i*4)+j)*10 );
      data_copyb3( (i+1)*10-1+j*280 downto i*10+j*280 ) <=
                                        data_copy( ((i*4)+j+2+1)*10-1 downto ((i*4)+j+2)*10 );
          END GENERATE ctrb3;

    data_copyb2( (i+1)*10-1 downto i*10 ) <=
                                        data_copy( ((i*4)+3+1)*10-1 downto ((i*4)+3)*10 );
```

```
  data_copyb4( (i+1)*10-1+3*280 downto i*10+3*280 ) <=
                                    data_copy( ((i*4)+1)*10-1 downto ((i*4))*10 );
 END GENERATE ctr;

 -- the same for L0 IDs
 l0id_data_copyb1 <= l0id_data_copy(79 downto 70) & l0id_data_copy(39 downto 30)
                                                  & l0id_data_copy(109 downto 100)
             & l0id_data_copy(69 downto 60) & l0id_data_copy(29 downto 20)
                                                  & l0id_data_copy(99 downto 90)
             & l0id_data_copy(59 downto 50) & l0id_data_copy(19 downto 10)
                                                  & l0id_data_copy(89 downto 80)
             & l0id_data_copy(49 downto 40) & l0id_data_copy(9 downto 0);
 l0id_data_copyb2 <= l0id_data_copy(109 downto 100)
             & l0id_data_copy(69 downto 60) & l0id_data_copy(29 downto 20)
                                                  & l0id_data_copy(99 downto 90)
             & l0id_data_copy(59 downto 50) & l0id_data_copy(19 downto 10)
                                                  & l0id_data_copy(89 downto 80)
             & l0id_data_copy(49 downto 40) & l0id_data_copy(9 downto 0)
             & l0id_data_copy(79 downto 70) & l0id_data_copy(39 downto 30);
 l0id_data_copyb3 <= l0id_data_copy(99 downto 90)
             & l0id_data_copy(59 downto 50) & l0id_data_copy(19 downto 10)
                                                  & l0id_data_copy(89 downto 80)
             & l0id_data_copy(49 downto 40) & l0id_data_copy(9 downto 0)
             & l0id_data_copy(79 downto 70) & l0id_data_copy(39 downto 30)
                                                  & l0id_data_copy(109 downto 100)
             & l0id_data_copy(69 downto 60) & l0id_data_copy(29 downto 20);
 l0id_data_copyb4 <= l0id_data_copy(89 downto 80)
             & l0id_data_copy(49 downto 40) & l0id_data_copy(9 downto 0)
             & l0id_data_copy(79 downto 70) & l0id_data_copy(39 downto 30)
                                                  & l0id_data_copy(109 downto 100)
             & l0id_data_copy(69 downto 60) & l0id_data_copy(29 downto 20)
                                                  & l0id_data_copy(99 downto 90)
                                                  & l0id_data_copy(59 downto 50) &
l0id_data_copy(19 downto 10);

 -- copy data from circular to readout buffers
 -- after L1 received, 8 samples is coppied
 PROCESS( g_clk, reset ) IS
 BEGIN
   IF reset = '1' THEN
                 copy_en <= '0';
                 copy_en_v <= "0";
                 rbuf_waddr <= "0000000000"; -- to start writing to 0
                 rbuf_wbase <= "0000000000"; -- to start writing to 0
                 rbuf_wplus <= "0000000000"; -- to start writing to 0
                 cbuf_raddr <= x"00";
                 sbuf_waddr <= (OTHERS => '0');
                 L2_del <= '1'; -- no trigger
                 wrinc_del <= '0'; -- no command
          ELSIF rising_edge( g_clk ) THEN
                 -- shift the base address on L2 (or write increase command)
                 -- L2 lasts at 2 cycles at least, so one wants to be sure to shift the registry
                 -- once only. Old-new value comparison, trigger on new L2 only
                 --
                 IF (L2 = '0' AND L2_del = '1') OR (wrinc = '1' AND wrinc_del = '0') THEN
```

```
                rbuf_wbase <= rbuf_wbase + ( eventwidth4 AND meb_onoff_v );
                sbuf_waddr <= sbuf_waddr + ("000" & meb_onoff);
               END IF;
    L2_del <= L2; -- save current L2 value
               wrinc_del <= wrinc;

        CASE copystate IS
               WHEN idle =>
                -- no need to do the same trick for L1, since whole machinery starts on L1 start
                --
                copy_en_v <= "0";  -- disable copy write procedure
                IF L1 = '0' THEN
                  copy_en <= '1'; -- enable copy read procedure

      -- roll back in circular buffer defined ammount of samples
                  cbuf_raddr <= cbuf_waddr - L1rollback;

                        copystate <= copystart;
     ELSE
                        copystate <= idle;
               END IF;
    WHEN copystart =>
               cbuf_raddr <= cbuf_raddr + 1; -- rotate the circular read address
               rbuf_wplus <= "0000000000"; -- zero the in-event address

               copystate <= copying;
              WHEN copying =>
               -- chose transformation type for write data
               CASE rbuf_wplus(1 downto 0) IS
               WHEN "00" =>
      data_copyb <= zeroes & data_copyb1(1119 downto 840)
                           & zeroes & data_copyb1(839 downto 560)
                                    & zeroes & data_copyb1(559 downto 280)
                                    & zeroes & data_copyb1(279 downto 0);

      l0id_data_copyb <= zeroes(19 downto 0) & l0id_data_copyb1(109 downto 90)
                           & zeroes(9 downto 0) & l0id_data_copyb1(89 downto 60)
                                    & zeroes(9 downto 0) &
l0id_data_copyb1(59 downto 30)
                                    & zeroes(9 downto 0) &
l0id_data_copyb1(29 downto 0);
               WHEN "01" =>
      data_copyb <= zeroes & data_copyb2(1119 downto 840)
                           & zeroes & data_copyb2(839 downto 560)
                                    & zeroes & data_copyb2(559 downto 280)
                                    & zeroes & data_copyb2(279 downto 0);

      l0id_data_copyb <= zeroes(9 downto 0) & l0id_data_copyb2(109 downto 80)
                           & zeroes(9 downto 0) & l0id_data_copyb2(79 downto 50)
                                    & zeroes(9 downto 0) &
l0id_data_copyb2(49 downto 20)
                                    & zeroes(19 downto 0) &
l0id_data_copyb2(19 downto 0);
               WHEN "10" =>
      data_copyb <= zeroes & data_copyb3(1119 downto 840)
```

```
                                & zeroes & data_copyb3(839 downto 560)
                                                & zeroes & data_copyb3(559 downto 280)
                                                & zeroes & data_copyb3(279 downto 0);

        l0id_data_copyb <= zeroes(9 downto 0) & l0id_data_copyb3(109 downto 80)
                                & zeroes(9 downto 0) & l0id_data_copyb3(79 downto 50)
                                                & zeroes(19 downto 0) &
l0id_data_copyb3(49 downto 30)
                                                & zeroes(9 downto 0) &
l0id_data_copyb3(29 downto 0);
                    WHEN "11" =>
        data_copyb <= zeroes & data_copyb4(1119 downto 840)
                                & zeroes & data_copyb4(839 downto 560)
                                                & zeroes & data_copyb4(559 downto 280)
                                                & zeroes & data_copyb4(279 downto 0);

        l0id_data_copyb <= zeroes(9 downto 0) & l0id_data_copyb4(109 downto 80)
                                & zeroes(19 downto 0) & l0id_data_copyb4(79 downto 60)
                                                & zeroes(9 downto 0) &
l0id_data_copyb4(59 downto 30)
                                                & zeroes(9 downto 0) &
l0id_data_copyb4(29 downto 0);
                    WHEN OTHERS =>
        data_copyb <= zeroes & data_copyb1(1119 downto 840)
                                & zeroes & data_copyb1(839 downto 560)
                                                & zeroes & data_copyb1(559 downto 280)
                                                & zeroes & data_copyb1(279 downto 0);

        l0id_data_copyb <= zeroes(9 downto 0) & l0id_data_copyb2(109 downto 80)
                                & zeroes(9 downto 0) & l0id_data_copyb2(79 downto 50)
                                                & zeroes(9 downto 0) &
l0id_data_copyb2(49 downto 20)
                                                & zeroes(19 downto 0) &
l0id_data_copyb2(19 downto 0);
      END CASE;

                    IF rbuf_wplus = eventwidth4-2 THEN
                      copy_en <= '0'; -- disable read two cycles prior to write
                    ELSIF rbuf_wplus = eventwidth4-1 THEN
                      copystate <= idle;
                    ELSE
                      cbuf_raddr <= cbuf_raddr + 1; -- rotate the circular read address
                    END IF;

                    copy_en_v <= "1"; -- enable copy write one cycle after copy read

                    rbuf_waddr <= rbuf_wbase + rbuf_wplus; -- rotate write buffer
                    rbuf_wplus <= rbuf_wplus + 1;
                  END CASE;
          END IF;
  END PROCESS;

  -- sparse readout channel list build during circular to meb copy
  PROCESS( g_clk, reset ) IS
  BEGIN
```

```
 IF reset = '1' THEN
          sparse_data <= (OTHERS => (NOT zsup_setup));
                sparsestate <= idle;
          scopy_en_v <= "0";

        ELSIF rising_edge( g_clk ) THEN
                CASE sparsestate IS
                WHEN idle => -- wait for circular copy start
                  -- mark all channels as full of data, when no zsupp
                  -- or mark as empty, when zsupp on
                  sparse_data <= (OTHERS => (NOT zsup_setup));

    -- memory save disable
                  scopy_en_v <= "0";

                  IF copy_en = '1' THEN
                    sparsestate <= listmake;
                  ELSE
                    sparsestate <= idle;
                  END IF;

                WHEN listmake => -- data from circular valid, make comparisons
                  -- keep previous state if data <= zsup threshold
                  -- mark as non-empty, if over threshold
                  FOR i in 0 TO 95 LOOP -- data channels
                    IF data_copy(i*10+9 downto i*10) > zsup_thrsh THEN
                      sparse_data(i) <= '1';
                    END IF;
                  END LOOP;
                  FOR i in 0 TO 10 LOOP -- l0id channels
                    IF l0id_data_copy(i*10+9 downto i*10) /= "0000000000" THEN
                      sparse_data(i+96) <= '1';
                    END IF;
                  END LOOP;

                  IF copy_en = '0' THEN
                    sparsestate <= listdone;
                  ELSE
                    sparsestate <= listmake;
                  END IF;

                WHEN listdone =>
                  -- copy the mask to memory
                  scopy_en_v <= "1";
                  sparsestate <= idle;

   END CASE;
        END IF;
END PROCESS;

-- define the read address composition
-- the sub variables are modified by the read cycle
-- the formula remains the same
rbuf_raddr <= ( rbuf_rframe + rbuf_rplus4 + ( "0000000000" & ram_addr ))
                & ( rbuf_rframe + rbuf_rplus3 + ( "0000000000" & ram_addr ))
```

```
                                & ( rbuf_rframe + rbuf_rplus2 + ( "0000000000" & ram_addr ))
                            & ( rbuf_rframe + rbuf_rplus1 + ( "0000000000" & ram_addr ));

-- compose the L0 buffer read address
-- since write addressing is exactly similar to channel data, just the write
-- width is 8 times less, the read addressing is also +-the same, except of being
-- 8 times divided, ram addr is subtracted by 96 to have 0 on 0 l0id channel
l0id_rbuf_raddr <= ( rbuf_rframe(14 downto 3) + rbuf_rplus4(14 downto 3)
            + ( "0000000" & l0id_ram_addr ))
                & ( rbuf_rframe(14 downto 3) + rbuf_rplus3(14 downto 3)
                                    + ( "0000000" & l0id_ram_addr ))
                                & ( rbuf_rframe(14 downto 3) + rbuf_rplus2(14 downto 3)
                                + ( "0000000" & l0id_ram_addr ))
                            & ( rbuf_rframe(14 downto 3) + rbuf_rplus1(14 downto 3)
                                    + ( "0000000" & l0id_ram_addr ));

-- receive prepare data signal and provide the data to sender
-- 40 bits on each clock. Assert data_ready one clock cycle before data
-- and de-asserd with asserting of last data
PROCESS( rcu_clk, reset ) IS
BEGIN
  IF reset = '1' THEN
          prepstate <= idle;
    rbuf_rbase <= (OTHERS => '0');
    rbuf_rframe <= (OTHERS => '0');
    rbuf_rplus1 <= (OTHERS => '0');
    rbuf_rplus2 <= (OTHERS => '0');
    rbuf_rplus3 <= (OTHERS => '0');
    rbuf_rplus4 <= (OTHERS => '0');
                sbuf_raddr <= (OTHERS => '0');
                rdinc_del <= '0';

                channel_data <= (OTHERS => '0');
                data_ready <= '0';
          prep_en <= '0';
          sprep_en <= '0';
          counter <= (OTHERS => '0');
        ELSIF rising_edge( rcu_clk ) THEN
          -- increase event read pointer by one event on RCU read inc command
                IF rdinc = '1' AND rdinc_del = '0' THEN
                  rbuf_rbase <= rbuf_rbase + ( eventwidth4 AND meb_onoff_v );
                  sbuf_raddr <= sbuf_raddr + ( "000" & meb_onoff );
                END IF;

    -- keep current value
    rdinc_del <= rdinc;

          CASE prepstate IS
                WHEN idle =>
                  -- reset all
                  data_ready <= '0';
                        prep_en <= '0';
                        sprep_en <= '0';
                        counter <= (OTHERS => '0');
```

```
                IF prepare_data = '1' THEN
                        prep_en <= '1' AND (NOT is_sparse); -- enable memory read for data
                                sprep_en <= '1' AND is_sparse; -- enable memory read for sparse
request

        -- set the base addres according to event read pointer position
                        rbuf_rframe <= rbuf_rbase & "00000";

        -- set 320bit shifts according to which channel is being read out
                        -- write width 320bits, read width 10 bits -> one shift is 32 = 100000
                        CASE ram_id IS
                        WHEN "00" =>
                          rbuf_rplus1 <= "000000000000000"; -- no shift     32*0
                          rbuf_rplus2 <= "000000000100000"; -- one shift    32*1
                          rbuf_rplus3 <= "000000001000000"; -- two shifts   32*2
                          rbuf_rplus4 <= "000000001100000"; -- three shifts 32*3
                        WHEN "01" =>
                          rbuf_rplus1 <= "000000001100000"; -- three shifts
                          rbuf_rplus2 <= "000000000000000"; -- no shift
                          rbuf_rplus3 <= "000000000100000"; -- one shift
                          rbuf_rplus4 <= "000000001000000"; -- two shifts
                        WHEN "10" =>
                          rbuf_rplus1 <= "000000001000000"; -- two shifts
                          rbuf_rplus2 <= "000000001100000"; -- three shifts
                          rbuf_rplus3 <= "000000000000000"; -- no shift
                          rbuf_rplus4 <= "000000000100000"; -- one shift
                        WHEN "11" =>
                          rbuf_rplus1 <= "000000000100000"; -- one shift
                          rbuf_rplus2 <= "000000001000000"; -- two shifts
                          rbuf_rplus3 <= "000000001100000"; -- three shifts
                          rbuf_rplus4 <= "000000000000000"; -- no shift
                        WHEN OTHERS =>
                          rbuf_rplus1 <= "000000000000000"; -- no shift     32*0
                          rbuf_rplus2 <= "000000000100000"; -- one shift    32*1
                          rbuf_rplus3 <= "000000001000000"; -- two shifts   32*2
                          rbuf_rplus4 <= "000000001100000"; -- three shifts 32*3
        END CASE;

                          prepstate <= readstart;
                        END IF;

                WHEN readstart =>
                  data_ready <= '1'; -- signal data ready (one cycle prior to data)
                        sprep_en <= '0'; -- disable sparse mask read

        IF counter = eventwidth-1 THEN
                          prep_en <= '0'; -- disable memory readout
                        ELSE
          -- rotate the read buffer
                        rbuf_rframe <= rbuf_rframe + "000000010000000"; -- 32*4=128
                        END IF;

        counter <= counter + 1;
                        prepstate <= sending;
```

```
                WHEN sending =>
                  IF is_sparse = '1' THEN -- sparse channel mask request
                          IF counter = 4 THEN
                     data_ready <= '0'; -- signal data ready done (with the last data)
                                prepstate <= idle;
                              END IF;
                          counter <= counter + 1;
                  ELSIF counter < eventwidth-1 THEN
                          rbuf_rframe <= rbuf_rframe + "000000010000000"; -- 32*4=128
                          counter <= counter + 1;
                  ELSIF counter = eventwidth-1 THEN
                          prep_en <= '0'; -- disable ram readout
                          counter <= counter + 1;
        ELSE
                    data_ready <= '0'; -- signal data ready done (with the last data)
                              prepstate <= idle;
                         END IF;

        IF is_sparse = '1' THEN -- sparse channel mask
                          IF counter = 1 THEN -- first block
                                channel_data <= x"00" & sparse_data_prep(31 downto 0);
                              ELSIF counter = 2 THEN -- second block
                                channel_data <= x"00" & sparse_data_prep(63 downto 32);
                              ELSIF counter = 3 THEN -- third block
                                channel_data <= x"00" & sparse_data_prep(95 downto 64);
                              ELSE -- fourth block
                                channel_data <= x"0000000" & '0' & sparse_data_prep(106 downto
96);
                              END IF;
        ELSIF ram_addr < "11000" THEN -- channel data
                -- assign data in correct order
                          CASE ram_id IS
                          WHEN "00" =>
                            channel_data <= data_prep; -- no shift
                          WHEN "01" =>
                            channel_data <= data_prep(9 downto 0) & data_prep(39 downto 10); -- one
shift
                          WHEN "10" =>
                            channel_data <= data_prep(19 downto 0) & data_prep(39 downto 20); --
two shifts
                          WHEN "11" =>
                            channel_data <= data_prep(29 downto 0) & data_prep(39 downto 30); --
three shifts
                          WHEN OTHERS =>
                            channel_data <= data_prep; -- no shift
         END CASE;

                         ELSE -- L0 id data
                          CASE ram_id IS
                          WHEN "00" =>
                            channel_data <= l0id_data_prep; -- no shift
                          WHEN "01" =>
                            channel_data <= l0id_data_prep(9 downto 0) & l0id_data_prep(39 downto
10); -- one shift
                          WHEN "10" =>
```

```
                                channel_data <= l0id_data_prep(19 downto 0) & l0id_data_prep(39
downto 20); -- two shifts
                              WHEN "11" =>
                                channel_data <= l0id_data_prep(29 downto 0) & l0id_data_prep(39
downto 30); -- three shifts
                              WHEN OTHERS =>
                                channel_data <= l0id_data_prep; -- no shift
          END CASE;
                    END IF;

                  END CASE;
    END IF;
  END PROCESS;


  -- circular RAM buffer
  fake_altro_circular_inst : fake_altro_circular
    PORT MAP (
            clka            => g_clk,
            dina            => data_in,
            addra           => cbuf_waddr,
            wea             => "1",
            clkb            => g_clk,
            addrb           => cbuf_raddr,
            enb             => copy_en,
            doutb           => data_copy);

  memgen: FOR i IN 0 TO 3 GENERATE
    -- read buffer
    fake_altro_meb_inst : fake_altro_meb
      PORT MAP (
              clka          => g_clk,
              dina          => data_copyb((i+1)*320-1 downto i*320),
              addra                 => rbuf_waddr,
              wea                   => copy_en_v,
              clkb          => rcu_clk,
              addrb                 => rbuf_raddr((i+1)*15-1 downto i*15),
              enb                   => prep_en,
              doutb                 => data_prep((i+1)*10-1 downto i*10));
  END GENERATE memgen;

  -- L0ID circular RAM buffer
  fake_altro_l0id_circular_inst : fake_altro_l0id_circular
    PORT MAP (
            clka            => g_clk,
            dina            => l0id_data_in,
            addra           => cbuf_waddr,
            wea             => "1",
            clkb            => g_clk,
            addrb           => cbuf_raddr,
            enb             => copy_en,
            doutb           => l0id_data_copy);

  memgenl0id: FOR i IN 0 TO 3 GENERATE
   fake_altro_l0id_meb_inst : fake_altro_l0id_meb
```

```
    PORT MAP (
      clka                  => g_clk,
            dina            => l0id_data_copyb((i+1)*40-1 downto i*40),
      addra         => rbuf_waddr,
            wea             => copy_en_v,
            clkb            => rcu_clk,
            addrb           => l0id_rbuf_raddr((i+1)*12-1 downto i*12),
            enb             => prep_en,
            doutb           => l0id_data_prep((i+1)*10-1 downto i*10));
  END GENERATE memgenl0id;

  fake_altro_sparse_inst : fake_altro_sparse
  PORT MAP (
    clka            => g_clk,
    dina            => sparse_data,
    addra           => sbuf_waddr,
          wea               => scopy_en_v,
          clkb              => rcu_clk,
          addrb     => sbuf_raddr,
          enb               => sprep_en,
          doutb     => sparse_data_prep);

END ARCHITECTURE str21;
```

APPENDIKS E Utlesning fra Fake ALTRO buffer

```
  ****************************************
  *                                      *
  *       W E L C O M E  to  R O O T      *
  *                                      *
  *   Version  5.26/00b   9 February 2010   *
  *                                      *
  *  You are welcome to visit our Web site  *
  *          http://root.cern.ch          *
  *                                      *
  ****************************************

ROOT 5.26/00b (tags/v5-26-00b@32327, Aug 30 2010, 21:11:44 on
linuxx8664gcc)

CINT/ROOT C/C++ Interpreter version 5.17.00, Dec 21, 2008
Type ? for help. Commands must be C++ statements.
Enclose multiple statements between { }.

WELCOME to ALICE


Processing runAltroTest.C("")...
W-AliRawReaderFile::AliRawReaderFile: Can not read CDH header! The event
header fields will be empty!
enter nextBunchTimebin: 0
The 0 value is 0
The 1 value is 0
The 2 value is 0
The 3 value is 0
The 4 value is 0
The 5 value is 0
The 6 value is 0
The 7 value is 0
The 8 value is 0
The 9 value is 0
The 10 value is 3ce
The 11 value is 8c
The 12 value is 82
The 13 value is 0
enter nextBunchTimebin: 1
The 0 value is 0
The 1 value is 0
The 2 value is 0
The 3 value is 0
The 4 value is 0
The 5 value is 0
The 6 value is 0
The 7 value is 0
The 8 value is 0
The 9 value is 0
The 10 value is 3ce
The 11 value is 184
The 12 value is 82
The 13 value is 0
enter nextBunchTimebin: 2
The 0 value is 0
```

```
The 1 value is 0
The 2 value is 0
The 3 value is 0
The 4 value is 0
The 5 value is 0
The 6 value is 0
The 7 value is 0
The 8 value is 0
The 9 value is 0
The 10 value is 3ce
The 11 value is 8c
The 12 value is 82
The 13 value is 0
enter nextBunchTimebin: 3
The 0 value is 0
The 1 value is 0
The 2 value is 0
The 3 value is 0
The 4 value is 0
The 5 value is 0
The 6 value is 0
The 7 value is 0
The 8 value is 0
The 9 value is 0
The 10 value is 3ce
The 11 value is 184
The 12 value is 82
The 13 value is 80
enter nextBunchTimebin: 4
The 0 value is 0
The 1 value is 0
The 2 value is 0
The 3 value is 0
The 4 value is 0
The 5 value is 0
The 6 value is 0
The 7 value is 0
The 8 value is 0
The 9 value is 0
The 10 value is 3ce
The 11 value is 18c
The 12 value is c6
The 13 value is 0
enter nextBunchTimebin: 5
The 0 value is 0
The 1 value is 0
The 2 value is 0
The 3 value is 0
The 4 value is 0
The 5 value is 0
The 6 value is 0
The 7 value is 0
The 8 value is 0
The 9 value is 0
The 10 value is 3ce
The 11 value is 184
The 12 value is c6
The 13 value is 80
```

APPENDIKS F Forkortelser

ADC: Analogue-Digital Converter
ALICE: A Large Ion Collider Experiment
ALTRO: ALice Tpc Read-Out
APD: Avalanche Photo-Diode
ATLAS: A Toroidal Lhc ApparatuS
BC: Board Controller
CERN: Conseil Européenne pour la Recherche Nucléaire
CPLD: Complex Programmable Logic Device
CSP: Charge Sensitive Preamplifier
CTP: Central Trigger Processor
D-RORC: Data Read Out Receiver Card
DAQ: Data AcQuisition
DATE: Data Acquisition and Test Environment
DCS: Detector Control System
DDL: Detector Data Link
DIU: Destination Interface Unit
ECS: Experiment Control System
EMCal: Electro-Magnetic Calorimeter
FEC: Front End Card
FEE: Front-End Electronics. XIV, 15, 46
FPGA: Field Programmable Gate Array
GDC: Global Data Concentrator
GTL: Gunning Transceiver Logic
H-RORC: HLT Read Out Receiver Card
HLT: High Level Trigger
LDC: Local Data Concentrator
LED: Light Emitting Diode
LHC: Large Hadron Collider
LHCb: LHC-Beauty
LSB: Least Significant Bit
LTU: Local Trigger Unit
LVDS: Low-Voltage Differential Signaling
MEB: Multi-Event Buffer
MSB: Most Significant Bit
PDS: Permanent Data Storage
PHOS: PHOton Spectrometer
RAM: Random Access Memory
RCLK: Readout Clock
RCU: Read-out Control Unit
SCLK: Sampling Clock
SIU: Source Interface Unit
TDS: Transient Data Storage
TOR: Trigger-OR
TPC: Time Projection Chamber

TRU: Trigger Read-out Unit
TTC: Timing, Trigger and Control
VME: Virtual Machine Environment