

CMS Pixel Front End Controller

(Final draft)

1. Introduction

The Front End Controller (FEC) is an interface between CMS Timing, Trigger and Control system (TTC) and the front-end detector electronics. Its main function is to provide correct timing and trigger signals to the front-end event pipeline delays and event buffering logic. In addition, the FEC has to be able to generate the necessary initialization and setup commands and provide the means for monitoring and diagnostics of the front-end hardware. This is an attempt to write a coherent document describing the Pixel FEC.

2. Requirements

- Basic CMS requirements (as per current CMS documents)
 - a. Provide correct timing and trigger signals to the front-end event pipeline delays and event buffering logic
 - b. Generate necessary initialization and setup commands
 - c. Provide necessary means for monitoring and diagnostics of the front-end hardware
- Specific CMS Pixel requirements
 - a. Continuous setup of the front-ends during trigger inactivity intervals to recover from Single Event Upset (SEU) effects
 - b. Non-standard 40 MHz serial communication protocol

3. Block-diagram

The FEC is located in the CMS control room and connected to the detector electronics by optical fibers and digital optical links^[1] on the detector side. On the FEC end, a DTRx4^[2] optical module is used. The FEC communicates with the TTC system via optical fiber as well and uses a TTCrq daughter board developed by CERN^[3] as an interface. The block-diagram of the proposed FEC design is presented in Fig. 1. In this example, a standard 9U x 280 mm VME card is assumed as a housing. Different sized VME cards could be used, but that would affect the number of channels. A density of 12 channels per card would allow up to ~200 FEC channels per VME crate. The card would use an extended addressing A32:D32 VME slave interface. The timing signals and broadcast commands from TTC system are decoded by the TTCrq board and distributed on the Timing Bus between FEC channels. A standard I²C interface is used to control the TTCrq board. A JTAG connector on the card provides an easy option for in-circuit re-programmability. The expectation is that there will be a little or no space left on the front

panel for LED indicators. Therefore, internal status registers of the FEC should provide enough status information to overcome this limitation.

4. General functionality

The expectation is that the FEC crates will be equipped with a networked VME processor or PCI to VME interface. In any case, a running operating system (e.g. VxWorks or Windows) will introduce significant speed and reaction time limitations on communication with the FEC. It will be very difficult to guarantee that in this configuration a processor could reliably deliver necessary data synchronized with the accelerator beam structure to multiple FECs in the VME crate. Because of that problem, a hardware controller with a memory is proposed to periodically send data to the front-ends. The following description is based on the use of the networked VME processor in the FEC crate.

One can envision several major functional tasks of the FEC. After power up of the front-end electronics, the FEC has to generate “cold start” commands for the pixel Readout Chips (ROC), Token Bit Manager (TBM) chips and other programmable front-end hardware. This can be done under software control by the VME processor in the crate. It is presumed that the VME processor will be able to obtain the necessary information from a database via network connection (Ethernet). This process has no requirements of beam synchronization and practically is unlimited in time (if less than 5..10 minutes). It is possible that FEC will check the status and receive some other information from the front-end during this procedure. This mode of operation is called INITIALIZATION.

After initialization is complete, the FEC will continuously generate 40 MHz clock signal, but will block embedded TTC commands. B-G0 commands generated by TTC system will be encoded as L1 decisions and recognized by the Pixel TBM chips. There are no requirements for beam-synchronized processes in this mode. The FEC may send commands at arbitrary time to TBMs and other front-end hardware and receive responses to these commands. This mode of operation is called STOP.

The next mode of operation is called RUN since it corresponds to a normal data taking. In this mode the FEC will pass through encoded TTC broadcast commands and Level 1 accepts. One of the major features of the RUN mode is the ability to enable and disable the ROC refresh cycle. The expectation is that the ROCs will require continuous re-setting of the pixel thresholds during beam operations. It is possible that some other ROC registers will require refreshing as well. It is undesirable to refresh pixel settings during actual data processing in the ROC, therefore, refreshing during marked by TTC trigger gaps (“private gaps” and “private orbits”) is considered as possible solution to avoid changing pixel setting while processing L1 accepts.

The algorithm of refreshing ROC settings must be simple and efficient. The first approach will be to cycle through all the necessary settings periodically using the available time gaps in the accelerator beam structure mentioned above. Some estimates show that it will take up to 15..30 minutes to cycle through all the pixels in the front-ends (assuming one private orbit per second). Because of the beam synchronization requirement for the ROC refresh cycle, an arbitrary command to the front-end cannot be issued without disabling the refresh cycle. This can be accomplished by interrupting refreshing cycle for a short period of time and issuing command to the front-end. We

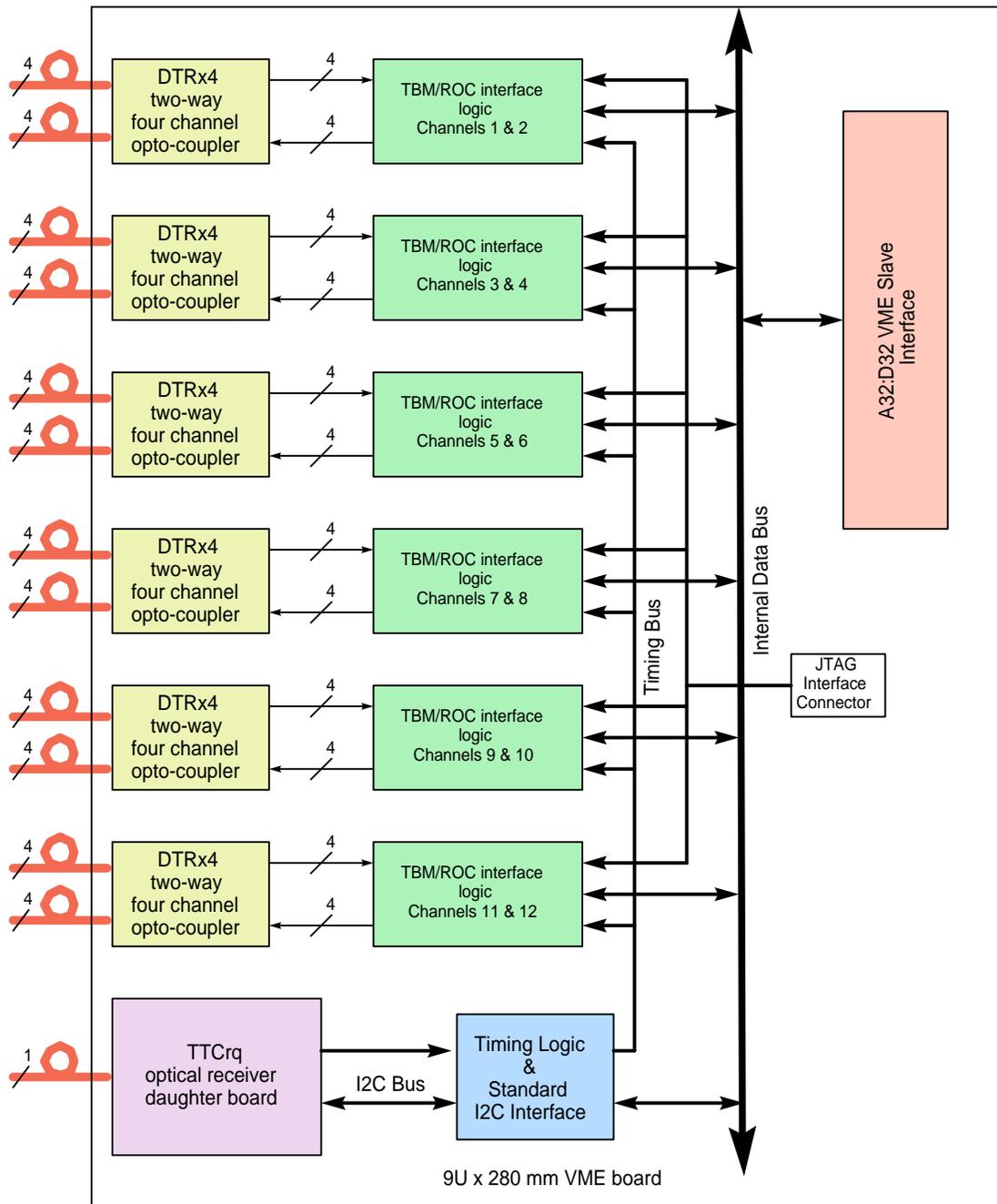


Fig. 1 Block-diagram of the Pixel Front End Controller

assume here that communications with other electronics not included in the data path (e.g. DCU) will not disturb data taking. After the command is processed, the refreshing cycle will be resumed. This can be used for periodic readout of the monitoring information from the front-ends.

In the absence of a TTC connection, the TTCrq will generate the LHC clock frequency internally. This mode of operation is called LOCAL and can be used for diagnostic purpose and stand-alone operations. There are no external trigger decisions

received in this mode, but a fixed sequence of Calibrate/L1 Accept signals could be generated locally if desired. Summarizing, there are four modes of operation:

- **Initialization**
- **Stop**
- **Run**
- **Local**

The associated VME crate software provides switching between different modes. The FEC status register should show current status of the module.

4. FEC channel interface

The FEC communicates to the TBM chips, the ROCs, the PLL chips, the opto-hybrids and the DCU chips. The TBMs and ROCs designed by CMS Pixel group require a custom high-speed communication protocol, which uses two differential pairs for serial clock and data and runs at the main TTC clock frequency of 40 MHz. This protocol has similar to the I²C standard^[4] start and stop definitions and uses 10 bit data words with the 5th and 10th bits representing inverted 4th and 9th bits respectively, and the remaining bits representing an 8-bit data byte. The first data word in this protocol comprises of hub and port addresses, which define an addressed device(s). The TBM can be directly addressed by the FEC using this protocol. The ROC chips associated with a particular TBM chip can be addressed using a specific port address within that TBM. Note that current version of the ROC has only write-only registers and cannot be read back. The TBM chip has the read back functionality as described in the I²C standard.

The PLL chips, opto-hybrids and DCU chips require a standard I²C communication protocol, which is provided by using a custom slow hub chip. The slow hub has a specific hub address to which it responds at the end of transmission of the first data word in the command. According to the current proposal^[5], the slow hub chip will decode device address transmitted at 40 MHz clock speed and multiplex following standard I²C command generated by the FEC to the standard I²C device. It will receive following device's response and direct it backward to the FEC. A clock frequency for the slow devices is set at 40/64 MHz = 625 kHz. In order to overcome low frequency cutoff of the optical links, the data and clock for slow devices are modulated using bi-phase encoding at 40 MHz. The proposed combination of two protocols is not very reliable and has to be improved (see Appendix A for details). The design of the FEC channel interface is based on a programmable logic and, in general, is not affected by changes in communication protocol, which could be implemented at a later time.

A block-diagram of one channel FEC interface is shown in Fig. 2. Each channel has a serial transmitter and serial receiver with associated logic. Serial data and clock are transmitted in both directions. Transmission of serial clock and data in both directions is widely used by other CMS sub-systems and is a proven industry standard. The serial receiver has a small FIFO memory to store response information from the front-end until the software can read it out. It is preferable to avoid storage of multiple responses in the FIFO to simplify software requirements.

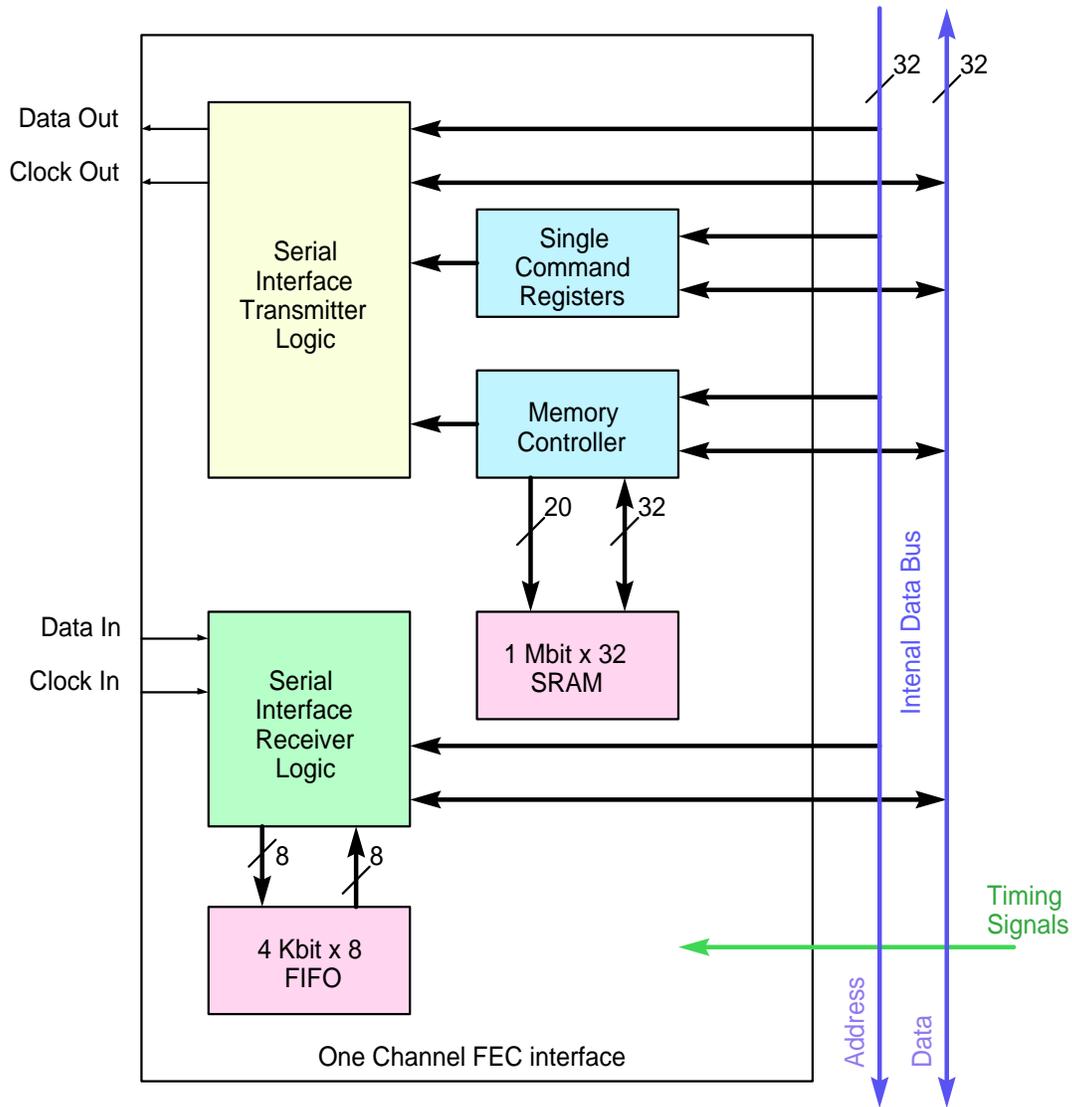


Fig. 2 One channel FEC interface

The transmitter logic can generate commands received from two sources: single command registers and command memory. Single command registers are used to perform single control commands (e.g. Read) that require a response from the receiver. This could be a request of status information or DCU readings. The other commands that do not require response from the receiver could be generated by either single command register or command memory. The command memory is also used to refresh ROC settings. In this case commands can only be generated at certain times synchronously with accelerator activities. The use of timing bus signals to synchronize FEC commands to the accelerator can be enabled or disabled by a bit in the control register.

The memory controller will cycle through the portion of the memory determined by a special register, repeating all the stored commands. The size of the memory should be large enough to accommodate all the necessary commands for the ROCs connected to one FEC channel. The contents of the memory will be loaded during the initialization

phase, but can be modified when necessary. It may be necessary to use DMA mode of the VME master to speed up the process of loading all the memories on the FEC card.

5. FEC registers

The definition of the control and status registers can be finalized when programmable logic is complete.

6. FEC setup

The setup procedure is design dependent and can be finalized after design of the programmable logic is complete.

7. FEC software

To be defined later.

References:

- [1] 80Mbit/s Digital Optical Links for Control, Timing and Trigger of the CMS Tracker, Proceedings of LECC 2002, Colmar, Sept.2002
- [2] Digital Transceiver Module DTRx4, <http://cms-tk-opto.web.cern.ch/cms-tk-opto/control/components/dtrx4.html>
- [3] TTCrx and QPLL Mezzanine Card (TTCrq), Paulo Moreira, Ernest Murer, Rev. 0.1, <http://ttc.web.cern.ch/TTC/intro.html>
- [4] The I²C Bus Specification, Version 2.1, January 2000, Philips Semiconductors, <http://www.semiconductors.philips.com/buses/i2c/>
- [5] Communications Control Hub, Ed Bartz, design document, <http://www.physics.rutgers.edu/~bartz/cms/hub/index.html>

Notes on Pixel FEC communication protocol

Initial requirements:

1. Down link (DTRx4 => DOH)
 - Use 40 MHz clock frequency for data transmission
 - Use 4b/9b data bits inversion to prevent reset detection by DOH
 - When not active, send stops continuously to maintain OL integrity
 - Use first byte to address fast/slow hubs
2. Up link (DOH => DTRx4)
 - Use 40 MHz clock frequency for data transmission
 - Send hub address to FEC in response to every command
 - Do not use 4b/9b data bits inversion
 - When not active, send stops continuously to maintain OL integrity

The current implementation of the readout chip (PSI46) requires 4b/9b data bits inversion. Because of the presence of the standard I2C devices (TPLL, LD, DCU), the proposed communication protocol is very complicated (requires two different encoding mechanisms for fast and slow hubs) and unreliable. It has potential pitfalls that may create serious problems during debugging and running of the system. The only advantage of this approach is simplification of the slow hub design.

Another approach will be to simplify protocol to only one encoding scheme (4b/9b) and implement additional logic in the slow hub design. In this case debugging and diagnostic of the running system becomes less cumbersome. The slow hub design will have to convert fast commands addressed to standard I2C devices forth and back using 32-bit shift registers.

A clock issue has to be resolved in a reliable fashion. For the sake of preventing TPLL from loosing lock to the LHC clock, two clock inputs have to be present in the slow hub design. After power up, the encoded clock is a default clock source for the slow hub. During this time no trigger decision are issued and TPLL can be addressed for status and PLL adjustments. A special command should be generated to switch slow hub to the TPLL clock source. The FEC has to use a clean clock to encode data sent to the front-end. The slow hub has to be the source of the return clock to the FEC. This does require additional clock lines from the TBMs, but can be avoided by re-timing TBM data within the slow hub.

In order to understand potential problems, as an example, let us consider single bit error in the serial data. The source of the error could be anything and is not important. Single bit errors are always present in communication links at some level. Table 1 shows possible results of a single bit error in different parts of the command for currently implemented bi-phase encoding and proposed fast encoding schemes.

If an error is present in the first data word (hub/port address), it creates either a non-existing address or modifies the address in such a way that it corresponds to the existing device. The latter one is a serious error and has to be detected. In the current provision, the first word is always returned to the FEC. In both cases (unless another error is present in the upstream data), the error will be detected and corrected by issuing another identical command. Addressing a wrong fast device with the fast protocol will not prevent it from communicating. If, on the other hand, the slow protocol is sent to a fast device (ROC) or fast protocol is sent to a slow hub, it may potentially create a mess.

Table 1

Command word affected	Bi-phase encoding	Fast protocol
Hub/port address	Addresses wrong or non-existing device, slow command can be send to a fast device and vice versa	Addresses wrong or non-existing device, makes no difference between fast and slow devices as all commands are fast
Command	Generates a 25 ns spike on the data line, can be fatal to a slow device	Addresses non-existing or wrong internal register
Data	Generates a 25 ns spike on the data line, can be fatal to a slow device	Sets incorrect or invalid data bit in a register

If one bit error is present in bi-phase encoded data (command or data byte), it will create a 25 ns glitch on the data bus. It is not clear at present what that will do to the slow I2C device. In a worst case scenario, the addressed device may stop functioning and will require a hard reset. In the fast protocol, this error only makes the command illegal without disturbing the actual device.

In a summary, a uniform fast protocol is much better and reliable approach for communicating with the Pixel front-end.