The CTQD Board
(DRAFT)

S.L. Linn - Florida State University

1 Jan 2001

(Revised 19 Mar 2003)

Introduction

This document provides a description of the programming and operation of the CTQD boards in the D0 L1/L2 central track trigger (CTT) system. The CTT fits central fiber tracker (CFT) hits to form a track characterized by a transverse momentum (*Pt*) and azimuth angle (*Fi*). Through an identical readout system, the CTT reconstructs the *Fi* position of energy clusters in central pre-shower (CPS) system. An overview of the CTT is shown in Fig.1. The four CTQD boards receive LVDS signals from the eight CTOC boards, and transmit the processed track and cluster information to the L2 preprocessors with optical fiber G-links. The CTQD board is a generic double-wide daughter board (DWDB). Each CTQD has two Xilinx 600 FPGA chips in the U3 and U5 positions. The track and cluster algorithms are implemented on the U3 chip alone. The inputs and outputs conform to the protocols , unless otherwise noted. The algorithms were first described in the system architecture document.

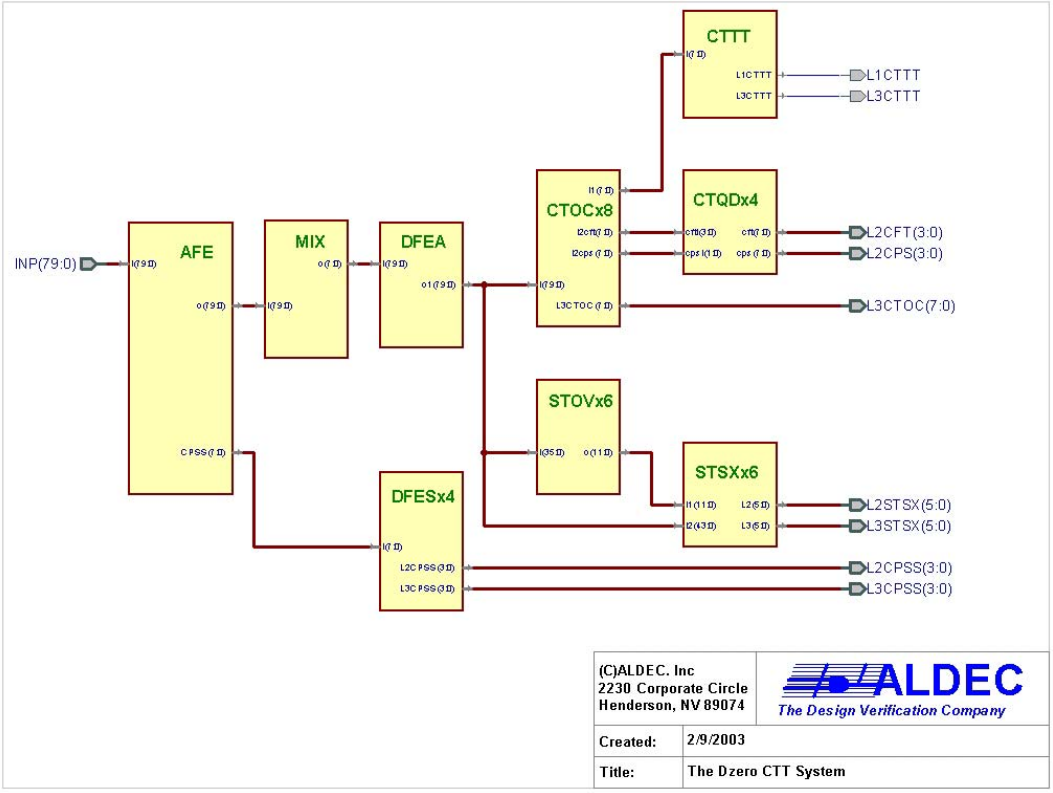

Figure 1 - Block diagram of the CTT system components: Analog Front End, MIXer, Digital Front End, Digital Front End Stereo, octant collector (CTOC), quadrant collector (CTQD), STT overlap (STOV), STT sextant (STSX), and CTT term collector.

Description of the algorithm

The CTQD board combines track and cluster information from two CTOC boards, and produces outputs which are ordered in *Pt* and *Fi* respectively. The inputs arrive ordered in *Pt* and *Fi*, so only a simple ordering is necessary. The most difficult and time-consuming part of the algorithm is the association of a track *Pt* with a cluster; because, upstream of the CTQD, this is done only when the track and cluster are in the same octant. The CTQD board must fill the *Pt* field of cluster records when it is missing. The situation where this occurs is shown in Fig. 2, where the cluster is resident in one octant, but the track hits are in the adjacent octant. In this case the cluster *Pt* bin is missing and all tracks in the boundary sector must be tested for possible matching.

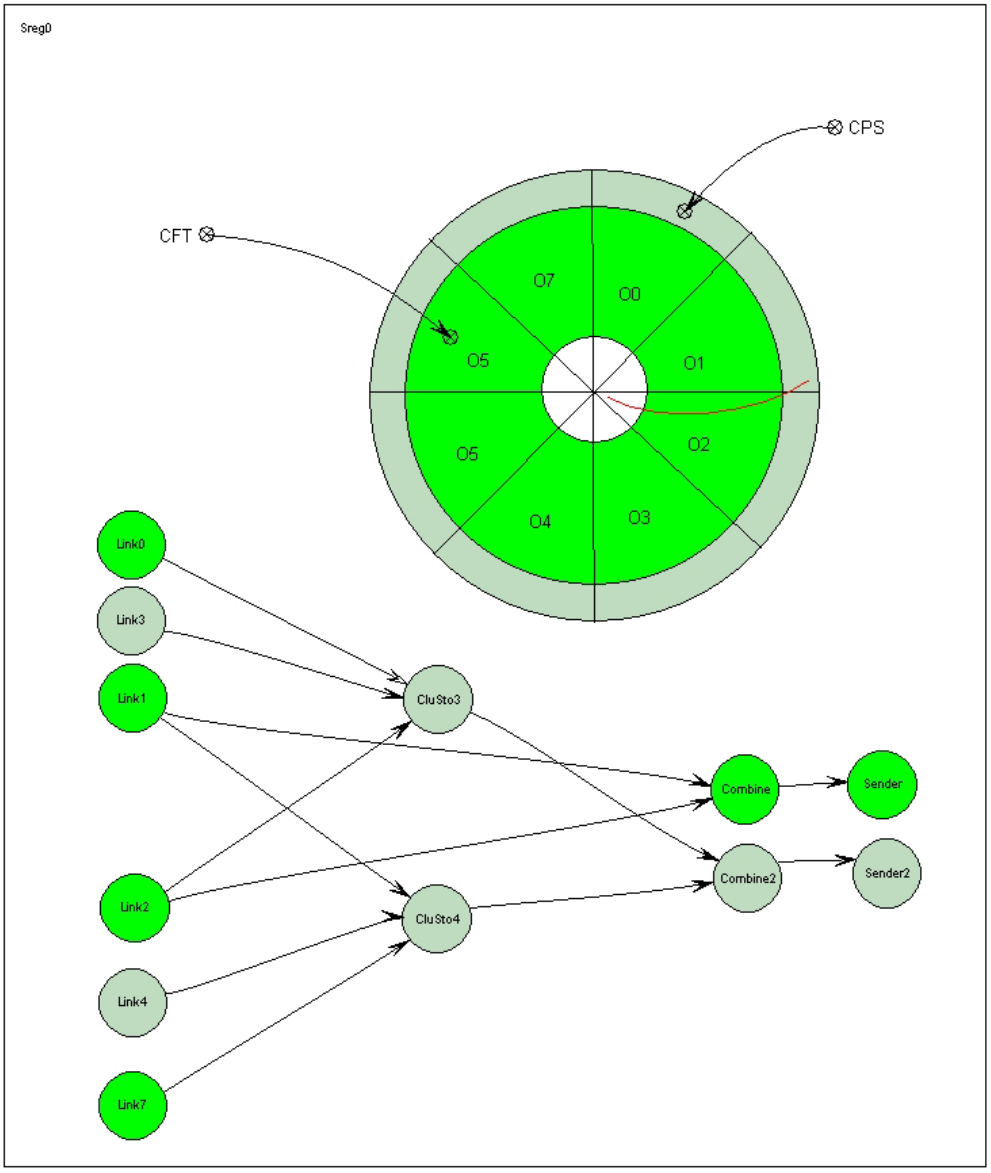

Figure 2 - Data flows in the CTQD for combining octants 0 and 1. Links 0,1,2,7 carry track information from like numbered octants. Links 3 and 4 carry cluster information for octants 0 and 1 respectively.

The data flow is shown graphically in Fig.2. For example, combination of information from octants O0 and O1 uses tracks from Links 0 and 1, and cluster information from Links 3 and 4. To effect the cluster corrections, it is also necessary to have tracks from O0 and O7 on Links 0 and 7 respectively. Data flowing between the major functional units is represented by the arrows in Fig.2. The six link receivers are shown on the left. Track information from two octants is ordered in *Pt* and transmitted to a level-2 sender, which formats and transmits the combined list. Cluster information is first corrected for possible "bend in" tracks, and then ordered in *Fi* before formatting and transmission to the L2 processors.

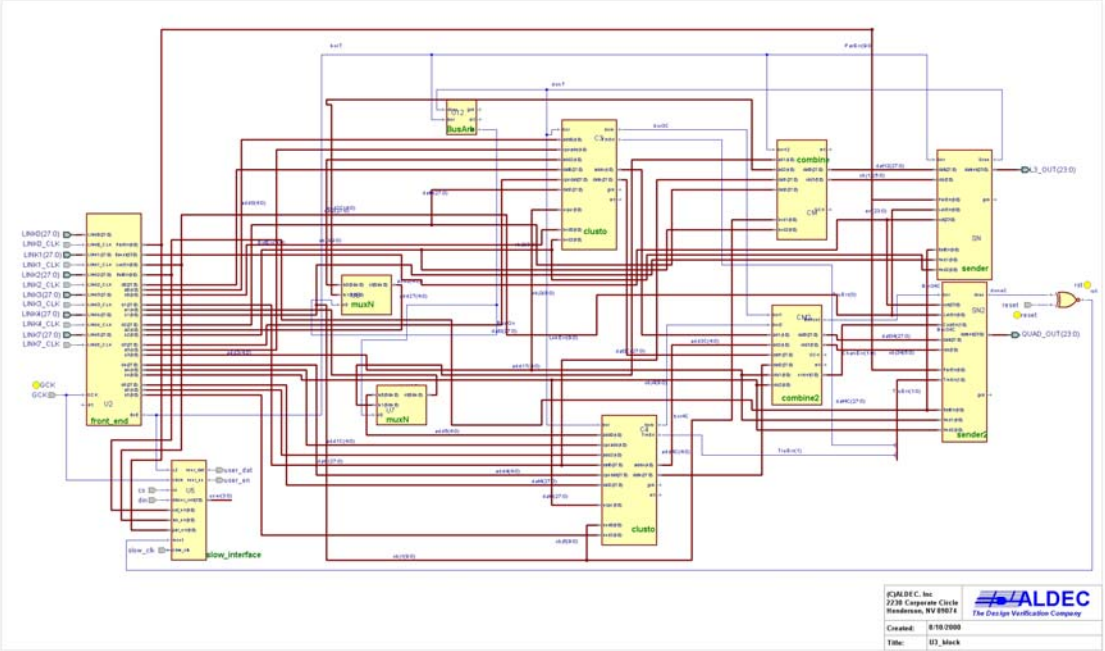

Figure 3 - CTQD top level block diagram

Firmware Implementation

Figure 3 is a block diagram (BDE) of the top-level design. The data flows primarily from left to right. The *front_end* module stores the event. The *clusto* modules are wrappers for the cluster *Pt* correction function. Each instantiation reads one octant of cluster data and two adjacent octants of track information. The corrected cluster frames are stored in block rams for later processing.

Ordering of track and cluster information is done by the finite state machines (FSM) *sort*, which is contained in the BDE *combine* and *combine2* respectively.

The modules *sender* and *sender2* are FSM that construct the six L2 G-link header frames, two trailer frames, 2x the number of tracks or clusters, and pad frames. The module *slow_interface* is responsible for slow monitoring, and is a modification of Olsen's *L1CTOC_interface*.

What follows are block diagrams and finite state machines, which illustrate the main functions of the implementation. If more detail is required the reader should RTFC.

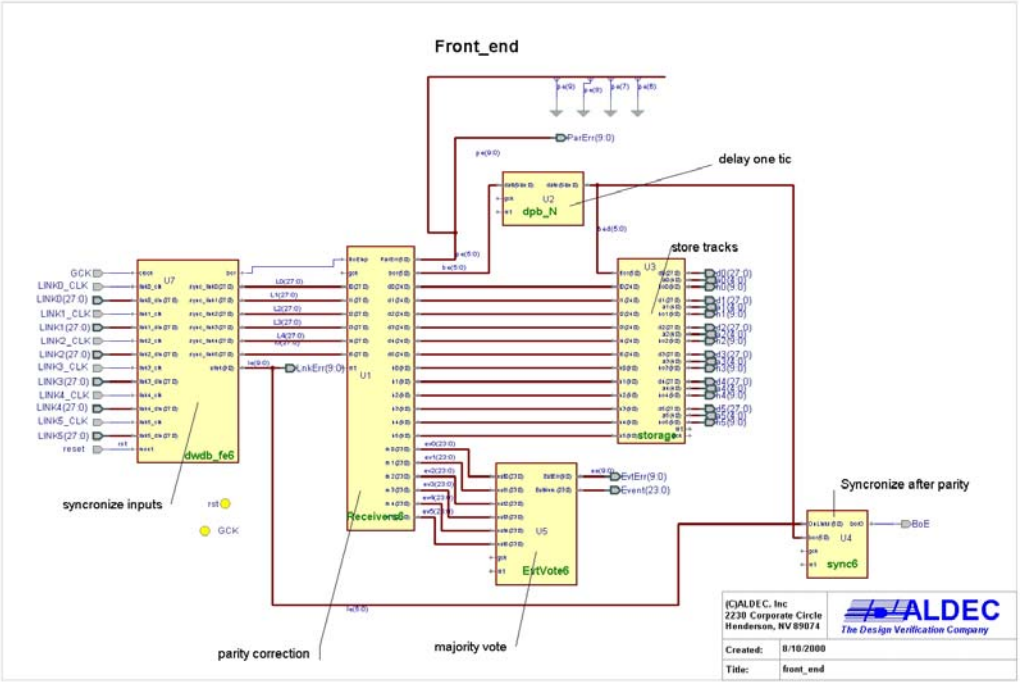

Figure 4 - Block diagram of front_end.

Block diagram *front_end* consolidates reception *(dwbd_fe), correction (*Receivers6), and storage (*storage*) of the input link data. The module *EvtVote6* determines the event number (tic & turn), and *sync6* re-syncronizes the time structure of all links after parity correction, which must wait till the end of each event stream. First, each link is synchronized to the global board clock using the Olsen's method, and missing link information is stored for later use. Then the data frames from each link are corrected for parity errors. The turn and rotation for the event are determined by a majority vote of links0, 1, and 2, and all links are compared to the winner. Finally the data frames are stored sequentially in a block ram where they can be accessed by the other functional units. Since parity correction requires reading all data frames from a link, a final begin of event (*BoE*) signal is created when all links have finished.

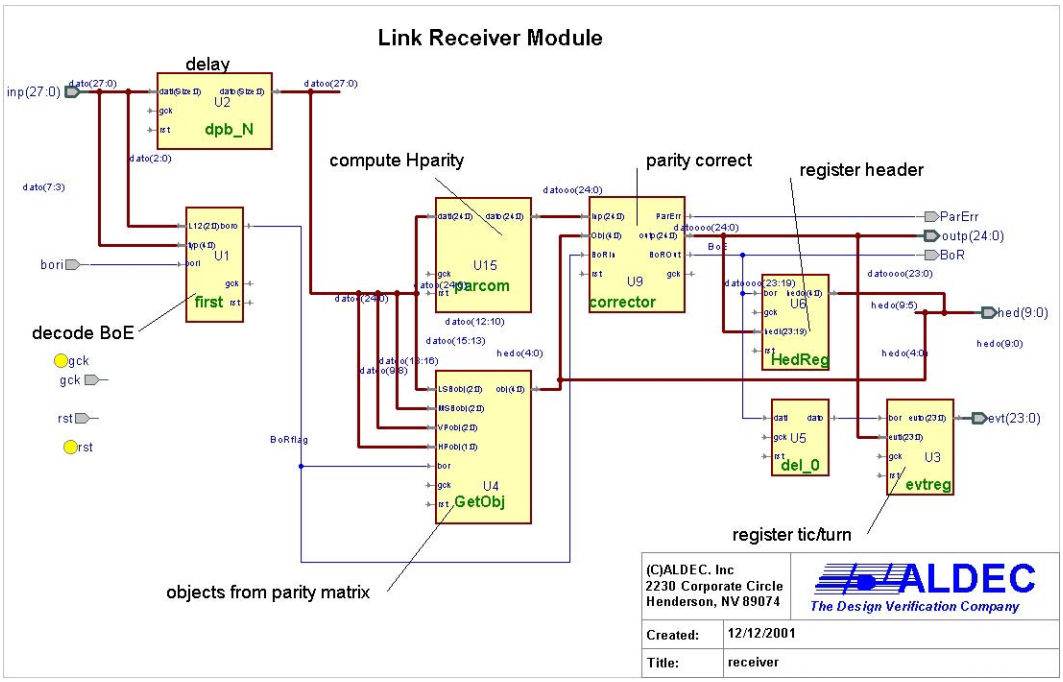

Figure 5 - Block diagram of receiver

The module *receiver* houses several functions: parity correction (*corrector*), determination of the number of data objects using a parity matrix (*GetObj*), determination of the first frame (*first*), and registering of the header and event frames (*HedReg* and *EvtReg*). For each frame, *parcom* computes the horizontal parity (*HPC*), and replaces the horizontal parity (*HP*), with *HP XOR HPC*.

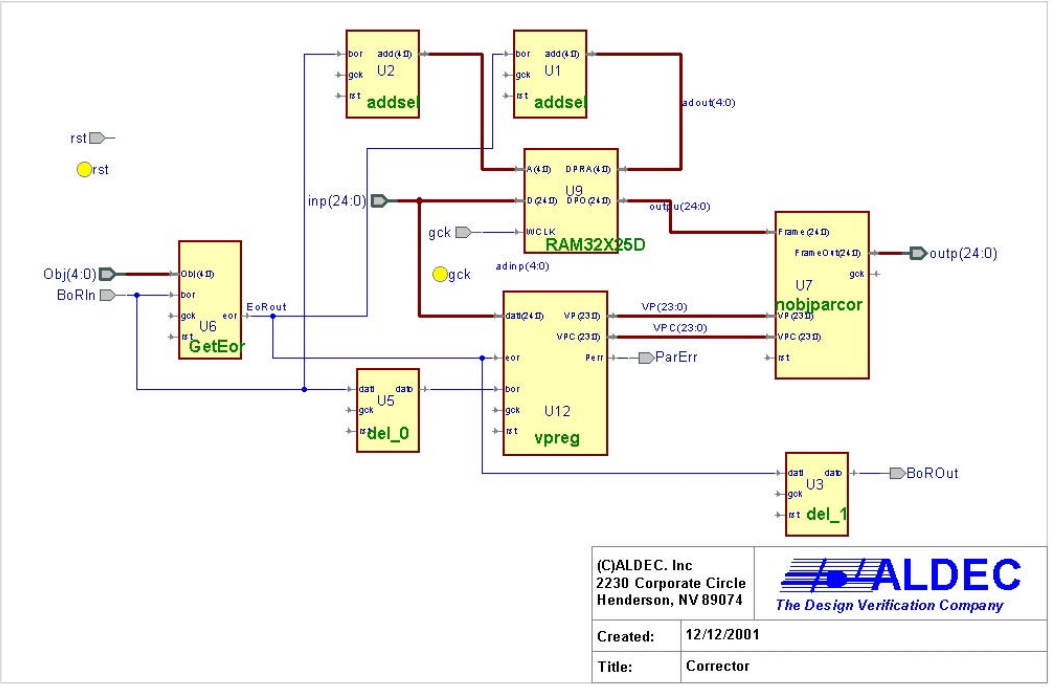

Figure 6 - Block diagram of Corrector.

The block diagram *corrector* performs the parity correction itself. The module *vpreg* calculates the vertical parity (*VPC*) and registers the vertical parity (*VP*). The module *GetEoR* counts words and issues and end-of-event signal. While reading all input data frames, the event is stored in a slice RAM (*ram32x25d*), and addresses are computed by *addsel.* Finally, the process *NobjParCor* computes the parity corrected frame.
The general expression for parity correction of bit *b* in frame *F* is given by:

$$F(b)corrected = F(b) \text{ XOR } FC(b), \text{ where}$$

$$FC(b) = (HP(F) \text{ XOR } HPC(F)) \text{ AND } (VP(b) \text{ XOR } VPC(b)), \text{ where}$$

HP(F) is the horizontal parity of the input frame, *HPC(F)* is the re-computed horizontal parity of the input frame, *VP(b)* is bit *b* of the vertical parity, and *VPC(b)* is bit *b* of the re-computed horizontal parity.

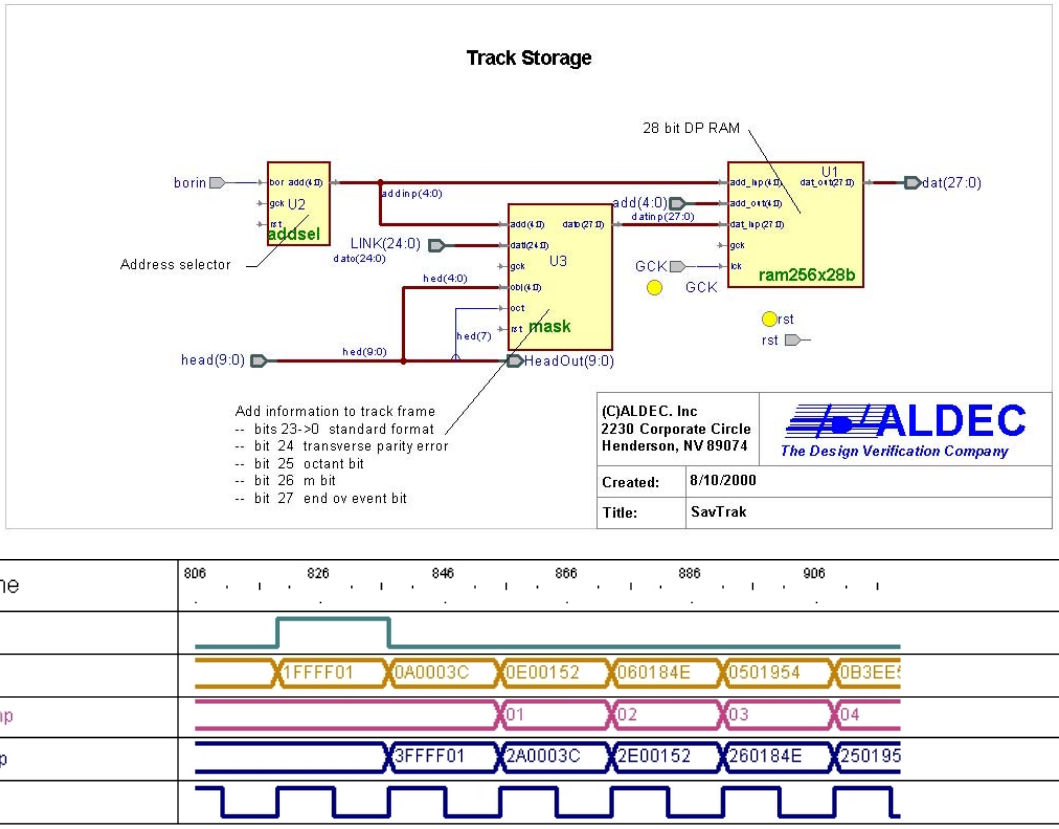

Figure 7 - Block diagram of SavTrk (top), and waveform of signals (bot).

Module *SavTrk* performs final storage function before link combination. The *BoRin* signal precedes the first track/cluster frame by one clock tic – this use of the Begin-of-Event signal is employed throughout the whole design, and is especially useful in all FSM.. The *mask* (U3) process extracts the octant *lsb* and inserts it into bit-25, and creates an end-of-event frame (bit-27=1) after the last track/cluster frame. BoRin starts the RAM address selector (*addsel*) as the track frames are sequentially stored in the RAM from address 1 to Ntracks.

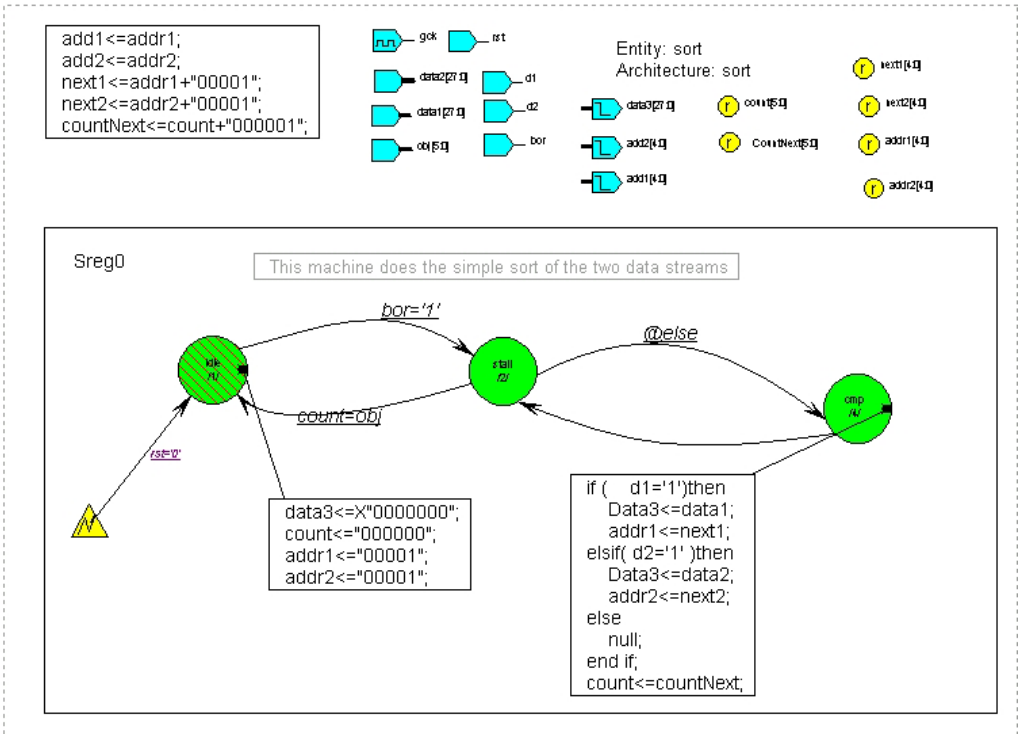

Figure 8 - State Machine of Sort

The state machine process performs a ordering of the two input streams. It relies on the concurrent computation of bits *d1* and *d2* inside *compare* and *compare2* respectively, where

d1<='1' when ((data1(27)='0' and not(data2(12 downto 8) <data1(12 downto 8))) or data2(27)='1') else '0';

d2<='1' when ((data2(27)='0' and data2(12 downto 8) <data1(12 downto 8)) or data1(27)='1') else '0';

Recall that data bits 12-8 are the *Pt* bin and bit 27 is signals the end of data.
For the cluster ordering

d1<='1' when ((data1(27)='0' or data2(27)='1') and err(0)='0') else '0';

d2<='1' when ((data2(27)='0' or data1(27)='1') and err(1)='0') else '0';

First, a track frame from each link is retrieved from the block ram. On the next clock cycle, the *d1* and *d2* values are compared and the largest *Pt* is placed in an output register and the RAM address is incremented. Processing each input track requires two clock cycles and formatting requires two G-link frames, so the output is synchronous and no intermediate storage is required. The ordering of clusters is identical except that no *Fi* comparison is necessary since all clusters from the lower octant have lower *Fi* values than those in the higher octant. To speed the process, all addresses concurrently precomputed.

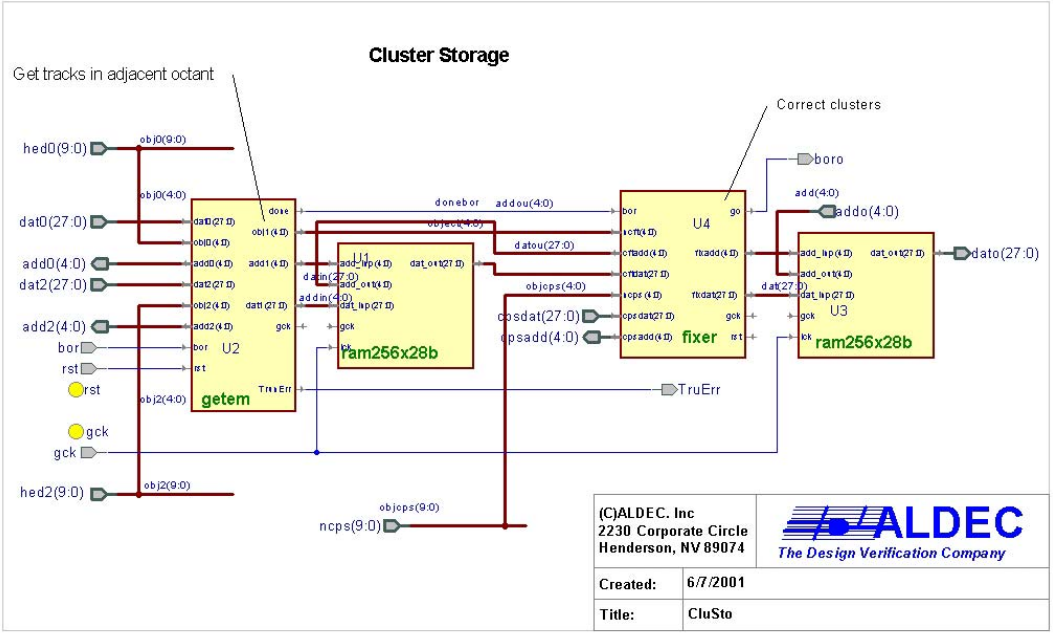

Figure 9 - Block diagram of CLUSTO

There is an instantiation of *clusto* for each cluster link, which stores clusters after correction for so called 'bend-in' tracks. The module *getem* (U2) gathers tracks from the two adjacent octants, and sends to the output RAM (U1). When all tracks have been processed, a *done* bit is set, and the number of correction tracks is available.
Upon receipt of the *done* bit, the *fixer* module (U4), applies the correction, and cluster frames are stored in the output RAM (U3).

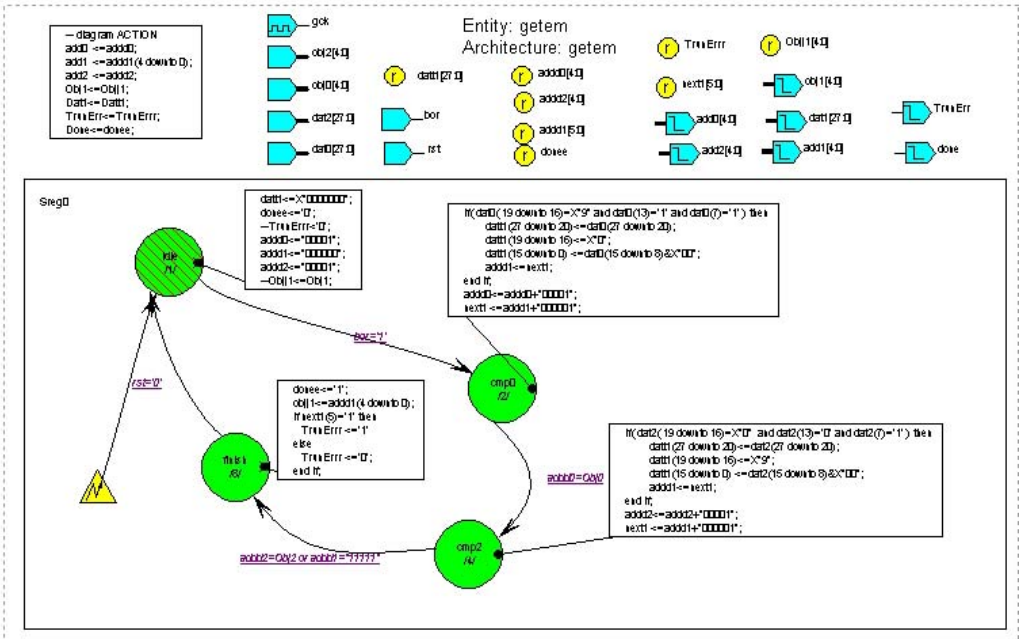

Figure 10 - FSM of GETEM

The FSM *getem* gathers tracks for use in cluster *Pt* bin correction from the *front_end* RAM. The inputs dat0(2) are track data from octants adjacent to a cluster octant. The outputs dat1 are the collected data for octant corrections. The algorithm is implemented as follows. The *R* bit indicates that the track has a cluster in a adjacent sector. If the low-*Fi* (dat0) frame has a track in sector 9 with positive bend (S=1) and R=1, then change RA_TS_CFT (dat(19 downto 16)) to 0 and send to output. If the high-*fi* (dat2) frame has a track in sector 0 with negative bend (S=0) and R=1, then change RA_TS_CFT to 9 and send to output. The output frame (dat1) will have the RA_TS_CFT of the cluster for later comparison with a cluster RA_TS_CPS.

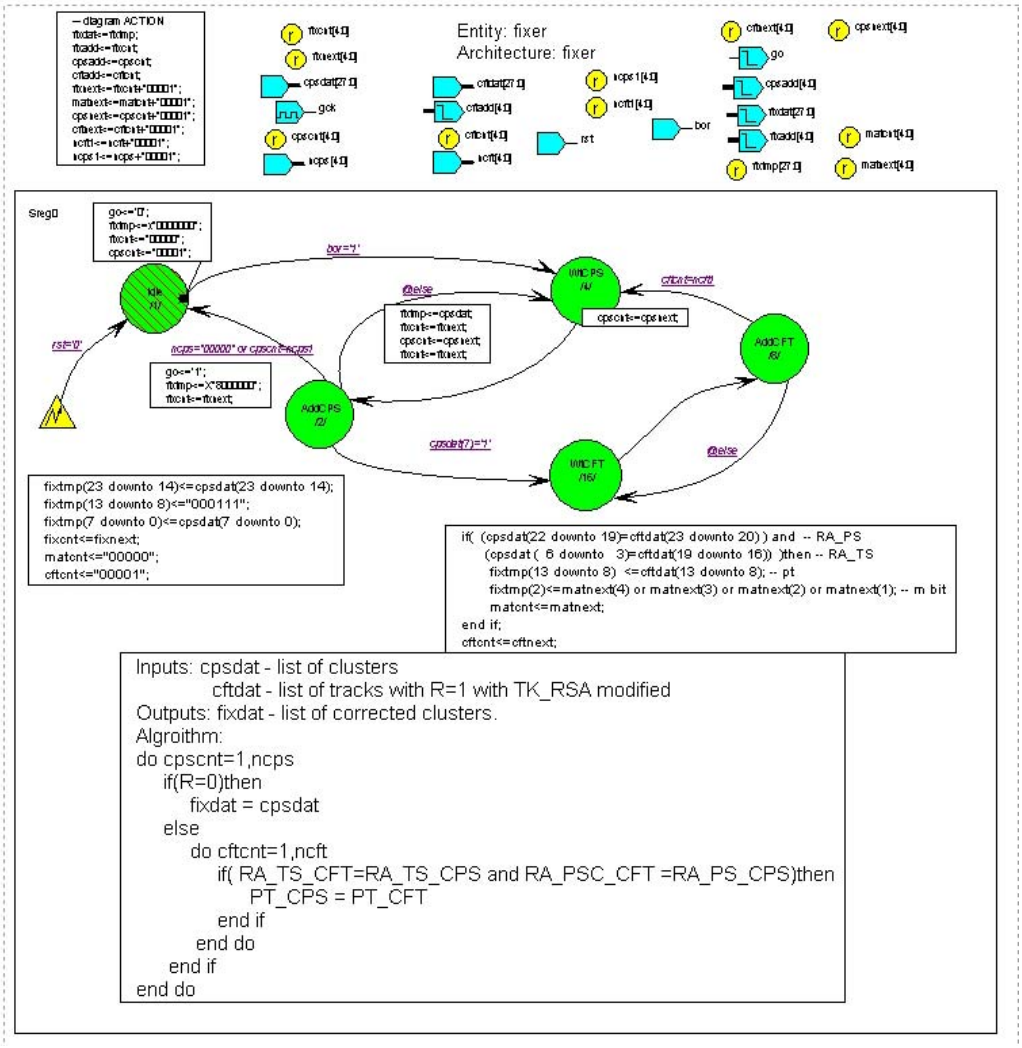

Figure 11 - FSM of FIXER

The FSM *fixer* implements a doubly nested loop by addressing dual port RAMs. The outside loop over the list of clusters checks if the cluster R-bit=0. If so, then no correction is applied and the cluster is output unchanged; otherwise, the list of tracks is looped. Each track and cluster are compared to see if they are in the same relative trigger sector (RA_TS) and cluster strip address (RA_PSC). If these are equal then the track is matched and the cluster *Pt* field is filled with the track *Pt*, and a counter is incremented as all tracks are checked. If the number of matching tracks is greater than one, the M-bit is set to 1. The process is repeated until all clusters are corrected. If no matching track is found, then *Pt*=7 (an illegal value). *Note: The last track satisfying the matching condition is always used, and a track can be used to correct more than one cluster.*

Outline of Hierarchy

- front_end - BDE
  - dwdb_fe6 - Jamiesons front end
  - receiver6 - BDE wrapper for 6 receivers
    - receiver - BDE
      - first - checks L1/L2 and datatype
      - parcom - compute horozontal parity
      - hedreg - register header and event frames
      - corrector - BDE for parity correction.
        - addsel - FSM increments read and write address given BoE
        - ram32x25d - 32 deep by 25 wide DP ram
        - GetEor - FSM counts objects and send and end signal
        - del_0 - delays signal by one clock tic.
        - del_0 - delays signal by one clock tic
        - vpreg - computes Vparity and flags Hparity errors
        - nobjparcor - parity correction
  - evtvote6 - six input majority vote for event number (tic/turn)
    - maj3_24 - 24 bit 3 input majority vote
    - maj3 - 1 bit 3 input majority vote
    - comevt6 - compare six values to the winner.
  - sync6 - FSM send BoE for good links
  - storage - BDE
    - savtrk - BDE for track storage
      - addsel - address incrementer
      - ram256x28d - 28 wide 256 deep DPram
      - mask - add end of event marker and octantinfo to frames
- clusto - BDE store and correct clusters
  - getem - FSM gather correction tracks
  - fixer - FSM correct tracks
  - ram256x28b - block ram for intermediate storage
- combine - BDE combine tracks
  - compare - compare Pt
  - sort - FSM does ordering
- combine2 – BDE, combine clusters
  - sync3 – FSM syncronize clusters after correction
  - compare2 - compare fi
  - sort - FSM does ordering
- sender - FSM output tracks
- sender2 - FSM output clusters
- slow_interface - interface to slow controls
- muxN - address multiplexer
- BusArb - does bus arbitration

Input Controls

No controls implemented yet

Outputs

Header frames 5 and 6 contain status information about the event. Header frame 5 has the following format:

bits(5 : 0) – parity errors for links(7,4,3,2,1,0)
bits(11 : 6) – link errors for links(7,4,3,2,1,0)
bits(15 : 12) - blank

and Header frame 6:

bits(5 : 0) – event errors for links(7,4,3,2,1,0)
bits(7 : 6) – truncation errors for links(4,3)
bits(9 : 8) – channel DNF errors for links(4,3)
bits(14 : 10) – blank
bits(15) – OR of all error bits

The first output frame of each track or cluster has three error bits with the following meaning:

Bit5 – parity error detected and corrected
Bit6 – unused
Bit7 - unused

Slow Monitoring

page0 - feature register;
page1 - "00" & parity_history (links 7,4,3,2,1,0)
page2 - "00" & link_history (links 7,4,3,2,1,0)
page3 - "00" & event_history (links 7,4,3,2,1,0)

Performance

Copyright (c) 1995-2002 Xilinx, Inc. All rights reserved.

```
Design Summary:
  Number of errors:      0
  Number of warnings:  100
  Number of Slices:            2,911 out of  6,912   42%
  Number of Slices containing
    unrelated logic:               0 out of  2,911    0%
  Number of Slice Flip Flops:    2,906 out of 13,824   21%
  Total Number 4 input LUTs:     3,938 out of 13,824   28%
    Number used as LUTs:                    2,833
    Number used as a route-thru:              205
    Number used for Dual Port RAMs:           900
    (Two LUTs used per Dual Port RAM)
  Number of bonded IOBs:          225 out of    404   55%
  IOB Flip Flops:                   136
  Number of Block RAMs:            20 out of     24   83%
  Number of GCLKs:                  4 out of      4  100%
```

Number of GCLKIOBs: 4 out of 4 100%
Total equivalent gate count for design: 431,267
Additional JTAG gate count for IOBs: 10,992
Peak Memory Usage: 105 MB

Device utilization summary:

Number of External GCLKIOBs 4 out of 4 100%
Number of External IOBs 225 out of 404 55%
Number of LOCed External IOBs 225 out of 225 100%

Number of BLOCKRAMs 20 out of 24 83%
Number of SLICEs 2911 out of 6912 42%

Number of GCLKs 4 out of 4 100%

Design statistics:
Minimum period: 15.939ns (Maximum frequency: 62.739MHz)
Maximum net skew: 0.946ns
Minimum input arrival time before clock: 2.900ns
Minimum output required time after clock: 7.352ns