20/08/2015

The New Small Wheel

Summer student report

BOUKLI HACENE Ghouthi

Abstract:

The E-Link is the part that has to be added to the firmware in order to allow communication and configuration through the E-Link, which is a serial link. In this step we have to integrate the E-Link, so the data must be encoded using 8b/10b encoding and serialized before send it out through the E-link. We have to use the Ethernet FIFO as the source of data, which will be used to communicate with the L1DDC (Level-1 Data Driver Card).

Introduction:

The Phase-I upgraded of the LHC provides now a high luminosity performance, and in order to benefit from this the first station of ATLAS which is the Small Wheel (SM) has to be replaced. The New Small Wheel is the project for replacing the SM and will operate in a high radiation background region (up to 15 kHz/cm2) while reconstructing muon tracks with high precision, as well as furnishing information for the Level-1 trigger.

The NSWs consist of eight Micromegas layers, and in this large project, there is a specific part, which is the development of the Micromegas Front-end board (MMFE8).

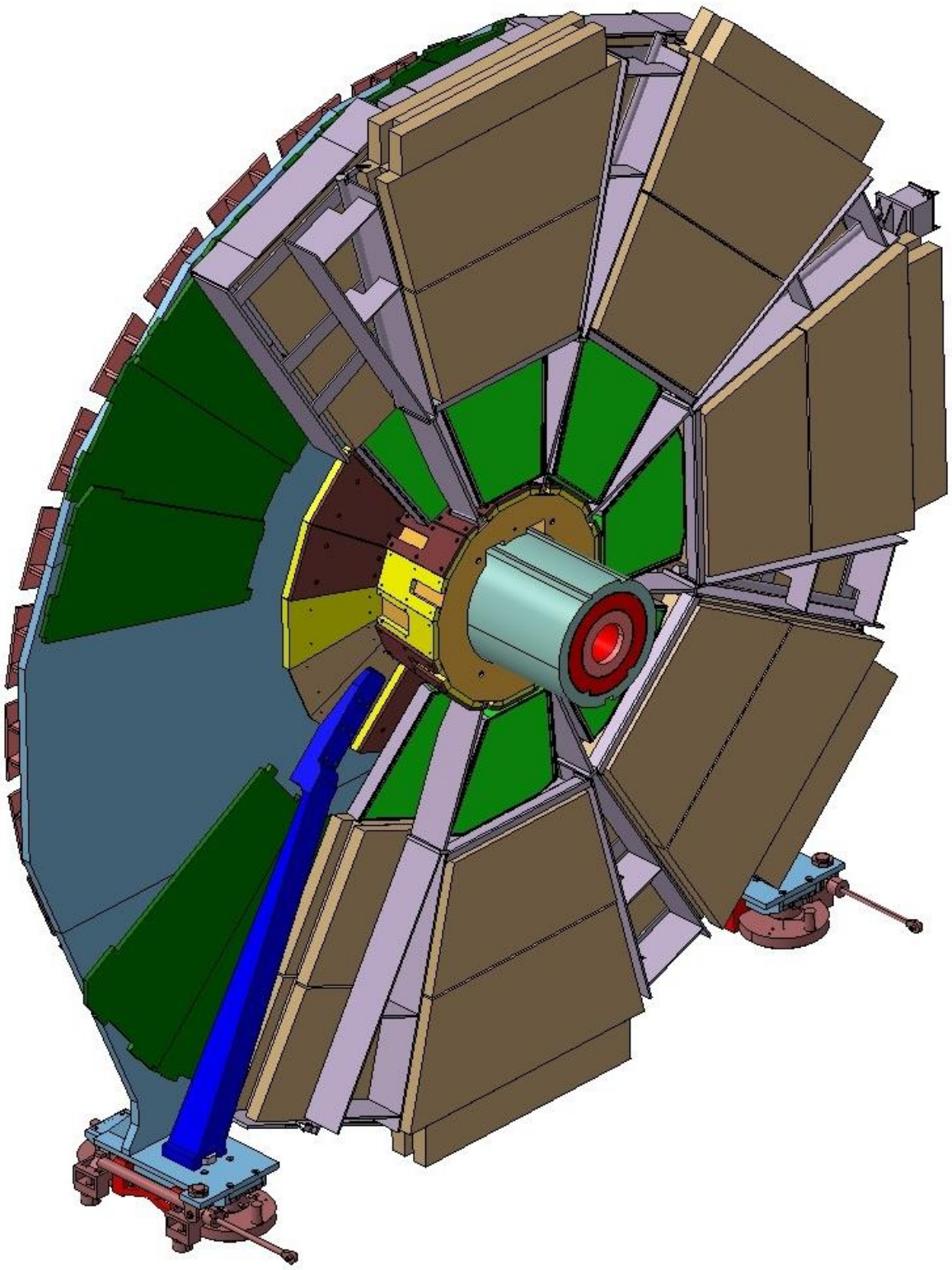
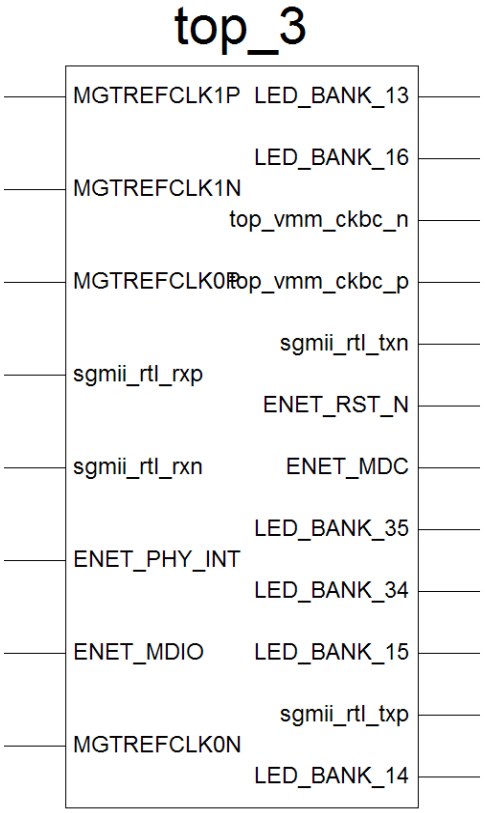


Figure 1 The New Small Wheel

MMFE8 :

The project of the front end electronics (MMFE8) involves the developing of a firmware which contains all functions as readout, communication and configuration via Ethernet, write data, storage data, etc. The MMFE8 is a board which contains 8 VMM. Its firmware is presented as a global bloc (Top-Level) which contains many blocks. So we have different clocks with different frequencies which are used for the synchronization and some registers to manage data information. There is a component (mbsys) which was not described by coding but it was done by vivado thanks to some ip-cores, like micro-blaze processor and the VHDL corresponding to this schema description was generated and integrated to the whole project (Top-Level).

This is a bloc of the Top-level (of the first version), with all of its in/out ports. There are two input clocks and the Ethernet communication. The main parts of the firmware are: communication and configuration parts which allow to choose one of these VMM, readout the data, encode this data and serialize it before sending it out through the E-link to the L1DDC (this part is not implemented yet) or to the PC via Ethernet.



Two signals are usually used for communication (for example Ethernet): positive and negative signals and comparing the difference between them to get the logical data. Because it's more easier to get data by this way. For this we use two important blocs which allow to get two signals (positive and negative, called differential signal too) from the logical signal (and vice versa) and these components are:

OBUFDS (out-put buffer differential signal) and IBUFDS (in-put buffer differential signal) which exist in Vivado by default that means we can use them without declare them.

The entity of these two important components is:

```
entity OBUFDS is
port(
        signal_in:in std_logic; --the logical signal
        positive_signal: out std_logic;
        negative_signal: out std_logic
);
end entity;
entity IBUFDS is
port(
        positive_signal: in std_logic;
        negative_signal: in std_logic;
        signal_out:out std_logic --the logical signal
end entity;
```

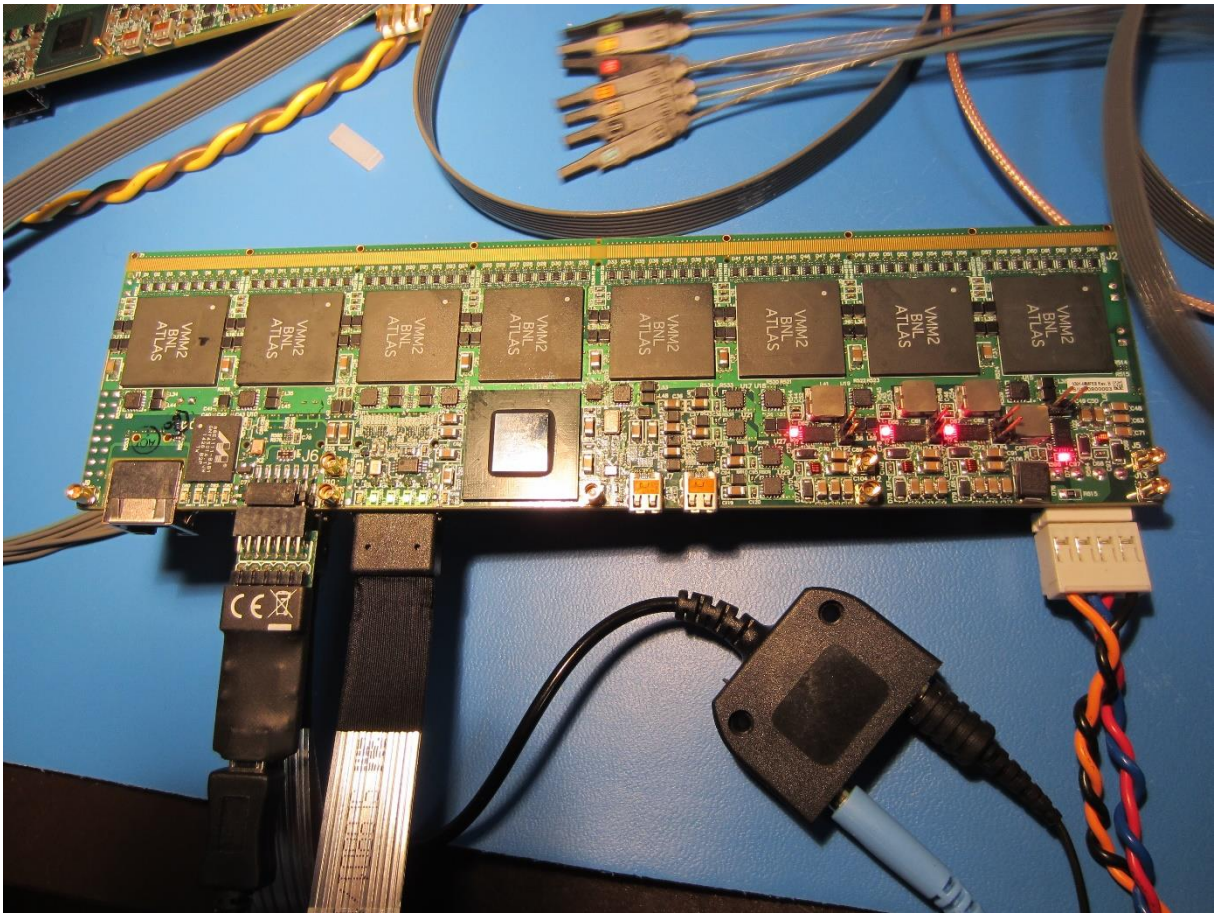


Figure 2 MMFE8

E-Link

The E-Link is the part of the of the firmware used for communication between MMFE8 and L1DDC. It consists of two main blocks which are : ELINK2FIFO and FIFO2ELINK.

```
entity FIFO2Elink is
generic (outputRate : integer := 80); -- 80 or 160 MBps
port (
   clk80        : in  std_logic;
   clk160       : in  std_logic;
   rst          : in  std_logic;
   eFIFOwclk                  : in  std_logic;
   eFIFOwe      : in  std_logic;
   eFIFOpfull                 : out  std_logic;
   eFIFOdin     : in  std_logic_vector (17 downto 0);
   eout         : out  std_logic
   );
end FIFO2Elink ;
```

```
entity Elink2FIFO is
generic (inputRate : integer := 80); -- 80 or 160 MBps
port (
   clk80        : in  std_logic;
   clk160       : in  std_logic;
   rst          : in  std_logic;
   ein          : in  std_logic;
   eFIFOrclk                  : in  std_logic;
   eFIFOre      : in  std_logic;
   eFIFOpfull                 : out  std_logic;
   eFIFOdout    : out  std_logic_vector (15 downto 0)
   );
end Elink2FIFO ;
```

Figure 3 ELINK interface

- The FIFO2ELINK consists of two blocs which are : eFIFO, and e-prouk_out.
  The eFIFO is a FIFO where data is stored. The size of this data are 18 bits, 2 additional bits as data code (“00”= data, “01”= EOP, end of packet, “10”= SOP, start of packet).
  The eFIFO interface is:

```
entity eFIFO is
port(
    eFIFOwclk : in std_logic;
    eFIFOwe : in std_logic;
    eFIFOpfull : out std_logic;
    eFIFOpempty : out std_logic;
    eFIFOre : in std_logic;
    eFIFOrclk : in std_logic;
    eFIFOdin : in std_logic_vector (17 downto 0);
    eFIFOdout : out std_logic_vector (17 downto 0)
    );
end entity;
```

Figure 4 eFIFO interface

the second component is the e-prouk_out. It receives the data directly from the eFIFO, encodes this data using 8b/10b encoding and serial this encoded data in order to send it through a serial link. The e-prouk_out interface is :

```
entity e-prouk_out is
port(
    EPROUC_in: in std_logic_vector(17 downto 0);
    clk80 : in std_logic;
    clk160 : in std_logic;
    rst : in std_logic;
    re: out std_logic;  --read enable for reading new data from eFIFO.
    eout : out std_logic

    );
end entity;
```

Figure 5 e-prouk_out interface

- The ELINK2FIFO consists of two blocs : eFIFO_16 and e-prouk_in.
  eFIFO_16 is a FIFO where the data size is 16 bits. Its interface is:

```
entity eFIFO_16 is
port(
    eFIFOwclk : in std_logic;
    eFIFOwe : in std_logic;
    eFIFOpfull : out std_logic;
    eFIFOpempty : out std_logic;
    eFIFOre : in std_logic;
    eFIFOrclk : in std_logic;
    eFIFOdin : in std_logic_vector (15 downto 0);
    eFIFOdout : out std_logic_vector (15 downto 0)
    );
end entity;
```

Figure 6 eFIFO_16 interface

The second component is the e-prouk_in. it receives serial data from FIFO2ELINK, it deserializes the data and decodes it using the 8b/10b decoding, and at the end it sends the data to the eFIFO_16 to store data there. The e-prouk_in interface is:

```
entity e-prouk_in is
port(
        clk80 : in std_logic;
        clk160 : in std_logic;
        eFIFOpfull : in std_logic;
        rst : in std_logic;
        ein : in std_logic;
        we: out std_logic;  --write enable for reading new data from eFIFO.
        inFIFO : out std_logic_vector(15 downto 0);
        packet : out std_logic_vector(15 downto 0)
    );
end entity;
```

Figure 7 e-prouk_in interface