Budapesti Műszaki és Gazdaságtudományi Egyetem
Villamosmérnöki és Informatikai Kar
Méréstechnika és Információs Rendszerek Tanszék

Monostori Balázs

FPGA INTERFÉSZ FEJLESZTÉSE HPTD DETEKTORHOZ

BELSŐ KONZULENS

Szántó Péter

KÜLSŐ KONZULENS

Dr. Dénes Ervin

BUDAPEST, 2012

Tartalomjegyzék

Kivonat **5**
Abstract **6**
1 Bevezetés **7**
1.1 CERN European Organization for Nuclear Research 7
1.2 ALICE kísérlet bemutatása 9
1.2.1 Részecskedetektorok 10
1.2.2 TPC detektor 10
1.2.3 HMPID és VHMPID detektorok 11
1.2.4 HPTD detektor 13
1.3 ALICE trigger és adatgyűjtő rendszere 16
1.3.1 Trigger rendszer bemutatása 16
1.3.2 Adatgyűjtő rendszer bemutatása 17
2 Tervezés FPGA-n **19**
2.1 Programozható logikákról általában 19
2.2 Actel ProAsic3E FPGA család 21
2.2.1 Konfigurálható Logikai Blokk 22
2.2.2 Nem felejtő FlashROM 23
2.2.3 SRAM és FIFO blokkok 23
2.2.4 CCC blokk 24
2.2.5 Általános Be-/Kimeneti Blokk (IOB) 25
2.2.6 Választott termék és fejlesztőkörnyezet 26
3 Front-end kártya ismertetése **27**
3.1 Az FPGA feladatai 28
3.2 Detektor oldali adatbemenet 29
3.3 DDL interfész 30
3.3.1 FEE-SIU adatvonalak és vezérlőjelek 30
3.3.2 Parancsszavak 33
3.3.3 Státusz szavak 34
3.3.4 Időzítések és a kommunikáció menete 35
3.4 Trigger bemenet 39
3.4.1 *L0* szintű trigger 39

3.4.2 *L1* szintű trigger 40
3.4.3 L2 szintű trigger 42
3.4.4 Trigger jelek szekvenciája 42
4 Rendszerterv megvalósítása FPGA-n 43
4.1 DDL interfész 44
4.2 Tömörítés 47
4.3 Detektor oldali adatbeolvasás 50
4.4 Memória 52
4.5 Trigger rendszer 58
4.6 Teljes rendszerterv 61
5 További feladatok 63
5.1 USB kommunikáció 63
5.2 Sugárzás elleni tesztelés 63
5.3 Mintakeresési feladat 65
6 Összefoglalás 66
Irodalomjegyzék 67
7 Függelék 69
7.1 Ábrák 69
7.2 CD melléklet tartalma 71

HALLGATÓI NYILATKOZAT

Alulírott **Monostori Balázs**, szigorló hallgató kijelentem, hogy ezt a szakdolgozatot meg nem engedett segítség nélkül, saját magam készítettem, csak a megadott forrásokat (szakirodalom, eszközök stb.) használtam fel. Minden olyan részt, melyet szó szerint, vagy azonos értelemben, de átfogalmazva más forrásból átvettem, egyértelműen, a forrás megadásával megjelöltem.

Hozzájárulok, hogy a jelen munkám alapadatait (szerző(k), cím, angol és magyar nyelvű tartalmi kivonat, készítés éve, konzulens(ek) neve) a BME VIK nyilvánosan hozzáférhető elektronikus formában, a munka teljes szövegét pedig az egyetem belső hálózatán keresztül (vagy hitelesített felhasználók számára) közzétegye. Kijelentem, hogy a benyújtott munka és annak elektronikus verziója megegyezik. Dékáni engedéllyel titkosított diplomatervek esetén a dolgozat szövege csak 3 év eltelte után válik hozzáférhetővé.

Kelt: Budapest, 2012. 12. 14.

...……

Monostori Balázs

Kivonat

Az MTA Wigner Fizikai Kutatóközpont Detektorfizikai Csoportja részt vesz a CERN ALICE kísérlet továbbfejlesztésében, a VHMPID és a HPTD detektorok tervezési és építési munkálataiban. A feladatom a fenti detektorrendszer front-end áramkörének firmware fejlesztése volt.

Az front-end elektronika alapvető funkcióit támogató digitális rendszert egy rugalmasan konfigurálható FPGA eszközre terveztem, és annak helyes működését szimulációkkal ellenőriztem. Az FPGA választás az Actel cég ProAsic3E A3P600 típusú eszközére esett, aminek szempontjai a dolgozatom részét képezik.

A választott eszköz memória kapacitása újabb problémákat vetett fel a tervezés során: a szűk memória kapacitás nem teszi lehetővé a kívánt mennyiségű adat egyszerű, nyers letárolását. A dolgozatban javaslatot teszek egy memória-tömörítési eljárásra és egy azt megvalósító, lehetséges memória-szervezési megoldásra.

Munkám során megismerkedtem az ALICE kísérlet adatgyűjtő és trigger rendszerével, a VHMPID és HPTD detektorok működésével a front-end elektronika funkcióinak alaposabb megértése érdekében.

A digitális rendszert egy, még csak körvonalaiban megtervezett kártyára terveztem. Azonban az FPGA-n lévő digitális rendszer párhuzamosan fejleszthető az eszközt tartalmazó hardver tervezési munkálataival. Az általam tervezett, szimulációkkal ellenőrzött működésű rendszer, a feladat kiírásban szereplő modulokon kívül, egy javaslat is a front-end elektronika feladatait ellátó, egész rendszer megvalósítására.

Abstract

Wigner Research Centre for Physics (Wigner RCP) is participating in the development of CERN ALICE experiment as well as in the improvement of VHMPID and HPTD particle detectors. My main task was to develop the firmware of the detector's font-end electronic card (FEE).

I designed a digital system which is implementable on a configurable FPGA. The design supplies the main functions of FEE and I validated the proper behavior with simulations. The Actel ProAsic3E A3P600 device was chosen for the design. The reason for the choice is given my thesis.

The memory capacity of the chosen device generated new problems: the small capacity of the memory does not allow to store the required amount of data in a simple way. In the current paper I recommend not only a method for a memory compression and but also a solution for a memory organization.

During my work I got to know the data acquisition and trigger system of ALICE experiment, VHMPID and HPTD detectors to understand the functions of FEE.

I developed a digital system design to a non-existing FEE. Nevertheless the developing of the digital system of the FPGA is possible in paralell with the FEE hardware development. In addition to my original tasks the design is a potential realization of the whole system.

1 Bevezetés

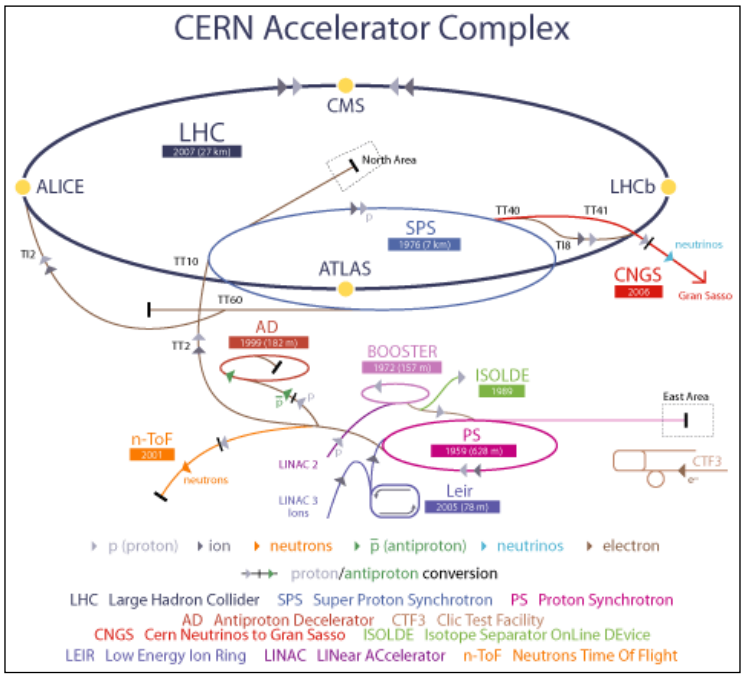
1.1 CERN European Organization for Nuclear Research

A CERN az Európai Nukleáris Kutatási Szervezet (a név a francia Conseil Européen pour la Recherche Nucléaire szóösszetételből ered), a részecskefizikai kutatások európai szervezete, jelenleg a világ legnagyobb részecskefizikai laboratóriuma, melyben immár 20 tagország folytat kutatásokat [1]. 1954-ben 12 ország együttműködésével alapították, a részecskefizikai kutatások központosítása céljából. 1992 óta Magyarország is teljes tagként vesz részt a fejlesztési és kutatási munkálatokban [2].

Már a hivatalos megalapítás előtt, 1952-ben az ideiglenes CERN bizottság elhatározta, hogy az új laboratóriumban két gyorsítóra is szükség lesz az atommag-kutatások során. A kezdeti sikerek óta, mára már a francia-svájci határon fekvő, Genf város közelében lévő kutatóközpontban hat fő gyorsító üzemel [3]. A gyorsító-rendszer egyre nagyobb energiaszintű gyorsítókból épül fel. A különböző energiaszintű gyorsító gyűrűk mentén különböző kísérletek állomásai találhatóak (1.1 ábra). A kisebb gyorsítók a saját energiaszintjükön végezhető kísérleteken kívül, a nagyobb gyorsítók előgyorsítóiként is szolgálnak. A részecskenyalábokat (legyenek azok elektronok, protonok, nehéz ionok…) elektromágneses tér segítségével tartják a gyorsításhoz és az ütközéshez megfelelő pályán.

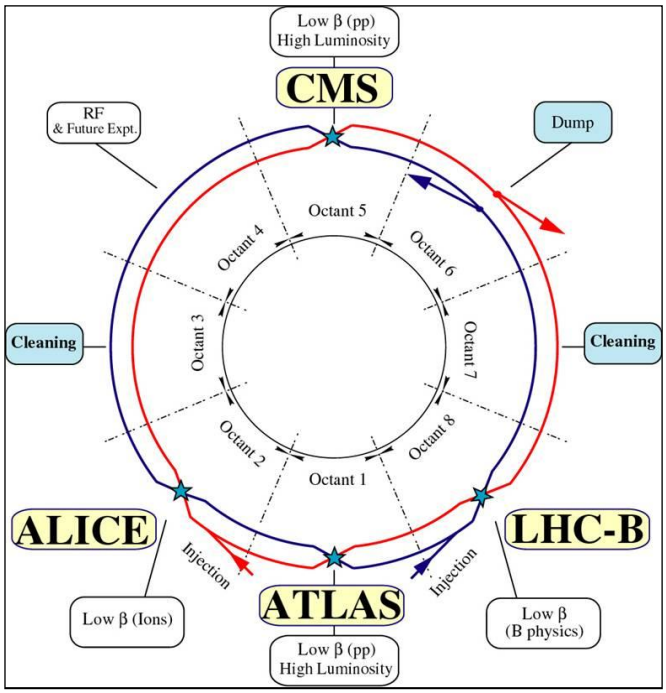
Abban az esetben, ha protonok ütköztetésén alapuló kísérletek során kialakult eseményeket szeretnének megfigyelni, hidrogén atomokat ionizálva, protonokat állítanak elő. A protonok a Linac2 lineáris gyorsító segítségével 50MeV-ra gyorsulnak fel. A következő segédgyorsítóban (proton szinkrotron, röviden PS) a fény sebességének 99.9%-ra gyorsulnak (25GeV). Majd a részecskék energiaszintje még egy nagyságrendet lép (450GeV) a szuper proton szinkrotron (SPS) gyorsító gyűrűben megtett útja során. Ezután kerülnek az LHC (*Large Hadron Collider*, Nagy Hadron Ütköztető) gyűrűjébe, mely 27 km hosszú kerültével a legnagyobb gyorsító gyűrű a világon. Itt akár 4 TeV energiát is elérnek a csomagok. (Ez ütközéskor $2 * 4 = 8\,TeV$ energiát jelent.)

Nehéz ion ütköztetés esetén a Linac3 lineáris gyorsítóból 4.2 MeV energiájú, ólomionok lépnek ki, amelyek utána, a protonokéval azonos útvonalon érik el a nagy LHC gyűrűt.



1.1 ábra: CERN gyorsító-rendszer felépítése [3]

Az LHC gyűrűben két ellentétes pályán haladnak a részecske csomagok. A két pálya a gyűrű mentén négy pontban keresztezi egymást (1.2 ábra).



1.2 ábra: LHC gyűrű két, ellentétes irányú pályája [4]

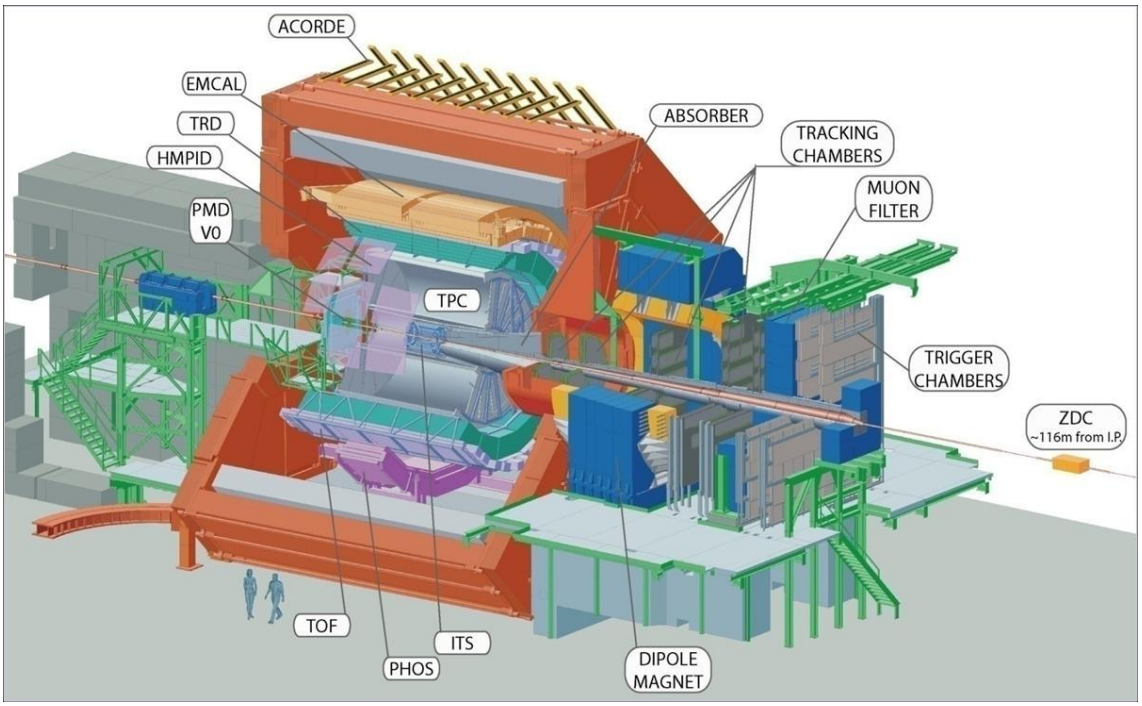
Ezen négy pontban kaptak helyet a nagyobb kísérletek [5]:

- ATLAS (A Toroidal Lhc Apparatus) és CMS (Compact Muon Solenoid,)
    - Standard-modell ellenőrzése, Higgs-bozon keresése, szuperszimmetria és extradimenziók keresése.
- LHCb (Large Hadron Collider-beauty)
    - Standard-modell precíziós mérése, bájos-kvarkok (beauty kvarkok) mérése.
- ALICE (A Large Ion Collider Experiment)
    - Nehéz ion és proton ütközések során keletkező kvarkok és a közöttük lévő kölcsönhatást közvetítő gluonok vizsgálata. Kvark gluon plazma előállítása és vizsgálata.

1.2 ALICE kísérlet bemutatása

Az ALICE (Nagy Nehézion Ütköztető Kísérlet) kísérlet Eiffel-toronyénál is nagyobb tömegével egyike a legnagyobb LHC menti kísérleti állomásoknak. A 10000 tonna tömegű, 16 m magas és 26 m hosszú mérőrendszer kiépítésén 29 ország mérnökei és fizikusai dolgoznak. Céljuk a kvark-gluon plazma előállítása és vizsgálata, aminek segítségével pontosabb képet kaphatunk az univerzum korai állapotáról. Körülbelül az univerzum keletkezése utáni, első száz mikroszekundum vizsgálatát segíti a kvark-gluon plazma vizsgálata. A plazmát a kísérlet során közel fénysebességre gyorsított ólom atommagok ütköztetésével hozzák létre.

Az ALICE rendszerében számos detektor foglal helyet (1.3 ábra). A detektorok és működésük részletes bemutatása nem célja a dolgozatomnak, azonban egyes detektorok szerepét és alapvető működését szükséges ismertetnem a fejlesztés érthetősége érdekében.



1.3 ábra ALICE detektor felépítése [6]

1.2.1 Részecskedetektorok

Részecskefizikai detektorok közül több féle típust különböztetünk meg. Léteznek félvezető technikán alapuló, gáztöltésű és szcintillációs elven működő detektorok [7]. A szilícium alapú detektorok kisebb méretű, gyors, jó felbontású detektorok. A gáztöltésű detektorok ezzel szemben viszonylag nagyobb méretűek, lassabbak és gyengébb felbontást szolgáltatnak, áruk azonban töredéke a félvezető technológián alapuló detektorokénak.

1.2.2 TPC detektor

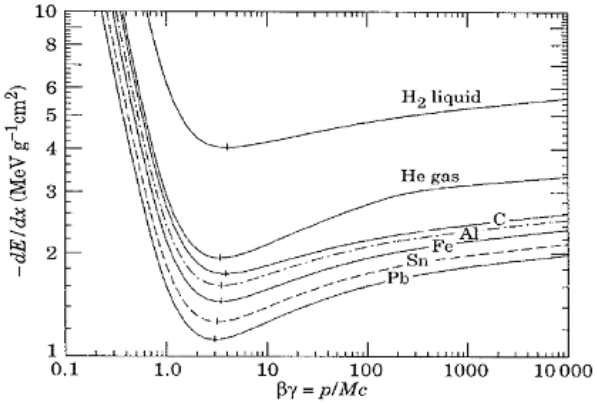
Az ALICE egyik fő detektor-rendszere a TPC (Time Projection Chamber, Időprojekciós Kamra) detektor, a gáztöltésű detektorok osztályába sorolható. A kamrán áthaladó részecskék a kamrába zárt gázt ionizálják, így egy háromdimenziós képet kaphatunk a részecske pályájáról. A cél a részecske azonosítása, ami a detektortérben lévő statikus mágneses tér által a részecske pályáján okozott görbületből és a leadott energiájából meghatározható. A kamra a 3 GeV energia alatti részecskékre jól működik, ezen energiatartomány feletti részecskék megkülönböztetésére azonban nem alkalmas.

A detektorban a töltött részecskék a körülöttük lévő gázmolekulákat ionizálják. Az ionizálás során szabad elektronok keletkeznek, melyek a kamrában lévő statikus elektromos tér hatására sodródni kezdenek. A kamra szélén egy érzékelő réteg,

úgynevezett sokszálas proporcionális kamra (*MWPC - Multi-wire Proportional* Chamber) helyezkedik el. Ezen a kamrán, az egyes elektronok becsapódása során elektron lavina alakul ki, amely már mérhető mennyiségű elektromos jelet kelt. A kamrát kiolvasva egy kétdimenziós képet kapunk az elektronok becsapódási helyéről. Ha ezt a kétdimenziós képet folyamatos időközönként leolvassuk, egy háromdimenziós képet kaphatunk a részecske pályájáról. Az ionizáció során keletkezett elektronok sodródási sebessége függ a kamrában lévő statikus elektromos tér nagyságától, a gáz minőségétől és típusától, illetve a detektor geometriájától is. Az ALICE kísérletben szereplő TPC detektor paraméterei mellett a kamra teljes sodródás ideje 88 μs [9], ami egyben megfelel a TPC detektor minimális kiolvasási idejének is.

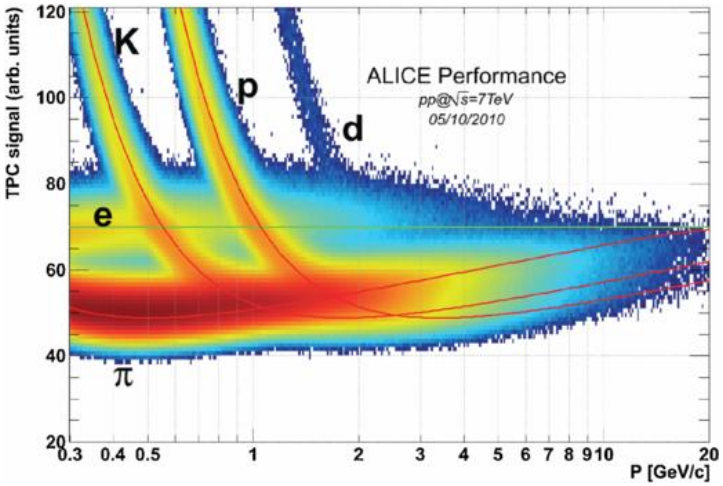
1.2.3 HMPID és VHMPID detektorok

A Bethe-Bloch formula [11] szerint a töltött részecskék ionizáló hatása a következő ábra szerint alakul:



1.4 ábra: Bethe-Bloch görbék [10]

Az ábra a töltött részecske leadott energiáját, a sebességének függvényében ábrázolja. Látható, hogy a Bethe-Bloch görbéknek, egy adott részecskesebesség esetén globális minimuma van. Ezt a pontot nevezzük MIP-nek [12] (*Minimum Ionizing Particle – minimálisan ionizáló részecske*). A MIP fogalma az ionizáló hatás mérésén alapuló részecske detektorok egyfajta mérési pontosságát is igyekszik meghatározni. A TPC detektor a görbe első részén (körülbelül 2-3 GeV-ig), viszonylag pontosan képes megkülönböztetni egymástól a különböző töltött részecskéket (1.5 ábra).



1.5 ábra: TPC detektor felbontása, (Bethe-Bloch görbék) [13]

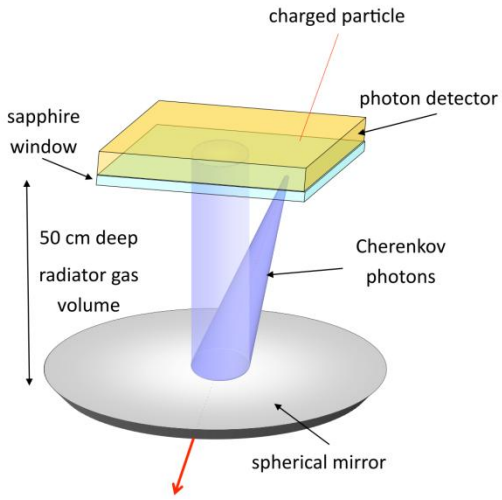
3 GeV feletti energiatartományra azonban pontatlan mérést produkál. Az ALICE kísérlet tervezése óta az új kutatási eredmények rávilágítottak, hogy az eredetileg tervezett 3 GeV-nál nagyobb energiás részecskék vizsgálata is igen fontos. Így az alapkoncepció kiegészült a HMPID detektorral (High Momentum Particle Identication Detector, Nagy impulzusú részecskéket azonosító detektor), mely a hadronokat a 2-5 GeV tartományon képes megkülönböztetni egymástól. A még nagyobb energiás részecskék (5-25 GeV) azonosítását a jelenleg kutatási/fejlesztési fázisban lévő VHMPID (Very High Momentum Particle Identication Detector, Nagyon nagy impulzusú részecskéket azonosító detektor) végzi majd a jövőben.

Mindkét fenti detektor a Cserenkov sugárzás jelenségén alapul. Az effektus lényege röviden a következő: hasonlóan ahhoz, ahogy egy levegőben haladó test a közeghez tartozó hangsebességnél gyorsabban haladva hangrobbanást okoz, úgy ha egy szigetelőben, a közegbeli fénysebességnél nagyobb sebességgel halad egy töltött részecske, akkor elektromágneses sugárzást bocsát ki kúp alakban [12]. Ezen kibocsátott sugárzás, a részecske pályájához viszonyított szöge a részecske sebességétől illetve az anyag törésmutatójától függ:

$$\cos\vartheta = \frac{c'}{v} = \frac{\frac{c}{n}}{v},$$

ahol c a vákuumbeli fénysebesség, n a közeg törésmutatója, ϑ a kúp szöge (továbbiakban Cserenkov szög), c' a közegbeli fénysebesség, v pedig a részecske sebessége. A keletkező fotonok száma a törésmutató függvénye, s körülbelül $(n-1)$-el arányos.

A részecske sebességének meghatározásához elegendő tehát, a Cserenkov szöget kell megmérni. A HMPID esetén a folyadék közegben keletkező sugárzás, egy távolabb lévő felületen gyűrűt rajzol ki. Az egyedi fotonok detektálása után, a gyűrű mért sugarából a kúpszög, s így a sebesség számolható. Nagyobb energiás részecskék esetén (amelyeket például a VHMPID detektor is mér) kis törésmutatójú anyagot kell használni, tipikusan gázt. A kis törésmutató miatt a keletkező fotonok száma alacsony, így hosszabb Cserenkov út szükséges, ami a VHMPID esetén 40-100 cm. A Cserenkov közeg végén körlapra érkező fotonokat gömb-, vagy parabolatükörrel visszatükrözve itt is gyűrűvé fókuszálhatjuk (1.6 ábra).



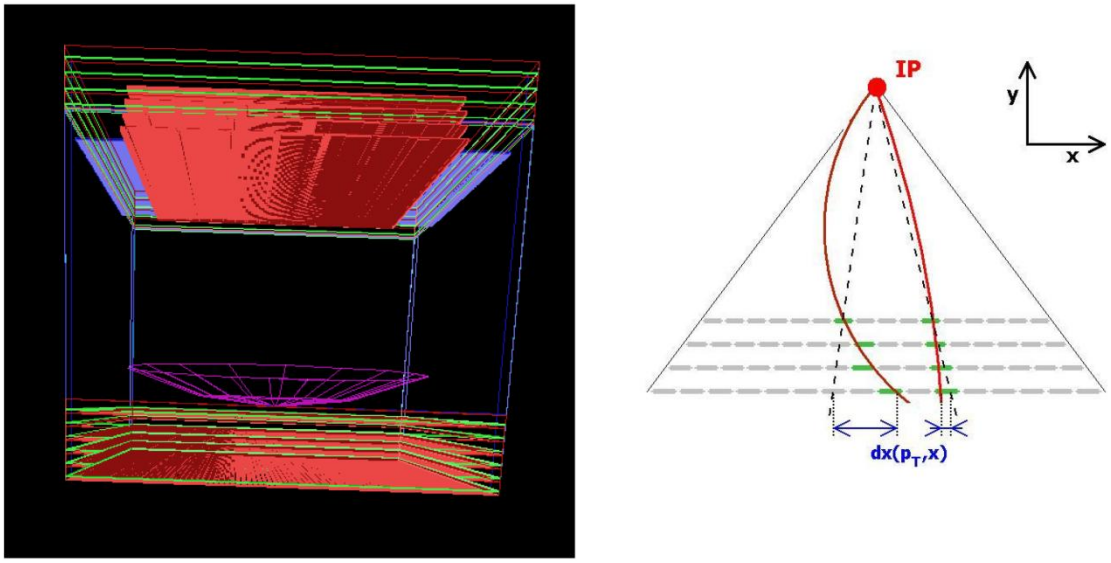
1.6 ábra: VHMPID detektor fókuszálása [15]

A VHMPID detektor egy ilyen módosított, gyűrűs képalkotású Cserenkov kamrából áll, amit egy trigger detektor is segít [14],[15].

1.2.4 HPTD detektor

A VHMPID kamra mindkét oldalához viszonylag közel, a HPTD (*High P_t Trigger Detector – nagy transzverzális impulzusú trigger detektor*) detektor két-két kamrája foglal helyet. Én erre az aktuális, 2+2 rétegű elrendezésre terveztem, a 1.7 ábra azonban egy 4+6 réteg elrendezést ábrázol, mely egy korábbi szimulációs elrendezés ábrája. A HPTD detektor egy rétege, egy úgynevezett közelkatódos elrendezésű kamrából áll (*Close Cathode Chamber, CCC*) [16]. Két katód lemez között található egy szálsík, ahol pozitív feszültségű vékony anódszálak helyezkednek el egymástól azonos távolságra. A kamrák gázzal vannak töltve. A gáztérbe csapódott részecske ionizálja a gáztérben lévő atomokat, aminek során elektronok szabadulnak fel. A pozitív anódszálak által keltett

elektromos térerősség gyorsítja ezen elektronokat a szálak felé. A mozgás során újabb atomokat ionizálnak és ez által újabb elektronok keletkeznek. Ez a szálak közelében még nagyobb hatásfokkal megy végbe, hiszen ott a legnagyobb a térerősség. Ezzel az elektron-lavinával 10^4-10^6 erősítést lehet elérni. A pozitív anódszálak között félúton, egy-egy negatív feszültségű szál is helyet kapott a jobb térkonfiguráció kialakítása végett. A szálsík aszimmetrikusan helyezkedik el a katódlemezek között (a két katódlemez különböző feszültségen üzemel). A közelebbi katód lemez általában földelt és 4 mm-enként szegmentált. Ezeket a szegmenseket „pad”-eknek nevezik. Az egyes pad-ek a szálak alatt helyezkednek el, így a kapacitív csatolás miatt a szálakon megjelenő jel a pad-ekről is leolvasható.



1.7 ábra: HPTD kamrák a VHMPID detektorban [14] (az ábra csak illusztráció)

Egy HPTD kamra 4 mm x 100 mm méretű pad-ekre van felbontva. Egy négyzetméternyi felületen ez 10 sort és soronként 250 pad-et jelent. A pad-ek száma a távolabbi rétegekben azonos, azonban nagyobb maga a kamra és így a pad-ek mérete is (1.7 ábra jobb oldali ábrája, az ábra csak illusztráció). A detektor kiolvasása által egy kétdimenziós képet kapunk a részecske becsapódásának helyéről. A pontosabb felbontás érdekében a VHMPID két felén lévő HPTD kamra párok kamrái, egymáshoz képest 90°-ban el vannak forgatva.

A HPTD detektor három fontosabb funkciót tölt be az ALICE rendszerében. A legfontosabb feladata, a VHMPID MIP detektálási funkciójának segítése az offline kiértékelések során. Ugyanis a VHMPID detektoron áthaladó részecskék

„származásáról”, egyedül a VHMPID detektor két oldalához közel elhelyezkedő HPTD detektor kamrái által bizonyosodhatunk meg. Azaz, hogy a detektált részecske, valóban átment a VHMPID detektoron és valóban az ütközésből származó, TPC által már detektált, nagy impulzusú részecskéről és nem egy sokadig állványzatról visszapattant, „kósza” részecskéről van szó. A négy HPTD kamrán megjelent koordináták pontosítják a részecske pályáját, ami nagyban segíti a Cserenkov kör azonosítását is, és így a kör sugarának mérését az offline adatfeldolgozás során. Ezek elengedhetetlen funkciók a VHMPID hatékony működéséhez. Ezen kívül egyfajta proton-detektálásra is képes, mivel a nehezebb proton részecske (5-9 GeV energiatartományon) nem kelt Cserenkov effektust a VHMPID detektorban. Így a TPC által és a HPTD által azonosított, megfelelően nagy impulzusú részecske, mely Cserenkov kör keltése nélkül halad át a VHMPID detektoron, protonként azonosítható.

A fent említett funkciók, az offline, utólagos adatfeldolgozást segítik. A HPTD detektornak ezen kívül, (nevéből is adódóan) trigger, online feladatai is vannak. Az ALICE kísérletben elsősorban nehézionok ütköznek. Másodpercenként akár 8000 ilyen esemény történik. Ólom atommagok ütköztetése esetén, az egy ütközés során tárolandó adat mennyisége nagyjából 90 MB méretű. Ekkora mennyiségű adat letárolásának technológiai korlátai vannak, amik miatt maximum 1.5 GB-ot vagyunk képesek másodpercenként letárolni. 8000 eseményből így összesen 20 darabot tarthatunk meg, amiknek kiválasztása nagy jelentőséggel bír. Ezért igen fontos az egyes detektorok triggerelése, mivel annak segítségével a számunkra fontos események adatait tudjuk rögzíteni. A TPC gáztöltésű detektor túl lassú ahhoz, hogy időben tudassa a VHMPID detektorral egy adott, nagyimpulzusú részecske áthaladását (korábban említett $88\ \mu s$ miatt (1.2.2 fejezet)). A VHMPID-nek saját trigger rendszerre van így szüksége, mely a megfelelő részecske áthaladása után $10\ \mu s$-on belül jelezni képes. A HPTD detektor elegendően gyors ahhoz, hogy *L1* szintű (lásd később a 1.3.1 fejezetben) triggerrel lássa el a VHMPID detektort. A HPTD detektor, négy rétegének köszönhetően, a részecske pálya görbületéből, impulzus mérésére is alkalmas. Ez a mérés igen pontatlan, de az említett ólom atommagok ütköztetése során keletkező rengeteg részecske válogatásában nagy jelentőséggel bír.

Protonütköztetés esetén kevesebb részecske keletkezik, így kevesebb a tárolandó adat is. Ebben az esetben olyan kevés részecskéről van szó, hogy a HPTD detektor *L0*

szintű trigger jelére (lásd 3.4 fejezet) is szükség van. Ez a funkciója kevésbé lényeges, de felhasználható.

1.3 ALICE trigger és adatgyűjtő rendszere

Ahogy az előző, 1.2.4 fejezetben már említettem, minden egyes esemény rögzítése igen magas (700 GB/s-os) adatmentési sebességet követelne meg, amely a nagy adatátviteli kapacitás ellenére is túl gyors ahhoz, hogy teljes mértékben lekezelhető lehessen. Így az adatgyűjtő rendszer kifejlesztésével párhuzamosan, egy trigger rendszert is kifejlesztettek, mely valamilyen szempontból értékesnek talált események szelektálására szolgál. A trigger rendszernek rugalmasnak kellett lennie a folyton változó új trigger detektorok lekezelésére és az adatgyűjtő rendszer hosszú távú támogatására.

1.3.1 Trigger rendszer bemutatása

Minden alrendszer közelében egy hierarchikus felépítésű trigger rendszer található, mely a következő egységekből épül fel:

Központi Trigger Feldolgozó Egység (*Central Trigger Processor (CTP)*) – trigger jelek fogadása és a jóváhagyott trigger jelek LTU felé való továbbítása.

Trigger, Időzítés és Órajel Egység (*Trigger, Timing and Clock (TTC)*) – A egyes trigger üzenetek front-end elektronika (*FEE*) felé való továbbítása optikai adatátviteli csatornán.

Helyi Trigger Egység (*Local Trigger Unit (LTU)*) – *L0* szintű trigger kezelése és továbbítása a front-end elektronika felé, valamint az FEE felől érkező *Busy (foglalt)* jel lekezelése.

A trigger rendszer kialakítása során arra is ügyelni kellett, hogy a kísérletenként változó detektorműködésre és különböző kiolvasási sebességgel rendelkező detekrotokra is univerzális legyen. Így többszintű trigger rendszert alakítottak ki, a következő időzítésekkel:

L0 trigger: A késleltetés kisebb, mint *800 ns*. A fizikai réteg csavart érpár, az üzenet formája impulzus.

L1 trigger: A késleltetés kisebb, mint *6,1 μs*. Optikai linken, a részecske csomagok azonosítását is tartalmazza.

L2 trigger: Tipikus késleltetése *87,6 μs*. Optikai linken kódolt információt is tartalmaz.

Signal Status	L0 (μs)	L1 (μs)	L2(μs)
Last trigger input at CTP	0.8	6.1	87.6
Trigger output at CTP	0.9	6.2	87.7
Trigger output at detector	1.2	6.5	88.0

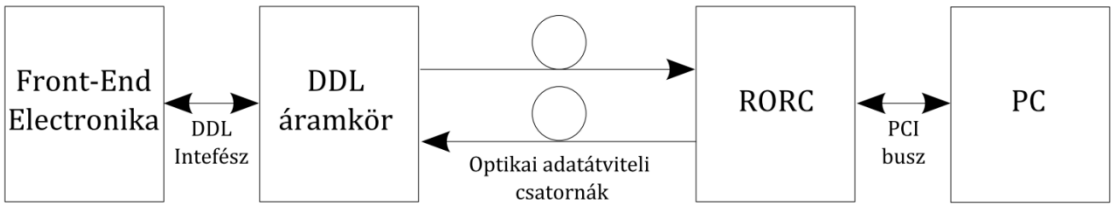
1.8 ábra: Trigger késleltetések [25]

Az időpontok a részecske csomagok ütközésének időpontjától számolódnak. A trigger detektorok által generált trigger jelek először a központi CTP egységbe futnak, mely jóváhagyásuk után, az egyes detektorok felé továbbítja a jeleket. A front-end elektronikához érkező jelek, az 1.8 ábra táblázatának harmadik sora szerinti késleltetéssel jelennek meg.

Ezen kívül létezik egy globális LHC szintű trigger. Az LHC gyűrűben az egyes részecske csomagok *40.08 MHz* frekvenciával keringenek. Ez a globális órajel az egész gyűrű mentén, optikai kábelen, minden részegység számára elérhető.

1.3.2 Adatgyűjtő rendszer bemutatása

Az adatgyűjtő rendszer felel az adatok továbbításáért, az azokat rögzítő számítógépek felé. Az 1.9 ábra vázolja az adatgyűjtő rendszer fontosabb részegységeit [18].



1.9 ábra: Az adatgyűjtő rendszer fontosabb részegységei [8]

A közvetlen a detektor közelében elhelyezkedő front-end elektronika, megfelelő trigger-feltételek teljesülése mellett, leolvassa a detektorról az adatokat és DDL (*Detector Data Link – detektor adat link (lásd később: 3.3 fejezet)*) interfészen keresztül továbbítja az adatokat az azokat feldolgozó DDL áramkör felé. A DDL áramkör miután csomagokba szervezte az adatokat, optikai kábelen továbbítja azt a következő fokozat RORC (*Read-Out Reciever Card – kiolvasó kártya (lásd később: 3.3 fejezet*) irányába. A két egység

közötti kommunikáció soros. Az RORC áramkör *PCI*[1] buszon keresztül csatlakozik egy személyi számítógép buszrendszeréhez. A fogadott adatok *DMA*[2] rendszeren keresztül, ideiglenesen tárolódnak az archiváló rendszer felé való továbbítás előtt.

[1] PCI - Peripheral Component Interconnect - a cpu és a perifériák összekötésére szolgáló processzorfüggetlen adatbusz

[2] DMA - Direct Memory Access – processzor független, direkt memória hozzáférés

2 Tervezés FPGA-n

Ebben a fejezetben először az FPGA (*Field-Programmable Gate Array – a felhasználás helyén programozható logikai kapumátrix)* eszközök tulajdonságait általánosan ismertetem, majd az általam használt Actel ProAsic3 FPGA családot mutatom be részletesebben.

2.1 Programozható logikákról általában

A digitális technika fejlődése során, több irány is kialakult digitális rendszerek tervezésére, fejlesztésére. Ezen irányok közül általában az adott időben meglévő technológia szint tette/teszi népszerűvé, vagy kevésbé alkalmassá az adott eszköztípus használatát. A programozható logikák a komplex PLD-k kialakulásával kerültek előtérbe. A mai technológiák mellett, az FPGA eszközök manapság is egyre népszerűbbek. Kis darabszám esetén, az ASIC, standard cellás tervezéshez képest sokkal költséghatékonyabb és rugalmasabb digitális tervet tudunk készíteni, mely később is újrakonfigurálható. (Léteznek egyszer írható FPGA-k is, de általában többször írhatókat használunk.). A felhasznált erőforrások pedig jobban illeszkednek az adott feladathoz, mint a processzoros alkalmazások, ami a kis fogyasztásra, a nagyobb sebességre való optimalizálásban és a párhuzamosíthatóságban segít, az időigényesebb tervezésért cserében. Használatuk, tehát vagy a fogyasztásra való optimalizálásban vagy pedig valamilyen speciális alkalmazás miatt indokolt (speciális teljesítmény igényű feladatok ellátása, párhuzamos architektúrák, elosztott aritmetikák megvalósítása…). Illetve ASIC tervezés esetén is nagy segítség egy programozható logikán való prototípus tesztelése. A jelen dolgozat tervében is hasonló okok domináltak: jól időzített, nem szabványos interfész kapcsolatok kiépítése, gyors, de rugalmasan változtatható adatbeolvasás és letárolás, nagy ki-/bemeneti kapacitás, illetve mintakeresési feladatok megoldása.

Az FPGA, programozható logikák és azok programozható huzalozásának általában valamely mátrixba rendezett halmaza [23]. Így bármilyen digitális rendszert felépíthetünk programozásukkal. Az egyes eszközök közötti különbségek az erőforrások mennyiségén kívül, a részegységek felépítéséből adódnak. Felépítésük általában konfigurálható logikai blokkokból (CLB - Configurable Logic Block), be- és kimeneti

illesztőegységekből (IOB – Input Output Block, be- és kimeneti blokk), memória elemekből, órajel kezelő egységekből és esetleg szorzókból vagy egyéb jelfeldolgozást segítő egységekből épülnek fel (mint például DSP48 a Xilinx cég eszközeinek esetében). A programozhatóság szempontjából három nagyobb csoportra oszthatjuk az eszközöket:

- SRAM alapú eszközök: Ez a legelterjedtebb típus az olcsó technológia (CMOS) és a rugalmas átprogramozhatóság miatt. Minden információ ugyanis memória cellákban van eltárolva, így azok akár futás alatt is beolvashatóak, átírhatóak. A programmemóriájuk felejtő, azaz csak tápfeszültség mellett őrzik meg azt, így használatuk előtt gyors újrakonfigurálás szükséges. Nagy komplexitás és sebesség érhető el velük, azonban a fogyasztásuk is nagyobb a többi típusnál. Sajnos kevésbé is biztonságosak, mivel érzékenyek a sugárzásra.
- Antifuse/OTP (One Time Programmable) eszközök: Az ilyen típusú eszközök csak egyszer programozhatóak. Előnyük a kis méret, kisebb fogyasztás és nagyobb biztonság.
- Flash alapú eszközök: Ezekben a típusokban flash alapú memóriába töltjük a programunkat, mely nem felejtő, azaz tápfeszültség nélkül is megőrzi azt. Ennek átírása esetén azonban először törölni kell a memóriát. Főbb előnye a biztonság (sugárzásra kevésbé érzékeny) és a kis fogyasztás. Az általam választott termékcsalád is ebbe a kategóriába sorolható.

Hordozható elektronikákban (mobiltelefon, mp3, pda), intelligens szenzorokban, kamerákban alacsony fogyasztásuk miatt az utóbbi két típust szokták alkalmazni.

Egy FPGA általános felépítése egyszerű: általános célú logikai elemekből, programozható huzalozásból és kivezetésekből áll. A mai FPGA-k azonban az univerzális, általános logikai elemeket is több bonyolultsági szintre bontják, így kisebb és nagyobb komplexitású logikai elemekből építkezhetünk, ami a feladathoz jobban illeszkedő erőforrás használatot eredményez.

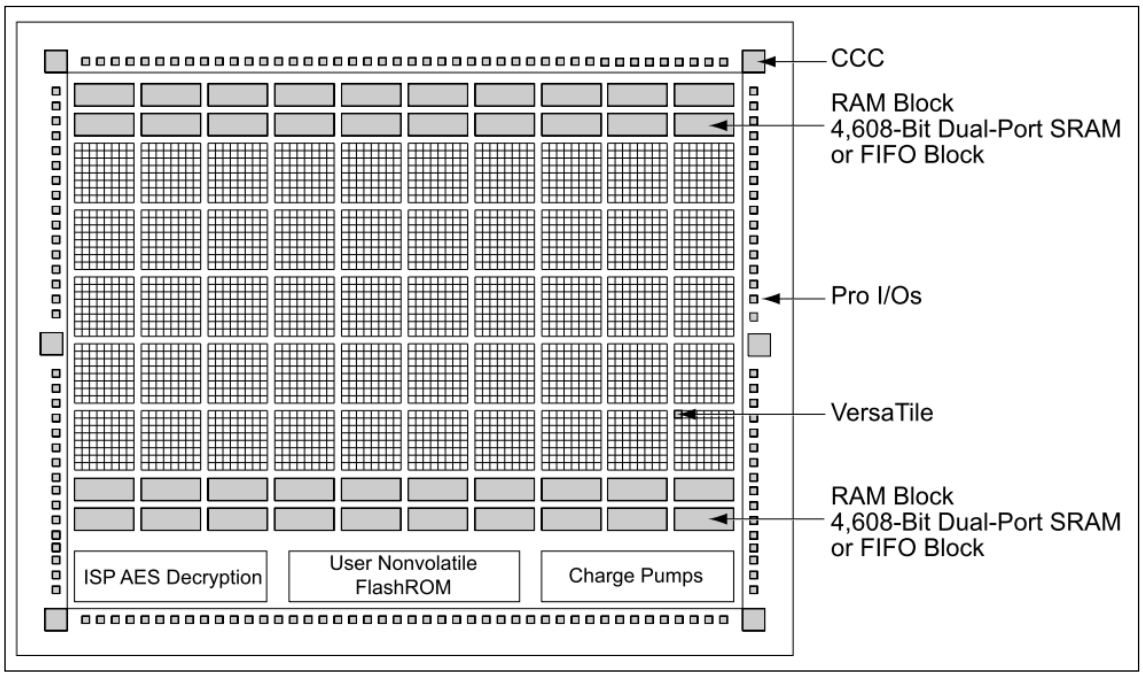
2.2 Actel ProAsic3E FPGA család

Ebben az alfejezetben a tervhez választott eszközcsaládot mutatom be, a tervezéshez szükséges tulajdonságai alapján. A ProAsic3 egy flash alapú FPGA család az Actel gyártótól. A flash alapú technológiára a jobb sugárzás-tűrése miatt esett a választás. A front-end kártya közvetlen a detektor térben helyezkedik majd el, ahol mindenképpen sugárzásnak lesz kitéve. A családon belül kifejezetten katonai és űrtechnológiai célokra is készültek eszközök. Így például egy sugárzás tűrő eszköz-csoport is (RT – Radiation Tolerant ProAsic3). Mi azonban egy előtanulmányt ([24]) alapul véve egy egyszerű ProAsic3E chipet tervezünk beépíteni. A tanulmány egy, a debreceni atommagkutató intézetben végzett vizsgálatot dokumentál, melynek során sugárzó térbe helyeztek, shift-regiszterként és memóriaként konfigurált különböző FPGA-kat, melyek közül az egyetlen flash alapú Actel ProAsic bizonyult alkalmasnak a megbízható működésre. Ezen tanulmány ellenére a végleges hardveren valószínűleg nekünk is hasonló méréseket kell majd végezni, de erről az 5.2 fejezetben lesz szó.

A ProAsic3 család a harmadik generációja a Microsemi flash FPGA-knak [25]. A flash technológiának köszönhetően, bekapcsolás után nincsen szükség a konfiguráció betöltésére egy PROM memóriából. Ez igaz az átmeneti tápfeszültség kiesésre is, így az egész fejlesztés költségei csökkennek. Továbbá a könnyen másolható, boot-oláshoz szükséges bit folyamot sem kell levédeni, titkosítani. Szabványos AES[3] titkosítású kulcs gondoskodik flash memória tartalmának biztonságáról. A flash technológia másik, már említett előnye az alacsony fogyasztás. Nagyon alacsony a készenléti áramfelvétele, továbbá alacsony statikus és dinamikus fogyasztással rendelkezik. A kis méretet pedig a 7 szintű vezető réteg és a 4 hierarchia szintű huzalozás teszi lehetővé.

A család tagjai 1kbit nem felejtő flash ROM programmemóriával, hat beépített PPL-el, akár 620 I/O felhasználói be- és kimenettel és akár 504kbit SRAM két-portos memóriával rendelkeznek. Számos eszköz a családon belül támogatja a Cortex M1-es szoftprocesszort is. A következőkben néhány tulajdonságot és építőelemet részletesebben is ismertetek.

[3] AES - Advanced Encryption Standard

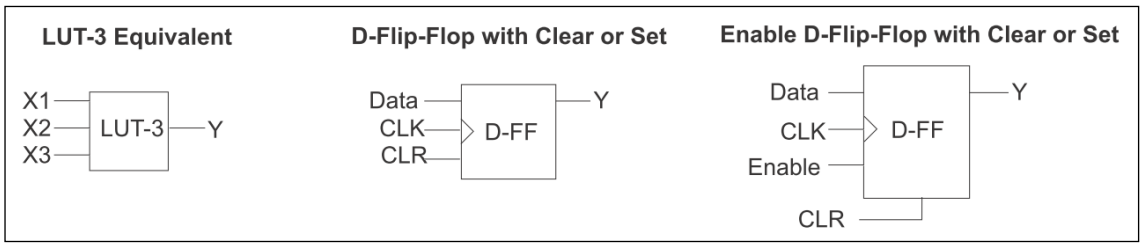


2.1 ábra: ProASIC3E architektúra [25]

Az eszköz öt alapelemből építkezik (2.1 ábra): VersaTile blokkok, flash memória blokkok, SRAM memória blokkok, I/O blokkok, CCC (Clock Conditioning Circuits – órajel előállító áramkörök) blokkok.

2.2.1 Konfigurálható Logikai Blokk

Ezen blokkok az FPGA eszközök programozható logikái, melyek összekapcsolásával és programozásával kombinációs és szinkron hálózatokat tudunk kialakítani, emellett memóriaként is használhatóak. Az FPGA eszköz erőforrásai alatt ezen logikai egységek számát is értjük, hiszen nagyban függ a megvalósítható digitális hálózat komplexitása a használható logikai blokkok számától. Sokszor ezt az adatlapokon logikai kapuk számára „átváltott” mérőszámmal jellemzik. Ez azonban megtévesztő lehet, hiszen kérdés, hogy az általunk kitalált adott kapuszámú rendszer, hogyan implementálható az eszközön található logikai elemekkel. Az Actel cég, a flash alapú FPGA-i ilyen alap építőelemét VersaTile-nak hívja. Minden egyes VersaTile egy adott három bemenetű logikai függvényt megvalósító blokká (LUT – Look-Up Table) és egy D flip-floppá konfigurálható (engedélyező jellel, vagy engedélyező jel nélkül), vagy pedig latch képződhet belőle a megfelelő programozással (2.2 ábra).



2.2 ábra: VersaTile konfigurálási módjai [25]

Ez a VersaTile blokk egységes az egész családra nézve.

2.2.2 Nem felejtő FlashROM

Minden, családban lévő eszköz rendelkezik 1kbit méretű, felhasználó által elérhető, nem felejtő, flash memóriával. A memória az IEEE 1532-es szabvány szerinti JTAG interfészen keresztül írható. 8 darab 128 bit méretű memória bankból épül fel. A 3 legmagasabb helyi értékű bit definiálja a bankot. A dokumentációk az Actel által kínált Libero fejlesztő szoftverrel javasolják a flash memória felkonfigurálását. Alkalmazások: internet-protokoll címzés, rendszer kalibrációs beállítások letárolása, eszköz sorozat-/leltárszám rögzítése, előfizetés alapú üzleti modellek támogatása (például set-top box), titkosított kommunikációhoz tartozó titkosítási kulcs(ok) tárolása. Ezen flash memória tartalmát a már fent említett szabványosított titkosítási rendszer védi (AES-128).

2.2.3 SRAM és FIFO blokkok

Az eszköz beépített SRAM[4] memória blokkokkal is rendelkezik. A 4608 bit méretű memória blokkok különböző dimenziók szerint konfigurálhatóak: 256 x 18, 512 x 9, 1k x 4, 2k x 2 és 4k x 1. A blokkhoz egymástól független írási és olvasási port-szélesség definiálható. Például az adatot 4-bit szélesen írom, de egy bitfolyamot olvasok ki belőle. Az SRAM memória blokkok JTAG interfészen keresztül inicializálhatóak. Minden SRAM blokk rendelkezik egy vezérlő egységgel, mely lehetővé teszi, hogy plusz erőforrás nélkül (plusz VersaTile nélkül) az adott memória blokkot egy szinkron FIFO[5]-ként használjuk. A stack szélessége és mélysége programozható, valamint az

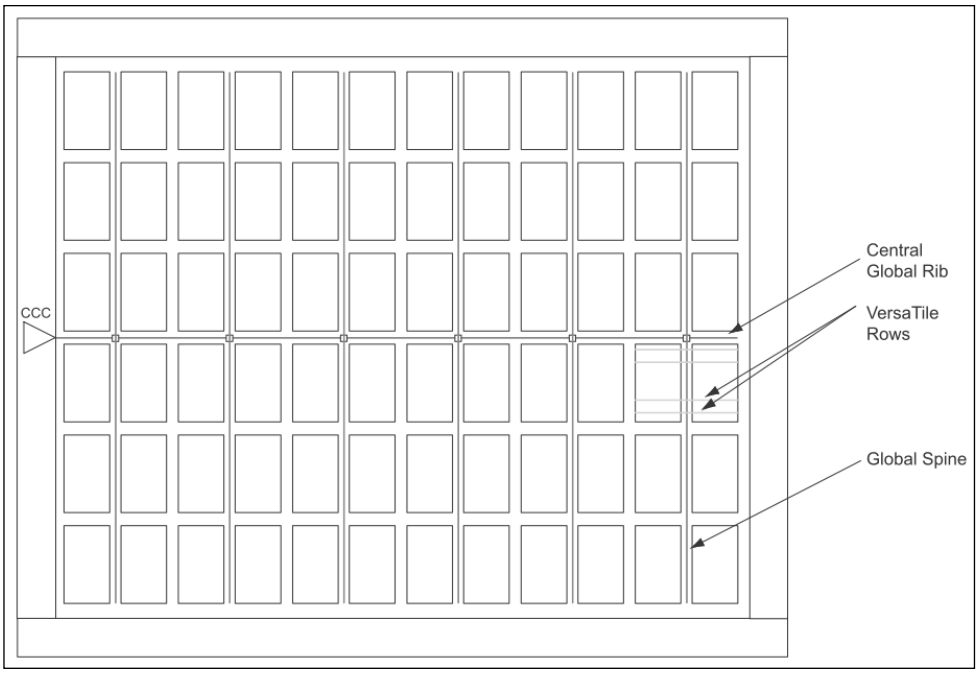
[4] SRAM – Static Random Access Memory (Statikus Véletlen Hozzáférésű Memória)

[5] FIFO – First In First Out (Első Be Első Ki) szervezésű memória, az először belekerült adat kerül ki belőle elsőként

üres (EMPTY) és teli (FULL) jelzőbiteken kívül (FLAG), majdnem üres (AEMPTY) és majdnem teli (AFULL) jelzőbitek is beállíthatóak. Mind az SRAM-ként és mind a FIFO-ként definiált blokkok összekapcsolhatóak.

2.2.4 CCC blokk

A nagy bonyolultságú programozható logikákban az órajel elosztó hálózatnak fontos szerepe van a pontos működés eléréséhez. A VersaTile blokkhoz tartozó órajel vezetékezése közel azonos hosszúságú kell, hogy legyen. Ezért általában először elvezetik a chip középpontjába, majd onnan egy „H” struktúrát követve terjesztik szét a chip-en. A ProAsic3E termék családnál ezt a 2.3 ábra látható módon tervezték meg:



2.3 ábra: Órajel elosztó hálózat „fa” topológiája [25]

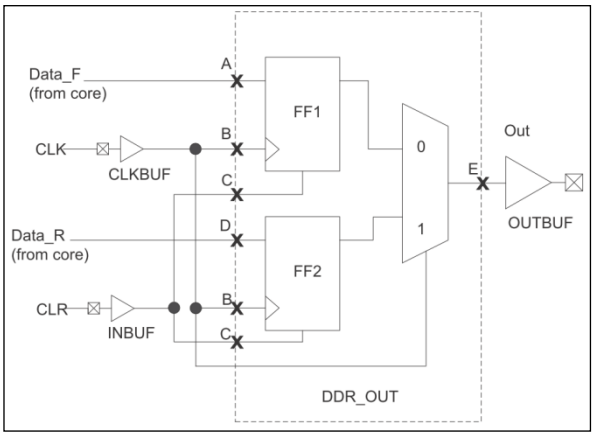
Ezek az órajelek (a hálózatunk egyes részei különféle órajelekre is működhetnek) egy órajelet előállító egységben születnek, a konfigurálásnak megfelelően. A család minden eszköze 6 CCC blokkot tartalmaz, melynek mindegyike egy beépített PLL[6]-el rendelkezik. A CCC blokkok a négy sarokban, valamint az eszköz keleti és nyugati oldalának felezőpontjában vannak (2.1 ábra). A felhasználói be- és kimenetek maximalizálása érdekében van olyan tokozás (PQ208), ahol csak az oldalsó két CCC található a chipen. A CCC blokkok külön órajel sebesség intervallumokkal

[6] PLL - Phase-Locked Loop (fázis zárt hurok)

rendelkeznek a bemeneti (1.5 Mhz – 350 Mhz) és a kimeneti (0.75 Mhz 350Mhz) oldalon. A fázistolás pedig 0°, 90°, 180° és 270° fokokban állítható.

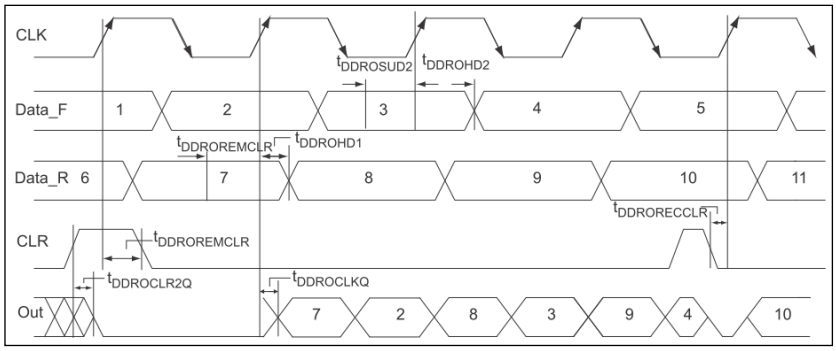
2.2.5 Általános Be-/Kimeneti Blokk (IOB)

Az FPGA eszközök egyik nagy előnye, a rugalmasan kialakítható nagy, párhuzamos struktúrákat tervesésének lehetősége mellett, hogy nagy felhasználói be- és kimeneti kapacitással rendelkeznek. Ennél az FPGA családnál ez akár 650 IO lábat jelenthet. Az FPGA-n kialakított rendszer a külvilággal való kapcsolatot az I/O lábakon keresztül teremti meg, így annak sokféle szabvány szerinti jelkondíciónak meg kell felelnie. Ezek előre konfigurálhatóak. A ProAsic3 család chipjei négy féle feszültségszintet támogatnak: 1.5V, 1.8V, 2.5V és 3.3V. Összesen 19 féle szabványos ki-/bemenetet támogat, beleértve az egyvezetékes (single-ended), a differenciális (differential) és a feszültségre vonatkoztatott (voltage-referenced) szabványokat is. Az I/O blokkok 8 bankra vannak felosztva, oldalanként kettő-kettő. Minden bank VREF mini-bankokra van felosztva, melyek referencia feszültségen üzemelnek. Egy VREF mini-bank 8-18 I/O-t tartalmazhat. Minden mini-bankban lévő I/O ugyanazon a referenciafeszültségen működik. Így ha egy mini-bankból valamely I/O referencia feszültségen üzemel, a többi mini-bankban lévő I/O használhatja ugyanazt a referencia feszültséget. Egy I/O blokk számos kimeneti, bemeneti és engedélyező regisztert tartalmaz. Így lehetséges, hogy egyszerre támogassa az egyszeres (single data rate) és dupla (double data rate) adatátviteli szabványokat is. Külön szót ejtek a DDR kimenet és bement használatáról, mert – mint azt később látni fogjuk – a DDR kimeneti modult (2.4 ábra) használni fogom az *foCLK* kimenethez (a belső órajel negáltja kikötve a kimenetre, a DDL oldali kommunikáció vezérléséhez, 4.1 fejezet).



2.4 ábra: Kimeneti DDR modul [25]

A modul a dupla adatátviteli (double data rate) szabványok (mint például DDR memória illesztése) kezelésére szolgál. A DDR átvitel lényege, hogy a busz szélesítése nélkül növelhető az átviteli sebesség, ha az órajel fel- és lefutó élére is történik adatátvitel. Ahogy ez a 2.5 ábra hullámformáin is látható:



2.5 ábra: Kimeneti DDR regiszter időzítése [25]

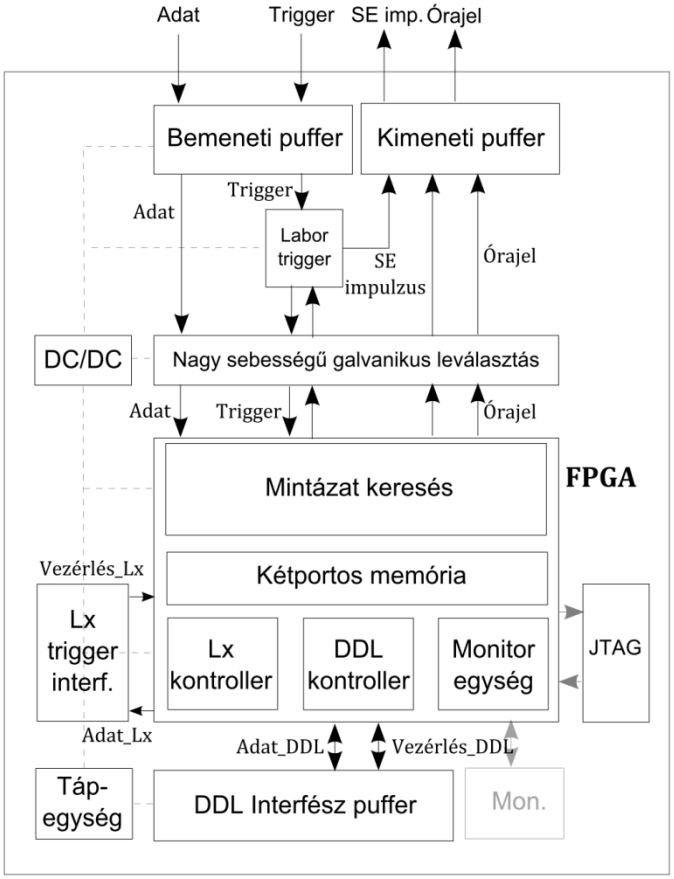
A DDR regiszter jelen alkalmazása nem dupla sebességű adatátvitelre szolgál. Ahogy minden kimenetünkről elvárjuk, hogy az adott belső órajellel egyszerre stabilizálódjon és váltson állapotot, úgy a belső órajelünk negáltjának kivezetését sem szeretnénk egy belső kombinációs hálózatból közvetlen kivezetni. A megoldás DDR regiszterrel egyszerű: Data_F = 1'b1, Data_R=1'b0. Ezt a megoldást fogom majd az implementálás során is alkalmazni.

2.2.6 Választott termék és fejlesztőkörnyezet

Az Actel cég a ProAsic3E termékcsaládhoz saját Libero IDE szoftveres fejlesztő környezetet biztosít, melyhez egy éves ingyenes licenc jár. A szoftvercsomaghoz Modelsim 10.1 szimulációs szoftver is tartozik. A Libero fejlesztő környezet a szintetizáláshoz a Synopsys cég Synplify szoftverét használja. A termékcsaládból az FG484 tokozású A3P600 típusú eszközre esett a választás. Az termék 270 szabadon használható, felhasználói ki- és bemenettel rendelkezik. 6 CCC blokkot, 24 darab 4096 bit méretű memória blokkot és 13824 VersaTile blokkot tartalmaz.

3 Front-end kártya ismertetése

Az általam tervezett digitális rendszerterv a detektorhoz közvetlenül csatlakozó, front-end elektronikus kártyán (*Front-end Electronic – továbbiakban FEE*) lévő FPGA eszközre készült. A kártya funkcióit ezen digitális rendszer működése biztosítja. Az *FEE* kártya felépítését és az FPGA által ellátott feladatokat ismertetem ebben a fejezetben. A 3.1 ábra a kártya funkcionális rendszertervét mutatja:



3.1 ábra: A Front-end elektronikus kártya funkcionális rendszerterve [8]

A detektor felől érkező digitális adatvonalak hullámimpedanciás lezárását és a jelformálást, jelleválasztást a *bemeneti puffer* modul végzi [8]. A detektort vezérlő jelek meghajtásához, a detektorban lévő kamrák távolsága miatt külön áramköri modul szükséges, ez a *kimeneti puffer*. A *labor trigger* egység tesztmérésekre használt funkciókat lát el. Előállítja a tesztméréseknél szükséges TTL szintű *L0_Local_Request* trigger jel alapján a belső *L0* trigger jelet, így az egyes korábbi fejlesztési fázisokban is tesztelhetővé válik a rendszer. Ezen kívül az ezekben az esetekben szükséges,

időkritikus időzítésekért is felelős. A detektor nagyfeszültségen működik, így a bemeneti puffert követő *galvanikus leválasztás* indokolt az FPGA, a vele közvetlen kapcsolatban álló DDL modul és a TTCrq (lásd később 3.4 fejezet) modulok védelmének érdekében. A TTC és az LTU felől érkező trigger információk fogadását és értelmezését az *Lx trigger interfész* végzi. Az interfész a CERN Mikroelektronikai Csoport (*CERN Microelectronics group)* által készített TTCrq kártyához csatlakozik, amely optikai kábelen fogadja az adatokat. A monitor modul, egy későbbi univerzális kommunikáció kialakításának lehetőségét biztosítja. Ez tenné lehetővé a rendszer működésének vizsgálatát, valamely szabványos protokollon keresztül (például USB). A kártyán szükséges különböző tápfeszültségek előállításáért a tápegység felel. Az FPGA felprogramozása *JTAG* porton keresztül történik. A DC/DC modul egy galvanikusan leválasztott tápegység, mely a detektor oldali fogadó áramkörök megtáplálását szolgálja. A nagyfeszültség felől az adatvonalakhoz hasonlóan a tápfeszültség galvanikus leválasztása is szükséges.

3.1 Az FPGA feladatai

Az FEE kártya funkcionális feladatait az FPGA látja el. Ezek a feladatok a fejlesztés során bővülhetnek, illetve változhatnak. Ezért is esett egy rugalmasabban kezelhető digitális áramköri elemre a választás a standard cellás technológia helyett. De nem ez volt az egyetlen szempont az erőforrás választásnál. A processzoros megvalósítás a később bemutatásra kerülő rendszer I/O kapacitását nem lett volna képes ellátni. Ahogy a bejövő adatokon történő mintakeresési feladatok elvégzését, az interfészek kialakítását (a DDL és a trigger oldalon), időzítéseik pontos tervezését sem.

Az általam készített rendszerterv a következő feladatokat látja el:

- megfelelő trigger impulzusra (*L0*) beolvassa a detektorról az adatokat,
- megfelelő trigger impulzusra (*L1*) beolvassa a trigger interfész felől az eseményhez és a részecske csomaghoz tartozó információkat, és azt a detektorról beolvasott adatokkal együtt letárolja a memóriában,
- megfelelő trigger impulzusra (*L2*) a memóriában tárolt eseményeket elküldi a DDL felé, a kívánt DDL protokoll (megszakítások, vezérlőjelek és időzítések) szerint.

3.2 Detektor oldali adatbemenet

Ahogy már a 1.2.4 fejezetben szerepelt, a HPTD detektor egy kamrája 10x250=2500 pad-ből épül fel. Ilyen kamrából a VHMPID detektor előtt és mögött kettő-kettő helyezkedik el. Egy FEE kártya feladata egy ilyen kamra-négyes kiolvasása. A kamrák mentén shift regiszter láncok helyezkednek el, melyekből a pad-ekről leolvasott értékek kiolvashatóak. A shift regiszterek minden vezérlőjelét az FPGA szolgáltatja. A detektor egyszeri beolvasására a trigger feltételekből adódóan körülbelül *10 μs* áll rendelkezésre. A shift regiszterek becsült működési sebessége a prototípus kifejlesztése esetén *50 MHz*. (Ez majd későbbi fejlesztések során változni, nőni fog.) A beolvasandó adat mennyisége $4 * 2500 = 10000\ bit$. A következő képlet egy esemény beolvasásához szükséges időt adja (*T*), a párhuzamos bemeneti adatvonalak számának (*B*) függvényében. A képlet *50 MHz* beolvasási sebesség és 10000 beolvasandó bit esetén adja másodpercben az időértékeket:

$$T = \frac{10000}{50 * 10^6} * \frac{1}{B}\ [s]$$

Néhány tipikus beolvasási vonalszámra az időértékeket a következő táblázat tartalmazza:

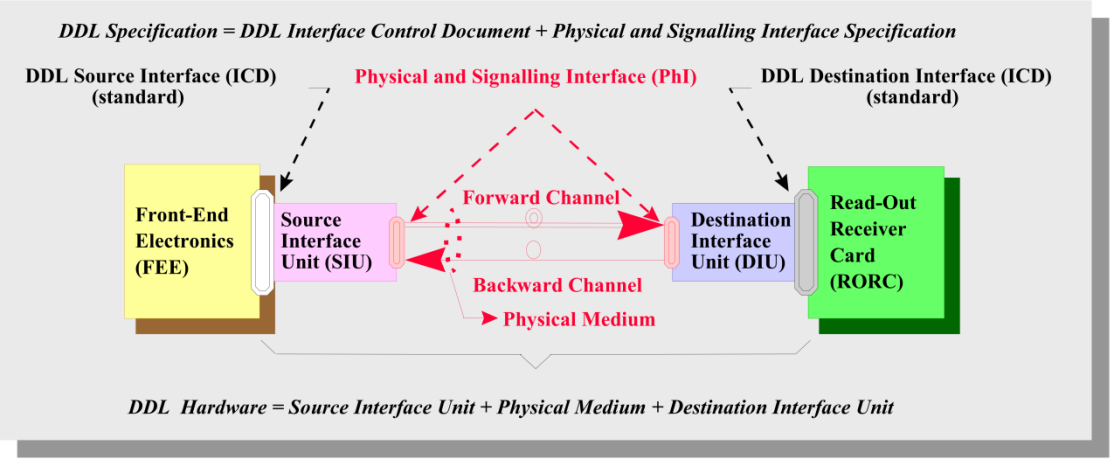
Párhuzamos bementi adatvonalak száma (*B*)	**Beolvasási idő (*T*)**
1	***200 µs***
5	***40 µs***
10	***20 µs***
20	***10 µs***

Ezek alapján a prototípus kifejlesztése során a *B=20* szálra esett a választás, mivel később az *50 MHz*-en valószínűleg gyorsítani fogunk (tervek szerint *100 MHz*) így *10 μs* alá szorítható a beolvasási idő. A *20* szál kamránként 5 szálat jelent, ami praktikusan a 10 sorból sorpárok összekapcsolását jelenti.

Egy beolvasás alkalmával tehát 20 szálon fog beérkezni, szálanként 500 bitnyi adat. Az órajelet, annak engedélyező jelét illetve a shift regiszterek párhuzamos adatbetöltésének engedélyező jelét is az FPGA szolgáltatja. Az időzítéseket a *74LV165A* típusú shift regiszterre terveztem [28].

3.3 DDL interfész

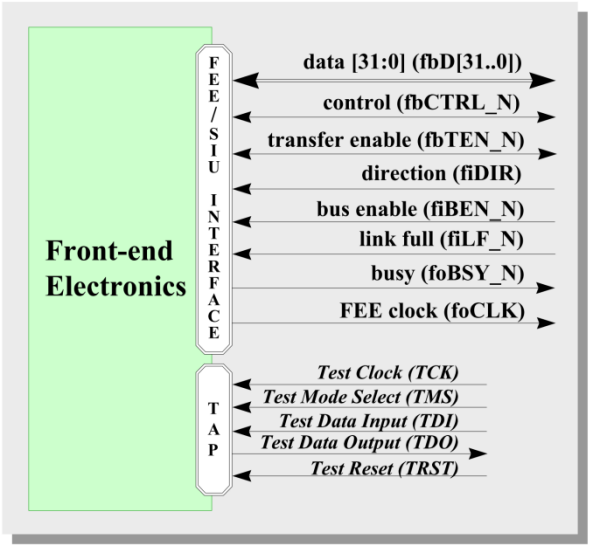
A front-end elektronika a detektoron és a trigger rendszeren kívül az adatgyűjtő rendszerrel is kommunikál a DDL interfészen keresztül [18]. Az ALICE mérőrendszer minden aldetektora egy standard adatkiolvasó láncon keresztül csatlakozik az adatgyűjtő rendszerhez. A lánc a DDL és az RORC (Read-Out Reciever Card – kiolvasó fogadó kártya) egységekből épül fel. A front-end kártya közvetlenül a DDL egységen keresztül küldi az adott esemény adatait az adatgyűjtő rendszer felé. A DDL SIU (Source Inteface Unit – Küldő Interfész Egység) a front-end elektronikus kártyához igen közel, a detektor térben helyezkedik el, míg a SIU és DIU (Destination Interface Unit – Cél Interfész Egység) egységek között akár 200 m távolságot is felölelni képes, duplex optikai kábel a fizikai réteg (3.2 ábra). A legújabb verziókban a DIU egység már az RORC egységre lett integrálva, funkciója azonban változatlan.



3.2 ábra: A DDL specifikáció hardver komponensei [20]

3.3.1 FEE-SIU adatvonalak és vezérlőjelek

A feladatom a SIU és FEE közti alapvető kommunikáció megvalósítása volt. A kommunikációs interfészt FPGA-n terveztem meg.



3.3 ábra: Az FEE-SIU egységek közti jelek nevei és a TAP (Test Access Port) jelek nevei [18]

Az FEE és SIU közti vezérlő és busz vonalakat a 3.3 ábra mutatja. Ezen kívül az FEE kártya teszteléséhez szükséges TAP (*Test Access Port*) jelek is fel vannak tüntetve. Ezek kezdetben az FPGA JTAG jeleinek, illetve a monitorozásra kialakított egység jeleinek feleltek volna meg. Az új verziókban a kivezetések ugyan megtalálhatóak, de a kezdeti elképzeléstől eltérően, már nincsenek használatban. A számítógép oldalon a kommunikációt kezelő szoftverből is kikerültek az ehhez tartozó rutinok.

Az adatgyűjtő rendszer konvencionális jelölése a következő. Az FEE-SIU jelek neveinek rövidítése **f** betűvel kezdődik. Minden jel nevében jelölve van, hogy FEE nézőpontból kimenet (**o**), bemenet (**i**) vagy kétirányú busz vonal (**b**). (Mindig a külső, DDL kommunikációhoz kapcsolódó egység nézőpontjából tekintik az jelirányokat, így például az RORC oldalon az RORC egység felől.) Ezt követően szerepel a jel rövidített elnevezése. A név után az **_N** jelölés az egyes vezérlő jelek alacsony aktív működésére utal. Minden DDL kommunikációhoz szükséges jel az **foCLK** órajel felfutó élére szinkronizált. Az FEE-SIU vonalak a következők:

- **Adatbusz (DATA, fbD[31..0]):** kétirányú, 32 bit széles adat vonal. Az **fiDIR** vezérlőjel magas állapota esetén az adat ezen a vonalon FEE-SIU irányban értelmezett, az **foCLK** jel felfutó élére szinkronizálva, amennyiben az **fbTEN_N** vezérlőjel aktív. Az adatbusz tartalmának értelmezését, egy szintén kétirányú vezérlő jel segíti. Az adat az FEE kártya állapotáról való információkat tartalmazó státusz szót tartalmaz, amennyiben az **fbCTRL_N** vezérlő jel aktív állapotban van. Az **fbCTRL_N** passzív állapota esetén normál

adatokat tartalmaz. Az **fiDIR** vonal alacsony logikai szintű állapota esetén az adatfolyam iránya SIU-FEE, amennyiben **fbTEN_N** aktív. Az **fbD[31..0]** RORC felől érkező parancsokat tartalmaz az **fbCTRL_N** vezérlő jel aktív állapota esetén, egyébként adatokat.

- **Kontrol vezérlőjel (CONTROL, fbCTRL_N):** alacsony aktív**,** kétirányú vezérlőjel az adatbusz értelmezésének segítésére. Aktív állapota esetén azt hivatott jelezni, hogy az adatbuszon megjelenő adat SIU felől érkező parancsszó, vagy FEE felől érkező státusz szó.
- **Átvitel engedélyezése (TRANSFER ENABLE, fbTEN_N):** szintén alacsony aktív, kétirányú vonal a busz vonalak vezérlésére. Aktív állapota engedélyezi az adatátvitelt a két egység között.
- **Adat iránya (DIRECTION, fiDIR):** SIU felől érkező vezérlőjel az adat irányának meghatározására. Magas állapota FEE-SIU irányt jelent az **fbD[31..0]** adatbuszon és a **fbCTRL_N**, **fbTEN_N** kétirányú vonalakon, míg alacsony állapota ellenkező irányra utal. Azaz magas állapot esetén minden kétirányú adatvonalat (**fbD[31..0]**, **fbCTRL_N**, **fbTEN_N**) a font-end elektronika hajt meg. Ellenkező esetben a tri-state vonalakat a SIU hajtja meg.
- **A link telített (LINK FULL, fiLF_N):** adatfolyam vezérlésére szolgáló jel a SIU egységtől. Aktív állapota minden adat transzfer felfüggesztését (legyen az adat vagy státusz szó) jelenti az FEE egység számára. Ennek oka mind a SIU, mind a DIU, mind pedig az RORC elfoglaltsága lehet.
- **Foglalt (BUSY, foBSY_N):** FEE vezérlő kimenete. Aktív állapota foglaltságot jelez a DDL felé, azaz ebben az esetben az FEE nem képes adatot fogadni a DDL-től.
- **FEE órajel (FEE CLOCK, foCLK):** a DDL kommunikáció az **foCLK** jel felfutó élére vezérelt. Ezt az órajelet a front-end kártya szolgáltatja. A kommunikáció maximális sebessége *50 MHz*.

A jelen fejlesztés esetében a front-end elektronika parancsszavakon kívül nem fogad adatokat a DDL felől. A parancsszavakat pedig minden esetben lekezeli. Így az foBSY_N, az FEE foglaltságát jelző vonal nem került megvalósításra (állandó magas állapotra van kötve). Ez nem jelent rendellenes működést a DDL kommunikációban. Ez

a kommunikáció az összes ALICE mérőállomáson található, nagyobb bonyolultságú detektorok elektronikái számára is készült, így megvalósítható vele adatküldés a front-end elektronikák irányába is. Számunkra ez a fejlesztés jelen állapotában nem lényeges funkció.

3.3.2 Parancsszavak

A DDL kommunikációk kialakításánál két parancsszó beépítése kötelező: *Ready to Recieve* (*RDYRX*, SIU kész a fogadásra) és *End of Block Transfer* (*EOBTR*, adatblokk átvitel vége). A rendszer működésébe ezen a két alap parancsszó lekezelését terveztem, mivel a kezdeti alapvető működések ellenőrzésére elegendő ezen parancsszavak beépítése. A parancsszavak felépítését a 3.4 ábra táblázata mutatja:

D31	D30 D12	D11 D8	D7	D6	D5	D4	D3	D2	D1	D0	data bits
	PARAMETER FIELD	IDENTIFIER FIELD	CODE FIELD				DESTINATION FIELD				command fields
X	FEE address	transaction ID	WRITE	UD	CLOSE	BTR	reserved	FEE	SIU	DIU	FECMD
X	is not used	transaction ID	0	0	0	1	0	1	0	0	RDYRX
X	is not used	transaction ID	1	0	1	1	0	1	0	0	EOBTR
X	address in FEE	transaction ID	1	1	0	1	0	1	0	0	STBWR#address
X	address in FEE	transaction ID	0	1	0	1	0	1	0	0	STBRD#address
X	address in FEE	transaction ID	1	1	0	0	0	1	0	0	FECTRL#address
X	address in FEE	transaction ID	0	1	0	0	0	1	0	0	FESTRD#address

block transfer
close block transfer
user defined command
write/read operation

3.4 ábra: DDL parancsszavak felépítése [18]

Az egyes mezők jelentései a következők: WRITE – írás/olvasás operáció, UD (User Defined command) – felhasználó által definiált parancs, CLOSE – az adott adatblokk lezárása, BTR (Block Transfer) – adatblokk küldése.

A két parancsszónál az FEE címzése mező nem használt. Az *RDYRX* parancs gyakorlatilag a mérés megkezdését jelenti. A SIU ezzel a paranccsal jelzi, hogy kész fogadni az adatokat a DDL-en keresztül. Egy esemény pedig akkor kerül átküldésre, ha az FEE egységre megfelelő *L2* trigger jel is érkezett. Az *EOBTR* parancs pedig egy adat transzfer lezárásának visszaigazolása a SIU felől.

Ezen kívül négy különböző, a felhasználó által definiált parancsszó megvalósítása lehetséges. Az FEE egységbe való adat írás az STBWR (*Start of Block*

Write – blokk írás indítása) paranccsal kezdődik. Az FEE egységből való adat olvasása az STBRD (*Start of Block Read – blokk olvasás indítása*) parancs elküldésével indulhat. Ezek az utasítások az FEE egység memóriájának bankokra osztott felépítését feltételezik. Az FECTRL (*Front-end Control – front-end vezérlés*) paranccsal az FEE egység vezérlésére vonatkozó üzenet küldhető (az *ADDRESS* címek a vezérlés különböző funkciói lehetnek). Az FESTRD (*Front-end Status Read Out – front-end státusz kiolvasás*) az FEE elektronika állapotára vonatkozó státusz szó olvasható ki (az *ADDRESS* címek a kiolvasás különböző funkciói lehetnek).

3.3.3 Státusz szavak

A státusz szavak szerepe, hogy információt szolgáltasson az FEE egység állapotáról az adatgyűjtő rendszer felé. Két féle státusz szót különböztet meg:

1. RORC által kért státusz szó, melyet az FESTRD#address paranccsal kérhet az FEE egységtől. Ez egy front-end kártyatervezők által definiált, aldetektor specifikus státusz szó. A 19 bit széles paraméter mező ennek az FEE specifikus státusz szónak van fenntartva.
2. FEE által automatikusan generált státusz szó, mely minden FEE-SIU irányú adatblokk átvitel végén automatikusan elküldendő. Ebben az esetben nem kötelezően a paraméter mező az elküldött adatblokk méretét tartalmazhatja.

D31	D30 D12	D11 D8	D7	D6	D5	D4	D3	D2	D1	D0
ERROR BIT	PARAMETER FIELD	IDENTIFIER FIELD	CODE FIELD				SOURCE FIELD			
error/OK	status parameter	transaction ID	status code		EOB	reserved	reserved	FEE	SIU	DIU
error	front-end status	transaction ID	0	1	0	0	0	1	0	0
error	data block length	transaction ID	0	1	1	0	0	1	0	0

at the end of an event or a data block

in the front-end status read-out transaction

3.5 ábra: DDL státusz szavak szerkezete [18]

A 3.5 ábra táblázat mutatja a DDL státusz szavak felépítését. Az első esetben az ERROR bit bármilyen előre definiált hibás állapot jelzésére szolgálhat. A második esetben ez a flag az FEE egységen belüli, adatblokk átvitel közben történt hibák jelzésére van. A két szó között az EOB (*End of Data Block – Adat Blokk Vége*) bit szerint tehetünk különbséget. Ez a bit mutatja meg számunkra, hogy egy adatblokk átvitel végén szereplő státusz szóról (EOB=1), vagy pedig egy RORC által igényelt, előre definiált státusz szóról van szó (EOB=0).

EOB=1 esetén, azaz adatblokk végi státusz szó esetén, a következő információkkal felruházott DTSTW (*Data Transmission Status Word – Adat küldés státusz szó*) szót küldi tovább a SIU egység az RORC felé (3.6 ábra): az *error bit* és *data block length* mezők egyszerűen továbbításra kerülnek, a *continuation* mezőben pedig „*0000*" szerepel, amennyiben az átküldendő adatblokk valóban befejeződött. Ellenkező esetben (például 2 MB-os határnál) „0001" szerepel, ami azt hivatott jelezni, hogy az adott blokk, a következő blokk transzfer folyamán egészül ki teljesen. A további mezőket a SIU egység generálja.

D31	D30 D12	D11 D8	D7	D6	D5	D4	D3	D2	D1	D0
ERROR BIT	PARAMETER FIELD	IDENTIFIER FIELD	CODE FIELD				SOURCE FIELD			
error	data block length	continuation	1	0	0	0	0	0	1	0

3.6 ábra: DTSTW DDL interfész parancs felépítése

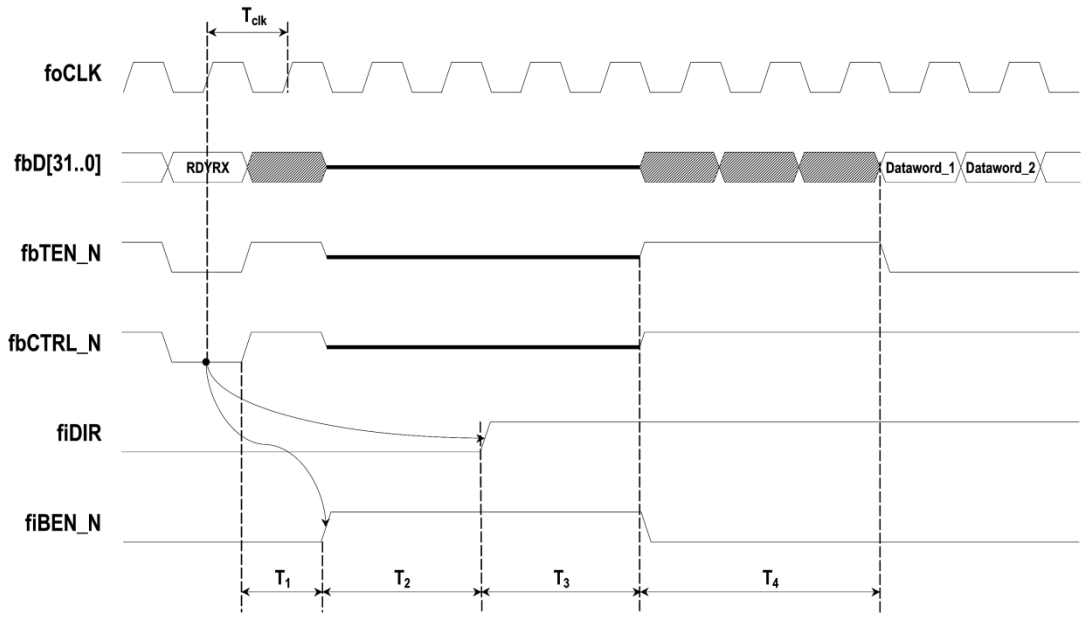
3.3.4 Időzítések és a kommunikáció menete

A DDL kommunikáció helyes, biztos működéséhez a tervezésnél figyelemmel kellett lennem a protokoll időzítéseire. A specifikáció minden vonal hold time[7] időzítési értékére *0 ns* minimumot ír elő. A setup time[8] az **foBSY_N** jel kivételével minden vonalra minimum *8 ns*, **foBSY_N** jel esetén pedig minimum *10 ns*. Az **foCLK** órajel periódus ideje minimum *20 ns*. A rendszer működését a maximális megengedett sebességre, *50 MHz*-re terveztem.

Ahogy azt már említettem, az általam tervezett kommunikációs protokoll nem használja ki a DDL kommunikáció minden lehetőségét. A következőkben a rendszer által használt funkciók működését részletezem. Adatküldésre az *RDYRX* parancs fogadása után van lehetőség. A protokoll az FEE **foCLK** órajelével szinkronban működik, a busz vonalak irányát azonban a SIU egység határozza meg, így az átvitelt ez az egység vezérli. Az *RDYRX* parancs érkezése után a busz átadása a vezérlő jelek segítségével, a következő ábrán (3.7 ábra) látható módon történik.

[7] Hold-time: az az időintervallum, ameddig az adat stabilan tartása szükséges a mintavételező órajel él után, a helyes működéshez.

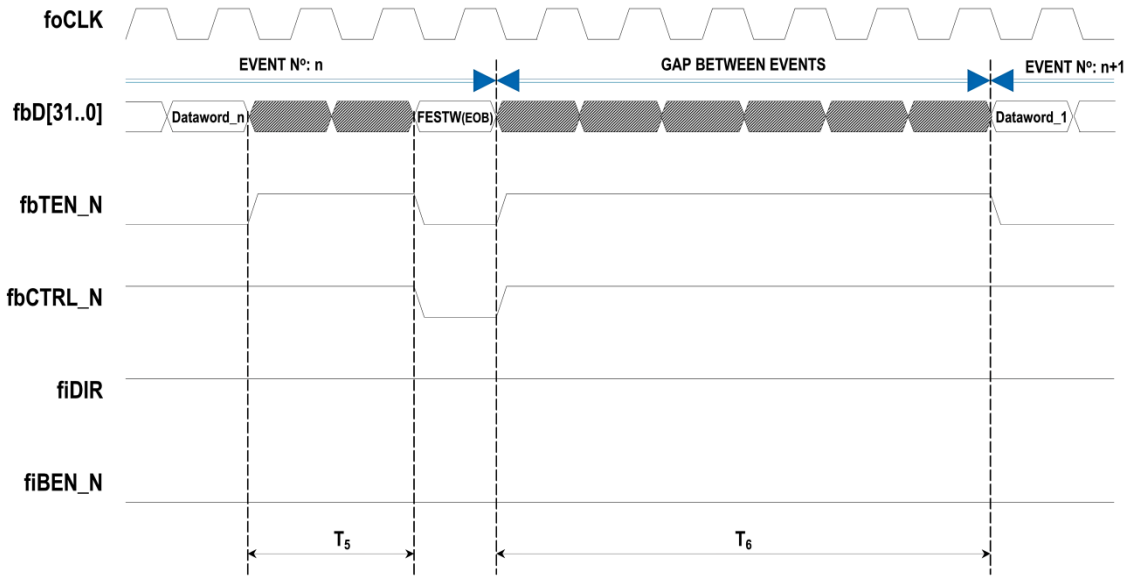
[8] Setup time: az az időintervallum, amennyivel a mintavételező órajel él előtt, stabil adat szükséges a helyes működéshez.



3.7 ábra: Adatblokk átviteli folyamat megkezdése [18]

Miután az *RDYRX* parancs átvitele megtörtént, a SIU egység nagyimpedanciás állapotba hozza a kétirányú vonalakat, amit az *fiBEN_N* jel passzív szintre emelésével jelez az FEE felé. Ezután az *fiDIR* vezérlő jellel jelzi a buszvonalak irányainak változását. Ezt követően az *fiBEN_N* vezérlőjel segítségével újra engedélyezi a buszvonalakat. Ekkor már a busz meghajtásáért a front-end elektronika felel. Az adatbuszra kikerült adatokat a buszvezérlő jelekkel együtt, az *foCLK* órajel felfutó éleivel szinkronban jelennek meg.

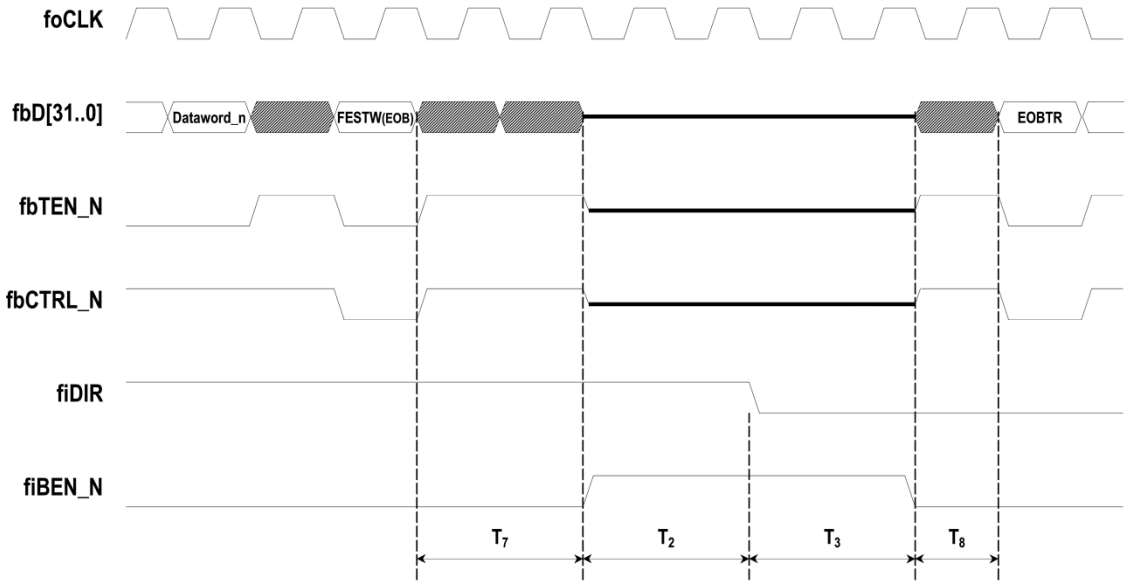
Több esemény kiküldése között a busz vezérlését nem feltétlen veszi át a SIU egység. Ekkor azonban az adatblokkok közötti protokoll a 3.8 ábra szerint történik:



3.8 ábra: Esemény küldésének vége és következő esemény küldésének kezdete [18]

Az utolsó adat után a FEE automatikusan elküldi a kiküldés utáni státusz szót, ami az *fbCTRL_N* vezérlőjel aktív állapota alapján különböztethető meg a többi adattól. A következő eseményhez tartozó adatok küldése előtt azonban legalább 16 órajel ciklust várnia kell az előírások szerint. Ezt követően kezdődhet az újabb adat küldése.

Az *EOBTR* parancs, azaz az esemény adatok küldésének végét jelző SIU üzenet elküldéséhez a SIU újból átveszi a buszjelek meghajtását. Ezt a 3.9 ábra szemlélteti:



3.9 ábra: Esemény adatok küldésének befejezése [18]

A buszvezérlés átvétele megegyezik az esemény adatok küldése elején történt metódussal: miután a SIU megkapta az adatblokk végi státusz szót, az *fiBEN_N* vezérlőjel segítségével utasítja a FEE egységet, hogy magas impedanciába téve a busz vonalakat, adja át a vezérlést. Ezt követően az *fiDIR* vezérlőjellel az adatok küldési irányának változását is jelzi, majd elküldi a megfelelő, esemény adatok küldésének végét jelentő *EOBTR* parancsot.

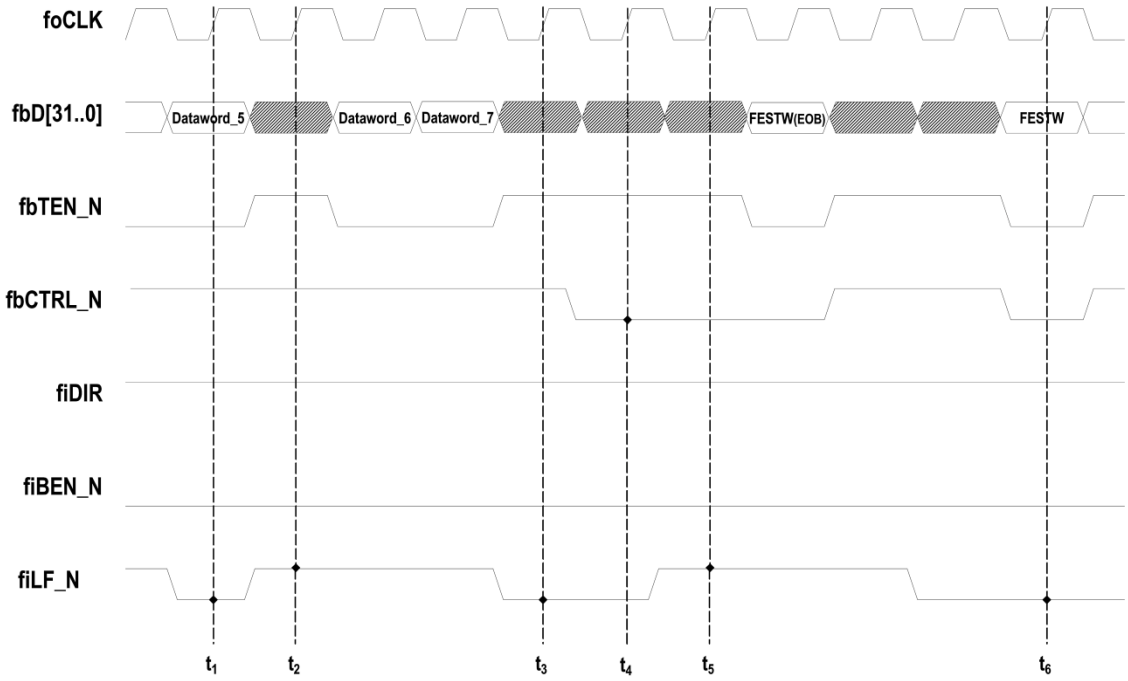
Az idődiagramokon szereplő, előírt időintervallumok értékét a következő a 3.10 ábra táblázat tartalmazza:

Idő szimbólum	Min [T_{clk}]	Max [T_{clk}]
T1	1	1
T2	2	2
T3	2	2
T4	1	∞
T5	0	4^{14}
T6	16	∞
T7	4	12
T8	1	1

3.10 ábra: Időzítési követelmények

Az ábrákon és a táblázatban szereplő T_{clk} minimum értéke *20 ns.* Látható, hogy a két esemény küldése közötti várakozási időnek minimum 16 órajel ciklusnak kell lennie. A T_5 értéke a FEE elektronikán múlik, ezért minimum értéke gyakorlatilag nincsen. A többi érték az SIU egység pontos működését határozza meg, így a busz vezérlés átvétele is pontosan megtervezhető.

A SIU egység az *fiLF_N* vezérlő bemeneten jelzi a *link full* megszakítás állapotát. Akkor az FEE kénytelen félbeszakítani az adatküldést, egészen addig, amíg a vezérlő jel újra magas állapotba nem kerül. A folyamatot a következő ábrán (3.11 ábra) látható:



3.11 ábra: *Link full* megszakítás folyamata

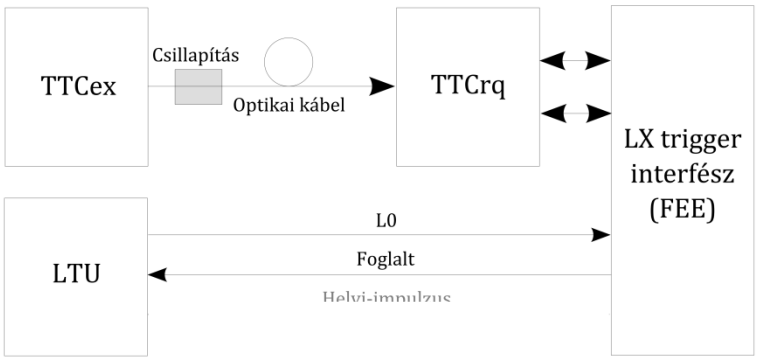
A megszakítás kétféle lehet: adatküldés közbeni és státusz szó küldése közbeni. A kettő között az *fbCTRL_N* vezérlőjel beállításában van különbség. A státusz szó küldésénél ugyanis ez a vezérlő vonal jelzi, hogy nem adatról van szó. A *fbCTRL_N* vezérlőjel, a megszakítás ellenére beállításra kerül, annak érdekében, hogy a SIU egység tisztában legyen vele, hogy már csak egy státusz szó elküldésére vár.

3.4 Trigger bemenet

Az FEE működése a trigger rendszer által vezérelt. Mind a detektorról való beolvasást és azon adatok letárolását, mind pedig a DDL-en keresztül, az adatgyűjtő rendszer felé való adattovábbítást trigger jelek vezérlik. A CTP (*Central Trigger Processor – Központi Trigger Processzor*) három trigger szintet különböztet meg: az összes 50 trigger bemenetből 24 *L0* szintű, 20 *L1* és 6 *L2* szintű trigger kezelésére alkalmas. *L0* szintű jeleknek olyan trigger jeleket értünk melyek az ütközéstől számítva kevesebb, mint *800 ns* alatt előállnak. Az *L1* szintű jeleknél ez *6.1 μs*-ot, míg *L2* szintű jelek esetén *87.6 μs*-ot jelent. A 3.1 fejezetben már vázlatosan említettem az egyes trigger jelek jelentését. Ebben a fejezetben ezen trigger jelek használatát részletezem. A trigger jelek értelmezése detektoronként változhat. A következő fejezetekben, a számomra előírt és általam megvalósított értelmezést ismertetem.

3.4.1 *L0* szintű trigger

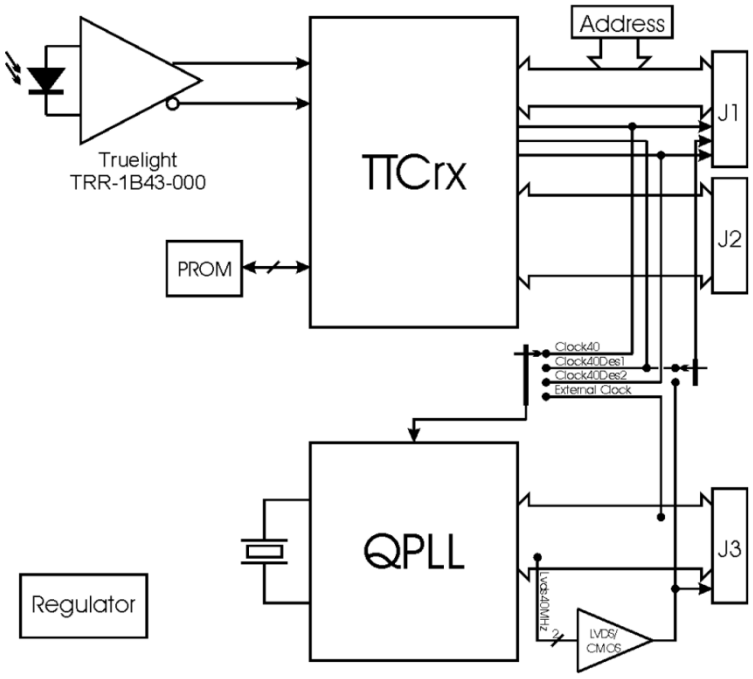
Az LHC gyűrűben az egyes részecske csomagok a globális órajel szerint, azaz *40 MHz* körüli frekvencia szerint ütköznek. Ezen eseményeket azonban nem is mind tudnánk és nem is mind rögzítjük. Az *L0* szintű trigger a legkisebb késleltetésű vonal, így jelentése sokszor az egyes aldetektorok felé, hogy az adott esemény érdekes számunkra és rögzíteni kívánjuk [17]. Az LTU (Local Trigger Unit – Helyi Trigger Egység) egységből érkező *L0* trigger jel és a felé közvetített *BUSY* jel csavart érpárok formájában kerül összeköttetésbe az FEE egységgel. Ahogy azt a 1.3.1 fejezetben a 1.8 ábra is szemlélteti, a *CTP* bemenetén az ütközéshez képest *900 ns*-al előáll az *L0* trigger impulzus. A *CTP* ezt 200 ns-on belül jóváhagyja és továbbítja az *LTU* felé. Az *LTU*-ból érkező *L0* impulzus jelenti számunkra a detektorról való beolvasás parancsát. Azaz ezen impulzus a detektor-beolvasás triggere. A rendszer válasza egy szintén csavart érpáron megjelenő *BUSY* (*foglalt*), foglaltságot jelző jel az *LTU* felé (3.12 ábra). A *BUSY* jel arra szolgál, hogy annak aktív állapota esetén az *LTU* ne adjon ki újabb *L0* parancsot. Praktikusan a beolvasás idejére állítjuk aktívra, de más indok is elfogadható.



3.12 ábra: A TTCrq fizikai interfésze front-end oldalon [8]

3.4.2 *L1* szintű trigger

Az *L1* szintű trigger egy magasabb szintű vezérlő jele a front-end elektronikáknak. Jelentése a HPTD detektor számára a következő: megerősítés a detektorról beolvasott adatok letárolására. Az *L1* trigger jel a TTCrq CERN Mikroelektronikai Csoport (*CERN Microelectronics group)* által fejlesztett egységen keresztül érkezik [21]. Az a TTCrq egység egy TTCrx és QPLL egységekből épül fel (3.13 ábra).



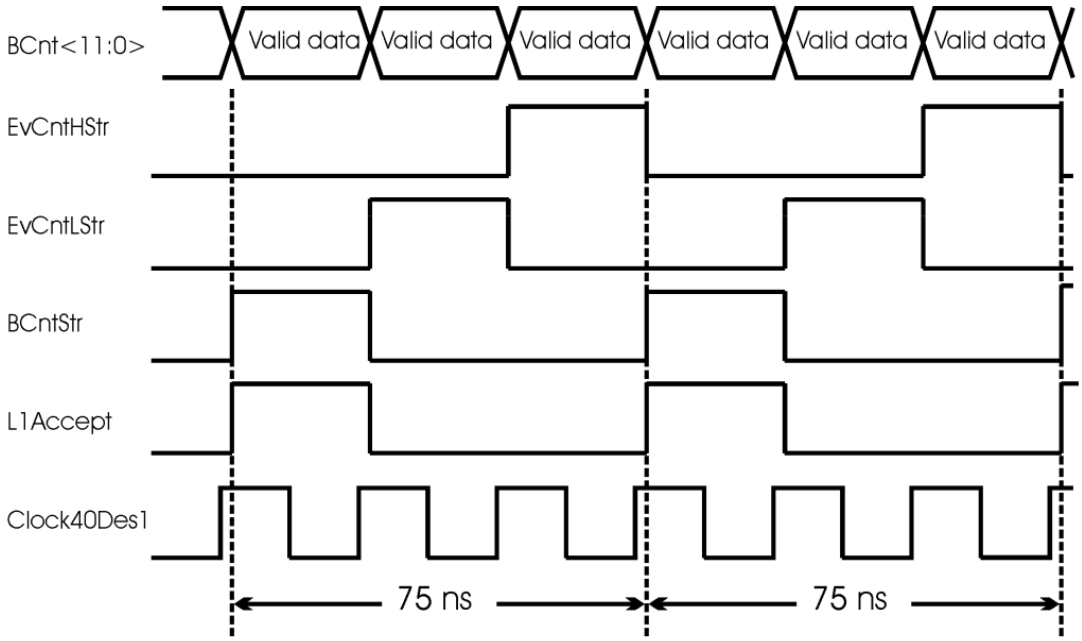
3.13 ábra: TTCrq blokk vázlata [21]

A TTCrq kártya az LHC mentén minden állomáson elérhető optikai vezetéket egy optikai csatolón (Truelight TRR-1B34-000) fogadja. Az optikai kábelen érkező információk tartalmazzák az *L1* trigger jelre vonatkozó információkat a front-end elektronika számára. Ezen információk: a globális *LHC* 40.08 MHz-es órajel, *bunch*

crossing counter (*részecskecsomag azonosító*) érték és *event counter* (*esemény azonosító*) érték.

3.4.2.1 A TTCrx és front-end kártya kommunikációja

A TTCrq kártyáról a front-end kártya gyakorlatilag a TTCrx egységgel kommunikál [22]. Ez egy 12 bit széles adatbuszon és 5 adatvonalon keresztül történik. A 12 bit széles, a részecske csomagok azonosítására használt, *bunch crossing counter* érték és a 24 bit széles, az események azonosításra használt *event counter* érték a 12 bit széles adatbuszon időosztásosan áll elő. Ennek vezérlésére három vezérlő „strobe" jel (*EvCntHStr, EvCntLStr, BCntStr*) szolgál. Az *L1* trigger impulzus jele az L1Accept adatvonalon jelenik meg. A kommunikáció a globális, *40.08 MHz LHC* frekvenciára történik, melyet szintén a TTCrq egység szolgáltat (*Clock40Des1*). A szekvenciát a 3.14 ábra mutatja.



3.14 ábra: L1 trigger szekvencia [22]

A front-end kártya feladata *L1* trigger jel érkezése esetén (*L1Accept*), a detektorról beolvasott adatok és a hozzájuk tartozó esemény és részecske csomag azonosító értékek letárolása a memóriában. A DDL felé pedig egy későbbi trigger (*L2*) érkezésekor, mind a detektor adatait, mind pedig az azonosítókat továbbítani kell egy esemény küldése esetén.

3.4.3 L2 szintű trigger

Az *L2* trigger szintén optikai linken érkezik a front-end egységhez. A fejlesztés jelen fázisában azonban csak a következő feltételezésű specifikációt kaptam: az *L2* trigger jelet tekintsem egy megfelelően hosszú (a rendszer belső, 50 MHz-es órajellel való mintavételezéshez elegendően hosszú) impulzusnak, ami készen (megfelelő jelkondíciókkal) érkezik az FPGA-hoz. Jelentése pedig, egy esemény adatainak elküldése a DDL-en keresztül. A kiküldés során a legkorábban beérkezett adat kerül először elküldésre (FIFO sorrend).

3.4.4 Trigger jelek szekvenciája

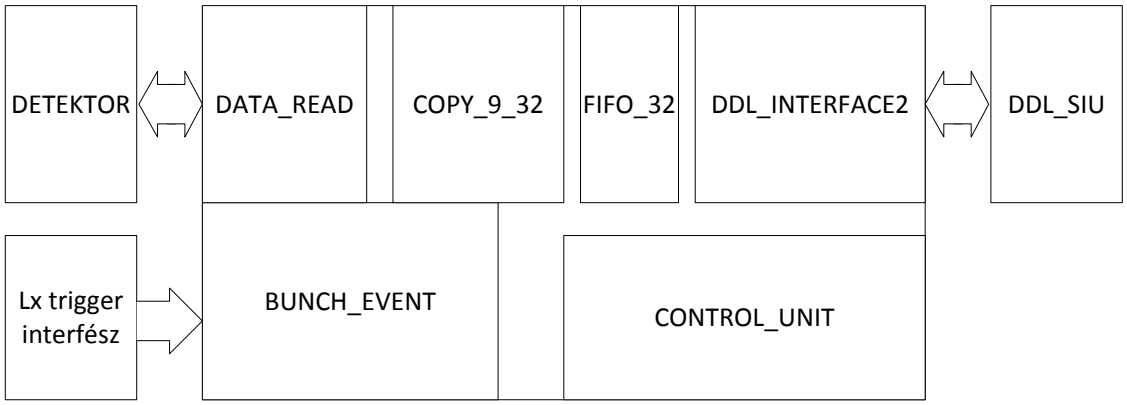
Az esetleges *L0* és a hozzá tartozó *L2* vezérlőjelek között érkezhetnek más trigger jelek is. Így a trigger rendszer szabályokban rögzíti a trigger jelek érkezésének lehetséges sorrendjét, a trigger-esemény konzisztenciájának megtartása végett. Az LHC trigger-rendszere a következő szekvenciákat engedi meg: *L0* és *L1* között nem érkezhet más trigger, míg *L1* és *L2* között megengedett. Például[9]: $L0_1 - L1_1$ - $L0_2 - L1_2 - L2_1 - L2_2$ megengedett, míg az $L0_1 - L0_2 - L1_1 - L1_2 - L2_1 - L2_2$ nem megengedett szekvencia. Továbbá amennyiben az adott időkereten belül nem érkezik *L1* trigger az adat eldobásra kerül. *L0* trigger érkezésétől számítjuk az időkereteket. Ehhez képest az *L1* trigger *5.8 μs* és *7.2 μs* között érkezhet. Ha ezen a kereten belül mégsem érkezik, akkor az *L0-ra* beolvasott adatot eldobjuk, töröljük a memóriából. *L2* trigger a *79 μs* és *96 μs* közötti időkeretben érkezhet. Amennyiben érezik, kiküldjük a legkorábban érkezett és letárolt adatot (FIFO sorrend), majd újra *L0*-ra várunk. Ha nem érkezik *L2* akkor az időkeret lejártával újabb *L0*-ra várunk.

[9] Az azonos indexű trigger jelek tartoznak egymáshoz. Azaz az $\{L0_1, L1_1, L2_1\}$ és az $\{L0_2, L1_2, L2_2\}$ trigger csoportok vonatkoznak ugyanarra az eseményre.

4 Rendszerterv megvalósítása FPGA-n

Az előző fejezetekben leírt feladatok ellátására Verilog nyelven írt, digitális rendszert terveztem, és annak helyes, kívánt működését szimulációkkal ellenőriztem. Ebben a fejezetben a rendszerterv működését szimulációs ábrák bemutatásával is illusztrálom. A fejlesztő környezet, az Actel cég által kínált, egy éves ingyenes licenccel letölthető Libero SoC v10.1 szoftvercsomag. A csomag tartalmaz egy Modelsim 10.1b verziójú szimulációs szoftvert és egy Synplify Pro szintetizáló szoftvert is. A bemutatott szimulációk a Modelsim szimulátorában készültek.

A rendszer felépítését, nagy vonalaiban a következő, 4.1 ábra mutatja be:



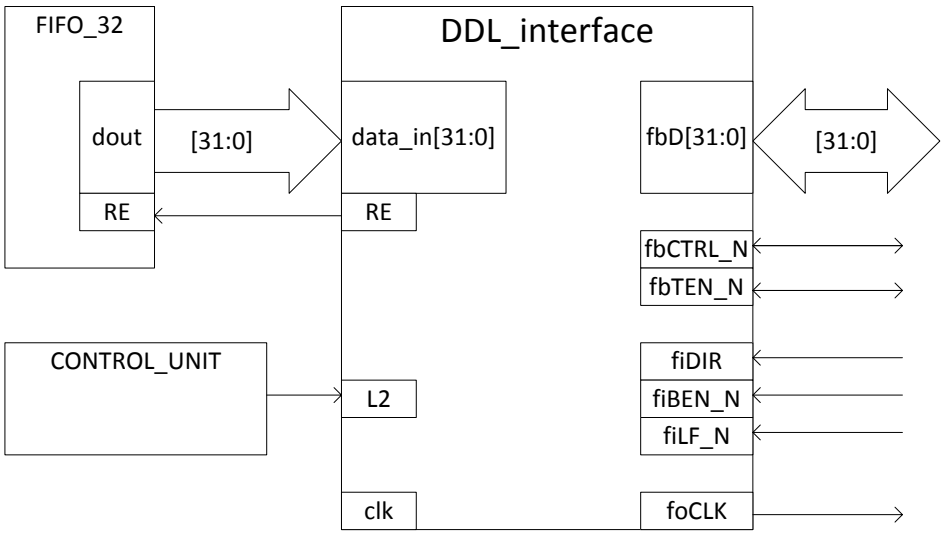
4.1 ábra: A rendszer vázlatos felépítése

A detektor beolvasását egy *DATA_READ* modul végzi, aminek felépítését a 4.3 és a 4.6 fejezetek mutatják majd be részleteiben. A *BUNCH_EVENT* modul az Lx trigger interfészen keresztül kommunikál, a 3.4.2 fejezetben bemutatott TTCrq kártyával. Az adatgyűjtő rendszer felé, a DDL kommunikációért a *DDL_INTERFACE2* modul felelős. A *COPY_9_32* egység az adatokat megfelelően rendezve, egy kiküldés előtti átmeneti memória egységbe (*FIFO_32*) másolja. A rendszer vezérlését pedig a *CONTROL_UNIT* egység végzi.

A következőkben az egyes modulok működését ismertetem, illetve bemutatom a működésük teszteléséhez végzett szimulációk eredményét. Majd 4.6 fejezetben egy részletesebb, teljes rendszerképet is bemutatok.

4.1 DDL interfész

A 3.3 fejezetben ismertetett kommunikáció kezelésére egy verilog modul szolgál (*DDL_interface2.v*). A 4.2 ábra mutatja a modul blokk diagramját illetve, hogy milyen más belső blokkokkal kommunikál. A külső oldalon lévő jelek, megfelelnek a 3.3 fejezetben ismertetett jelekkel.



4.2 ábra: DDL interfész modul és kapcsolatai

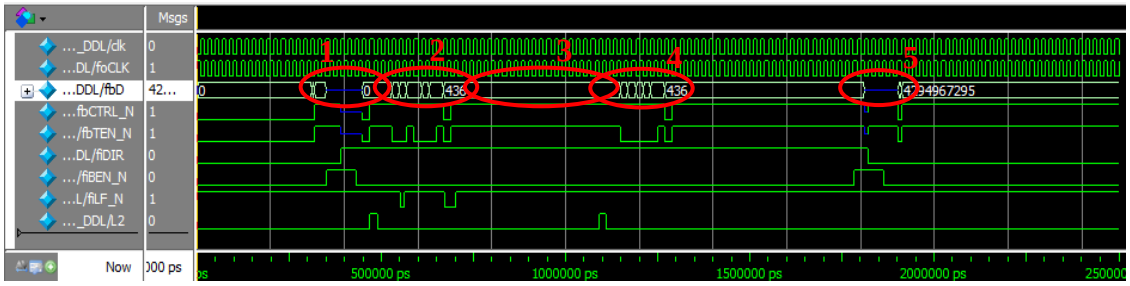
Miután a megfelelő beérkező parancs által, az FEE kártya átveszi a busz vezérlését, egy engedélyező jelre vár az adatok küldéséhez. Az engedélyező jelet, egy órajel periódusnyi impulzus formájában a *CONROL_UNIT L2* jele szolgáltatja. A *DDL_interface* modul ezen aktív magas jel, aktív állapotának mintavételezését követően, egy *RE (Read Enable – olvasás engedélyezés)* jelet küld egy 32 bit széles fifo blokknak (*FIFO_32*), mely a kiküldendő adatokat tárolja. A *FIFO_32* blokkban két *32'hFFFFFFFF* adat jelzi az esemény végét. A *DDL_interface* modul a 32 bit egyest tartalmazó adatsorig minden adatot elküld, majd újra engedélyező jelre vár a következő esemény adatainak küldéséhez. Az új adat legkorábbi küldése a DDL protokoll szerinti 16 órajel ciklus után kezdődhet.

A *foCLK* kimeneti órajel a belső órajel (*50 MHz*) negáltjának felel meg, mely egy DDR flip-flop-on keresztül kerül kivezetésre az FPGA-ból. Így minden setup-time követelményt betartunk. A DDR flip-flop a 2.2.5 fejezetben ismertetett módon került felhasználásra a kimeneten.

A modul működését egy 32-bites FIFO memóriából és a modulból összeállított teszt rendszer szimulációjával ellenőriztem (*testbench_FIFO_DDL.v*). A teszt rendszer

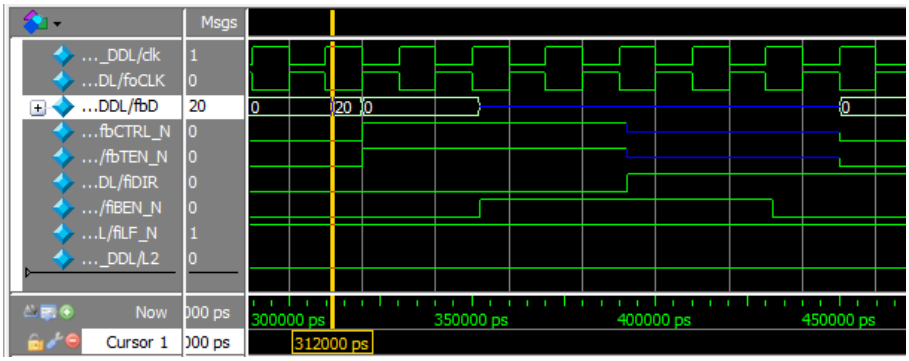
gerjesztéseinek időzítésénél ügyeltem az előírt setup és hold time-okra, azok szélsőértékeit használva. A FIFO memóriát előtte megfelelő szerkezetű adattal töltöttem fel. A szimuláció idődiagramját a 4.3 ábra mutatja. A szimuláció a következő DDL szerinti működéseket valósítja meg:

- a busz vezérlésének átvétele a megfelelő paranccsal (1),
- *L2* trigger jel érkezésére adatküldés az eseményhatárig (2),
- a küldés során különböző *link full* megszakítások (fiLF_N),
- a küldés után a 16 órajel ciklus kivárása (3),
- újabb *L2* jelre újabb adatküldés megszakítások nélkül (4), majd
- a második adat transzfer után a SIU, megfelelő parancs küldésével visszaveszi a busz vezérlését (5).



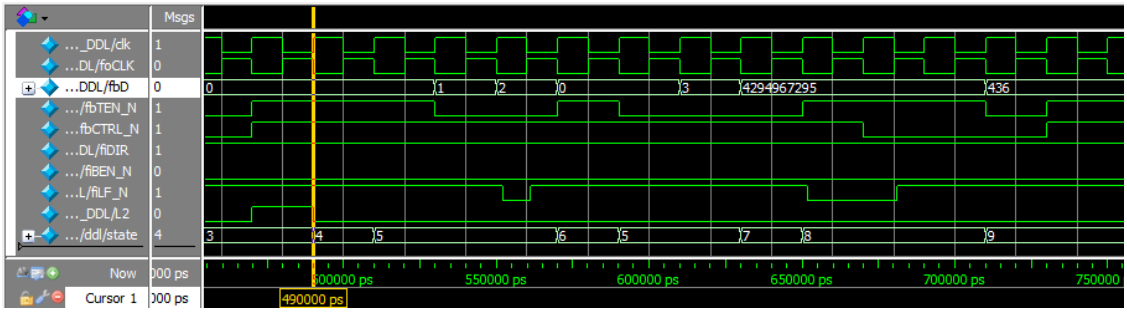
4.3 ábra: DDL_interface2.v szimulációja

A busz vezérlésének átvételét a 4.4 ábra mutatja. A buszon megjelenő *20*-as érték az a parancs, melyet a vezérlés átadása előtt küld a SIU egység az FEE felé. Ezt követően, a 3.3 fejezetben előírt időértékek betartásával, megtörténik a busz jelek vezérlésének átvétele. A buszvonalakat csak az átvételi szekvencia után hajtja meg az FEE egység.



4.4 ábra: Busz vezérlésének átvétele

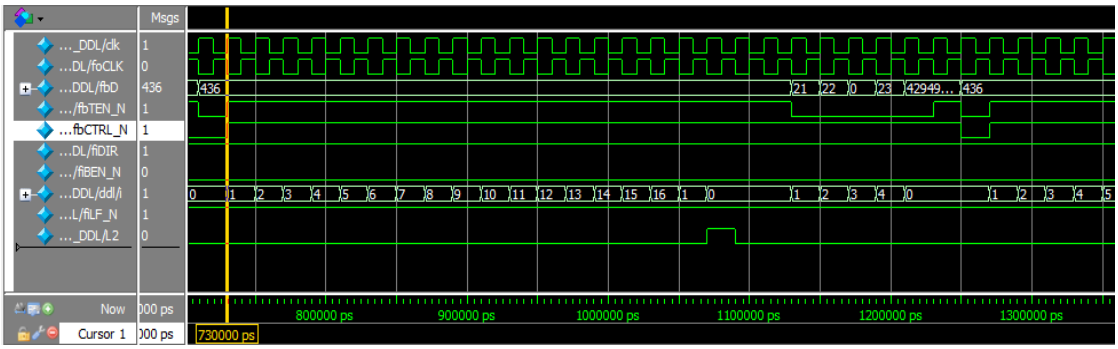
Az aktív *L2* bemenet mintavétele után megkezdődik a küldési szekvencia. A küldendő adatsor: *'1','2','0','3'* és az esemény határt jelző *'4294967295', 32* bitnyi egyes. A küldést a SIU felé az *fbTEN_N* aktív buszvonal jelzi. Az adatok küldését az *fiLF_N* bemenet által jelzett *link full* megszakítás kétféleképpen szakíthatja félbe (4.5 ábra): adat és státusz szó küldése közben. A *'2'* átvitele során az *fiLF_N* bemenet *link full* megszakítást jelez. Így a következő adat (*'0'*) küldése egészen addig nem történik meg, míg az *fiLF_N* vezérlő bemenet újból passzív állapotba nem kerül. Mivel ez azonnal megtörténik, a soron következő adat küldése egyetlen órajelet késik csak. Az esemény végét jelző adat (*'4294967295'*) szintén kiküldésre kerül. Ezt követően a kívánt protokoll szerint egy státusz szó elküldésére kerül sor (*'436'*). A státusz szó elküldése előtt szintén érkezhet megszakítás. Ekkor a protokoll szerint az *fbCTRL_N* vezérlő jel aktív állapotba állításával jelezni kell a SIU egység felé, hogy már csak a státusz szó kiküldése van hátra. A státusz szót azonban *fiLF_N* bemenet újbóli magas állapotba kerüléséig nem szabad kiküldeni.



4.5 ábra: Adatküldés megszakításokkal

A 4.5 ábra a kommunikáció vonalain kívül a modul belső állapotgép állapotait is ábrázolja: küldés állapota (5), küldés közbeni megszakítás állapota (6), státusz szó küldés közbeni megszakítás állapota (8).

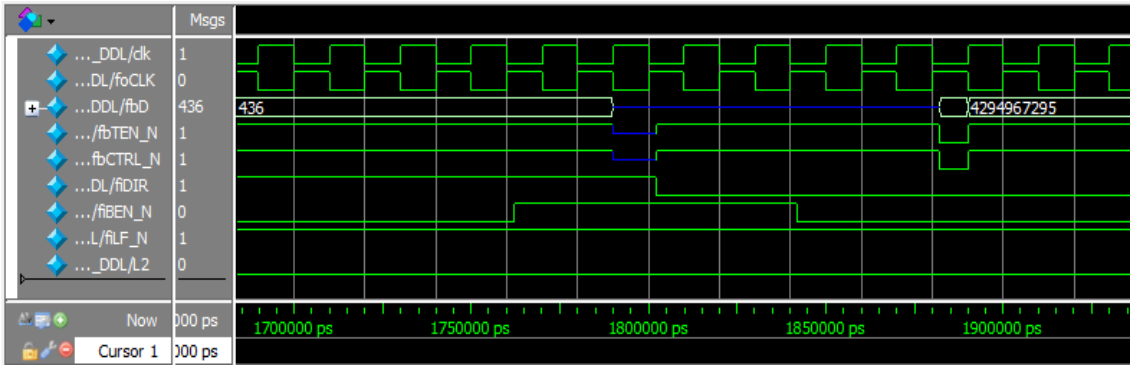
Két esemény küldése között előírt 16 órajel periódusnyi várakozás szükséges. Ezt az időt és a kiküldött adatok számát egy belső számláló számolja. A következő ábrán ezen számláló értékei is láthatóak (*'i'*).



4.6 ábra: Eseményküldések közötti várakozási idő

A 16 órajel ciklus letelte után az újabb *L2* bemenetre újabb adatküldés indul.

Egy esemény küldése után (az újabb *L2* impulzus helyett) a SIU egység visszaveheti a vezérlést (4.7 ábra). A vezérlés átvétele után újabb parancsokat küldhet a SIU egység az FEE felé. Parancsokat adhat a mérés befejezésére, majd újabb mérés indítására.



4.7 ábra: Busz vezérlésének visszavétele

4.2 Tömörítés

A fejlesztés célkitűzései szerint, a front-end elektronika kapacitása elegendő kell, hogy legyen legalább 20 esemény adatainak átmeneti tárolására. Ez $20 * 4 * 2500 = 200\ kbit$ memóriaterületnek felel meg. A választott eszközben 24 darab blokk RAM, azaz $24 * 4\ kbit = 96\ kbit$ memória áll rendelkezésre. Amíg ez a memóriaméret nem elegendő *200 kbit* nyers adat letárolására, addig az egyel nagyobb memóriamérettel rendelkező FPGA választása 60 RAM blokkot jelentene, ami elméletileg 24 esemény DDL felé küldés előtti letárolására is elegendő ($60 * 4\ kbit = 240\ kbit$). Azonban a nagyobb memóriával rendelkező eszközök, nagyobb árfekvésű termékek is egyben (24 memória blokk ~63$; 60 memória blokk ~140$). Így felmerült a kérdés, megoldható e valahogyan, hogy a kisebb memóriájú eszköz memória kapacitása is elegendő legyen. A

megoldás a tárolandó adat valamilyen tömörítése, hiszen a nyers *200 kbit* méretű adat semmiképpen sem fér bele a *96 kbit* méretű memóriába.

A tömörítés lehetőségének körbejárásához a tárolandó adat vizsgálata volt szükséges. Ehhez a [27] tanulmányt vettem alapul. A tanulmány egy tíz kamrából álló, ALICE rendszerbe épített HPTD detektor szimulációját tartalmazza. A szimuláció egyik kimenete, a detektoron történt részecske becsapódások adatbázisa. Ezt az adatbázist dolgoztam fel a tömörítés vizsgálatához. A szimulációban 4 kamra a VHMPID detektor előtt, 6 kamra pedig a VHMID detektor után kapott helyet. Ezen kívül a pad méretek is különböztek: a leendő $4\,mm * 100\,mm$ méretű pad-ek helyett, $4\,mm * 50\,mm$ felosztású pad-ek szerepeltek. Az adatfeldolgozásnál ezeket a pad méreteket átalakítottam, azaz minden két szomszédos pad-et összevontam. A szimulációs adatbázishoz egy strukturált szövegfájl formájában jutottam hozzá (a tanulmány szerzőjétől, Boldizsár Lászlótól). Ennek beolvasását és feldolgozását Matlab-ban végeztem. A matlab script (compression.m) és az adatbázis (100.txt) a CD-mellékleten megtalálható.

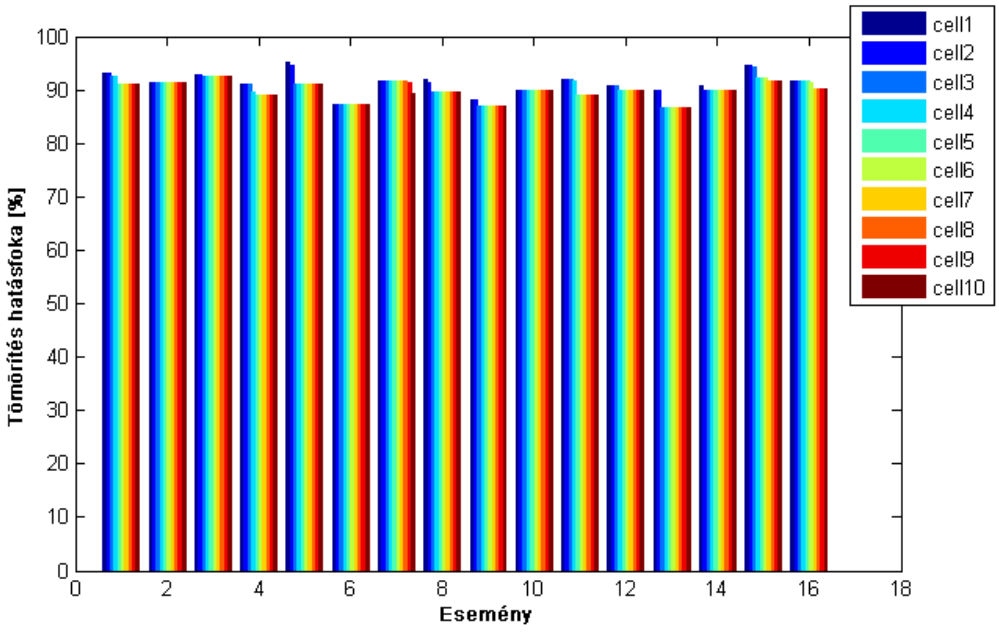
Az adatok beolvasása után megvizsgáltam, azok telítettségét, azzal a feltételezéssel, hogy a találatok elszórtan és ritkán helyezkednek majd el a kamrák négyzetrácsán. A detektor shift regiszterei a találathoz tartozó pad-ekről fognak ’1’-es bitet leolvasni és továbbítani. Így (adott energiaszintű vágással: >100eV) a találatokat ’1’-nek, a találat nélküli koordinátákat pedig ’0’-nak feleltettem meg. Ezután ezen adatfolyamok telítettségét vizsgálva, szignifikáns eltolódást tapasztaltam a nullák számának irányában. Az egyesekre vonatkoztatott telítettség mindig 10% alatt maradt. Emiatt egy egyszerű és könnyen implementálható tömörítési módot, az egyesek helyének, sorszámainak letárolását választottam. Az ilyen adatstruktúrák esetében ez a legegyszerűbben implementálható, de egyben leghatékonyabb tömörítési eljárás. Ezen kívül jól illeszthető az adatbeolvasás tagoltságához is.

Amint azt már a 3.2 fejezetben említettem, az egyes kamrákról való adatbeolvasás, kamránként 5 vonalon, 2 sort összevonva történik. Így egy vonalon $2 * 250 = 500\,bit$nyi adat érkezik be. A tömörítés vonalanként, az egyesek sorszámának letárolásával történik. Az egyesek sorszáma, pozíciója 1-től számozva 1-500 intervallumba esik, mely 9 biten ábrázolható. Az 500 bit „nyers” adat letárolása helyett tehát az egyesek számának megfelelő mennyiségű 9 bites adatokat tárolok le a memóriában. A tömörítés hatékonysága fordítottan arányos az adat kitöltöttségével.

A szimulációs adattömböt Matlab program segítségével olvastam be. A tömörítés szimulációja is ebben a környezetben történt. 16 eseményt olvastam be, melyek mindegyike a fentebb leírtak szerinti 6 kamra adatait tartalmazta. Megfelelő pad-transzformáció után a 4.8 ábra szerinti eredményeket kaptam. Az ábra az egyes cellák szerinti tömörítés hatásfokát mutatja eseményenként. A tömörítés a tervezett működés szerint két-soronkénti, 9 bites ’1’-es pozíciókat tárol le. A tömörítés hatásosságát az eredeti nyers adat méretének és a tömörített adat méretének arányával számoltam:

$$\text{tömörítés hatásfoka } [\%] = \frac{\text{eredeti adatméret} - \text{tömörített adatméret}}{\text{eredeti adatméret}} * 100$$

Az ábrán megfigyelhető, hogy a tömörítés hatásfoka minden esetben 80-% felett van. A szimulációk azt is mutatják, hogy a fizikusok sugallta alacsony telítettség milyen mértékű. Megfigyelhető, hogy a távolabbi cellákon kisebb a tömörítés hatékonysága, nagyobb a telítettség, mivel több részecske csapódik be a korábbi részecskék szóródása miatt.



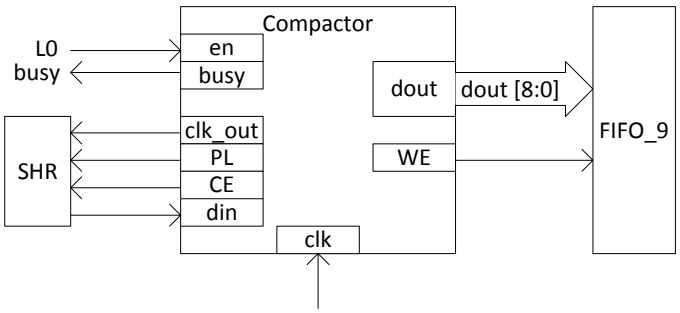
4.8 ábra: Tömörítés hatásfoka 16 esemény vizsgálatával

80 %-os tömörítési hatásfokot figyelembe véve a tárolandó adat $200\,kbit * 0.2 = 40\,kbit$ méretűre csökken. Így, elméletileg akár (a *96 kbit* memóriában) 40 esemény adatainak tárolására is lehetőségünk van.

4.3 Detektor oldali adatbeolvasás

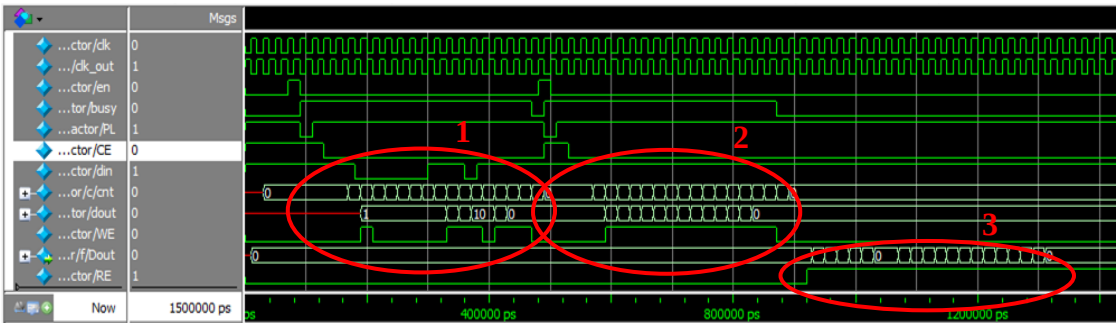
A HPTD detektor pad-jeit, láncokba rendezett shift regiszterek olvassák le és tárolják. Az FEE elektronika és a detektor kommunikációját a shift regiszter láncok beolvasása és vezérlése jelenti. A vezérléshez az órajelet (*clk_out*), a párhuzamos betöltés vezérlésére szolgáló jelet (*PL, parallel load – páhuzamos betöltés*), és a kiolvasás engedélyezésére szolgáló (*CE, clock enable – órajel engedélyezés*) jelet az FPGA szolgáltatja. A detektor shiftregiszter-láncai a Philips gyártmányú 74LV165A típusú shift regiszterekből épülnek fel [28], melyek párhuzamosan tölthetőek be és sorosan, bitenként olvashatóak ki. A detektor felől a 3.2 fejezetben leírtak szerint 20 szálon érkezik be az adat. Egy szál beolvasásáért egy verilog modul felelős (*compactor.v*). A modul a következő feladatokat látja el:

- *L0* trigger jel mintavételezését követően megfelelő vezérlőjeleket ad ki a shiftregiszterek felé (*PL, CE, clk_out*) a regiszterek betöltésére (*PL*) és a kiolvasásra (*CE*), majd
- elvégzi az adatok tömörítését és egy megfelelő méretű (9 bit széles) FIFO-ban tárolja a tömörített 9 bites adatokat (4.9 ábra).



4.9 ábra: Compactor modul blokk diagramja

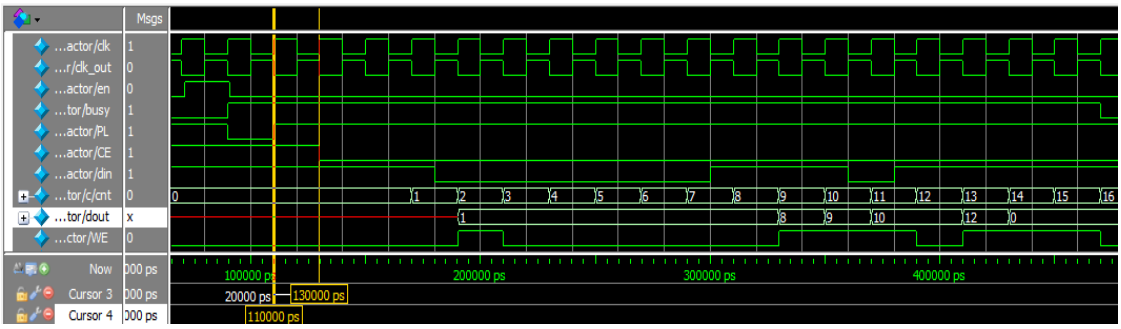
A modul a belső *50 MHz*-es rendszer-órajelre működik. A kimeneti órajel (*clk_out*) a belső órajel negáltja, hasonlóan a DDL interfész modulhoz. A modul működésének tesztelésére egy 9 bit széles FIFO-ból és a modulból álló rendszert állítottam össze (*testbench_compactor.v*). A beolvasandó adatmennyiség hossza a verilog modul egy paramétere. Az egyszerűbb ellenőrizhetőség kedvéért ezt a paramétert a szimulációhoz 12-re állítottam. A szimuláció során egy néhány egyest tartalmazó bitfolyam (1), egy tisztán egyesekből álló bitfolyam beolvasását (2), majd pedig a FIFO-ba beírt adatok ellenőrzését, a FIFO kiolvasásával szimuláltam (3) (4.10 ábra).



4.10 ábra: Compactor.v modul szimulációja

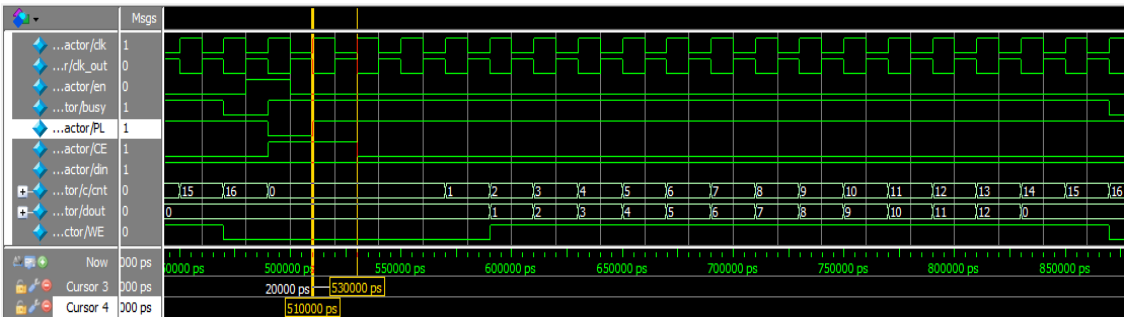
Az *L0* trigger jel mintavételezését követően kiadjuk a *PL* betöltő vezérlőjelet az adott shiftregiszter lánc regisztereinek. A *PL* impulzusnak *50 MHz*-es működési frekvencia mellett minimum *9 ns* szélesnek kell lennie (t_w) [28]. A *PL* impulzust a modul egy órajel hosszan, azaz *20 ns* hosszan aktívan tartja. A detektoron lévő adat több száz *ns*-ig stabil, így ez megfelelő működést eredményez [15]. A *PL* vezérlő jel felfutó éle és a *CE* vezérlő jel lefutó éle közti időnek minimum *8.5 ns* hosszúnak kell lennie (t_{rem}) [28]. Ezt az időt szintén *20 ns* hosszúra terveztem.

Az *L0 (en)* engedélyező jel mintavételezése után a modul kiadja a *busy* jelet, hogy a beolvasás alatt ne érkezhessen több beolvasásra igény (4.11 ábra). A *busy* jel kiadásával egy időben a *PL* impulzus is kiküldésre kerül, amit egy órajellel a *CE* jel is követ. Ezt a shift regiszter mintavételezi, majd a következő órajelre kiküldi az első bitet. Amennyiben a kiküldött bit (*din*) egyes, a belső számlálónk által számolt sorszám megjelenik a kimeneten (*dout*) egy FIFO-t vezérlő *WE* (*Write Enable – írás engedélyezés*) jellel együtt és így beírásra kerül a FIFO-ba. A bemeneten a következő bitfolyam jelenik meg: {1,0,0,0,0,0,0,1,1,1,0,1}. Az egyesek sorszáma egytől számozva: {1,8,9,10,12}. Az események elválasztására két darab ’0’ adat is beíródik, minden olvasás végén. Mivel az egyesek sorszáma 1-től kezdődik, ez nem okoz gondot, később viszont fontos szerepe lesz a memória szervezésében.



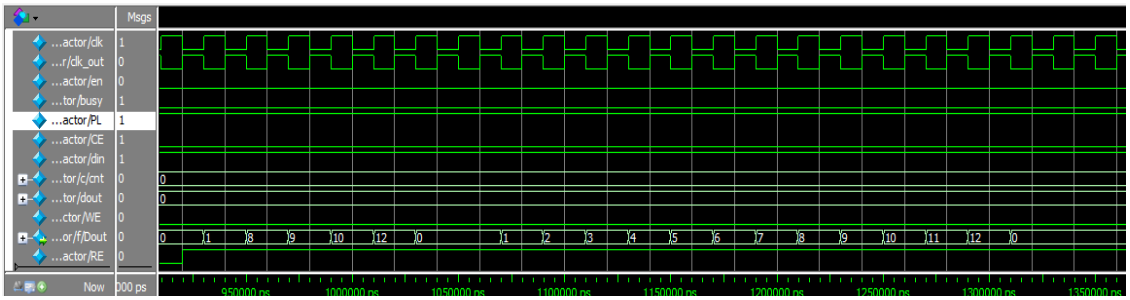
4.11 ábra: Detektor beolvasás

Az utolsó bit letárolását követően a *busy* jel ismét passzív állapotba kerül, jelezve, hogy a modul kész újabb *L0* fogadására. Az újabb *L0* azonnal meg is érkezik, a bemeneten (*din*) pedig csak egyesek várakoznak (4.12 ábra). Az újabb beolvasás során, 1-től 12-ig tárolunk le számokat, hiszen minden bemeneti érték egyes volt.



4.12 ábra: Újabb beolvasás

A két lezáró ’0’ adat ismét beírásra kerül a beolvasás után. A két egymást követő beolvasás után a FIFO memóriában a következő értékeknek kell szerepelniük: {1,8,9,10,12,0,0,1,2,3,4,5,6,7,8,9,10,11,12,0,0}. Ezt a FIFO-t a *RE* (*Read Enable – olvasás engedélyezése*) jel magas állapotba állításával kiolvashatjuk. A 4.13 ábra a kiolvasás eredményét mutatja, mely megegyezik a várt értékekkel:

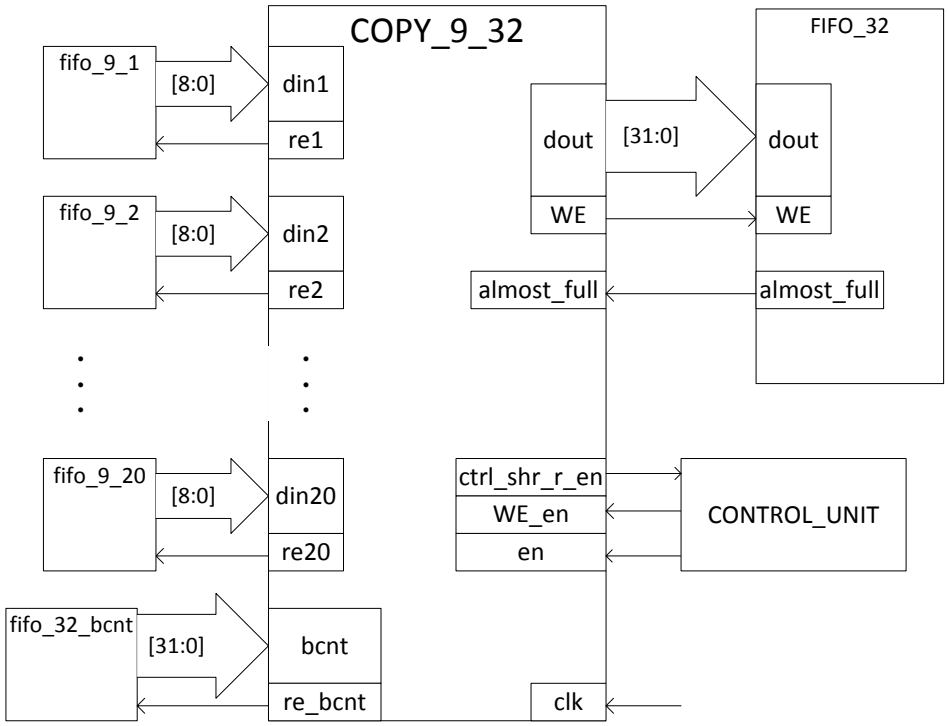


4.13 ábra: FIFO memória kiolvasása

4.4 Memória

A tömörítés miatt a tárolandó adatok mérete eseményenként változó. Illetve míg a DDL buszon az adatokat 32 bit széles formátumban küldjük tovább, addig a tömörítés után 9 bit széles adatokat tárolunk el. Ez mindenképpen egy bonyolultabb memória-szervezést igényel. Mivel a letárolás és adatküldés FIFO rendszerű, a memória szervezése is kényelmesen megoldható FIFO-ként konfigurált memória blokkok implementálásával. Ebben a fejezetben az adat tárolásának és küldésre való elkészítésének módját, illetve az ehhez tartozó verilog modulok működését ismertetem.

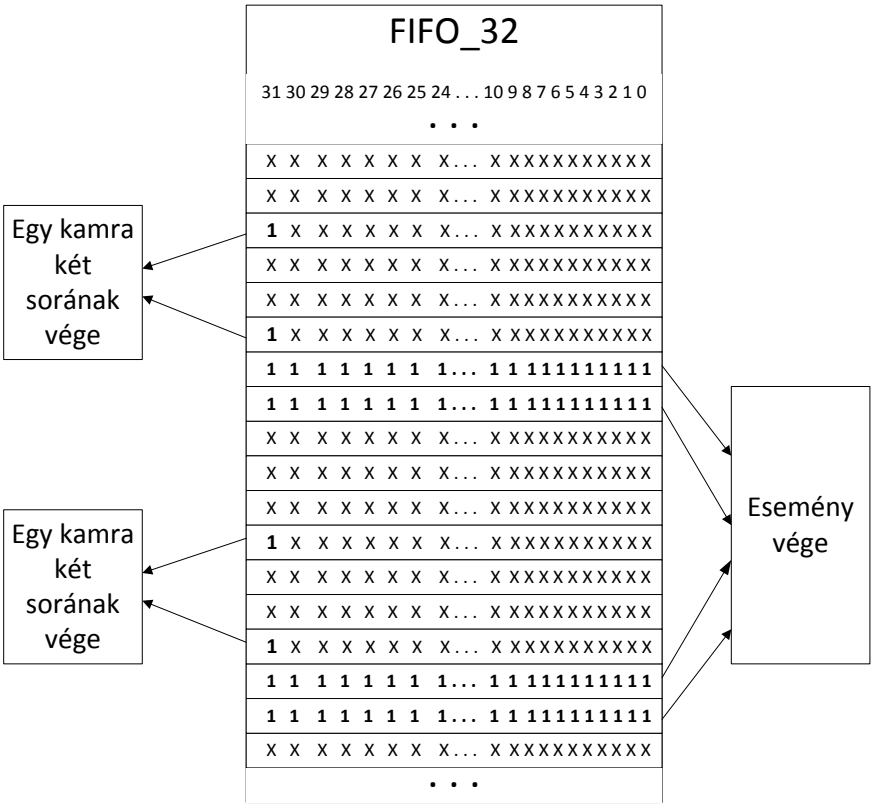
Az adatbeolvasást végző *compactor.v* modul szálanként egy 9 bites és 512 mély FIFO memóriába helyezi a tömörített adatokat, mivel így pontosan egy memória blokkot használunk el szálanként az adattárolásra. Ez összesen 20 darab, FIFO-ként konfigurált memória blokkot jelent. Ezen kívül a maradék 4 memória blokk is felhasználásra kerül. A DDL kommunikációt végző egység (*DDL_interface2.v*) egy 32 bites és szintén 512 mély FIFO-ból (*FIFO_32*) küldi az adatokat a SIU egység felé, mely két memória blokkból épül fel. A *bunch_event.v* modul egy 32 bit széles FIFO-ba (*fifo_32_bcnt*) írja a *bunch_crossing* és *event_counter* értékeket mely szintén 2 memória blokkból épül fel. Így az eszközben található összes memória blokkot felhasználom. A 9 bites tömörített adatoknak 32 biten kiküldhetőnek és konzisztensnek kell lennie. Ezért kiküldés előtt egy 32 bites FIFO memóriába (*FIFO_32*) másolódnak át. Ezt a másolást a *copy_9_32.v* modul végzi (4.14 ábra).



4.14 ábra: A *copy_9_32.v* modul és környezete

A *compactor.v* blokkok utáni 9 bites tárolókban az előző fejezetben említett két ’0’ adat jelzi két esemény határát. Úgy tűnhet, a két adat letárolása feleslegesen foglalja a helyet a memóriában, de az adatkiküldési szekvenciát nagyban gyorsítja, hogy az első ’0’ adat beolvasása után egy ’dummy’, jelentés nélküli adatot is beolvashatok, míg kiértékelem az első nullát és leállítom a beolvasást. A másoló modul, mind a 20 FIFO tárolón végig megy, és az eseményhatárokig átmásolja az adatokat a 32 bit széles *FIFO_32*-be a

következő módon: minden három 9 bitet egymásba fűzve, egy nullákkal 32 bitre kiegészített, de valójában *3 * 9 = 27* értékes bitet tartalmazó adatot készít. Ha egy modulban az adatok száma (a ’0’ esemény elválasztó adatokat nem beleértve), 3-mal osztva 2-t vagy 1-et ad maradékul, úgy aszerint a 18, vagy a 9 bitet egészíti ki 32-re nullákkal. Az utolsó adat (álljon az 9, 18 vagy 27 értékes bitből) legfelső bitje ’0’ helyett ’1’-re állítódik. Ezzel jelezve, hogy az eddig szereplő adatok, egy esemény és egy kamrához tartozó két sor adatainak felelnek meg. A ’0’ esemény elválasztó adatok így nem kerülnek átmásolásra. Ha a szekvencia minden egyes 9 bites FIFO-n végigment, akkor a fifo_32_bcnt, szintén 32 bites FIFO-ból is két adatot átmásol, azaz az adott eseményhez tartozó *bunch_crossing* és *event_counter* értékeket. Ezután pedig két, tisztán 32 darab egyesből álló adatmezőt helyez el, mely a *DDL_interface2.v* modulnak jelzi a kiolvasás során az esemény határ végét. A FIFO_32 memória szerkezetét a 4.15 ábra szemlélteti:

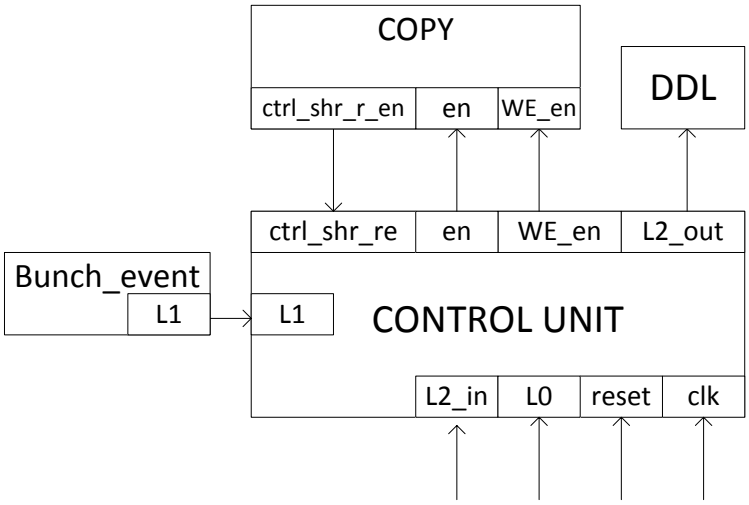


4.15 ábra: FIFO_32 memória szervezése

A memória átmásolását a *control_unit.v* és a *FIFO_32* modulok vezérlik. A másolást ugyanis fel kell függeszteni, amint a *FIFO_32* blokk megtelik, és csak akkor folytatható, ha újra felszabadul benne hely, azaz ha például egy kiküldés megkezdődik.

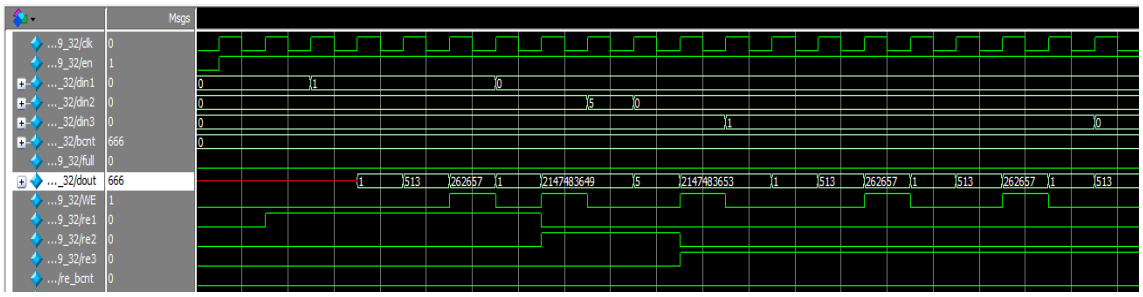
Ezt az információt a *FIFO_32* felől, a modul *almost_full* jelző bit kimenete szolgáltatja a *copy_9_32.v* felé (4.14 ábra).

A másolásnak még egy vezérlő feltétele van, melyet a *control_unit.v* vezérlő egység generál. Ha ugyanis nem érkezik időkereten belül *L1* trigger impulzus, akkor a már beolvasott adatok törlendőek. Ez a FIFO blokkok másolás nélküli kiolvasását jelenti, melyet szintén a *copy_9_32.v* modul végez, azzal a különbséggel, hogy a *FIFO_32* egységhez tartozó *WE* kimenete végig ’0’. A szekvencia ezen a különbségen kívül teljesen megegyezik az átmásolás folyamatával. A *control_unit.v* ezt a következőképpen vezérli. Az egység tartalmaz egy 1 bites FIFO-t (*one_bit_shr.v*), melybe az *L1* időkeret végén egy bit íródik annak függvényében, hogy érkezett e *L1* trigger. Ha igen ez a bit ’1’-es, amennyiben nem, ez a bit ’0’. A másolás akkor kezdődik meg, ha ez az egy bites FIFO nem üres. Ezt a FIFO-hoz tartozó *empty* bit negáltjaként kivezetett *en* kimenet szolgáltatja (4.16 ábra). Amint a másoló modul ezt 1-nek mintavételezi, megkezdi a másolást. Az egy bites FIFO kimenete egyben a *control_unit.v* modul *WE_en* kimenete is. Ennek függvényében történik másolás, vagy csak kiolvasás. Az egy bites FIFO olvasását a másoló modul vezérli a *ctrl_shr_r_en* kimenetén keresztül. Ez gyakorlatilag az egy bites FIFO olvasást engedélyező bemenete. Így a másoló modul az egy bites FIFO-t minden másolási szekvencia végén léptetni tudja. A *control_unit.v* továbbá megkapja az *L2* trigger impulzust is bemenetként (*L2_in*), amely alapján előállítja a *DDL_interface2.v* modul küldés indítást jelző, vezérlő bemenetét (*L2_out*).



4.16 ábra: Vezérlő egység kommunikációja

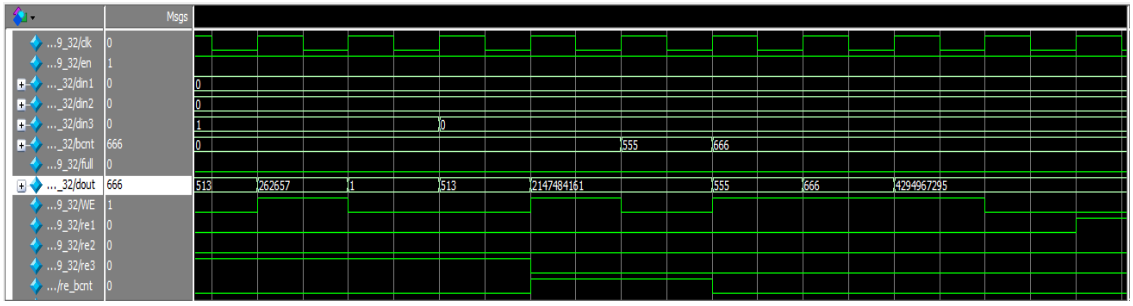
A másoló modul szimulációját három darab 9 bites FIFO memóriából, egy 32 bites FIFO memóriából és a *copy_9_32.v* modulból építettem fel, mely három 9 bites FIFO bementre volt konfigurálva (*testbench_copy_9_32.v*). A szimuláció során tesztelnem kellett az alapvető működést, az *en* engedélyező jelre való indulást, a *WE_en* vezérlőjel működését és az *almost_full* megszakítását. Ezen funkciók bemutatása hosszadalmas lenne, így néhányat kiválasztva belőlük, mutatom be a *copy_9_32.v* modul működését. A szimuláció a három 9 bites FIFO feltöltésével kezdődik. A minta adatoknak alkalmasaknak kellett lenniük az egyszerű ellenőrzésre, de a lehető legtöbb eset vizsgálatát is magában kellett foglalniuk (például csak esemény elválasztó ’0’-kat tartalmazó FIFO-k). Az összes vizsgált esetet itt nem szándékozom a nagy terjedelem miatt bemutatni. A következőkben bemutatásra kerülő szimulációban, az első FIFO tartalma {1,1,1,1,0,0}, a másodiké {5,0,0}, a harmadiké pedig {1,1,1,1,1,1,1,1,0,0} volt. A *fifo_32_bcnt* tartalma pedig {555, 666} volt. Amint az engedélyező jel megjelenik, a modul kiadja az első FIFO olvasásra engedélyező jelét (*re1*). Az első FIFO ezt a következő órajel felfutó élén mintavételezve, kiadja a legrégebben bele írt 9 bites adatot, amit a másoló modul az ezt követő felfutó élre mintavételez. Amint összefűződött három 9 bites adat, a *WE* írás-engedélyezés jellel beírja a *FIFO_32* tárolóba. Látható, hogy három 9 biten ábrázolt egyes összefűzése és nullákkal 32 bitre való kiegészítése ’*262657’*-et ad (4.17 ábra). Az első tároló utolsó egyesét szintén 32 bitre egészíti ki, de mivel ez az utolsó adat, ezért a legfelső, egyébként használaton kívül lévő bitet ’1’-be állítva ’*2147483649’* adódik (4.17 ábra).



4.17 ábra: Másoló modul 1. ábra

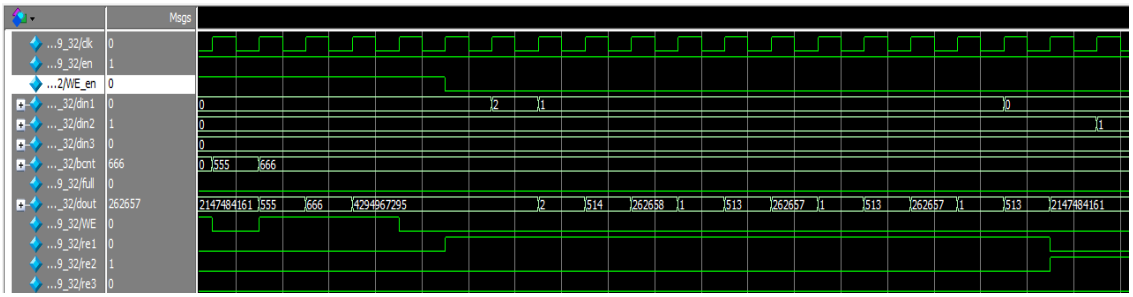
Ezt követően a második tároló kap engedélyező jelet az olvasásra (*re2*), miközben az elsőtől elvesszük azt. A második csak egy darab 5-ös számot tartalmaz így, a kiküldendő adat, ennek a 32-bitre való kiegészítése és a legfelső helyi érték ’1’-re állítása. A *WE*-el kiírt eredmény így: *2147483653*. A harmadik tároló sok egyest tartalmaz így a *262657*-es szám többször is átíródik a nagy FIFO-ba. A harmadik tároló

egység kiolvasása két egyessel végződik, azaz két egyes összefűzéséből és a legmagasabb bit ’1’-re állításából adódik a *’2147484161’* érték (4.18 ábra). Ezt követően a *fifo_32_bcnt* tároló kap két órajel periódusnyi, olvasásra engedélyező jelet, amivel mind a két értéke (*555* és *666*) átmásolódik. A szekvencia utolsó kiírandó adata az esemény határt jelző *32’hFFFFFFFF* adat, mely decimálisan a *’4294967295’* érték (4.18 ábra).



4.18 ábra: Másoló modul 2. ábra

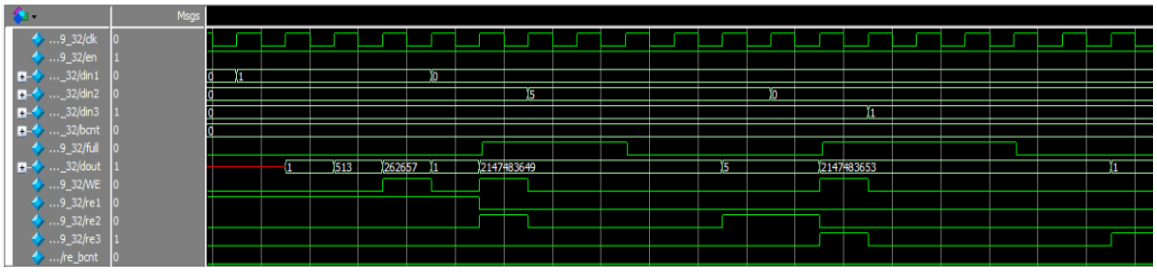
A szekvencia befejezése után, ha az *en* engedélyező jel magas, azaz a *control_unit* egy bites vezérlő FIFO-ja még nem üres, újabb másolási szekvencia indul. Ha az egy bites FIFO következő értéke ’0’-t tartalmaz, azaz a másoló modul *WE_en* bemenete ’0’, ugyanúgy előállnak a másolandó, összefűzött értékek, de a *WE* írást engedélyező jel nem jelenik meg. Így a FIFO tárolókból még egy eseményt kiolvastunk, de nem írtuk be sehova, azaz eldobtuk, töröltük a memóriából az adatokat. Ez a jelenség figyelhető meg a következő ábrán (4.19 ábra). (Közben az előző 9 bites FIFO tárolók újabb adatokkal lettek feltöltve. Az első például: {2,1,1,1,1,1,1,1,1,1,1}).



4.19 ábra: Másoló modul 3. Ábra

A másolás *copy_9_32.v* harmadik vezérlő bemenete a *full.* Ez a megnevezésével ellentétben a *FIFO_32* tároló *almost_full* (majdnem tele van) kimenete. E jel magas állapotának mintavétele a másoló modult, annak bármely állapotában a másolás felfüggesztésére készteti. A következőkben erre mutatok néhány példát. A példák nem

merítik ki az összes lehetőséget, ez azonban nem azok vizsgálatának hiányát jelenti. Minden lehetőséget ellenőriztem. A következő ábrán (4.20 ábra) két küldés közbeni megszakítást láthatunk. A megszakításból látható, hogy a *full* jelentése valójában *almost_full* hiszen csak az garantált, hogy a megjelenése után egynél több adat már nem kerül beírásra. Ez az állapot azonban akármilyen hosszan fenn állhat. Jelen példában 3 és 4 órajelig áll fenn a küldés félbeszakított állapota. Látható azonban, hogy a kiírt értékek megegyeznek az előző, megszakítás nélküli értékekkel. Azaz a megszakítás ellenére a megfelelő adatok kerülnek átmásolásra.

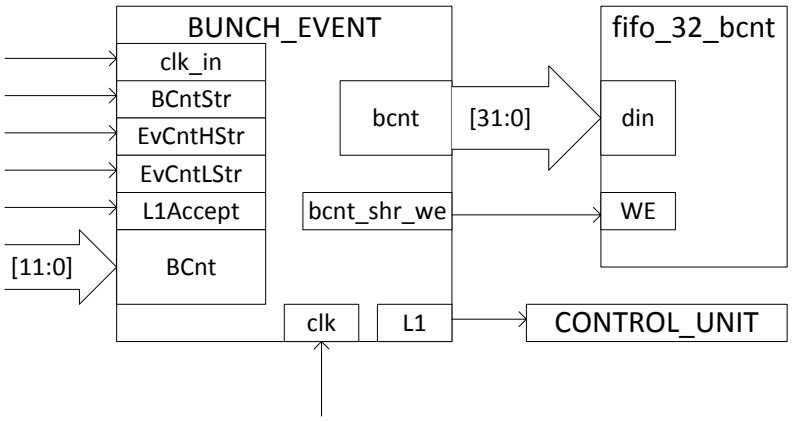


4.20 ábra: Másolás megszakítása

4.5 Trigger rendszer

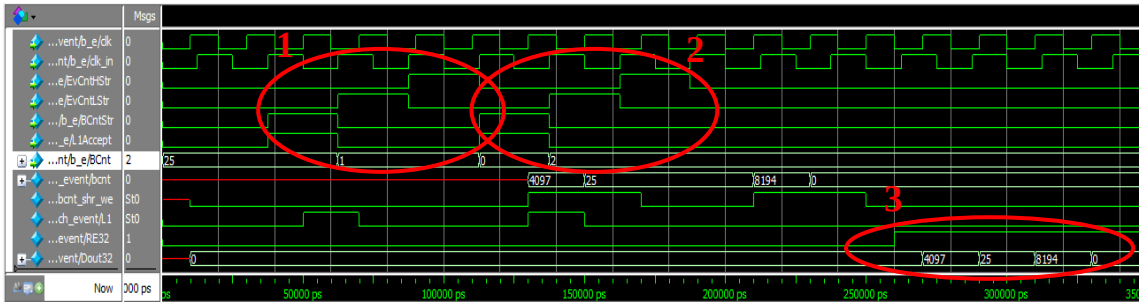
Amint azt már a 3.4 fejezetben ismertettem, a HPTD detektor az ALICE trigger rendszer része is. Az FEE detektorról való beolvasását, adattárolását és adattovábbítását három trigger jel vezérli (*L0, L1* és *L2*). Ebben a fejezetben ezen trigger jelek okozta belső vezérlést és az *L1* trigger beolvasását mutatom be.

Az *L1* trigger a 3.4.2 fejezetben bemutatott protokoll szerint kapcsolódik az FEE kártyához. Az FEE és a TTCrx egység közti kommunikációt a *bunch_event.v* verilog modul vezérli (4.21 ábra). A kommunikáció 40.08MHz-es órajelét a TTCrx modul szolgáltatja (*clk_in*). Vezérlő jelek (*BCntStr, EvCntHStr, EvCntLStr*) segítik a 12 bit széles buszon (*BCnt*) megjelenő adatok értelmezését. A modul a beolvasott 12 bit széles *bunch crossing counter* és a 24 bit széles *event counter* értékeket 32 bitre, nullákkal kiegészítve egy 32 bites FIFO-ba tárolja le. Azaz egy *L1Accept* érkezése során két 32 bites adat kerül letárolásra a *fifo_32_bcnt.v* modulba. A további vezérlések miatt, az *L1* beérkezéséről a *bunch_event.v* modul egy központi vezérlő egységet (*control_unit.v*) is értesít az *L1* kimenetén. A 4.21 ábra a *bunch_event* modult és a TTCrx illetve a többi modullal való kommunikációját ábrázolja.



4.21 ábra: A *bunch_event.v* modul és környezete

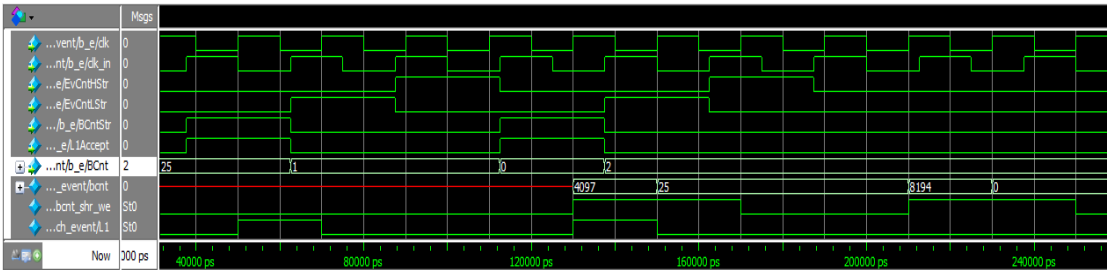
Két egymás utáni különböző *L1* eseményt (1, 2) és a FIFO kiolvasását (3) szimuláltam (4.22 ábra).



4.22 ábra:Testbench_bunch_event.v szimulációja

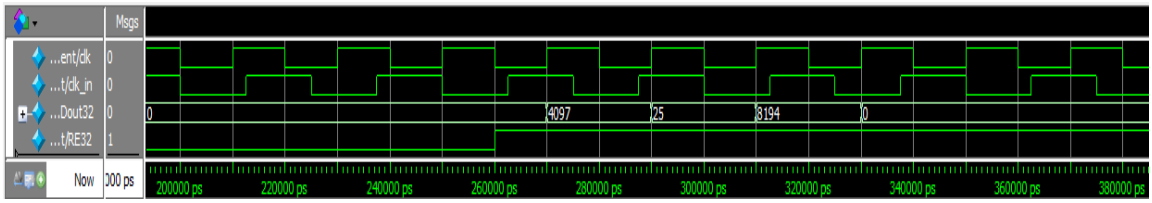
A modul a bemeneti órajelre (*clk_in*) mintavételezi a TTCrx oldali bemeneteit, de a kimenetét egy a belső, *50 MHz*-es órajel frekvencián működő kimeneti regiszterbe tárolja (*bcnt*). A FIFO memóriába ezen regiszter értéke a *bcnt_shr_we* vezérlő kimenet hatására kerül beírásra.

Az első *L1* esemény során *BCnt* busz a következő értékeket vette fel (4.23 ábra): {25, 1, 1}. A protokoll szerint az első érték a *bunch_crossing* számláló értékét, a második az *event_counter* alsó 12 bit értéke, míg a harmadika az *event_counter* felső 12 bitjét tartalmazza. Így *event_couter = 4097 és bunch_crossing = 25.* A második beolvasás során, mely az elsőt azonnal követi a *BCnt* busz a következő értékeket tartalmazza: {0, 2, 2}. Az előzőek alapján *event_counter = 8194* és *bunch_crossing = 0.* A folyamatot a 4.23 ábra mutatja. Az *L1* kimenet az *L1Accept* bemenet mintavételezésével azonnal megjelenik.



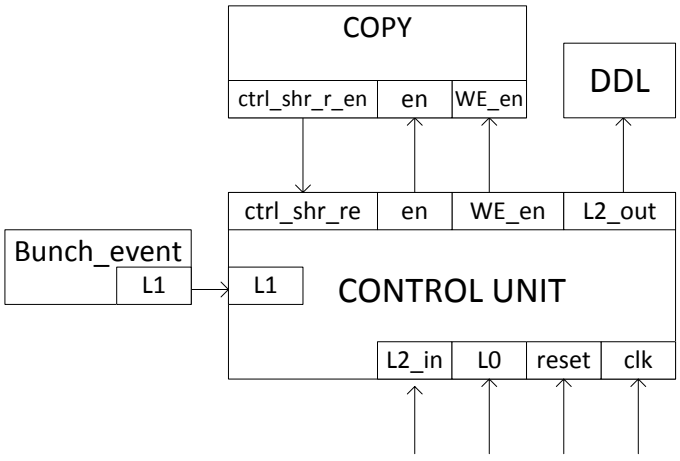
4.23 ábra: TTCrx kommunikáció

A FIFO-ba írt értékek helyességét ismét a FIFO kiolvasásával ellenőriztem. A FIFO *RE32* jelének magasba állítására, a belső órajel felfutó éleire a FIFO kimenetén (*Dout32*) a várt értékek jelentek meg: {4079, 25, 8194, 0}. Ezt a 4.24 ábra mutatja.



4.24 ábra: fifo_32_bcnt kiolvasása

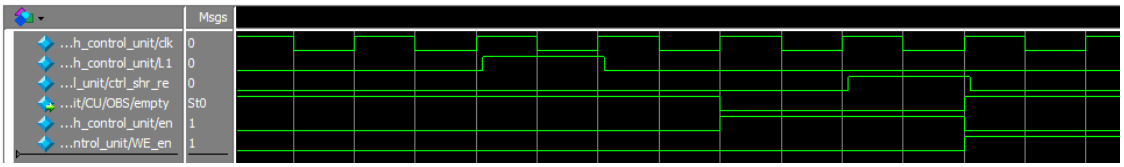
Az *L0* trigger jel, amint azt már a detektorról beolvasó *compactor.v* modul fejezetében, a 4.3 fejezetben írtam, a *compactor.v* beolvasó/tömörítő modulok kapják meg. Ezen kívül a *control_unit.v* vezérlő egység is bemenetként kezeli, mivel a trigger időkereteket ez alapján méri, az egység egy belső számlálója. A *control_unit.v* kezeli az összes trigger bemenetet, így az *L2* adatkiküldés indítását jelző vonalat is:



4.25 ábra: *Control_unit.v* és környezete

A belső számláló L0 impulzus mintavételezésére indul. Ezzel egy időben a *compactor* modulok is megkezdik a beolvasást. Ha időkereten belül L1 trigger is érkezik a *bunch_event* modul felől az *L1* bemeneten, akkor a *one_bit_shr*, egy bites

FIFO tárolóba egy ’1’-es íródik. Amennyiben ez az első alkalom az *en* kimenet is egybe vált. Az *en* jel magas szintjét mintavételezve a másoló modul egy engedélyező impulzust ad az egy bites FIFO olvasására (*ctrl_shr_re*). Ezáltal a FIFO újra üres lesz, azaz az *en* újra nullába vált. A *WE_en* kimenet viszont egybe, mert a FIFO kimenete egy lesz (4.26 ábra). Időkereten kívüli *L1*-re az egy bites tárolóba ’0’ bit kerül, aminek hatására a vezérlő modul a *WE_en* egybe állításán kívül ugyanúgy reagál.



4.26 ábra: *Control_unit* válasza időkereten belüli *L1*-re

Ha időkereten belül érkezik impulzus az *L2_in* bemenetén, akkor a következő órajel periódusra már megjelenik egy impulzus az *L2_out* kimeneten. Amennyiben az időkeret *L2* impulzus érkezése nélkül jár le, újból *L0* váró állapotba kerülünk.

4.6 Teljes rendszerterv

A teljes rendszer felépítése nem egyértelmű az előzőek alapján, ezért ebben az alfejezetben a legfelső verilog modult (*top_module*) ismertetem, amely reményeim szerint, az egész rendszer működési mechanizmusára is rávilágít.

A detektorról való beolvasást szálanként egy-egy *compactor.v* egység végzi, melyekhez egy 9 bites tároló is tartozik. Ezeket az egységeket egy modulba rendezve (*data_read*) láthatjuk a 7.1 függelék fejezet első ábráján. A modul 20 bemenete (*din1, ... , din20*) a detektor felől érkező shiftregiszter láncok végeinek felelnek meg. A *compactor* modulok vezérlő kimeneteikkel képesek a saját shiftregisztereiket vezérelni, de mivel a beolvasás teljesen egyszerre történik és ugyanannyi ideig tart, elegendő a vezérlő jelek egyszeri kivezetése. Ez bármelyik modulból történhet, az ábrán ez az első modulból valósul meg. Az *L0* vezérlő bemenetet minden *compactor.v* egység megkapja. A *busy* jel a többi, shiftregisztereket vezérlő jelekhez hasonlóan csak az első modulból kerül kivezetésre. Az egész modul a belső órajelre működik, melyet szétosztottan minden egység megkap a *DATA_READ* modulon belül. A modul belső oldali be- és kimenetei a tárolók olvasását szolgálják.

Az teljes rendszer felépítését (*top_module*) a 7.1 függelék fejezet második ábrája mutatja. Itt jól látható, hogy az előbbiekben ismertetett *DATA_READ* egységgel a

copy_9_32 másoló modul kommunikál. Az *L0* trigger bemenetet az időkeretek számolásához a *control_unit* is megkapja. Eddig a rendszer újraindításáról, alapállapotba hozataláról nem beszéltem. A control_unit *reset* bemenete, minden FIFO tárolót alapállapotba hoz a működés megkezdéséhez. A belső állapotgépek alapállapotból indulnak, ehhez a belső regisztereknek kezdőértékek adhatók. A teljes rendszer a belső *50 MHz* órajel frekvencián üzemel, kivéve a *bunch_event* modul kommunikációját, mely a *clk_in* bemenetén érkező órajel szerint ütemezve kommunikál a TTCrx egységgel.

5 További feladatok

A digitális rendszerterv egy, még nem létező, de körvonalaiban megtervezett FEE kártyához készült. A kész hardver után még sok feladat vár a rendszer élesztéséig. Az FPGA terv fejlesztése azonban párhuzamosítható a hardverfejlesztéssel, így ebben a fejezetben nem csak a kész rendszer tesztelésére vonatkozó teendőket, de a digitális rendszerterv továbbfejlesztésére vonatkozó feladatokat is megemlítek.

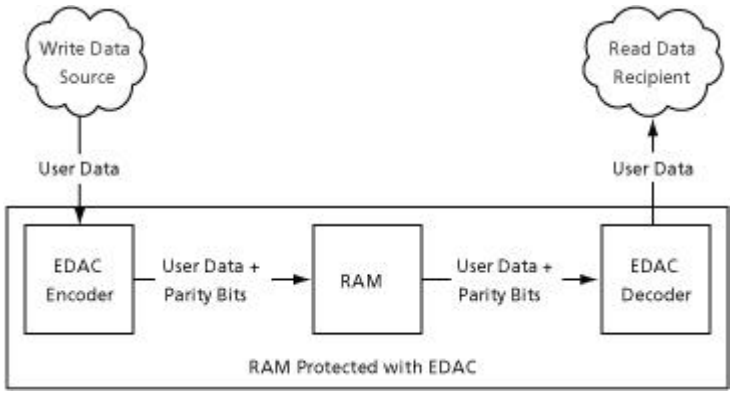
5.1 USB kommunikáció

A 3.1 ábra egy monitorozásra szolgáló egységet is tartalmaz. Egy monitorozásra is alkalmas USB kommunikáció kialakítása nagyban megkönnyítené a rendszer végső tesztelését és élesztését is. Ellenőrizni lehetne a detektorról való beolvasás helyességét, a letárolás és a tömörítés helyességét DDL protokoll nélkül, USB kommunikáción keresztül. Az újabb modul lehetőséget adna arra, hogy az adatokat a DDL interfészen kívül, egy USB interfészen keresztül is elküldjük egy személyi számítógép felé. Így könnyen ellenőrizhetjük az FEE működését. Ez a kommunikáció azért is fontos, mert lehetővé teszi, hogy az FEE kártya egyéb (nem az ALICE mérőállomáson történő) mérések felhasználására is alkalmas lehessen. A kommunikációt ebből a szempontból és a kész FEE kártya tesztelése szempontjából nem helyettesíti a JTAG interfész, mely az FPGA-n belüli rendszer helyes működésének vizsgálatára nyújt megoldást.

5.2 Sugárzás elleni tesztelés

A front-end elektronika kártya közvetlenül a detektor térben helyezkedik el, így biztosan sugárzásnak lesz kitéve. Ebben a környezetben való helyes működéshez előzetes teszteket kell majd végezni. A biztosabb működéshez egy hardverben redundáns digitális rendszert lehet kialakítani. Az adatok letárolásában, másolásában kialakított térbeli és időbeli redundancia és annak vizsgálata pontosabb képet ad a sugárzás veszélyeiről. A sugárzásnak kitett rendszer működésének vizsgálatához szintén sokat segít egy előzőekben említett USB kommunikáció kialakítása.

Az Actel cég egy *CoreEDAC v2.1* RTL[10] szintű generátora a sugárzásra érzékeny SRAM memória védelmét biztosító kódoló/dekódoló logikát generál. Ez a szolgáltatás a ProASIC3E család eszközeivel is kompatibilis. A telekommunikációban használatos ECC (*Error-Correcting Code – hibajavító kód*) kódolási eljárások, redundáns adatok továbbításával lehetővé teszik, az adat fizikai rétegen való áthaladása során keletkezett hibák detektálását és az adatok helyreállítását is. Ha a „veszélyes" fizikai réteget a rendszerünk SRAM memóriájának feleltetjük meg, ahol részecskesugárzás hatására sérülhet a tárolt adat, akkor dekódoló/kódoló eljárásokkal írva és olvasva memóriánkat, itt is detektálhatjuk és helyreállíthatjuk a hibás adatokat. Ezeket a dekódoló/kódoló logikákat automatikusan generálja az Actel *CoreEDAC* generátora, akár külső, akár belső SRAM memóriákhoz (5.1 ábra). A generátor, a Hamming kódoláson alapuló SEC-DED (*Single Error Correction – Double Error Detection – egyszeres hibajavítás – kétszeres hibadetektálás*) kódolási és hibadetektálási eljárást alkalmazza [29]. Az eljárás a nevéből is adódóan nem képes kétszeres, két egymás mellett lévő bithiba kijavítására. Egy elegendően nagy energiájú részecske azonban két egymás mellett lévő memória cellát is átbillenthet. A Microsemi SRAM memória blokkjai azonban úgy vannak kialakítva, hogy egy RAM adatszó egymás melletti bitjei sose kerüljenek egymás melletti memória cellába. Így ez az egyszerű, kis késleltetésű, kis memória redundanciát okozó kódolás is elegendő biztonságot nyújt a sugárzás elleni védelemben. A modul használatához a memóriaszervezés újragondolása szükséges. Ennek kialakítása és megtervezése szintén a hardverfejlesztéstől független, önállóan végezhető feladat.



5.1 ábra: Példa a CoreEDAC modul használatára [29]

[10] RTL – *Register-Transfer Level* – egy digitális rendszer regiszterekből és logikai kapukból felépített absztrakciós modellje

Azonban ECC kódolással sem lehetünk teljesen biztosak, a sugárzás elleni védelemben, mivel az Actel cég adatlapjain, semmilyen konkrét, számszerű információ nem szolgál sem a biztonság minősítésérem sem a sugárzás mértékét. Így a sugárzó térben való vizsgálatok nélkülözhetetlenek.

5.3 Mintakeresési feladat

A nagy I/O kapacitás és a pontos, rugalmas interfész kialakítás lehetősége mellett, az FPGA eszközre esetleges mintakeresési feladatok miatt is esett a választás. A HPTD detektor 1.2.4 fejezetben említetett impulzus mérési feladata egy mintakeresésen alapul. A pad-ekről leolvasott becsapódási pontok koordinátái által, egy pontatlan részecske pálya határozható meg. Ezen pálya görbülete alapján, egy pontatlan, de használható becslést adhatunk a részecske impulzusára. A mintakeresési feladat adott impulzus-vágáshoz tartozó görbületeket azonosítását jelenti. A keresett minták alapján, a bejövő adatokon egy ablakozó jellegű mintakeresés megvalósítása a cél. A mintakeresési modul, a beérkező események mellé, a mintakeresés eredményére vonatkozó információkat is letárolná a memóriába. A mintakereső logika bonyolultsága, nagyban függ, a detektálandó minták számától és tulajdonságától, azonban az FPGA-ra való optimalizálás során, egy, a mintákhoz erőforrásaiban rugalmasan illeszkedő logikát kaphatunk.

6 Összefoglalás

Bár az általam tervezett front-end elektronikus kártya még csak körvonalaiban megtervezett, a firmware fejlesztések párhuzamosíthatók a kártya hardver-tervezésével. A digitális rendszer feladatban kiírt moduljait megterveztem és helyes működésüket szimulációkkal ellenőriztem. A munkám végeredményeképp, a feladatkiírásban szereplő modulokon kívül, egy előre rögzített trigger-értelmezés szerinti, teljes, működő rendszer született, mely nagyban segíti a FEE kártya jövőbeni fejlesztési munkálatait.

Irodalomjegyzék

[1] http://public.web.cern.ch/public/en/About/About-en.html *(2012 október)*

[2] http://hu.wikipedia.org/wiki/CERN *(2012 október)*

[3] http://public.web.cern.ch/public/en/Research/AccelComplex-en.html *(2012 október)*

[4] http://lhc-machine-outreach.web.cern.ch/lhc-machine-outreach/lhc_in_pictures.htm *(2012 október)*

[5] http://public.web.cern.ch/public/en/LHC/LHCExperiments-en.html *(2012 október)*

[6] http://www.isgtw.org/feature/isgtw-feature-alice-prepares-data *(2012 november)*

[7] http://en.wikipedia.org/wiki/Particle_detector *(2012 november)*

[8] Melegh Hunor: *„Front-end áramkör fejlesztése HPTD detektorhoz" – Diplomatervezés 1, Budapest, 2012 május 11.*

[9] CERN/LHCC 2000-001 ALICE: *„Technical Design Report of Time Projection Chamber" – 7 January 2000* https://edms.cern.ch/file/398930/1/ALICE-DOC-2003-011.pdf *(2012 december)*

[10] http://mxp.physics.umn.edu/s06/Projects/S06_ParticleShowerEnergy/theory.htm *(2012 december)*

[11] Jackson, John David: *„Klasszikus elektrodinamika" (Typotex Kft.,2004, Budapest)*

[12] F. Sauli: *„ Principles of operation of multiwire proportional and drift chambers" (1977)* http://cdsweb.cern.ch/record/117989/files/CERN-77-09.pdf *(2012 november)*

[13] http://ej.iop.org/images/0954-3899/39/12/123001/Full/jpg397605f2_online.jpg *(2012 december)*

[14] G.G. Barnaföldi, R. Bellwied, A. Di Mauro: *"A Very High Momentum Particle Identification Detector (VHMPID)" –Version 18.0, January, 10 2012* https://twiki.cern.ch/twiki/pub/Sandbox/VHMPIDLoI/vhmpidLOI_v18.pdf *(2012 november)*

[15] G.G. Barnaföldi, R. Bellwied, G. Bencze, M. Weber: *"A Very High Momentum Particle Identification Detector (VHMPID)" –Version 20.0, June, 15 2012* https://twiki.cern.ch/twiki/pub/Sandbox/VHMPIDLoI/vhmpidLOI_v20.1.pdf *(2012 november)*

[16] D. Varga, G. Hamar, G. Kiss: *"Asymmetric multi-wire proportional chamber with reduced requirements to mechanical precision", Nuclear Instruments & Methods*

in Physics Research Section A - Accelerators Spectrometers Detectors and Associated Equipment A648 (2011) 163-167

[17] ALICE Collaboration, *„Trigger, Data Acquisition, High Level Trigger, Control System Technical Design Report", Alice Technical Design Report, ALICE-DOC-2004-001 v.2, 2004.*

[18] György Rubin, Csaba Soós: *„Hardware guide for front-end designers/ALICE detector data link version 2.4", ALICE Internal note/DAQ, 2007*

[19] György Rubin, Csaba Soós: *„Hardware guide for front-end designers/ALICE detector data link, version 4.1", ALICE Internal note/DAQ,1998*

[20] György Rubin, Csaba Soós: *„Hardware guide for front-end designers/ALICE detector data link, version 11", ALICE Internal note/DAQ,2004*

[21] P. Moreira: *„TTCrq manual", 2005.*, *http://proj-qpll.web.cern.ch/proj-qpll/images/manualTTCrq.pdf, (2012 november)*

[22] J. Christiansen, A. Marchioro, P. Moreira, and T. Toifl: *„TTCrx Reference Manual – A Timing, Trigger and Control Reciever ASIC for LHC Detectors", 2004. Version 3.9.*

[23] Fehér Béla: *„Digitális rendszerek tervezése FPGA áramkörökkel"* (2012 október)
http://home.mit.bme.hu/~szanto/education/vimim286/FPGA_I_DigRendszTerv.pdf (2012 október)

[24] E. Dénes, A. Fenyvesi, A. Hirn, A. Kerék, T. Kiss, J. Molnár, D. Novák,C. Soós: *„ALICE DDL Radiation Tolerance Tests for the FPGA Configuration Loss" 10th Workshop on Electronics for LHC and Future Experiments, Boston, MA, USA, 13 - 17 Sep 2004, pp.384-388*
http://cdsweb.cern.ch/record/814707/files/p384.pdf (2012 december)

[25] Actel Documentation: *„ProASIC3E Flash Family FPGAs"*
http://www.actel.com/documents/PA3E_DS.pdf (2012 október)

[26] *http://lhc-machine-outreach.web.cern.ch/lhc-machine-outreach/images/lhc-schematic.jpg (2012 december)*

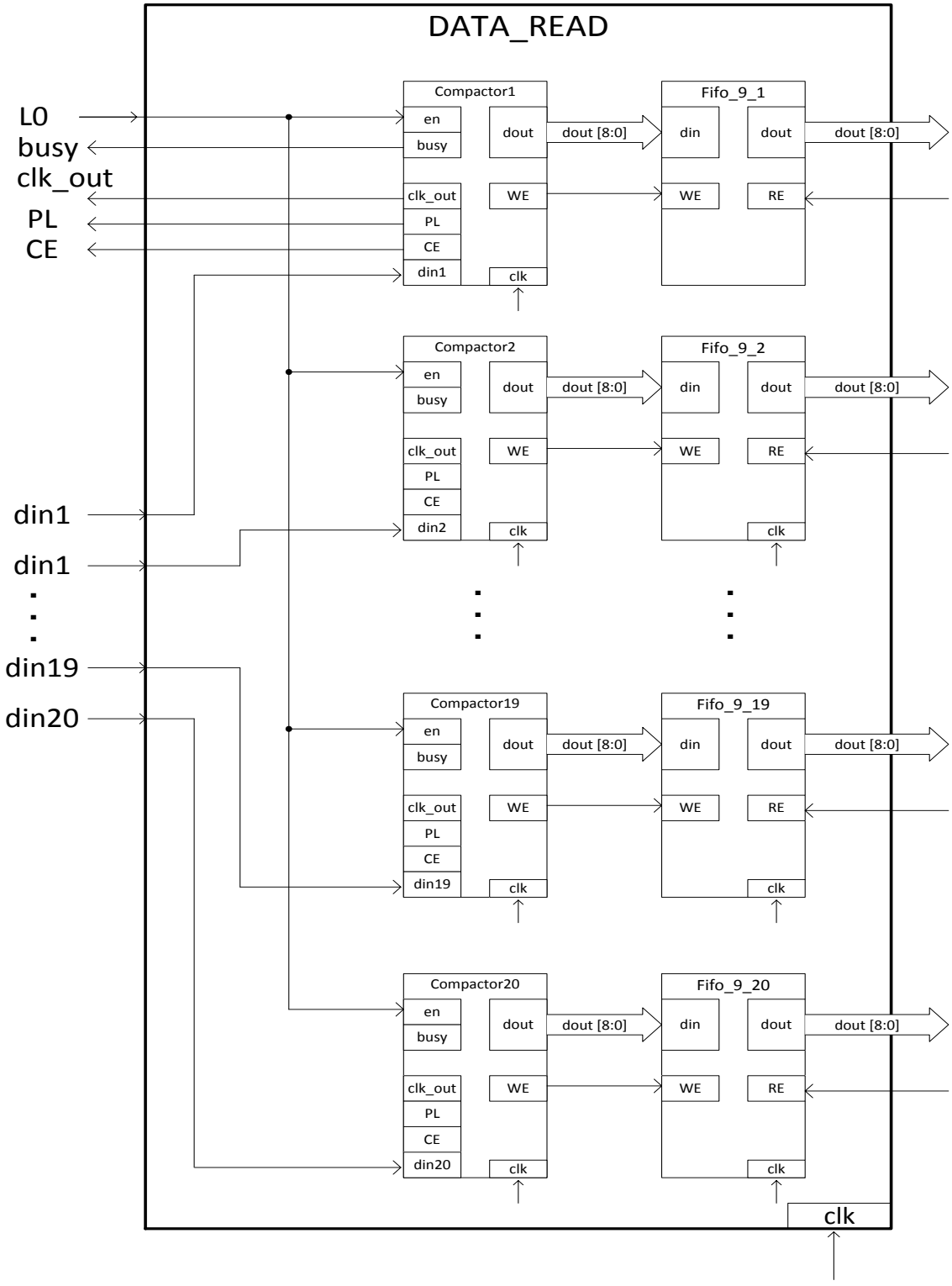
[27] Boldizsár László (Wigner RCP, Budapest): *„HPTD: The High-pT Trigger Detector for ALICE VHMPID, feasibility and Monte Carlo simulations",* 6th International Workshop High-pT physics at LHC 2011, Utrecht, Netherlands, 4 - 7 Apr 2011, pp.144-147
http://cdsweb.cern.ch/record/1471007/files/p144.pdf (2012 november)

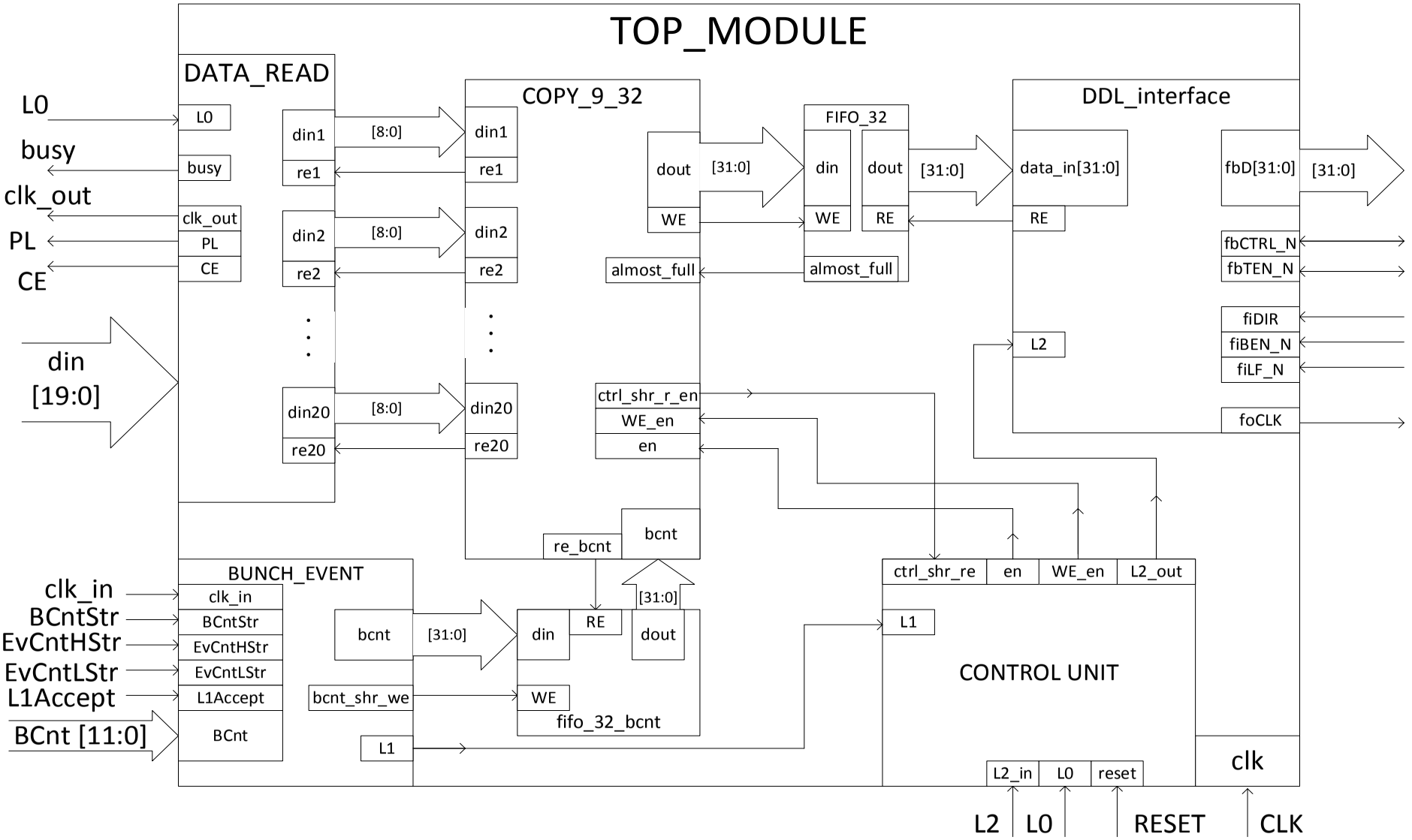
[28] Philips Semiconductors: *„74LV165A - 8-bit parallel-in/serial-out shift register – Product Specification Data Sheet", 2003.07.23*

[29] Microsemi: *„CoreEDAC v2.1 Handbook",*
http://www.actel.com/ipdocs/CoreEDAC_HB.pdf (2012 december)

7 Függelék

7.1 Ábrák




TOP_MODULE
DATA_READ
COPY_9_32
FIFO_32
DDL_interface
L0
busy
clk_out
PL
CE
din [19:0]
din1
re1
din2
re2
din20
re20
[8:0]
dout
[31:0]
WE
almost_full
din
RE
data_in[31:0]
fbD[31:0]
fbCTRL_N
fbTEN_N
fiDIR
fiBEN_N
fiLF_N
foCLK
L2
ctrl_shr_r_en
WE_en
en
re_bcnt
bcnt
BUNCH_EVENT
clk_in
BCntStr
EvCntHStr
EvCntLStr
L1Accept
BCnt [11:0]
BCnt
bcnt_shr_we
fifo_32_bcnt
L1
ctrl_shr_re
L2_out
CONTROL UNIT
L2_in
reset
clk
L2
L0
RESET
CLK

7.2 CD melléklet tartalma

- Verilog fájlok:
  - Verilog modulok:
    - *DDL_interface2.v*
    - *compactor.v*
    - *copy_9_32.v*
    - *bunch_event.v*
    - *control_unit.v*
    - *one_bit_shr.v*
    - *fifo.v* – 9 bites fifo
    - *fifo_32.v* - 32 bites fifo
    - *syncore_sfifo.v* – a SynCore[11] fifo generátor, fifo-szimulációkhoz szükséges modulja
  - Testbench verilog modulok:
    - *testbench_FIFO_DDL.v*
    - *testbench_compactor.v*
    - *testbench_copy_9_32.v*
    - *testbench_bunch_event.v*
    - *testbench_control_unit.v*
    - *testbench_one_bit_shr.v*
- Tömörítés szimlációjához tartozó fájlok:
  - *compression.m* – Matlab script
  - *100.txt* – beolvasott szimulációs adatbázis

[11] A verilog megvalósítás során, a memória blokkok FIFO-ként való konfigurálását, az ACTEL cég SynCore generátorával, automatikusan generáltam. (Az egy bites *one_bit_shr.v* FIFO kivételével).

Köszönetnyílvánítás

Szeretnék köszönetet mondani belső konzulensemnek, Szántó Péternek, külső konzulensemnek, dr. Dénes Ervinnek, Hamar Gergőnek és Melegh Hunornak, hogy munkámat hasznos tanácsaikkal segítették és konzultációs lehetőségek biztosításával, folyamatosan rendelkezésemre álltak.