MCC-I2.1 Specifications

MCC Design Group

Event Builder

Revision:	1.0
Reference:	ATLAS ID-xx
Created:	8 December 2003
Last modified:	1 September 2004 10:59
Edited By:	R. Beccherle and G. Darbo
	ATLAS/Pixel Collaboration

Abstract

This document describes the EventBuilder of the *MCC-I2.1*.

Table Of Contents

List of Figures

4 Event Builder

This chapter will describe the EventBuilder block of the *MCC*.

Event building is performed by two concurrent processes running in the *MCC*. The first one deals with the filling of the 16 input FIFO’s with data received from the corresponding *FE* chip and is performed by the Receiver (see Chapter 3, "Receiver") while the second one (EventBuilder) extracts data from the ReceiverFifo’s and builds up the event and is described in this chapter.

During normal conditions, each Receiver fills data hits from the *FE* chips into the ReceiverFIFO. When an EoE word is detected, the Scoreboard in the EventBuilder is updated for the corresponding Receiver. As soon as the EventBuilder finds at least one complete event, i.e. that all 16 FE’s have sent a comlete event (the Scoreboard is used for that), it starts building up and transmitting the event. The first information written to the output data stream is the last word in the PendingLv1Ffifo. This FIFO is 16 bit wide and 16 word deep and stores the bunch crossing ID (BCId) and the Lev1 ID (LEV1Id). The two ID’s are the output bits of two counters: bunch crossing and Trigger counters. Only the 4 less significant bits of the TriggerCounter are written to the output data stream. The BunchCrossingCounter is updated once every clock cycle, while the TriggerCounter is incremented each time a Trigger is sent to the *FE* chips. The contents of the two counters are transferred to the PendingLv1FIFO each time a Lev1 is transmitted to the *FE* chips from the *MCC*. At this point the EventBuilder starts fetching data from the first ReceiverFifo that contain data. Once data of this Receiver is completely read the EventBuilder switches to the next Receiver that contains data. After having transmitted data from all ReceiverFifo’s a Trailer word is written which ends the event data transmission. For more information on the output data format please refer to Chapter 5.4, "MCC to ROD Event Format".

The next section will describe in detail all building blocks that form the EventBuilder. In the second one we will describe the implemented test features, the third section will explain how the different types of reset signals action the EventBuilder, while the last one will describe how the EventBuilder is protected against the occurrence of SEU events.

4.1 Building blocks

Figure 4-1 shows the logical scheme of the EventBuilder. As can be seen from this figure there is a direct data path going from the Receiver blocks to the output of the *MCC*.

4.1.1 EoE Encoder

The EoE encoder is the block that selects which, out of the 16 input channels, is the active one. This block reads the information of which of the 16 *FE* chips are active from the FEEN register and provides the address of the next ReceiverFIFO that contains non null data. This is done with a priority encoder that enables one FIFO at a time. The first available data stream is selected based on the FEEN value and the presence of a Hit or a WngEoE word in the FIFO. If a Hit word is found it is copied to the output data stream, while if an EoE word containing flags (generated by the corresponding *FE* chip or by the Receiver itself) is found a FEFlag word is generated. The selection remains valid until the control state machines issues a NextFE signal, in which case the priority encoder will point to the next FE with data, if there is any.

This is done only if the *MCC* is in RunMode. If we are reading one ReceiverFifo location with a RdFifo command the EoE encoder block simply provides the address that was specified in the RdFifo command.

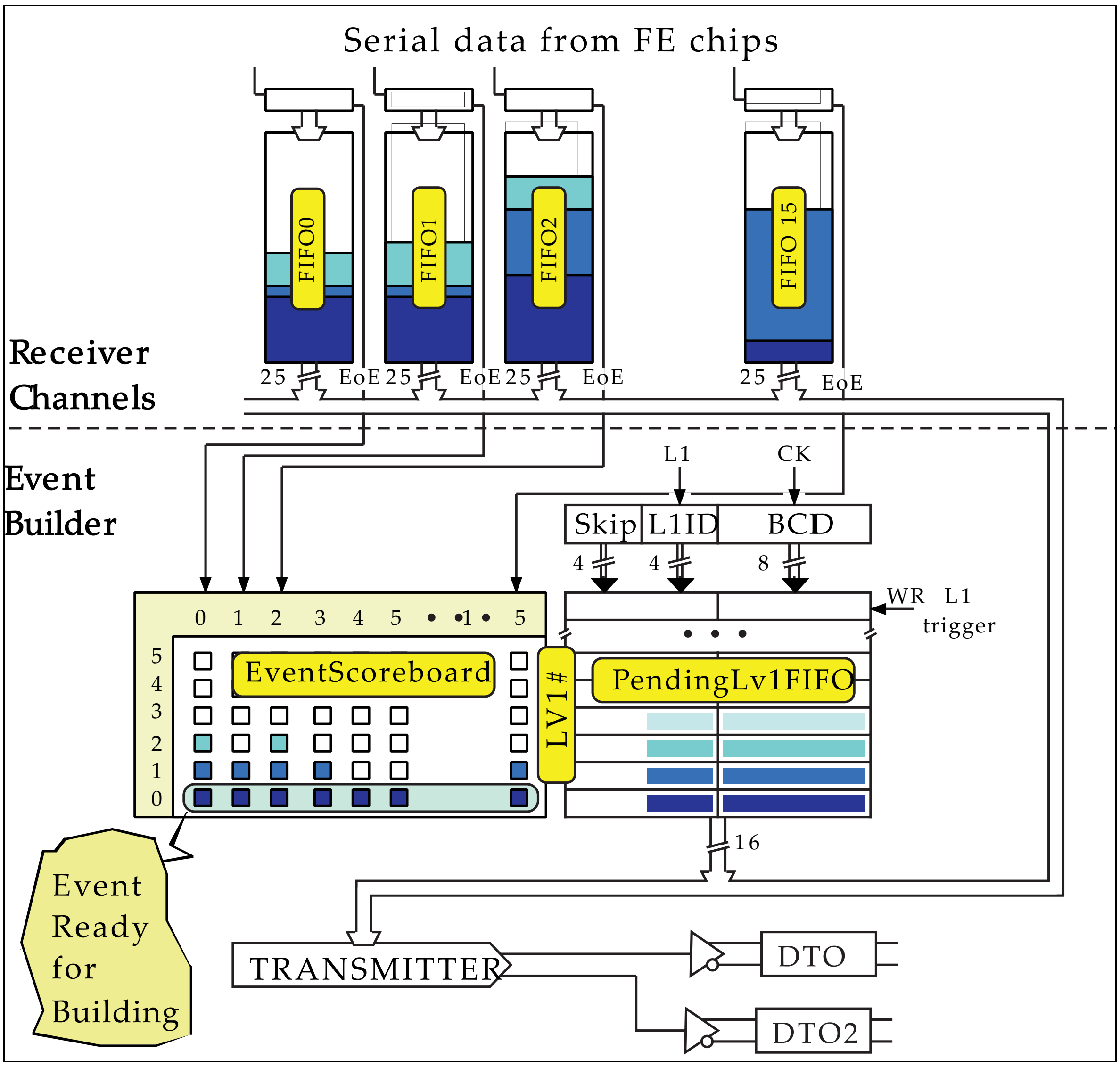

Figure 4-1 Logical scheme of the EventBuilder block.

This block also generates a signal that is active until there is data to be read out by any of the Receiver-FIFO (FEHaveData). This information is used by the EventBuilder control state machine.

4.1.1.1 Event Building Example

As an example lets suppose that all *FE*'s are enabled and that for that particular event FIFO # 1, FIFO # 7 contain valid data, FIFO # 2 contains a WngEoE word while all other ReceiverFifo's are empty (contain only the TrueEoE word). FEHaveData will be activated. After that, address "1" will be selected until all data is read out, then FIFO # 2 will be addressed in order to read the warning messages present in the EoE word, a FEFlag word will be generated due to the presence of this warning, than the priority encoder will provide address "7" until data completion and finally it will signal that there is no more valid data for that event disabling the FEHaveData signal which tells the control state machine that the event is finished. At this point, if the Receiver's do not contain anymore full events (one EoE word in each of the enabled ReceiverFifo's), the FEHaveData signal will be lowered until the ScoreBoard signals that a new complete Event has been detected.

4.1.2 Input Multiplexer

Based on the address information provided by the EoE encoder the multiplexer selects which data stream has to go to the control state machine for event building.

This block also multiplexes which of the 16 TrueEoE and WngEoE words coming from the 16 Register blocks has to be sent to the control state machine.

4.1.3 ScoreBoard

This block records the EoE information coming from all Receiver blocks that are enabled and generates a signal to be used by the control state machine for starting event building (StartBuild).

The ScoreBoard is formed by 16 independent 5 bit up-down counters that are incremented as soon as an EoE word (with or without warning flags) has been detected by a Receiver.

As soon as all the counter values are not zero (all Receiver blocks have seen at least one event) the signal is activated. This signal stays active until at least one of the counters is zero.

Immediately after having built a complete event the control state machine activates the DecrScore signal and all counters are decremented by one.

The counters cannot overflow as the *MCC* can not send more than 16 Trigger signals to the *FE* chips and inside each Receiver block there is a HitOverFlow and an EoEOverflow check that ensure that there will never be more than 112 Hits and 16 EoE words per ReceiverFifo.

Also an underflow of the Scoreboard counters is not possible as the PendingLV1Fifo ensures that there will never be more than 16 Trigger commands sent to the Fe chips and the DecrScore signal is only issued after one *MCC* event has been reconstructed and this happens only after all 16 *FE* chips have sent their EoE word and this, in turn, can only happen after a *FE* chip has received a Trigger command.

4.1.4 PendingLV1FIFO

The PendingEventFIFO is 16 bit wide and 16 word deep fully synchronous, single-clocked FIFO.

The RdPtr is controlled by the EventBuilder control state machine while the WrPtr is controlled by the Trigger, Timing and Control block (TTC).

One word is written to the PendingEventFIFO each time a Trigger command is issued to the *FE* chips.

Each word of the PendingEventFIFO is 16 bit wide and is subdivided in three distinct fields.

PEF<15:12>: SkippedLV1 field. This 4 bit field contains the number of skipped LV1, as determined by the TTC. The maximum number can be 15 and means that 15 or more events have been skipped. There is a skipped LV1 each time a new Trigger command is received by the *MCC*, if there are still 16 Trigger commands sent to the *FE* chips without having started at least one EventBuilding process. This information is added by the EventBuilder control state machine to the output data stream in order to inform the *ROD* that a certain number of Trigger commands have been skipped.

PEF<11:8>: LV1Id field. This 4 bit field contains the 4 least significative bit of the LV1Id counter, an 8 bit up-down counter implemented in the TTC. This counter is incremented each time a Trigger command is detected by the *MCC*.

PEF<7:0>: BCID field. This 8 bit field contains the value of the BCID counter. This 8 bit counter is incremented at each clock cycle inside the TTC.

The PendingEventFIFO information is added immediately after the Header to all events formatted by the *MCC* (see Chapter 5.4, "MCC to ROD Event Format").

4.1.5 Control State Machine

This block is by far the most complex block of the EventBuilder as it has to deal with all event building control signals and to implement the complete event formatting according to the event grammar explained in Chapter 5.4, "MCC to ROD Event Format".

Each block that forms the Control State Machine is described in detail in the following sections.

4.1.5.1 FeFlag Generator

This block prepares the eventual FeFlag word to be added in the output data.

This block also performs an event consistency check between data belonging to the same event but to different *FE* chips. Each EoE word of a *FE* that has data, or just a WngEoE word, is checked. The LV1Id information of the first EoE word is stored in a 4 bit register and compared with EoE words belonging to other *FE* chips. As soon as a discrepancy is found the Lv1ChkFail signal for that particular *FE* is set. If there is only one *FE* with data no check is performed. In case the wrong LV1Id is in the first *FE* with data all subsequent *FE* will have the Lv1ChkFail bit set in their data stream. The same check is also performed for the BCId which is stored in each EoE word. In case of a discrepancy of this kind the BCIdChkFail signal is set.

If a *FE* chip does not have any data associated to an event no check is performed on both LV1Id and BCId fields of the EoE word.

Both consistency checks of the BCId and the LV1Id can be individually disabled by setting WBITD<11> and WBITD<12>. By default both checks are enabled.

During EventBuilding as soon as a BCIdChkFail or a Lv1ChkFail signal is found a FeFlag word is added to the *MCC* output data stream. This FeFlag word will contain the information of which of these checks has caused the error.

The FeFlag word is an 16 bit word subdivided in two logically distinct parts:

FeFlag<7:0>: This 8 bit contain the warning messages generated by the *FE* chips. As explained in Chapter 3.1.3, "Control State Machine", these messages are added by the *FE* chips to their EoE word and are stored by the Receiver inside the FIFO in the EoE word. The EventBuilder simply copies this information in this field. See the *FE* chip specification document for the full description of the single bits.
The presence of such a flag is written to the corresponding bit of WFE register.

FeFlag<15:8>: This 8 bit contain the warning messages generated by the *MCC*. As explained in Chapter 3.1.3, "Control State Machine" three bits are generated in the Receiver block. The consistency checks between hits belonging to different *FE* chips, but to the same event, instead are performed inside the FeFlag generator block. The three remaining bits are not implemented in this version of the *MCC* but space is allocated for eventual extensions in future versions of the chip.

The meaning of the single bits of FeFlag<15:8> is the following:

FeFlag<15:13>: These three bits are always “000” in this version of the *MCC*.

FeFlag<12>: There has been a Lv1ChkFail in the event. This means that there is data inconsistency inside the current event due to a different LV1Id number between the current *FE* and the first *FE* with data.
This information is also written to CSR <12>.

FeFlag<11>: There has been a BCIdChkFail in the event. This means that there is data inconsistency inside the current event due to a different BCId number between the current *FE* and the first *FE* with data.
This information is also written to CSR <11>.

FeFlag<10>: There has been a Lv1ChkFail in the corresponding Receiver. This means that there is data inconsistency between the Hits belonging to that particular *FE*.
This information is also written to the corresponding bit of the WMCC register.

FeFlag<9>: There has been an EoEOverflow. This happens if, for a particular Receiver, there are already 16 EoE words written to the ReceiverFIFO and another EoE word is sent by the *FE*. In this case the exceeding EoE word is dropped and the flag is set.
This information is also written to the corresponding bit of the WMCC register.

This should never happen as the *MCC* ensures, by construction, that there are never more than 16 Trigger commands sent to the *FE* chips. This is a strong indication that some data corruption happened in the *FE* chip (for example a SEU turned a Hit into an EoE word).

FeFlag<8>: There has been a HitOverflow. This happens if, for a particular Receiver, the maximum number of Hits (112) has been reached and that *FE* keeps sending data. In this case all additional Hits are dropped and this bit is set.
This information is also written to the corresponding bit of the WMCC register.

There is no FeFlag generation in case of skipped LV1’s. In such a case the number of skipped triggers is simply added to the LV1 field of the last event without skipped events.

4.1.5.2 State Machine

This state machine is responsible for the event grammar decoding and generates a control signal for the event formatting block (DoutSelect). This is a 3 bit signal that by default is cleared. A new signal is generated, during event building, only after the OutputPort has sent a DataRequest signal. This signal is issued only once the OutputPort has “almost finished” sending out data.

The state machine knows when to start event building listening the FEHaveData signal issued by the EoEEncoder block.

If the *MCC* is not in RunMode the only possible values states of the machine are the idle state and the one that produces data in response to a RdFifo command.

The state machine also to generates all control signals needed during event building by the whole *MCC*. This signals are:

NextFE:	Signal used by the EoEEncoder to select the next FE with available data.
DecrScore:	Signal used by the ScoreBoard to decrement all it’s counters.
RdPendingEventFifo:	Signal used by the ScoreBoard to read the next location of the PendingLV1Ffifo.

4.1.5.3 Event Formatting

This block formats the event word according to DoutSelect signal generated by the state machine. This word is put in a 26 bit wide register that contains the data (DataOut). In addition it generates a 26 bit mask registers that tells the OutputPort which of the 26 bits are significative (DataMask) and a 5 bit word (DataSize) that tells the OutputPort how many bits of the word are used.

DoutSelect can have 7 different values with the following meaning:

DoutSelect = 0:	This is the default case and means that there are no data to be sent out. In this case DataOut, DataMask and DataSize are all zero.
DoutSelect = 1:	In this case Sync, “0111” and FeNumber are written to DataOut and DataSize is 9.
DoutSelect = 2:	In this case Sync, Row#, Col# and ToT information are written to DataOut and DataSize is 22.
DoutSelect = 3:	In this case Sync, “1_1111” and FeFlag are written to DataOut and DataSize is 22.
DoutSelect = 4:	In this case Sync and Trailer information are written to DataOut and DataSize is 23.
DoutSelect = 5:	In this case nothing is written to DataOut and DataSize is 24. This case is used when a certain number of Trigger commands were skipped (not sent to the *FE* chips) in order to allow the *ROD* to pad the data stream with the missing events.
DoutSelect = 6:	In this case the Header and ReceiverFIFO data are written to DataOut and DataSize is 26.
DoutSelect = 7:	In this case the Header, SkippedLv1, LV1Id, Sync and BCID are written to DataOut and DataSize is 22.

⚠ Setting accordingly registers WBITD, WRECD and the RECD bit of the CSR register, as explained in Section 2.1.1.2, "RECD: Receiver disable", one can selectively block flag generation (also just for some Receivers) in the *MCC* output data stream. This allows for some bandwidth reduction in case of a known problem in one of the components of the module.

4.1.6 Output Port

This block deals with data formatting of the *MCC* depending on the selected output mode OM (see Chapter 2.1.1.1, "OM<1:0> : Output Mode select"). All data transmitted by the *MCC* on the MCC-DTO pins which involves data coming from the ReceiverFIFO’s is generated by this block.

The OutputPort will fetch data from the control state machine 26 bit wide DataOut register. All bits that are not masked by DataMask will be copied to a 32 bit wide register (DataStore). This register is organized as a one-dimensional FIFO with read and write pointers. Data will be copied starting from the first free location of this register. At the same time the DataSize value will be added to an up-down counter that controls the pointers to DataStore. Depending on the selected OM the counter is then decremented by 1, 2 or 4 each clock cycle. As soon as the counter is below a fixed threshold new data will be copied to the DataStore register. This is done sending the DataRequest message to the EventBuilder state machine and a RdReceiverFifo to the correct Receiver. When the DataStore register is empty “0” will always be written to the output.

This mechanism allows to asynchronously write data to the DataStore as soon as enough space becomes available and read data from DataStore at different speeds depending on the selected OM.

The output of this block is a 4 bit wide register that holds the information to be written on both MCC-DTO pins. Data are written to this register according to the selected output mode.

OM = 0: Same data on both links at 40 Mb/s with 40 Mb/s bandwidth. In this case four copies of the same information are written to the register.

OM = 1: Data on both links at 40 Mb/s with 80 Mb/s bandwidth. In this case two bits of data are written at each clock cycle to the output register. The first bit of information is written to register location “0” and “2” while locations “1” and “3” will hold the second bit of data.

OM = 2: Same data on both links at 80 Mb/s with 80 Mb/s bandwidth. In this case two bits of data are written at each clock cycle to the output register. The first bit of information is written to register location “0” and “1” while locations “2” and “3” will hold the second bit of data.

OM = 3: Data on both links at 80 Mb/s with 160 Mb/s bandwidth. In this case four bits of data are written at each clock cycle.

The ModulePort will sample this data and distribute it correctly on both links. This module will always sample location “0” of this register on the rising edge of the clock and transmit it on MCC-DTO 0, location “1” on the rising edge of the clock and transmit it on MCC-DTO 1, location “2” on the falling edge of the clock and transmit it on MCC-DTO 0 and location “3” will be sampled on the falling edge of the clock and transmitted on MCC-DTO 2 (see Section 5.5, "MCC to ROD Physical Layer Protocol").

If we have to process a RdFifo command the OM is automatically switched to “0”, in order to ensure that data in response to configuration commands is always written at 40 Mbit/s on both links.

Data will always be formatted in such a way to ensure that the first bit of the Header word is always on the rising edge of MCC-DTO 0.

4.2 Test Features

In the *MCC* there are many different test features implemented in order to maximize the debugging capability both of the *MCC* itself and a whole module with all *FE* chips connected. This is particularly useful during module production tests in order to decouple the *MCC* behaviour from the rest of the module, and during the experiment where one has no more physical access to the module.

This test modes should allow the final user to perform any significant check that is needed to understand if the chip is still working and to eventually be able to uderstand which part of the chip is not working properly, even once the *MCC*'s have been installed inside the detector.

In the next two sections we will describe the two main *MCC* test modes.

4.2.1 MCC PlayBackMode

In order to be able to completely test the EventBuilder a special operating mode was built in the system. This mode is called PlayBackMode and is enabled by setting PLBK register in the CSR <6>.

After the MCC is in PlayBackMode one can write data to the ReceiverFIFO's using the WrReceiver command. This command (see Section 1.2.3.7, "WrReceiver: Write data into a Receiver") allows to write data to any ReceiverFIFO, using the FEEN register, as if data would be coming from the *FE* chips. This allows the EventBuilder's ScoreBoard and PendingEventFifo to be set up correctly in order to start event building. Once one has written all data to the FIFO's one can put the *MCC* in RunMode with the EnableDataTake command and start sending Trigger commands as if in real RunMode. At this point the whole EventBuilder works as if with real data and allows a complete check of it's functionality. One can check all possible conditions, FeFlag generation capability, Hit or EoE overflows, etc. writing data that has some errors in it.

⚠ During PlayBackMode all MCC-DTI lines (the lines that transmit data from the *FE* chips to the Receivers), are still connected and therefore one has to make sure that all enabled *FE* chips do not send data. This should be ensured by the fact that no Trigger commands are sent to the *FE* chips.

This is a very useful operation mode in order to be able to test the system prior to connecting the *FE* chips or in a real system if there are problems in order to understand if they are related to the *MCC* itself or to one or more *FE* chips.

⚠ As usual if one issues a Slow command the chip is taken out of RunMode but the PLBK bit is not cleared and therefore one has to clear it writing the correct word to the CSR register.

4.2.1.1 Event Building Test Example

Suppose we would like to test the EventBuilder with all 16 *FE* chips enabled and with a FeFlag due to a wrong LV1Id present in FE # 3. Suppose also that only FE # 1 and FE # 14 have both just one Hit.
The OM to be selected is data on both links at 40 Mb/s with 80 Mb/s bandwidth.
In this case we would have to issue the following commands:

1. GlobalResetMCC: Resets the *MCC*.
2. WrRegister CSR 0x0041: Sets the desired output speed (`0001`) and sets the PLBK bit (`0040`).
3. RdRegister CSR: Checks that we wrote the correct value. This command returns the header followed by 0x0041 on both DTO-0 and DTO-1 lines.
4. WrRegister FEEN 0x0001: Enables FE # 1.

5. RdRegister FEEN: Checks that we wrote the correct value. This command returns the header followed by `0x0001` on both DTO-0 and DTO-1 lines.
6. WrRegister CNT 0x0007: Writes the length of the data stream to be used by the WrReceiver command. In this case commands with up to 56 bits will be accepted.
7. RdRegister CNT: Checks that we wrote the correct value. This command returns the header followed by `0x0007` on both DTO-0 and DTO-1 lines.
8. WrRceiver 0000 HitData EoEData: Writes the Hit and the EoE words to FE # 1 (note the dummy but required field 0000).
9. WrRegister FEEN 0x8000: Enables FE # 14.
10. RdRegister FEEN: Checks that we wrote the correct value. This command returns the header followed by 0x8000 on both DTO-0 and DTO-1 lines.
11. WrRceiver 0000 HitData EoEData: Writes the Hit and the EoE words to FE # 14 (note the dummy but required field).
12. WrRegister FEEN 0x0004: Enables FE # 3.
13. RdRegister FEEN: Checks that we wrote the correct value. This command returns the header followed by 0x0004 on both DTO-0 and DTO-1 lines.
14. WrRceiver 0000 EoEData: Writes the EoE words with the wrong LV1Id to FE # 3 (note that one does not have to change the value of CNT in order to write less bits).
15. WrRegister FEEN 0x7ffa: Enables all remaining *FE* chips.
16. RdRegister FEEN: Checks that we wrote the correct value. This command returns the header followed by 0x7ffa on both DTO-0 and DTO-1 lines.
17. WrRceiver 0000 EoEData: Writes the EoE words to all remaining *FE* chips (note that one single command is needed as all EoE words are the same).
18. EnDataTake: Sets the *MCC* in RunMode.
19. Trigger: Starts the EventBuilder and writes the event at 80 Mbit/s on both DTO lines.

4.2.2 MCC Internal Status Ckecks

In addition to the ability to perform an EventBuilding Test one has the ability to readout all relevant internal structures that contribute to the EventBuilding process.

One can for example completely disable the EventBuilder Control State Machine setting EVBDIS register in the CSR <7>. If this mode is enabled the EventBuilder Control State Machine is always in it's idle state and therefore does not start the Eventbuilding process.

⚠ In order to use this test mode one is advised (even if it is not mandatory) to set the PLBK register in the CSR <6> in order to disable Trigger sending to all *FE* chips, and thus avoiding that *FE* chips produce independent input to the *MCC*.

At this point one can easily populate the ReceiverFifo's using the WrReceiver command as in the EventBuilding Test Example explained before. The contents of all ReceiverFofo's can the be checked using the RdFifo command (ensuring that one reads each FIFO an integer multiple of 128 times in order to leave the ReceiverFifo pointer positions unaffected).

An additional option at this point is the ability to check the contents of the ScoreBoard status using the RdRegister 1000 (SBSR address). This operation can be performed as many times one desires in order to moitor that the ScoreBoard populates correctly.

At the end one may send a congruous sequence of Trigger commands (still staying in PlayBackMode) in order to completely read out the data which has been stored in the ReceiverFifo's.

Of course, one can also monitor the status of the PendingLV1Fifo during this process remembering to set the *MCC* in RunMode after each Slow command and making sure all pointers of the PendingLV1Fifo do not change their actual locations (issue an integer multiple of 16 RdRegister 0111 commands).

4.3 Reset Actions

This section will describe the response of the EventBuilder on all available reset commands that can be used in the *MCC*.

The only block that is not affected by any reset signal is the input multiplexer as it contains only combinational logic. For all the others block the reset is synchronous, as throughout the whole chip, and therefore one needs to have the clock active in order to perform a reset of this block.

4.3.1 Response to a Pin Reset

The pin reset (MCC-RSIb) has the same functionality as the GlobalResetMCC command. Even the pin reset is a synchronous reset and therefore one has to ensure that the clock signal is applied in order to be able to reset the chip in this way.

4.3.2 Response to a GlobalResetMCC command

In case of a GlobalResetMCC command the whole EventBuilder gets cleared.

All the counters are reset to ‘0’, the pointers of the PendingEventFifo are re initialized and after a GlobalResetMCC command point both to location ‘0’, the ScoreBoard is cleared and the EventBuilder state machine is put to it’s idle state. All data transmission in the OutputPort is stopped and eventual flags are cleared. All test modes are cleared.

4.3.3 Response to a BCR command

The BCR command has no effect on the EventBuilder.

4.3.4 Response to an ECR command

An ECR command clears all structures in the EventBuilder. This means that for the EventBuilder this command is completely equivalent to a GlobalResetMCC command.

This allows for a fast reset of the DataPath of the *MCC* without exiting RunMode, as prescribed by the ECR command.

4.4 SEU protection

In order to protect the Event Builder against the occurrence of SEU we decided to triplicate as much control logic as possible. Triplicating the entire EventBuilder was not feasible because of it's size.

The Input Multiplexer contains only combinational logic and has therefore no need to be protected.

The OutputPort and EoEEncoder blocks only contain data in the eventbuilding data path (data that is not stored inside the MCC for longer period of time) and are therefore less critical even if they could cause errors during the EventBuilding process in case of a SEU. An error in one of this blocks would cause an MCC Event format error but cannot block the chip or even the EventBuilding process.

The Control State Machine is by far the most critical part of the whole Eventbuilder and it was decided to fully triplicate it adding amajority voting circuit to all of it's outputs in order to always take the correct output. Also this state machine is designaed in order to return to its idle state as soon as eventBuilding has completed or an error is detected. Therfore the risk of upsetting two different state machines before their return to idel condition is minimized even if not completely absent.

⚠ The ScoreBoard block is the only block that could cause problems, as it stores important information for correct event readout and syncronization but, due to it's' size, it was decided not to triplicate it. An upset in one of the counters that form the ScoreBoard would cause alignment problems but this would induce the *MCC* to generate many FE-Flag words in its output data stream and therefore one can solve the problem issuing an ECR command.