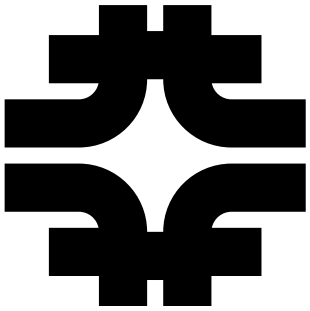


FERMILAB

Mu2e Experiment

Firmware development for Mu2e electronics

Internship Report

Intern student:
Ian Biagioni

Supervisor:
Vadim Rusu

October 2016

Contents

List of Figures

Abstract

The following document describes the design flow to implement the firmware for the onboard communication for the electronics used in the tracker in the Mu2e experiment.

A brief overview of the experiment setting, components and interfaces for communication are given before starting the project development description.

The design process is described step-by-step, understanding the architechture proposed and discussing the problems encountered during the design.

Chapter 1

Setting overview

1.1 The tracker

The Mu2e experiment is meant to study the conversion processes amongs particles of the same family, like muons and electrons. To do that, charactestics of the electrons generated from muons conversion have to be measured. The tracker is used to look for their trajectory. It is composed by 20 stations supported by a rigid frame. Each station consists in 2 planes which are composed by 6 panels. Each one of them has two layer of straws that are sensitive to the electrons transit.

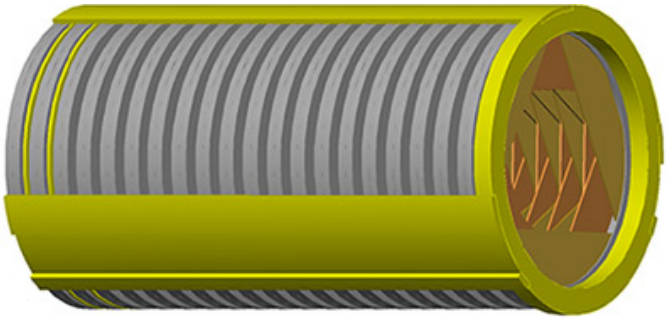


Figure 1.1: Tracker

The tracker will operate in a vacuum, with 1 atmosphere of ArCO2 gas (80%/20%) inside the straws. A charged particle passes through the straws, leaving a trail of ions in its wake. The inner surface of the straw is at ground, and the sense wire at the center is a 1500V positive voltage. The electrons then drift toward the sense wire.

The electric field near the sense wire is strong enough that the electrons gain enough kinetic energy to ionize more atoms of the gas, creating an avalanche, which can be detected by the electronics.

Each straw is connected to a preamplifier circuit mounted on the AMB (Analog MotherBoard). The signal is then transmitted to the DRAC through the DMB (Digital MotherBoard) to be collected.

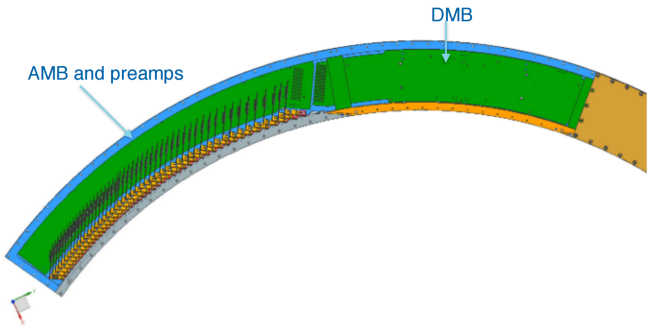


Figure 1.2: Panel overview

1.2 DRAC

The DRAC (Digitizer Readout and Assembler Controller) is the central unit that digitize and collect all the signals from the straws. There are two FPGAs that control and receive the signals from the ADCs and TDCs used to convert the analog signal coming from the straws. These two FPGAs have been called HV and CAL. The names are unrelated to their function.

There is a third FPGA called ROC (ReadOut Controller) that collects and organizes the datas from HV and CAL before sending them to an external memory. The purpose of this project is to develop a firmware for communication between these three FPGAs.

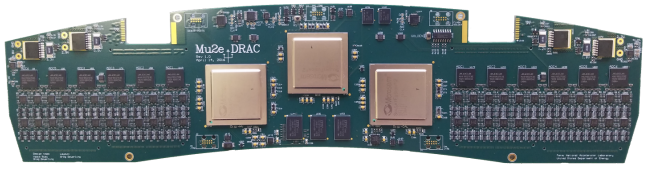


Figure 1.3: DRAC board

Chapter 2

Design flow

2.1 Communication interface

Due to the high number of signals to be digitized, a considerable part of the DRAC is occupied by the ADCs and TDCs so that serial communication between FPGAs has been prefered instead of parallel comunication to saves FPGAs pins and to reduce number of tracks on the board.

2.1.1 SerDes

SerDes stands for Serialized/Deserializer and it generally refers to all the interfaces used to send serial data on a single/differential lane starting from parallel data.

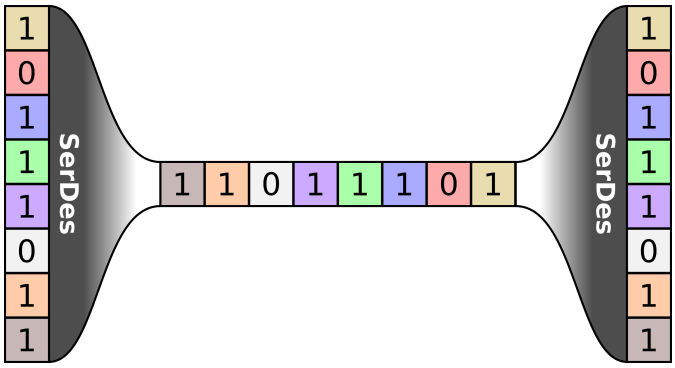


Figure 2.1: SerDes concept

On the transmitter side, a PISO block may use an internal or external PLL to multiply the incoming parallel clock up to the serial frequency. The simplest form of the PISO has a single shift register that receives the parallel data once per parallel clock, and shifts it out at the higher serial clock rate.

On the receiver side, a SIPO block collect the serial data and recover the transmitting clock from the stream received. It then divides the clock down to the parallel frequency at which the datas will be available.

2.1.2 8b/10b encoding

8b/10b encoding is a line code that maps 8-bit words to 10-bit symbols to achieve long term DC-balance, bounded disparity and provide enough state changes to allow reasonable clock recovery. The 10b code is generated splitting the 8 bits and using a 5b/6b and 3b/4b encoding. There is a special sequence in the 6 bits code that is used to send control characters defined with the 4 bits code.

Due to its characteristics, 8b/10b encoding is commonly used in serial communication like Gigabit Ethernet, PCIe and USB 3.0.

2.2 Development platforms

The FPGAs used on the DRAC belongs to SmartFusion2 family from Microsemi Corporation. Their development environment has been used:

- Libero SoC Design Suite 11.7 SP1.1 for system design
- SoftConsole 4.0 for software development
- FlashPro4 Programmer for FPGAs programming

Figure 2.2: FlashPro4 programmer

For the system testing, two different hardware settings have been used in different design phases.

2.2.1 M2S150T

The FPGAs mounted on the DRAC are the M2S150T-FC1152. The characteristics relevant for chosing this model or for the communication firmware are:

- ARM Cortex-M3 Processor
- 4 SerDes interfaces with up to 16 total serial lanes
- SEU protected memories
- 150K LE
- UART lane

The protection from high energy events is needed due to the characteristic of the experiment. High number of logic elements is relevant for the complete system but is not an important factor for the part developed in this work.

2.2.2 M2S050T

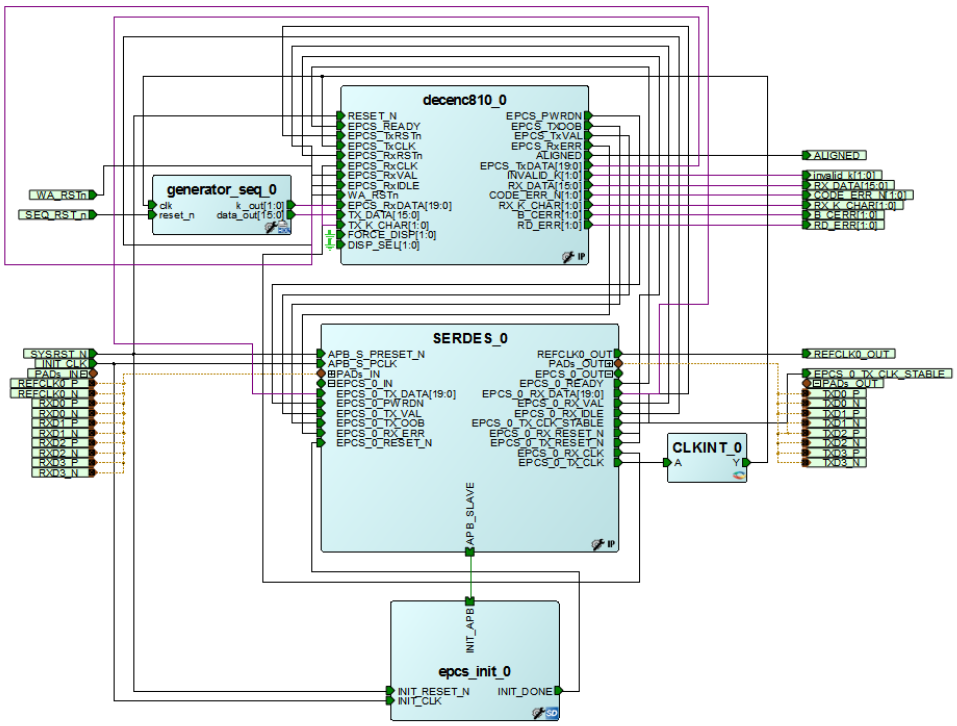
The high number of LE presents on the M2S150T affects the synthesis time. To speed up the testing process, first designs have been tested on a smaller FPGA. In particular the M2S050T mounted on the SF2-STARTER-KIT-ES-2 board. This FPGA belongs to the same family than the M2S150T so they have similar characteristics except for having 50K LE instead of 150K and just one SerDes interface instead of four.

2.3 Initial simulations

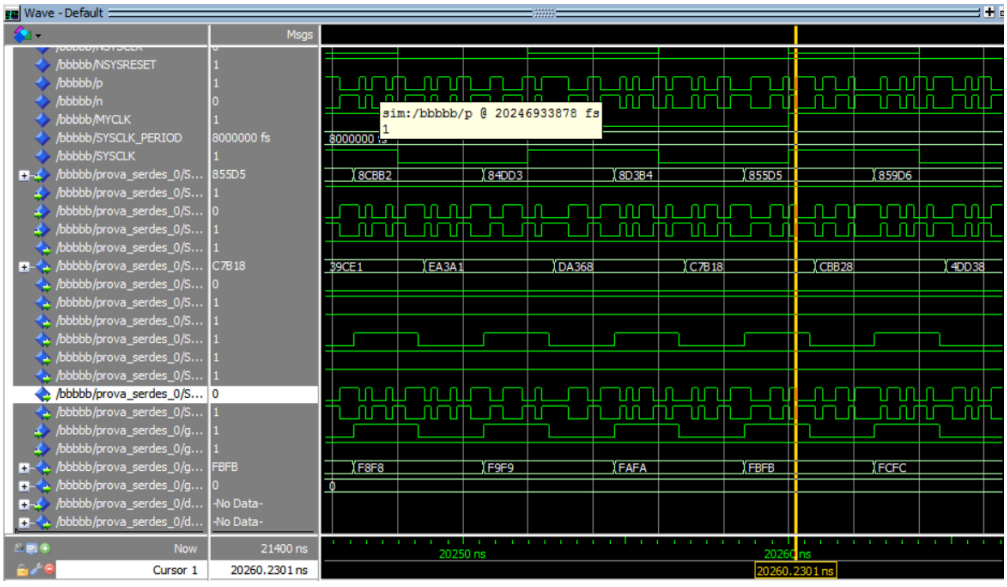
Microsemi provides two IPs usefull for the purpose of this work:

- *High speed serial interface* is a configurable IP that permits to transmit and receive data on the four lanes of a SerDes interface. The configuration consists in chosing the SerDes interface used, the protocol for the comunication, the reference clock frequency and the values for the configuration registers
- *CorePCS* is a configurable IP used for encoding, decoding or both for the 8b/10b code.

First goal has been to have a design to be simulated to look for the behaviour of these two IPs. The *CorePCS* has been configured as receiver and transmitter. This IP took 16 parallel bits as input, corresponding to two symbols to be encoded, and two control bits to define if the input is a data symbol or a control symbol. These inputs are generated using a counter with some logics to generate control signals. The *High speed serial interface* is configured to use only one lane with a custom communication protocol (EPCS) and the reference clock at 125 MHz, corresponding to a 2.5 GHz serial communication. Setting a bit in the *PRBS_CTRL* register, the *High speed serial interface* has been used in loopback configuration so that the transmitted bit are also received by the same interface. The data received are decoded by the *CorePCS* and used to check if they are correct. Another block is used to simulate the initialization for the *High speed serial interface*.



(a) Design



(b) Simulation

Figure 2.3: SerDes implementation

2.4 On-board tests

Even though the previous design could be successfully simulated, it is still not completed for on board testing. For simulation purpose, the embedded processor could not be instantiated into the project but it come to use for on board testing and it is needed for the final system implementation.

First designs have been tested on the M2S050T FPGA to speed up the testing process.

Costraints for the routes timing have to be taken into account to generate a working design. Footprint costraints are not needed thanks to the simplicity of the design and the small dimension of the FPGA.

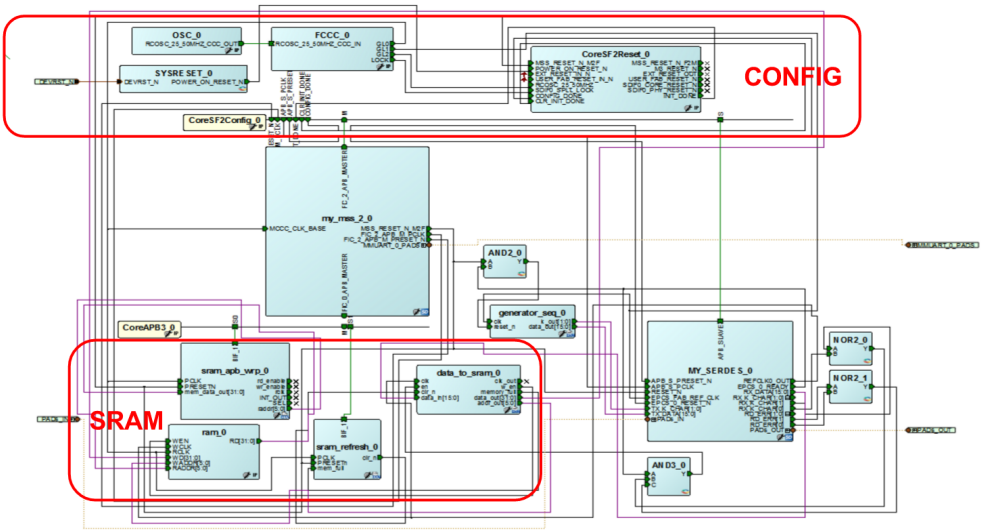


Figure 2.4: System design

To achieve a design functional for testing, the processor only needs to provide a link to commmunicate with the system. In particular it provides an APB bus towards the system and a UART communication towards a console. In addition to the processor, other functional block have been added to the project:

- Reset and configuration blocks
- Blocks for buses management
- Some logic for interfacing with a SRAM

Reset, configuration and buses management are provided by Libero along with documentation and setup examples . Interface with SRAM needs further description.

2.4.1 SRAM interface

FPGAs from SmartFusion2 family provides SRAM blocks for memorization purpose. A memory is needed to store datas received through the SerDes interface waiting to be read by the user. To achieve this functionality, the SRAM interface can be splitted in 4 parts (figure 2.5):

- *ram* : SRAM block provided by Libero. Since the reading speed depends on bus clock and the writing speed depends on receiving clock from SerDes, in the configuration options, separate clock interfaces for reading and writing have to be set. Word width is 32 bits.
- *data_to_sram*: A writing block from SerDes to SRAM. It waits for two set of datas to be received then it generates the address for the SRAM and the logic to enable writing. The address is simply generated with a binary counter. SRAM dimension can be set so that the block knows which is the last location in the memory. *memory_full* signal is asserted reaching the last address of the SRAM.
- *sram_apb_wrp*: A reading block from SRAM to processor. It provides a slave interface to connect the SRAM to the APB bus. Reading from the slave addresses area is translated into reading from SRAM addresses.
- *sram_refresh*: A reset block for SRAM contents connected to the APB bus with a slave interface. Write action to the slave first location triggers the *clr_n* signal that resets the counter in the writing block. It checks the *memory_full* signal to check if the resets has been done.

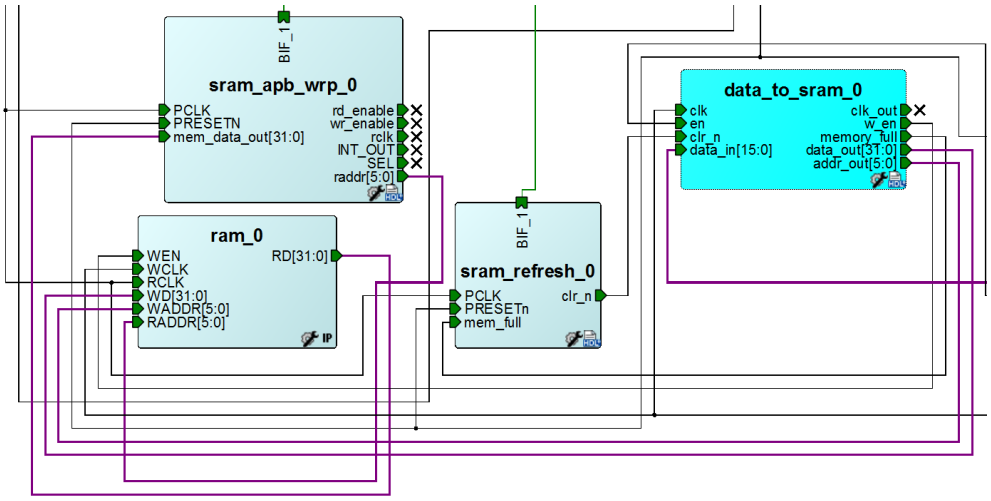


Figure 2.5: SRAM interface

2.4.2 Timing costraints

Libero provides a tool to automatically generate basic timing costraints for the design. Due to the low complexity of this system, these costraints are sufficient for the synthesis, placing and routing processes to be correct.

The maximum clock frequency that could be achieved in the implementation has been 100 MHz for both the processor/bus clock and the SerDes clock. This implies a slower communication speed, from 2.5 Gbit/s to 2 Gbit/s for the single lane. This issue will be discussed in more details in the final chapter.

2.5 DRAC tests

2.5.1 Porting

Some precautions have to be taken into account for porting the system to the M2S150T FPGA.

- Change the FPGA associated with the project file and update the components
- Revising the timing constraints
- Definition of footprint constraints
- If needed, revising the design

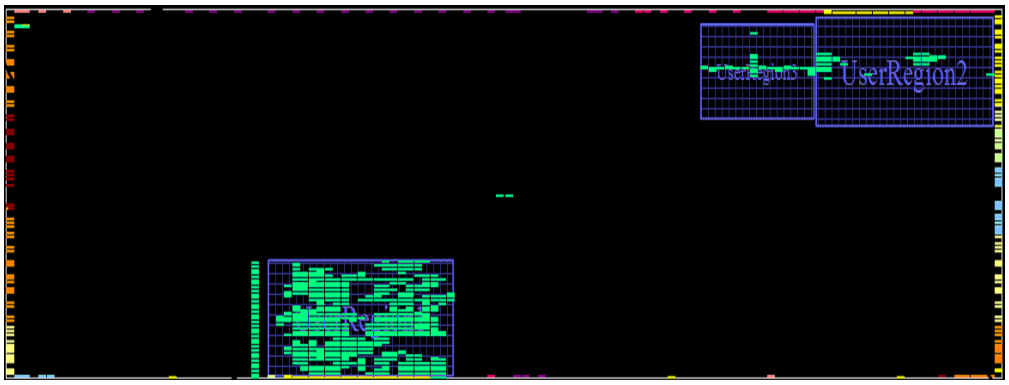
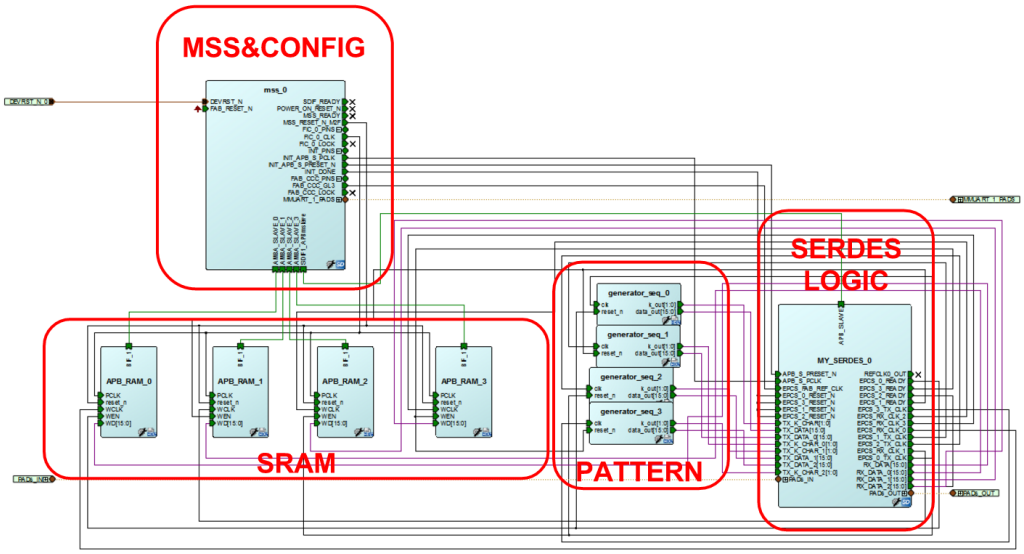


Figure 2.6: M2S150T footprint constraints

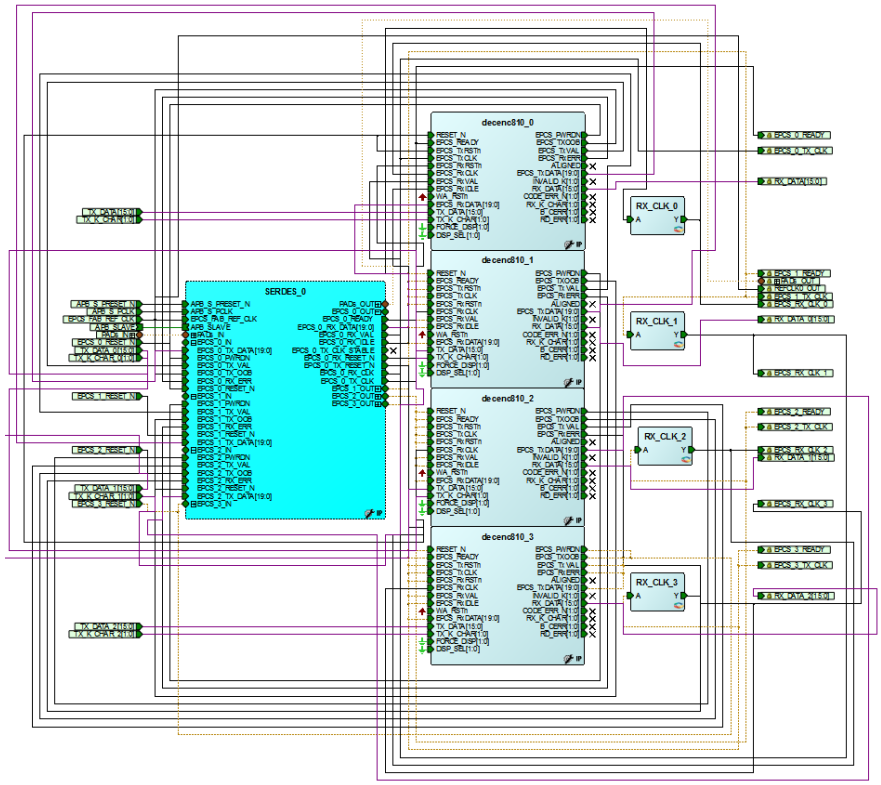
Even after the changes needed to satisfy all the constraints, the tests on the design have had negative results. It came out that the buses, the initialization block and the reset block were not working properly. Even though the reason for this behaviour is still not clear, the solution has been to use a wizard provided by Libero to automatically generate the processor and all the logics related to the buses, resets and initialization. Since the wizard was not providing some options needed for the design, the generated logic has been manually modified acording to the needs.

2.5.2 Upgrade

Each SerDes interface has four differential lane for communication and all of them should be used. The *CorePCS* block and the ram with its logics have to be duplicated for each lane. Having some problems resetting the SRAM, changes to the design have been made. In particular SRAM logics have been simplified. The memory reset is triggered by a signal from the processor that can be sent via software by the user.



(a) General design



(b) SerDes logics

Figure 2.7: New design

2.5.3 Communication between the FPGAs

All the designs tested so far, had the SerDes interface set in loopback configuration. Communication between FPGAs has to be tested.

For HV and CAL the previous design can be used changing the SerDes interface configuration so that the one connected to the ROC is used. For both of them, SerDes_1 is the interface to be chosen.

Having to communicate with both HV and CAL, ROC needs two SerDes interfaces, in particular SerDes_1 for HV and SerDes_3 for CAL. Adding one interface to the ROC project causes timing problems that could not be solved. To test the communication functionality, connections with HV and CAL have been tested separately.

In loopback configuration, syncronization issues between transmitter and receiver could not be spotted. Communication test among different FPGAs spotted these issues resulting in malfunctioning connections. 8b/10b encoding uses commas to syncronize the communication. *CorePCS* needs at least a certain number of consecutive commas to achieve syncronization but this quantity was not reached in previous communication. Increasing the number of consecutive commas transmitted, functionality of the communication has been restored.

2.5.4 On-board SerDes clock

For all the tests, the clock for the SerDes interface has been generated by a PLL at the wanted frequency. On the DRAC, a 156.25 MHz crystal should provide the clock signal for the SerDes interface. The SerDes IP can operate at that frequency only transmitting and receiving 16 bit words with a data rate of 2.5 Gb/s. This crystal can't be used with 8b/10b encoding that needs to send 20 bits per word.

Chapter 3

Solutions and implementation

3.1 Two SerDes on ROC

Even if the communication between two board has been achieved, simultaneous communication from HV and CAL to ROC could not be obtained. Two possible solutions have been proposed:

- Use the 156.25 MHz on-board crystal for the SerDes interface. 8b/10b encoding should not be used anymore.
- Communication from ROC to CAL and HV is not needed so the link between them can be unidirectional. Doing that *CorePCS* can be setted as transmitter or receiver only.

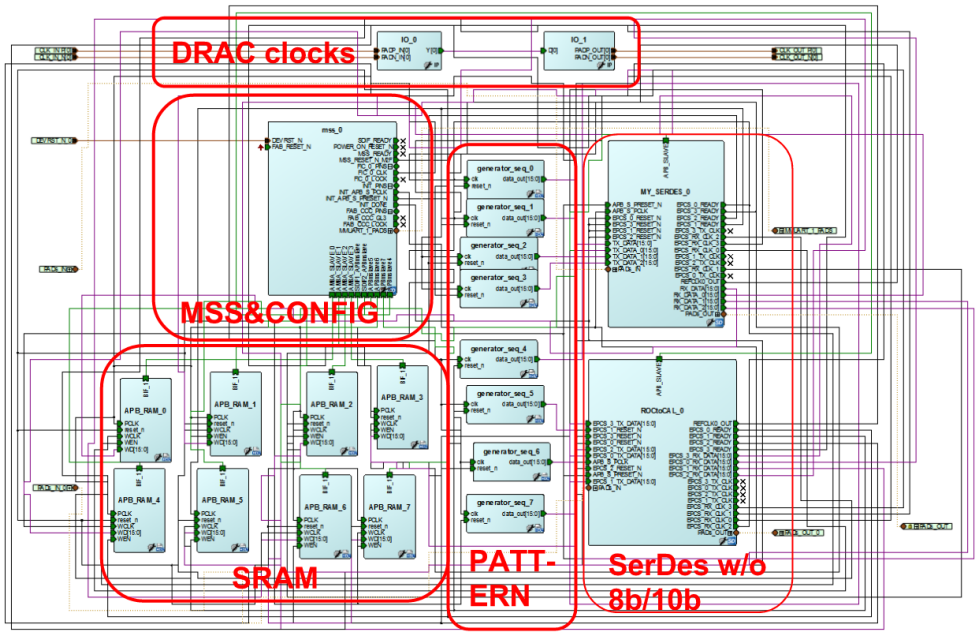


Figure 3.1: ROC project without encoder

The first solution could resolve all the timing problem and could succesfully transmit and receive datas. Not using a code, syncronization is harder and the system suffered of bit slip. The second solution had better timing margins than the case without any solution applied but most of the logic for the *CorePCS* is used in the decoder. So the ROC still suffered for timing problems.

3.2 FIFO

Datas arrives from the ADC at a frequency that is different from the one used for SerDes transmission. Moreover, datas are generated by electrons that transit through the straws and the number of event per second is not constant. A FIFO is needed to connect collected data source to the SerDes interface. Some logic is needed too because datas generated from the ADCs are 12 bits of lenght but the encoder wants 16 bits. Filling the last 4 bits with control signals or bits from other datas are both worth options to be implemented depending on the critical issues that will be found.

In the clock domain transit, timing violation are showed. Splitting the clocks in different group, solves this problem. This is achieved through the timing constraints file.

3.3 Generic timing violations

Most of the timing violations showed were minimum timing violations. These violations are automatically solved by the synthetizer going through more iteration of synthesis and adding specific delays on each lane without violating maximum timing constraints. This can be done only if there are not maximum timing violations in the design.

The timing violations presented in this report are all maximum timing violations and are caused by an unexpected behaviour of the synthetizer.

There are 3 type of links inside the FPGA:

- Normal
- Local
- Global

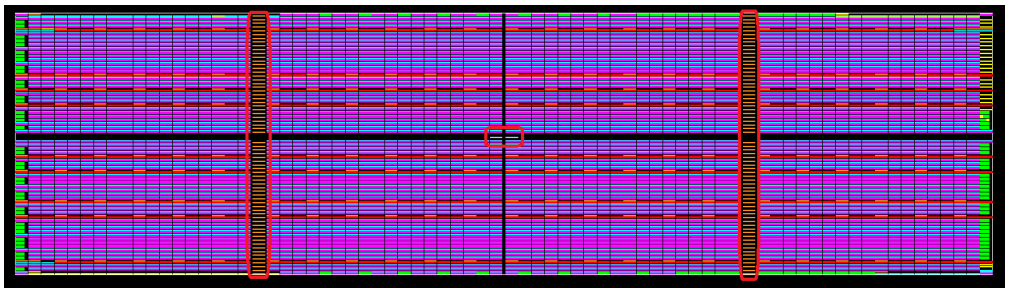


Figure 3.2: Footprint geometrical blocks with the two vertical lane of local connection and a central global connection

First ones are slow lanes with a low fanout. Second ones are high fanout lane that can be connected only inside the same geometrical block on the FPGA footprint. Global can distribuite signals everywhere on the FPGA.

The receiving clocks have to charge a lot of different logic element because of the big decoding circuit and the logic to write on the SRAM. To optimize the behaviour of the links, the synthetizer automatically promote to global the connections with highest fanout. But sending the clock signal from the SerDes zone to the global connections, adds too much delay to respect timing constraints. Specifing that these clocks have to be local signals, high fanout has been taken care of and there was no more delay on the connection.

Solving this issue, make all the previous design works better and faster. The SerDes speed for all the designs have been increased to 125 MHz transmission clock frequency resulting in 2.5 Gb/s data rate without anymore issues.