Xavier Gremaud, EPFL, 17.05.2011

TELL10 Velo data processing note

Contents

1 Introduction

This document will explain the different functions of the data processing in the FPGA of the Tell10 card, separating the Velo specific blocks and the blocks used for all the detectors. The name Tell10 references to the intermediate step data acquisition board (the next step is Tell40) designed at LPHE EPFL. It has not the identical properties as the TELL40 designed by the Marseille group but it is based on the same type of FPGAs. The studies and conclusions done for the data processing regarding the VELO are certainly valid to a large extend for both TELL10 and TELL40.

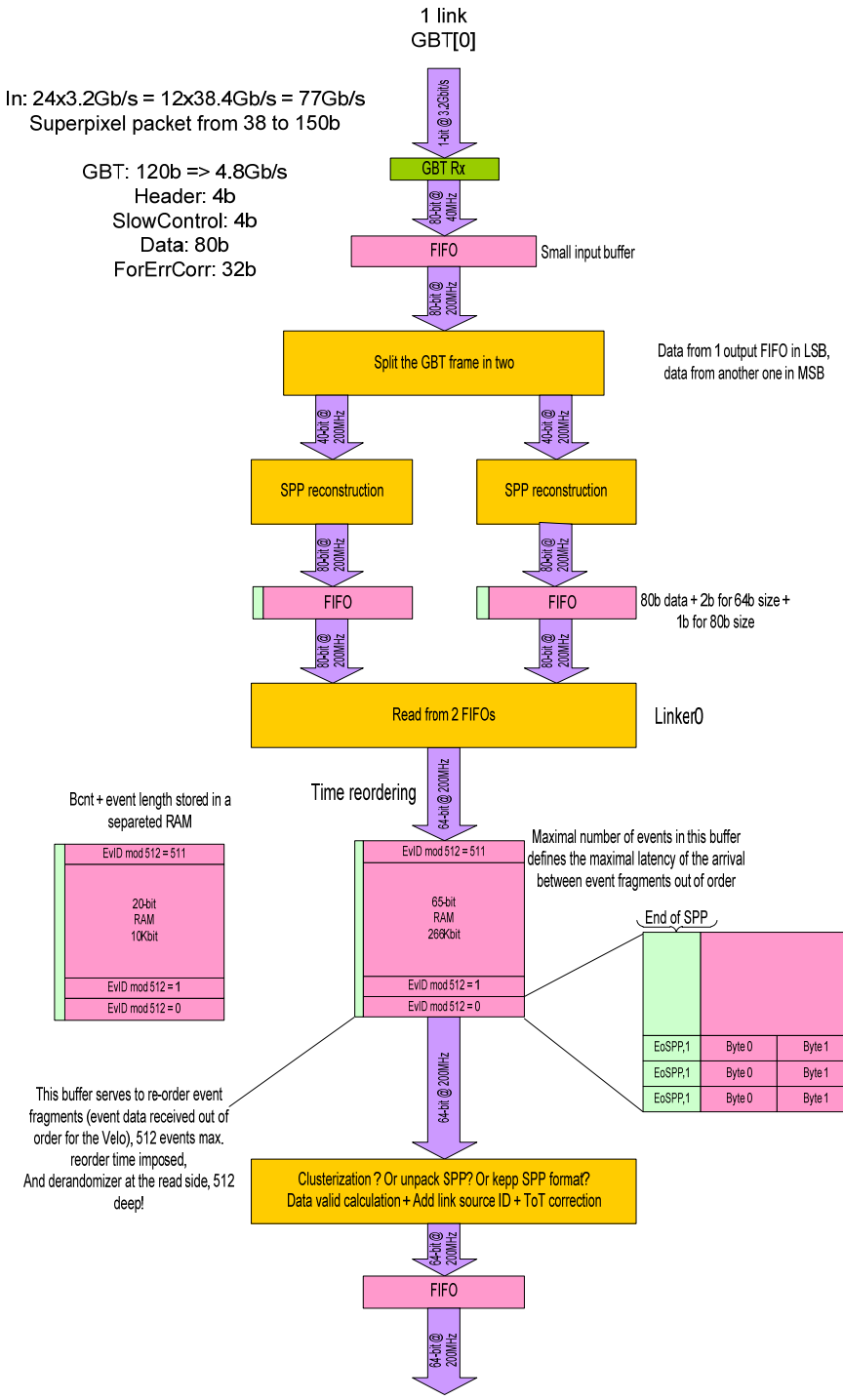
The TELL10 card has 24 GBT serial links providing each data steams 80b wide @ 40MHz. The data rate of 1 GBT link is 80b/25ns = 3.2Gb/s, so for 24 links, the maximum input data rate is 77Gb/s.

The data is assembled from the different links, stored in two 4GB DDR3 SDRAM memory and then send out to the CPU with 4x 10Gigabyte Ethernet links in the MEP format.

Note that in contrast to the TELL1, no data reduction due to zero suppression is expected to be obtained at the FPGA. Data reduction is done in the FE electronics.

The LHCb bunch crossing rate of 40MHz and the TELL10 data processing frequency of 200MHz imposes a maximum of 5 clock cycles per event in all processing stages. A large amount of pipeline processing is required.

2 Data flow and resources utilization



Resource utilization:
[ALUT]

SPP splitter:
260 (x24)

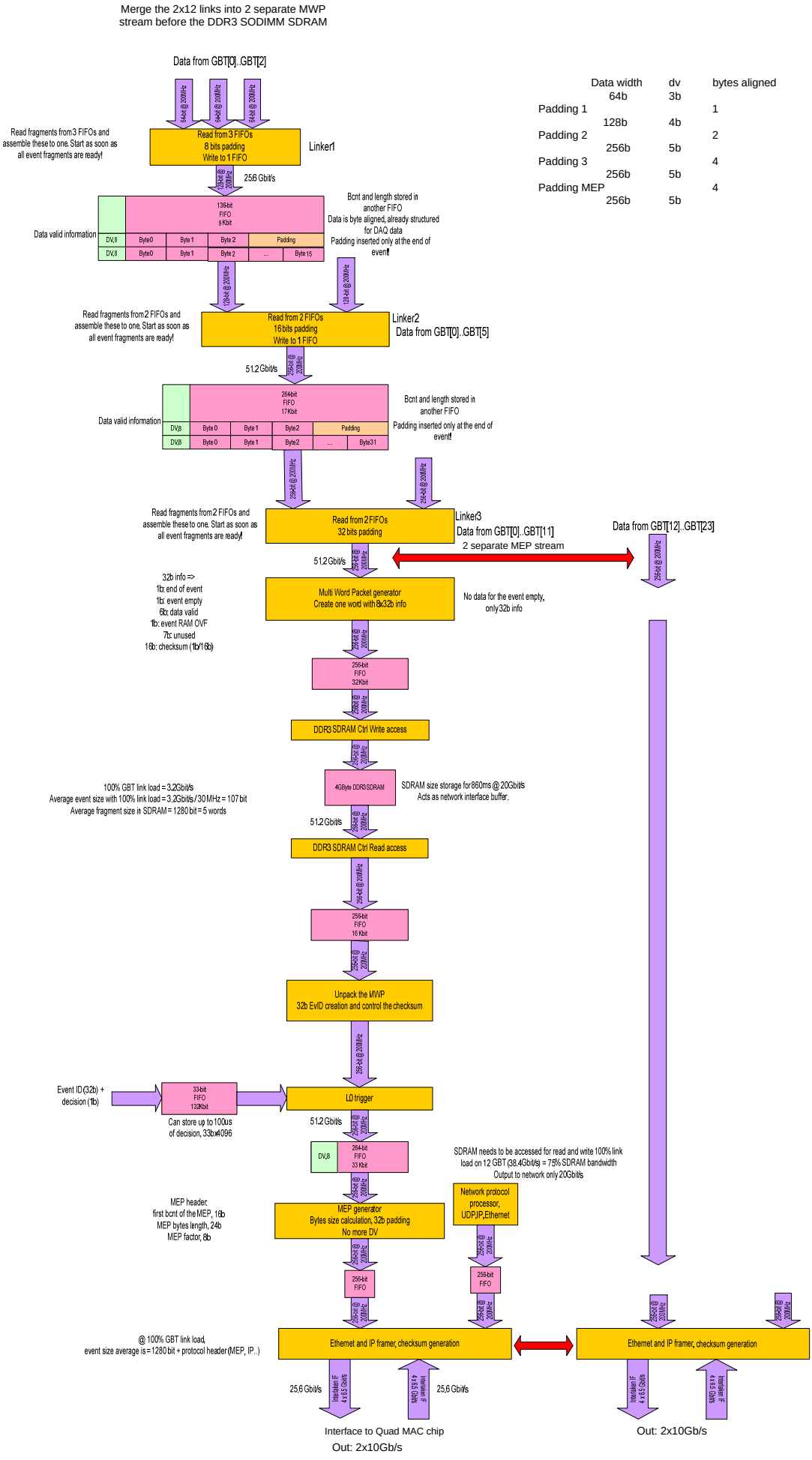
SPP reconstruction:
2x 1910 (x24)
nSPP reconstruction:
2x1380 (x24)

Linker 0:
260 (x24)

Time reordering:
1100 (x24)

Unpack SPP:
~2000 (x24)
Unpack nSPP:
650 (x24)

Create dv :
65 (x24)



	Data width	dv	bytes aligned
	64b	3b	
Padding 1			1
	128b	4b	
Padding 2			2
	256b	5b	
Padding 3			4
	256b	5b	
Padding MEP			4
	256b	5b	

Resource utilization:
[ALUT]

Linker 1:
400 (x8)
Padder 1:
1280 (x8)

Linker 2:
540 (x4)
Padder 2:
2420 (x4)

Linker 3:
960 (x2)
Padder 3:
2800 (x2)
MWP gen:
1450 (x2)

SDRAM driver:
1640 (x2)

Unpack MWP:
640 (x2)

L0T:
640 (x2)

MEP gen:
4490 (x2)

	Logic (ALUT)	Total memory bits	M144k blocks
Data processing	200'000 (48%)	7'500'000 (37%)	48 (75%)
FPGA	424'960	21'233'664	64

Figure 2-1 : FPGA resource utilization with unpacking the nSPP format and not much monitoring

3 Velo specific parts of the data processing

3.1 Super Pixel Packet (SPP) reconstruction

In the FE, the 80b wide GBT word is filled by two distinct data streams each 40b wide, one for the lower and one for the higher part of the 80-bit words.

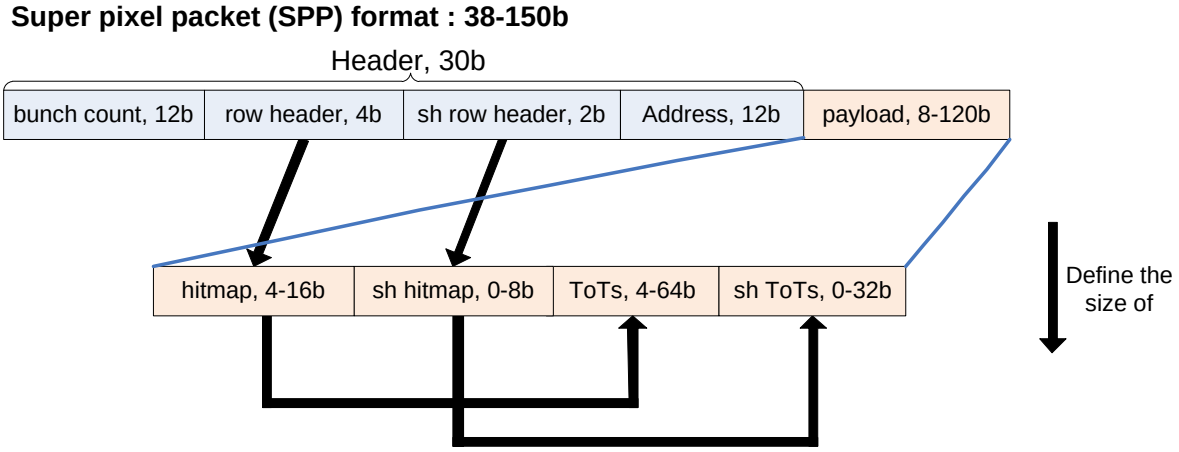


Figure 3-1 : SPP format

With this format, half of the logic of the data processing is used to reconstruct the data into SPPs. The main cause of the huge resource usage is the fact that the data is aligned to 8 bits (the length of one SPP varies from 38 to 150b) and the data of two consecutive SPP are aligned on 8b. The consequence is that a huge multiplexer of (184x150/8=3k5)x48 is required! (Twice for each link) For the longest SPP of 150b, a shift register of the length of 152-8+40=184b is required.

With that format, it needs at least 54b to know the length of the SPP. It depends of four different fields: row header, shared row header, hitmap and shared hitmap. Moreover the length of hitmap and shared hitmap can vary. So it takes a lot of tests to find the hitmap and the shared hitmap, and then it need to count the number of hit in the two fields to calculate the length of the SPP. The number of bits that need to be checked is too large in order to use a LUT. When the length of the

SPP is known, it needs to check if there are enough bytes already read, otherwise it need to wait for more data.

With the new SPP format “nSPP” only the “hit count” field is needed to know the length of the SPP. It can be more easily reconstructed, but it will still need quite large multiplexers (192x150/8 x48).

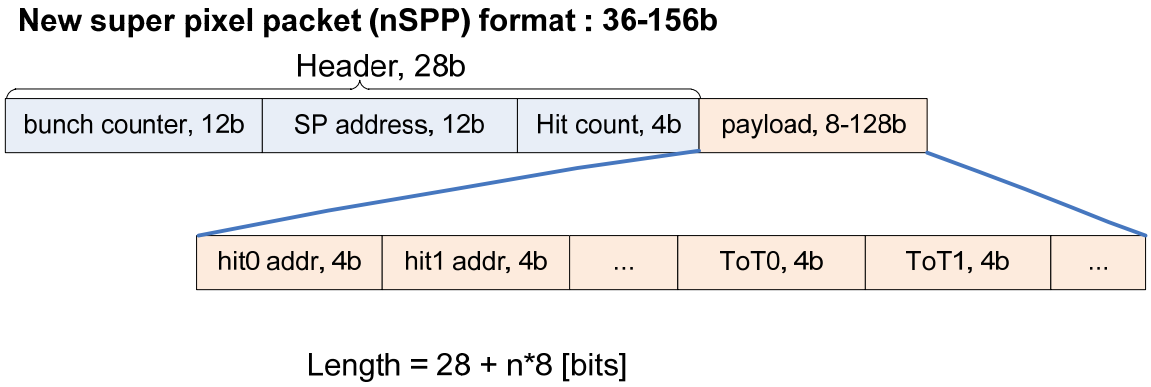


Figure 3-2 : nSPP format

	Logic (ALUT)		Logic (ALUT)
SPP reconstruction	92700 (22%)	**Unpack SPP**	~48000 (11%)
nSPP reconstruction	66200 (15%)	**Unpack nSPP**	15600 (4%)

Figure 3-3 : SPP and nSPP format resource utilization

3.2 Time reordering

The SPP are sent by the Velopix in disorder concerning the time. They need to be time reordered in the FPGA.

The time reordering is done using RAM blocks. In the current FPGA EP4SGX530 (largest Altera Stratix IV device), a total of 64x144kB memory blocks are available. The RAM space is divided in 512 equally sized memory blocks, the space reserved for data arriving in random order. The RAM location is defined with LSBs of the BCnt. The total memory space required is the maximum time delay allowed multiplied by the maximum event size allowed, space for every event has to be reserved, even for empty events(512 events x 8 words x 65b = 266kB, max. 288kB/links). Each GBT link is restricted to 8 SPP smaller than 64b. Choosing a time reorder buffer of 512 events deep and 8 words event size occupies 48 memory blocks (maximum size reached!). There are no other large memories required for the other processing steps. With a margin of 16 events for the time reordering, the time reorder is possible for up to 512-16=498 events.

The time reordering need a second RAM to store the length and the BCnt for each event. There are too many events to store the information in registers. However it takes 3 clocks to read from a RAM. This limitation applies to write the data in the RAM (need to know the length of the current event before writing) but also to read the data out of the RAM (to read the correct number of word from the RAM for the event).

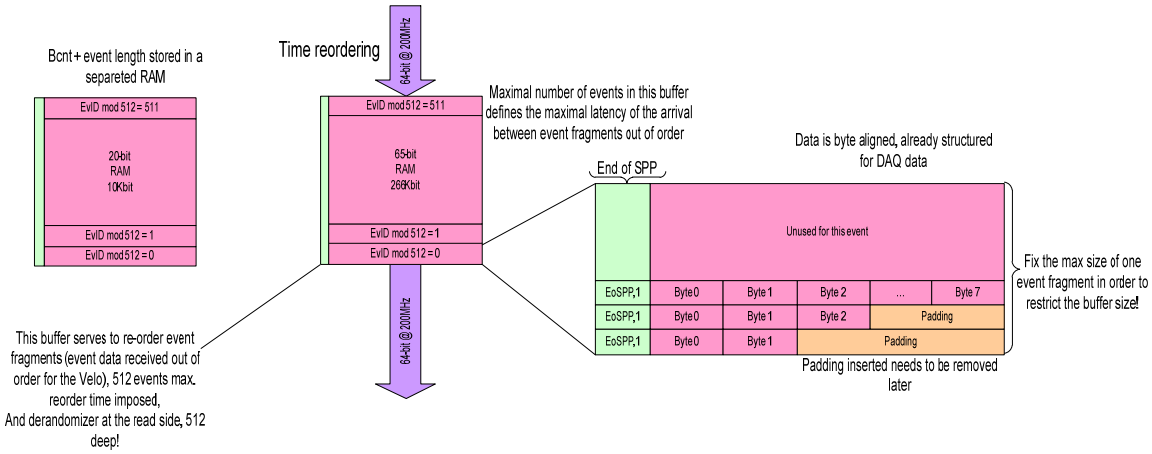


Figure 3-4 : Time reordering block diagram

Write:		
4 clk for SPP<64b	5 clk for 64b<SPP<128b	6 clk for 128b<SPP
Read:		
1 clk + 1 clk/word, but at least 4clk/event		

Figure 3-5 : Timing time reordering

3.3 Unpack SPP

To unpack the SPP format is difficult because the length and the position of the different fields are variable for each SPP, and in addition not easy to find. It depends on the length of all the previous fields, like said for the SPP reconstruction. An additional “link source id” (5bits) is required to identify the data from 24 different GBT links which has to be added to the date like an address extension.

To unpack the SPP, the length of the “hitmap” using the “row header”, the position of the “shared hitmap” using the “row header” and its length using the “shared row header”, the position and the length of the “ToTs” (Time over Threshold)/”shared ToTs” using the “hitmap” and the “shared hitmap” and finally the number of hit to unpack need to be considered.

As said before, with the nSPP format, it just needs to check the “hit count” field to know everything about the SPP: the length of the « hit address » fields, the position of the ToT and the number of hit to unpack. This new format really simplifies the SPP unpacking.

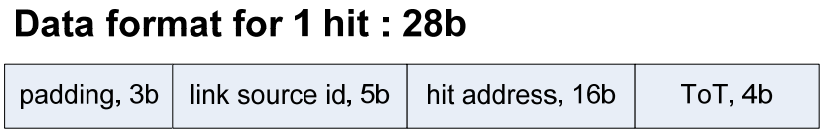


Figure 3-6 : Data format for 1 hit

3.4 Clusterization

Only 25ns for each pipeline step.

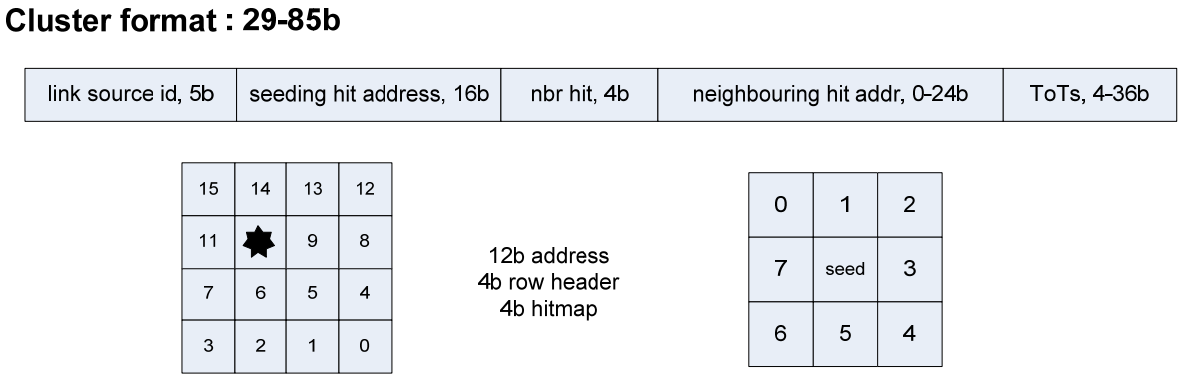


Figure 3-7 : Cluster format

The clusterization requires splitting up the SPP format because two isolated pixels can be in the same SPP. The most obvious approach for clusterization is to use one seeding pixel and search for possible neighbours. But it is very difficult to perform “perfect” clusters because average time per cluster is limited to 25ns if done in a pipeline, otherwise 25ns for the complete event! The 16b seeding hit address is reconstructed from the 12b address, the 4b row header and the 4b hitmap. The cluster form depends of the seeding hit, which is the first hit. One “normal cluster” can be split in two clusters.

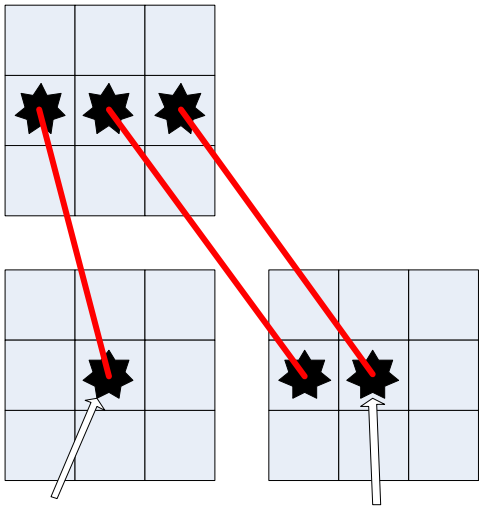


Figure 3-8 : Example of not perfect cluster

The cluster size is restricted to multiple of bytes! (Data processing on the FPGA but also on the CPU becomes very difficult otherwise). The expected data reduction from clustering taking for 50% 1-hit and 50% 2-hit clusters is order of 14%. The principal goal of the clusterization is data reduction, “perfect” clustering like for TELL1 is not possible anymore. Additional processing in a CPU is required to finish: Forming clusters over boundaries of GBT links, combining separated clusters and forming clusters for events with too high pixel count.

To pipeline the cluster search, only one cluster per pipeline step is formed. One pipeline step takes 25ns. In average the hottest region has 2 to 4 pixels “only” per event and per GBT (10 to 20 pixels per chip)! Choosing a number of 6 stages should do enough clusters for most of the event. The total number of clusters that can be formed is limited by the number of pipeline stages.

The cluster search is performed by searching neighbors from the first hit in the data. Each consecutive pipeline stage has the same function. The functioning of the one stage is: the block “cluster search” take the one hit in the “FIFO hits not tested” of the previous stage and compare it with the other hits of the “FIFO hits not tested” of the previous stage, if the hits are not a cluster, they are written in the “FIFO hits not tested”. Once all the hits of the current event have been tested, the cluster is written in the “FIFO clusters”. Then the clusters already formed for the current event are moved from the “FIFO clusters” of the previous stage to the “FIFO clusters of the next stage.

At the moment, the clusterization has been given up because it would need too many resources for 24 links, for only a few data reduction.

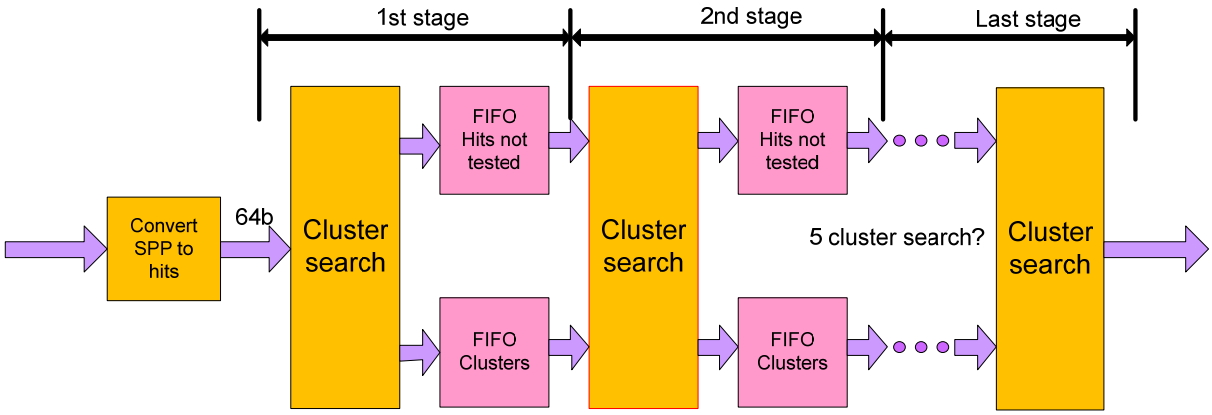


Figure 3-9 : Clusterization block diagram

	With clusterization	Without	Data reduction
1 hit	29b => 32b	25b => 32b	0%
2 hits	36b => 40b	50b => 56b	28.5%
3 hits	43b => 48b	75b => 80b	40.0%
4 hits	50b => 56b	100b => 104b	46.1%
5 hits	57b => 64b	125b => 128b	50.0%
6 hits	68b => 72b	150b => 156b	53.8%

Figure 3-10 : Cluster format data reduction

4 Data processing part for all the detector

4.1 Linker + Padding

The principle of the TELL10 data processing is to assemble the data from the links, 2 or 3 links each time, to avoid that the number of clock spent for each event is more than 5. There are three linking stages in TELL10 as we can see it in the data flow. The first stage assembles 3 links because the probability to have no data on one link is higher than for the 2 next stages. Each time the data from different links are assembled together, the bus width needs to be increased to increase the data rate.

At the beginning, there was a fourth stage with a 512b wide but it required really large multiplexers (a 512b bus requires for byte padding a multiplexer of 512x512/8=32K connections) that introduce timing problem. So we decided to split up the data processing design in two to run two independent 256b MEP stream with a external DDR3 SDRAM bus with the same width, removing the stage4 (linker4 + padder4, 512b width). It gives us more flexibility, allowing to place and route the design more freely in the FPGA, especially with the I/O, to obtain better timings.

The principle of padding is to place the next data right after the last byte of the previous data, watching the data valid, to group the data of the same event, reducing the data loss. Padding is done only for data from the same event, except in the MEP assembly,

It is possible to change the byte alignment of the padder, it allow reducing the number of interconnections for multiplexer, reducing the number of logic needed and improving the timings.

Padding stage	Bytes aligned	Data in width	Data out width
1	1	64b	128b
2	2	128b	256b
3	4	256b	256b
MEP	4	256b	256b

Figure 4-1 : Byte padding alignment for the different stages

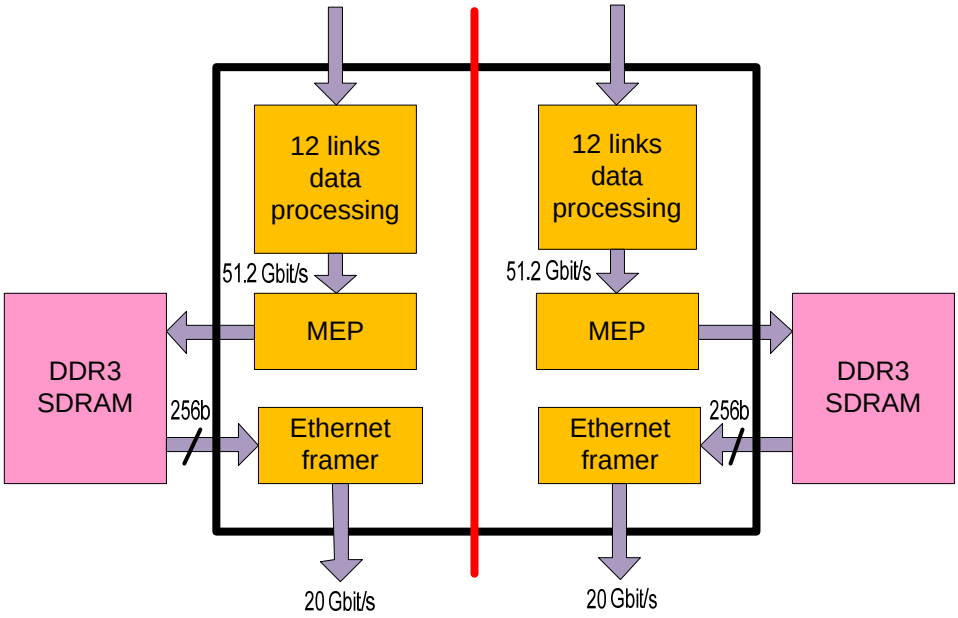


Figure 4-2 : TELL10 split in two independent MEP stream

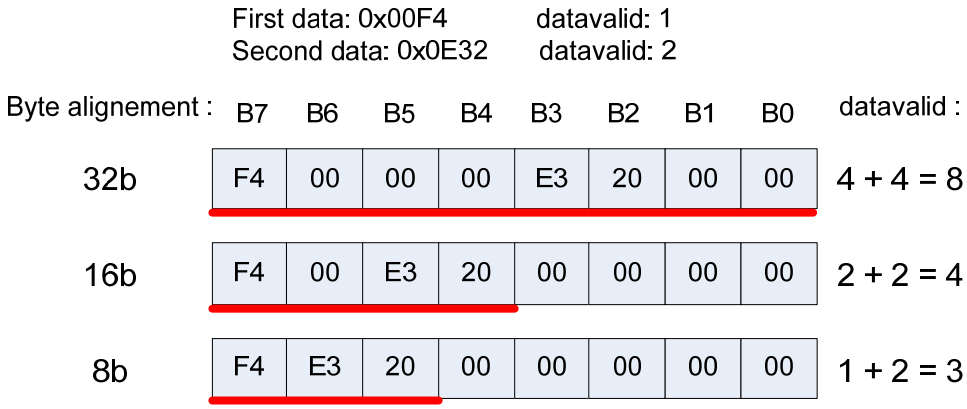


Figure 4-3 : Padding example

4.2 Multi Word Packet (MWP) assembly

The 4GB DDR3 SDRAM 256b wide is used to de-randomize the data rate. It can store up to 625ms without reading. The total bandwidth is 51.2 Gb/s for input and output. The average data rate will be balanced between read and write. The utilization of an external memory is needed because the internal memory of the FPGA is small and the network needs considerable buffering if for example a pull protocol is implemented.

The memories that are implemented on the board (SO-DIMM) do not have parity bits implemented so some bits must be reserved to ensure correct data transmission. In addition data valid bits in order to flag the valid section of each word need to be added. So, 8b data valid and the 16b checksum are added.

For an effective use of the data bandwidth of the memory, event based read write operations can not be allowed. Block transfer of data of previously assembled data in transmission buffers have to be used. We introduced the MWP format for this reason. For each data word, the data valid, some flags and the checksum are written in a 32b info packet. Once seven words have been read, the seven info words form the header with the 16b checksum for the header. So at the output of this block, there will be one header word every seven data words. This introduced a data loss of 12.5% but allows to have a efficient usage of the bandwidth and data consistency checking implemented.

The BCnt is not anymore given with the event. There is only one bit “End of event” that is set to ‘1’ for each last word of the event. This bit will be used to reconstruct a new event ID after the DDR3 SDRAM. Another important flag is the “Event empty” to tell that an event have no data word.

Another possibility for the MWP would be to reduce the data from 256b to 224b and to use the 32b for the data valid, the flags and the checksum. But this solution introduced two padder instead of one.

Explain this

At the moment, the checksum is calculated for 16b. It's possible to calculate it for 32b to use one header word for fifteen data word instead of seven which allows to make the loss of bandwidth smaller and the block transfer size bigger.

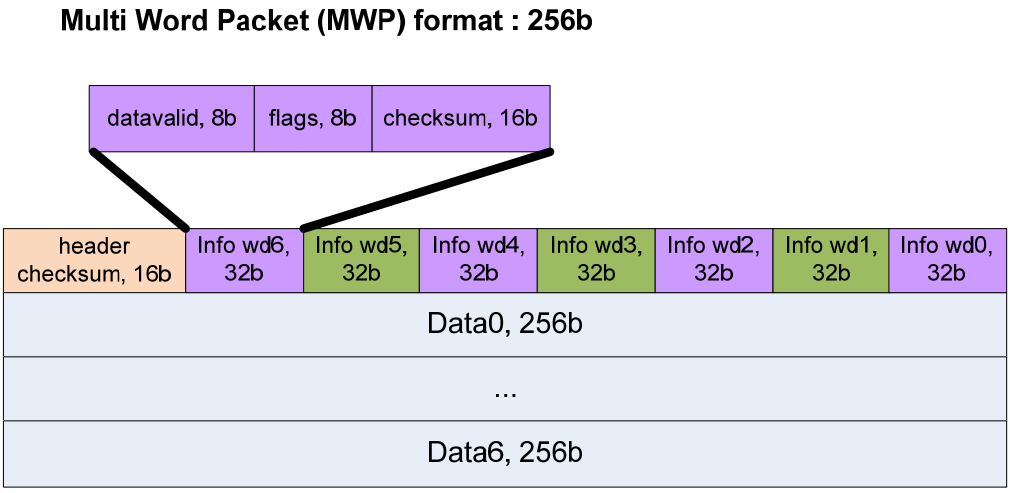


Figure 4-4 : MWP format

4.3 Level 0 Trigger (L0T)

The level 0 trigger tells for each event if the data for the current event should be kept or deleted. The decision is read from a 33bx4096, which can store up to 100us of event. A decision is constituted of a 32b event id and 1b for the decision. The event linked is not linked to the BCnt. It's reconstructed from the bit "end of event". The L0T is placed after the DDR3 SDRAM to allow a large time for the trigger decision to arrive.

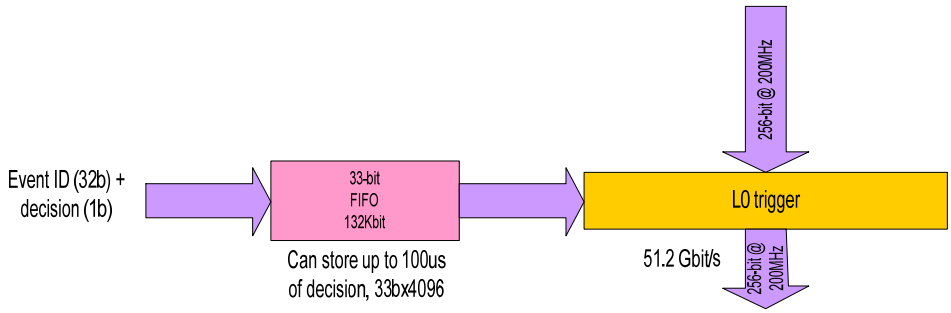


Figure 4-5 : Level0 trigger block diagram

4.4 Multi Event Packet (MEP) assembly

The MEP assembly in TELL10 keep the same principle as in TELL1, one header word followed by a number of event defined by the MEP factor. The goal of the MEP is the data reduction, not adding a word of header for each event. The data from several events are kept in a buffer, in order to lower

the packet rate on the network. After the MEP, there are no more data valid required since the length is encoded in the data.

The header word is composed of the event ID of the first event in the MEP, the length of the MEP in bytes and the length of the MEP in event (MEP factor). Then comes the data, for each event, it starts with the length of the event in bytes following by the data (hit, SPP or cluster). For the event empty, the length will be zero and there will be no data for the event.

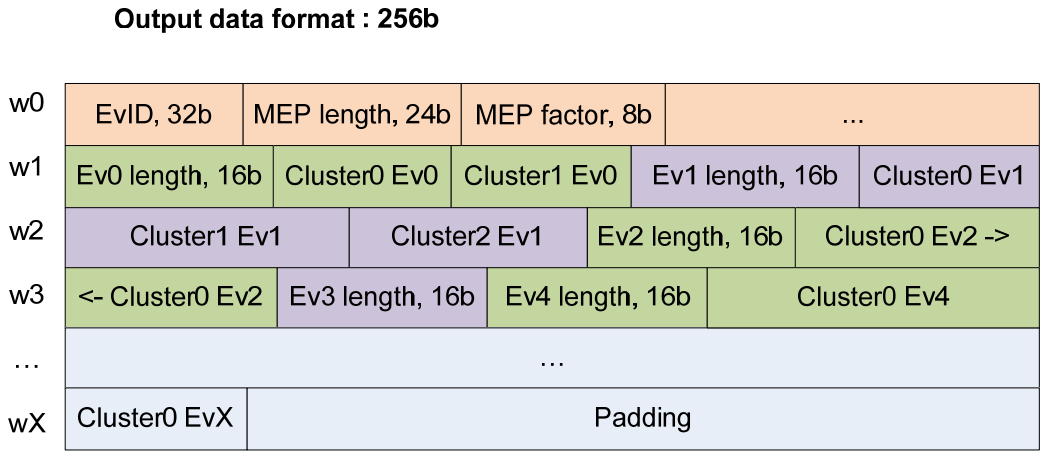


Figure 4-6 : Output data format

5 Conclusions