Proposed SCT Readout Architecture

*plus some hooks for future trigger innovation**

Francis Anghinolfi, Michelle Charriere, Joel Dewitt, Philippe Farthouat, Daniel La Marra, Mitch Newcomer, Elliot Lipeles, Matt Warren,

* (*proposed*)^2

Objectives for High Rate SCT Readout

Minimize power and Material (# fibers…) in Readout Path

→ Avoid reading out data that will very likely be rejected early in the filtering process. …Keep data on detector longer.. but

→ Avoid Large memory associated with long fixed latency triggers.

Realization

→Fixed latency trigger (renamed to **L0**.) Remains nearly as L1 is presently.

- **L0** data *transferred* to holding buffer registered with L0 ID **(L0ID)** counter.

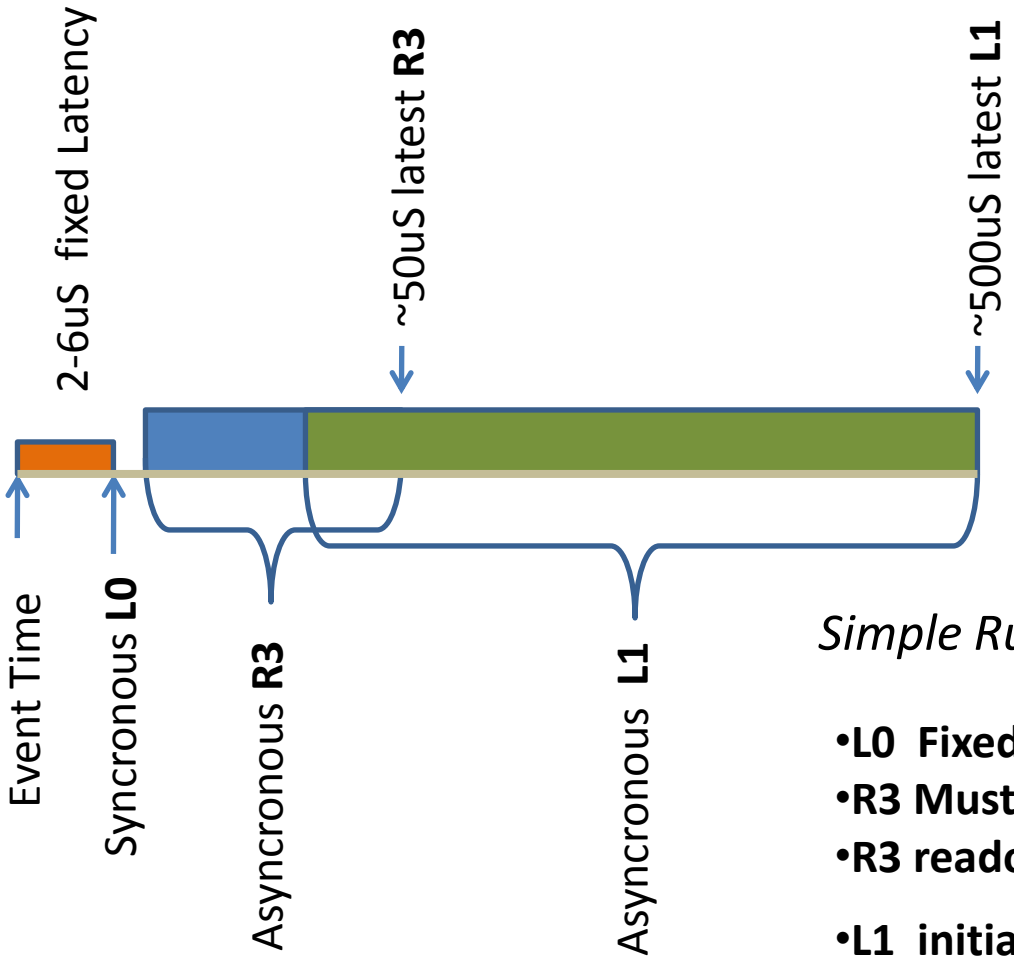
  Not sent to a readout buffer...

- **R**egional **R**eadout **R**equest (**R3**) triggers, addressable by module, are sent out with an asyncronous 5-50uS. Latency. Referencing data by **L0ID** tag.

  R3 data has the highest priority for readout but should be sent to a few percent of the modules on the detector at a time.

  L1 triggers initiate *readout of all modules* with a lower priority for readout than **R3** triggers. The expected rate is 100KHz or lower.

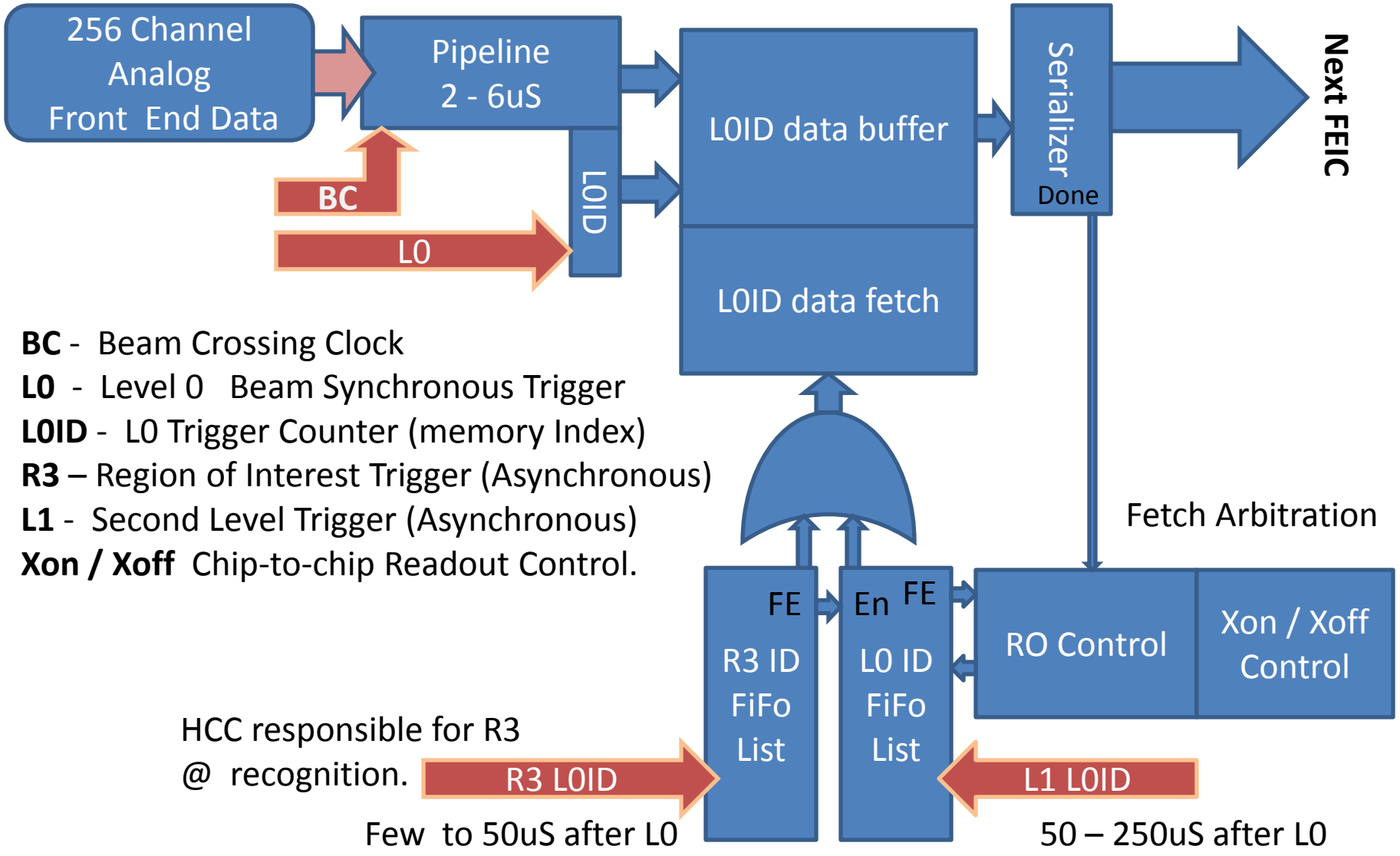
HL-LHC Trigger/Readout Timeline



Simple Rules for Maximal Flexibility

- **L0 Fixed at Initial Setup**
- **R3 Must precede L1**
- **R3 readout highest priority**
- **L1 initiates Off Detector Readout**

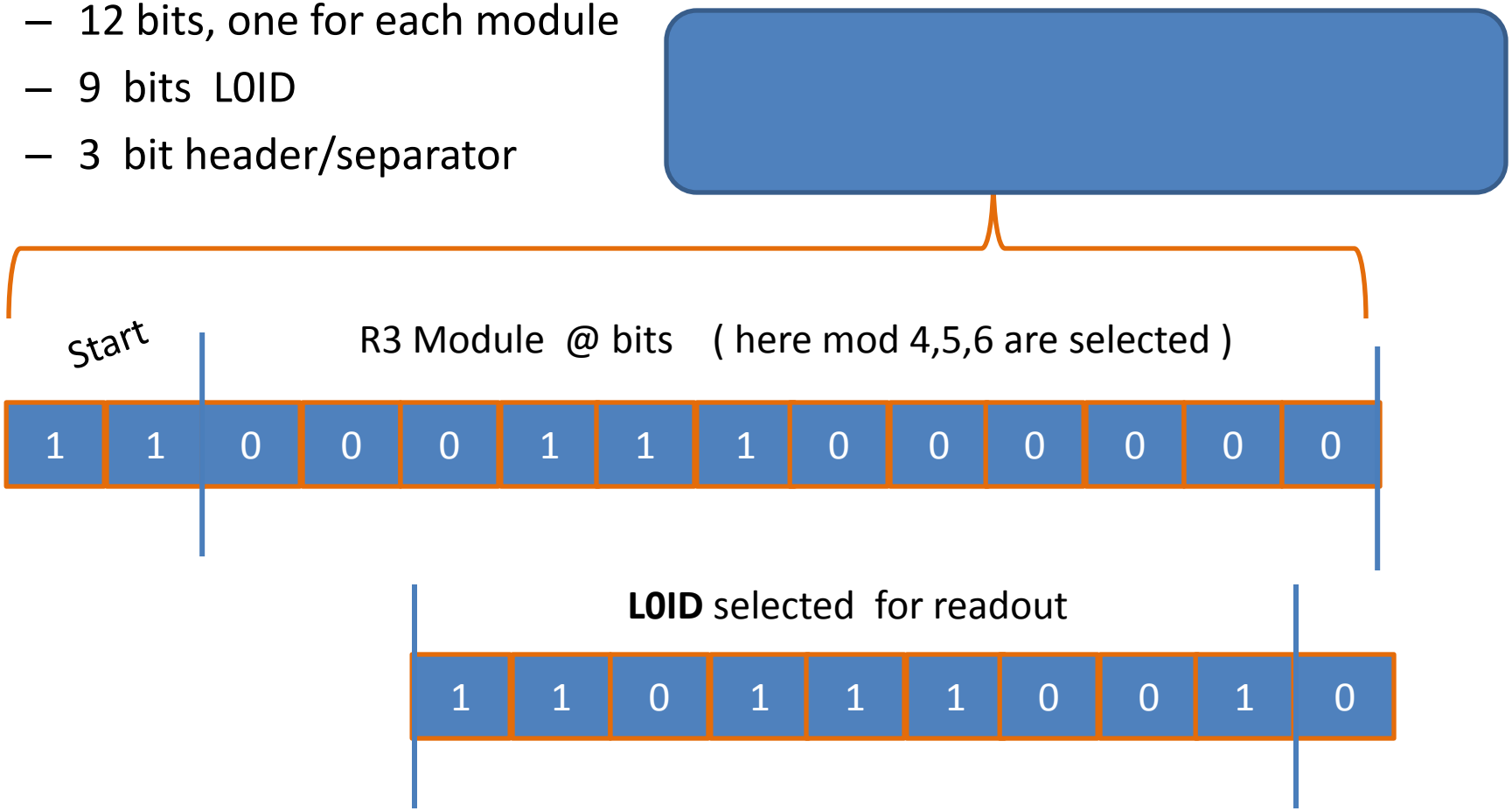
Front End ASIC based Readout Architecture



BC - Beam Crossing Clock
L0 - Level 0 Beam Synchronous Trigger
L0ID - L0 Trigger Counter (memory Index)
R3 – Region of Interest Trigger (Asynchronous)
L1 - Second Level Trigger (Asynchronous)
Xon / Xoff Chip-to-chip Readout Control.

Asyncronous R3 Trigger packet (proposal)

- 12 bits, one for each module
- 9 bits L0ID
- 3 bit header/separator



Total = 24 bit times @ 80MHz, → 300ns per R3 packet

Allows a stave to have a maximum R3 rate of 3.3MHz

Elink generated Stave Signals

Four Differential Pairs

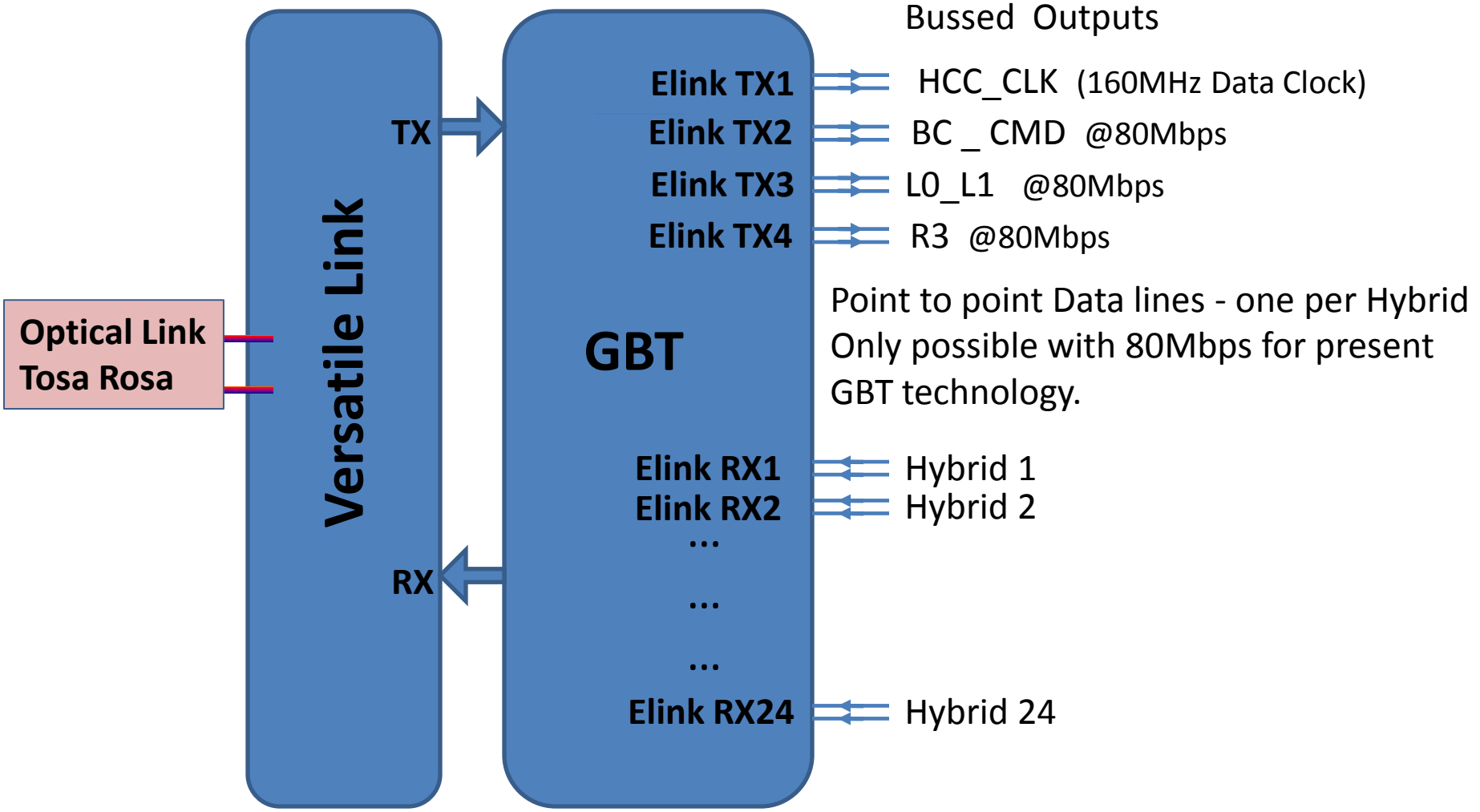
- **160MHz Data Clock –** HCC data readout clock.
  HCC generates ABC Data Clock / BC clock
- **BC_CMD –** BC sync interspersed with CMD (80Mbps)
- **L0_L1 –** (80Mbps) L0 Sync'd with BC @ (40MHz)
  L1 utilizes other 40MHz phase* to send L0ID for L1 trig
- **R3** - Dedicated line to handle rate for 12 modules on Stave.
  R3 Packets (headr+Staveaddrs+L0ID) described on Slide 4.

→**Simple MCC/ABC decoding with physically separated functions and plenty of spare bandwidth.**

→**Robust approach for a high radiation environment.**

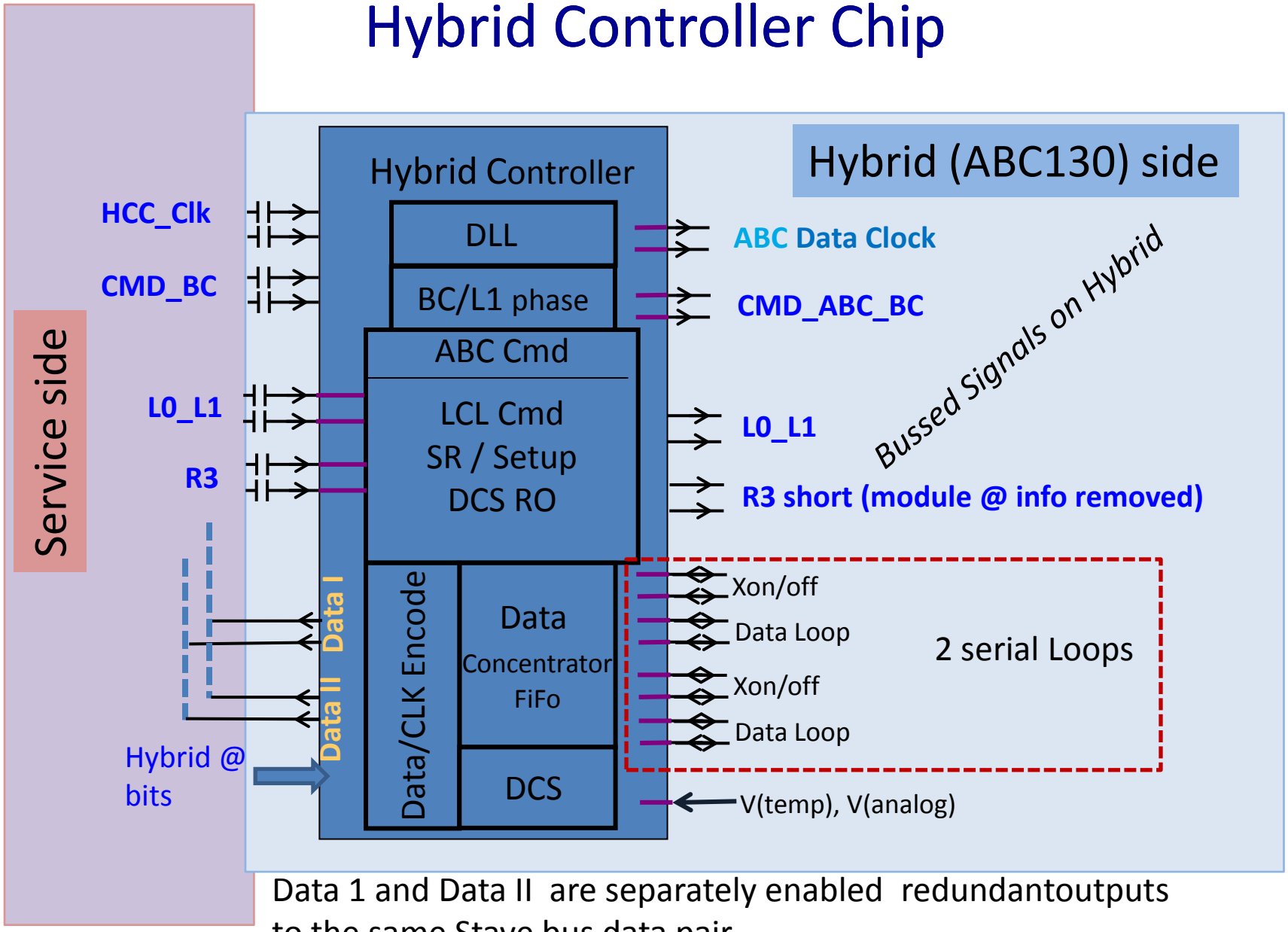
*** far more bandwidth than necessary for L1: ~2MHz rate possible.**

End of Stave (GBT controlled) Signals



Note that for a longer stave we would be forced to go to 2 GBT's per Stave.

Hybrid Controller Chip

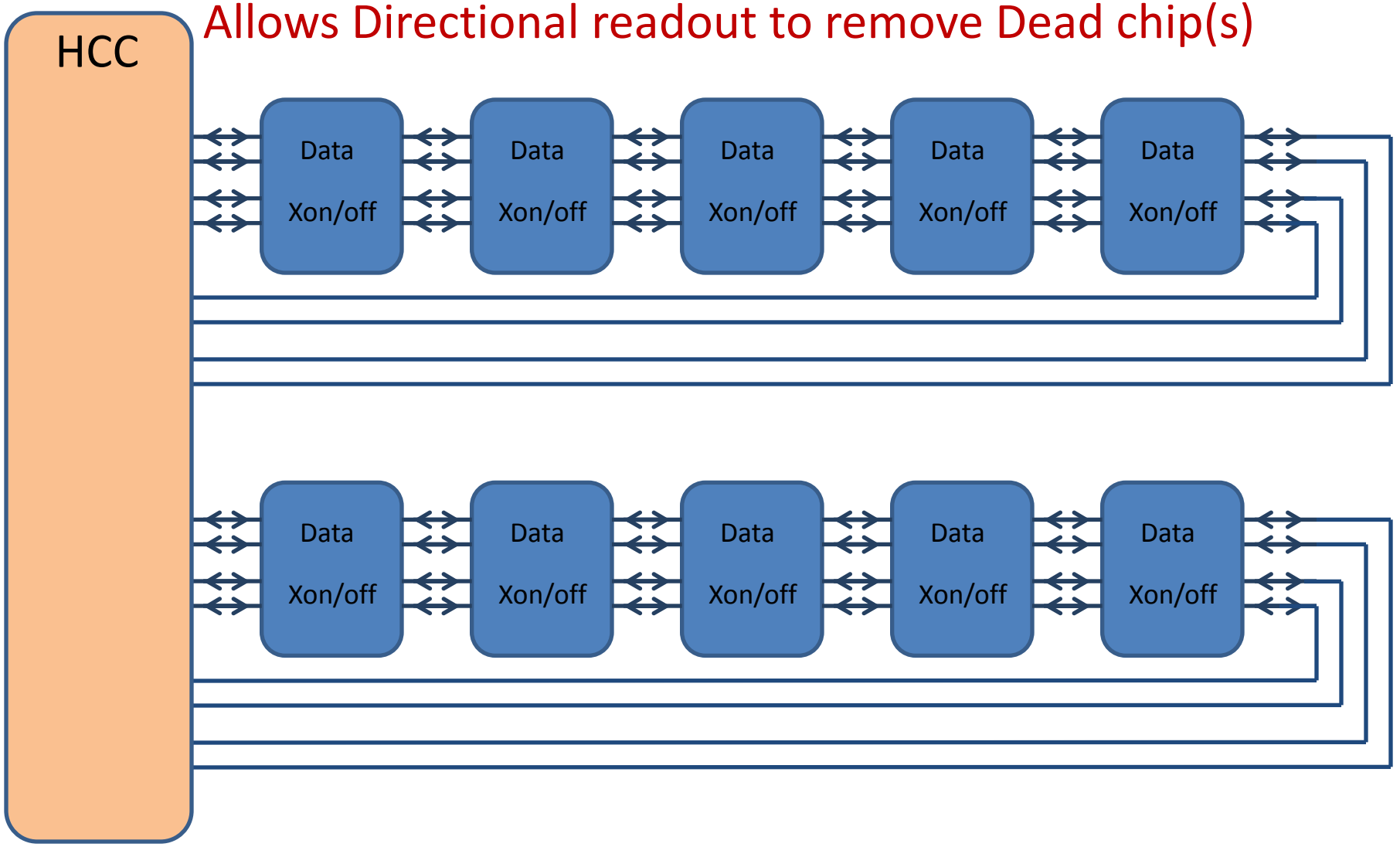


Data 1 and Data II are separately enabled redundantoutputs to the same Stave bus data pair.

2 Loop Serial Data readout Path on Hybrid

Splits Readout Loop

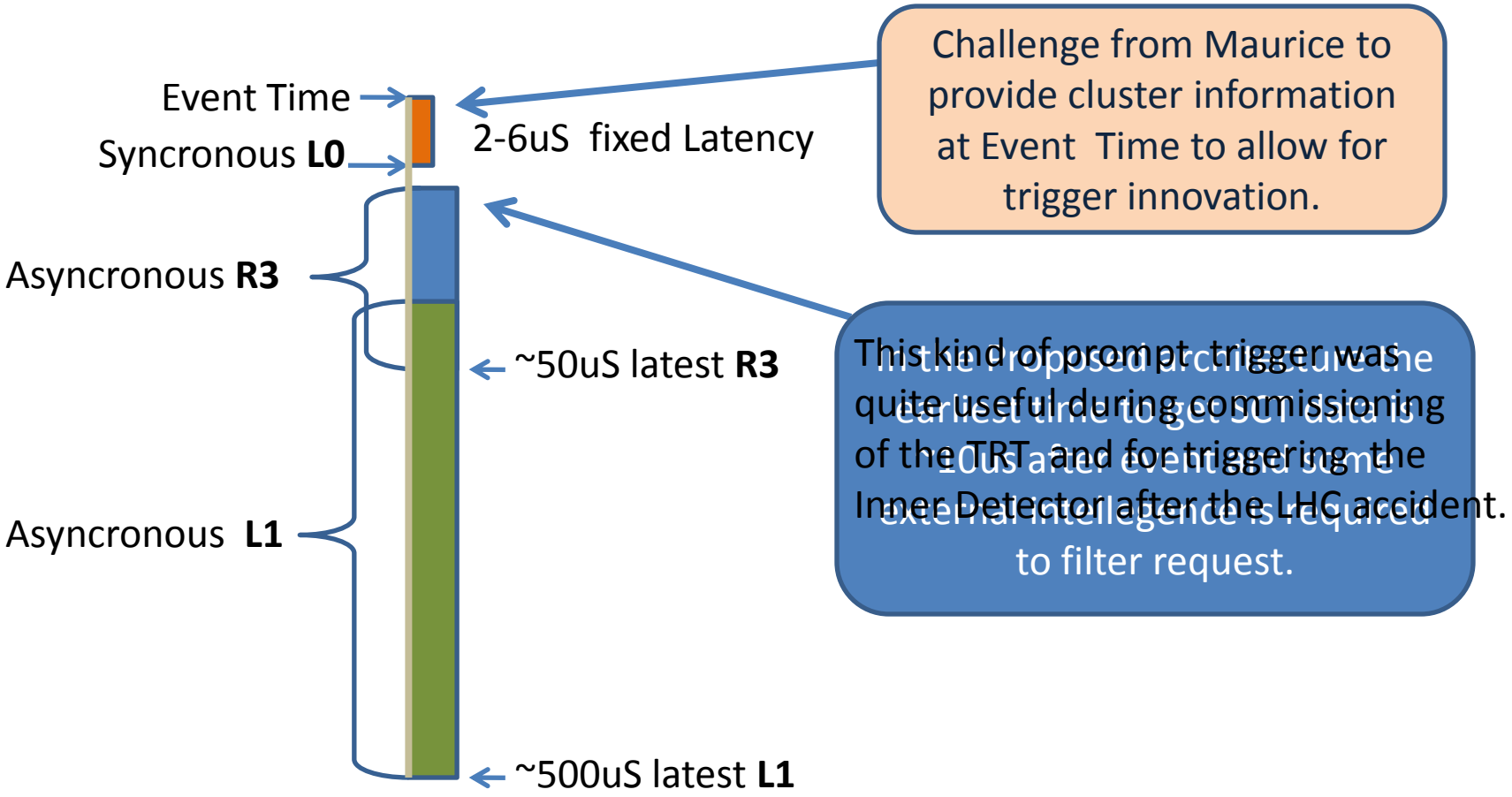
Allows Directional readout to remove Dead chip(s)



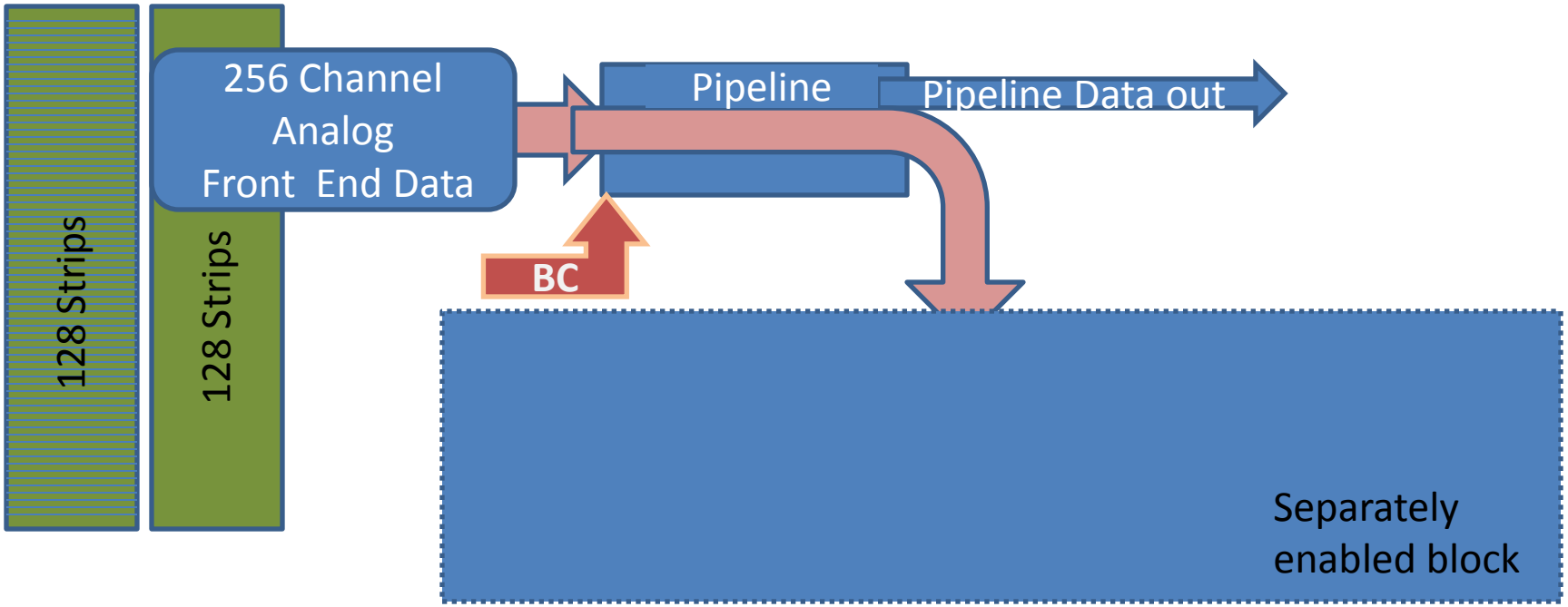
Summary

- An Architecture with a versatile two level trigger capability has been developed for the upgraded SCT and is being detailed with the intention to fabricate in IBM's 130nm process. (~18months)
- Details about the ASIC design will be presented in the next talk by: Francis Anghinolfi.

Trigger Primitives for Commisioning & Innovation



Fast Readout Conceptual Block Diagram



- **Store Data as two banks of 128 strips at BC time.**
  One buffer register per bank.
- Perform clustering algorithm in multiple steps @ 160Mhz using tightly restricted acceptance rules (next page)
- Send Fixed Length Cluster information at a fixed #BC after the event to each dedicated LVDS output.
- *Due to low occupancy it is allowable to allocate several BC each time a cluster is located.*

Rules for Prompt Cluster

- Treat Each Bank independently.
- Cluster must have no more than 2 strips in a row.
  ie 3+ = Veto prefers energetic track.
- No more than 2 Clusters in a Bank.
- New BC cluster information locked out to prevent overwrite.
  - Restart processing as soon as overwrite condition is avoidable.
- Data → Two 8 bit values 16 bits total = 4 BC's at 160Mbps
  - Valid Data = 2, 7 bit cluster addresses + "00" trailer
  - "Null" second Cluster is a repeat of first cluster.

Several approaches will be explored/compared for complexity versus execution time. An initial attempt in development is to split the 128 bits into 8 parallel 16 bit banks and find clusters within each in one step, examine border cases and then count all clusters. The 16 bit cluster finder is shown in the next pages.

Summary

A fast output will allow the exploration and development of prompt trigger information from the inner detector with little overhead.

This block will be **OFF** except when enabled.

It could also turn out that these outputs prove useful during prototyping and commissioning.

16 bit cluster finding Algorithm (in Development) 1/3

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date:    19:46:15 03/28/2011
// Design Name:
// Module Name:    taskCheckCluster
// Project Name: readout
// Target Devices:
// Tool versions:
//
// DESCRIPTION: This module accepts clock as input. It consists of a single task.
// The task has  16 bits data as input and gives out the following information in a 13 bit register 'netaddress'-
//               1. The locations where hits have occurred and if there are two adjacent bits high.
//                     The netaddress[3:0] and netaddress [9:6] hold 4 bit addresses. The bits 4 and 10 are '0' if adjacent
//                     bits are high in netaddress[3:0] and netaddress[9:6] respectively.
//
//               2.    If there are three hit clusters occuring in a 16 bit data group, the bit 12[MSB] is '0'. This acts as a flag
//                     to the calling module to indicate that entire 128 bits of data need to be discarded.
//
//               3.    If bit 5 = 0, netaddress[3:0] has valid address and if bit 11 = 0, netaddress[9:6] has valid
//                     address.
//
//               4. By default, all the bits of register are high.
//
//               5. If there are continuous strings of ones (> 2 consecutive bits), the entire string is not valid.
//
//               6. The checking for the '1' string goes on only till bit 13 (bit 14 & 15 are not individually checked.
//                     They are checked in the main module along with checking the bit 0 & 1 of next block)
//
//
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//
//
//
//
//////////////////////////////////////////////////////////////////////////////////
```

16 bit cluster finding Algorithm (in Development) 2/3

```
module taskCheckCluster(clk);//,data_in,netaddress_out);
input clk;
//input [15:0]data_in;
//
//output [12:0]netaddress_out;

//always @ (posedge clk)
//begin
task CheckCluster;
      input [15:0] data;
      output [12:0]netaddress;
      reg [12:0]netaddress;
      reg [5:0] temp; // temporarily holds each location of a hit.
      reg flag;// high when one hit has occurred
      reg flag2;// high when two hits have occurred in discrete locations
      integer i, j ;
      //initial
begin
      netaddress [12:0] = 13'b1111111111111;// output high by default
      i = 0;
      j = 0;
      flag = 1'b0;
      flag2 = 1'b0;
      //end

      while (i <= 15)// 16 bit data
            begin
            temp[5:0] = 6'b1;
            if (data[i])// if there is a hit in location i
                  begin
                  temp[5] = 1'b0; // valid hit
                  if( i <= 13 && data[i+1] && data[i+2])// 3 or more adjacents bits are high, cluster discarded
                        begin
                        temp[5] = 1'b1; // not a valid hit
                        for (j = i +3 ; j < 16 ; j = j + 1)
                              begin
                              if(data[j] == 1'b0)// discontinue the for loop if data at that location is low
                                    begin
                                    i = j;
                                    j = 16;// false value to j simply to discontinue the for loop
                                    end
                              end
                        end
                  else
                        begin
```

16 bit cluster finding Algorithm (in Development) 3/3

```
                temp[3:0] = i;
                if( i <= 14 && data[i+1])// if 2 consequtive  bits are high
                      begin
                      temp[4] = 1'b0;
                      i=i+2;
                      //continue;
                      end
                else
                      begin // only one bit is high
                      temp[4]=1'b1;
                      i = i + 2;
                      //continue;
                      end
                end
          end
    if (flag == 1'b0 && temp[5] == 1'b0)/*first hit occured*/
          begin
          netaddress[5:0] = temp;
          flag = 1'b1;
          end
    else if(flag == 1'b1&&temp[5]==1'b0)//second hit occured within the 16 bit sub-division
          begin
          netaddress[11:6] = temp;
          flag2=1'b1;
          end
    else if (flag2==1'b1 && temp[5]==1'b0) // if three bits havve occured within the 16 bit data group
          begin
          netaddress[12] = 1'b0;
          i = 16;// false value given to i simply to get out of while loop
          end

    i=i+1;// increment i for checking the next data bit.
    end
end

endtask
endmodule
```